



EMUL12-PC B32, BC32 Getting Started Manual

Version 3.1

Contents

Chapter 1: Nohau EMUL12B-PC: The Hardware Parts	4
Introduction.....	4
HC12 Family as Supported by Nohau Corporation	5
Jumper Settings.....	8
Chapter 2: The Software Parts	10
The Nohau User Interface Seehau	10
Seehau: A General Introduction	10
Connecting the Emulator to your PC	10
Software Installation	10
Directory Structure	11
Configuring the emulator software Seehau.	11
Important Software and Hardware Notes:	12
Starting the Emulator and Seehau	13
Problems ?	13
Shutting down Seehau	14
Selecting the Startup Macro	15
Chapter 3: The Example program and Data Display	16
The Example Program and Data Display:	16
Loading the example program Hc12time.695	16
Viewing Data in Real-time with the Shadow RAM	17
Graphical Display of Data: Gauge	18
Graphical Display of Data: Graph	18
Data Hints	19
Watch Window	19
Inspect Window	20
Setting Hardware and Software Breakpoints	21
Chapter 4: Commands and Macros	23
Seehau Commands and Macros	23
Macro Construction	23
Macro Execution	23
Command Format	24
Command Examples	24
Command Groups:	25
Quick Saving a Configuration Menu using the Apply Button	25
Creating New Buttons	26

Chapter 5: The Trace and Triggers	27
Trace and Triggers	27
Trace and Trigger Overview	27
Trace Memory and Trigger Mechanism Board	27
Trace Board Jumpers	27
Trace Definitions	29
Trace Configuration Menu	29
Includes for Triggers and Filter	31
Trace Window Display	32
Trace Examples	33
Trigger Example on an Address and Data Qualifier	33
Trace Filter Example	36
Conclusions	36

Chapter 1: Nohau EMUL12B-PC: The Hardware Parts

Introduction

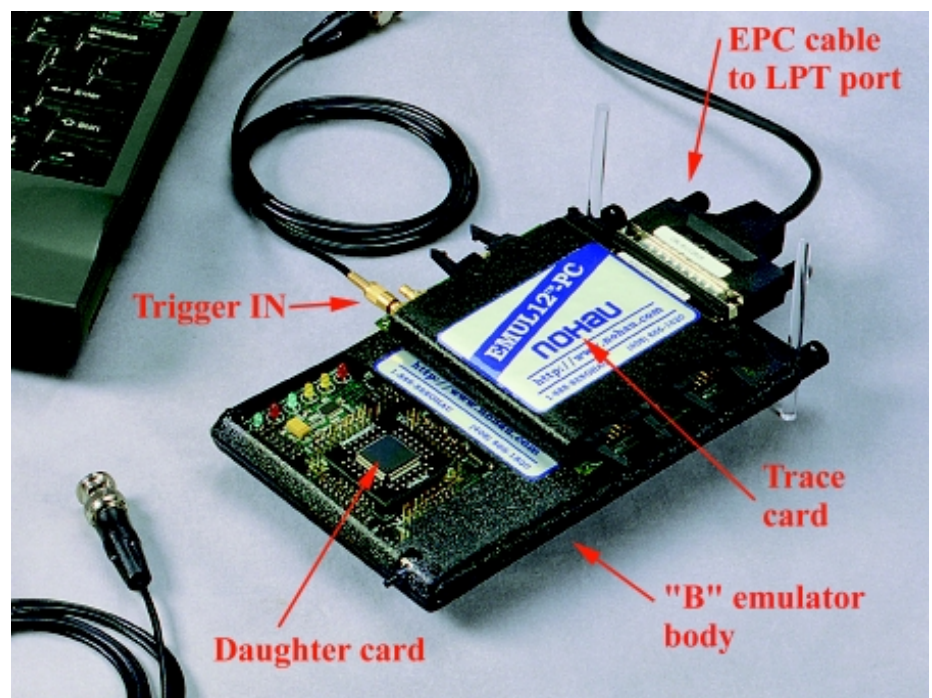
The EMUL12-PC “B” full emulator POD board supports the 68HC912B32 and BC32 microcontrollers. The part number for this board is EMUL12-B/128-16. It provides for 128 Kbytes of emulation memory at a 16 MHz maximum clock speed which translates to 8 MHz bus speed. Support for the B32 or BC32 is accomplished by changing the small economical daughtercard and selecting the appropriate controller in the software configuration menu. Figure 1A shows the B32 emulator with the trace card installed.

The D60, DA128 and DG128 microcontrollers are supported with the “D” emulator base body. The part number is EMUL12D-512-16. This emulator also has various daughtercards available for the supported controllers. The EMUL12-PC is a combination BDM and full feature emulator. All Nohau emulators are Made in the USA. Figure 1B illustrates how Nohau supports the various Motorola HC12 family members.

These emulators are not BDM emulators but full function and provide real-time non-intrusive operation at all CPU speeds. The BDM port is used for some functions when they cannot be provided for by the emulator such as reading internal addresses while running.

The trace is optional and can be added later. The trace provides 128 K frames and can be configured for your needs. Seehau, the new Windows user interface, is standard. Seehau has an extensive Macro scripting language with a Visual BASIC clone. The Macros are easy to configure and use.

Figure 1A
Emulator “B”
POD Board



This document is available as a pdf file on the Nohau website at www.nohau.com.

Figure 2A is the top view of the B32/BC32 emulator showing the internal parts. Figure 2B is the bottom view.

HC12 Family as Supported by Nohau Corporation

FULL EMULATOR TYPE

- ☐ “D” emulator base
- ☐ use the D60, DA128 or DG128 daughtercard
- ☐ use the TR128-16 Trace Board

DG128

128K FLASH
2 CAN
banking

DA128

128K FLASH
1 CAN
banking

D60

60K FLASH
1 CAN
no banking

- ☐ “B” emulator base
- ☐ use B32 or BC32 daughtercard
- ☐ use the TR128-16 Trace Board

BC32

32K FLASH
1 CAN
no banking

B32

60K FLASH
J1850 (BDLC)
no banking

- ☐ No full emulator is available
- ☐ use the HC12 BDM

A4

no FLASH
no CAN
banking
demuxed address bus

General Notes:

- 1) The HC12 BDM emulator operates with all the HC12 Family members shown. New derivatives will also be supported as they are announced.
- 2) The “B” emulator base part number is EMUL12-B/128-16.
- 3) The “D” emulator part number is EMUL12-D/512/16.
- 4) The TR128-16 Trace card works with both the “B” and “D” emulator bases. It does not work with the BDM emulator.
- 5) Each full emulator sold with the trace option comes with a HC12 BDM emulator at no extra charge. This promotion will continue for all of the year 2000.
- 6) The emulators need a communications cable (EPC or ISA card) to connect to the PC.
- 7) Various adapters and compilers are available from Nohau.
- 8) All the HC12 derivatives except for the A4 have a multiplexed address and data bus when operated in external bus mode.
- 9) These HC12 parts have the prefix 68HC912 (ie 68HC912B32) except for the A4 which has the part number 68HC812A4.
- 10) All HC12 derivatives have internal EEPROM memory.
- 11) The A4, D60, DA128 and DG128 usually come in a 112 pin TQFP package. The B32 and BC32 usually come in a 80 pin QFP package.
- 12) See the price list for details.
- 13) For devices not listed contact Nohau at rboys@nohau.com.

