EMUL12-PC DA/DG128

Hardware Manual

Version 2.0

Contents

Emulator for the Motorola 68HC912DA128 and 68HC912DG128

General

The Emulator for Motorola's 68HC912DA128 and 68HC912DG128 (which will be referred to as POD-12DA128 and POD-12DG128 from this point on) supports full emulation of Motorola's 68HC912DA128 or 68HC912DG128 micro-controller.

The basic system is available in two different configurations:

1 A PC plug-in ISA board, a 25-pin cable, a hand-held HC12 Emulator Motherboard, a 68HC912DA128 or 68HC912DG128 CPU-Module, and a 5V power supply.
2 An EPC device, which connects the Emulator System to a standard PC parallel port, a hand-held HC12 Emulator Motherboard, a 68HC912DA128 or 68HC912DG128 CPU-Module, and a 5V power supply. This configuration is also suitable for laptop computers.

A trace is optional and can be added to both basic configurations. The trace in this case is a small card, which is connected on top of the hand-held Emulator Motherboard.

Most of the described emulator components are general to the HC12 family and can be used to support many other HC12 derivatives. The 68HC912DA128 CPU-Module and 68HC912DG128 CPU-Module are the only components specific only to the MC68HC912DA128 or MC68HC912DG128 CPU. By purchasing other CPU-Modules you will be able to upgrade your emulator to support new derivatives.

POD-12DA128 and POD-12DG128 are designed to plug into one of the following 112 pin QFP solder-down adapters in order to interface to the target:

1 No part-number known yet – Solder down adapter, Emulator to Target, using a standard Tokyo-Eletech TQFP 112 pin solder-down base.
2 No part-number known yet – Solder down adapter, Emulator to Target, using a Tokyo Eletech TQFP 112 pin solder-down base, which has a socket to place an optional D60 chip, in order to compare a stand-alone configuration to an emulator configuration.

Replacement solder-down base units are also available for the two adapters:

1 No part-number known yet – standard Tokyo-Eletech TQFP 112 pin solder down base replacement for the adapter in 1 above.
2 No part-number known yet – Tokyo-Eletech solder-down base replacement for the adapter in 2 above, which has a socket to place an optional D60 chip, in order to compare a stand-alone configuration to an emulator configuration.

All of the above mentioned adapters can be bought through Nohau.

Supported Features of POD-12DA128 and POD-12DG128

- All of the HC12 Operating Modes are supported. These include:
  A. Normal Single-Chip Mode.
  B. Normal Expanded Narrow Mode.
  C. Normal Expanded Wide Mode.
  D. Special Single-Chip Mode.
  E. Special Expanded Narrow Mode.
  F. Special Expanded Wide Mode.
  Switching between operating modes in run-time is also possible (see details below in section 'Switching the HC12 Operating Mode On-The-Fly').

- Bank Switching to emulate program paging of the internal 128Kbyte DA128 and DG128 Flash memory is fully supported.

- The sub-system which interfaces the Emulator to the Target, can be powered by any voltage within the range of 2.7V to 5.25V. On the Emulator there are 5V and 3.3V voltage sources which can be selected to supply power to this sub-system. This sub-system can also be configured to receive its power from the target.

- Ports A, B, E, and K are recreated by a Nohau Designed PRU (Port Replacement Unit) on the Emulator Motherboard, since these ports in the HC12 chip on the CPU-Module are required for the emulator use. The ports recreated by the PRU are fully CMOS compatible over the full 2.7V to 5.25V power supply range, in order to recreate the controller ports accurately. The pull-ups and pull-downs functions of the recreated ports are also supported. The reduced drive function of these ports is not supported. The recreated functionality of these ports is determined according to the HC12 operating mode used. In single-chip mode these ports will recreate I/O ports function. In expanded modes these ports will recreate the HC12 address, data, and control signals.

- The user software can be run out of either the emulation memory on the Emulator Motherboard, memory on the target (in expanded modes), or the internal HC12 RAM, EEPROM, and FLASH memory (utilities to program the internal HC12 FLASH and EEPROM are also supplied).

- The internal HC12 bus is always visible on the internal address/data bus of the Emulator. Thus, advanced features, such as tracing both internal and external HC12 accesses, use of Hardware Break-Points to break on internal and external accesses, and Shadow recording of internal and external accesses, are always available.

- All 112 pins going from the Emulator to the Target are available for probing by the user, using 112 header pins located on the CPU-Module.

- Emulation memory of 512K bytes (256K * 16bit) is available on the Emulator (expandable to 1Mbyte), to be used to emulate internal memory in Single-Chip Mode, or to replace target memory in Expanded Wide and Narrow Modes. This memory can be used to load the user program for emulation and to store user data. This memory supports emulation both the linear 64K memory and the 128K Program Paged memory to replace the internal DA128 and DG128 flash memory.

- Shadow Memory of 128K bytes records in real time all HC12 write accesses: both external and internal, including single-cycle miss-aligned word writes to the internal HC12 RAM, and writes to the internal HC12 SFR registers. This shadow memory is always available during emulation (when the emulator is running user code) to be read by the PC software and to display the HC12 memory content as it is changed by the user program.

- The Run-Time-Data Window allows shadow-like functionality. This window reads the actual HC12 memory locations during emulation (when the emulator is running user code) through the BDM interface. The advantage of using the Run-Time-Data over the Shadow memory is that it reads the actual HC12 memory locations. This is useful for reading internal registers, like timers or port pins, which can change during emulation with no information transferred on the internal HC12 data bus. These values will not update the Shadow Memory (since there is no data transfer on the internal HC12 data bus). The disadvantage of using the Run-Time-Data over the Shadow memory is that it is intrusive and may steal HC12 bus cycles (unlike the Shadow). The Run-Time-Data Window is also not always available, such as in cases where Hardware Breakpoints are being used, the HC12 is being reset, the HC12 is in Stop or Wait power-down modes, or whenever the BDM communication is not allowed by the HC12 controller on the CPU-Module. By contrast, the Shadow memory is always available.

- User writes from the software user interface to the HC12 memory (internal and external) are possible during emulation (when the emulator is running user code), through the BDM interface. This is done simply by selecting a location in an open RUN-TIME or SHADOW data window, typing a new value for the selected location, and pressing Enter. Writes to the HC12 memory are possible only when the BDM interface is available for communication (see the previous paragraph for details on when the BDM is available for communication).

- The internal HC12 Flash and EEPROM can be Erased, Blank-Checked, Programmed and Verified by the emulator software. Utilities within the PC software are supplied for this purpose. To blank-check and verify the content of the EEPROM no external power supply is required. To erase and program the Flash, an external 12V voltage is required. This voltage can be supplied either from special pins on the CPU-Module, or from the Target. Programming the internal HC12 Flash and EEPROM may be required, in late steps of the debug cycle, to verify that the program is executing properly from the actual internal HC12 memory, as it does in a real target. Since the DA128 and DG128 controllers are rated by Motorola for 100 cycles of Flash erase and program, it is recommended to program the internal Flash only in late design stages, to minimize the number of erase-program cycles. In case the internal Flash goes bad as a result of being programmed more than 100 times, you will need to replace the CPU-Module with a new one. Replacement DA128 and DG128 CPU-Modules can be bought from Nohau.

- An unlimited number of no-skid Hardware and Software breakpoints are available for use - Hardware Breakpoints are available for the entire linear 64K addresses and the entire 4M addresses of the Program Page. In the case of the DA128 and DG128 only the Linear 64K Hardware Breakpoints and 128K Program Page Hardware Breakpoints are useful. Hardware breakpoints are especially useful when the user program is running from the internal HC12 Flash or EEPROM memory, or in Expanded Modes when the user program is running from a non-volatile memory on the target. Software breakpoints can be used in most cases - when the user code is running out of a RAM based memory such as the Emulator emulation memory, the internal HC12 RAM, or (in expanded modes) target RAM memory. To place a software breakpoint click on the left gray area in the source window near the line

you want to place a breakpoint on. To place a hardware breakpoint press the ALT key and click on the left gray area in the source window near the line you want to place a breakpoint on.

- In Expanded Modes mapping the external HC12 memory to the target or the emulation memory on the Emulator Motherboard is possible. The mapping granularity is 1 byte, each external memory byte can be individually mapped to either the target or the emulator. Program Page memory is mapped only to the emulator in all cases (cannot be mapped to the target).

- All of the HC12 Reset sequences are supported and emulated. This includes internal HC12 resets such as COP watchdog Reset, clock monitor Reset, etc., as well as Reset from the Target, Reset from the emulator user interface software, and Reset on insufficient HC12 VDDX voltage.

- After every HC12 Reset the BDM communication has to be enabled before the emulator can be fully used. This enabling process takes approximately 1000 ECLK cycles. For this purpose, there is a short Reset-Transition code, which is run by the emulator until the HC12 BDM becomes available for communication. During this Reset-Transition phase the emulator also supplies the ability to configure the HC12 controller. This is in order to make it emulate the selected operating mode more accurately as defined by the user. The Reset-Transition phase always takes place when the emulator is coming out of Reset to Monitor Mode, where the HC12 is stopped and its internal state can be viewed and changed by the software user interface. When the emulator comes out of Reset to Emulation Mode (when the emulator is running user code), the Reset-Transition may or may not take place first before the user program is restarted. This depends on the setting of the 'Reset Transition' software switch in the Hardware Configuration window.

- The emulator keeps track of the changes to the base address of the SFR registers (the INITRG register). As a result, the SFR registers may always be inspected and edited referred by their names, using the software user interface by adding the SFR registers to a Register window.

- A low cost BDM emulator is offered as well.

Emulator and CPU-Module

Connectors and Connection Headers Description

The 25 Pin D-Type (J180) Connector (on the Emulator Motherboard)

The 25 Pin D-Type Connector located at the right side of the Emulator Motherboard is used to communicate between the Emulator and the PC. It is designed to connect either through a 25 pin cable to small ISA plug-in communication card, which should be installed in the PC, or through a parallel port communication device cable (referred to as 'EPC' by the Nohau terminology) which plugs to the PC parallel port.

The Power-Jack (J181) Connector (on the Emulator Motherboard)

The Power-Jack Connector located at the upper right side of the Emulator Motherboard is used to supply a 5V operating voltage to the emulator system (the Emulator Motherboard, the CPU Module, the Trace and the EPC - Parallel Port Communication Device). It is designed to plug into an external 5 volts power supply, which is supplied with the system.

The Two 50 pin Trace Connectors (J160 and J161) (on the Emulator Motherboard)

The two 50 pin Trace Connectors are designed to connect between the Emulator Motherboard and the Trace. These connectors carry all the required HC12 signals to be analyzed and recorded in the trace, as well as the signals required for the software to communicate with the Trace.

The Four Target Connectors (CON1, CON2, CON3 and CON4)

The four Target Connectors CON1 – CON4 carry up to 188 signal from the CPU-Module and PRU (Port Replacement Unit) to the Target, in order to emulate the HC12 controller. In case of the DA128 and DG128 controllers only 112 pins are used to carry all the 112 pin signals between the CPU-Module, the emulator motherboard and the target. These double sided header pins on the Emulator Motherboard are designed to connect between the Emulator and the Target system (either directly or through one of the adapters mentioned above at the beginning of this document), connecting on the bottom side of the Emulator. On the DA128 and DG128 CPU-Modules there are four Male-Female connectors which mate with the connectors on the Emulator Motherboard. The pins of these connectors stick-up from the CPU-Modules in order to allow all 112 DA128 or DG128 signals going to the target, to be monitored by the user by connecting them to a Trace Miscellaneous Connector, to an external Logic-Analyzer, an external Oscilloscope or another test device. All 112 signals are labeled according to the DA128 and DG128 pin names.

The Two Emulator Connectors (CON5 and CON6)

The two Emulator connectors – CON5 and CON6 carry internal emulator signals between the emulator motherboard and the HC12 controller on the CPU-Module. Found among these signals are the Address Data and Control signals. On the CPU-Module there are two connectors as well, which mate with the connectors on the emulator motherboard.

The GND (Ground) Wire

The GND Wire is a 6" black wire coming out of the Emulator Motherboard and ending with an EZ-Hook. This GND wire should be connected to the Ground of the Target. It is designed to supply a low resistance Ground connection between the Emulator and the Target, in addition to the Ground connection, through the adapter to the Target.

The VFP-IN Header (on the CPU-Module)

The VFP-IN Header is used to supply the erase and program voltage for the internal Flash memory of the HC12 controller on the CPU-Module (in case it is selected to supply this voltage by the VFP Jumper – see section 'The VFP Jumper' below for details). It is required to supply voltage to this header only in case Flash-Erase or Flash-Program is required. This header has 2 pins which are marked as GND and VFP. The Flash voltage supplied through this header should be applied when the positive potential is connected to the VFP pin and the negative potential to the GND pin.

Note: The CPU-Module has a Schottky Diode connected between the VFP pin of the HC12 and the VDD voltage of the HC12, in order to supply a default VDD voltage to the VFP pin of the HC12, when no other voltage is supplied. This is done in order not to damage the HC12 part. Another Schottky Diode is connected in series to the VFP pin of HC12 controller on the CPU-Module in order to protect the HC12 from voltage lower than VDD that also might damage the HC12. This Schottky Diode causes a voltage drop of less than 0.2V between the VFP supply voltage and the HC12 controller VFP pin.

The VCC-TP and GND-TP Test-Points (on the Emulator Motherboard)

The VCC-TP and GND-TP Test-Points may only be used as test points to measure the 5V operating voltage of the emulator. The GND-TP may also be hooked to the Ground of a Scope, a Logic Analyzer, or another test device in order to measure HC12 signals from the CON1-CON4 connectors on the CPU-Module. Do not supply voltage to these two test points since they are not designed for this purpose.

Select Jumpers Description

The EPC Jumper (on the Emulator Motherboard)

The EPC jumper enables or disables routing power supply voltage from the power jack on the emulator motherboard to the 25 pin D-Type connector also on the emulator motherboard. When a jumper top is present, 5V is supplied from the power jack to the EPC cable via the 25 pin D-Type connector, otherwise it is not.

The way in which the EPC jumper should be configured depends on your system configuration. Its name implies this, and is explained here for the different system configurations:

For Parallel Port Configuration

If your emulator system includes an EPC device which communicates to the PC through the parallel port, then a jumper top should be in place on the EPC jumper. This is in order to supply power to the EPC communication device from the external power supply through the power jack on the emulator motherboard, and through the 25 pin D type connector (the EPC cannot be powered by the PC parallel port).

PC Plug-In ISA Card Configuration

If your emulator system includes a small PC plug-in ISA card, then a jumper top should not be in place on the EPC jumper. This is in order to isolate the internal PC power supply from the external power supply connected to the emulator motherboard through the power jack.

The MC-PWR Jumper (on the CPU-Module)

The MC-PWR jumper selects which voltage source of the Emulator will power the VDD and VDDX voltage groups of the HC12 controller on the CPU-Module and the PRU buffers. This jumper has significance only in case these voltage groups are configured to be powered by an emulator voltage source (selected by the VDD and VDDX jumpers - see below for details).

