
# EMUL12-PC B32, BC32

## Hardware Manual

*Version 1.0*

# Contents

# POD-12B32/128-16 and POD-12BC32/128-16

**General**

POD-12B32/128-16 (which will be referred to as POD-12B32 from this point on) supports full emulation of Motorola's 68HC912B32 micro-controller.

POD-12BC32/128-16 (which will be referred to as POD-12BC32 from this point on) supports full emulation of Motorola's 68HC912BC32 micro-controller.

These two PODS are very similar to one another, and can be upgraded to support both chips by simply adding another controller module, which is available from Nohau.
In this way if you have POD-12B32, you can buy an additional BC32 controller module to replace the existing B32 controller module, and support the 68HC912BC32 derivative as well, and vice versa.

The basic system is available in two different configurations:
1. PC plug-in ISA board, 25 pin cable, the POD and a 5V power supply.
2. EPC device, which connects the POD to a standard PC parallel port, the POD and a 5V power supply. This configuration is also suitable for laptop computers.

A trace is optional and can be added to both these basic configurations. The trace in this case is a small card, which is connected on top of the POD.

POD-12B32 and POD12BC32 are designed to plug into one of the following 80 pin QFP solder-down adapters:
1. ES/180-5550-10 – Emulation-Solutions solder down adapter, POD to TARGET, using a standard Tokyo-Eletech solder-down base.
2. ES/180-5550-00 – Emulation-Solutions solder down adapter, POD to TARGET, using a Tokyo Eletech solder-down base, which has a socket to place an optional B32 or BC32 chip, in order to compare a stand-alone configuration to an emulator configuration.
3. ET/EPP-080-QF14A-LG – Emulation-Technology solder-down adapter, POD to TARGET, using a standard solder-down base.

Replacement solder-down base units are also available for the two Emulation-Solutions adapters:
1. ES/000-4534 – standard Tokyo-Eletech solder down base replacement for the ES/180-5550-10 adapter.
2. ES/000-2177/78/79-00 – Tokyo-Eletech solder-down base replacement for the ES/180-5550-00, which has a socket to place an optional B32 or BC32 chip, in order to compare a stand-alone configuration to an emulator configuration.

All the above mentioned adapters can be bought through Nohau, or directly from the vendor (Emulation-Solutions or Emulation-Technology).

**Supported Features of POD-12B32 and POD-12BC32**

- All of the HC12 Operating Modes are supported. This 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 bellow in section 'Switching the HC12 Operating Mode On-The-Fly').

- The sub-system which interface the POD to the target, can be powered by the full voltage range of 2.7V to 5.25V. On the POD there are 5V and 3.3V voltage sources which can be selected, to power this sub-system. This sub-system can be also configured to be powered by the target.

- Ports A, B & E are recreated by a PRU (Port Replacement Unit) on the POD, since these ports in the HC12 chip on the POD 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 functionality of the recreated ports is also supported. The reduced drive functionality 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 functionality, and in expanded modes these ports will recreate the HC12 address, data and control signals.

- The user software can be ran out of either the emulation memory on the POD, memory on the target (in expanded modes), and the internal HC12 RAM, EEPROM, and FLASH memory (utilities to program the internal HC12 FLASH & EEPROM are also supplied – see details below).

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

- All 80 pins going from the POD to the Target are available for probing by the user, using 80 header pins located on the top side of the POD.

- Emulation memory of 128K bytes (64K * 16bit) on the POD, to be used to emulate internal memory in Single-Chip Mode, or to replace target memory in Expanded Wide & Narrow Modes. This memory can be used to load the user program for emulation, and to store user data.

- Shadow Memory of 128K bytes, records in real time all HC12 write accesses external & 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 display the HC12 memory content as it is changed by the user program.

- Run-Time-Data window allows shadow-like functionality. This window read 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 good for example to read internal registers like a timer 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: 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 when Hardware Breakpoints are being used, when the HC12 is being Reset, when the HC12 is in Stop or Wait power-down modes, and whenever the BDM communication is not available by the HC12 controller on the POD (unlike the shadow which is always available).

- User writes from the software user interface to the HC12 memory (internal & external) are possible during emulation (when the emulator is running user code), through the BDM interface. This is done by, simply 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 when the BDM is available for communication).