Figure 1B
Emulator “B”
POD Board

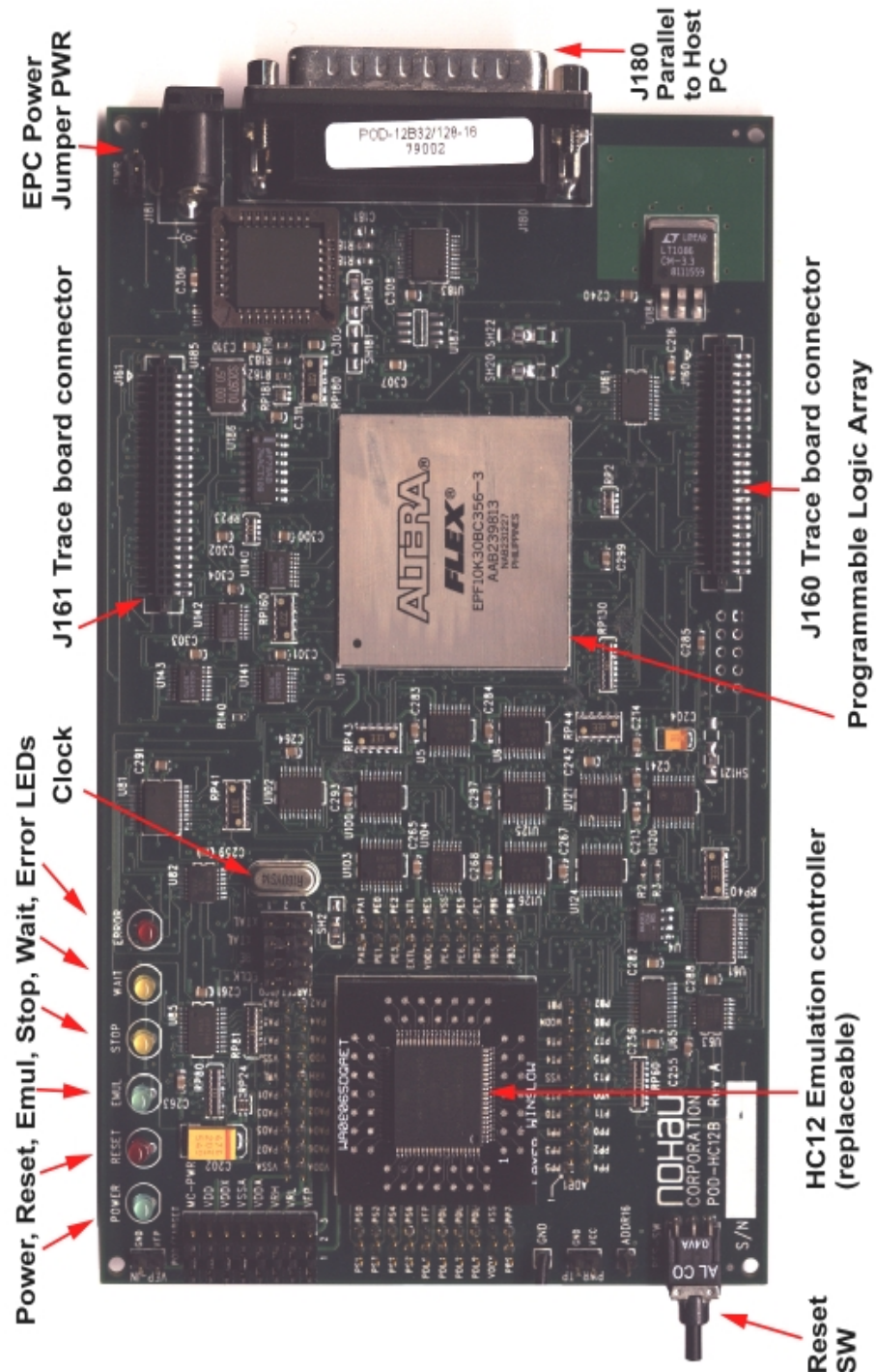


Figure 2A
Emulator POD Board: Top View

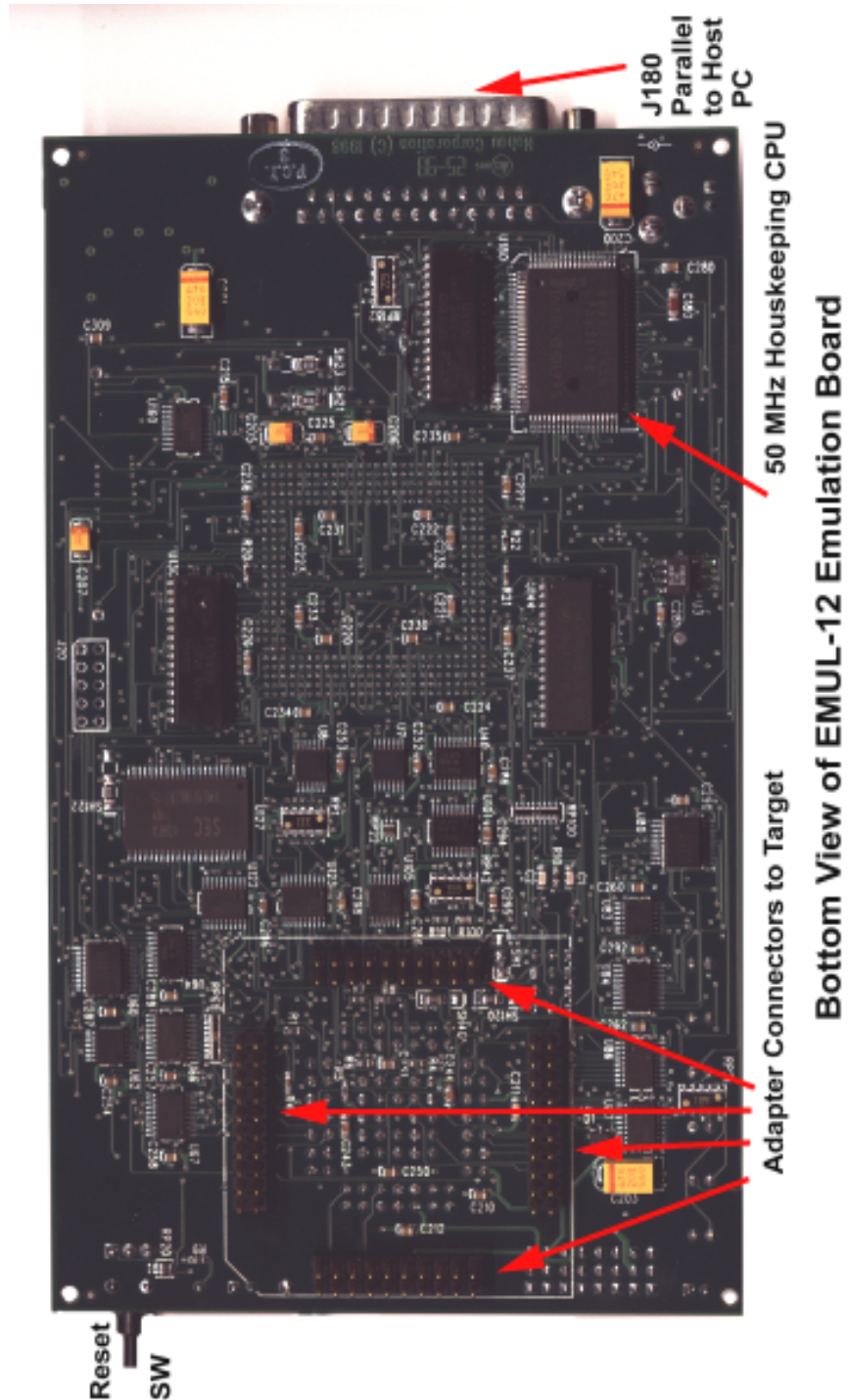


Figure 2B
Emulator POD Board: Bottom

Jumper Settings

Figure 2C shows the top view of the B32/BC32 emulation pod. Figure 2D is an expanded view of the area around the emulation microcontroller and shows the default positions of the jumpers. See the online help or the hardware manual for the meaning of the jumpers.

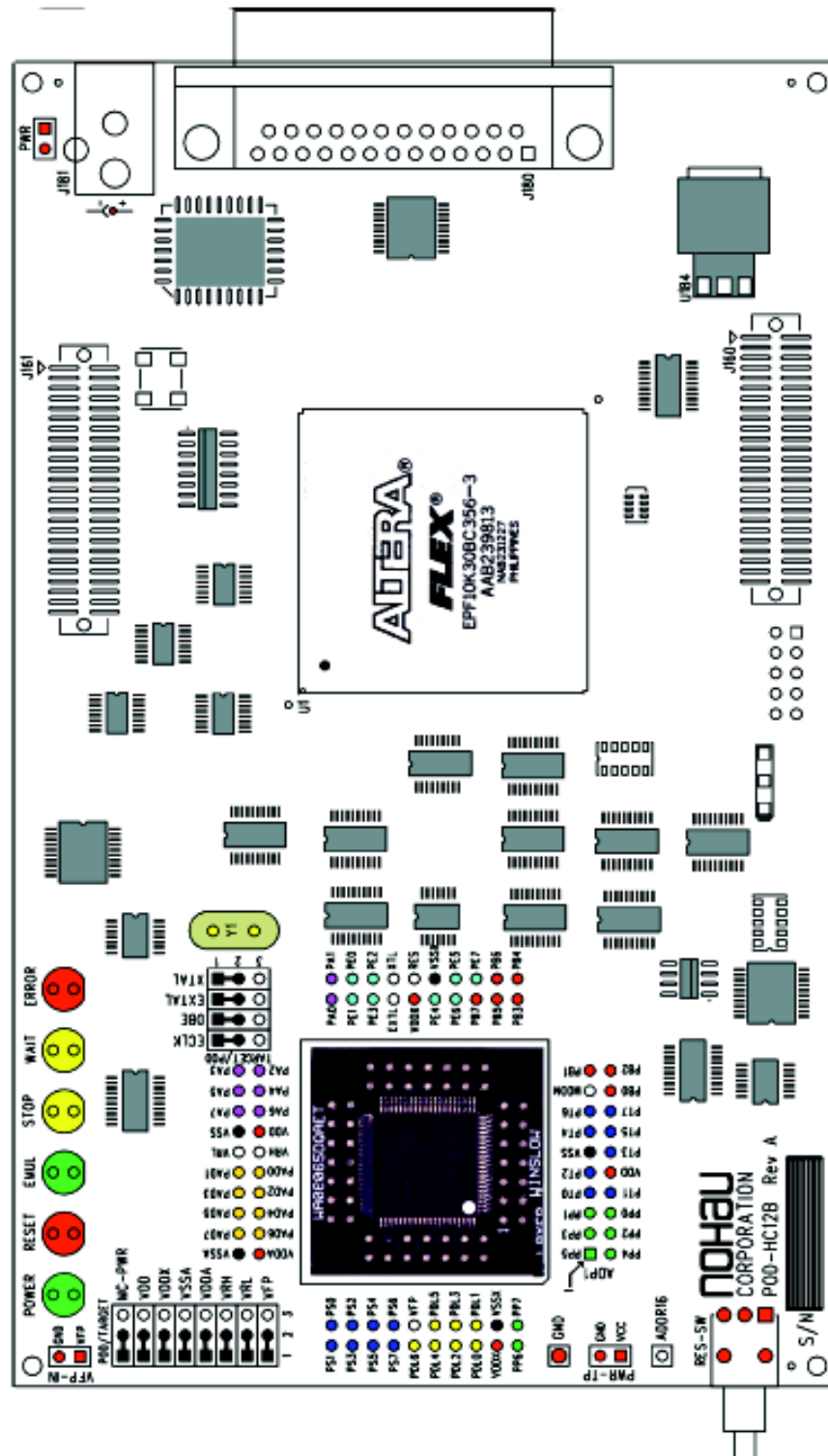


Figure 2C
Line Drawing of the Emulator POD Board: Top View

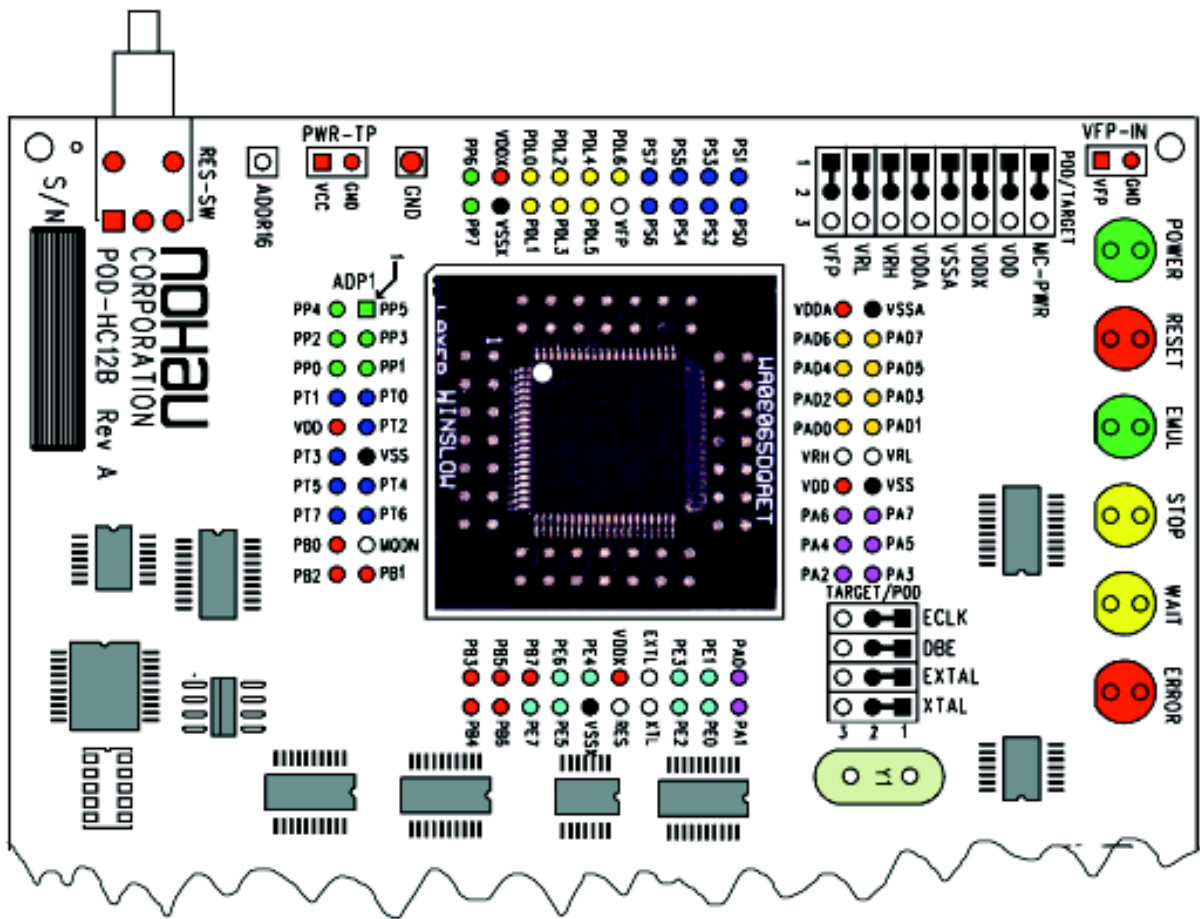


Figure 2D
Line Drawing of the Emulator POD Board: Expanded View

Chapter 2: The Software Parts

The Nohau User Interface Seehau

Seehau: A General Introduction

The SeeHau Macro based GUI is designed to provide a consistent user friendly interface for all Nohau in-circuit emulator families. SeeHau is a High Level Language (HLL) debugger that allows you to load, run, single-step and stop programs, set and view trace and triggers, modify and view memory contents including SFRs, and set software and hardware breakpoints. SeeHau runs under Windows 95, 98 and NT.

Seehau has the capability to run over a TCP/IP stack. You can easily control the emulator over the Internet from anywhere. SeeHau is also an OLE Automation server. This means that SeeHau based emulators can be manipulated from an application developed in any environment that supports OLE Automation. OLE (Object Linking and Embedding) Automation allows one application to drive another application. The driving application is known as an automation client or automation controller, and the application being driven is known as an automation server or automation component. You can control SeeHau from C++, Java, Delphi or Visual Basic.

Connecting the Emulator to your PC

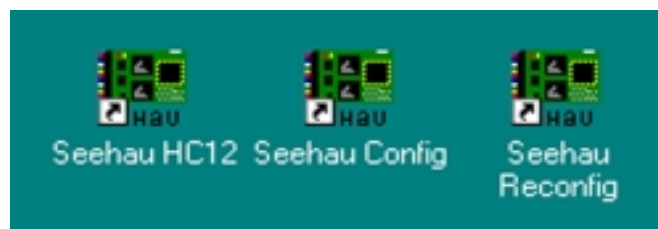
There are two methods available to connect the emulator to your PC. One is the EPC (Emulator Parallel Cable) cable as shown on the cover of this manual. This connects to a LPTx printer port. This provides the most portable method of connection and is perfect for laptops. The second is a ISA card that installs in your PC. No PC interrupts are used or needed by either method. The EPC typically uses port 378 (LPT1) and the ISA card uses 200. These values can be modified as needed. If you have trouble with your ISA card try 210.

Software Installation

Seehau comes on a CD ROM and works with Windows 95, 98 and NT. Install the software to your PC hard drive in the manner required by the version of Windows you are using. This document uses Windows 95 in the examples. The examples used do not require a target hardware. The emulator will be used in stand-alone mode. No target is needed for operation by any Nohau HC12 emulator except the BDM models.

Figure 1 shows the icons that are installed on your Windows Desktop at the end of the installation program setup.exe found on the CD-ROM. SeeHau HC12 is the debugger software while SeeHau Config is the utility used to configure SeeHau to your particular hardware setup. SeeHau Reconfig recalls previous settings for convenience allowing you to change only selected items. Depending on your Windows version, you may be able to drag these icons on to your desktop. SeeHau Config is used to construct startup.bas which is the

Figure 1



startup macro file. It is an ASCII file hence can be edited with any editor. You can modify many configuration items when SeeHau is running under the Config, emulator menu selection. You are unable to modify the controller selected this way. You must run the config or reconfig utility or manually edit the startup.bas file to accomplish this.