When there is a jumper top on pins 1 and 2, a 5V voltage will be selected, and when there is a jumper top on pins 2 and 3, a 3.3V voltage (from an internal regulator on the Emulator Motherboard) will be selected. This jumper is factory configured to jumper top on pins 1 and 2 (5V is selected).

The VDD Jumper (on the CPU-Module)

The VDD jumper selects how to power the VDD pins of the HC12 controller on the CPU-Module. When the jumper top is on pins 1 and 2, the HC12 VDD pins are powered by the Emulator voltage, selected by the MC-PWR jumper (either 5V or 3.3V can be selected – see details above in paragraph 'The MC-PWR Jumper'). When the jumper top is on pins 2 and 3, the HC12 VDD pins are connected to the VDD pins of the adapter to the target, allowing the HC12 VDD pins to be powered by the target. This jumper is factory configured to jumper top on pins 1 and 2 (the VDD pins of the HC12 controller on the CPU-Module are powered by the Emulator power supply).

Note: It is recommended to have both the VDD jumper and the VDDX jumper configured the same, to have both of these HC12 voltage groups powered by the same voltage source – either the Emulator power supply or the Target.

The VDDX Jumper (on the CPU-Module)

The VDDX jumper selects whether the target or the emulator power supply powers the VDDX supply. This powers the VDDX pins of the HC12 controller on the CPU-Module and the PRU buffers on the Emulator Motherboard. When the jumper top is on pins 1 and 2(Pod), the Emulator VDDX is powered by the Emulator voltage and the voltage is selected by the MC-PWR jumper (either 5V or 3.3V can be selected – see details above in paragraph 'The MC-PWR Jumper'). When the jumper top is on pins 2 and 3(TRG), the Emulator VDDX is connected to the VDDX pins of the adapter to the target, allowing the Emulator VDDX to be powered by the target.

Note: It is recommended to have both the VDD jumper and the VDDX jumper configured the same, to have both of these HC12 voltage groups powered by the same voltage source – either the Emulator power supply or the Target.

This jumper is factory configured to jumper top on pins 1 and 2 (the Emulator VDDX is powered by the Emulator power supply).

The VSTBY Jumper (on the CPU-Module)

The VSTBY jumper selects how to power the VSTBY pin of the HC12 controller on the CPU-Module (powering the HC12 internal RAM when no power is supplied to the HC12 part). When the jumper top is on pins 1 and 2, the HC12 VSTBY pin is powered by the Emulator 3.3V voltage-regulator. When the jumper top is on pins 2 and 3, the HC12 VSTBY pin is connected to the VSTBY pin of the adapter to the target allowing the HC12 VSTBY pin to be powered by the target.

This jumper is factory configured to jumper top on pins 1 and 2 (the VSTBY pin of the HC12 controller on the CPU-Module is powered by the Emulator 3.3V voltage-regulator).

The VSSA Jumper (on the CPU-Module)

The VSSA jumper selects how to power the VSSA pin of the HC12 controller on the CPU-Module (powering the HC12 internal A/D converter). When the jumper top is on pins 1 and 2, the HC12 VSSA pin is connected to the general Emulator ground. When the jumper top is on pins 2 and 3, the HC12 VSSA pin is connected to the VSSA pin of the adapter to the target allowing the HC12 VSSA pin to be powered by the target.

Note: It is recommended to have both the VDDA jumper and the VSSA jumper configured the same, to power the internal HC12 A/D converter by either the Emulator power supply or the Target, and not by a combination of the two.

This jumper is factory configured to jumper top on pins 1 and 2 (the VSSA pin of the HC12 controller on the CPU-Module is connected to the Emulator ground).

The VDDA Jumper (on the CPU-Module)

The VDDA jumper selects how to power the VDDA pin of the HC12 controller on the CPU-Module (powering the HC12 internal A/D converter). When the jumper top is on pins 1 and 2, the HC12 VDDA pin is powered by the Emulator 5V power-supply. When the jumper top is on pins 2 and 3, the HC12 VDDA pin is connected to the VDDA pin of the adapter to the target allowing the HC12 VDDA pin to be powered by the target.

Note: It is recommended to have both the VDDA jumper and the VSSA jumper configured the same, to power the internal HC12 A/D converter by either the Emulator power supply or the Target, and not by a combination of the two.

This jumper is factory configured to jumper top on pins 1 and 2 (the VDDA pin of the HC12 controller on the CPU-Module is powered by the Emulator 5V power-supply).

The VRH0 and VRH1 Jumpers (on the CPU-Module)

The VRH0 and VRH1 jumpers select how to power the VRH0 and VRH1 pins of the HC12 controller on the CPU-Module (supplying the reference voltages for the HC12 internal A/D converters). When a jumper top is on pins 1 and 2, the respective HC12 VRH0 or VRH1 pin is powered by the Emulator 5V power-supply. When a jumper top is on pins 2 and 3, the respective HC12 VRH0 or VRH1 pin is connected to the respective VRH0 or VRH1 pin of the adapter to the target allowing the HC12 VRH0 and VRH1 pins to be powered by the target.

Note: It is recommended to have both the VRH0 jumper and the VRL0 jumper configured the same, to supply the reference voltage for the internal HC12 A/D converter zero by either the Emulator power supply or the Target, and not by a combination of the two. Similarly, the VRH1 and VRL1 jumpers should also be configured the same.

These jumpers are factory configured to jumper tops on pins 1 and 2 (the VRH0 and VRH1 pins of the HC12 controller on the CPU-Module are powered by the Emulator 5V power-supply).

The VRL0 and VRL1 Jumpers (on the CPU-Module)

The VRL0 and VRL1 jumpers select how to power the VRL0 and VRL1 pins of the HC12 controller on the CPU-Module (supplying the reference voltages for the HC12 internal A/D converters). When a jumper top is on pins 1 and 2, the respective HC12 VRL0 or VRL1 pin is connected to the general Emulator ground. When a jumper top is on pins 2 and 3, the respective HC12 VRL0 or VRL1 pin is connected to the respective VRL0 or VRL1 pin of the adapter to the target allowing the HC12 VRL0 or VRL1 pins to be powered by the target.

Note: It is recommended to have both the VRH0 jumper and the VRL0 jumper configured the same, to supply the reference voltage for the internal HC12 A/D converter zero by either the Emulator power supply or the Target, and not by a combination of the two. Similarly, the VRH1 and VRL1 jumpers should also be configured the same.

These jumpers are factory configured to jumper tops on pins 1 and 2 (the VRL0 and VRL1 pins of the HC12 controller on the CPU-Module are connected to the Emulator ground).

The VFP Jumper (on the CPU-Module)

The VFP jumper selects how to power the VFP pin of the HC12 controller on the CPU-Module (supplying voltage to erase and program the internal HC12 Flash memory). When the jumper top is on pins 1 and 2, the HC12 VFP pin is powered by a voltage received from the 2 pin VFP-IN header on the CPU-Module. When the jumper top is on pins 2 and 3, the HC12 VFP pin is connected to the VFP pin of the adapter to the target allowing the HC12 VFP pin to be powered by the target.

When no voltage is applied to the VFP pin of the HC12, a Schottky Diode on the CPU-Module supplies a default voltage of VDD (selected by the VDD jumper – see section 'The VDD Jumper' above for details) to this pin, as specified by Motorola. This is done in order to prevent damage to the HC12 part.

Another Schottky Diode is connected in series to the VFP pin of HC12 controller on the CPU-Module, in order to protect the HC12 from voltage lower than VDD (such low voltage might damage the HC12 controller). This Schottky Diode causes a voltage drop of less than 0.2V between the VFP supply voltage (either from the VFP-IN header or the target adapter VPF pin – as selected by the VFP jumper) and the HC12 controller VFP pin.

It is recommended by Motorola that whenever erase and program of the internal HC12 flash are not required, a VDD voltage will be supplied to the VFP pin of the HC12 controller. Thus it is also recommended to supply a default voltage of VDD to the VFP pin of the target adapter whenever Flash erase and program are not required.

Warning: It is very important when an erase and program voltage is supplied to the VFP pin of the HC12, that it would be within the recommended range specified by Motorola. This range tends to sometimes change from one mask-set to another. Therefore it s recommended to check the specified range for the mask-set used, before applying a voltage higher than VDD.

This jumper is factory configured to jumper top on pins 1 and 2 (the VFP pin of the HC12 controller on the CPU-Module is powered by the voltage received from the VFP-IN header also on the CPU-Module). For the default configuration, since no voltage source is supplied to the VFP-IN header, the VFP pin of the HC12 defaults to VDD using a Schottky Diode on the CPU-Module.

The VPLL Jumper (on the CPU-Module)

The VPLL jumper selects the emulator voltage that will power the VDDPLL voltage group of the HC12 controller and the CPU-Module. This jumper has significance only in case the VDDPLL jumper is configured to select the voltage from the Emulator (see below for details in section 'The VDDPLL Jumper').

When there is a jumper top on pins 1 and 2, a 5V voltage from the emulator will be selected to enable the internal HC12 PLL. When there is a jumper top on pins 2 and 3, the emulator Ground will be selected to disable the internal HC12 PLL, and increase the drive of the HC12 crystal pins.

This jumper is factory configured to jumper top on pins 2 and 3 (Ground is selected).

The VDDPLL Jumper (on the CPU-Module)

The VDDPLL jumper selects how to power the VDDPLL voltage group of the HC12 controller and the CPU-Module (powering the HC12 internal PLL and Crystal Generator). When the jumper top is on pins 1 and 2, the VDDPLL voltage group is powered by the emulator voltage as selected by the VPLL jumper (either 5V or GND – see details above in section 'The VPLL Jumper'). When the jumper top is on pins 2 and 3, the VDDPLL voltage group is connected to the VDDPLL pin of the adapter to the target allowing the VDDPLL voltage group to be powered by the target.

Note: It is recommended to have both the VDDPLL jumper and the VSSPLL jumper configured the same, to power the internal HC12 PLL and Crystal Oscillator by the same source - either the Emulator or the Target.

This jumper is factory configured to jumper top on pins 1 and 2 (the VDDPLL voltage group is connected to the emulator Ground through the VPLL jumper).

The VSSPLL Jumper (on the CPU-Module)

The VSSPLL jumper selects how to power the VSSPLL voltage group of the HC12 controller and the CPU-Module (powering the HC12 internal PLL and Crystal Generator). When the jumper top is on pins 1 and 2, the VSSPLL voltage group is connected to the general Emulator ground. When the jumper top is on pins 2 and 3, the VSSPLL voltage group is connected to the VSSPLL pin of the adapter to the target allowing the VSSPLL voltage group to be powered by the target.

Note: It is recommended to have both the VDDPLL jumper and the VSSPLL jumper configured the same to power the internal HC12 PLL and Crystal Oscillator by the same source - either the Emulator or the Target.

This jumper is factory configured to jumper top on pins 1 and 2 (the VSSPLL voltage group on the CPU-Module is connected to the Emulator ground).

The PLL/EXTAL and PLL/XTAL Jumpers (on the CPU-Module)

The PLL/EXTAL and PLL/XTAL jumpers select which clock source will feed the EXTAL and XTAL pins of the HC12 controller on the CPU-Module.

When the jumper tops of the two jumpers are on pins 1 and 2, the HC12 EXTAL pin will be connected to a PLL clock generator on the emulator motherboard, and the HC12 XTAL will be left open. The Emulator PLL clock generator automatically generates any frequency between 50KHz and 50MHz to an accuracy of 0.1% or higher, according to the specified frequency by the PC Hardware Configuration Window in the Clock field. When selected by these two jumpers it feeds the EXTAL pin of the HC12 controller on the CPU-Module.

When the jumper tops of the two jumpers are on pins 2 and 3, the HC12 EXTAL and XTAL pins are connected to the crystal and capacitor sockets: Y1, C-XT1 and C-XT2 on the CPU-Module. To supply the HC12 clock from the target, leave the Y1, C-XT1 and C-XT2 sockets unpopulated, and install jumper tops on the TRG-EXTAL and TRG-XTAL jumpers (see details below in section 'The TRG-EXTAL and TRG-XTAL Jumpers').

Note: It is recommended to have both the PLL/EXTAL and PLL/XTAL jumpers configured the same to supply the clock frequency to the HC12 either from the Emulator PLL Clock Generator or from the Crystal pin sockets or the Target, and not from a combination of these sources.

These two jumpers are factory configured to jumper tops on pins 1 and 2 (the EXTAL and XTAL pins of the HC12 controller on the CPU-Module are connected to the PLL clock generator on the emulator motherboard).

The TRG-EXTAL and TRG-XTAL Jumpers (on the CPU-Module)

The TRG-EXTAL and TRG-XTAL jumpers have significance only when the two PLL/EXTAL and PLL/XTAL jumpers have jumper tops on pins 2 and 3 (see details above in section 'The PLL/EXTAL and PLL/XTAL Jumpers').

When this is true, the TRG-EXTAL and TRG-XTAL jumpers allow feeding the EXTAL and XTAL pins of the HC12 controller on the CPU-Module by the EXTAL and XTAL pins of the adapter to the target.

When jumper tops are installed on these two jumpers, the EXTAL and XTAL pins of the HC12 controller on the CPU-Module are connected to the EXTAL and XTAL pins of the adapter to the target. When jumper tops are not installed on these two jumpers, the HC12 EXTAL and XTAL pins are isolated from the EXTAL and XTAL pins of the adapter to the target. This allows connecting components on the Y1, C-XT1 and C-XT2 sockets on the CPU-Module to drive the EXTAL and XTAL pin of the HC12 controller on the CPU-Module. See more details about the component sockets Y1, C-XT1 and C-XT2 below in section: 'The Y1, C-XT1 and C-XT2 Component Sockets'.

Note: It is recommended to have both the TRG-EXTAL and TRG-XTAL jumpers configured the same, to supply the clock frequency to the HC12 either from the Crystal pin-sockets on the CPU-Module or the Target and not a combination of the two sources.

These two jumpers are factory configured to jumper tops not installed (the Y1, C-XT1 and C-XT2 sockets on the CPU-Module may be populated with components to drive the EXTAL and XTAL pins of the HC12 controller on the CPU-Module).

The TRG-XFC Jumper (on the CPU-Module)

The TRG-XFC jumper selects how to feed the XFC pin of the HC12 controller on the CPU-Module. When a jumper top is installed on this jumper, the XFC pin of the HC12 controller on the CPU-Module is connected to the XFC pin of the adapter to the target allowing a PLL filter on the target to be used. When a jumper top is not installed on this jumper, the XFC pin of the HC12 controller on the CPU-Module is isolated from the XFC pin of the adapter to the target and may be connected to components on the R0, C0 and Ca sockets on the CPU-Module. See more details about the component sockets R0, C0, and Ca below in section: 'The R0, C0, and Ca Component Sockets'.

This jumper is factory configured to jumper tops not installed (the C0, R0, and Ca sockets on the CPU-Module may be populated with components to drive the XFC pin of the HC12 controller on the CPU-Module).

