MCS12 Debugger

Version 06-Jun-2024

Brief Overview of Documents for New Users

Architecture-independent information:

- **"Training Basic Debugging"** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **"T32Start"** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **"General Commands"** (general_ref_*<x>*.pdf): Alphabetic list of debug commands.

Architecture-specific information:

- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- **"OS Awareness Manuals"** (rtos_*<os>*.pdf): TRACE32 PowerView can be extended for operating system-aware debugging. The appropriate OS Awareness manual informs you how to enable the OS-aware debugging.

Warning

WARNING:	To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF. Recommendation for the software start: 1. Disconnect the Debug Cable from the target while the target power is off. 2. Connect the host system, the TRACE32 hardware and the Debug Cable. 3. Power ON the TRACE32 hardware. 4. Start the TRACE32 software to load the debugger firmware. 5. Connect the Debug Cable to the target. 6. Switch the target power ON. 7. Configure your debugger e.g. via a start-up script. Power down: 1. Switch off the target power. 2. Disconnect the Debug Cable from the target. 3. Close the TRACE32 software. 4. Power OFF the TRACE32 hardware.

Troubleshooting

SYStem.Up Errors

The SYStem.UP command is the first command of a debug session where communication with the target is required. If you receive error messages while executing this command this may have the following reasons.

- The target has no power.
- The target is in reset - another device may hold the reset line active.
- There is a short circuit on at least one of the output lines of the core.
- There is a problem with the electrical connection between ICD12 and the target - check if the BDM connector is plugged correctly and if the target is built corresponding to the **definition** of the used BDM connector.
- The 68HC12 has no clock - check the frequency on the EXTAL pin with a scope.
- The clock for the BDM transmission is not set correctly - refer to **SYStem.BdmClock**

FAQ

Please refer to https://support.lauterbach.com/kb.

CPU Specific Implementations

Breakpoints

There are two types of breakpoints available: Software breakpoints and on-chip breakpoints.

Software Breakpoints

Software breakpoints are the default breakpoints for program breakpoints. A software breakpoint is implemented by patching a break code into the memory.

There is no restriction in the number of software breakpoints.

On-chip Breakpoints

The resources for the on-chip breakpoints are provided by the CPU.

The following list gives an overview of the supported on-chip breakpoints:

- **On-chip breakpoints:** Total amount of available on-chip breakpoints.
- **Instruction breakpoints:** Number of on-chip breakpoints that can be used to set Program breakpoints into ROM/FLASH/EEPROM.
- **Read/Write breakpoints:** Number of on-chip breakpoints that can be used as Read or Write breakpoints.
- **Data breakpoint:** Number of on-chip data breakpoints that can be used to stop the program when a specific data value is written to an address or when a specific data value is read from an address.

Family	Onchip Breakpoints	Program Breakpoints	Read/Write Breakpoints	Data Value Breakpoints
68HC12 **68HC12A**	up to 2	up to 2 single address	up to 2 single address	1
MCS12 **MCS12C**	up to 3	up to 3 single address	up to 3 single address	1
S12X **S12Z**	4	up to 4 single address or 2 address ranges	up to 4 single address or 2 address ranges	1

Quick Start of the ICD Debugger for HC12

This chapter should help you to prepare your Debugger for 68HC12. Depending on your application not all steps might be necessary.

For some applications additional steps might be necessary, that are not described in this Quick Start section.

1. Prepare the Start

Connect the Debug Cable to your target. There are two different connectors on the Debug Cable. Refer to chapter **BDM Connector** for detailed information.

It is not necessary to connect the Clock Cable for the first start.

Power up your TRACE32 system (This is not necessary on PODPC).

Start the TRACE32 Debugger Software.

Power up your Target!

⚠	To prevent damage please take care on this sequence all the time you are preparing a start.

2. Select the Clock for the BDM Communication

For the first start the BdmClock should be set to INT and the value in the field clock should meet the frequency of ECLK (Signal on PortE bit 4). Generally ECLK runs at the half frequency of EXTAL after reset. Alternative to using the menu the clock can be selected with the command **SYStem.BdmClock** *<value_of_eclk>*.

⚠	With the TRACE32 user interface all numbers are interpreted as hex numbers. To enter a decimal number, please type a dot after the last digit (e.g. 10.).

⚠	The internal clock generator can only generate each frequency entered in the clock field. The software will select the best one for your application. So don' t worry if the value you see differs from the one you entered.

3. Configure the Debugger according to the Needs of the Application

Most of the configuration can be done with the **SYStem** Window which provides all CPU specific settings. Use **System Settings …** in the **CPU** menu to open this window

Inform the debugger about the CPU type on your target. Select the correct CPU type from the pull down menu in the field **CPU** or with the command **SYStem.CPU** *<cpu_type>*.

NOTE: If you type `SYStem.CPU` to the command line followed by blank, the softkeys (below the command line) provide you with all supported derivatives.

```
SYStem.CPU M68HC12A
```

Set the **SYStem Options** in the option field corresponding to your target configuration and application program. Generally the SYStem Options can remain at the default values for the first start, except for banked applications.

If banking is in use, the debugger must know how the CPU is configured to access the banks. Items of interest are:

- Which of the higher address lines are used as address lines, which are used as ports (only HC12A4/F8)? This information is given with the command **SYStem.Option.Axx**.
- Which of the chip select lines are used as chip selects, which are used as ports (only HC12A4/F8)? This information is given with the command **SYStem.Option.CSxx**.
- How are the according bits in the MISC register set (only HC12DA128/HC12DG128)? This information is given with the command **SYStem.Option.ROMTST** and the **SYStem.Option.ROMHM**.

To use banking on HC12DA128 or HC12DG128 the **SYStem.Option.MEMEXP** must be switched to on.

For details please refer to the chapters **Using the MMU for HC12A4/F8** and **Using the MMU for HC12DA/DG128**.

4. Map the EPROM Simulator if Available (optional)

```
MAP.ROM 0x0--0x1FFFF
```

This command maps a standard 8 bit wide 27x010 EPROM.

5. Tell the Debugger Where it should use On-chip Breakpoints (optional)

```
MAP.BOnchip 0x1000--0x0ffff
```

By default the In Circuit Debugger for 68HC12 (ICD12) modifies the code to realize a breakpoint. This will not work for ROM or FLASH. To provide breakpoints in ROM/FLASH areas the CPU’ s on-chip breakpoints can be used (not HC12A4).

6. Enter Debug Mode

```
SYStem.Up
```

This command asserts a reset to the CPU and drives the line BKGD to GND. So the CPU will enter the “special” variant of the operating mode defined by the pins MODA and MODB, which must be configured by the target.

