68HC12 Trace Information

What Is a Trace and How Does It Help Debug a Microcontroller System?

A trace is an optional part of an emulator system that supplies advanced debugging capabilities that include:

- Recording the execution of instructions, the bus activity of the microcontroller system under development, and displaying it in an intuitive way for debugging purposes; the executed instruction are displayed disassembled, the address and data are arranged in fields, and so on.

- Recording only a small portion of the bus activity and/or instruction execution, such as a range of addresses, a range of data, and a specific type of bus cycle that you set. This function is called filtering and displays only the information that is of interest at any given point (for example, to allow solving a specific software bug in a specific function).

- Setting complex conditions that identify very specific events caused by the microcontroller system under development. These conditions can include a sequence of several events, and/or a repeat of a certain event for a number of times, which can then cause the trace to stop recording, and for the recorded data to be displayed. This then can or can not also cause the microcontroller to break emulation based on the user settings which makes it behave like a sophisticated breakpoint. This function is called triggering.

- Recording timestamp information for every recorded frame. This allows you to check the timing of the microcontroller system, measure how long it takes to execute certain functions, and so on.

- Recording code coverage information that specifies which instructions were executed at least one time, and which were not. This code coverage function is useful for late debugging stages, in order to make sure all the instructions get executed.

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

Features

- The trace is a small 3.1-inch by 4-inch card that you plug on top of any HC12 emulator board.
- Supports all HC12 emulator boards and all HC12 derivatives.
- Records all HC12 memory and SFR accesses, external and internal to the HC12 microcontroller, in all cases.
- Supplies reliable triggering and filtering of executed instructions and data accesses. Triggering and filtering can be configured to happen only when an instruction is executed, and not when it is fetched. This feature is supported using reconstruction of the internal HC12 instruction queue, and instruction decoder in the trace logic.
- The trace size is 128K frames (records).
- Records up to 24 miscellaneous signals in addition to the address, data and control signal.
- Records up to eight additional MSB address lines (A23-A16) for derivatives that have more than 16 address signals.
- Records up to eight chip-select signals for derivatives that have chip-selects.
- Forty-four bits of timestamp information is recorded, in resolution down to 40 nanoseconds, allowing the timestamp counter to run without overflowing for more than a week.
- Records and displays frames in an *execution oriented* manner. This means that it can record and present the executed instructions and accessed data in the order they occur without having to record the fetched (unexecuted instructions). (Each recorded and displayed instruction is followed immediately by the Data read and writes which are caused by this instruction.) This makes the trace display more useful for software debugging.
- The trace can record and display raw bus frames, including instruction fetches data read and write, free cycles, and so on. This mode is more useful for hardware debugging.
- The trace can 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 reconfigurations of the trace.
- The trace is optional.

How to Control and Display the Trace Content Using the Emulator User Interface

The HC12 Trace has two main windows in the emulator user interface software (Seehau):

Trace Configuration Window

Click on **Config** , and select **Trace** to open this window. From this window, you can configure the HC12 Trace.

Trace Display Window

Click on the Trace icon in the control bar to open this window. This window displays the information that is recorded in the trace.

Details on how to use these two windows to utilize the trace are described later.

Information 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, when fills up, starts overwriting its oldest frames.

The following fields are recorded for each trace frame:

- Sixteen bits for address information that is recorded from the HC12 address bus.
- Sixteen bits for data information that is recorded from the HC12 data bus.
- Eight bits for control bus information that specifies the type of each frame (instruction execution, data read, data write, how wide the data is, and so on).
- Up to 24 miscellaneous signals, which are arranged into three 8-bit fields (Misc A, Misc B and Misc C).
- Up to 8 additional MSB address bits, used for some HC12 derivatives that use more than 16 address signals.
- Up to eight chip-select signals that are used for some HC12 derivatives, and generate chip-select signals.
- Forty-four bits for timestamp information, in resolution down to 40 nanoseconds that allows the timestamp counter to run without overflowing for more than a week.
- Indication for the memory source that the recorded frame originated from, for example, internal HC12 memory, external emulation memory on the emulator board, or external memory on the target.

Timestamp Information

Each recorded frame contains 44 bits of timestamp information. This 44-bit timestamp is generated by a 44-bit counter in the trace. The clock used for the timestamp counter is a 25-MHz signal generated on the pod that can be pre-scaled by any factor in the range 1-255. The prescale value for the timestamp counter can be configured under the **Trace Setup** tab in the Trace Configuration window, in the **Timestamp Prescaler** box.