Component Sockets on the CPU-Module

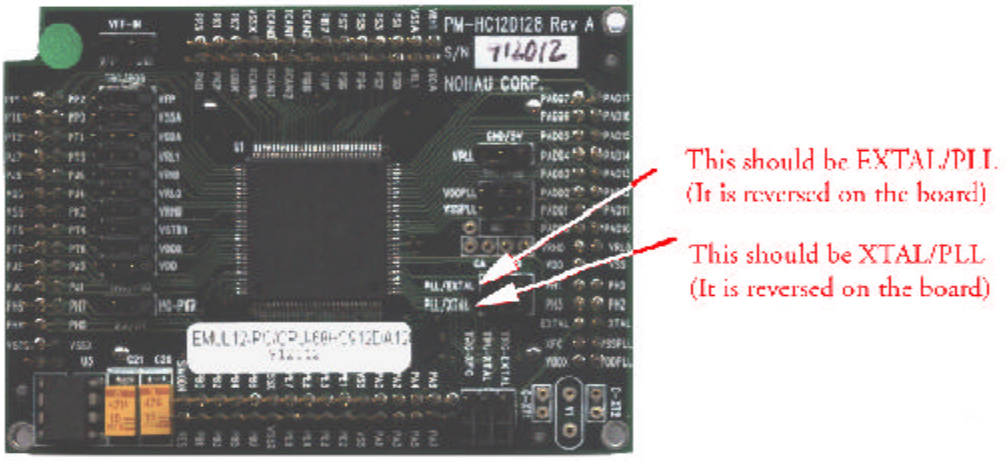
The Y1, C-XT1, and C-XT2 Component Sockets (on the CPU-Module)

The Y1, C-XT1, and C-XT2 component sockets may be used to mount a crystal and two capacitors, which may be connected to the EXTAL and XTAL pins of the HC12 controller on the CPU-Module. These component sockets are available on the CPU module due to the sensitivity of the internal HC12 clock generator circuit, and in order to shorten the length of the traces for the EXTAL and XTAL signals for stability.

In order to use these component sockets to mount a crystal and two capacitors which will drive the EXTAL and XTAL pins of the HC12 controller on the CPU-Module, the following jumpers must be configured as follows:

- The PLL/EXTAL jumper must have a jumper top on pins 2 and 3.
- The PLL/XTAL jumper must have a jumper top on pins 2 and 3.
- The TRG-EXTAL jumper must not have a jumper top installed over its two pins.
- The TRG-XTAL jumper must not have a jumper top installed over its two pins.

Note: On early versions of the boards, there was an error on the silkscreen. The error is described below. We will place stickers over the incorrect PLL/EXTAL and PLL/XTAL that will read: EXTAL/PLL and XTAL/PLL, respectively.



These component sockets are connected (when the above jumpers have jumper tops in the specified positions) as specified in the Motorola DA128 and DG128 data sheets:

- Y1 is connected between the HC12 EXTAL pin and VSSPLL.
- C-XT1 is connected between the HC12 XTAL pin and VSSPLL.
- C-XT2 is connected between the HC12 EXTAL pin and the HC12 XTAL pin.

Note: VSSPLL is selected by the jumper top position of the VSSPLL Jumper on the CPU-Module. Either the Emulator Ground of the VSSPLL pin of the adapter to the target can be selected (see details above in Section 'The VSSPLL Jumper').

The R0, C0, and Ca Component Sockets (on the CPU-Module)

The R0, C0, and Ca component sockets may be used to mount a resistor and two capacitors which may be connected to the XFC pin of the HC12 controller on the CPU-Module. These component sockets are available due to the sensitivity of the XFC PLL Filter circuit, and in order to shorten the length of the traces for the XFC signal for stability.

In order to use these component sockets to mount a resistor and two capacitors, which will be connected to the XFC pin of the HC12 controller on the CPU-Module, the TRG-XFC jumper must not have a jumper top installed over its two pins.

These component sockets are connected as specified in the Motorola data sheets:

- Ca is connected between the HC12 XFC pin and VDDPLL.
- R0 in series to C0 are connected between the HC12 XFC pin and VDDPLL as well.

Note: VDDPLL is selected by the jumper tops position of the VPLL Jumper and the VDDPLL Jumper on the CPU-Module (see details above in Sections 'The VPLL Jumper' and 'The VDDPLL Jumper').

LEDs Description

The POWER LED

The POWER LED, when turned on signals that a 5 volts power supply voltage is supplying operating voltage to the Emulator Motherboard and Emulator System.

The RESET LED

The RESET LED is used to signal when a Reset to the HC12 controller is active. Whenever any Reset source becomes active (internal to the HC12 or external) this LED will turn on for the duration of the Reset.

Note: Upon Power-Up of the Emulator before the software is started, and before the emulator is configured, the HC12 controller is held in Reset state in order to prevent it from executing before the emulator is initialized. In this specific case, although Reset is active, the RESET LED is not turned on and that's because the Emulator FPGA devices are not configured yet, and therefore, cannot turn this LED on.

The EMUL LED

The EMUL LED, when turned ON, signals that the emulator is running user code (Emulation Mode active). When turned OFF, it signals that the HC12 is stopped in order to inspect its internal state by the emulator (Monitor Mode active).

The STOP LED

The STOP LED signals that the HC12 controller is in the STOP Power-Down Mode. It is turned on whenever STOP Power-Down mode is entered, and is turned off as soon as the STOP mode becomes inactive by an interrupt or by a Reset.

The WAIT LED

The WAIT LED signals that the HC12 controller is in the WAIT Power-Down Mode. It is turned on whenever WAIT Power-Down mode is entered, and is turned off as soon as the WAIT mode becomes inactive by an interrupt or by a Reset.

The ERROR LED

The ERROR LED, when turned on, signals that an illegal emulator state has been invoked by the user code, or by the emulator. This illegal state is initiated by writing an illegal second value after the last HC12 Reset to the PEAR or the MODE HC12 registers (see details below in section 'The PEAR and MODE HC12 SFR Registers'). In such a case an illegal state is detected, the ERROR LED turns on, along with the HC12 and the emulator being forced into Reset state (in order to prevent the illegal ERROR-state from taking place causing bus contention and other emulator misbehaviors).

In order to leave the ERROR-state (if the ERROR LED is turned on), the user must click on the Reset icon in the software user interface or initiate Reset from the PC software in some other way.

Signal Description of the 112 DA128 and DG128 Target Signals

The VDD, VDDX, VSTBY, VDDA, VSSA, VRH0, VRH1, VRL0, VRL1, VFP, VDDPLL, VSSPLL, EXTAL, XTAL, and XFC Pins of the Adapter to the Target

These signals are associated with jumpers on the CPU-Module. Therefore refer to section 'Select Jumpers Description' for more details about these signals.

The VSS and VSSX Pins of the Adapter to the Target

There are two VSS signals and two VSSX signals on the adapter to the target. All four of these signals are connected to the Emulator System Ground (GND). These signals should also be connected to the Target main Ground (if a target is connected to the emulator) in order to supply a low resistance path between the Emulator Ground and the Target Ground.

The SMODN Pin of the Adapter to the Target

The SMODN pin is used as an input to determine the HC12 Operating Mode, which is set when Reset becomes active. It selects between Special and Normal Operating Modes.

This pin has a constant 33K pull-up resistor. This is used to default the HC12 to Normal Operating Modes in case no target signal is driving this pin during Reset.

During Reset which results in the emulator coming out in Monitor Mode (where the HC12 is stopped and its state may be viewed and changed by the user interface), the SMODN pin does not have effect on the Operating Mode, which will be implemented by the emulator. In this case, the Operating Mode, which will be implemented by the emulator, is set by the 'Operating Mode' field in Hardware Configuration window.

On the other hand, during Reset, which results in the emulator coming out in Emulation Mode (where the emulator is running user code), the SMODN pin does have effect on the Operating Mode, which will be implemented by the emulator. In this case, the SMODN signal along with the PE6 and PE5 signals, set the Operating Mode which will be implemented by the emulator right after Reset is over.

Refer to section 'HC12 Reset' below for details on which Reset sources cause the emulator to come out in Monitor Mode, and which cause it to come out in Emulation Mode. This SMODN pin does not recreate the BKGD or the TAGHI functionality. It only recreates the SMODN functionality as described above.

The RESET Pin of the Adapter to the Target

The functionality of the Reset signal is recreated by the PRU (Port Replacement Unit) on the Emulator Motherboard for the Target use. The recreated Reset signal is bi-directional as the Reset of the DA128 and DG128 controllers. When no Reset is active, this signal will be maintained high using a 33K pull-up resistor.

When Monitor mode is active (when the HC12 is stopped and its internal state can be viewed and changed by the user interface), a low level on the Reset signal driven by the target will not cause an HC12 Reset. This is in order to prevent an undesired Reset from occurring when the user code is stopped and the user is trying to inspect the state that the emulator has reached.

When the 'Reset From Target' check-box in the software Hardware Configuration Window is unchecked, a low level on the Reset signal driven by the target will also not cause an HC12 Reset. The 'Reset From Target' check box in the software Hardware Configuration Window is used to enable or disable recognition of Reset from the target to cause the HC12 and the emulator to be Reset.

Only when emulation is active (the emulator is running user code) and the 'Reset From Target' check-box in the software Hardware Configuration Window is checked, Reset from the target will cause the HC12 and the emulator to be Reset. In this case, a low level driven by the target to the Reset pin will cause the HC12 and the emulator to be Reset, and then resume operation after Reset becomes high again.

In the opposite direction, the recreated Reset pin will be driven low by the PRU, when an internal HC12 Reset (such as COP watchdog Reset for example) is detected by the emulator. When the internal Reset is over, the Reset pin will be driven high by the PRU through a 1K resistor for a short duration (40nSEC), and then maintained high by the constant 33K pull-up resistor.
The recreated Reset pin may also be driven low by the PRU with a Reset from the Software User-Interface, Reset on a low HC12 VDDX voltage, or Reset on ERROR. In order to enable this pin to be driven low by the PRU in these cases, the 'Emul Reset To Target' check-box in the software Hardware Configuration Window should be checked (reporting emulator Resets to the Target Reset pin is enabled). In these cases, when the Reset is over, the Reset pin will be driven high by the PRU through a 1K resistor for a short duration (40nSEC) and then maintained high by the constant 33K pull-up resistor.
After Reset is over a delay of up to 1000 ECLK cycles will usually take place before the user program will start running again. See more details below in section: 'Reset Transition'.

Port Pins PAD17-PAD10, PAD07-PAD00, PT7-PT0, PS7-PS0, PP3-PP0, PJ7-PJ0, PH7-PH0, PIB7-PIB6, TCAN2-TCAN0, and RCAN2-RCAN0 of the Adapter to the Target

These Port pins are connected directly between the DA128 or DG128 HC12 controllers on the CPU-Module and the Target adapter. The emulator does not interfere with these port pins in any way. The full functionality of these pins is supported.

Port Pins PE7-PE0 of the Adapter to the Target

The eight PE7-PE0 Port E signals are recreated for the target use by the PRU (Port Replacement Unit) implemented on the Emulator Motherboard. The reason these pins are not available directly from the HC12 controller on the CPU-Module is that the emulator is using them for its internal use as HC12 control signals.

The functionality recreated by the PRU for the target use of these eight Port E signals depends on the selected HC12 operating mode, and the configuration of the PEAR, MODE, DDRE and PORTE SFR registers. It is following the Port E functionality described in the DA128 and DG128 manuals, with some minor adjustments to make these signals suitable for the emulator use. The recreated functionality for these signals is described in detail on a pin by pin basis below.

The pull-up and pull-down functionality of the Port E signals is also recreated.

The PE7, PE3, PE2, PE1, and PE0 signals have 33K pull-up resistors which may be activated and deactivated by manipulating the PUPE bit in the PUCR register.

The PE6 and PE5 signals have 33K pull-down resistors which are activated during every HC12 Reset. These pull-down resistors are used, along with the SMODN input, to default the emulator to Normal Single-Chip Mode in case no target signals are driving these pins during Reset.

The reduced drive functionality of the Port E signals is not recreated.

The PE7/DBE\/ECLK\/CAL Signal

In Single-Chip Modes the recreated PE7 signal always functions as a general-purpose I/O signal or the Calibration Reference. In Expanded-Wide and Expanded-Narrow Modes, the recreated PE7 signal may be configured to function either as the DBE control signal, an inverted ECLK control signal, the Calibration Reference signal, or as a general-purpose I/O signal. The selected functionality of this signal depends on the NDBE, DBENE, and CALE bits in the PEAR register.

When configured as the DBE control signal, in Expanded-Wide and Expanded-Narrow Operating Modes, the recreated DBE signal will become low on external real non-free read accesses.

Note: At the time of writing the DA128 and DG128 controllers had insufficient DBE timing to be used to enable read data to the data bus (the DBE signal becomes active very late during external read access, after the time the read data is latched inside the HC12 controller). Therefore, it is recommended to avoid using the DBE for such purpose.

When configured as an inverted ECLK signal, in Expanded-Wide and Expanded-Narrow Operating Modes, the recreated inverted ECLK signal will switch states (become low and then high again) during every bus cycle and ECLK cycle, regardless of which memory is accessed by the bus cycle. Meaning, in

Expanded Operating Modes the inverted ECLK signal will also switch states in case of free cycles and accesses to internal HC12 memory resources. This inverted ECLK functionality differs from the inverted ECLK behavior of the HC12 DA128 and DG128 controllers, which do not switch in the described cases. The recreated inverted ECLK signal will stretch for external bus accesses which are configured to include wait states (by the EXSTR0 and EXSTR1 bits in the HC12 MISC register).

When configured as the Calibration Reference, this signal will only be synchronized to the HC12 controller in case the emulator's PLL clock generator is selected to feed the HC12 EXTAL pin. (This is done by the PLL/EXTAL and PLL/XTAL jumpers on the CPU-Module - see details in section 'The PLL/EXTAL and PLL/XTAL Jumpers' above). Otherwise this signal will have a very close frequency to the real HC12 Calibration Reference, but will not be synchronized to the HC12 internal clocks.

When the recreated PE7 signal functions as general-purpose I/O signal it may be configured, read, and written by accessing the DDRE and PORTE SFR registers.

The PE6/MODB/IPIPE1 and PE5/MODA/IPIPE0 Signals

During Reset the recreated PE6 and PE5 behave as the MODB and MODA inputs.

During Reset which results in the emulator coming out in Monitor Mode (where the HC12 is stopped and its state may be viewed and changed by the user interface), the PE6 and PE5 pins do not have effect on the Operating Mode, which will be implemented by the emulator. In this case, the Operating Mode, which will be implemented by the emulator is set by the 'Operating Mode' field in Hardware Configuration window.

On the other hand, during Reset which results in the emulator coming out in Emulation Mode (where the emulator is running user code), the PE6 and PE5 pins do have effect on the Operating Mode which will be implemented by the emulator. In this case, the PE6 and PE5 signals along with the SMODN signal determine the Operating Mode which will be implemented by the emulator right after Reset is over.