LAUTERBACH recommends to use single chip mode for starting from reset. In this case the CPU will enter Special Single Chip Mode without executing any code. So all registers will contain reset values. In all other cases the CPU will try to execute code after reset, until the debugger gets control on it. So some registers may contain unexpected values.

7. Load the Program

Before loading the code the CPU must be configured so that the memory can be accessed. Take care of the registers MODE, PEAR, MISC, **INITRG**, INITRM, **INITEE**.

If banking is in use the registers for the memory expansion and chip select unit must also be set correctly. Refer to the CPU' s technical summary and the chapters **Using the MMU for HC12A4/F8** and **Using the MMU for HC12DA/DG128**.

If you want to load code to internal FLASH or EEPROM memory, refer to the chapters

FLASH EEPROM Management and **EEPROM Management**.

When the CPU is prepared the code can be loaded. This can be done with the command **Data.Load.<file_format> <file>**. Applications can be loaded by various file formats. The format depends from the compiler. Here are some typical load commands for 68HC12 applications:

```
Data.Load.COSMIC <file>.cos12       ; load application file generated
                                    ; with a COSMIC compiler

Data.Load.Elf <file>.abs /verify    ; load application file generated
                                    ; with a HIWARE compiler and verify
                                    ; if it is written correct to memory

d.load.u <iar6812>.dbg              ; load application file generated
                                    ; with a HIWARE compiler
```

8. Initialize Program Counter and Stackpointer

Many compilers add these settings in the start-up code to the user program automatically. In this case no action is necessary. You can check the contents of Program Counter and Stack Pointer in the ***Register*** Window which provides the contents of all CPU Registers. Use ***CPU Registers*** in the ***CPU*** menu to open this window.

The Program Counter and the Stackpointer and all other registers can be set with the commands **Register.Set PC** *<value>* and **Register.Set SP** *<value>*. Here is an example of how to use these commands:

```
Register.Set PC 0xC000          ; Set the Program Counter to address
                                ; 0xC000

Register.Set SP 0xBFF           ; Set the Stack Pointer to address
                                ; 0xBFF

Register.Set PC main            ; Set the PC to a label (here: function
                                ; main)
```

9. View the Source Code

Use the command **Data.List** to view the source code at the location of the Program Counter.