- The internal HC12 Flash and EEPROM can be Erased, Blank-Checked, Programmed & Verified by the emulator software. Utilities within the PC software are supplied for this purpose. To erase and program the EEPROM no external power supply is required. To erase and program the Flash, an external 11.4V voltage is required. This voltage can be supplied either from special pins on the POD, or from the target. Programming the internal HC12 Flash and EEPROM may be required, in later steps of the debug cycle, to verify that the program is executing properly from the actual internal HC12 memory, as it should in a real target. Since the B32 & BC32 controllers are rated by Motorola, at 100 cycles of Flash erase and program, the controller module is socketted, so it can be replaced if needed. Nohau offers Replacement controller modules for a low price.

- Unlimited number of no-skid Hardware & Software breakpoints are available for use. Hardware breakpoints are especially useful when running the user program from the internal HC12 Flash or EEPROM memory, or in Expanded Modes where the user program is running from a non-volatile memory on the target. Software break-points can be used in most cases, when the user code is running out of a RAM based memory such as the POD emulation memory, or (in expanded modes) target RAM memory.

- In Expanded Modes, mapping the external HC12 memory to the target or the emulation memory on the POD, is possible. The mapping granularity is 1 byte (each external memory byte can be individually mapped to either the target or the emulator).

- 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, Reset from the POD Reset push-button, and Reset on insufficient HC12 VDD and 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 ran by the emulator, until the BDM becomes available for communication. During this Reset-Transition phase, the emulator also supplies the ability to configure the HC12 controller, in order to bring it to emulate the selected operating mode more accurately as defined by the user. The Reset-Transition phase alwayes 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 be always inspected and edited, by the software user interface, using SFR registers in the Register window.

- A low cost BDM emulator is offered as well.

# Pod Connectrors Description

## The 25 Pin D-Type (J180) Connector

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

## The Power-Jack (J181) Connector

The Power-Jack Connector located at the upper right side of the Pod is used to supply a 5V operating voltage to the emulator system (the Pod, the Trace & 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 & J161)

The two 50 pin Trace Connectors are designed to connect between the Pod 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 80 Target Signals Header pins ADP1

The 80 pin ADP1 headers carry all 80 B32 or BC32 signals going to the target, to emulate the B32 or BC32 controller. These double sided header pins are designed to connect between the Pod and the Target system (either directly or through one of the adapters mentioned above at the begging of this document), connecting on the bottom side of the Pod. These 80 header pins also allow all 80 signals going to the target, to be monitored by the user by connecting their top side to either the Trace Miscellaneous Connectors, to an external Logic-Analyzer or to an external Oscilloscope. All 80 of these header pins are labeled according to the B32 pin names. In case of the BC32 all pins are named the same as the B32 pins, except for the seven PCAN6-PCAN0 signals which are available on the PDL6-PDL0 header pins, accordingly.

## The GND (Ground) Wire

The GND Wire is a 6" black wire coming out of the Pod 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 Pod and the Target, in addition to the Ground connection, which exists through the adapter to the Target.

# Select Jumpers Description

## The PWR Jumper

The PWR jumper enables or disables routing power supply from the power jack on the pod to the 25 pin D type connector (also on the pod). When a jumper top is in place, power is routed to the D type connector. Otherwise it is not.

The way in which the PWR jumper should be configured depends on the system configuration which is used, and is explained in the following paragraphs:

**For Parallel Port Configuration**

If your emulator system include an EPC device, which communicate to the PC through the parallel port, then a jumper top should be in place on the PWR jumper. This is in order supply power to the EPC device from the external power supply through the power jack on the pod, and through the 25 pin D type connector (the EPC is not powered through 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 PWR jumper. This is in order to isolate the internal PC power supply from the external power supply connected to the pod through the power jack.

## The MC-PWR Jumper

The MC-PWR jumper selects which voltage source of the POD, will power the VDD & VDDX voltage groups of the HC12 controller on the pod & the PRU buffers. This jumper has significance only in case these voltage groups are configured to be powered by a voltage source on the POD (selected by the VDD & VDDX jumpers - see below for details).

When there is a jumper-top on pins 1 & 2, a 5V voltage will be selected, and when there is a jumper-top on pins 2 & 3, a 3.3V voltage (from an internal regulator on the POD) will be selected.

This jumper is factory configured to jumper-top on pins 1 & 2 (5V voltage is selected).

## The VDD Jumper

The VDD jumper selects how to power the VDD pins of the HC12 controller on the POD. When the jumper-top is on pins 1 & 2, the HC12 VDD pins are powered by the POD 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 & 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.