Always use the Windows Uninstall utility to unload any existing version of SeeHau from your computer before loading in another copy. Do not simply delete the SeeHau files and/or folders. Do not install SeeHau on top of an existing copy. SeeHau will not allow this action. Under My Computer select Control panel, then Add/Remove Programs. After Uninstall has run, you might want to delete any files and folders that Uninstall did not delete. Save your personal macros and source files first. Startup.bas in the Macro directory is created by the configuration program which will overwrite an existing copy of startup.bas. You may want to save your old copy. An entry in the file seehau.ini in the c:\Nohau\SeehauHC12 directory specifies the name of the startup macro file.

Directory Structure

The installation program will create Nohau\SeehauHC12 as the default directory. In the root of this directory is an utility for creating an in-line compilation macro called CompileMacGen.exe. Ncore.log contains a listing of commands useful for Nohau technical support to solve any problems you might report. Seehau.ini contains the name of the macro file used upon the startup of Seehau. Startup.bas is the default. Read_Me.txt and WhatsNew.doc contain useful information. Please read these files. They will help you. FeedBack.doc is a template for sending Nohau useful feedback. You can also send email to support@nohau.com.

Configuring the emulator software Seehau.

When Seehau Config is started, the emulator and trace configuration windows in Figure 2, 3 and 4 will be used to configure the emulator system and create the file startup.bas. The file startup.bas is not provided by the new install. It must be created by the configuration program which is started by clicking on the Seehau Config icon. If you start Seehau HC12 without startup.bas existing in the Macro directory (as in a new install), it will pass you to the configuration program automatically. Seehau will make startup.bas from your input and other information. The Startup.bas file is an ASCII file in the default directory c:\nohau\SeehauHC12\Macro and contains the commands and data used to configure Seehau at startup time.

Note you do not need to have the emulator connected to the PC to run the Seehau Config icon. You do need the emulator connected and with its jumpers properly set for the Seehau regular executive to operate properly. For more information on macros; see the Macro Section in this manual.

There are two windows you must configure: Figure 2 or 3 and Figure 4. You can also access the window in Figure 4 from within Seehau under the Config menu item in the main window. Click on the Emulator or Trace for the appropriate setup desired. Note that a more detailed setup configuration window is available this way.

It is better to get familiar with the emulator in stand-alone mode before attempting to connect to a target hardware system. The added complications of the target hardware may cause you undue problems at this time. Once you have gained some skills at operating the emulator, it will be easier to connect to your target.

This Getting Started manual uses the default jumper settings of the emulator board.

- 1) Click on the Seehau Config icon on your desktop. You do not need the emulator connected at this time.
- 2) A blank Figure 2 will open. You will now configure the emulator. You will need to know what interface connector you are using: EPC or the LC-ISA card.
- 3) Confirm the settings as indicated. If you are using the BC32, select POD-12BC32. The EPC cable and the LC ISA card are the only appropriate choices for the EMUL12. Figure 2 shows the settings used if you are using the EPC cable. The EPC cable is distinguished by a male/female connector on the end that plugs into your PC parallel port. Figure 3 shows the settings for the LC-ISA card. The Emulator Board Address dialog box is for the address of the internal communication link from your computer. For the ISA card, the most common address is 200. This setting can be changed on the board. If you have conflicts,

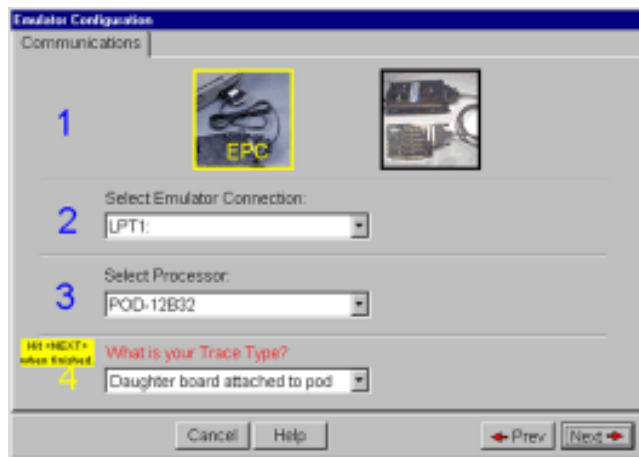


Figure 2

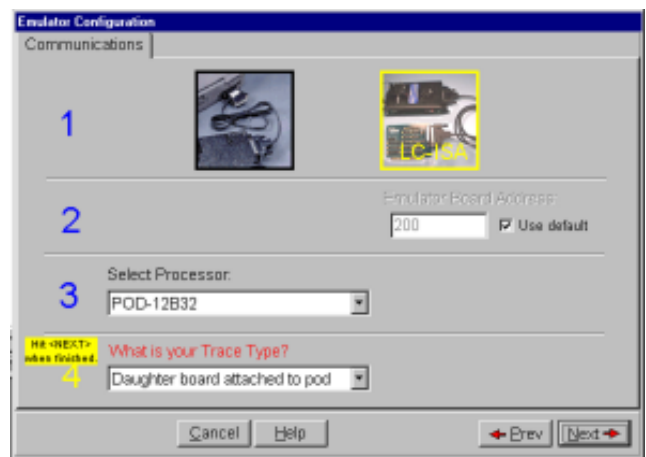


Figure 3

try address 210. If you are using the EPC (Enhanced Parallel Cable), this address is not applicable. The EPC cable by default uses address 378 which represents the LPT1 port on your PC. You can select other LPT addresses.

- 4) If you have the trace board, set this appropriately in the window "What is your trace type?".
- 5) When all the information has been entered in Figure 2 or 3, pressing Next will open up Figure 4.

Clock: The CPU clock frequency. Set it to 16 for the EMUL12-PC. This setting is used only for the calculation of the Trace timestamp. It has no effect on the operating speed of the "B" emulation controller. This window does change the clock speed on the "D" emulator.

Processor: set this to POD-12B32.

Operation Mode: This configures which model the emulator is to operate in. Select Normal Single Chip. All valid operating modes of the HC12 are supported.

These settings will be the default values.

Figure 4

The image shows a Windows-style dialog box titled "Emulator Configuration". It has a tab labeled "Hdw Config". The dialog is divided into several sections:

- Processor:** A dropdown menu showing "POD-12B32".
- Operation Mode:** A dropdown menu showing "Normal Single Chip".
- Port Address:** A text box containing "200".
- Int. Reg. Start Address:** A dropdown menu showing "0000".
- Clock:** A text box containing "16.000000" followed by "MHz".
- Base Clock Divide By:** A dropdown menu showing "1".
- PLL in Use:** An unchecked checkbox.
- Loop Divider Register:** A text box containing "FFF".
- Reference Divider Reg:** A text box containing "FFF".
- Miscellaneous:** A group of checkboxes:
 - ☐ Enable COP Watchdog
 - ☐ Enable Internal Flash
 - ☐ Disable Internal EEPROM
 - ☒ Reset Transition
 - ☐ Reset From Target
 - ☐ Emul Reset to Target
 - ☐ Mask Interrupt on step

At the bottom left, there is a red text label "Required Fields". At the bottom right, there are buttons for "Cancel", "Help", "Prev" (with a left arrow), and "Finish".

- 6) The Seehau configuration program creates startup.bas and Seehau is now configured to run your emulator. The configuration program will ask you if you want to start the emulator. If the emulator is connected and powered up, you may click on Yes, otherwise select No. For this tutorial, please select No.

After you have connected and powered up the emulator EMUL12-PC you will be ready to operate the emulator system. Do not connect the power supply to the emulator at this time.

The emulator is powered by 5 volts regulated at the standard power supply socket. The center pin is positive. The jumper settings will be the default for this part of the manual.

It is possible to operate the emulator remotely via a TCP/IP stack. Contact Nohau Technical Support at support@nohau.com or (408) 866-1820.

Important Software and Hardware Notes:

To reinstall Seehau, use the Add/Remove program in Windows to properly uninstall the files and correct the registry files. Then reinstall a fresh or updated version of Seehau.

Pay attention to the power-up and power-down sequences of the emulator and a target system. When powering up or down the system: the target must never be powered when the pod is not. Power flowing from the target into the controller will damage it. Power up the pod first, and power it down last.

Starting the Emulator and Seehau

- 1) The four jumpers labeled ECLK, DBE, EXTAL and XTAL must be on pins 1 & 2 (POD). This is default.
- 2) The eight jumpers located next to the POWER led must be in positions 1 & 2 (POD). This is the default.
- 3) If you are using the EPC, the PWR jumper must be connected to send power to the EPC. This jumper is located beside the power socket. If you are using the ISA card, this jumper must be left open.
- 4) All other jumpers must be unconnected. Note PWR-TP is Vcc and ground for use to monitor the 5 volt supply. Never put a jumper on this connector.
- 5) Connect the cable between the PC parallel port and the 25 pin connector on the emulator board. If you are using the ISA card connect the cable from this card to the emulator.
- 6) It might be a good idea to double check your settings.
- 7) Connect the Nohau 5 volt power supply to the emulator.
- 8) The green POWER led only will illuminate.
- 9) Double click on the Seehau HC12 icon on your PC desktop.
- 10) Seehau will configure itself from startup.bas. “Macro” will be displayed in red at the bottom of the main window while startup.bas is running. Wait for this message to go out.
- 12) If you click on the Reset icon, the red RESET led will then light and after about 1/2 second will go out. Position and size the main window to your preference. You can open up new windows. They are found in the New menu item on the main Seehau window or by clicking on the add new window icons labelled SRC, DAT, TR, WA and REG.

Problems ?

Problems are usually from the PWR or the clock incorrectly connected. Review instructions 1, 2, 3 and 4. Make sure the cable to the PC is correctly connected. If you are using the EPC, try it without a printer connected. Do not have the emulator connected to a target system or adapter. You should have a 16 MHz oscillator crystal installed. A crystal with RY160YS14 stamped on it is a 16 MHz crystal.

Try another PC. Reload Seehau. If you do this, use the Windows Remove Programs found under My Computer and the System folder. This way, all files and registry entries are properly deleted.

If all this fails, please contact your Nohau representative for technical support: support@nohau.com.

I got my emulator to work and I set my windows up as in Figure 5. I resized the existing windows and opened the Call Stack window under the New menu item.

I also set the program counter to equal 800. You can do this by right clicking on the source window and selecting Go To Address and entering 800 and clicking on OK. You can the PC to a default value when the RESET icon is pressed by selecting Config, Emulator and selecting the Misc. Config tab. Enter the value of 9800 and click on the Program Counter box. Note you can also preset the Stack Pointer in this manner.

Note that you may need to provide program counter and stack pointer reset values under Config, Emulator, Misc. Normally, this information will be provided by the compiler setting the reset vector and stack pointer value. Asserting and entering values in this window will override your compiler's actions at reset time.