During Reset, the PE6 and PE5 have two 33K pull-down resistors activated and the SMODN has a constant 33K pull-up resistor. This defaults the emulator to Normal Single-Chip Operating Mode when no signal is overriding these pull-downs and pull-up values. After reset is over, the two pull-down resistors on PE6 and PE5 are automatically disabled, which recreates the functionality of the DA128 and DG128 controllers.

When Reset is not active, the PE6 and PE5 signals are used either as general-purpose I/O port signals or as the IPIPE1 and IPIPE0 control signals. In Single-Chip Mode or when the PIPOE bit in the PEAR register is cleared, the PE6 and PE5 pins function as general-purpose I/O port signals (configured read and written by accessing the DDRE and PORTE SFR registers). When Expanded Wide or Expanded Narrow Operating Modes are being emulated, and the PIPOE bit in the PEAR register is set, these pins function as the IPIPE1 and IPIPE0 control signals. The CGMTE functionality of PE6 is not supported.

The PE4/ECLK Signal

In Expanded-Wide and Expanded-Narrow Operating Modes, the recreated PE4 signal always functions as the ECLK control signal. In Single-Chips Mode the recreated PE4 signal may be configured to function either as a general-purpose I/O signal or as the ECLK control signal, depending on the NECLK bit in the PEAR register and the ESTR and IVIS bits in the MODE register.

In Expanded-Wide and Expanded-Narrow Operating Modes, the recreated ECLK signal will switch states (become high and then low again) during every bus cycle, regardless of which memory is accessed by the bus cycle. Meaning, in Expanded Operating Modes the ECLK signal will also switch states in case of free cycles and accesses to internal HC12 memory resources. This ECLK functionality differs from the ECLK behavior of the HC12 DA128 and DG128 controllers, which do not switch in the described cases. The recreated ECLK signal will stretch for external bus accesses, which are configured to include wait states (by the EXSTR0 and EXSTR1 bits in the MISC register).

In Single-Chip Modes when the recreated PE4 pin is configured to function as a general-purpose I/O port signal. It may be configured, read, and written by accessing the DDRE and PORTE SFR registers.

In Single-Chip Modes when the recreated PE4 pin is configured to behave as the ECLK signal, it drives a free-running ECLK signal to the target.

The PE3/LSTRB\ Signal

In Single-Chip Modes and Normal Expanded-Narrow Mode the recreated PE3 signal always functions as a general-purpose I/O port signal.

In Expanded-Wide Modes and Special Expanded-Narrow Mode, the recreated PE3 signal may be configured to function either as a general-purpose I/O signal or as the LSTRB control signal, depending on the LSTRE bit in the PEAR register.

When the PE3 signal is configured to function as a general-purpose I/O port signal, it may be configured, read, and written by accessing the DDRE and PORTE SFR registers.

When the PE3 signal is configured to function as the LSTRB control signal, it will indicate during every ECLK cycle if the Port B pins carry valid data. The LSTRB signal may change its state during every bus cycle (internal and external) which differs from the LSTRB behavior of the DA128 and DG128 controllers, which switch states only during external bus cycles. This difference should not cause any problem however.

The PE2/R/W\ Signal

In Single-Chip Modes the recreated PE2 signal always functions as a general-purpose I/O port signal.

In Expanded-Wide and Expanded-Narrow Modes, the recreated PE2 signal may be configured to function either as a general-purpose I/O signal or as the R/W\ control signal, depending on the RDWE bit in the PEAR register.

When the PE2 signal is configured to function as a general-purpose I/O port signal, it may be configured, read and written by accessing the DDRE and PORTE SFR registers.

When the PE2 signal is configured to function as the R/W\ control signal, it will indicate during every ECLK cycle if data read access or data write access is in progress (all free cycles behave as data read cycles). The R/W\ signal may change its state during every bus cycle (internal and external) which differs from the R/W\ behavior of the DA128 and DG128 controllers in this case, which switch states only during external bus cycles. This difference should not cause any problem however.

The PE1/IRQ\ Signal

The recreated PE1 signal always functions as a general-purpose input port signal and the IRQ interrupt request input in all Operating Modes.

Reading the PORTE register will read the state of the PE1 signal to the HC12.

When the IRQ interrupt is enabled internally in the HC12, a low level or a falling edge on the PE1 pin (depending on the internal configuration), will cause an IRQ interrupt request to the HC12 controller.

The PE0/XIRQ\ Signal

The recreated PE0 signal always functions as a general-purpose input port signal and the XIRQ interrupt request input in all Operating Modes.

Reading the PORTE register will read the state of the PE0 signal to the HC12.

When the XIRQ interrupt is enabled internally in the HC12, a low level on the PE0 pin will cause an XIRQ interrupt request to the HC12 controller.

Port Pins PA7-PA0 and PB7-PB0 of the Adapter to the Target

The Port A and Port B signals are recreated for the target use by the PRU (Port Replacement Unit) implemented on the Emulator Motherboard. The reason these pins are not available directly from the HC12 controller is that the emulator is using them for its internal use as its address/data bus. The functionality recreated by the PRU for the target use of the Port A and Port B signals depends on the selected HC12 operating mode. The reduced drive functionality of the Port A and Port B signals is not recreated.

In Case Single-Chip Mode is implemented

In Single-Chip mode the PRU recreates the required I/O ports functionality for these pins. This means that, in this case, the value of the Port A and Port B pins may be read and written by the HC12 reading and writing the PORTA, PORTB, DDRA and DDRB registers. In this case also the pull-up functionality of the Port A and Port B pins is recreated and may be activated and deactivated by manipulating the PUPA and PUPB bits in the PUCR register. The recreated value of the pull-ups resistors for these pins is 33Kohm.

In case Expanded-Wide or Expanded-Narrow Mode is implemented

In Expanded Wide and Expanded Narrow Modes, the PRU recreates the required address/data bus functionality for the Port A and Port B pins. This means that, in this case, the emulator will drive the HC12 address on the Port A and Port B pins during the address phase of every bus cycle.

During the data phase of every bus cycle, the recreated functionality of the Port A and Port B pins depends on which resource is accessed by the HC12, and the bus width implemented.

- If the access is to an internal HC12 memory resource or the SFR register then the recreated Port A and Port B signals will be tri-stated during the data phase of the cycle (since no target memory should be accessed in this case).
- If the access is to an external memory, but the address accessed is mapped to the emulator emulation memory, then the recreated Port A and Port B signals will also be tri-stated during the data phase of the cycle (since no target memory should be accessed in this case either).
- If the access is to an external memory and it is mapped to the target, then the recreated Port A and Port B signals will drive/read the written/read data to/from the target, during the data phase of the bus cycle. The target memory should be accessed in this case. The width of the data read/written depends on the required bus width: In Expanded Narrow Mode 8 bit data is written/read using the Port A signals, while the Port B signals are tri-stated (during the data phase of the bus cycle). In Expanded Wide Mode (in all cases except for the next one) 16 bit data is written/read using the Port A and Port B signals, during the data phase of the bus cycle. In Expanded Wide Mode, when the NRDF bit in the MODE register is set and the access is to one of the 200H addresses following the SFR registers, the bus behaves as a narrow data bus. Therefore, in this case, 8 bit data is written/read using the Port A signals while the Port B signals are tri-stated during the data phase of the bus cycle.

Port Pins PK7 and PK3-PK0 of the Adapter to the Target

The five Port K signals (PK7 and PK3-PK0) are recreated for the target use by the PRU (Port Replacement Unit) implemented on the Emulator Motherboard. The reason these pins are not available directly from the HC12 controller on the CPU-Module is that the emulator is using them for its internal use as HC12 page address and chip select signals.

The functionality recreated by the PRU for the target use of these five Port K signals, depends-on the selected HC12 operating mode, and the configuration of the MODE, DDRK and PORTK SFR registers. It is following the Port K functionality described in the DA128 and DG128 manuals. The recreated functionality for these signals is described in detail below.

The recreated Port K signals have two major functions which may be recreated. The recreated function is selected by the emulated operating mode and the value of the recreated EMK MODE register bit.

In case Single-Chip mode is emulated or in case the EMK bit in the recreated MODE register is cleared, the target Port K pins will recreate the General Purpose I/O Port function. In this case, the value of the Port K pins may be read and written by the HC12 reading and writing the PORTK and DDRK registers.

In case Expanded wide or Narrow Mode is emulated and the EMK bit in the recreated MODE register is set, the Port K signals will recreate the Page Address signals and Chip Select signal. In this case, the Port K pins 2,1, and 0 will drive the value of the HC12 PPAGE register to the target. The Port K pin 7 will drive the ECS signal to the target (which will become active – low, for any access to the PPAGE window at 8000H-BFFFH). Also in this case the Port K pin 3 will be constantly tri-stated.

The pull-up functionality of the Port K signals is also recreated.

The PK7 and PK3-PK0 signals have 33K pull-up resistors, which may be activated and deactivated by manipulating the PUPK bit in the PUCR register.

The reduced drive functionality of the Port K signals is not recreated.

The PEAR and MODE HC12 SFR Registers

The PEAR (Port E Assignment Register) and the MODE SFR registers, behave in a special way in the emulator. The HC12 controller on the CPU-Module runs in Special Expanded Wide Mode in order emulate the Single-Chip and the Expanded-Wide Operating Modes. Similarly it runs in Special Expanded-Narrow Mode in order to emulate the Expanded-Narrow Operating Modes.

The emulator, however, should emulate the required operating mode and the required Port E control signals, as defined by the user and not the Special Expanded-Wide or Special Expanded-Narrow Mode, which the HC12 controller on the CPU-Module runs.

For this reason, the emulator and the HC12 controller on the CPU-Module maintain separate different copies of the PEAR and the MODE registers. In the emulator the PEAR and the MODE registers are configured to support the emulated operating mode and the recreated Port E and Port K I/O signals for the target use. In the HC12 controller on the CPU-Module, on the other hand, the PEAR and the MODE registers are configured to support the actual HC12 operating mode which is Special Expanded-Wide or Special Expanded-Narrow Mode as mentioned above.

Thus, the MODE register in the HC12 controller on the CPU-Module always contains the value 01111011B or 00111011B to implement the Special Expanded-Wide Mode or Special Expanded-Narrow Mode accordingly. Similarly, the PEAR register in the HC12 controller on the CPU-Module always contains the value 2CH to generate the Port E control signals required by the emulator for its internal use.

On the other hand, the MODE register and the PEAR register, in the emulator, may contain any value in order to recreate all of the operating modes and all of the Port E signals properly. After Reset, these registers will be set to their default Reset values, according to the recreated operating mode as described in the DA128 and DG128 manual.

In order to enable the user to change the emulator configuration from its default Reset state, the emulator MODE register and the emulator PEAR register, may be written ONCE each after HC12 Reset. During the first write to the MODE register and the first write to the PEAR register, the emulator MODE and PEAR registers are written by the data from the HC12 data bus.

During the first write to the MODE register and the first write to the PEAR register after each Reset, the MODE and PEAR registers in the HC12 controller on the CPU-Module maintain their value. The HC12 controller, on the CPU-Module, runs in Special Operating Mode where the first write to these registers is ignored Thus, the state of the Special Operating Mode and the Port E and Port K control signals generated by the HC12 controller on the CPU-Module are maintained in this case, to maintain the valid internal configuration of the emulator.

Attempting to write to the MODE register or the PEAR register more than one time after each Reset will cause the ERROR state to become active if the value written is different than the allowed values as described below.

After the PEAR register is written once after the last HC12 Reset, it may not be written again by any value different than 2CH, until another HC12 Reset is issued. If it is written by a value different than 2CH (on the second or later write after the last HC12 Reset), then the ERROR state will be entered, turning on the ERROR LED and forcing the HC12 and the emulator into Reset state. The ERROR state is entered because the described second or later write to the PEAR register changes the configuration of the Port E

control signals of HC12 controller on the CPU-Module in a way which does not support emulation. Therefore the ERROR state is entered to prevent bus contention and other undesired misbehaviors of the emulator.

After the MODE register is written once by the user code or from Seehau user interface after the last HC12 Reset, it may be written again only to change the EBSWAI bit or to conclude a complex operating mode switch as described in the next section called "Switching the HC12 operating mode on-the-fly". During a second or later write (after the last Reset), the EME, IVIS and ESTR bits must be written all ones and the SMODN, MODB and MODA bits must be written as 011B when Single-Chip or Expanded-Wide Mode is emulated. When Expanded-Narrow Mode is emulated write them as 001D.

An exception is during a complex operating mode switch as described in the next section called "Switching the HC12 operating mode on-the-fly". If the mode register is written by an illegal value, which is different than the allowed values described above on the second or later write after the last HC12 Reset, then the emulator ERROR state will be entered. The ERROR LED will be turned on and the HC12 and the emulator will be forced into the Reset state.

The ERROR state is entered because the described second or later write to the MODE register has changed the internal configuration of the HC12 controller on the CPU-Module in a way which does not support emulation. The ERROR state is entered to prevent bus contention and other undesired operations of the emulator.

If the ERROR state becomes active (the ERROR LED and the RESET LED turn on) the user can exit this state only by clicking on the Reset icon in the software user interface, or sending a Reset from the PC in any other way. See more details about the ERROR state in section 'The ERROR LED' above.

When the MODE and the PEAR registers are read by the user code, or read to display in a data window in the software user-interface, they will always read the values of the internal MODE and PEAR registers from the HC12 controller on the CPU-Module (01111011B or 00111011B will be read for the MODE register and 2CH will be read for the PEAR register).

A way to view the recreated MODE and PEAR register values in the emulator (which are the values which are likely to be of interest to the user), is to view these registers in a Register Window. This is done by adding new SFR registers to a Register Window, and selecting the MODE register and the PEAR register, from the SFR registers list.

Switching The HC12 Operating Mode On-The-Fly

Switching the emulated HC12 Operating-Mode on-the-fly is possible. It is limited to one time only after every HC12 Reset, and is enabled only when the MODE register has not been written yet since the last HC12 Reset.

There are two types of HC12 Operating Mode switches: A simple one, and a complex one. The following table shows the information about the allowed operating mode switches and what type each one of them belongs to.

		Original Operating Mode					
		Normal Exp. Wide	Normal Exp. Narrow	Normal Single Chip	Special Exp. Wide	Special Exp. Narrow	Special Single Chip
New Operating Mode	Normal Exp. Wide	Simple	Complex	Simple	Simple	Complex	Complex
	Normal Exp. Narrow	Complex	Simple	Complex	Complex	Simple	Complex
	Normal Single Chip	Simple	Complex	Simple	Simple	Complex	Complex
	Special Exp. Wide	Illegal	Illegal	Illegal	Simple	Complex	Complex
	Special Exp. Narrow	Illegal	Illegal	Illegal	Complex	Simple	Complex
	Special Single Chip	Illegal	Illegal	Illegal	Simple	Complex	Complex

Pay attention that an Operating Mode switch from Normal Mode to Special Mode is illegal. Such an attempt will result in the ERROR state becoming active (see details in section ' The ERROR LED' on how to exit from the ERROR state).