*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 POD or the Target.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VDD pins of the HC12 controller on the pod are powered by the POD).

**The VDDX Jumper**

The VDDX jumper selects how to power the VDDX pins of the HC12 controller on the POD and the PRU buffers. When the jumper-top is on pins 1 & 2, the HC12 VDDX pins and the PRU buffers are powered by the POD 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 & 3, the HC12 VDDX pins and the PRU buffers are connected to the VDDX pins of the adapter to the target, allowing the HC12 VDDX pins and the PRU buffers 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 POD or the Target.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VDDX pins of the HC12 controller on the pod are powered by the POD).

**The VSSA Jumper**

The VSSA jumper selects how to power the VSSA pin of the HC12 controller on the POD (powering the HC12 internal A/D converter). When the jumper-top is on pins 1 & 2, the HC12 VSSA pin is connected to the general POD ground. When the jumper top is on pins 2 & 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 POD or the Target, and not by a combination of the two.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VSSA pin of the HC12 controller on the pod is connected to the POD ground).

**The VDDA Jumper**

The VDDA jumper selects how to power the VDDA pin of the HC12 controller on the POD (powering the HC12 internal A/D converter). When the jumper-top is on pins 1 & 2, the HC12 VDDA pin is powered by the main POD 5V voltage. When the jumper top is on pins 2 & 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 POD or the Target, and not by a combination of the two.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VDDA pin of the HC12 controller on the pod is powered by the main POD 5V voltage).

## The VRH Jumper

The VRH jumper selects how to power the VRH pin of the HC12 controller on the POD (supplying the reference voltage for the HC12 internal A/D converter). When the jumper-top is on pins 1 & 2, the HC12 VRH pin is powered by the main POD 5V voltage. When the jumper top is on pins 2 & 3, the HC12 VRH pin is connected to the VRH pin of the adapter to the target, allowing the HC12 VRH pin to be powered by the target.

*Note: It is recommended to have both the VRH jumper and the VRL jumper configured the same, to supply the reference voltage for the internal HC12 A/D converter by either the POD or the Target, and not by a combination of the two.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VRH pin of the HC12 controller on the pod is powered by the main POD 5V voltage).

## The VRL Jumper

The VRL jumper selects how to power the VRL pin of the HC12 controller on the POD (supplying the reference voltage of the HC12 internal A/D converter). When the jumper-top is on pins 1 & 2, the HC12 VRL pin is connected to the general POD ground. When the jumper top is on pins 2 & 3, the HC12 VRL pin is connected to the VRL pin of the adapter to the target, allowing the HC12 VRL pin to be powered by the target.

*Note: It is recommended to have both the VRH jumper and the VRL jumper configured the same, to supply the reference voltage for the internal HC12 A/D converter by either the POD or the Target, and not by a combination of the two.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VRL pin of the HC12 controller on the pod is connected to the POD ground).

## The VFP Jumper

The VFP jumper selects how to power the VFP pin of the HC12 controller on the POD (supplying voltage to erase & program the internal HC12 Flash memory). When the jumper-top is on pins 1 & 2, the HC12 VFP pin is powered by a voltage received from the 2 pin VFP-IN header. When the jumper top is on pins 2 & 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 Pod, will supply 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. It is recommended 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.

*Warning: It is very important not to supply a voltage lower than the HC12 VDD to the VFP pin of the HC12 controller. Such a low voltage will fry the Schottky diode, which supply the default voltage of VDD to this pin, and might also cause a failure of the HC12 control-*

*ler. Also, when an erase & program voltage is supplied to the VFP pin of the HC12, it should always be within the recommended range specified by Motorola (currently 11.4V nominal).*

This jumper is factory configured to jumper-top on pins 1 & 2 (the VFP pin of the HC12 controller on the pod is powered by the voltage received from the VFP-IN header). Also, for the default configuration, since no voltage source is supplied to the VFP-IN header, the VFP pin of the HC12 will be forced to VDD by a Schotkky Diode on the pod.

**The EXTAL & XTAL Jumpers**

The EXTAL & XTAL jumpers select which clock source will feed the EXTAL & XTAL pins of the HC12 controller on the pod. When the jumper-tops of the two jumpers are on pins 1 & 2, the HC12 EXTAL & XTAL pins will be connected to a 16MHz crystal on the pod. When the jumper-tops of the two jumpers are on pins 2 & 3, the HC12 EXTAL & XTAL pins will be connected to the EXTAL & XTAL pins of the adapter to the target, allowing the HC12 clock to be supplied from the target.