Now the quick start is done. If you were successful you can start to debug. Lauterbach recommends to prepare a PRACTICE script file (*.cmm) to be able to do all the necessary actions with only one command. Here is a typical start sequence without EPROM simulator and banking:

```
B::                                ; Select the ICD device prompt

WinCLEAR                           ; Clear all windows

SYStem.Reset                       ; Bring all settings in the SYStem
                                   ; window to default value

SYStem.BdmClock 2000000.           ; Set frequency for BDM communication
                                   ; to 2 MHz

SYStem.CPU M68HC12B.               ; Select CPU type

MAP.BOnchip 0x08000--0x0ffff       ; Select on-chip breakpoints for the
                                   ; FLASH EEPROM area

SYStem.Up                          ; Reset the target and enter special
                                   ; mode

Data.Set 0x11 0x08                 ; Move the internal registers to 0x0800
                                   ; by writing the INITRG register

SYStem.Option.BASE 0x0800          ; Tell the debugger that the internal
                                   ; registers now can be found at address
                                   ; 0x0800

Data.LOAD.Elf hic.abs              ; Load the application - here an
                                   ; absolute file in ELF/DWARF format

Register.Set PC main               ; Set the PC to function main

Register.Set SP 0xBFF              ; Set the stack pointer to address 0xBFF

Data.List                          ; Open disassembly window *)

Register.view /SpotLight           ; Open register window *)

Frame.view /Locals /Caller         ; Open the stack frame with
                                   ; local variables *)

Var.Watch %SpotLight flags ast     ; Open watch window for variables *)
```

```
PER.view                            ; Show clearly arranged peripherals
                                    ; in window *)

Break.Set 0x100 /Program            ; Set software breakpoint to
                                    ; address 100
                                    ; (address 100 is outside of read-only
                                    ; range)

Break.0xSet 8024 /Program           ; Set on-chip breakpoint to
                                    ; address 8024
                                    ; (address 8024 is within read-only
                                    ; range)
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

You can find suggestions for such PRACTICE script files (*.cmm) in the TRACE32 demo folder **~~/demo/m68hc12/compiler**.

Refer to the **"MCS12 Debugger"** (debugger_hc12.pdf) and **"Debugger Tutorial"** (debugger_tutorial.pdf) or to the **"PRACTICE Script Language User's Guide"** (practice_user.pdf) how to do this. You can also find some information on basic actions with the debugger.

Please keep in mind that only the **Processor Architecture Manual** (the document you are reading in at the moment) is CPU specific, while all other parts of the online help are generic for all CPUs. So if there are questions related to the CPU the **Processor Architecture Manual** should be your first choice.

Restrictions

Reset Configuration	For using the In Circuit Debugger (BDM) the CPU should be configured for Single Chip Mode at reset. To meet this the target should have capability to pull the pins MODA and MODB to GND.
COP Function	The watchdog can only be used with the longest timer period. Refer to **SYStem.Option.WATCHDOG**.
On-chip Breakpoints on derivatives with paging	The built in breakpoints on some derivatives with paging react only on the addresses within the 64K memory map. They do not take care of the paging (e.g.: HC12DA/DG128). This is a problem of the CPU, which cannot be solved by the debugger.
TrBus commands do not work	Neither Trigger_Out, nor Trigger_In do work on the HC/MCS12. They are not implemented, because the reaction time would be too long.

SYStem.BdmClock — Select clock for BDM communication

Format: **SYStem.BdmClock AUTO** | **EXT** | **EXT/2** | **ECLK** | **INT** | *<value>*

AUTO	Clock source for BDM communication is the internal oscillator. The clock is calculated and set automatically with help of a synchronization mechanism. This selection can only work if the used processor contains a BDM implementation which offers the SYNC command (not available on LA-7714 and LA-7717).
EXT	Clock source is the frequency from clock cable or from pin 10 of the 10 pin connector. The Clock Cable is a flying lead connection.
EXT/2	Clock source is the frequency from clock cable or from pin 10 of the 10 pin connector divided by two. The Clock Cable is a flying lead connection.
ECLK	Clock source for BDM communication is the ECLK (PE4) of the processor. The ICD12 expects this signal on Pin 3 of the 6pin connector or Pin 8 of the 10 pin connector. If ECLK is selected, the CLKSW bit of the BDM Status Register is set.
INT	Clock source for BDM communication is the internal oscillator, which can be set with the command SYStem.BdmClock *<value>*.
<value>	SYStem.BdmClock *<value>* is used to set the frequency for the internal oscillator, which can operate in the range [1000. … 25000000. Hz]. If a value is entered the clock source for BDM communication is switched to the internal oscillator.

This command selects the source for the clock to be used for the communication between BDM an CPU. Generally this frequency has to match the frequency of ECLK (unstretched). At derivatives without PLL this frequency is half of the frequency applied to the CPU's EXTAL pin.

The internal oscillator cannot provide every frequency. The software uses the best setting for the selected value and shows it in the SYStem window in the field BdmClock.

If your application requires to debug with active PLL please refer to the chapter **Debugging with active PLL**.

SYStem.CONFIG — Configure debugger according to target topology

The **SYSTem.CONFIG** command group is not supported.

SYStem.CPU — Select CPU type

Format:	**SYStem.CPU** <*type*>
<*type*>:	**M68HC12A** | **M68HC12B** | **M68HC12BC** | **M68HC12D** | **M68HC12DA** | **M68HC12DG** | **M68HC12G** | **M68HC12F**

With this command the processor type is selected.

SYStem.LOCK — Lock and tristate the debug port

Format:	**SYStem.LOCK** [**ON** | **OFF**]

Default: OFF.

If the system is locked, no access to the debug port will be performed by the debugger. While locked, the debug connector of the debugger is tristated. The main intention of the **SYStem.LOCK** command is to give debug access to another tool.

SYStem.MemAccess — Select run-time memory access method

Format:	**SYStem.MemAccess Enable | StopAndGo | Denied** **SYStem.ACCESS** (deprecated)

Enable **CPU** (deprecated)	Memory access during program execution to target is enabled.
Denied (default)	Memory access during program execution to target is disabled.
StopAndGo	Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. For more information, see below.

Format:	**SYStem.Mode** <*mode*> **SYStem.Attach** (alias for SYStem.Mode Attach) **SYStem.Down** (alias for SYStem.Mode Down) **SYStem.Up** (alias for SYStem.Mode Up)
<*mode*>:	**Attach** **Down** **Go** **Up** **NoDebug**

Attach	This command works similar to Up command. The difference is that the target CPU is not reset. The BDM/JTAG/COP interface will be synchronized and the CPU state will be read out. After this command the CPU is in the SYStem.Up mode and can be stopped and debugged.
Down	Disables the Debugger. The state of the CPU remains unchanged.
Go	Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is in the system.up mode and running. Now, the processor can be stopped with the break command or until any break condition occurs.
StandBy	Not available for HC12/MCS12/S12X.
Up	Resets the target and sets the CPU to debug mode. After execution of this command the CPU is stopped and prepared for debugging. All register are set to the default value.
NoDebug	Disables the Debugger. The state of the CPU remains unchanged.

SYStem.Option.BASE — Base address of internal registers

Format:	**SYStem.Option.BASE** *<address>*

The **SYStem.Option.BASE** defines the base address of the internal registers. On HC12 target systems the user should always keep this address on the same value as the internal CPU register INITRG. (*<address>* is a 16 bit value). On MC9S12 and S12X targets the value of INITRG can be read by the DBM interface and so there is no action necessary.

The ICD needs to know, where the CPU' s internal registers are assigned to. This information is used to show the CPU' s internal registers in the peripheral window, which can be opened by the **PERipheral** command and to control some of the internal registers, when the EEPROM and FLASH programming algorithms or the on-chip Hardware Breakpoints are in use.

Technical background: The registers of the CPU12 family members can be assigned to any 2-Kilobyte boundary within the standard 64-Kilobyte address space. The location of the registers in the memory map of M68HC12X depends from the register INITRG (Adr. 0x0011 after reset). The difficulty with this is that INITRG is also mapped with the rest of the register block. If the INITRG is changed from 0x00 to 0x28 the address of INITRG changes from 0x0011 to 0x2811. After this the **SYStem.Option.BASE** is to be changed from 0x0 to 0x2800:

⚠	After reset (at SYStem.Up the CPU receives a reset) the register block resides at location 0x0000. If the **SYStem.Option.BASE** holds a value other than 0x0, the peripheral window for example shows nonsense.

SYStem.Option.CLKSW — Force BDM to work on CPU bus frequency

Format:	**SYStem.Option.CLKSW** [**ON** | **OFF**]

If the **SYStem.Option.CLKSW** is set to on, the debugger will set the CLKSW bit in the BDM Status register. This configures the BDM interface of the CPU to work on bus frequency. If this frequency changes (for example if an application switches to PLL clock) BDM communication will be lost if the user does not change the contents of **SYStem.Option.BdmClock** accordingly.

SYStem.Option.DUALPORT All memory accesses are done hidden

Format:	**SYStem.Option.DUALPORT** [**ON** | **OFF**]

If the **SYStem.Option.DUALPORT** is active, the debugger does all memory accesses by hidden access. So there will be no window freeze while executing user code. Disadvantage of doing so is that the hidden memory access may slow down the target application.

SYStem.Option.GLOBAL Memory accesses are done global

Format:	**SYStem.Option.GLOBAL** [**ON** | **OFF**]

On S12X targets two different views on the memory map are possible. One is similar to the view used for the MC9S12 with a 64k address room and memory expansion using page pointers in some ranges (Local Memory Map), the other view has one linear address map (Global Memory Map).

If you use commands without memory class and the **SYStem.Option.Global** is **off**, the TRACE32 software will use the memory classes p: and d: which are intended to be used for logical addresses (Local Memory Map).

If you use commands without memory class and the **SYStem.Option.Global** is **on**, the TRACE32 software will use the classes gp: and gd: which are intended to be used for global addresses (Global Memory Map).

SYStem.Option.MonBase — Monitor relocation

Format:	**SYStem.Option.MonBase** *<address>*

SYStem.Option.SOFTWORD — Breakpoints are set by word access

Format:	**SYStem.Option.SOFTWORD** [**ON** | **OFF**]

When a software breakpoint is set, the first byte of the original opcode is replaced with the command BGND (Opcode 0x00). If the desired memory location is located in a range which can only be written by word access, a special mechanism is required which does a read-modify-write operation. With the option you can activate this mechanism:

OFF	Software breakpoints are set by byte access.
ON	Software breakpoints are set by word access.

Format:	**SYStem.Option.VFP** [**ON** | **OFF**]

OFF	Programming voltage generation is inactive
ON	Programming voltage is applied to VFP pin

To program the FLASH EEPROM on some derivatives the necessary programming voltage can be applied to the VFP pin. This option is activated and deactivated automatically if the debugger' s capability to program the FLASH EEPROM is used.

The line VFP on the **BDM Connector** is at 5 V if **SYStem.Option.VFP** is off and at 12 V if **SYStem.Option.VFP** is on.

Further information can be found in the Chapters **FLASH EEPROM Management**.

Only HC12

Format:	**SYStem.Option.WATCHDOG** [**ON** | **OFF**]

The ICD12 can trigger the COP while in background Mode:

OFF	The Watchdog (COP) is **not** triggered by the emulator
ON	The Watchdog (COP) is triggered by the emulator while the CPU is in break mode. The debugger continuously generates write accesses to address 0x3F altering the write data between 0xAA and 0x55.

This mechanism allows to debug an application which needs an active Watchdog (COP). To make the mechanism work properly, the Watchdog Timer Rate must be switched to the slowest mode.

For details, please refer to **“Debugging with active Watchdog”**, page 33.

Hardware Breakpoints

Most derivatives of the CPU12 family contain a breakpoint module with two 16 bit registers. Refer to the *Technical Summary* of your derivative to check if hardware breakpoints are available.

⚠	The built in breakpoints on some derivatives with paging react only on the addresses within the 64K memory map. They do not take care of the paging (e.g.: HC12DA/DG128). This is a problem of the CPU, which cannot be solved by the debugger.

Example: If an on-chip breakpoint is set for example to address 68004 the CPU will break on fetching from 58004 (or from any other page).

⚠	The hardware breakpoints are controlled by registers located in the CPU' s register block. This block can be mapped with the INITRG register. To have the hardware breakpoints working proper the user must take care that the debugger knows the position of the registers all the time. This information is given by the **SYStem.Option.BASE**.

The on-chip hardware breakpoints can be used in the following modes:

Program Breakpoints

Generally the In Circuit Debugger for HC12 (ICD12) uses software breakpoints to realize program breakpoints. Software breakpoint means that the code at the desired memory location is modified by the debugger to make the CPU break when the program counter meets this address. After this the original contents of this memory location is restored.

This mechanism does not work proper if a certain memory area can only be written by word accesses. In this case use the **SYStem.Option.SOFTWORD**.

This mechanism cannot work in Read Only Memory. To provide breakpoints in ROM areas the CPU' s hardware breakpoints can be used. The memory ranges, where hardware breakpoints should be used, have to be defined with the command **MAP.BOnchip**.