The 25-MHz clock used for the timestamp is always available, and its rate is not affected by behavior of the HC12 microcontroller (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 40 nanoseconds. Thus it is ideal to use as the timestamp clock, to supply reliable timing information.

Additional MSB Address Bits, Chip-Selects and Miscellaneous Signals

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

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

Each Misc A bit can be configured to record the data on the Misc.-A connector, or an MSB address signal that is generated by the HC12 microcontroller on the pod. This selection is made in the **Misc. A** tab in the Trace Configuration window.

Similarly, each Misc B bit can be configured to record the data on the Misc B connector, or a chip-select signal that is generated by the HC12 microcontroller on the emulator board. This selection is made in the **Misc. B** tab in the Trace Configuration window.

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

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

The selections are:

- On ECLK falling edge
- After ECLK fall (approximately 30 nanoseconds after ECLK falling edge)

- On ECLK rising edge
- After ECLK rise (approximately 30 nanoseconds after ECLK rising edge)

These selections are configured in the Misc. A, Misc. B and Misc. C tabs in the Trace Configuration window.

Global Filtering

The HC12 Trace has global filtering functionality that allows it to 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. Global filtering is configured by selecting the check boxes under the **Includes for Triggers and Filter** field in the **Trace Setup** tab in the Trace Configuration window. The check boxes configuring the global filtering are suffixed as *Filter*, not as *Trigger*.

The selections are:

Instruction Execution enables recording and displaying instructions as they are executed. (This does not record opcode fetches or any instructions which are fetched but not executed.)

CPU Writes enable recording write cycles caused by the HC12 user program.

CPU External Reads enable recording external HC12 read cycles caused by the HC12 user program.

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 derivatives, internal read cycles and free cycles cannot be distinguished from each other.

BDM Reads and Writes enable recording reads and writes that are not caused by the user program but by the HC12 BDM interface. BDM reads happen whenever there is an open Run Time Data window. BDM writes happen whenever you are writing to the HC12 memory during emulation (when user program is running), from an open Run Time Data window, or an open Shadow window.

Interrupt Vectors enable recording interrupt vector read cycles. Identifying these cycles depend on correct cycle type information output by the HC12 micro-controller.

Fetch Cycles enable recording opcode fetch cycles caused by the user program.

Free Cycles enable recording HC12 free cycles. These are cycles during which internal operations are made by the HC12 controller but no actual bus activity. Although during free cycles there is no actual bus activity made by the HC12 controller, to the emulator the HC12 displays information as if there is a read cycle in progress. There is no reliable way to distinguish all free cycles from real read cycles, therefore depending on the HC12 derivative used, some (or even all) free cycles can be classified by the trace as read cycles. For derivatives that 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 can be displayed under this selection.

Wait Power-Down State enables recording a wait power-down indication frame each time a wait power-down mode is initiated by the user program.

Stop Power-Down State enables recording a stop power-down indication frame each time a stop power-Down mode, is initiated by the user program.

Reset State enables recording a reset indication frame each time the HC12 reset becomes active when the user program is running (for example, when an internal watchdog reset occurs, or a target reset occurs).

Reset-Transition State enables recording a Reset-Transition indication frame each time Reset-Transition becomes active after reset. (For details about Reset-Transition, refer to the emulator manual.)

Error State enables recording an error indication frame each time the error mode becomes active when performing illegal write to some internal SFR register. (For details about the error mode, refer to the emulator manual.)

All the frames listed previously are divided to three main groups:

- Instruction execution frames
- Raw bus cycles frames (bus cycles information that is recorded directly from the HC12 bus)
- State frames (indicate on states that became active in the emulator and HC12 system)

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

Fields Displayed in the Trace Display Window

You can customize the Trace Display window to display only the necessary information.

Frame

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. The frames which were recorded after the trigger occurred, are numbered with positive frame numbers.

Address

This field displays the address that was used by the HC12 microcontroller for all the instruction execution and raw bus cycles frames.

Timestamp

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Timestamp**.

The timestamp field displays the timestamp information in one of the following formats:

- Absolute Timestamp Clock Counts (referred as Absolute cycle)
- Relative Timestamp Clock Count to the previous displayed frame (referred to as Relative cycle)
- Absolute Timestamp in time units (referred to as Absolute time)
- Relative Timestamp in time units to the previous displayed frame (referred to as Relative time)

To display the format of the timestamp field, right-click in the Trace window and manipulate the **Relative Timestamp** and **Convert Cycles to Time** menu items. In order to perform time measurements you can also select the absolute time display, then scroll to a specific frame, right-click in the Trace window, and then select **Zero Time at Cursor**. This assigns time 0 to the specified frame, and measures the time of the other frames relative to the specified frame. Thus, you can, for example, measure how long it took to execute a certain function by assigning time 0 to the first instruction of a function, and then scrolling to the RTS instruction of this function.