*Note: It is recommended to have both the EXTAL and the XTAL jumpers configured the same, to supply the clock frequency to the HC12 either from the POD or from the Target, and not from a combination of the two.*

These two jumpers are factory configured to jumper-tops on pins 1 & 2 (the EXTAL & XTAL pins of the HC12 controller on the pod are connected to a 16MHz crystal on the Pod).

**The ECLK Jumper**

The ECLK jumper selects what kind of signal will be recreated and driven on the PE4/ECLK pin going to the target. This signal can be configured as the ECLK output of the HC12 or an I/O signal, depending on the HC12 operating mode selected, and the configuration of the PEAR & MODE registers.

The HC12 controller on the Pod is generating control signals on most of the Port E pins (including PE4/ECLK) which are used internally by the emulator. Therefore the emulator is recreating the required functionality of PE4 for the target use, either as the ECLK control signal or an I/O signal. When the ECLK jumper has a jumper-top on pins 1 & 2, the described recreated version of PE4/ ECLK pin is connected to the PE4 pin of the adapter to the target. This configuration should be used for most cases.

The recreation of the PE4/ECLK, introduce a delay of 10nSEC – 15nSEC, which may be critical in some cases. Such a case is, when Expanded Wide or Narrow operating mode is being emulated, where the PE4 pin should carry the HC12 ECLK signal, which is required to de-multiplex the address & data on the target. In such a case, the jumper top on the ECLK jumper can be mounted between pins 2 & 3, which will connect the ECLK signal directly from the HC12 controller to the PE4 pin of the adapter to the target, preventing unnecessary delay.

More differences between the two ECLK jumper configurations are:

- The recreated PE4 version (jumper top on pins 1 & 2) supports the use of the target PE4 pin as both I/O signal and ECLK signal (depending on the operating mode and the PEAR & MODE registers). It also causes the ECLK signal in Normal Expanded Wide and Narrow operating modes, when IVIS bit in the MODE register is cleared, to stay low during internal HC12 accesses, and switch states only during external memory accesses (as specified in the B32 & BC32 manuals).
- The non-recreated ECLK version (jumper top on pins 2 & 3) does not support the use of the target PE4 pin as an I/O signal. It supports the use of the target PE4 pin only as an ECLK signal. It switches the state of the ECLK signal in Normal Expanded Wide and Narrow operating modes during all cycles - internal & external accesses (which is different from the specified in the B32 & BC32 manuals).

*Note: It is recommended to have the ECLK jumper configured with jumper top on pins 1 & 2 whenever possible. This configuration supports all operating modes, and is safer to use.*

This jumper is factory configured to jumper-top on pins 1 & 2 (the recreated PE4/ECLK signal is connected to pin PE4 of the adapter to the target).

**The DBE Jumper**

The DBE jumper selects what kind of signal will be recreated and driven on the PE7/DBE pin going to the target. This signal can be configured as the DBE control signal or an I/O signal, depending on the HC12 operating mode selected, and the configuration of the PEAR & MODE registers.

The HC12 controller on the Pod is generating control signals on most of the Port E pins (including PE7/DBE) which are used internally by the emulator. Therefore the emulator is recreating the required functionality of PE7 for the target use, either as the DBE control signal or an I/O signal. When the DBE jumper has a jumper-top on pins 1 & 2, the described recreated version of PE7/DBE pin is connected to the PE7 pin of the adapter to the target. This configuration should be used for most cases.

The recreation of the PE7/DBE, introduce a delay of 10nSEC – 15nSEC, which may be critical in some cases. Such a case is, when Expanded Wide or Narrow operating mode is being emulated, and the PE7 pin should carry the DBE signal, which is required to enable read data to be driven to the address/data bus. In such a case, the jumper top on the DBE jumper can be mounted between pins 2 & 3, which will connect the DBE signal directly from the HC12 controller to the PE7 pin of the adapter to the target, preventing unnecessary delay.

More differences between the two DBE jumper configurations are:
- The recreated PE7 version (jumper top on pins 1 & 2) supports the use of the target PE7 pin as both I/O signal and DBE signal (depending on the operating mode and the PEAR & MODE registers). It also causes the DBE signal to become active (low) only during external read accesses to addresses which are mapped to the target.
- The non-recreated DBE version (jumper top on pins 2 & 3) does not support the use of the target PE7 pin as an I/O signal. It supports the use of the target PE7 pin only as a DBE signal. It also causes the DBE signal to become active (low) during every external read access, regardless of the address mapping.