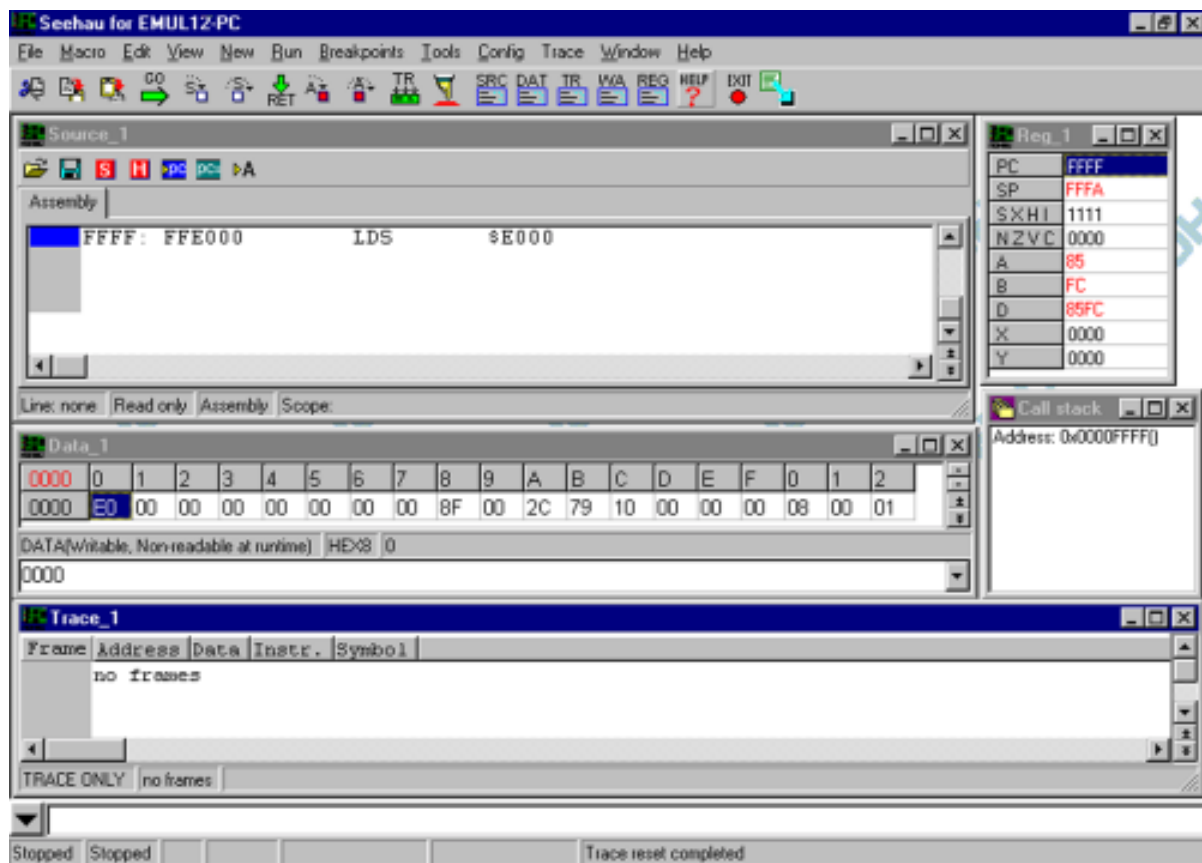
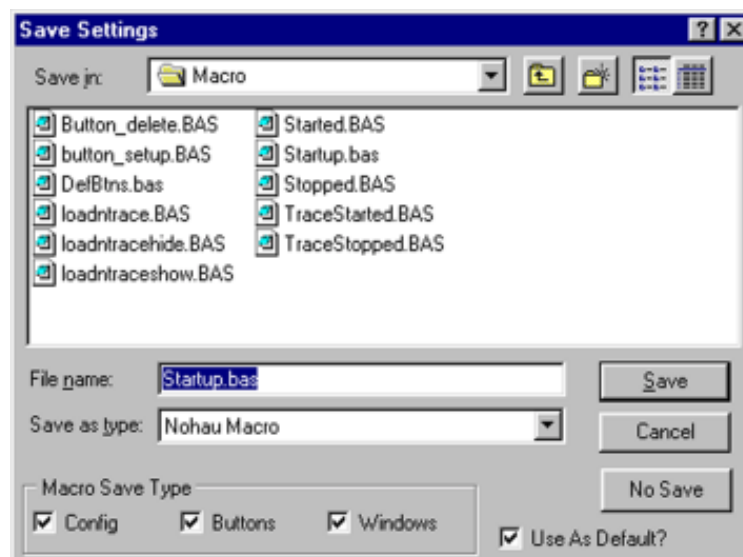


Figure 5

Shutting down SeeHau

- 1) Click on the X in the upper right corner, press ALT X or select the menu item File, Exit. Figure 8 will appear.
- 2) You can save your setup in startup.bas or a macro file of your own naming.
- 3) If the box Use as Default? is enabled, this file will be used the next time SeeHau is started.
- 4) This macro will save those items enabled in the Macro Save Type area.
- 5) Select Save and exit from SeeHau. All your settings will be saved to startup.bas or a file of your choosing.

Figure 6

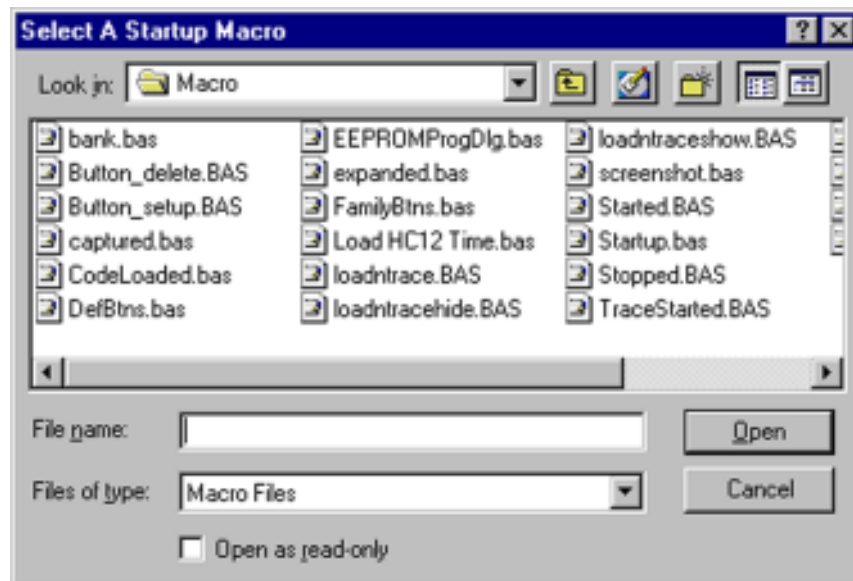


Selecting the Startup Macro

When Sechau is started, the file seechau.ini specifies the macro file that will be executed to configure the emulator. You can modify this file with any ASCII editor to specify any suitable macro. You can also activate a window to query you in order to select the desired macro file:

In the Config, Environment menu, select “Use Startup Dialog?” and click on OK. The next time Sechau is started Figure 7 will appear. You can select the desired macro file.

Figure 7



Chapter 3: The Example program and Data Display

The Example Program and Data Display:

Loading the example program Hc12time.695

- 1) Nohau provides a small example program called hc12time.695. This file, source code and other compiler output files, are found in the c:\nohau\seehauHC12\examples default directory.
- 2) Start the emulator as per the instructions in the preceding section *Starting the Emulator and Seehau*.
- 3) Open the menu item File, Load Code or press CTRL L.



Figure 1

- 4) A window similar to Figure 1 opens up. This shows all the files in the examples directory with the file extension of .695. Your window will now look like Figure 1.
- 5) Highlight Hc12time.695 and click on Open. You can also double-click on Hc12time.695. Hc12time.695 will load into the emulator. This file is the located object file in IEEE 695 format. The source window will show a CLR A instruction at 800 as pointed to by the the reset vector (at FFFE). The JSR at 811 is to the start of the Main program. You may need to scroll to see it or make the Source window larger. Note the emulator software Seehau displays all applicable labels. Seehau can load other file formats such as ELF-DWARF and compiler proprietary formats from companies such as Cosmic, IAR and HiWare.
- 6) Click on the Step Into icon (or press F7) and the program will run to the start of MAIN.
- 7) Note that the hc12time.c tab appears on the Source window. You can easily switch between assembly and source language by clicking on these tabs. No more awkward module selections.
- 8) Right click on the source window with the Hc12time.c tab selected and select Mixed Mode or click on the MM icon. Select *At PC Line*. You will see assembly code mixed in with the appropriate source lines as in Figure 2. Note the program counter (PC) indicated by the blue block at the start of MAIN.
- 9) Remove the Mixed Mode from the Source window so only the C source code remains by right clicking on the Source window and deselecting Mixed Mode or clicking on the MM icon.
- 10) Click on the Source Step Into icon repeatedly and the program counter will advance through the CPU initialization code. Note that where there is assembly code only, the steps are at that level.

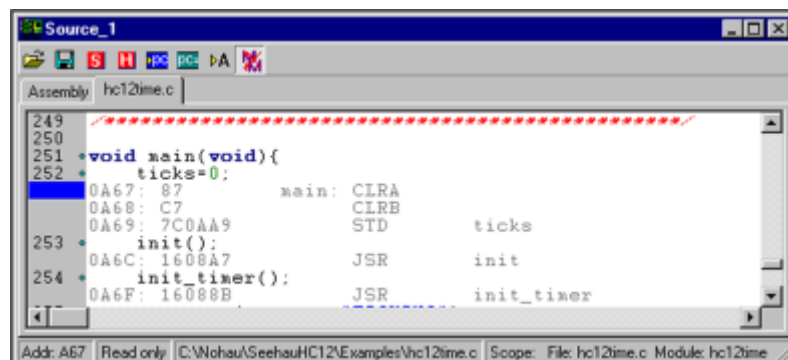
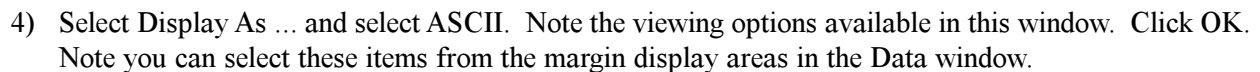


Figure 2

The Nohau Shadow RAM feature allows you to view memory contents in real-time without stealing cycles from the emulation CPU. This example assumes you have completed all the steps so far in this manual and that Hc12time.695 is still loaded in your emulator. You must have done a source step or run the program.

- ### Figure 3

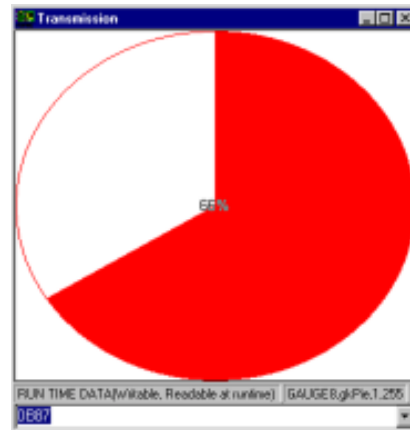


-

Graphical Display of Data: Gauge

Seehau can display data in various graphical methods in real-time.

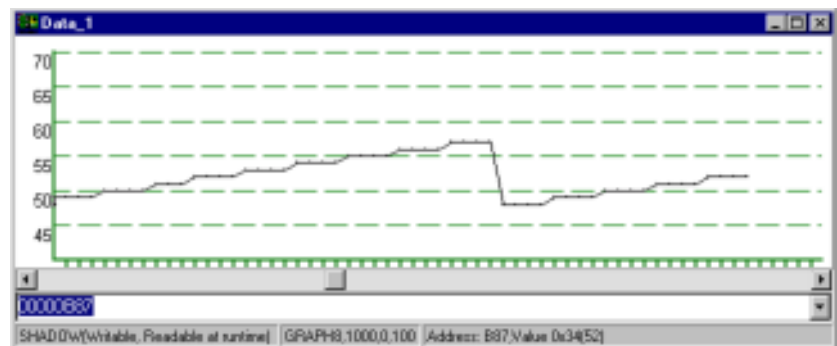
- 1) Open a new Data window by clicking on the data icon or select under View
- 2) In the Data window margin areas where ASCII is elected: change this to Gauge. Note you can do this while emulation is running. Seehau does not steal CPU cycles to do this in the Shadow RAM and usually not in the Run Time Data memory unless Seehau cannot find a free cycle.
- 3) Size the window similar to Figure 5.
- 4) Make sure Shadow RAM or Run Time Data is selected and set the address at the bottom left of B87.

Figure 5

- 5) The gauge will display the value of B87 in terms of its percentage of FF hex.
- 6) The range is not very suitable. Note that the B87 values range from 30 to 39 hex which corresponds to 48 to 57 decimal. These represent the ASCII characters 0 through 9.
- 7) Right click on this Data window and select Setting, Gauge. Enter the range of 48 and 57b for the Min and Max values. Note you can change the display mode to various types of charts. Click on OK.
- 8) The display will be more suitable looking. I also changed the colour to red.
- 9) Right click on the Data window and select Change Caption and do so. I used Transmission. You can have as many data windows as you like and you can save them and recall them with a button.

Graphical Display of Data: Graph

- 1) Open another Data window.
- 2) Select Graph from the local menu or in the margin of the Data window.
- 3) Figure 6 will appear: resize it to a suitable size. Once again note you do not have stop emulation to create and to configure these windows.
- 4) Set the range to around 40 to 70 to get a display range similar to Figure 6.
- 5) Note you can make many of the same changes as you can for the graph display in Figure 5.

Figure 6

Seehau can display the contents of variables and structures by holding the cursor over them in the Source window. The emulation must be stopped for this to display.

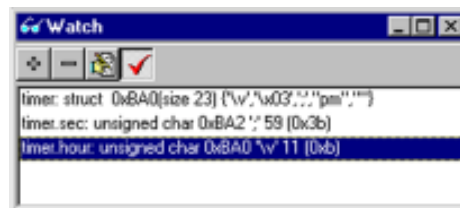
Figure 7

```
timer = struct 0xBA0(size 23)
0: hour unsigned char '\v' 11 (0xb)
1: min unsigned char '\03' 3 (0x3)
2: sec unsigned char '\59' 59 (0x3b)
3: am_pm unsigned char [10] 0xBA3 (size 10)
D: status unsigned char [10] 0xBAD (size 10)
```

Watch windows allow you to view data variables and structures. Multiple Watch windows can be opened and they can contain multiple variables and structures. Figure 8 shows the Watch window we will now create.

- 1) Open a Watch window by clicking on the Watch window icon or selecting the New, Add Watch menu item. Position it according to your preferences. You can also select View, Watch.
- 2) Click on the plus (+) sign to add symbol name. I used timer, timer.sec and timer.hour. These will be added as in Figure 8.
- 3) The red check mark is used to “park” the symbol name.
- 4) An alternative method of creating a Watch window is to block the variable name in the source window and right click on it and select Debug Windows , Add to Watch.
- 5) You can also double click on a symbol in the Symbol Browser which is found under View, Symbol Browser.