```
MAP.BOnchip 0x1000--0x0ffff
```

With the command **Break.List** the actual breakpoint configuration can be checked.

Read and Write Breakpoints

Read and Write Breakpoints always use the CPU' s Hardware Breakpoints regardless off the ranges defined with **MAP.BOnchip**.

Read and Write Breakpoints can be set with the command **Break.Set…**:

```
Break.Set 0x4738 /w          ; The CPU will be stopped if there is a write
                             ; access to address 0x4738

b.s 0x0b223 /r               ; The CPU will be stopped if there is a read
                             ; access to address 0xB223
```

Data Breakpoints

Data Breakpoints always use the CPU’ s Hardware Breakpoints regardless off the ranges defined with **MAP.BOnchip**. To provide a Breakpoint on address match and data match both 16 bit registers are needed. So there is only one breakpoint of this type available.

Data Breakpoints can be set with the TrOnchip menu. Use ***OnChip Trigger…*** in the ***Trigger*** menu to open this window. The address is specified with the following command:

```
Break.Set 0x4738 /a          ; The CPU will be stopped if this address is
                             ; accessed and the data matches the value
                             ; specified in the TrOnchip window.
```

Onchip.Mode.DetailTrace — Detailed trace recording mode

```
Format:          Onchip.Mode.DetailTrace
```

Address and data of all cycles except for fetches and free cycles are recorded.

Onchip.Mode.EventTrace — Start recording after trigger event

```
Format:          Onchip.Mode.EventTrace
```

Recording starts after trigger event.

Onchip.Mode.FlowTrace — Flow trace mode

```
Format:          Onchip.Mode.FlowTrace
```

Flow trace mode.

Onchip.Mode.LoopTrace — Inhibit redundant entries

```
Format:          Onchip.Mode.LoopTrace
```

Flow trace inhibiting redundant entries to capture memory.

Onchip.Mode.CPU — Select CPU as onchip trace source

Format:	**Onchip.Mode.CPU**

Selects CPU as Onchip trace source.

Onchip.Mode.XGATE — Select XGATE as onchip trace source

Format:	**Onchip.Mode.XGATE**

Selects XGATE as Onchip trace source.

Onchip.Mode.BOTH — Select CPU adn XGATE as onchip trace source

Format:	**Onchip.Mode.BOTH**

Selects CPU and XGATE as Onchip trace source.

Onchip.Mode.TimeStamp — Enable onchip timestamps

Format:	**Onchip.Mode.TimeStamp**

TrOnchip Commands

TrOnchip.state

Display on-chip trigger window

Format:	**TrOnchip.state**

Opens the **TrOnchip.state** window.

Format:	**TrOnchip.Mode** <*mode*>
<*mode*>:	**BreakAORB** **BreakATHENB** **TraceAORB** **TraceATHENB** **OFF**

BreakAORB	Real-time execution is stopped if one of the two comparators A or B matches.
BreakATHENB	Real-time execution is stopped if as a first step the comparators A matzohs and then as a second step the comparator B matches.
TraceAORB	Recording cycles to the on-chip trace is stopped if one of the two comparators A or B matches.
TraceATHENB	Recording cycles to the on-chip trace is stopped if as a first step the comparators A matzohs and then as a second step the comparator B matches.
OFF	Complex on-chip debug features are disabled.

Some derivatives of 9S12 have an enhanced on-chip debug module which offers some complex trigger features and a small on-chip trace. Please refer to your CPU' s manual to check if your chip offers this feature. These debug features are based on two address comparators which can be specified in the following way:

```
Break.Set 0x8000 /Alpha /Read    ; Set the Debug Comparator A to match on
                                 ; a read access from the address 0x8000.