*Note: It is recommended to have the DBE jumper configured with jumper top on pins 1 & 2 whenever possible. This configuration supports all operating modes, and is safer to use. It should also be noted that at the time of writing (November 1998), the B32 & BC32 controllers had insufficient DBE timing. For this reason, even the non-delayed DBE version cannot be used to enable read data to the multiplexed address/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).*

This jumper is factory configured to jumper-top on pins 1 & 2 (the recreated PE7/DBE signal is connected to pin PE7 of the adapter to the target).

# Connection Headers Description

## The VFP-IN Header

The VFP-IN Header is used to supply the erase & program voltage for the internal Flash memory of the HC12 controller on the pod (in case it is selected to supply this voltage by the VFP Jumper – see section 'The VFP Jumper' above 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 & 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 Pod 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. In order to prevent a failure of this Schottky Diode do not supply a lower voltage than the HC12 VDD to the VFP-IN connector. The voltage applied to the VFP-IN header should always be greater than or equal to the HC12 VDD, and lower than or equal to the recommended Flash Erase & Program voltage.*

## The PWR-TP Header

The PWR-TP Header is carrying the internal Pod VCC (+5V) and GND signals for user monitoring. Its two pins are labeled VCC & GND accordingly. Do not supply voltage to this header, since it is not designed for this purpose.

## The ADDR16 Header

The ADDR16 Header is used to enable the user to take advantage of all 128K bytes of emulation memory, and all 128K bytes of shadow memory. The ADDR16 header is connected to both of these memory devices on the pod, so the user can connect some signal (Port signal etc.), to this header pin, to emulate two 64Kbytes pages. On the Pod the ADDR16 signal is connected to a 15K pull-up resistor, to default this signal to high, when no signal is connected to the ADDR16 header.

*Notes: The ADDR16 header is optional, and in most cases will not be used. When this header pin is used, the 128Kbytes content of the Shadow memory may be viewed in the SHADOW FAR window.*

# Switches Description

The Reset Pushbutton (RES-SW)

The Reset Pushbutton is used to Reset the emulator and the HC12 controller. The result when pressing the Reset Pushbutton is similar to the result of invoking Reset from the software (by clicking on the Reset icon in the user interface for example), except that it will not initialize any register to a specified value as the PC software Reset will.

# LEDs Description

The POWER LED

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

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 Pod before the software is started, and before the pod 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 turn on, and that's because the POD FPGA is not configured yet, and therefore cannot turn this LED on.*

The EMUL LED

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

The STOP LED

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

The WAIT LED

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

## The ERROR LED

The ERROR LED, when turned on, is used to signal 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 & MODE HC12 SFR Registers'). In case such 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 either click on the Reset icon in the software user interface, or press the Reset pushbutton on the Pod.

# Signal Description of the 80 B32 & BC32 Target Signals

**The VDD, VDDX, VDDA, VSSA, VRH, VRL, VFP, EXTAL & XTAL Pins of the Adapter to the Target**

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

## The VSS & 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 POD Ground (GND). These signals should also be connected to the Target main Ground (if a target is connected to the Pod) in order to supply a low resistance path between the Pod 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, which 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 & 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 Pod, for the Target use. The recreated Reset signal is bi-directional as the Reset of the B32 and the BC32 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 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 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, when Reset from the Software User-Interface, Reset from the pod push-button, Reset on low HC12 VDDX and VDD voltage, or Reset on ERROR condition become active. In order to enable thispin 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 also, 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 PAD7-PAD0, PT7-PT0, PS7-PS0, PP7-PP0, PDLC6-PDLC0 (B32)**
& PCAN6-PCAN0 (BC32) of the Adapter to the Target

These Port pins are connected directly between the HC12 controller on the Pod and the Target adapter. The emulator does 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 Pod. The reason these pins are not available directly from the HC12

controller 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 & PORTE SFR registers. It is following the Port E functionality described in the B32 & the BC32 manuals, with some minor adjustments to make these signals suitable for the emulator use. The recreated functionality for these signals is described in details on a pin to pin basis below.

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

The PE1 signal has a pull-up resistor constantly enabled.