Figure 8



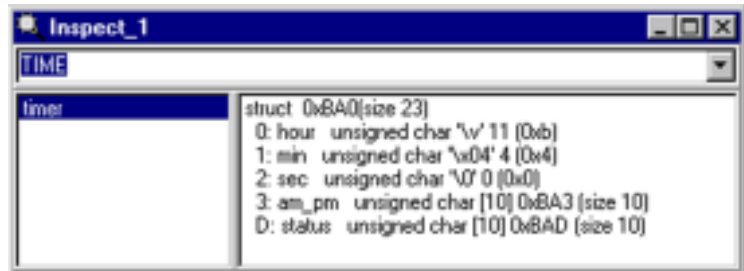
Inspect Window

An Inspect window is used to view the contents of only one variable or structure. The contents of the variable or of an element can be modified but not in real-time while emulation is running. Data can be modified in real-time with the Run Time Data and Shadow windows.

Opening the Inspect Window

- 1) Select the View, Inspect menu item. A window similar to Figure 9, but empty, will open.

Figure 9



- 2) Enter timer in the top data entry bar. The contents of the structure time will display as in Figure 9.
- 3) Run, then stop the timer.c program and the data values will change. Only the sec element will change unless you run the program for greater than one minute.

Modifying the Data in the Inspect Window

- 1) Block the hour element and make a right mouse click. Select Modify from the menu.
- 2) Enter a new value and click on OK. The new value will be entered into the Inspect window.
- 3) You can also block a variable name, right mouse click and select Inspect to accomplish the same thing.

Viewing more members of the structure

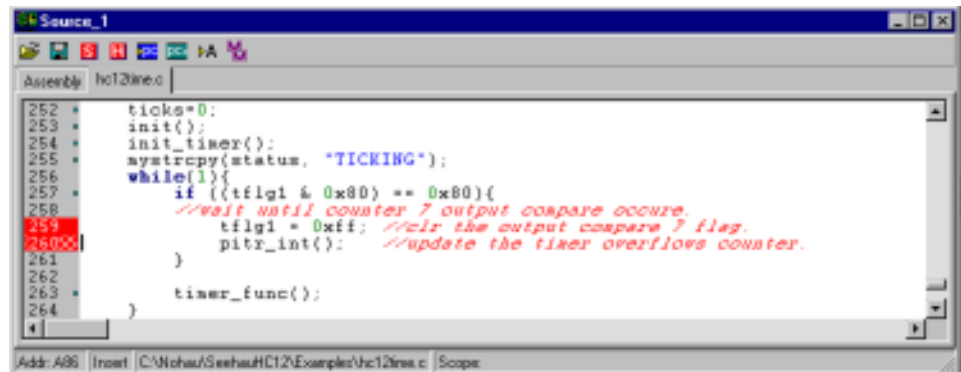
If you double click on a structure element or select Inspect Space from the local menu, this element will be displayed. If the element is a structure within a structure, this new structure will be displayed.

Setting Hardware and Software Breakpoints

The Nohau EMUL12-PC allows you to set both hardware and software breakpoints. The number of possible breakpoints is effectively unlimited. All breakpoints are no skid. This means the instruction a breakpoint is set on is not executed when the program counter reaches it. Hardware breakpoints are the only way to set breakpoints in ROM. This example assumes you have completed all the steps so far in this manual and that Hc12time.695 is still loaded in your emulator. If not, please reload it at this time.

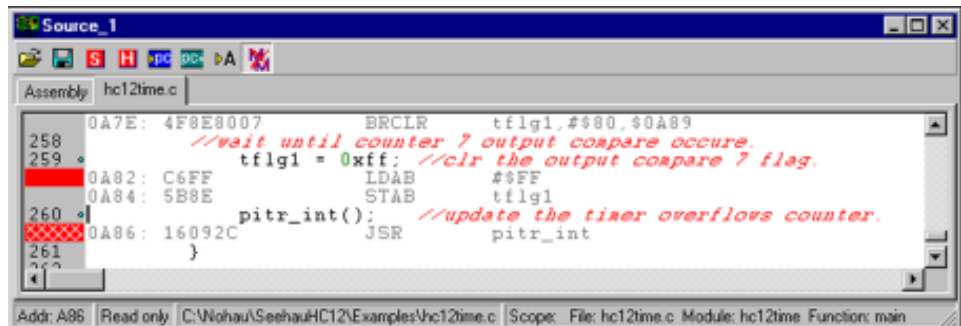
- 1) Click on the Reset icon. Select the hc12time.c tab in the Source window. You may need to click on the Step Into icon to activate this tab.
- 2) Scroll to lines 259 and 260. Note the two small green dots beside each line number. This indicates that there is at least one assembly instruction that a breakpoint can be set on.
- 3) Click on the number 259. The 259 turns red indicating a software breakpoint has been set.
- 4) Click on the number 260 while holding down the ALT key. The 260 turns a cross-hatched red indicating a hardware breakpoint has been set. Figure 10 shows the resulting screen.

Figure 10



- 5) Right click on the Source window and a local menu will open up. Select Mixed Mode. Or click on the MM icon. Scroll if necessary so the breakpoints at lines 259 and 260 appear as in Figure 11.

Figure 11



- 6) Note the software breakpoint is set to physical address 0A82 and the hardware breakpoint to 0A86.
- 7) Click on GO. The PC will advance to line 259 and the emulation will stop. The red block turns magenta indicating the PC is pointing to this location. Note in the Register window in Figure 12, the PC has changed to red and is 0A82. The instruction at the breakpoint has not been executed. The instruction at 0A82 is Load Accumulator B with the immediate value of FF. The Register window clearly shows this has not happened as B contains the value 02.
- 8) Click on GO and the red crosshatch turns magenta. The PC in the Register window is 0A86. Note that the accumulator B contains FF indicating the LDAB instruction was executed. The instruction JSR at 0A86 has not been executed. This is an important feature. Had this instruction been executed, the program flow would be

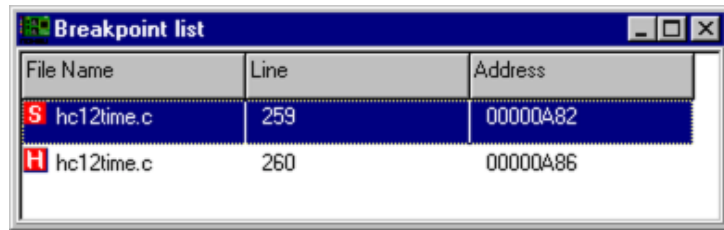
Figure 12

Reg_1	
PC	0A82
SP	0BFD
SXHI	0000
NZVC	0000
A	00
B	02
D	0002
X	0BA3
Y	0B89

diverted to the routine `pitr_int`. There is no indication at that location where the actual JSR was located. It could have been anywhere. A Trace memory would indicate the JSR location that caused `pitr_int` to be entered.

- 9) Click on the Assembly Step Into icon and `pitr_int` will be entered. Click on GO and the program counter will once again stop on 0A82.
- 10) Open the menu item Breakpoints and select Setup. Figure 13 appears. Note the breakpoints you set are shown here with indication whether they are a software or hardware breakpoint. Note that breakpoints can be temporarily disabled to “park” them or deleted entirely under the breakpoints menu item. Breakpoints can be set on a single address or a range of addresses. Close the Breakpoint list window.

Figure 13



	File Name	Line	Address
S	hc12time.c	259	00000A82
H	hc12time.c	260	00000A86

- 11) Click on the number 259 to remove the software breakpoint. The blue block appears which is the program counter position when no breakpoint is activated there. Remove the hardware breakpoint similarly.

Chapter 4: Commands and Macros

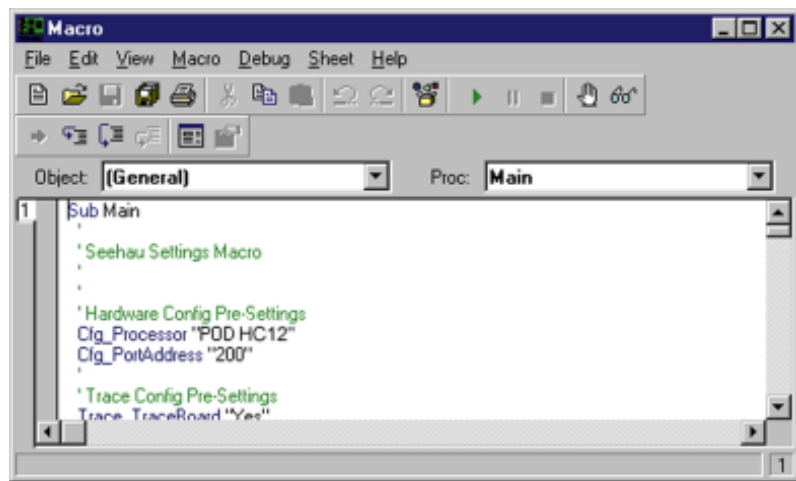
Seehau Commands and Macros

Seehau Commands are used to communicate and control the emulator hardware and the Seehau Software Debugger. These commands are used in the Seehau Macro facilities such as user macros, emulator startup configuration and for Seehau configuration items in general. Commands allow the Seehau debugger software to be a very flexible and powerful tool. Modifications and preference settings are easily made by the user. A Macro as defined here is a collection of commands in a text file that are executed by the Seehau debugger.

Macro Construction

These Macros can be supplied by Nohau or constructed by the user using the built-in recording facilities or by hand with any ASCII editor or with the Seehau Macro Editor. The Macro Edit window is shown in Figure 1. The macro recorder and editor are found under the *Macro* menu. The resulting filename.bas files are simple text files and can be modified at any time with a text editor regardless of how the macro was originally constructed. These files are stored in the *nohau/seehauHC12/macro* directory on your hard drive.

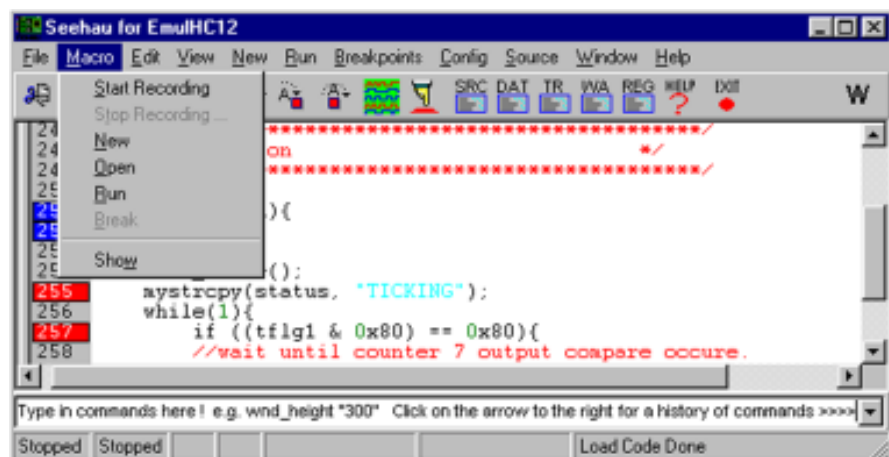
Figure 1



Macro Execution

Macros can be executed by Seehau during the startup phase or manually by the user using the *Macro/Run* command. Individual commands can be executed by direct entry in the main Seehau window. Figure 2 illustrates the dialog box at the bottom of the main window for direct entry. An example is shown. This box can be opened up to display a list of the recently used commands. This eliminates the need to retype most used commands. The box at the bottom of Figure 2 where the word "Done" is where Seehau puts the results of command operations. Not all commands display a result. Note the Macro menu item expanded at the top.

Figure 2



Command Format

Commands are of the format *groupname_commandname_fields*. They are in ASCII text and the names are self-explanatory. Commands are written in upper and lower case letters for easy readability but Seechau is case insensitive here. An example of a command is: *Cfg_PortAddress "378"*. This command tells Seechau that the emulator hardware is connected to the PC port at hex 378 (LPT1). Descriptions of various fields (if applicable) are contained in the Seechau online help.

Command Examples

An example of directly entering a command:

In the dialog entry box enter: `Wnd_height "300"` and press return.