Miscellaneous

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Misc. Data**.

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

Pod Pins

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Pod Pins**.

This field 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.)

Opcode

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Opcode**.

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

Status

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Status**.

The status field 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.

Instruction

The instruction field displays the disassembled instructions and the memory access type (read, write, fetch, word, byte, and so on).

Symbol

This field can be enabled or disabled by right-clicking in the Trace window, and selecting **Show Symbol**.

The symbol field displays symbols that are associated with displayed instructions and data cycles when a symbol is available for the displayed frame.

Types of Cycles for Triggering

The HC12 Trace can be configured to determine what type of bus cycles can be used in the triggering logic. This is configured by selecting the 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 are:

Instruction Execution enables using instruction execution in the triggering logic. (This enables triggering on the instruction only when it is executed and not when it is fetched, but not executed.)

CPU Writes enable using CPU write cycles caused by the HC12 user program in the triggering logic.

CPU External Reads enable using external HC12 read cycles caused by the HC12 user program in the triggering logic.

CPU Internal Reads enable 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.)

BDM Reads and Writes enable using reads and writes that are not caused by the user program, but by the HC12 BDM interface in the triggering logic. BDM reads happen when there is an open Run Time Data window. BDM writes happen when you write to the HC12 memory during emulation (when user program is running), from an open Run Time Data window, or an open Shadow window.

Interrupt Vectors enable using interrupt vector read cycles in the triggering logic. Identifying these cycles depend on correct cycle type information output by the HC12 microcontroller.

Fetch Cycles enable using opcode fetch cycles caused by the user program in the triggering logic.

Free Cycles enable using HC12 free cycles. These 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 can 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 can be used in the triggering logic here.

The instruction execution information and the described raw bus cycles, when enabled by these eight global triggering enable check boxes, can 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 following description for the triggering system.)

Triggering and Filtering Systems

Trigger Mode

There are two trigger modes: opcode and data.

You can select the active trigger mode under **Trigger Mode** in the **Trace Setup** tab in the Trace Configuration window, by clicking on one of the **Opcode** or **Data** option buttons.

In opcode mode, the programmed triggers and filter ranges under the **Trigger1**, **Trigger2**, **Trigger3** and **Filter** tabs help you to distinguish between:

- Opcodes (Execution and Fetch)
- Data Read and Write

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

In data mode, the programmed triggers and filter ranges under the **Trigger1**, **Trigger2**, **Trigger3** and **Filter** tabs help you distinguish between:

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

Note: *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 and data mode to the following three categories:

Opcode includes instruction execution information and opcode fetch cycles.

Data Read includes CPU external read cycles, CPU internal read cycles, BDM read cycles, interrupt vector cycles, and free cycles.

Data Write includes CPU write cycles and BDM write cycles.

Thus, for example, selecting data mode and configuring a trigger or filter range for data read, enables all the enabled cycles of group B to participate in the particular trigger or filter logic.

Filter Mode

There are two filter modes: normal and window.

You can select the active filter mode under **Filter Mode** in the **Trace Setup** tab in the Trace Configuration window, by clicking on one of the **Normal** or **Window** option buttons.

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 three sequential conditions, select all three check boxes under the **Active Triggers** field (in the **Trace Setup** tab in the Trace Configuration window). To enable only one condition, select **Trigger 1**, and clear **Trigger 2** and **Trigger 3**. In this mode the last enabled trigger can also be assigned a repeat counter that causes 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 can be assigned any value between 1 and 65536.

Window Filter Mode

Trigger 1 starts recording, Trigger 2 stops recording, 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. Each time Trigger 2 is met, the trace disables recording new frames. These occurrences of Trigger 1 and Trigger 2 can enable and disable trace recording an unlimited number of times. In this mode, Trigger 3 functions as a regular trigger, which can cause the trace to stop recording and display the recorded frames. In this mode Trigger 3 can also be assigned a repeat counter which causes 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 can be assigned any value between 1 and 65536.