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

The PE6 & 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 Signal**

In Single-Chip Modes the recreated PE7 signal always functions as a general-purpose I/O signal. In Expanded-Wide & Expanded-Narrow Modes, the recreated PE7 signal may be configured to function either as the DBE control signal, or as a general-purpose I/O signal depending on the NDBE bit in the PEAR register. When configured as the DBE control signal, it will become active (low) only for external read accesses, which are mapped to the target.

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

*Note: In applications where PE7 is used only as the DBE control signal, the DBE jumper may be configured to supply the DBE signal directly from the HC12 to the target, in order not to introduce additional PRU delay for this signal. In such cases the DBE signal will become active during every external HC12 read, regardless of the map (See more details above in the section 'The DBE Jumper').*

**The PE6/MODB/IPIPE1 & PE5/MODA/IPIPE0 Signals**

During Reset the recreated PE6 & PE5 behave as the MODB & 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 & 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 & PE5 pins do have effect on the Operating Mode, which will be implemented by the emulator. In this case, the PE6 & PE5 signals along with the SMODN signal, set the Operating Mode, which will be implemented by the emulator right after Reset is over.

During Reset, the PE6 & PE5 have two 33K pull-down resistors activated, and the SMODN has a constant 33K pull-up resistor, which default the emulator to Normal Single-Chip Operating Mode, when no signal is overriding these pull-downs & pull-up values.

When Reset is not active, the PE6 & PE5 signals are used either as general-purpose I/O port signals, or as the IPIPE1 & IPIPE0 control signals. In Single-Chip Mode or when the PIPOE bit in the PEAR register is cleared, the PE6 & PE5 pins function as general-purpose I/O port signals (configured read & written by accessing the DDRE & 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 & IPIPE0 control signals.

**The PE4/ECLK Signal**

In Expanded-Wide and Expanded-Narrow Operating Modes, the recreated PE4 signal always function 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 & 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 which accesses an external memory location of the HC12. This is in order to latch the address from the multiplexed address/ data bus, and signal when the data phase of the external bus cycle is in progress. During bus cycles, which access an internal HC12 resource, the recreated ECLK signal will be maintained low in order to signal that this is an internal HC12 access.

*Note: In applications where PE4 is used only as the ECLK control signal, the ECLK jumper may be configured to supply the ECLK signal directly from the HC12 to the target, in order not to introduce additional PRU delay for this signal. In this case however, the ECLK signal will switch states during all bus cycles external & internal. (See more details above in section 'The ECLK Jumper')*

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 & written by accessing the DDRE & 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 & written by accessing the DDRE & 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 signals carry valid data. The LSTRB signal may change its state during every bus cycle (internal & external) which differs from the behavior of the actual B32 & BC32 controllers in this case. This behavior should not cause any problem however, since the recreated ECLK signal switches states only during external HC12 accesses.

**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 & written by accessing the DDRE & 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 & external) which differs from the behavior of the actual B32 & BC32 controllers in this case. This behavior should not cause any problem however, since the recreated ECLK signal switches states only during external HC12 accesses.

**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.

This PE1 pin does not support the VPP functionality.

**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 & 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 Pod. 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, depend-on the selected HC12 operating mode. The reduced drive functionality of the Port A & 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 & Port B pins may be read and written by the HC12 reading and writing the PORTA, PORTB, DDRA & DDRB registers. In this case also the pull-up functionality of the Port A & Port B pins is recreated, and may be activated and deactivated by manipulating the PUPA & 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 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, 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 cycle. (Since 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).

# The PEAR & 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 pod is running in Special Expanded Wide Mode in order emulate the Single-Chip and the Expanded-Wide Operating Modes. Similarly it is running 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 pod is running).

For this reason, the emulator and the HC12 controller on the pod 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 control signals, for the target use. In the HC12 controller on the pod, 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 pod always contains the value 01111X01B or 00111X01B to implement the Special Expanded-Wide Mode or Special Expanded-Narrow Mode accordingly. Similarly, the PEAR register in the HC12 controller on the pod 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 B32 and BC32 manuals.

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 pod maintain their current value. (Since the HC12 controller on the pod is running 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 control signals generated by the HC12 controller on the pod 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 bellow.

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 Reset to the HC12 and the emulator. This ERROR state is entered in this case, because the described second or later write to the PEAR register, has changed the configuration of the Port E control signals of HC12 controller on the pod, in a way which does not support emulation. Therefore the ERROR state is entered, in this case to prevent bus contention and other undesired misbehaviors of the emulator.