To perform a SIMPLE operating mode switch, a single write to the MODE register should be performed. This write value should specify the new required operating mode and the other MODE register parameters to be implemented.

To perform a COMPLEX operating mode switch, two writes to the MODE register should be performed: The first write should write a value which specifies the new required operating mode, and the other MODE register parameters to be implemented. The second write value depends on the 'New Operating Mode' which is about to be implemented.

In case the 'New Operating Mode' to be implemented is Single-Chip or Expanded-Wide a value of 7BH should be written on the second write, and in case the 'New Operating Mode' to be implemented is Expanded-Narrow a value of 3BH should be written on the second write.

During a SIMPLE operating mode switch, the recreated MODE register in the emulator changes its value (to the value written to the MODE register), but the MODE register in the HC12 controller on the CPU-Module does not. In this case, the original recreated operating mode will be maintained until the end of the write to the MODE register, after which, the new recreated operating mode will take place.

During COMPLEX operating mode switch, on the other hand, both the recreated MODE register in the emulator and the MODE register in the HC12 controller on the CPU-Module change their values. In this case, the recreated MODE register in the emulator changes its value to the first write value to the MODE register, whereas the MODE register in the HC12 controller on the CPU-Module changes its value to the second write value to the MODE register. In this case, the original recreated operating mode will be maintained until the end of the second write to the MODE register, after which, the new recreated operating mode will take place.

HC12 Reset

The HC12 controller on the CPU-Module and the emulator system may be forced into Reset in several ways, using different Reset Sources, which will be described below.

After Reset is done, in most cases, a short internal emulator 'Reset Transition' program will take over the HC12 controller for approximately 1000 ECLK cycles. This Reset Transition program is required because after Reset, it takes the emulator 1000 ECLK cycles to re-enable full emulation support by the HC12 (this is a limitation of the HC12 controller itself, and not of the emulator). For more details about Reset Transition, and when it takes place, refer to section 'Reset Transition' below.

The Reset Sources may be divided into three main groups according to their functionality:

1 Group A - Reset Sources, which force the HC12 and the emulator to monitor mode where user code is stopped and the internal HC12 status may be inspected and changed by the user.
2 Group B - Reset Sources, which force the HC12 and the emulator to emulation mode where the user code is running.
3 Group C - Reset Sources, which do not change HC12 and the emulator to monitor-mode or emulation-mode.

Group A - Resets which force the HC12 to Monitor Mode

For these Resets sequences in 'Group A', the operating-mode, which will be implemented by the emulator after the Reset sequence is over, is determined by the state of the 'Operating Mode' field in the software Hardware Configuration Window.

- Reset from the Software user interface. This Reset is forced upon software startup and by clicking on the Reset icon in the software user interface. This Reset may also set some default values for some CPU and/or SFR Registers according to user configuration.
- Reset when an ERROR state becomes active. This Reset becomes active when an ERROR condition, which was initiated by an illegal write to the MODE or PEAR SFR registers, is detected by the emulator. Reset is forced in order to protect the emulator and the HC12 controller on the CPU-

Module from bus contention and other misbehaviors. See more details in sections 'The ERROR LED' and 'The PEAR and MODE HC12 SFR Registers', on how to end this ERROR state.

Group B - Resets which force the HC12 to Emulation Mode

For these Resets sequences in 'Group B', the operating-mode, which will be implemented by the emulator after the Reset sequence is over, is determined by the state of the SMODN, PE6, and PE5 pins of adapter to the target during the Reset.

- Reset from the target system or through the Reset pin of the adapter to the target. This Reset will be recognized and processed by the emulator. This is done only when emulation mode is active (user code is running), the 'Reset From Target' check box in the software Hardware Configuration Window is checked, and a low level is detected on the Reset pin of the adapter to the target. If either one of these three conditions is not true, then Reset to the emulator and the HC12 controller on the CPU-Module will not become active. For proper processing of this Reset, the user should make sure that the Reset pin of the adapter to the target is held low for a sufficient time, Motorola recommends a duration of at least 32 ECLK cycles.
- Reset and Go is initiated by the software user interface. This will force Reset to the emulator and the HC12 controller on the CPU-Module during emulation (when user code is running), from the software user interface. This Reset sequence may be used to simulate a Reset, which behaves similar to a target Reset.

Group C - Resets which maintain the active Monitor or Emulation Mode

For these Resets sequences in 'Group C', the operating-mode, which will be implemented by the emulator after the Reset sequence is over, depends on whether Monitor Mode is in progress or Emulation Mode is in progress.

When Monitor Mode is in progress (the HC12 is stopped and its internal state may be viewed and changed by the user interface), the operating-mode, which will be implemented by the emulator after the Reset sequence is over, is determined by the state of the 'Operating-Mode' field in the software Hardware Configuration Window.

When Emulation Mode is active (the emulator is running user code), the operating-mode, which will be implemented by the emulator after the Reset sequence is over, is determined by state of the SMODN, PE6 and PE5 pins of adapter to the target during the Reset.

- Reset on insufficient HC12 VDDX power supply. When the emulator detects that there is insufficient HC12 VDDX voltage, it forces Reset to the emulator and the HC12 controller on the CPU-Module. This is done in order to prevent internal misbehaviors because in this case, the HC12 will not output valid control signals. This Reset may be used during emulation mode (when user code is running) in order to emulate power cycling of the HC12.
- Reset from an internal HC12 source, such as COP watchdog Reset, Clock Monitor Reset etc.

Reset Transition

After every HC12 Reset, it takes the emulator approximately 1000 ECLK cycles to re-enable full emulation support by the HC12 controller on the CPU-Module. This limitation results from the design of the HC12 controller itself.

During this 1000 ECLK cycles period, the emulator offers a short Reset-Transition program which will take over the HC12 and the emulator for this duration, and then resume normal operation when full emulation support is re-enabled. The Reset-Transition program offers some services to the user to configure the HC12 more closely to the operating mode emulated and enable the use of some internal HC12 resources according to the user configuration.

The services offered by the Reset-Transition program are:

- When Single-Chip Mode is emulated, the Reset-Transition program will configure the external HC12 bus-interface to 0 wait-states, which emulates single-chip mode more accurately when accessing the external emulation memory on the emulator motherboard (since real single chip accesses are to the internal HC12 resources, and require 0 wait states). This is required since the default HC12 external bus-interface configuration after Reset is to 3 wait states.
- The 'Enable COP Watchdog' check box in the software Hardware Configuration Window, enables or disables activating the internal HC12 COP watchdog by the Reset-Transition program. The default state of the internal HC12 COP watchdog when Reset-Transition does not take place, is disabled.
- When Single-Chip Mode is emulated, the 'Enable Internal Flash' check box in the software Hardware Configuration Window, enables or disables activating the internal HC12 Flash memory by the Reset-Transition program. The default state of the internal HC12 Flash memory when Reset-Transition does not take place, is disabled.
- The 'Disable Internal EEPROM' check box in the software Hardware Configuration Window, enables or disables deactivating the internal HC12 EEPROM memory by the Reset-Transition program. The default state of the internal HC12 EEPROM memory when Reset-Transition does not take place, is enabled.

On Resets which come out to Monitor Mode (where the HC12 is stopped and its internal state is available to be viewed and changed by the user-interface), the Reset-Transition program always takes place.

On Resets which come out to Emulation Mode (where the emulator is running user code), the Reset-Transition may or may not take place depending on the configuration of the 'Reset Transition' check box in the software Hardware Configuration Window. When this check box is checked, Reset-Transition will take place, otherwise, it will not.

It is recommended by Nohau to always have the 'Reset-Transition' check box checked (always enable Reset-Transition).

The user may consider disabling the Reset-Transition program when it is very important that the user program start running in the emulator immediately after Reset, with no delay of 1000 ECLK cycles. In this case, where the Reset-Transition is disabled, there are some limitations which need to be considered:

- During the first 1000 ECLK cycles after Reset, no breakpoint should be met. After the initial period of 1000 ECLK cycles breakpoints can be met and used freely. If a breakpoint is met during the initial period of 1000 ECLK cycles, unspecified behavior of the emulator and the HC12 will occur.

- The external HC12 bus will be set to 3 wait states in all cases, including when single-chip mode is emulated. The user may overwrite this, by writing to the EXSTR1 and EXSTR0 bits of the MISC register.
- The internal HC12 COP watchdog will be disabled. The user may enable the internal COP watchdog by clearing the DISR bit in the COPCTL register.
- The internal HC12 Flash Memory will be disabled in all cases including Single-Chip Mode. The user may override these settings by writing to the ROMON32 and ROMON28 bits in the MISC register.
- The internal HC12 EEPROM memory will be enabled. The user may disable the internal HC12 EEPROM by writing to the EEON bit in the INITEE register.

When the Reset-Transition program is enabled, and Reset to Emulation Mode (where the emulator is running user code) takes place, the following steps will be executed in the following order:

1 After the end of Reset is detected by the emulator, the Reset Vector at the address specified by the HC12 controller, will be read and stored by the emulator. This Reset vector will be read from the emulation memory on the emulator motherboard when Single-Chip Mode is emulated, or from the emulation memory on the emulator motherboard or the target memory when Expanded Mode is emulated, depending on where the Reset vector is mapped to. The Reset vector will not be read from the internal HC12 Flash memory in any case, even when Single-Chip Mode is emulated and the 'Enable Internal Flash' check box in the software Hardware Configuration Window is checked.
2 The Reset-Transition program will start executing by the HC12 controller on the CPU-Module and the emulator. It will implement the services described above in the HC12 controller.
3 When Emulation is re-enabled again the Reset-Transition program will conclude its execution by jumping to the address specified by the Reset Vector, which was read at step 1, in order to resume execution of the user program.
4 The user program will resume execution.

Clock and Clock-Rate Aspects

The valid communication between the emulator and the DA128 or DG128 controller on the CPU-Module depends on the user inputting the frequency of the signal fed to the DA128 or DG128 EXTAL pin, in the 'Clock' field in the software Hardware Configuration Window. If this frequency differs from the value in the Clock field, no valid communication will be established between the DA128 or DG128 controller on the CPU-Module and the emulator system.

The reason for this limitation is that the EXTAL and XTAL pins of the DA128 and DG128 controllers are very sensitive to external load, and therefore cannot be sampled even by a high impedance comparator, without affecting the internal clock oscillator of the DA128 or DG128 controller on the CPU-Module. Therefore, the required frequency for communication is generated independently by the DA128 or DG128 controller on the CPU-Module and the emulator system.

It was also found that for frequencies higher than 4MHz (and especially higher than 8MHz), which are fed to the EXTAL pin of the controller on the CPU-Module, problems with the controller internal Clock Generator circuit are detected, especially if the VDDPLL pin is not connected to Ground. Therefore it is recommended whenever feeding a frequency higher than 4MHz to the EXTAL pin of the controller on the CPU-Module, to have the VDDPLL pin of the controller connected to Ground. Connecting the controller VDDPLL pin to the Emulator Ground is considered most optimal (by using the VDDPLL and VPLL jumpers – see description in sections: 'The VPLL Jumper' and 'The VDDPLL Jumper' above). This will also disable the internal HC12 PLL oscillator. This limitation is of the DA128 and DG128 controllers, and

not of the emulator. The default setting of the VPLL and VDDPLL jumpers connects the controller VDDPLL pin to the emulator Ground, as recommended here.The HC12 Emulator system is designed to assure continuous uninterrupted BDM communication between the HC12 controller on the CPU-Module and the emulator system through the BDM interface of the HC12 controller. This is done by the emulator system monitoring changes made to the HC12 system clock selections and frequency settings. This is done in order to match the emulator communication rate to any BDM communication rate changes made by the HC12 controller on the CPU-Module. This includes changes which result from the HC12 system clock changing to Slow Mode and PLL Mode, and different frequency settings within these modes.

At the time of writing this document, the latest DA128 mask set available – the 0H55W, had a bug which prevents the emulator from communicating with the DA128 controller on the CPU-Module when the internal DA128 PLL oscillator is selected to drive the internal HC12 System clock. (When the BCSP bit in the CLKSEL register is set.) This bug is documented in the Motorola Errata Sheets for the DA128 0H55W mask-set. The nature of this bug is that in PLL mode, the DA128 controller recurrently changes the state of the CLKSW bit in the BDM Status register in a way which prevents the emulator from communicating to the DA128 controller on the CPU-Module. As a result, with the 0H55W mask set, no valid DA128 emulator operation in internal HC12 PLL Mode is possible until this bug is fixed by Motorola in a later mask-set. This problem does not affect the proper emulator function in crystal mode and slow mode (when the BCSP bit in the CLKSEL register is cleared).

For the DG128, on the other hand, (for mask sets 5H55W and later mask sets) the above mentioned problem does not exist. Thus, for the DG128 the internal PLL can be used freely with no limitations at all. The emulator will detect the user frequency settings for the internal PLL as they happen during user program execution and will adjust the BDM communication rate accordingly to sustain continued uninterrupted communication between the emulator and the DG128 controller.

As a temporary work around for the above mentioned DA128 problem, Nohau suggests feeding the intended internal PLL frequency directly to the EXTAL and XTAL pins of the DA128 controller on the CPU-Module and avoiding setting the BCSP bit in the CLKSEL register. This will emulate the same intended internal HC12 system frequency, bus frequency, etc. It will not emulate the dynamics of using the internal HC12 PLL generator however. As mentioned above, this does not apply to the DG128, as it does not have the specified DA128 bug.

Feeding any selected frequency to the DA128 or DG128 controller on the CPU-Module is as easy as typing a new clock frequency in the software 'Clock' field in the Hardware Configuration Window. The typed frequency will automatically be implemented by the emulator using a PLL clock oscillator on the Emulator Motherboard to an accuracy of 0.1% or higher. Then by mounting jumper tops on pins 1 and 2 of both the PLL/EXTAL and PLL/XTAL jumpers on the CPU-Module (the default factory configuration), this emulator PLL clock generator frequency is driven to the EXTAL pin of the DA128 or DG128 controller on the CPU-Module. (See more details about the PLL/EXTAL and PLL/XTAL jumpers in section 'The PLL/ EXTAL and PLL/XTAL Jumpers' above)

Program Paging Emulation of Internal DA/DG128 Flash Memory

In the DA128 and DG128 emulator, the internal 128K Flash memory is usually replaced by external RAM emulation memory chips which are located on the emulator motherboard. This emulation memory behaves in a similar way to the behavior of the internal DA128 and DG128 Flash memory. It is accessed using the HC12 Program Paging mechanism, dividing the 128K of the emulation memory (which replace the internal Flash memory) to eight 16K pages, which can be accessed through the address window at 8000H – BFFFH and using the PPAGE register. Page 7 can also be accessed at addresses C000H – FFFFH. Page 6 can also be accessed at addresses 4000H – 7FFFH when the ROMHM bit in the MISC register is cleared. All as specified in the DA128 and DG128 manual.

This emulation memory for the Flash exists in addition to the regular emulation memory, which covers the linear 64Kbyte of HC12 address space.