After a trigger is recognized by the trace in both normal filter and window filter mode, you 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 selects the type of information in an address range, and the type of data that is recorded in the trace memory. This function is achieved by configuring the filter under the **Filter** tab in the Trace Configuration window. When you select **Filter** (under the **Trace Setup** tab in the Trace Configuration window), filtering is enabled according to the programmed ranges in the **Filter** tab. When this check box is cleared, all the enabled filter cycles under **Includes for Triggers and Filter** (regardless of what the address is and what the data is), are recorded in the trace.

Setting Triggers and Filter Ranges

Setting triggers and filter ranges is done in the **Trigger 1**, **Trigger 2**, **Trigger 3** and **Filter** tabs in the Trace Configuration window.

The cycles that can be used for the triggers are determined under the **Includes for Triggers and Filter** field, by the **Trigger** check boxes. (For more details about these selections, see the “Types of Cycles for Triggering” section.)

The cycles that can be used for the filter are determined under the **Includes for Triggers and Filter** field, by the **Filter** check boxes. (For more details about these selections, see the “Global Filtering” section.)

The triggers and filter are configured as address ranges and data ranges, which when identified, cause a trigger or a filter to be recognized. In opcode trigger mode the ranges can also be associated with either:

A. Instruction opcode execution and fetch

B. Data reads and writes

C. Both A and B

In data trigger mode the ranges can be associated with either:

A. Data reads

B. Data writes

C. Both A and B

For more details about how the different cycles are classified into these categories, refer to the “Trigger Mode” section.

The address ranges programmed can be in the range of 00000H - FFFFFH (1-MB space), which properly covers programs of size up to 1 MB. For applications where only 64K 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, set the data mask to 00FF. For 16-bit data accesses, set the data mask to FFFF.

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

Trigger Input and Trigger Output

The trigger input and trigger output are input and output signals to and from the trace, and can be used to connect the trace to other devices (such as logic analyzers and scopes) 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 can 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, and can 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 +5 V, which defaults this input to be inactive when nothing is driven on it.

This input can function in one of two modes: trigger and inhibit. You can select these modes in the **Trace Setup** tab in the Trace Configuration window using the **Trigger Input** option buttons.

When configured as trigger, this input, when becomes low, causes 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 stops recording and displays the recorded frames. This can also lead to emulation break, in case emulation break on trigger or on trace stop is enabled by selecting **Yes, on Trigger** or **Yes, on Trace Stop** in the **Trace Setup** tab in the Trace Configuration window.

When configured as inhibit, this input, when becomes low, inhibits 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, if a trigger is set on address X, and address X is met, the trace triggers only if the trigger input is high. If the trigger input is low, no trigger occurs.

Extend Recording

The extend recording function 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 function is enabled by selecting **Extend Recording?** under the **Trace Setup** tab in the Trace Configuration window. When extend recording is enabled, you can 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 begins, 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 executed, and then stop recording until the next filtered frame.

Stopping the Trace

Stopping the trace can be done in three ways:

- When emulation is stopped, the trace stops automatically
- After a trace trigger or an external trigger occurs, and all the specified post-trigger samples are collected
- When the entire trace memory fills up with frames and **Trace Stop on buffer full** is selected in the **Trace Setup** tab under the Trace Configuration window

Breaking Emulation from the Trace

Emulation break by the trace is configured by selecting one or more of the check boxes under the **Break Emulation?** field in the **Trace Setup** tab in the Trace Configuration window. 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 executes a couple of extra instructions before it finally breaks emulation. The exact number of additional instructions that are executed in these cases, varies from instruction to instruction and depends on the CPU clock. You can configure the trace to break emulation (stop the user code from running), on one of the following two events.

Break Emulation on Trace Trigger

This option causes user code to stop execution as soon as a trigger is recognized by the trace (after the sequence of Trigger 1, then Trigger 2 then, Trigger 3, and so on, as configured by the user, or after trigger input becomes active).

Break Emulation on Trace Stop

This option causes user code to stop execution whenever the trace stopped for some reason. This can happen, for example, after a trigger is recognized by the trace, and all the additional frames are recorded as specified by the **Post Trigger Count** field, or if the trace becomes full and stops because of the **Trace Stop** on buffer full in the Trace Configuration window.

Examples

This example shows how to set up the HC12 trace for various scenarios, and how to utilize its full power for your debug requirements.

All the following examples use the Nohau **hc12time** example that is available with the Seehau software in directory examples. To load this example:

1. Click on the **File** menu.

2. Choose **Load Code**.

3. Select **hc12time.695**.

This example uses the memory range between 0800h - 0BFFh for both instructions and data. This program generates a running clock that can be viewed during run-time. To view this clock:

1. Open a Data window.

2. Using the right mouse button, right-click in the data window and choose **Address Space Shadow**.

3. Right-click again and choose **Display As**, and **Type ASCII**.

4. Switch the address of the window to 0B80 by typing at the bottom of the data window 0B80, in order to view the clock.

5. Click on the GO icon on the control bar. You should now see the clock updating at address 0B80h.

Example A

This example shows how to record information in the trace and display it.