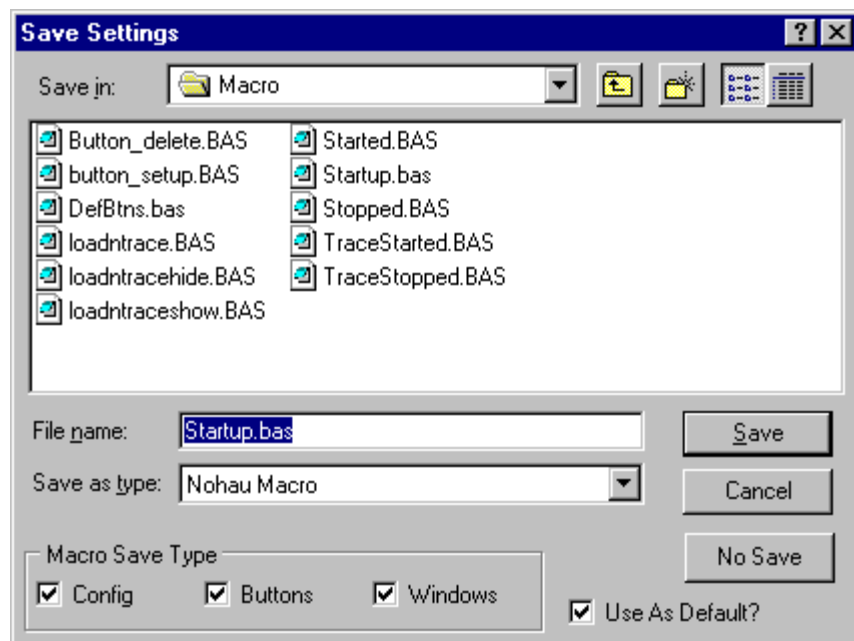
The open window or the one most recently selected by a macro command will have its height adjusted to 300 pixels. To modify the main window, you may have to type in this command first:

`Wnd_select "Seechau for EmulHC12"`

Enter the appropriate fields for the window you want to adjust as indicated as the caption on the window.

If you enter these commands in the file `/Macro/startup.bas`, they will be executed everytime you start up Seechau. You can also make your own Macro that saves Seechau configuration setup. You are given this opportunity when you leave Seechau. Figure 3 shows the options you can select:

Figure 3



Config:

This enables the saving of hardware configuration items such as breakpoints and triggers.

Buttons:

This enables the saving of the position and type of buttons displayed on the main window.

Windows:

This enables which windows are displayed and their position and size.

Use as Default ?:

The specified file will be the one used by Seechau on startup. `Startup.bas` is the initial default. If `startup.bas` is not present, Seechau will make one from a composite of various existing .bas files. You can specify the default to be your own Macro.

Command Groups:

Macro commands are divided into 12 groups. Each group represents one basic function. The group is specified in the first part of the command name as in *Brkpt_clrBrkPoints*. Seehau Online Help files contain detailed information of all the Macro commands. Consult the Help file for details on fields associated with the commands and for more examples.

- 1) **BrkPT:** these commands are associated with breakpoints. *Brkpt_clrBrkPoints* clears all the breakpoints.
- 2) **Cfg:** general emulator configuration settings. *Cfg_map* maps emulator memory to the target.
- 3) **Data:** operations on data such as move, search and set. *Data_Move* moves data from a source to a destination address.
- 4) **File:** operations on files such as symbol table and user code loads into the emulator or target memory. *File_LdCode filename* loads the specified user object code into the emulator.
- 5) **Hlp:** Help commands. Displays help on various subjects from the Seehau Online help file.
- 6) **Prg:** Program Commands. These refer to the user code in the source window. *Prg_SelectModule* selects the source module to be displayed in the source window.
- 7) **Reg:** CPU Register commands used to inform Seehau of the name, size and address of the CPU registers. These are used in the operation of Seehau and also to display registers in the Register window. *Reg_Size "16"* informs Seehau the previously selected register is 16 bits wide.
- 8) **Run:** commands used to start the emulation CPU. Commands include Run Break, Run Stepper and Run GoForever.
- 9) **Symbr:** Used to access commands regarding the source code.
- 10) **Trace:** Commands that control the hardware Trace functions. *Trace_TraceBegin* is used to start the Trace.
- 11) **View:** These commands control the Inspect, Evaluate and Watch windows. *View_Evaluate* passes the expression to Seehau it is to evaluate.
- 12) **Wnd:** Windows commands. These are used to control aspects of the windows including position and size, button definition and show/hide windows.

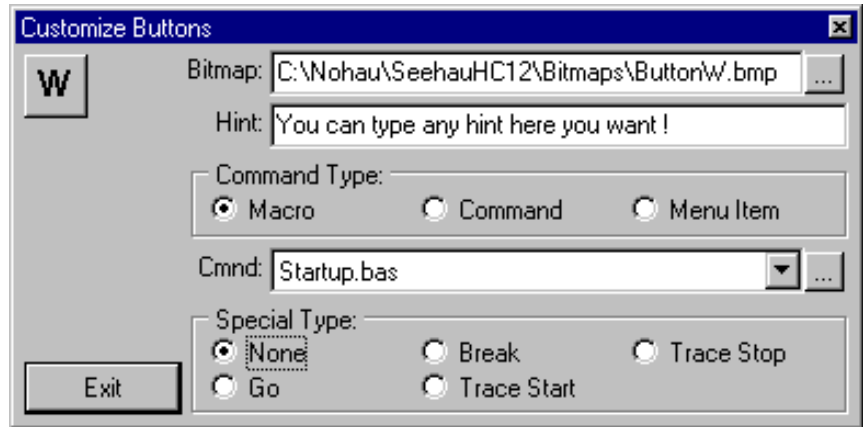
Quick Saving a Configuration Menu using the Apply Button

- 1) The configuration menu must be open and the Apply button visible.
- 2) In the Macro menu item, select Start Recording. The configuration window will disappear.
- 3) Re-open the configuration window from the appropriate menu item. Click on Apply. The settings associated with this window will be saved.
- 4) In the Macro menu, select Stop Recording.
- 5) The Save Macro window will open giving you the opportunity to choose the filename for the newly created macro. See Figure 3. Enter a filename of your choosing and click on Save. The macro is ready to use and will accurately re-create your configuration settings.

Creating New Buttons

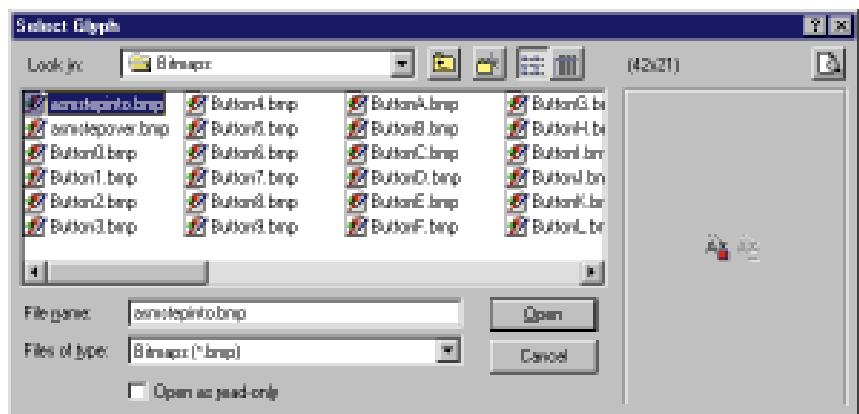
- Buttons can be created and deleted on the icon bar in the main menu. A created button can be attached to a macro you have created. The icons themselves are Windows bitmaps (*.bmp) and are easily created with virtually any graphics program.
- Open the menu Config, Buttons and an empty Figure 4 will open.

Figure 4



- Open the bitmap browser by clicking on the ... button at the right side of the dialog box.
- Figure 5 will open up showing the bitmaps in the Bitmaps directory. Note that when a bitmap is highlighted, its image appears on the right side of the window as in Figure 5.
- Highlight ButtonW.bmp. Click on Open and its image will appear as in Figure 5 on the right side.

Figure 5



- In the Hint: dialog box shown in Figure 4, type in the sentence as shown. Position the cursor on the W icon and note this sentence will appear in a yellow box. This will be used in the main window.
- In the Command Type: area, click on Macro. This area indicates what type of action will be associated with the new button. The options in the Cmd: box will change according to the Command Type setting.
- In the Cmd: dialog box, select startup.bas. You can choose the down arrow to open the Macro directory or use the browser by clicking on the ... button.
- Click and drag the W icon to the main window to the right of the EXIT icon. The W icon will be placed at this point. Click on Exit in the Customize Buttons window shown in Figure 4.
- Confirm that if you place the cursor on the W icon the hint you entered appears.
- Click on the W icon in the main window and the startup.bas sequence will be executed.
- To remove a button, drag it back to the Customize Buttons window. To quickly access the Customize Buttons window, right click on the button and select Buttons. The Customize Buttons window will appear.

Chapter 5: The Trace and Triggers

Trace and Triggers

Trace and Trigger Overview

The trace memory and triggers can be configured and viewed without intrusion into real-time emulation. This is accomplished with a dedicated 50 MHz housekeeping controller rather than stealing cycles from the emulation CPU. The triggers are versatile, yet easy to configure and modify.

Triggers can be set on addresses and data ranges. They control trace recording or cause the emulator to stop the emulation depending on the options set. Trace and Triggering can trigger and record all internal and external accesses.

Full pipeline decoding ensures only executed instructions and data read/writes are captured as such and no false triggering occurs. The trace memory can be saved to a file. Source and assembly code is displayed in the trace window. The trace window can be configured as to the fields displayed.

The EMUL12 provides three levels of triggers. These are represented by the three Level tabs in Figure 2. The Filter tab is used to determine what is recorded in the trace memory.

The 3 levels are sequentially prioritized. Level 3 can become true only after 1 and 2 have become true. Associated with each level are 50 data qualifiers and 50 address qualifiers.

The Misc. A, B and C tabs select the function of the three data input connectors on the trace board. There are 24 user input data bits that can be recorded in the trace memory.

Trace Memory and Trigger Mechanism Board

The Trace board contains hardware and firmware for the trace memory, triggers, Performance Analysis, Code Coverage, external triggers (in and out), and three general purpose user input connectors. The trace board uses the housekeeping microcontroller located on the main emulator board. This allows the trace to be started and stopped and other functions to be operated without stealing cycles from the emulation controller. Nohau trace boards can be setup and viewed in real-time and on-the-fly.

Trace Board Jumpers

There are no jumpers on the EMUL12 trace board. All options are set with software by configuring the FPGA shown in Figure 1. When L3 is on, there is no firmware loaded into the FPGA. This is accomplished by the Seehau software. There are 4 LEDs:

L2: TRC Collect

The trace is running and collecting and storing data when this green LED lights.

L3: FPGA Not PGM

This red LED lights when the FPGA is not programmed such as during initial powerup. The FPGA is RAM based and is loaded by Seehau.

L4: Trig

This green LED lights when a trigger condition is met. A break or manual emulation stop is considered to be a trigger condition in this case. If Trigger 2 or Trigger 3 is enabled, the last one must become true for this LED to come on.

L5: Status:

Indicates the trace buffer is full wrapping around with new instructions recorded. This LED is green.

The trace board has a trigger in and trigger out connector plus 24 user inputs that can be recorded in the trace memory. See the trace hardware manual for more information.

All the LEDs will go out if the emulation is stopped and you turn the trace on. The trace will not be collecting data at this point as none is being created by the user program.