After the MODE register is written once 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. During these second or later writes (after the last Reset), the EME, IVIS & ESTR bits should all be written set, and the SMODN, MODB & MODA bits should be written as 011B when Single-Chip or Expanded-Wide Mode is emulated, or as 001B when Expanded-Narrow Mode is emulated. (Except during complex operating mode switch as described in the next section). If it is written by an illegal value different than the allowed values described above (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 Reset to the HC12 and the emulator. This ERROR state is entered in this case, because the described second or later write to the MODE register, has changed the internal configuration of the HC12 controller on the pod, in a way which does not support emulation. Therefore the ERROR state is entered, in this case to prevent bus contention and other undesired misbehaviors of the emulator.

If the ERROR state becomes active (the ERROR LED and the RESET LED turn on), the user can exit this state by either clicking on the Reset icon in the software user interface, or pressing the pod Reset push-button. 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 pod. (01111X01B or 00111X01B 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 switch: 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, with a write value which specifies the new required operating mode, and the 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 MODE register parameters to be implemented. The second write value depend 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 79H should be written on the second write, and in case the 'New Operating Mode' to be implemented

is Expanded-Narrow a value of 39H 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 pod 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 pod 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 pod changes its value to 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 pod and the emulator may be forced to Reset in several ways, using different Reset Sources, which will be described bellow.

After Reset is done, in most cases a short internal emulator 'Reset Transition' program will take over the HC12 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 to 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 set 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 accoring to user configuration.
- Reset From the pod Reset Push Button. This Reset is forced by pressing on the Reset push button on the pod.
- 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 this case in order to protect the emulator and the HC12 controller on the pod from bus contention and other misbehaviors. See more details in section 'The ERROR LED', 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 set by the state of the SMODN, PE6 & 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, 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 pod 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 duration of at least 32 ECLK cycles).

- Reset & Go is initiated by the software user interface. This will force Reset to the emulator and the HC12 controller on the POD 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, is depended on whether Monitor Mode is in progress or Emulation Mode is in progress.

When Monitor Mode is active (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 set 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 set by state of the SMODN, PE6 & PE5 pins of adapter to the target, during the Reset.

- Reset on insufficient HC12 power supply. When the emulator detect that there is insufficient HC12 VDD or VDDX voltage, it forces Reset to the emulator and the HC12 controller on the pod. 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 HC12controller on the pod. 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, when it takes place, also 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 pod (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, and mapping it at addresses 8000H-FFFFH. The default state of the internal HC12 Flash memory when Reset-Transition does not take place, is disabled, and mapped at addresses 0000H-7FFFH.
- 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), 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 program is disabled) there are some limitation, 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 signle-chip mode is emulated. The user may overwrite this, by writing to the EXSTR1 & 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, and will be mapped to addresses 0000H-7FFFH. The user may override these settings by writing to the ROMON & MAPROM 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 pod when Single-Chip Mode is emulated, or from the emulation memory on the pod 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 pod 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.
4. The user program will resume execution at the address specified by the Reset Vector, which was read at step 1.

# The 68HC12 Trace

## 1. 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:

A. 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 instruction are displayed disassembled, the address and data are arranged in fields etc).

B. 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).

C. 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'.

D. Record Time Stamp information for every recorded frame. This allows the user to check the timing of their micro-controller system, measure how long does it take to execute certain functions etc.

E. Record 'Code Coverage' information, which 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 gets executed.

The HC12 Trace is a very powerful tool, which 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.

## 2. Features of the Nohau HC12 Trace

- The Trace is a small 3.1" by 4" card, plugged on top of any HC12 emulator board.
- The same Trace is used to support all HC12 emulator boards and all HC12 derivatives.
- The Trace can record all HC12 memory & SFR accesses, external & internal to the HC12 micro-controller, in all cases.
- The Trace supplies reliable triggering & 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.

- The Trace size is 128K frame (records).

- Up to 24 miscellaneous signals can be recorded in addition to the address, data & control signal.

- Up to 8 additional MSB address lines (A23-A16) may be recorded for derivatives which have more than 16 address signals.

- Up to 8 chip-Select signals may be recorded, for derivatives which have chip-selects.

- 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.

- 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, and 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.

- The Trace may record and display raw bus frames, including instruction fetches data read & write, free cycles etc. This mode is more useful for hardware debugging.

- 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 stops and re-configurations of the trace.

- The Trace is optional.