After installing Seehau and running Seehau Config, the Seehau software should be automatically configured to record executed instructions and CPU reads and writes in the trace. To record in the trace:

1. Start Seehau.

2. Load the hc12time example.

3. Open a Trace window by clicking on the TR icon in the control bar.

4. 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.

Note: *The information is displayed in the order it is executed. Every instruction is followed immediately by the data reads and writes which resulted by its execution. Also, all the displayed instructions were actually executed. Unexecuted fetches are not displayed in the Trace window.*

If you don't see anything displayed in the Trace window, make sure you executed all the steps specified here correctly. If you still don't see anything displayed in the Trace window, contact Nohau Technical Support, or your local Nohau representative.

Example B

This example shows how to record and display all the raw bus cycles in the trace display.

1. Bring up the Trace Configuration window by clicking on the **Config** menu and choosing **Trace**.

2. Under the **Includes for Triggers and Filter** field, scroll down. Clear the **Instruction Execution** (Filter) check box.

3. Select the next seven check boxes: **CPU Write (Filter)**, **CPU External Read (Filter)**, **CPU Internal Read (Filter)**, BDM Reads and Writes (Filter), Interrupt Vectors (Filter), Fetch Cycles (Filter) and Free Cycles (Filter). This enables recording all of these raw bus cycles in the trace.

4. Click **OK**.

5. Run the emulator by clicking on the GO icon

6. Stop the trace by clicking on the Trace icon. The recorded raw bus cycles will display in the trace window. Among the raw bus cycles you will find fetch cycles, free cycles, and data read and write cycles.

7. Enable the timestamp field. Right-click in the Trace window and choose **Show Time-stamp**.

8. Configure to show the timestamp as relative time. Right-click in the Trace window and select **Relative Timestamp** and **Convert Cycles to Time**.

In the trace display, every raw bus cycle is between 120 nanoseconds and 160 nanoseconds after the previous bus cycle, which reflects the ECLK rate of 8 MHz (ECLK period is 125 nanoseconds).

Note 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 can 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

This example shows how to trigger and stop the trace on instruction execution.

This example sets a trace trigger on the execution of the instruction at address 082F (LDY 2,SP). To set up this example:

1. Bring up the Trace Configuration window by clicking on the **Config** menu and choosing **Trace**.

2. Under the **Includes for Triggers and Filter**, select these three check boxes: **Instruction Execution (Trigger)**, **Instruction Execution (Filter)**, and **Fetch Cycles (Filter)**. Make sure that all the other check boxes on this list are cleared. This allows triggering on instruction execution, and to record instruction execution and opcode fetches only.

3. Set the **Post Trigger Count** field to 10 to enable recording 10 extra frames after the trigger is recognized.

4. Click on the **Switch to the Trigger 1** tab.

5. Right-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 opens.

6. Click **Opcode Fetch**. This selects to trigger on instruction execution.

7. Click in the **Begin** box and type 82f.

8. Click **End**, and 82f automatically displays.

9. Click **OK**.

The new opcode fetch range 82f-82f is displayed in the upper white field. The **Enabled** check box at the bottom of the **Trigger 1** tab is automatically selected to enable Trigger 1. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable Trigger 1. Leave this check box selected for this example, and click on **OK**. If the emulator is not already running, run it right now by clicking on the GO icon.

The trace stops recording after a second or two, and displays the recorded information, that includes, in this example, execution and opcode fetches as configured, and on the left side of the Trace Display window are, 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 example, the LDY 2,SP instruction at address 82f which trigger 1 was set on. To insure that the trace did not trigger when this instruction was fetched but not executed, scroll backward in the trace window. Note 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

This example shows 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 up this example:

1. Bring up the Trace Configuration window by clicking on the **Config** menu and choosing **Trace**.

2. Under the **Active Trigger** field, clear any activated triggers (if there are any) by clearing the **Trigger 1**, **Trigger 2** and **Trigger 3** check boxes.

3. Under **Includes for Triggers and Filter**, select: **Instruction Execution (Filter)**, and **CPU Write (Filter)**. Clear the remaining filtering check boxes to enable recording only instruction execution and CPU writes in the trace.