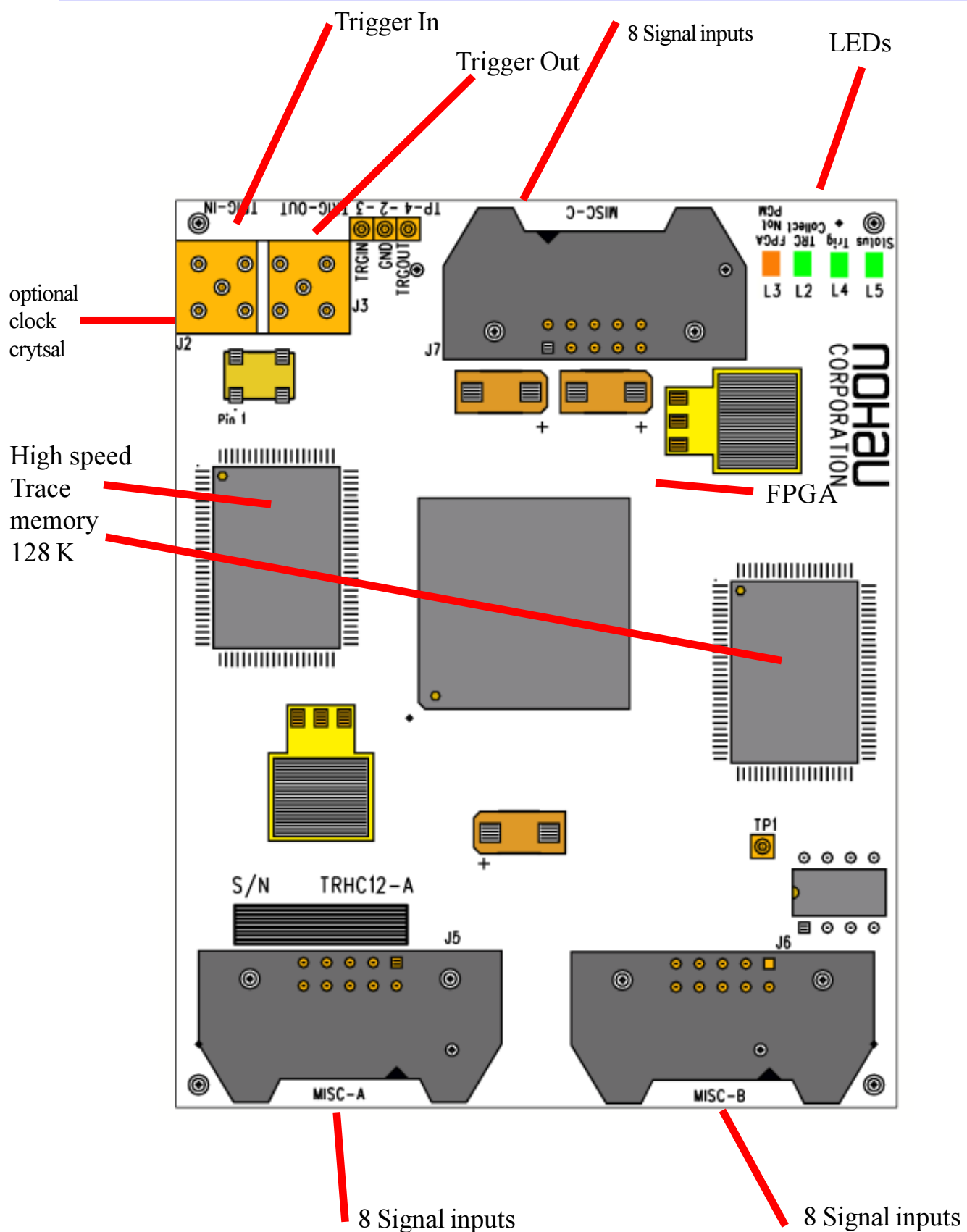


Figure 1
Trace Board

Trace Definitions

- Qualifier:** The specifications that partially define a condition you want to examine. “If address = 5012” is a qualifier. “If address = 5012 AND data write = FE” is also a qualifier.
- Event:** This is the qualifier becoming true.
- Task:** What is to be done. When the qualifier becomes true and therefore the event has happened: do this task. A task is actually a signal or a flag passed to somewhere else in the emulator system. There are plenty of examples of tasks. A trigger is a signal passed to the trace hardware to tell it to start or stop recording.

The trigger is also available as an active low TTL output signal on a connector and can be used to trigger an oscilloscope or logic analyzer. A break is also a signal to the hardware to stop the emulation. A task can be a flag in a memory location used to pass information about an event to another qualifier. An example is when one event has occurred and this fact is needed as input by another qualifier somewhere else.

Note: When most people mention a trigger, they are usually referring to the qualifier or the entire system.

Trace Configuration Menu

The trace and trigger configuration menu is accessed by clicking under Config, Trace in the main menu or the local menu by right clicking on the trace window and selecting Trace Config. Figure 2 will open up.

Trace Type

Specifies whether a trace board is installed. The trace is optional and can be added later.

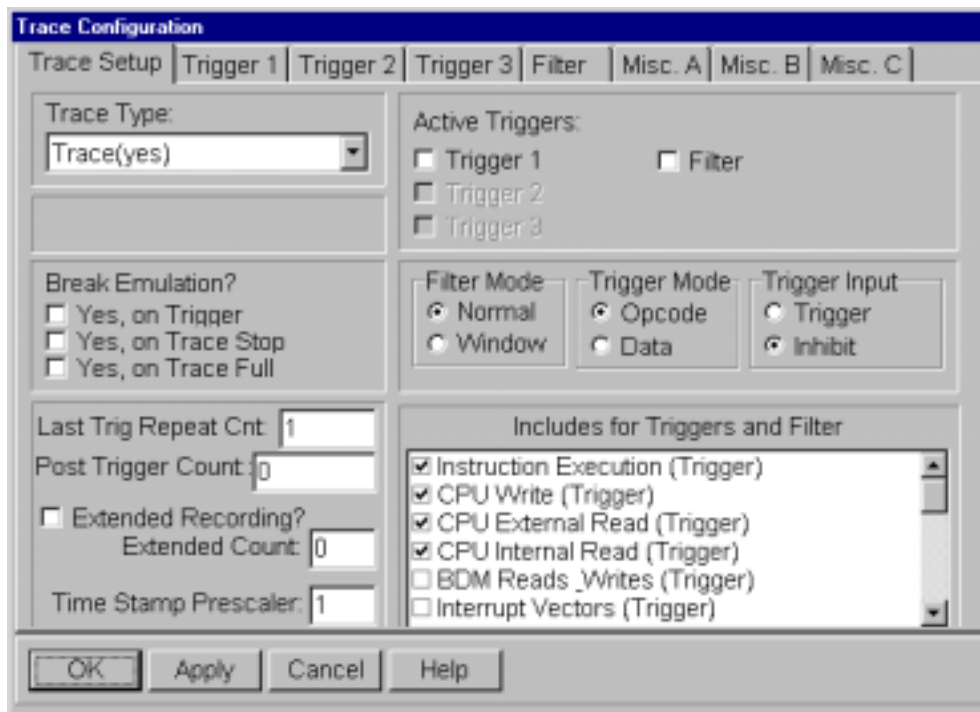
Break Emulation?

Defines whether a trigger will stop emulation or not. The triggers have the capability to determine what is recorded in the trace memory, stop the trace recording and display the trace contents and to stop the program execution. They do this according to the qualifiers entered in the specific Trigger and Filter tabs.

Last Trig Repeat Cnt:

Last Trigger Repeat Count. Specifies how many times the last active qualifier must become true before the trigger creates an event. The current value will show at the bottom of the trace window as “Trigger Count: “.

Figure 2
Trace
Configuration
Menu



Post Trigger Count:

This specifies how many frames are recorded in the trace memory after a trigger has occurred. The trigger point will be at or very close to the “0” frame. This is useful to determine not only what the state is at the trigger point, but also after the fact.

Extended Recording?

The Filter mechanism is used to define what information is recorded in the trace. Extended Recording adds a specified number (0 to 255) of additional instructions that are recorded after the filtered instruction. This is used to record some events that happen after the filter but not specified in the filter.

If 0 (zero) is used, the opcode filtered will be recorded plus the associated data bytes fetched (if any) plus all read and write cycles caused by this opcode. The appropriate filter Includes must be activated.

TimeStamp Prescaler

The trace timestamp is derived from a 25 MHz clock that also drives the housekeeping controller. This was done because the E clock is not suitable as it can be stretched which in turn will cause errors in the timestamp. This setting varies the prescaler of this clock before it is used by the trace logic. Setting it to “1” produces the greatest resolution.

If the timestamp is set to relative cycles, the value will be in terms of 25 MHz clocks which is 40 nsec. Because of skewing and that the value is an integer, the cycles can be rounded up or down by 1.

If the timestamp is set to relative time, the time can be out by synchronization delays. For example, a 125 nsec cycle can show up as 120 nsec. The time stamp resolution is 40 nsec and will always be updated even if the HC12 controller is in STOP or WAIT mode.

Active Triggers:

This window is used to enable or disable the specified triggers or the filter. Each one will be activated automatically when you enter qualifiers.

Filter Mode:

This specifies if Trigger 1 through 3 are used as regular sequential triggers for stopping emulation and/or the trace recording or for turning the trace on and off. Nohau has three methods of specifying or filtering what cycles are recorded in the trace. One is specified here. The second is under Includes for Triggers and Filters and is explained later. The third is the Filter tab and is also explained later. This Filter Mode is not the same as the Filter tab feature.

Normal

Normal mode specifies Trigger 1 through 3 are activated sequentially. Trigger 1 must be true before 2 can become true which must be true in order for 3 to become true as the appropriate qualifiers become true. Qualifiers under the Filter tab are still functional.

Window

Triggers 1 and 2 are used to start and stop trace recording. Trigger 1 is used to start the recording when it becomes true and a trigger 2 is used to stop the trace when it becomes true. This is often used to record the cycles occurring not only inside a specified function, but also those of any functions called by this function. In contrast, the Filter would record only those cycles executed within the specified function. Combinations with external events from the Trigger Input are legal. If this check box is active, Trigger 1 and 2 are identified as Start and Stop respectively. Qualifiers under the Filter tab are still functional.

Trigger Mode:

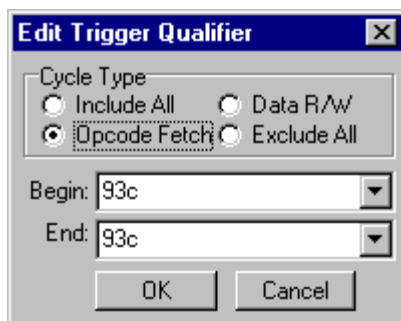
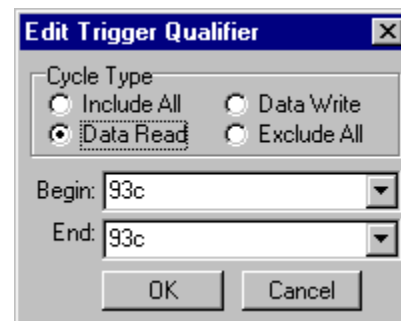
This specifies the options available in the dialog boxes that are available when qualifiers are entered in the trigger and filter menus. Select a Trigger or Filter tab and right click on the Address Cycle Type window and select ADD to see these menus that are shown in Figures 3 and 4.

Opcode

Options are Opcode Fetch, Data R/W, Exclude All, and Include All. See Figure 3.

Data

Options are Data Read, Data Write, Exclude All, and Include All. See Figure 4.

Figure 3**Figure 4****Includes for Triggers and Filter**

The entries in this box are types of bus cycles that are selected for the triggers or the filter. Items enabled that end with a (Trigger) are those that will be included in the trigger qualifiers. These items extend the capabilities of the trigger mechanism. If no trigger Includes are selected, no trigger can occur. Items that end with a (Filter) are those cycles that will be recorded in the trace memory. If no Filter Includes are selected, nothing will be recorded in the trace.

Trigger: *These cycles are included in the trigger qualifiers.*

Instruction Execution: Instructions that are actually executed. This is as opposed to those fetched, placed in the pipeline, then flushed and never executed.

CPU Write: Internal and external memory area data writes.

CPU External Read: Data reads to external memory.

CPU Internal Read: Data reads to internal memory.

BDM Reads & Writes: These cycles are reads and writes via the BDM port. The EMUL12 is a full feature emulator but will use the BDM port when applicable. Cycles include reads to be displayed in the run time data window and BDM port data writes to internal and external memory areas during the emulation time.

Interrupt Vectors: When an interrupt vector is used, this event can be used as a trigger.

Fetch Cycles: This is all opcode and data cycles going into the pipeline, whether they were executed or not.

Free Cycles: These are cycles that look like regular cycles on the data and address buses, but are actually null cycles when the CPU performs an internal operation such as increment the A register.

Filter: These cycles are recorded in the trace memory and are similar to the trigger with 5 more.

Instruction Execution: Instructions that are actually executed. This is as opposed to those fetched, placed in the pipeline, then flushed and never executed.

CPU Write: Internal and external memory area data writes.

CPU External Read: Data reads to external memory.

CPU Internal Read: Data reads to internal memory.

BDM Reads & Writes: These cycles are reads and writes via the BDM port. The EMUL12 is a full feature emulator but will use the BDM port when applicable. Cycles include reads to be displayed in the run time data window and BDM port data writes to internal and external memory areas during the emulation time.

- Interrupt Vectors:** When an interrupt vector is used, this event will be recorded.
- Fetch Cycles:** This is all opcode and data cycles going into the pipeline, whether they were executed or not.
- Free Cycles:** These are cycles that look like regular cycles on the data and address buses, but are actually null cycles when the CPU performs an internal operation such as increment the A register.

WAIT Power Down State Whenever the CPU enters the WAIT state, a frame is recorded in the trace. The WAIT LED also illuminates.

STOP Power Down State Whenever the CPU enters the STOP state, a frame is recorded in the trace. The STOP LED also illuminates.

RESET State The event that caused the CPU to enter the reset state is recorded in the trace. The RESET LED also illuminates.

RESET Transition State The emulator can configure the HC12 controller during the startup time. The Reset Transition State checkbox must be activated in the Config. Emulator, in the Miscellaneous field. This time is the first 1000 cycles after reset. The Nohau monitor code is running at this time. When the Reset Transition is active, this fact will be recorded in the trace memory.