In case of Expanded Wide or Narrow modes, this emulation memory which replaces the internal DA128 and DG128 Flash memory is not enabled. This is since, in these operating modes, the internal flash of the DA128 and DG128 controllers is disabled by default. This emulation memory cannot be turned on in these modes. By setting the ROMON bit in the MISC register, the actual internal Flash memory of the DA128 and DG128 controllers can be turned on.

In Single-Chip mode this emulation memory which replaces the internal DA128 and DG128 Flash memory is enabled by default. It cannot be disabled while Single-chip mode is emulated.

When after a Reset, Single-Chip mode is emulated and then the operating mode is switched to Expanded Wide or Narrow mode emulation for the Flash memory is enabled both before the mode switch and after the mode switch. After the mode switch, this emulation memory may be disabled (if it is required to disable it) by writing a 0 to the ROMON bit in the MISC register. Emulation memory replaces the internal DA128 and DG128 Flash memory

If the user would like, he or she may use the actual internal Flash of the DA128 and DG128 instead of the described external RAM emulation memory. This may be done in Single-Chip mode by checking the ‘Enable Flash’ and ‘Reset Transition’ check boxes in the Hardware Configuration window, or in any operating mode by writing a 1 to the ROMON bit in the MISC register.

However, emulator users usually prefer using the external replacement RAM based emulation memory instead of using the actual internal Flash memory. The reason is that the external RAM based memory offers more convenience for debugging purposes: It can be written easily without having to program the entire internal Flash memory every time a single byte value needs to be changed, it allows using software breakpoints and not hardware breakpoints only, etc.

The 68HC12 Trace

What Is A Trace, and how does it help debugging a micro-controller system

A Trace is an optional part of an emulator system which is used to supply advanced debugging capabilities. The advanced debugging capabilities include:

1 Recording the execution of instructions, and the bus activity of the micro-controller system under development, and displaying it to the user in an intuitive way for debugging purposes. The executed instructions are displayed disassembled, the address and data are arranged in fields, etc.
2 Being able to record only a small portion of the bus activity and/or instruction execution, which is of interest to the user (such as a range of addresses, a range of data, and a specific type of bus cycle as set by the user). This function is called 'Filtering' and is used in order to display only the information which is of interest at any given point (to allow solving a specific software bug in a specific function for example).
3 Being able to set complex conditions which are used to identify very specific events as they are caused by the micro-controller system under development. These conditions may include a sequence of several events, and/or a repeat of a certain event for a number of times, which may then cause the trace to stop recording, and for the recorded data to be displayed to the user. This then may or may not also cause the micro-controller to break emulation based on the user settings which make it behave like a sophisticated breakpoint. This function is called 'Triggering'.
4 Record Time Stamp information for every recorded frame. This allows the user to check the timing of their micro-controller system, measure how long it takes to execute certain functions, etc.
5 Record 'Code Coverage' information. This specifies which instructions were executed at least one time and which were not. This 'Code Coverage' function is useful for late debugging stages in order to make sure all the instructions get executed.

The HC12 Trace is a very powerful tool. It supplies the above mentioned capabilities to support software and hardware development for all the existing (and future) 68HC12 derivatives. This document explains how to use the HC12 trace and how to maximize its advanced capabilities in order to simplify the debugging, make it easier, and minimize the development time spent to find elusive bugs.

Features of the Nohau HC12 Trace

1 The Trace is a small 3.1" by 4" card plugged on top of any HC12 emulator board.
2 The same Trace is used to support all HC12 emulator boards and all HC12 derivatives.
3 The Trace can record all HC12 memory and SFR accesses, external and internal to the HC12 micro-controller, in all cases.
4 The Trace supplies reliable triggering and filtering of executed instructions and data accesses. Triggering and Filtering can be configured to happen only when an instruction is executed and not when it is fetched. This feature is supported using reconstruction of the internal HC12 instruction Queue and instruction decoder in the trace logic.
5 The Trace size is 128K frame (records).
6 Up to 24 miscellaneous signals can be recorded in addition to the address, data, and control signal.
7 Up to 8 additional MSB address lines (A23-A16) may be recorded for derivatives which have more than 16 address signals.
8 Up to 8 chip-Select signals may be recorded for derivatives which have chip-selects.

9 44 bits of time stamp information is recorded in resolution down to 40nSEC – allowing the time stamp counter to run without overflowing for more than a week.
10 The Trace may record and display frames in an 'Execution Oriented' manner. This means that it may record and present the executed instructions and accessed data in the order they occur, without having to record the fetched – unexecuted instructions. Each recorded and displayed instruction is followed immediately by the Data read and writes which are caused by this instruction. This makes the Trace display more useful for software debugging.
11 The Trace may record and display raw bus frames, including instruction fetches data read and write, free cycles, etc. This mode is more useful for hardware debugging.
12 The Trace may be stopped, reconfigured and started, as many times as required, without stopping the user program or interrupting its running at full speed during these starts and stops and re-configurations of the trace.
13 The Trace is optional.

How to control and display the trace content using the emulator user interface

The HC12 trace has two main windows associated with it in the emulator user interface software (Seehau):

1 The Trace Configuration window. In order to display this window, click on the 'Config' menu and select the 'Trace…' menu-item. This window is used to configure the HC12 trace as required by the user.
2 The Trace display window. In order to display this window click on the trace icon on the control bar. This window is used to display the information which is recorded in the trace.

Following in this document you will find details on how to use these two windows to utilize the trace.

What information gets recorded in the trace

The information recorded in the trace memory is arranged into frames. The HC12 trace (part number EMUL12-PC/TR128-16) can store up to 128K frames. This frame memory functions as a circular buffer which, when filled up, starts overwriting its oldest frames.

The following fields are recorded for each trace frame:

1 16 bit address information - recorded from the HC12 address bus.
2 16 bit data information - recorded from the HC12 data bus.
3 8 bits of control bus information which specify what type each frame is(instruction execution, data read, data write, how wide is the data etc.).
4 Up to 24 miscellaneous signals which are arranged into three 8 bit fields – Misc A, Misc B, and Misc C.
5 Up to 8 additional MSB address bits, used for some HC12 derivatives, which use more than 16 address signals.
6 Up to 8 Chip-Select signals used for some HC12 derivatives, which generate chip-select signals.
7 Time Stamp information of 44 bits in resolution down to 40nSEC – allowing the time stamp counter to run without overflowing for more than a week.
8 Indication for the memory source that the recorded frame originated from – internal HC12 memory, external emulation memory on the emulator board, or external memory on the target.

The Time-Stamp Information

Each recorded frame contains 44 bit time-stamp information. This 44 bit time-stamp is generated by a 44 bit counter in the trace. The clock used for the time stamp counter is a 25MHz signal generated on the pod, which may be pre-scaled by any factor in the range 1-255. The pre-scale value for the time stamp counter may be configured under the Trace Setup tab in the Trace Configuration window, in the Time Stamp Prescaler field.

The 25MHz clock used for the time stamp is always available. Its rate is not affected by behavior of the HC12 micro-controller (such as ECLK stretch, halt of the ECLK when HC12 stop power down mode becomes active, and hc12 clock rate changes). It also supplies a time resolution of 40nSEC. Thus it is ideal to be used as the time stamp clock to supply reliable timing information.

The miscellaneous signals, additional MSB address bits, and Chip-Selects signals

The 24 miscellaneous signals are arranged into three 8 bit fields – Misc A, Misc B, and Misc C. They are sampled from the 3 connectors on the trace.

One 8 bit probe-set, which may be used with any one of these 3 miscellaneous connectors to sample signals from the target or the pod, is supplied with the trace kit. Additional probe-sets may be ordered separately from Nohau (part number EZ/8 BIT PROBE SET). Two of these three miscellaneous field (Misc.-A and Misc.-B) may also be used to record up to 8 additional MSB address bits and up to 8 Chip-Select signals, which may be used for some HC12 derivatives. These additional MSB address bits and Chip-Select signals are routed to the trace directly from the emulator board through the trace connectors and do not require the user to connect them using additional wires.

Each Misc A bit may be configured to record the data on the Misc.-A connector or an MSB address signal, which is generated by the HC12 micro-controller on the pod. This selection is made in the 'Misc. A' tab in the trace configuration window.

Similarly, each Misc B bit may be configured to record the data on the Misc B connector or a Chip-Select signal, which is generated by the HC12 micro-controller on the emulator board. This selection is made in the 'Misc. B' tab in the trace configuration window.

The Misc C field does not share its functionality. It is dedicated for use as a general-purpose miscellaneous sample port which records the data on the Misc C connector.

The 24 miscellaneous signals, when configured to sample the data from the Misc A, Misc B and Misc C connectors, may each be individually configured on a bit by bit basis for the edge which will be used to sample the signals on the Misc connectors.

The selections are:

- On ECLK falling edge.
- After ECLK fall (approximately 30nSEC after ECLK falling edge).
- On ECLK rising edge.
- After ECLK rise (approximately 30nSEC after ECLK rising edge).

These selections are configured in the Misc. A, Misc. B and Misc. C tabs in the trace configuration window.

What Type of Cycles and States may be recorded

The HC12 trace can be configured to determine what type of bus cycles and states are enabled to be recorded in the trace memory and be displayed in the Trace Display window. This is configured by checking check-boxes under the 'Includes for Triggers and Filter' field in the Trace Setup tab in the Trace Configuration window. The check boxes configuring this functionality are suffixed as '(Filter)' and not as ('Trigger').

The following selections exist:

1 Instruction Execution – enables recording and displaying instructions as they are executed (this will not record opcode fetches or any instructions which are fetched but not executed).
2 CPU Writes – enable recording write cycles caused by the HC12 user program.
3 CPU External Reads - enable recording external HC12 read cycles caused by the HC12 user program.
4 CPU Internal Reads - enable recording internal HC12 read cycles (from internal HC12 resources) caused by the HC12 user program. (The distinction between external and internal reads is made because on many HC12 derivative internal read cycles and free cycles cannot be distinguished from each other)
5 BDM Reads and Writes – enable recording reads and writes which are not caused by the user program but by the HC12 BDM interface. BDM reads will happen whenever there is an open 'RUN TIME DATA' window. BDM writes will happen whenever the user is writing to the HC12 memory during emulation (when user program is running), from an open 'RUN TIME DATA' window or an open 'SHADOW' window.
6 Interrupt Vectors – enable recording interrupt vector read cycles. Identifying these cycles depends on correct cycle type information output by the HC12 micro-controller.
7 Fetch Cycles – enable recording opcode fetch cycles caused by the user program.
8 Free Cycles – enable recording HC12 free cycles. These are cycles during which internal operations are made by the HC12 controller but no actual bus activity. Although during free cycles there is no actual bus activity made by the HC12 controller, to the emulator the HC12 displays information as if there is a read cycle in progress. There is no reliable way to distinguish all free cycles from real read cycles, therefore, depending on the HC12 derivative used, some (or even all) free cycles may be classified by the trace as read cycles. For derivatives which have a DBE control signal free cycles which look as internal HC12 read cycles will be classified as read cycles while the other free cycles will be correctly identified. Those correctly identified Free cycles may be displayed under this selection.
9 Wait Power-Down State – enable recording a Wait Power Down indication frame each time Wait Power-Down mode, is initiated by the user program.
10 Stop Power-Down State – enable recording a Stop Power Down indication frame each time Stop Power-Down mode is initiated by the user program.
11 Reset State – enable recording a Reset indication frame each time HC12 Reset becomes active when user program is running (such as when internal watchdog reset occurs, or target reset occurs).
12 Reset-Transition State - enable recording a Reset-Transition indication frame each time Reset-Transition becomes active after reset (for details about Reset-Transition refer to the emulator manual).
13 Error State - enable recording an Error indication frame each time Error mode becomes active when performing illegal write to some internal SFR register (for details about ERROR mode refer to the emulator manual).

The instruction execution information and the raw bus cycle information is then subjected to further filtering using the filtering system (For details look under the description for the filtering system below).

What Fields may be displayed in the Trace Display Window

The Trace Display window may be customized according to the user requirements to display only the necessary information.

The 'Frame' Field
This field displays the frame number for each displayed frame.

Frame 0 is the frame during which the trigger which caused the trace to stop, occurred. Thus scrolling to frame 0, will always take you to the frame that triggered the trace, or to the next frame which was recorded, in case the trigger frame was filtered out. The frames, which were recorded before the trigger occurred, are numbered with negative frame numbers, and the frames, which were recorded after the trigger occurred, are numbered with positive frame numbers.

The 'Address' Field
This field displays the address which was used by the HC12 micro-controller for all the instruction execution and raw bus cycles frames.

The 'Time-Stamp' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Time-Stamp menu item.

The Time-Stamp field when displayed, may display the time-stamp information in one of the following formats:

- Absolute Time Stamp Clock Counts (referred as Absolute cycle).
- Relative Time Stamp Clock Count to the previous displayed frame (referred to as Relative cycle).
- Absolute Time Stamp in time units (referred to as Absolute time).
- Relative Time Stamp in time units to the previous displayed frame (referred to as relative time).

The display format of the time stamp field is selected by right clicking in the Trace window and manipulating the 'Relative Time-Stamp' and 'Convert Cycles to Time' menu-items. In order to perform time measurements you may also select the Absolute time display, then scroll to a specific frame, right-mouse click in the Trace window and select Zero Time at Cursor. This will assign time 0 to the specified frame, and will measure the time of the other frames relative to the specified frame. Thus, you may in this way for example, measure how long it took to execute a certain function, by assigning time 0 to the first instruction of a function and then scrolling to the RTS instruction of this function.

The 'Misc.' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Misc. Data menu item.

This field when enabled displays the 24 Miscellaneous signals sampled from the Misc A, Misc B and Misc C connectors on the trace board. All the Misc A and Misc B bits, which are used to sample additional address and chip-select signals are displayed as zeros in this field.

The 'Pod Pins' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Pod Pins menu item.

This field when enabled displays the active chip-select signal for the current frame. (This field has meaning only when some of the Misc B bits are configured to record Chip-Select signals).

The 'Opcode' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Opcode menu item.

This field when enabled displays the opcode bytes of executed instructions and the data read or written for displayed raw bus cycles (CPU reads, writes etc.).

The 'Status' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Status menu item.

The status field when enabled displays the memory source that the displayed memory access was accessing: Internal HC12 memory source, external emulation memory on the emulator board, or external memory on the target.

The 'Instr' (Instruction) Field
The instruction field displays the disassembled instructions and the memory access type (read, write, fetch, word, byte, etc.).

The 'Symbol' Field
This field may be enabled or disabled based on the user selection. In order to enable or disable it, right mouse click in the Trace window - a menu will show up – select the Show Symbol menu item.

The symbol field when enabled displays symbols, which are associated with displayed instructions and data cycles (when a symbol is available for the displayed frame).

How to Determine what type of cycles may be used for the Triggerring

The HC12 trace can be configured to determine what type of bus cycles can be used in the triggering logic. This is configured by checking check-boxes under the 'Includes for Triggers and Filter' field in the Trace Setup tab in the Trace Configuration window. The check boxes configuring this functionality are suffixed as '(Trigger)' and not as ('Filter').