4. Next, set up the trace to record only the three instructions at address range 82F-836. Select the **Filter** tab. Remove any existing qualifiers that exist in this tab by selecting them, and then right-clicking and choosing **Remove**.

5. Right-click in the upper white field, and choose **Add**. A new dialog box opens.

6. Click **Opcode Fetch** to select to filter instruction execution information.

7. Click on the **Begin** field and type 82F.

8. Click on the **End** field and change it to 836.

9. Click **OK**.

The new instruction execution address range shows in the upper white field. The **Enabled** check box at the bottom of the **Filter** tab is automatically selected to enable the filter. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable the filter. Leave this check box selected this example, and click **OK**. 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 that include only the three instructions at addresses 82F-836. If the timestamp is not displayed in the trace window already, display it now by right-clicking in the Trace window and choosing **Show Timestamp**. Also change the timestamp to be displayed in relative time units by again right-clicking in the Trace window and choosing **Relative Timestamp** and **Convert Cycles to Time** to select these two options. It is evident that between every execution of the MOVB D,X,D,Y instruction and the LDX 2,SP instruction a long time period has passed indicating that some unrecorded instructions were executed between these two instructions.

Example E

This example shows how to add to the filtered instructions in Example D, and add a filter of data write (or read) to a certain address space.

To set this example, first set the trace as explained in Example D. Note that in Example D under the **Includes for Triggers and Filter** field, CPU write cycles are enabled, which allow recording of these cycles in the trace.

1. Bring up the Trace Configuration window, and click on the **Filter** tab.

2. In the **Filter** tab, right-click on the upper white field and choose **Add**. A dialog box opens.

3. Click **Data R/W**. This selects an address range of data read and write cycles. In this example, the **Includes for Triggers and Filter** list (in the **Trace Setup** tab), the CPU external reads and CPU internal reads are not enabled. Only data writes cycles are recorded by this filter.

4. Click in the **Begin** field and type B80.

5. Click in the **End** field and change to B8F.

6. Click **OK**.

The new Data R/W range is displayed in the upper white field (in addition to the Opcode-Fetch range, which was already present from Example D). Click **OK**. 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 are displayed, as programmed by the filter.

Example F

This example shows how to filter to record a specific instruction and the data cycles which are caused by it, without knowing what addresses these data cycles access.

This example sets a filter that records the MOVB D,X,D,Y instruction at address 820, and all the data cycles that result from its execution. To set up this example:

1. Bring up the Trace Configuration window and clear all the programmed qualifiers for the filter.

2. Click on the **Trace Setup** tab.

3. Select the **Instruction Execution (Filter)**, **CPU Write (Filter)**, **CPU External Read (Filter)**, and **CPU Internal Read (Filter)** check boxes listed under **Includes for Triggers and Filter** to enable recording instruction execution and data read and write cycles. Clear the remaining filtering check boxes.

4. Select the **Filter** tab, and clear all the existing qualifiers (if any).

5. Right-click on the upper white field in the **Filter** tab and select **Add**. A new dialog box opens.

6. Click **Opcode Fetch** to select to filter instruction execution information.

7. Click on the **Begin** field and type 820, and then click on the **End** field and change it to 823. This records the required MOVB D,X,D,Y instruction that occupies addresses 820-823.

8. Click **OK**.

The new instruction execution address range shows in the upper white field. The **Enabled** check box at the bottom of the **Filter** tab is automatically selected to enable the filter. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable the filter. Leave this check box selected for this example.

To record the data cycles which are cause by the MOVB D,X,D,Y instruction, use the Extend-Recording function. Select the **Trace Setup** tab, select **Extend Recording?**, and type 0 to the **Extended Count** field. This configures 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, are recorded. In this example, this enables recording instruction execution and data read and write cycles.

Click **OK**. 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 are displayed when each instruction is followed by a CPU byte read and a CPU byte write cycles, which result from the MOVB instruction.

Example G

This example shows how to filter data write (or read) cycles at specific addresses, and record information to allow figuring which instruction and function caused each of the filtered data write cycles.

This example sets a filter that records the data write cycles to address B80, and the next two instructions which are followed immediately after each of the these data write cycles of address B80.

To set up this example

1. Bring up the Trace Configuration window, clear all the programmed qualifiers for the filter, and disable all the triggers.

2. Select the **Trace Setup** tab and enable recording instruction execution and data write cycles by selecting under the **Includes for Triggers and Filter** field the **Instruction Execution (Filter)** and **CPU Write (Filter)** check boxes. Clear the remaining filtering check boxes.