Break.Set 0x8010 /Beta /Write    ; Set the Debug Comparator B to match on
                                 ; a write access from the address
                                 ; 0x8100.
```

Note that it is a must to specify whether the access is read or write.

TrOnchip.RESet Set on-chip trigger to default state

Format: **TrOnchip.RESet**

Sets the on-chip trace and trigger module to reset state.

TrOnchip.XBreakt Enable crossbreak between S12 core and XGATE

Format: **TrOnchip.XBreak** [**ON** | **OFF**]

The hardware breakpoints of the HCS12 can only cover specific ranges. If you want to set a marker or breakpoint to a complex variable, the on-chip break resources of the HCS12 CPU may be not powerful enough to cover the whole structure. If the option **TrOnchip.VarCONVert is on** the breakpoint will automatically be converted into a single address breakpoint. This is the default setting. Otherwise an error message is generated.

TrOnchip.RESERVE Deprive debugger of address comparator use

Format: **TrOnchip.RESERVE** [**BPx**] [**ON** | **OFF**]

Reserve on-chip breakpoint comparators to be used by the target application (only HC11).

OFF	The on-chip breakpoint can be used by the debugger.
ON	The on-chip breakpoint is used by the target application.

Memory Class	Description
C:, P:, D:	Specify the same address-area (CPU-access)
G:, GC:, GP:, GD:	Global Memory accesses, specify the same address-area (CPU-access)
A:	Absolute memory access (requires MMU-table)
EEPROM:	EEPROM write
E:	Emulation memory access (dual-ported)
AP:	Physical address (68HC12A4/F8/DA128/DG128 only)

C:, P: and D:

This storage classes operate on the same physically memory. They are only used to be compatible with other emulation probes.

On S12X targets the numbers following the semicolon are taken as addresses in the local memory map.

G:, GC:, GP: and GD:

This storage classes are only available on S12X targets and operate on the same physically memory. They are only used to be compatible with other emulation probes. The numbers following the semicolon are taken as addresses in the global memory map.

EEPROM:

This storage class is used to program the internal EEPROM. On read cycles there is no difference to the access mode with **C:** or **D:**. On write cycles the monitor program executes an EEPROM write protocol.

```
Data.Set EEPROM:0E00 12 34
D.s      EE:    0E00 12 34            EE: can be used as short form
```

Debugging with active PLL

Here is a list of derivatives with suggestions on how to set up the debugger for using the PLL:

CPU	Mask	Action
MC68HC912A4	all Masks	Select ECLK as clock source.
MC68HC912D60	0F68K 1F68K 0K75F	Select ECLK as clock source.
MC68HC912D60	XF73K	Select any option than ECLK and take care that the BDM interface is supplied with a frequency at half value of EXTAL. Warning: Due to a chip problem the CPU might stop unexpected while debugging with active PLL.
MC68HC912DA128	0H55W 4H55W	Select any option than ECLK and take care that the BDM interface is supplied with a frequency at half value of EXTAL. Warning: Due to a chip problem the CPU might stop unexpected while debugging with active PLL.
MC68HC912DG128	0H55W 4H55W	Select any option than ECLK and take care that the BDM interface is supplied with a frequency at half value of EXTAL. Warning: Due to a chip problem the CPU might stop unexpected while debugging with active PLL.
MC68HC912DG128	0K50E	Select ECLK as clock source.

Debugging with active Watchdog

Applications with active Watchdog (COP) need a special treatment when the CPU is in background mode. Activating the COP without any special action will force the CPU to reset shortly after entering background mode (after a COP timer period), because the COP is not triggered. Most HC12/S12 derivatives can stop the COP in this case by setting a bit, typically named RSBCK, in the respective control register. Refer to the *Clocks and Reset Generator Module* specification of your CPU and search for registers named CPMUCOP or COPCTL. For older devices without RSBCK bit the **SYStem.Option.WATCHDOG** can be used.

For convenience, TRACE32 offers the following mechanism: When a change from the running state to the stopped state is detected, the debugger checks if the COP is enabled. If yes, it attempts to set the RSBCK bit and a message is generated to inform that the bit was modified by the debugger. If the bit cannot be set by some reason a warning is generated to the message **AREA** window.

Another way to deal with an active watchdog can be found in a modification of the application. To keep it disabled for debugging, you can patch the write access to COPCTL with NOPs.

FLASH EEPROM Management

The PRACTICE script file **flash.cmm** is located in the TRACE32 demo folder **~~/demo/m68hc12/flash/flash**. This file contains the necessary settings for the different derivatives with on-chip FLASH EEPROM.

- Take a copy of this file and modify the **Data.LOAD** command to the needs of your application.
- If you do not use the Programming Voltage Generator of the ICD12 supply the CPU' s VFP pin with the necessary voltage (12 V FLASH types only).
- Select the correct CPU type in the SYStem window.
- Start your modified copy of the file flash.cmm.

To be able to debug within FLASH EEPROM areas the CPU' s Hardware Breakpoints must be activated. Refer to chapter **Using Hardware Breakpoints**.

⚠	Using **FLASH.Erase** *<unit>* with **MC9S12DP** family members will also erase the three other units which belong to the same flash block. The reason can be found in the different sizes of flash pages in the memory (16 KByte) and block erasable hardware units (64 KByte). To avoid this, you can use the command **FLASH.Erase** *<range>*.

FLASH EEPROM on S12X Derivatives

Use **FLASH.AUTO ALL** instead of **FLASH.Program ALL** for programming a FLASH module with ECC check.

Technical background: Some S12X derivatives contain a FLASH module with ECC (Error Correction Code) functionality. These modules do not allow cumulative program cycles on the bytes within a phrase (Group of bytes associated with a group of ECC bits). Since there is no need to place code sections on phrase boundaries, cumulative writes within phrases can occur when downloading code. The mechanism of the **FLASH.AUTO** command avoids these cumulative writes since it collects all writes to the flash and allows to write this collection in one go.

EEPROM Management

The ICD12 supports easy writing to the internal EEPROM. It executes the necessary EEPROM program sequence instead of the normal write operation if a write access to the EEPROM is indicated. There are two ways to do this:

1. Using the storage class EEPROM (not MC9S12DP family)

   Any write access done by a command using the storage class EEPROM (shortening: EE) starts the EEPROM program sequence. So the EEPROM can be initialized with standard set or load commands using the storage class **EEPROM or EE:**