The following selections exist:

1 Instruction Execution – enables using instruction execution in the triggering logic. (This enable triggering on the instruction only when it is executed and not when it is fetched, but not executed.)
2 CPU Writes – enables using CPU write cycles caused by the HC12 user program in the triggering logic.
3 CPU External Reads - enables using external HC12 read cycles caused by the HC12 user program in the triggering logic.
4 CPU Internal Reads - enables using internal HC12 read cycles caused by the HC12 user program in the triggering logic. (The distinction between external and internal reads is made because on many HC12 derivative internal read cycles and free cycles cannot be distinguished from each other.)
5 BDM Reads and Writes – enables using reads and writes which are not caused by the user program but by the HC12 BDM interface in the triggering logic. BDM reads will happen whenever there is an open 'RUN TIME DATA' window. BDM writes will happen whenever the user is writing to the HC12 memory during emulation (when user program is running), from an open 'RUN TIME DATA' window

or an open 'SHADOW' window.

6 Interrupt Vectors – enables using interrupt vector read cycles in the triggering logic. Identifying these cycles depend on correct cycle type information output by the HC12 micro-controller.

7 Fetch Cycles – enables using opcode fetch cycles caused by the user program in the triggering logic.

8 Free Cycles – enables using HC12 free cycles, which are cycles during which internal operations are made by the HC12 controller but no actual bus activity in the triggering logic. Although during free cycles there is no actual bus activity made by the HC12 controller, to the emulator, the HC12 displays information as if there is a read cycle in progress. There is no reliable way to distinguish all free cycles from real read cycles, therefore, depending on the HC12 derivative used, some (or even all) free cycles may be classified by the trace as read cycles. For derivatives which have a DBE control signal, only free cycles which look as internal HC12 read cycles will be classified as read cycles, while the other free cycles will be correctly identified. Those correctly identified free cycles may be used in the triggering logic here.

The instruction execution information and the described raw bus cycles, when enabled by these 8 global triggering enable check-boxes, may be used in the triggering logic, to cause the trace to stop and show the recorded frames, and to break emulation (stop the user program) as specified by the user. (For details look under the description for the triggering system below.)

Triggering and Filtering System Overview

Trigger Mode

There are two basic Trigger modes: Opcode and Data.

The active trigger mode is selected under 'Trigger Mode' in the Trace Setup tab in the Trace Configuration window, by clicking on one of the 'Opcode' or 'Data' radio buttons.

In Opcode mode, the programmed triggers and filter ranges under the 'Trigger1', 'Trigger2', 'Trigger3', and 'Filter' tabs, enable the user to distinguish between:

- Opcodes (Execution and Fetch)
- Data Read and Write
- Both Opcodes (Execution and Fetch) and Data Read and Write
- None

In Data mode, the programmed triggers and filter ranges under the 'Trigger1', 'Trigger2', 'Trigger3', and 'Filter' tabs, enable the user to distinguish between:

- Data Read
- Data Write
- Both Data Read and Data Write
- None

Pay attention that in Data mode it is not possible to trigger-on or filter instruction execution.

The enabled cycles by the 'Includes for Triggers and Filter' check box list (in the Trace Setup tab in the Trace Configuration window), are classified for the triggering and filtering purposes (in order to trigger and filter correctly in both Opcode mode and Data Mode) in the following three categories:

1 Opcode – include Instruction Execution information and Opcode Fetch cycles.
2 Data Read – include CPU External Read cycles, CPU Internal Read cycles, BDM Read cycles, Interrupt Vector Cycles, and Free Cycles.
3 Data Write – include CPU Write cycles and BDM Write cycles.

Thus, for example, selecting Data Mode and configuring a trigger or filter range for Data Read, will enable all the enabled cycles of group B to participate in the particular trigger or filter logic.

Filter Mode

There are two basic Filter modes: Normal and Window.

The active Filter mode is selected under 'Filter Mode' in the Trace Setup tab in the Trace Configuration window, by clicking on one of the 'Normal' or 'Window' radio buttons.

In Normal Filter mode – Trigger 1, Trigger 2 and Trigger 3 function as three conditions which should be recognized sequentially (Trigger 1, then Trigger 2, then Trigger 3) in order for a trigger to be recognized by the trace. To enable all of these 3 sequential conditions all 3 check-boxes under the 'Active Triggers' field (in the Trace Setup tab in the Trace Configuration window) should be checked. To enable only one condition check only the 'Trigger 1' check-box, and leave the 'Trigger 2' and 'Trigger 3' check boxes unchecked. In this mode the last enabled trigger may also be assigned a repeat counter which will cause the trace to look for this last trigger a number of times before a trigger is recognized. The value for this repeat counter is configured in the 'Last Trig Repeat Count' field, and may be assigned any value between 1 and 65536.

In Window Filter mode – Trigger 1 is used to start filtering, Trigger 2 is used to stop filtering, and Trigger 3 is used as a trigger. In this mode, each time Trigger 1 is met, the trace enables recording new frames as filtered by the filtering system, and each time Trigger 2 is met, the trace disables recording new frames. These occurrences of Trigger 1 and Trigger 2 may enable and disable trace recording unlimited number of times. In this mode, Trigger 3 functions as a regular trigger, which may cause the trace to stop recording and display the recorded frames. In this mode Trigger 3 may also be assigned a repeat counter which will cause the trace to look for this Trigger 3 a number of times before a trigger is recognized. The value for this repeat counter is configured in the 'Last Trig Repeat Count' field, and may be assigned any value between 1 and 65536.

After a trigger is recognized by the trace (in both Normal Filter mode and Window Filter Mode), the user can configure the trace to continue to record any number of additional frames between 0 and 2,097,151 (1FFFFFH). This number is configured in the 'Post Trigger Count' field.

Filtering

Filtering is required in order to select what type of information, in what address range, and with what data is recorded in the trace memory. This function is achieved by configuring the filter under the Filter tab in the Trace Configuration window. When the 'Filter' check-box (under the Trace Setup tab in the Trace Configuration window) is checked, filtering is enabled according to the programmed ranges in the Filter tab. When this check-box is not checked, all the enabled Filter cycles under 'Includes for Triggers and Filter' (regardless of what the address is and what the data is), will be recorded in the trace.

Setting Triggers and Filter Ranges

Setting Triggers and Filter is done in the 'Trigger 1', 'Trigger 2', 'Trigger 3', and 'Filter' tabs in the Trace Configuration window.

What cycles may be used for the triggers is determined under the 'Includes for Triggers and Filter' field, by the check-boxes labeled as '(Trigger)'. (For more details about these check boxes refer to Section 7).

What cycles may be used for the filter is determined under the 'Includes for Triggers and Filter' field, by the check-boxes labeled as '(Filter)'. (For more details about these check boxes refer to Section 5).

The triggers and filter are configured as address ranges and data ranges, which when identified, will cause a trigger or a filter to be recognized. In Opcode Trigger mode the ranges may also be associated with either: A) instruction opcode execution and fetch, B) data reads and writes, or C) both A and B. In Data Trigger mode the ranges may be associated with either: A) data reads, B) data writes, or C) both data reads and writes. For more details about how the different cycles are classified into these categories refer to section 8 under Trigger Mode.

The address ranges programmed can be in the range of 00000H – FFFFFH (1MByte space) which covers programs of size up to 1MByte. For applications where only 64Kbyte of HC12 memory space is used, simply specify a 16 bit address range.

The data ranges can be in the range of 0000H to FFFFH which covers the 16 bit data bus used by the HC12. For 8 bit data accesses the data mask should be set to 00FF. For 16 bit data accesses the data mask will normally be set to FFFF.

The three triggers and the filter have enable check boxes which are displayed both in the Trace Setup tab under 'Active Triggers' and in the 'Trigger 1', 'Trigger 2', 'Trigger 3', and 'Filter' tabs respectively. These check boxes control the same functionality in both places, and so they are tied together. For example setting the 'Trigger 1' check-box in the Trace Setup tab will automatically check the 'Enabled' check box under the Trigger 1 tab and vice-versa.

Extend Recording

The Extend Recording functionality allows extending trace recording a number of additional instructions, each time the filtering system is recording a frame to the trace memory. The Extend Recording functionality is enabled by checking the 'Extend Recording?' check-box, under the Trace Setup tab, in the Trace Configuration window. When extend recording is enabled, the user may specify how many additional instructions will be recorded (0 to 255). This is specified in the Extended Count field under the Trace Setup tab in the Trace Configuration window. For example, the value 0 will record all the remaining frames in the current instruction, and when the next instruction will begin, will stop recording until the next filtered frame. The value 2 for example will record all the remaining frames in the current instruction, plus all the frames which result from the next two instructions which will be executed, and then stop recording until the next filtered frame.

Break Emulation From the Trace

You can break emulation (stop the user code from running) using the trace, on one of the following events:

1 On Trace Trigger
2 On Trace Stop
3 On Trace Full

This is configured by checking one or more of the check-boxes under the 'Break Emulation?' field in the Trace Setup tab in the Trace Configuration window.

Break Emulation on Trace Trigger
This option will cause user code to stop execution as soon as a trigger is recognized by the trace (after the sequence of Trigger 1, then Trigger 2, then Trigger 3, etc. as configured by the user).

Break Emulation on Trace Stop
This option will cause user code to stop execution after a trigger is recognized by the trace, and all the additional frames are recorded as specified by the Post Trigger Count field.

Break on Trace Full
This option will cause the user code to stop execution as soon as the trace memory is filled with frames.

All emulation-break options by the trace skid. This means that due to pipelining of the trace logic, it takes the trace a few cycles to identify the cause to the required emulation break, and therefore, the emulator will execute a couple of extra instructions before it will finally break emulation. The exact number of additional instructions which will execute in these cases vary from instruction to instruction, and depends on the CPU clock.

Trigger-Input and Trigger-Output

The Trigger-Input and Trigger-Output are input and output signals to and from the trace, which may be used to connect between the trace and other devices such as logic analyzers, scopes, etc. for triggering purposes. These two signals are available both on coax connectors and on test points on the trace board.

Trigger-Output

The Trigger-Output signal is an active-low output from the trace, which may be used to trigger an external device such as a logic analyzer or a scope when the trace trigger occurs. This signal is available both on the TRIG-OUT coax connector, and on TP3.

Trigger-Input

The Trigger-Input signal is an active-low input to the trace, which may be used to carry an external trigger generated by an external device such as the target, a logic analyzer or a scope, in order to cause the trace to trigger. The Trigger-Input signal has a 10K pull-up resistor to +5V, which defaults this input to be inactive when nothing is driven on it.

This input may function in one of the two following modes which may be selected in the Trace Setup tab in the Trace Configuration window using the Trigger Input radio buttons:

- When configured as Trigger – this input, when it becomes low, will cause the Trace to trigger and stop recording and will display the recorded frames to the user. If it is not zero then the trace continues to collect number of frames as specified by the post trigger count field in the Trace Setup tab in the Trace Configuration window. This may also lead to emulation break, in case Emulation Break on trigger or on trace stop is enabled by checking the 'Yes, on Trigger' or 'Yes, on Trace Stop' check boxes in the Trace Setup tab in the Trace Configuration window.
- When configured as Inhibit – this input when becomes low will inhibit the trace trigger logic from advancing to look for the next trigger or from counting an occurrence of a trigger as specified by the repeat count field in the Trace Setup tab in the Trace Configuration window. This means for example that if a trigger is set on address X, and address X is met, the trace will trigger only if the Trigger-Input will be high. If the Trigger-Input is low – no trigger will occur.

Examples

The following examples illustrate how to set up the HC12 trace, for various scenarios we have found to be useful, and how to utilize its full power for your debug requirements.

All the following examples use the Nohau 'hc12time' example, which is available with the Seehau software in directory examples. To load this example, click on the File menu, choose Load Code and select hc12time.695. This example uses the memory range between 0800h - 0BFFh for both instructions and data. This program is generating a running clock which may be viewed during run-time. To view this clock, open a data window, right mouse click in the data window and choose Address Space Shadow. Right mouse-click again and choose Display As, and Type Ascii. Then switch the address of the window to 0B80 by typing at the bottom of the data window 0B80, in order to view the clock. Click on the Go icon on the control bar. You should now see the clock updating at address 0B80h.

Example A
How to record information in the trace and display it?

After installing Seehau and running Seehau Config, The Seehau software should be automatically configured to record executed instructions and CPU reads and writes in the trace. To record in the trace start Seehau, load the hc12time example, open a trace window by clicking on the 'TR' icon in the control bar, and click on the 'Source Step Into' icon in the control bar. The program executed a source step to the first source line after the initialization code, and the initialization code which was executed, should be displayed in the trace window.

Pay attention that the information is displayed in the order it is executed – Every instruction is followed immediately by the data reads and writes which resulted by its execution. Also, all the displayed instructions were actually executed – unexecuted fetches are not displayed in the trace window.

If you don't see anything displayed in the trace window, please make sure you executed all the steps specified here correctly. If you still don't see anything displayed in the trace window, please call Nohau technical support at this point.

Example B
How to record and display all the raw bus cycles in the trace display.

To record all the raw bus cycles in the trace bring up the Trace Configuration window (by clicking on the Config menu and choosing Trace…). Under the 'Includes for Triggers and Filter' field scroll down. Uncheck 'Instruction Execution (Filter)', and check the next 7 check boxes which are: 'CPU Write (Filter)', 'CPU External Read (Filter)', 'CPU Internal Read (Filter)', 'BDM Reads and Writes (Filter)', 'Interrupt Vectors (Filter)', 'Fetch Cycles (Filter)', and 'Free Cycles (Filter)'.

This will enable recording all of these raw bus cycles in the trace. Click on the OK button.

Run the emulator by clicking on the GO icon, then stop the trace by clicking on the Trace icon. The recorded raw bus cycles will display in the trace window. Among the raw bus cycles you will find Fetch cycles, Free Cycles, and Data read and Write cycles.

Enable the time stamp field (right click in the trace window and choose Show Time Stamp). Configure to show the time stamp as relative time (right click in the trace window and check Relative Time Stamp and Convert Cycles to Time). You will see in the trace display that every raw bus cycle is between 120nSEC and 160nSEC after the previous bus cycle, which reflects the ECLK rate of 8MHz (ECLK period is 125nSEC).

Also pay attention that the operation of the trace was stopped, and recorded data was read from it and displayed without any interruption to the program, which is still running at full speed. Also, the trace may be reconfigured with new configuration, started and stopped, as many times as required without any interruption to the program which is running at full speed by the emulator during this time.

Example C
How to Trigger and stop the trace on Instruction Execution.

We will set a trace trigger in this example on the execution of the instruction at address 082F (LDY 2,SP). To set up this example do the following:

Bring up the Trace Configuration window by clicking on the Config menu and choosing the Trace menu-item. Under the Includes for Triggers and Filter check-boxes list, check the 3 check boxes: 'Instruction Execution (Trigger)', 'Instruction Execution (Filter)', and 'Fetch Cycles (Filter)'. Make sure that all the other check-boxes on this list are unchecked. This is in order to be able to trigger on instruction execution, and to record instruction execution, and opcode fetches only.