3. Select the **Filter** tab.

4. Right-click on the upper white field in the **Filter** tab and select **Add**. A new dialog box opens.

5. Click **Data R/W** 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.

6. Click on the **Begin** field and type B80, and then click on the **End** field. B80 automatically appears. This will select to record the required data write cycles to address B80.

7. Click **OK**.

The new Data R/W address range shows in the upper white field. The **Enabled** check box at the bottom of the **Filter** tab is automatically selected to enable the filter. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable the filter. Leave this check box selected for this example.

To record the next two instructions that follow each occurrence of Data Write to address B80, use the Extend-Recording function. Select the **Trace Setup** tab, select **Extend Recording?** and type 2 in the **Extended Count** field. This configures 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, are recorded. In this example, this enables the recording of instruction execution and data write cycles.

Click **OK**. 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 is displayed in the trace window. Analyzing it shows that address B80 is roughly being written to every 50uSEC by looking at the timestamp information (enabled by right-clicking in the Trace window and selecting **Show Timestamp**), and is followed by two instructions at addresses 877 and 879. Break emulation, click on the **Assembly** tab in the Source window, and scroll to address 877 (or press CTRL+A and then type 877). By looking at this window you see that the data writes to address B80 are being caused by the STAB 1,Y+ instruction in function mysprintf.

Example H

This example shows how to set a filter to record only data cycles which are caused by a specific instruction, or a specific range of instructions, without knowing what addresses these data cycles are accessing and without recording the instructions themselves (see “Example F”).

This example sets a filter to record all the data cycles that are caused by the MOVB D,X,D,Y instruction at address 820, without recording the instruction itself. The window filter mode is used for this purpose.

To set up this example:

1. Bring up the Trace Configuration window and clear all the programmed qualifiers for the triggers and the filter. Disable all the triggers and filter.

2. Click on the **Trace Setup** tab and enable recording data read and write cycles by selecting, under the **Includes for Triggers and Filter** field, the **CPU Write (Filter)**, **CPU External Read (Filter)** and **CPU Internal Read (Filter)** check boxes.

3. Clear the remaining filtering check boxes. To enable start and stop recording on instruction execution by Trigger 1 and Trigger 2, select the **Instruction Execution (Trigger)** check box, and clear the remaining triggering check boxes.

4. Under the **Filter Mode** field, click **Window** to select to operate the trace in the window filter mode. This enables recording whenever the MOVB D,X,D,Y instruction at address 820 is executed, and stops recording whenever the instruction at address 824 (which is the next instruction after the MOVB D,X,D,Y instruction) is executed.

To set up the enable recording condition, Trigger 1 will be used.

1. Select the **Trigger 1** tab, and clear all the existing qualifiers (if any). Right-click on the upper white field in the **Trigger 1** tab and select **Add**. A new dialog box opens.

2. Click **Opcode Fetch** to enable recording on instruction execution.

3. Click on the **Begin** field and type 820.

4. Click on the **End** field and 820 automatically shows. This enables recording each time the MOVB D,X,D,Y instruction at address 820 is executed.

5. Click **OK**.

The new instruction execution address range shows in the upper white field. The **Enabled** check box at the bottom of the **Trigger 1** tab is automatically selected to enable Trigger1. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable Trigger 1. Leave this check box selected for this example.

To set up the Stop recording condition, Trigger 2 will be used.

1. Select the **Trigger 2** tab, and clear all the existing qualifiers (if any).

2. Right-click on the upper white field in the **Trigger 2** tab and select **Add**. A new dialog box opens.

3. Click **Opcode Fetch** to select to stop recording on instruction execution.

4. Click on the **Begin** field and type 824, then click on the **End** field and 824 automatically shows. This will select to stop recording each time the ADDD #$0001 instruction at address 824 is executed (this instruction comes immediately after the MOVB D,X,D,Y instruction at address 820).

5. Click **OK**.

The new instruction execution address range shows in the upper white field. The **Enabled** check box at the bottom of the **Trigger 2** tab is automatically selected to enable Trigger2. This check box also appears in the **Trace Setup** tab under the **Active Triggers** field. This check box can always be cleared to disable Trigger 2. Leave this check box selected for this example.

Now, each time the MOVB D,X,D,Y instruction at address 820 executes, recording is enabled by Trigger 1. Then, when the following instruction at address 824 executes, recording is stopped by Trigger 2. The recording enable and stop will then take place many times, whenever these two instructions are executed. Since only data read and write are enabled to be recorded (under the **Includes for Triggers and Filter** check-boxes list) then only the data cycles which occur between the record enable and record stop, will be recorded in the trace.