```
d.s EEPROM:0D00 2 3 4 5                ; setting bytes
d.s     EE:0D00 2 3 4 5                ; short form
d.s EEPROM:0D00++2ff 0ff               ; clear EEPROM
d.load.b eepromdat.bin EEPROM:0D00     ; loading a binary file named
                                       ; eepromdat.bin
```

   It is possible to load the application using the command

   Data.Load <file> EEPROM:

   because write accesses to RAM done by this command will not cause an error. The only disadvantage of this method is that it takes a lot of time.

⚠	IF the EEPROM is mapped to an address higher than 0x4000 on the **next generation HC12** derivatives **(MC9S12DP family members)** the debugger needs to know the position to separate the EEPROM from the FLASH EEPROM. This can be done with the command MMU.Create. Example: EEPROM is mapped to 0x4000 (INITEE = 0x41) MMU.Create 0x4000++0fff 0x4000

2. Using the program mechanism for the FLASH

   This method allows you to download an applications in one step into EEPROM and into FLASH. The set of commands used for programming FLASH was expanded for this purpose by the memory type EEPROM. To specify which writing mechanism must be used, the system has to be informed where is the FLASH and where the EEPROM memory. The definition for the EEPROM memory range is done by the command **FLASH.Create EEPROM** *<range>*. (This is not necessary on **next generation HC12** derivatives. Here the EEPROM is created by default as unit 1.)The rest of the handling is the same as for the FLASH EEPROM.

⚠	The command FLASH.ERASE… has no effect on the EEPROM. To erase the EEPROM use **Data.Set <range of EEPROM>** 0ff if flash programming is active or **Data.Set EEPROM:<range of EEPROM>** if flash programming is inactive.

An example of how to load an application to RAM, ROM, EEPROM for the HC12D60 in one step can be found in ~~**/demo/m68hc12/flash/eeprom.cmm**

⚠	Writes to EEPROM can only be successful, when the according protection bits in the register EEPROT (0x00F1, EEPROM Block Protect) are cleared.

Banked Applications

To support applications which use more than the 64K direct accessible memory paging is required. To activate banking on targets switch the **SYStem.Option.PAGING** to ON and set the **SYStem.Option.ROMHM** and the **SYStem.Option.TRANS** according to the needs of your application.

On MC9S12 and S12X targets the **SYStem.Option.PAGING** to ON

Background and Compatibility Information

There are two memory schemes to support banked applications. One is based on the memory model used in the ICE12 with the artificial expanded physical addresses, the other one is based on the memory model used in FIRE12 where all commands are based on logical addresses. The last one is easier to use and available for all derivatives except for HC12A4. For applications with HC12A4 refer to the chapter **Using the MMU for HC12A4/F8**.

To activate banking with the FIRE12 similar memory model switch the **SYStem.Option.PAGING** to ON. This is available for all MCS12 (Star12) derivatives.

For HC12DA/DG/DT128 both options are available. For new designs LAUTERBACH recommends to use FIRE12 based memory scheme. The ICE12 based memory model is still there to be compatible with old command files (*.cmm - files). See chapter **Using the MMU for HC12DA/DG/GT128** for further information.

SYStem.Option.PAGING — Banked applications

Format:	**SYStem.Option.PAGING** [**ON** | **OFF**]

The **SYStem.Option.PAGING** enables the support for banked applications on HC12 applications. It activates a memory scheme similar to the one used for FIRE12. No MMU is required, all address based commands (MAP.Bonchip, flash programming) are based on logical addresses.

⚠	Do not activate this option on S12X targets. On MC9S12 targets, you can use this option for compatibility to old PRACTICE (*.cmm) files. Do not use it for new designs.

SYStem.Option.RAMHM — Alternate RAM mapping

Format: **SYStem.Option.RAMHM** [**ON** | **OFF**]

The **SYStem.Option.RAMHM** must be set if the bit RAMHM is set in the CPU’s MISC register.

SYStem.Option.ROMHM — ROM in second half of map

Format: **SYStem.Option.ROMHM** [**ON** | **OFF**]

The **SYStem.Option.ROMHM** must be set if the bit ROMHM is set in the CPU’s MISC register. In this case page 6 of the FLASH EEPROM is visible from 0x4000--0x7fff.

SYStem.Option.TRANS — Transparent mode

Format: **SYStem.Option.TRANS** [**ON** | **OFF**]

The **SYStem.Option.Trans** has effect on logical addresses smaller then 64K. If it is on then accesses in this area show the 64K of memory as seen by the CPU in the current paging configuration. This is the transparent mode. If it is off then in banked areas page zero of this area is shown and the contents of the according page register has no influence. It has no effect on the memory access of the CPU executing user code.

Address	**Access to**
000000--00ffff	current 64K address space (when TRANS is on)
000000--00ffff	page 0 (when TRANS is off)
010000--0fffff	pages 1..0ff
100000--0fffffff	current 64K address space

A logical address alone doesn't unique identify the physical address, as the address depends also on the setup of the INITRG, WINDEF, MXAR, MISC, CSCTL0 and CSCTL1 registers. As a result, logical addresses should only be used, if the MMU registers were already setup. Accessing internal resources (RAM or peripherals) is handled like an access outside of the MMU window. The following schematic shows these relations for some examples:

```
preset: CSCTL0=30,CSCTL1=10, WINDEF=40

logical address:    0    3     8    5    6    7    (Hex)
                    |    |   |    16 bit       |
                   A21..A14  logical CPU address

                  -->  exp. physical address: 0bfc567


logical address:    0    1     4    5    6    7    (Hex)
                    |    |   |    16 bit       |
                     PAGE    logical CPU address

                  -->  exp. physical address: 0ff4567


logical address:    0    0     8    5    6    7    (Hex)
                    |    |   |    16 bit       |
                 current-mmu logical CPU address

                  -->  inside PROG , assume PPAGE=1
                  -->  exp. physical address: 0bf4567



logical address:    0    0     c    d    e    f    (Hex)
                    |    |   |    16 bit       |
                 current-mmu logical CPU address

                  -->  outside pages
                  -->  exp. physical address: 0ffcdef
```

To activate the correct address translation for breakpoints, the **MMU** command must be activated. The creation and activation of the MMU translation can be done automatically for some file formats during download. The following script will prepare the 68HC12A4 for using the MMU without additional address lines and with CSP0 line to select between RAM and ROM:

```
sys.res
y.res
mmu.res
map.res
sys.o csp0e on
sys.m ai
map.m fast
map.ram 0x0200--0x0ffff
map.ram 0x0bf0000++0x0ffff
map.opf 0x8000--0x0ffff
map.opf 0x0bf0000++0x0ffff
map.i

d.s 0x013 0x0e                 ; disable rom
d.s 0x16 0x0                   ; disable the watchdog
d.s 0x3c 0x30                  ; CSCTL0
d.s 0x3d 0x10                  ; CSCTL1
d.s 0x12 0x11                  ; set EEPROM to 0x1000
d.s 0x3e 0x5
d.s 0x0f0 0x0fc
d.s 0x0f1 0x0
d.s 0x37 0x40                  ; enable P-Paging