Set the Post Trigger Count field to 10, to enable recording 10 extra frames after the trigger is recognized. Switch to the Trigger 1 tab by clicking on it. Right mouse click on the upper white field in this tab. Remove all the existing records in this field (if there are any) and then select Add. A new 'Edit Trigger Qualifier' dialog box will appear. Click on the Opcode Fetch radio button (this will select to trigger on instruction execution). Click in the Begin field and write 82f. Click on the End field, and 82f will automatically display. Click on the OK button – the new opcode fetch range 82f-82f will display in the upper white field. The 'Enabled' check-box in the bottom of the Trigger 1 tab will automatically be checked to enable Trigger 1. This check box also appears in the Trace Setup tab under the 'Active Triggers' field. This check box can always be unchecked to disable Trigger 1. Leave this check box checked for this example, and click on the OK button. If the emulator is not already running, run it right now by clicking on the GO icon.

The trace will stop recording after a second or two, and will display the recorded information, which includes in this case instruction execution and opcode fetches, as configured. On the left side of the Trace display window you will find Frame numbers. Frame 0 is always the frame during which the trigger occurred (or the next recorded frame after the trigger occurred in case the trigger frame is not recorded) – in this case this is the LDY 2,SP instruction at address 82f which Trigger 1 was set on. In order to prove to yourself that the trace did not trigger when this instruction was fetched, but not executed, scroll backward in the trace window. Pay attention that the previous instruction before the LDY 2,SP, the BNE $081C which is a conditional branch, was executed many times before the trace triggered. All of these times the conditional-branch was taken, and therefore the LDY 2,SP instruction at address 82f was not executed. If you look carefully, you will also see that during all of these sequences the opcodes at addresses 82E and 830 were fetched many times, but the trace did not trigger on them since they were not executed.

Example D
How to record only an address range of executed instructions.

To record an address range of executed instructions, the Filtering mechanism needs to be used. To set this up, bring up the Trace Configuration window by clicking on the Config menu and choosing the Trace menu-item. Disable any activated triggers (if there are any) by unchecking the Trigger 1, Trigger 2 and Trigger 3 check-boxes under the Active Triggers field.

Under the 'Includes for Triggers and Filter' field check the 'Instruction Execution (Filter)', and 'CPU Write (Filter)' check-boxes. Uncheck the remaining Filtering check boxes. This is in order, to generally enable recording only instruction execution and CPU writes in the trace.

We will set up the trace to record only the 3 instructions at address range 82F-836. To do this, select the Filter tab. Remove any existing qualifiers which may already exist in this tab, by selecting them, then right clicking, and choosing remove. Then, right-click in the upper white field, and choose Add. A new dialog box will appear. Click on the Opcode Fetch radio button in order to select to filter instruction execution information. Click on the Begin field and write 82F, then click on the End field and change it to 836. Click on the OK button – the new instruction execution address range will show up in the upper white field. The 'Enabled' check-box in the bottom of the Filter tab will automatically be checked to enable the Filter. This check box also appears in the Trace Setup tab under the 'Active Triggers' field. This check box can always be unchecked to disable the Filter. Leave this check box checked for this example, and click on the OK button. Run the emulator by clicking on the GO icon and then Stop the trace by clicking on the Trace icon.

The Trace displays the recorded frames, which include only the 3 instructions at addresses 82F-836. If the Time-Stamp is not displayed in the trace window already, display it now by right-clicking in the trace window and choosing Show Time Stamp. Also change the time stamp to be displayed in relative time units by again right clicking in the trace window and choosing 'Relative Time Stamp' and 'Convert Cycles to Time' to check these two menu-items. You will now see that between every execution of the MOVB D,X,D,Y instruction and the LDX 2,SP instruction, a long time period has passed which reflects the fact that some unrecorded instructions were executed between these two instructions.

Example E
How to add to the filtered instructions in Example D also a filter of Data write (or read) to a certain address space.

To set this example up first set up the Trace as explained in example D. Pay attention that in example D under the Includes for Triggers and Filter field, CPU write cycles were enabled, which will allow us now to record these cycles in the trace.

Bring up the Trace Configuration window, and select the Filter tab. In the filter tab, right mouse-click on the upper white field and choose Add. A dialog box will display. Click on the Data R/W radio button – this will select an address range of data read and write cycles. In this specific case, since in the Includes for Triggers and Filter list (in the Trace Setup tab) CPU external reads and CPU internal reads are not enabled, only data writes cycles will be recorded by this filter. Click in the Begin field and write B80, then click in the End field and change to B8F. Click on the OK button. The new Data R/W range will be displayed in the upper white field (in addition to the Opcode-Fetch range, which was already present from example D). Click on the OK button. Run the Emulator by clicking on the GO icon (or start the trace by clicking on the Trace icon, if the emulator is already running), and then stop the trace by clicking on the Trace icon.

In the trace window, both data write cycles from addresses B80-B8F and instruction execution from addresses 82F-836 will be displayed, as programmed by the filter.

Example F
How to Filter to record a specific instruction, and the data cycles which are caused by it, without knowing what addresses these data cycles are accessing. In this example we will set a filter which will record the MOVB D,X,D,Y instruction at address 820, and all the data cycles which result from its execution.

To set up this example, first bring up the Trace Configuration window and clear all the programmed qualifiers for the Filter. Select the Trace Setup tab and enable recording instruction execution and data read and write cycles by checking under the 'Includes for Triggers and Filter' field, the 'Instruction Execution (Filter)', 'CPU Write (Filter)', 'CPU External Read (Filter)' and 'CPU Internal read (Filter)' check-boxes. Uncheck the remaining Filtering check boxes.

Select the Filter tab, and clear all the existing qualifiers (if any). Right mouse click on the upper white field in the Filter tab and select Add. A new dialog box will appear. Click on the Opcode Fetch radio button in order to select to filter instruction execution information. Click on the Begin field and write 820, then click on the End filed and change it to 823. This will select to record the required MOVB D,X,D,Y instruction which occupy addresses 820-823. Click on the OK button – the new instruction execution address range will show up in the upper white field. The 'Enabled' check-box in the bottom of the Filter tab will automatically be checked to enable the Filter. This check box also appears in the Trace Setup tab under the 'Active Triggers' field. This check box can always be unchecked to disable the Filter. Leave this check box checked for this example. In order to record the data cycles which are caused by the MOVB D,X,D,Y instruction, we will use the Extend-Recording function. Select the Trace Setup tab, check the 'Extend Recording?' check box, and write 0 to the 'Extended Count' field. This will configure the trace every time when a filtered frame is recorded, to extend the recording for the duration of the current instruction, and stop extending the recording when the beginning of the next instruction execution is detected. In the duration of the extend recording, all the cycle types which are enabled by the filter check boxes under the 'Includes for Triggers and Filter' list, will be recorded. In this example this will enable recording instruction execution and data read and write cycles.

Click on the OK button. Run the Emulator by clicking on the GO icon (or start the trace if the emulator is already running), and then stop the trace by clicking on the Trace icon.

In the trace window, the MOVB D,X,D,Y instructions from address 820 will be displayed, where each instruction is followed by a CPU byte read and a CPU byte write cycles, which result from this MOVB instruction.

Example G
How to Filter Data Write (or Read) cycles at specific addresses, and record information to allow figuring which instruction and which function caused each of the filtered data write cycles.

In this example we will set a filter which will record the data write cycles to address B80, and the next 2 instructions which are followed immediately after each of the these data write cycles of address B80.

To set up this example, first bring up the Trace Configuration window, clear all the programmed qualifiers for the Filter, and disable all the triggers. Select the Trace Setup tab and enable recording instruction execution and data write cycles by checking under the 'Includes for Triggers and Filter' field the 'Instruction Execution (Filter)' and 'CPU Write (Filter)' check-boxes. Uncheck the remaining Filtering check boxes.

Select the Filter tab, right mouse click on the upper white field in the Filter tab and select Add. A new dialog box will appear. Click on the Data R/W radio button in order to select to filter data read and write cycles. In this specific case, this will filter only data write cycles, since the 'CPU External Read (Filter)' and 'CPU Internal Read (Filter)' check boxes under the 'Includes for Triggers and Filter' list are disabled. Click on the Begin field and write B80, then click on the End field, and B80 will automatically appear. This will select to record the required Data write cycles to address B80. Click on the OK button – the new Data R/W address range will show up in the upper white field. The 'Enabled' check-box in the bottom of the Filter tab will automatically be checked to enable the Filter. This check box also appears in the Trace Setup tab under the 'Active Triggers' field. This check box can always be unchecked to disable the Filter. Leave this check box checked for this example. In order to record the next 2 instructions, which follow each occurrence of Data Write to address B80, we will use the Extend-Recording function. Select the Trace Setup tab, check the 'Extend Recording?' check box, and write 2 to the 'Extended Count' field. This will configure the trace every time when a filtered frame is recorded, to extend the recording for the duration of the current instruction and the next two instructions, and stop extending the recording when the beginning of the third instruction execution is detected. In the duration of the extend recording, all the cycle types which are enabled by the filter check boxes under the 'Includes for Triggers and Filter' list, will be recorded. In this example, this will enable recording instruction execution and data write cycles.

Click on the OK button. Run the Emulator by clicking on the GO icon (or start the trace if the emulator is already running), and then stop the trace by clicking on the Trace icon.

The recorded information will be displayed in the trace window. Analyzing it will show that address B80 is roughly being written to every 50uSEC (by looking at the time stamp information – enabled by right clicking in the trace window and checking Show Time Stamp), and is followed by two instructions at addresses 877 and 879. Break emulation, click on the assembly tab in the source window, and scroll to address 877 (or press CNTRL and A and then type 877). By looking at this window you will be able to conclude that the data writes to address B80, are being caused by the STAB 1,Y+ instruction in function mysprintf.

Example H

How to set a complex trigger which includes detection of two trigger events sequentially, repeat the detection of the last trigger event, then record extra number of frames, and finally break emulation.

In this example we will set a trigger which will wait for address B87 to be written with the data 39 (which is the ascii code of '9' – and will be satisfied when the clock seconds become equal to x9). It will then look for the execution of the BNE $081C instruction at address 82D to occur three times, which will then cause the trace to trigger. After this trace trigger, the trace will continue to record an extra 1000 frames, then it will stop and break HC12 emulation (stop the HC12 from running code).

To set up this example, first bring up the Trace Configuration window and clear all the programmed qualifiers for the Filter and Triggers, and disable all the triggers. Select the Trace Setup tab. Enable Triggering on instruction execution and data write by checking the 'Instruction Execution (Trigger)' and 'CPU Write (Trigger)' check-boxes under the 'Includes for Triggers and Filter' field. Uncheck all of the other trigger check boxes in this list. Also in this list, enable recording instruction execution and data read and write cycles by checking the 'Instruction Execution (Filter)', 'CPU Write (Filter)', 'CPU External Read (Filter)' and 'CPU Internal Read (Filter)' check-boxes. Uncheck the remaining Filtering check boxes.

Select the Trigger 1 tab by clicking on it. Right mouse click on the upper white field and select Add. A new dialog box will appear. Click on the Data R/W radio button in order to select to trigger on data read and write cycles. In this specific case, this will trigger only on data write cycles, since the 'CPU External Read (Trigger)' and 'CPU Internal Read (Trigger)' check boxes under the 'Includes for Triggers and Filter' list are disabled. Click on the Begin field and write B87, then click on the End filed, and B87 will automatically appear. This will select to first look for the required Data write cycles to address B87. Click on the OK button – the new Data R/W address range will show up in the upper white field. Right mouse click on the lower white field and select Add. A new dialog box will appear. Click on the Begin field and write 39, then click on the End field, and 39 will automatically appear. This will select to trigger on data 39 at address B87. Click on the OK button – the new Data range of 39-39 will show up in the lower white field. Now click on the Data Mask field on the bottom of the Trigger 1 tab, and write 00FF to it. This is in order to specify that the data, which is being looked for, is 8 bit wide and not 16 bit wide. Pay attention that the 'Enabled' check-box in the bottom of the Trigger 1 tab will automatically be checked to enable Trigger 1.

Select the Trigger 2 tab. Right mouse click on the upper white field and select Add. A new dialog box will appear. Click on the Opcode Fetch radio button in order to select to trigger on instruction execution in this trigger. Click on the Begin field and write 82D, then click on the End field, and 82D will automatically appear. This will allow the triggering of the BNE $081C instruction at address 82D. Click on the OK button – the new Instruction Execution address range will show up in the upper white field. Pay attention that the 'Enabled' check-box in the bottom of the Trigger 2 tab will automatically be checked to enable Trigger 2.

Select the Trace Setup tab. Make sure the Trigger 1 and Trigger 2 check boxes are checked, and the Trigger 3 and Filter check boxes are unchecked. Make sure the Extend Recording check box is unchecked.

In the Last Trig Repeat Count field write down 3, in order to cause the trace to look for trigger 2 (which is the last programmed trigger in this example) 3 times before the trace triggers.

In the Post Trigger Count field write 1000, in order to cause the trace to record an extra 1000 frames after it will trigger and before it will stop recording.

Under the ‘Break Emulation?’ field check the ‘Yes, on Trace Stop’ check box, in order to instruct the trace to break emulation (stop the program from running), after trigger 1 was met, then trigger 2 was met 3 times, 1000 extra frames were recorded, and the trace stopped recording.

Click on the OK button in order to close the Trace Configuration window. Click on the Reset icon, and then click on the GO icon in order to run the program.

If you have a Shadow Data window open and displaying address B87, you will be able to see that a short while after this address displays the ascii character ‘9’ (becomes equal to 39) the trace will stop, and so will the program. This whole process may take about 10 second from the time you run the emulator, until the trace recognize the triggers and break emulation.

Look at the bottom of the trace window. You will see that 131072 frames were recorded, and 4 triggers occurred (Trigger 1 once, and Trigger 2 three times).

Scroll to frame 0. You will see that this frame recorded the last occurrence of the BNE $081C instruction at address 82D, which caused the last recognition of Trigger 2, and caused the trace to trigger. If you scroll more back in the trace window you will also find the two previous occurrences of the BNE $081C instruction at address 82D, which caused the first two trigger 2 occurrences (at frames –21 and –42). Also, the data write of 39 to address B87 which caused the occurrence of Trigger 1, can be found as well at frame –79.

If you scroll in the trace window forward to the end of the trace window, you will find out that an extra 1000 frames were recorded after the trace triggered, as configured. If you will check in the assembly tab in the source window, you will also find out that the emulator emulation stopped a few instructions after the last recorded instruction in the trace. The reason it took the trace a few instructions to break emulation, is because of its internal pipeline structure, which causes it to detect events only a few cycles after they actually happen, and then a delay of a few more cycles takes place until the emulator breaks emulation. This is normal when breaking emulation from the trace, and will always take place when the trace is configured to break emulation. This obviously does not happen when regular software on hardware breakpoints are used. These breakpoints never execute the instruction they are set on.