Click **OK**. 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, a list of CPU read and CPU write cycles will display. Each set of a CPU read and a COU write, result from one execution of the MOVB D,X,D,Y instructions at address 820.

Example I

This example shows how to set a complex trigger that includes detection of two trigger events sequentially, repeats the detection of the last trigger event, then records an extra number of frames, and finally breaks emulation.

This example sets a trigger that waits 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, the execution of the BNE $081C instruction at address 82D occurs three times, causing the trace to trigger. After this trace trigger, the trace continues to record an extra 1000 frames, then stops the trace, and breaks HC12 emulation (stops HC12 from running code).

To set up this example:

1. Bring up the Trace Configuration window and clear all the programmed qualifiers for the filter and triggers, and disable all the triggers.

2. Select the **Trace Setup** tab. Enable triggering on instruction execution and data write by selecting the **Instruction Execution (Trigger)** and **CPU Write (Trigger)** check box under the **Includes for Triggers and Filter** field. Clear all of the other trigger check boxes in this list. Also in this list, enable recording instruction execution and data read and write cycles by selecting the **Instruction Execution (Filter)**, **CPU Write (Filter)**, **CPU External Read (Filter)** and **CPU Internal Read (Filter)** check boxes. Clear the remaining filtering check boxes.

3. Select the **Trigger 1** tab.

4. Right-click on the upper white field and select **Add**. A new dialog box opens.

5. Click **Data R/W** to trigger on data read and write cycles. In this example, this triggers only on data write cycles, since the **CPU External Read (Trigger)** and **CPU Internal Read (Trigger)** check box under the Includes for **Triggers and Filter** list are disabled.

6. Click on the **Begin** field and type B87, then click on the **End** field. B87 automatically appears. This will select to first look for the required data write cycles to address B87.

7. Click **OK**. The new Data R/W address range shows in the upper white field.

8. Right-click on the lower white field and select **Add**. A new dialog box opens.

9. Click on the **Begin** field and type 39, and then click on the **End** field. 39 automatically appears. This will select to trigger on data 39 at address B87.

10. Click **OK**. The new data range of 39-39 shows in the lower white field.

11. Click **Data Mask** at the bottom of the **Trigger 1** tab, and type 00FF. This specifies that the data being looked for is 8 bits wide and not 16 bits wide. Note that the **Enabled** check box at the bottom of the **Trigger 1** tab is automatically selected to enable Trigger 1.

12. Click on the **Trigger 2** tab.

13. Right-click on the upper white field, and then select **Add**. A new dialog box opens.

14. Select the **Opcode Fetch** option to trigger on instruction execution in this trigger.

15. Click on the **Begin** field and type 82D, and then click on the **End** field. 82D automatically appears. This identifies the BNE $081C instruction at address 82D.

16. Click **OK**. The new instruction execution address range shows in the upper white field. Note that the **Enabled** check box at the bottom of the **Trigger 2** tab is automatically selected to enable Trigger 2.

17. Click on the **Trace Setup** tab. Select the **Trigger 1** and **Trigger 2** check boxes. Clear the **Trigger 3**, the **Filter**, and the **Extend Recording** check boxes.

18. Type 3 in the **Last Trig Repeat Count** field to enable the trace to look for trigger 2 (which is the last programmed trigger in this example) for three times before the trace triggers.

19. Type 1000 in the **Post Trigger Count** field to enable the trace to record an extra 1000 frames after it triggers and before it stops recording.

20. Under the **Break Emulation?** field, select the **Yes, on Trace Stop** check box to instruct the trace to break emulation (stop the program from running), after Trigger 1 was met, then Trigger 2 was met three times, 1000 extra frames were recorded, and the trace stopped recording.

21. Click **OK** to close the Trace Configuration window. Click on the Reset icon, and then click on the GO icon to run the program.

If a Shadow Data window is open and displays address B87, you can 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 takes 10 seconds from the time you run the emulator, until the trace recognize the triggers and break emulation.

Looking at the bottom of the Trace window, you can see that 131072 frames were recorded, and four triggers occurred (Trigger 1 once, and Trigger 2 three times). Scroll to frame 0. You 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 back in the Trace window, you can see 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 at frame -79.

If you scroll forward in the Trace window to the end of the window, you see that an extra 1000 frames were recorded after the trace triggered, as configured. If you check in the **Assembly** tab in the Source window, you will also see 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.)