d.load.elf bankdemo.abs /spath /mmu

enddo
```

⚠	When accessing memory with physical addressing (A:) by the CPU the address for the CPU is transformed to a bank and offset using the MMU table. Physical addressing of emulation memory is always possible without transformation (EA:).

Local and Global Memory Map on S12X Targets

On S12X targets two different views on the memory map are possible. One is similar to the view used for the MC9S12 with a 64k address room and memory expansion using page pointers in some ranges (Local Memory Map), the other view has one linear address map (Global Memory Map)

If you use the memory classes g, gp: or gd: the debugger expects global addresses. This means that you have a linear memory map from 0x000000 to 0x7FFFFF (Global Memory Map).

If you use the memory classes p: or d: the debugger expects logical addresses. This means that you have a 64k address map (Local Memory Map). The S12X uses paging to do memory expansion for

- EEPROM using the 1K window from 0x800--0xBFF
- RAM using the 4K window from 0x1000--0x1FFF
- FLASH using the 16K Window from 0x8000--0xBFFF

The logical address in these ranges is composed by the address within the 64k memory map (A15..A0) and the byte above to select the desired page (A23..A16).

Examples:

Local Address	**Global Address**	**Comment**
P:0xFD8000	GP:0x7F4000	third flash page counting back from the last one.
D:0x021000	GD:0x003000	third ram page counting from the first one
D:0xFB0A23	GD:0x13EE23	5th eeprom page counting back from the last one
P:0xFDC100	GP:0x7FC100	there is no window with paging in this address range

Using the MMU for HC12DA/DG/DT128

Banked applications on HC912DA128, HC912DG128 or HC12DT128 are supported similar to HC812A4/F8. Refer to that chapter to get **BASIC** information. Different to that derivatives is that HC12Dx128 have no chip selects or address lines higher than A15. The memory expansion is done with the PPAGE register which contains the page index (Bit2--Bit0 of the PPAGE register are called PIX2--PIX0). So there is a different table for the expanded physical address:

Address in 64K area	**SYStem. Options**	**A23**	**A22**	**A21**	**A20**	**A19**	**A18**	**A17**	**A16**	**A15**	**A14**
0x0000--0x3FFF	MEMEXP off	0	0	0	0	0	0	0	0	0	0
	MEMEXP on	1	1	1	1	1	1	1	1	0	0
0x4000--0x7FFF	MEMEXP off	0	0	0	0	0	0	0	0	0	1
	MEMEXP on ROMHM off	1	1	1	1	1	1	1	1	0	1
	MEMEXP on ROMHM on	1	1	1	1	1	1	1	0	1	1
0x8000--0xBFFF	MEMEXP off	0	0	0	0	0	0	0	0	1	0
	MEMEXP on ROMTST off	1	1	1	1	1	PIX 2	PIX 1	PIX 0	1	1
	MEMEXP on ROMTST on	1	1	1	1	1	PIX 2	PIX 1	0	1	1
0xC000--0xFFFF	MEMEXP off	0	0	0	0	0	0	0	0	1	1
	MEMEXP on ROMTST off	1	1	1	1	1	1	1	1	1	1
	MEMEXP on ROMTST on	1	1	1	1	1	PIX 2	PIX 1	1	1	1

The expanded physical address lines A13 to A0 contain the same level as the according pins of the CPU.

The table shows that the expanded physical address depends on the address, the page index and on the bits ROMTST and ROMHM in the MISC register of the CPU. The emulator needs to know how these bits are configured in the application. This information is given with the following SYStem.Options:

SYStem.Option.MEMEXP Memory expansion

Format:	**SYStem.Option.MEMEXP** [**ON** | **OFF**]

The **SYStem.Option.MEMEXP** enables the support for banked applications. If it is off, then the address information on the CPU's pins is put on the emulator's memory, break and trace system directly. If it is on the expanded physical address is put on instead.

SYStem.Option.ROMTST FLASH EEPROM test mode

Format:	**SYStem.Option.ROMTST** [**ON** | **OFF**]

The **SYStem.Option.ROMTST** must be set if the bit ROMTST is set in the CPU's MISC register. In this case the CPU is running in the Flash EEPROM TEST mode, where the FLASH EEPROM is in use as four 32K windows located from 0x8000--0xffff. This option is only available if **SYStem.Option.MEMEXP** is activated.

Using the MMU for HC12A4/F8

Basics

To support memory expansion beyond 64K the ICD12 needs to know how the memory expansion and chip select unit of the CPU is used in the target application. To make the system work, an exact relation must be given between the logical address (address in the 64K area combined with the selected program or data page) and the physical address combined with the chip select generated by the CPU. This relation is given by an expanded physical address and an MMU table (Memory Mapping Unit table).

There were a few important expressions in this first paragraph. The following lines will describe these expressions.

Logical Address

The logical address is a combination of an address in the 64K address area and the selected program or data page. It contains 6 hexadecimal digits. The lower four digits contain the 64K address and the upper two digits contain the number of the program or data page. The following table shows a few examples:

Address in 64K Address Range	Contents of affected Page Register	Logical Address
0x8000	PPAGE = 0xF1	0xF18000
0x7124	DPAGE = 0x10	0x107124
0x400	EPAGE = 0x03	0x30400

Physical Address

The physical address is the address the CPU shows on its bus. It depends on the application which address lines are used and which not. To make the ICD12 know if an Address or ChipSelect is used or not there is a switch for each of the Addresses ADDR[21..16].

```
Format:     SYStem.Option.A16E [ON | OFF]
            SYStem.Option.A17E [ON | OFF]
            SYStem.Option.A18E [ON | OFF]
            SYStem.Option.A19E [ON | OFF]
            SYStem.Option.A20E [ON | OFF]
            SYStem.Option.A21E [ON | OFF]
```

If a line of PortG is used as address line the according SYStem.Option must be set to ON if it is used as general I/O it should be set to OFF.