**3. 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):

- 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.
- 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.

**4. 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 fills up, start overwriting its oldest frames.

The following fields are recorded for each trace frame:

A. 16 bit address information - recorded from the HC12 address bus.
B. 16 bit data information - recorded from the HC12 data bus.
C. 8 bits of control bus information, which specify what is the type of each frame (instruction execution, data read, data write, how wide is the data etc.).
D. Up to 24 miscellaneous signal, which are arranged into three 8 bit fields – Misc A, Misc B & Misc C.
E. Up to 8 additional MSB address bits, used for some HC12 derivatives, which use more than 16 address signals.
F. Up to 8 Chip-Select signals, used for some HC12 derivatives, which generate chip-select signals.
G. 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.
H. 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, and 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 a 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 & 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 & 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 required 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 emolator board. This selection is made in the 'Misc. B' tab in the trace configuration window.

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

The 24 miscellaneous signal, 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.

**5. What Type of Cycles & 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:

A. 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).
B. CPU Writes – enable recording write cycles caused by the HC12 user program.
C. CPU External Reads - enable recording external HC12 read cycles caused by the HC12 user program.
D. 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)
E. BDM Reads & 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.
F. Interrupt Vectors – enable recording interrupt vector read cycles. Identifying these cycles depend on correct cycle type information output by the HC12 micro-controller.
G. Fetch Cycles – enable recording opcode fetch cycles caused by the user program.
H. Free Cycles – enable recording HC12 free cycles, which 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, only free cycle 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.

I. Wait Power-Down State – enable recording a Wait Power Down indication frame, each time Wait Power-Down mode, is initiated by the user program.

J. Stop Power-Down State – enable recording a Stop Power Down indication frame, each time Stop Power-Down mode, is initiated by the user program.

K. 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).

L. 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).

M. 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).

### 6. 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 did it take to execute a certain function, by assign time 0 to the first instruction of a function and then scroll 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 & 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).
7. 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:

A. 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).
B. CPU Writes – enables using CPU write cycles caused by the HC12 user program, in the triggering logic.
C. CPU External Reads - enables using external HC12 read cycles caused by the HC12 user program, in the triggering logic.
D. 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)
E. BDM Reads & 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.
F. 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.
G. Fetch Cycles – enables using opcode fetch cycles caused by the user program, in the triggering logic.
H. 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 cycle 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).

## 8. Triggering and Filterring 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 & Fetch)
- Data Read & Write
- Both Opcodes (Execution & Fetch) and Data Read & 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) to the following three categories:

A. Opcode – include Instruction Execution information and Opcode Fetch cycles.
B. Data Read – include CPU External Read cycles, CPU Internal Read cycles, BDM Read cycles, Interrupt Vector Cycles, and Free Cycles.
C. 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' & '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.

## 9. 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 & fetch,
B) data reads & writes, C) both A & B. In Data Trigger mode the ranges may be associated with either: A) data reads, B) data writes, C) both data reads & 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 properly 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' & '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.

**10. 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.

11. Break Emulation From the Trace

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

A. On Trace Trigger
B. On Trace Stop
C. 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 the 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 therefor 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.

12. 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 becomes low will cause the Trace to trigger, continue to collect number of frames as specified by the post trigger count field in the Trace Setup tab in the Trace Configuration window, then stop recording and displaying the recorded frames to the user. 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 will be low – no trigger will occur.

## 13. 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 & writs 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 & 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 & 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 & 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).

Pay also 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 appear 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 include 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 yourself 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 & Trigger 3 check-boxes under the Active Triggers filed.

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 filed 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 appear 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 appear 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 cause 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 examples 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, when each instruction is followed by a CPU byte read and a CPU byte write cycles, which result from the MOVB instruction.

Example F
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 filed, 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 appear 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 also 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 examples 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 & 879. Break emulation, click on the assembly tab in the source window, and scroll to address 877 (or press CNTRL & 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+ instruc-

tion in function mysprintf.

Example G
How to set a complex trigger which include 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 byte 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). Then look for the execution of the BNE $081C instruction at address 82D three times, which will then cause the trace to trigger. After this trace trigger, the trace will continue to record extra 1000 frame, then stop the trace, and break HC12 emulation (stop to 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 & 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 filed, 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 filed, and 82D will automatically appear. This will allow to identify 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) for 3 times before the trace triggers.

In the Post Trigger Count field write 1000, in order to cause the trace to record 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, till 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 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).