```
Format:    SYStem.Option.CSP0E [ON | OFF]
           SYStem.Option.CSP1E [ON | OFF]
           SYStem.Option.CSDE [ON | OFF]
           SYStem.Option.CSD2E [ON | OFF]
           SYStem.Option.CS3E [ON | OFF]
```

If a line of PortF is used as chip select line the according SYStem.Option must be set to ON. If it is used as general I/O is should be set to OFF.

Expanded Physical Address

Physical address combined with the information on the chip select lines select a location in memory. To be compatible with the modular concept of TRACE32 the information on the chip select lines is translated to additional address lines. The following table shows the translation table for HC12A4/F8.

Active Chip Select	**A23**	**A22**	**A21**	**A20**
CS3	**0**	**0**	**1**	**0**
CSD	**0**	**0**	**0**	**1**
CSD2	**0**	**0**	**0**	**0**
CSP1	**0**	**1**	**CPU A21**	**CPU A20**
CSP0	**1**	**0**	**CPU A21**	**CPU A20**
all other cases	**1**	**1**	**CPU A21**	**CPU A20**

The expanded physical address range contains 23 address lines though the CPU has only 21 address lines. The chip select lines affect A[23..20] of the expanded physical address. IF CSD, CSD2 or CS3 are active the lines A21 and A20 contain levels which may be different to the levels on the CPU’s pins. A[19..0] contain the same levels as the CPU’s pins (These statements and the table are only valid if the address lines A21 to A16 on the CPU are in use as address).

The following table gives an overview on the relation between logical address and expanded physical address on the HC12A4:

Address in 64K Area	**Active Chip Select**	**A23**	**A22**	**A21**	**A20**	**A19**	**A18**	**A17**	**A16**
0x0000--0x03FF EWDIR = 1 EWEN = 1	CS3	0	0	1	0	1	1	PEA17	PEA16
0x0400--0x07FF EWDIR = 0 or EWEN = 1	CS3	0	0	1	0	1	1	PEA17	PEA16
0x7000--0x7FFF	CSD	0	0	0	1	PDA19	PDA18	PDA17	PDA16
0x8000--0xBFFF PWEN = 1	CSP1	0	1	PPA21	PPA20	PPA19	PPA18	PPA17	PPA16
0x8000--0xBFFF PWEN = 1	CSP0	1	0	PPA21	PPA20	PPA19	PPA18	PPA17	PPA16
all other cases		1	1	1	1	1	1	1	1

Address in 64K Area	**Active Chip Select**	**A15**	**A14**	**A13**	**A12**	**A11**	**A10**	**A9--A0**
0x0000-- 0x03FF EWDIR = 1, EWEN = 1	CS3	PEA15	PEA14	PEA13	PEA12	PEA11	PEA10	A9--A0
0x0400-- 0x07FF EWDIR = 0 or EWEN = 1	CS3	PEA15	PEA14	PEA13	PEA12	PEA11	PEA10	A9--A0
0x7000-- 0x7FFF	CSD	PDA15	PDA14	PDA13	PDA12	A11	A10	A9--A0
0x8000-- 0xBFFF PWEN = 1	**CSP1**	**PPA15**	**PPA14**	**A13**	**A12**	**A11**	**A10**	**A9--A0**
0x8000-- 0xBFFF PWEN = 1	**CSP0**	**PPA15**	**PPA14**	**A13**	**A12**	**A11**	**A10**	**A9--A0**
all other cases		**A15**	**A14**	**A13**	**A12**	**A11**	**A10**	**A9--A0**

Memory Mapping Unit

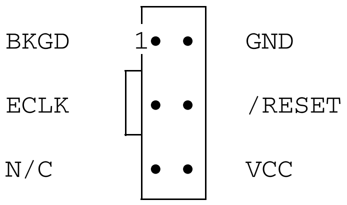
The MMU (Memory Mapping Unit) translation table is used for translating logical addresses to expanded physical addresses and vice versa. This table is specified with the commands concerning **MMU.**

On ICD12 there is a mechanism which calculates the correct expanded physical address from the logical address. This mechanism is started if the physical address is not specified when using the command **TRANSlation.Create**. In this case the logical to expanded physical address translation is done by reading the MMU registers of the CPU and calculating the expanded physical address dependent on the SYStem.Options concerning chip selects and higher address lines. This calculation doesn't take care about memory areas, which are overlaid by internal memory or I/O. It is strongly recommended to define all logical and physical addresses in the MMU table.

The breakpoints are based on the expanded physical address. So the MMU must be set correct to make them work proper.

BDM Connector

BDM Connector ICD - MCS12, ICD-S12X



This Pinout shows the male connector on the target board.

Some MCS12 devices need the ECLK (PE4) if debugging with active PLL is required. In this case the application must configure the register PEAR so, that PE4 drives ECLK. PE4 cannot be used as general I/O in this case.

⚠	On S12X devices ECLK (PE4) must not be connected if the ECLK frequency is higher than 25 MHz.

BDM Connector ICD - HC12

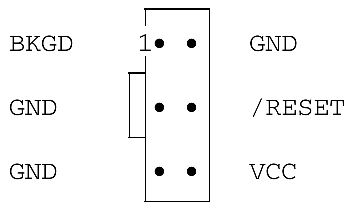
There are several connector definitions to connect the BDM Interface to a target. The type of supported connector(s) can be found by the serial number. The number to check is shown by the last 4 digits of the Serial Number on the BDM Dongle:

If the last 4 digits of the Serial Number show a number **smaller than 3000** the BDM supports original 6-pin version from FREESCALE **(1)** and if there is a 10-pin connector the 10-pin version from LAUTERBACH **(3)**.

If the last 4 digits of the Serial Number show a number **higher than 3000** the BDM supports modified 6-pin version from FREESCALEFREESCALE **(2)** and the 10-pin version from LAUTERBACH **(3)**.

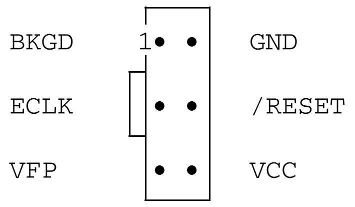
There is an additional small female connector on the BDM-Dongle. With the according wire you can supply the BDM with an external clock. This clock has to match EXTAL or EXTAL divided by 2. See **SYStem.BdmClock**.

1. Original 6-pin Version from FREESCALE



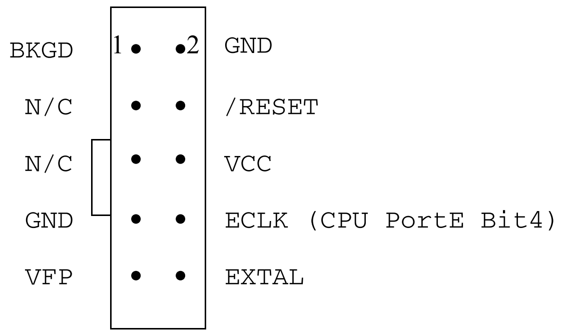
The pins 3 and 5 are connected to GND on the BDM's connector. Be careful in using this connector with the modified 6-pin Version from FREESCALE. You might cause short circuits to ECLK and VFP! If you want to stay compatible add jumpers to your design to disconnect ECLK and VFP from the BDM connector.

2. Modified 6-pin Version from FREESCALE



This Connector is supported if the last 4 digits of the Serial Number show a number higher than 3000. It is recommended to add a capacitor (100 nF … 1 uF between Pin5 (VFP) and GND) to the target, if the BDM should supply the cpu with the necessary programming voltage for the FLASH EEPROM.

3. 10-pin Version from LAUTERBACH



Be careful with the pins 3 and 5. They are connected to GND on the BDM's female connector. The drawing above shows how the male connector on the target should be connected.

Pin 10 (EXTAL) must not be connected, if there is a crystal used as clock source. It is recommended to add a capacitor (100 nF … 1 uF between Pin9 (VFP) and GND) to the target, if the BDM should supply the cpu with the necessary programming voltage for the FLASH EEPROM.