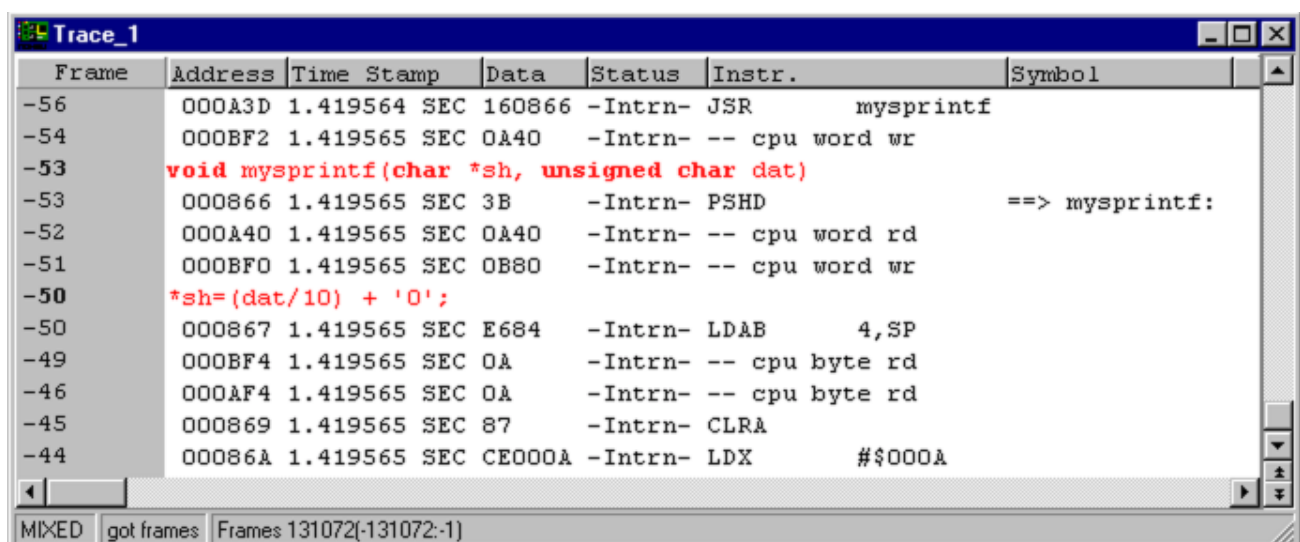
Error State A fatal error has occurred. The emulator shuts down and the ERROR LED illuminates. This is usually from an illegal write the PEAR or MODE registers.

Trace Window Display

Load the HC12time.c program and run the emulator for a few seconds and then stop it. Open the trace window and you will get a window similar to Figure 5. The fields displayed in the trace window can be selected depending on your needs. Right click on the trace window to display the available options. The meaning of these fields is explained in the on-line Help files. The relative timestamp for a given instruction is actually on the next line below. The time is from the start of one instruction to the start of the next.

The fields displayed in the trace window can be selected depending on your needs. Figure 6 shows the trace local window. The meaning of these fields is explained in the on-line Help files.

Note the source code is available in the trace window. It can be activated in the local menu under Display Mode, Trace and Source.



Frame	Address	Time Stamp	Data	Status	Instr.	Symbol
-56	000A3D	1.419564 SEC	160866	-Intrn-	JSR	mysprintf
-54	000BF2	1.419565 SEC	0A40	-Intrn-	-- cpu word wr	
-53						void mysprintf(char *sh, unsigned char dat)
-53	000866	1.419565 SEC	3B	-Intrn-	PSHD	==> mysprintf:
-52	000A40	1.419565 SEC	0A40	-Intrn-	-- cpu word rd	
-51	000BF0	1.419565 SEC	0B80	-Intrn-	-- cpu word wr	
-50						*sh=(dat/10) + '0';
-50	000867	1.419565 SEC	E684	-Intrn-	LDAB	4,SP
-49	000BF4	1.419565 SEC	0A	-Intrn-	-- cpu byte rd	
-46	000AF4	1.419565 SEC	0A	-Intrn-	-- cpu byte rd	
-45	000869	1.419565 SEC	87	-Intrn-	CLRA	
-44	00086A	1.419565 SEC	CE000A	-Intrn-	LDX	#\$000A

MIXED got frames Frames 131072(-131072:-1)

Figure 5

The trace recording can be manually turned off and on with the trace icon on the main menu. When the trace is not recording, the Shadow RAM continues providing a realtime view of the system memory.

Try activating certain features in Figure 6 to see the effects on the trace window.

The trace memory is where the frames are stored. The triggers have the power to stop the trace recording, and/or emulation and to control what is recorded in the trace memory (filtering).

Figure 6
Trace Window
Local Menu



Trace Examples

All Nohau emulators contain sophisticated and versatile triggers. The EMUL12 offers exceptional trigger capabilities. You can configure the triggers and view the trace contents in real-time. This example will show how to setup a simple yet effective trigger.

Triggers can be set on addresses and data ranges. They control trace recording or cause the emulator to stop the target depending on the options set. Trace and Triggering can trigger and record all internal and external accesses.

Trigger Example on an Address and Data Qualifier

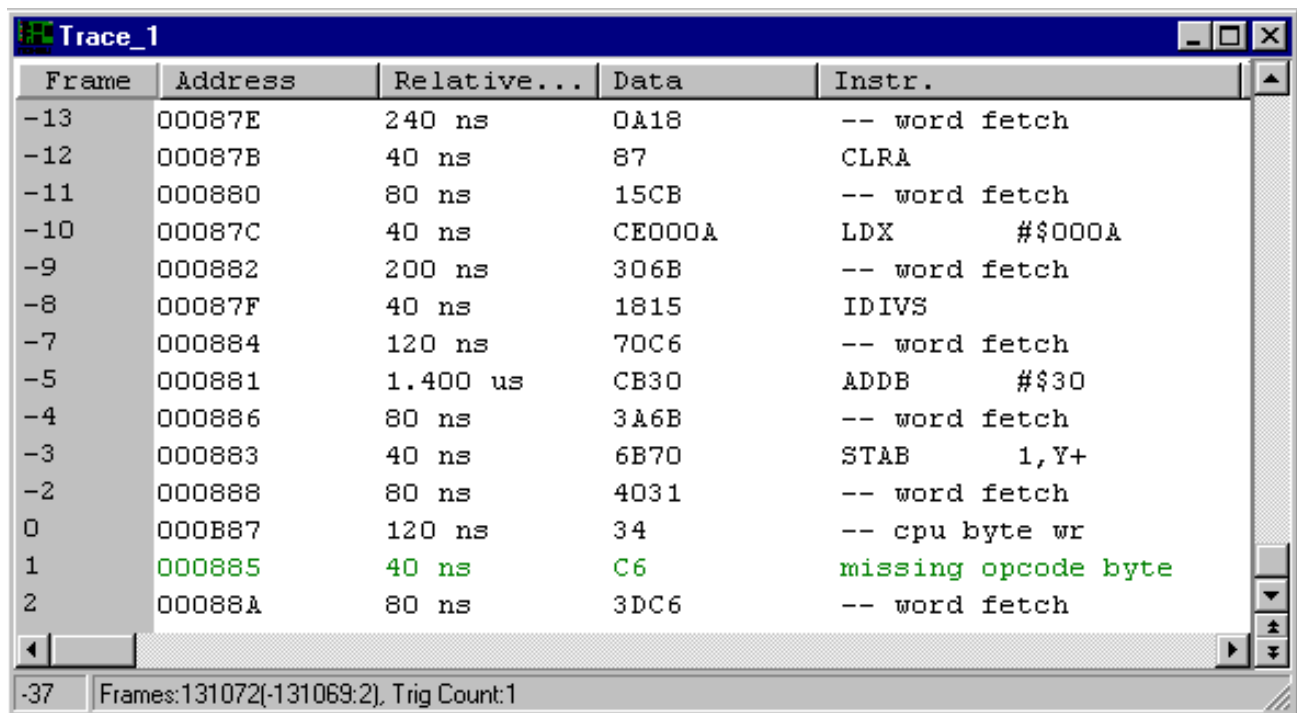
Triggers are the most powerful facility possessed by an emulator. This example will stop the emulator when the seconds field in the clock equals 4. This is represented by a byte write to memory location B87 with the ASCII value of 34.

- 1) Setup the emulator to run the timer example as described in Chapter 3. Make sure the data window is set to view ASCII data as in Figure 3 in Chapter 3. This is not needed for the trigger to work but for your inspection of the B87 memory location. Open the trace window and position the windows so the Source, Data and Trace windows are visible.
- 2) Open the menu Config, Trace from the main window. Figure 2 appears. This is where the triggers are configured. Note the tabs at the top showing the three Triggers and the Filter as well as the Trace Setup plus the Misc. inputs.
- 3) Activate Yes, on Trigger on the Break Emulation? box. When a trigger occurs, the emulation will be stopped.
- 4) Click on the Trigger 1 tab. A blank Figure 7 opens up. The values of the qualifiers are inserted here.
- 5) Right click on the Address Cycle Type window and select Add. The Edit Trigger Qualifier dialog box in Figure 3 or 4 opens up depending on the setting of the Trigger Mode under the Trace Setup tab shown in Figure 2 opens up. Fill in the fields as shown. Make sure you select Data R/W. Press OK.
- 6) Repeat for the Data Trigger Type and enter the value of 34 in each field. Click OK. You can put in 50 qualifiers in each field. The addresses are OR'd together then ANDed with the OR'd data qualifiers.

Figure 7
Trigger 1
Configuration

Trace Configuration		
Trigger 1		
Address Cycle Type	Start Address	End Address
Data RW	b87	b87
< AND >		
Data Trigger Type	Low Value	High Value
Pattern	34	34
<input checked="" type="checkbox"/> Enabled Data Mask: 00FF		
<input type="button" value="OK"/> <input type="button" value="Apply"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>		

- 7) In the Trace Configuration menu as shown in Figure 7, enter 00FF in the Data Mask field.
- 8) Click on OK to enter the data.
- 9) If the emulation and trace were running during this time, temporarily stop and start the trace to register the trigger data. Note that emulation will not be slowed or stopped during trigger setting or trace stopping. In order to activate the break, you will have to stop the emulation momentarily.
- 10) Start emulation if not already running and when the seconds indicator reaches the value of 4, emulation will stop. The GO and TR icons will turn green and the words Stopped will appear twice in the lower left corner of the main menu.
- 11) You will get a trace window similar to Figure 8 shown on the next page.



Frame	Address	Relative...	Data	Instr.
-13	00087E	240 ns	0A18	-- word fetch
-12	00087B	40 ns	87	CLRA
-11	000880	80 ns	15CB	-- word fetch
-10	00087C	40 ns	CE000A	LDX #\$000A
-9	000882	200 ns	306B	-- word fetch
-8	00087F	40 ns	1815	IDIVS
-7	000884	120 ns	70C6	-- word fetch
-5	000881	1.400 us	CB30	ADDB #\$30
-4	000886	80 ns	3A6B	-- word fetch
-3	000883	40 ns	6B70	STAB 1,Y+
-2	000888	80 ns	4031	-- word fetch
0	000B87	120 ns	34	-- cpu byte wr
1	000885	40 ns	C6	missing opcode byte
2	00088A	80 ns	3DC6	-- word fetch

-37 Frames:131072[-131069:2], Trig Count:1

Figure 8

- 12) Note the trigger point at Frame 0. This is the byte cycle that caused the write of 34 to address B87.
- 13) The instruction that caused this write was the STAB executed at Frame -3.
- 14) This STAB was fetched to the pipeline at Frame -9 (the 6B) and Frame -7 (the 70).
- 15) Frames -2 and 2 are flushed instructions. Frame 1 represents an opcode execution but is incomplete because of the emulation halt and the fact that the instructions are on odd addresses. This instruction will be executed once emulation is started again and properly recorded in the trace memory. Note that fetches on the odd addresses are correctly displayed in the trace.
- 16) Change the filter includes to see their effect on the trace window. You must rerun the program to refresh the trace.

Trace Filter Example

This example uses the Filter tab to record only certain frames in the trace memory. We will record only byte writes of 34 to address B87 in the trace memory.

- 1) Open the Trace Configuration menu in the Config menu item. Figure 2 will open.

Frame	Address	Relative time	Data	Instr.
-168	000B87	51.480 us	34	-- cpu byte wr
-166	000B87	51.480 us	34	-- cpu byte wr
-164	000B87	51.480 us	34	-- cpu byte wr
-162	000B87	51.480 us	34	-- cpu byte wr
-160	000B87	51.480 us	34	-- cpu byte wr
-158	000B87	51.480 us	34	-- cpu byte wr
-156	000B87	51.480 us	34	-- cpu byte wr
-154	000B87	51.480 us	34	-- cpu byte wr
-152	000B87	51.480 us	34	-- cpu byte wr
-150	000B87	51.480 us	34	-- cpu byte wr
-148	000B87	51.480 us	34	-- cpu byte wr
-146	000B87	51.520 us	34	-- cpu byte wr
-144	000B87	51.480 us	34	-- cpu byte wr
-142	000B87	51.480 us	34	-- cpu byte wr
-140	000B87	51.480 us	34	-- cpu byte wr
-138	000B87	51.480 us	34	-- cpu byte wr
-136	000B87	51.480 us	34	-- cpu byte wr
-134	000B87	51.480 us	34	-- cpu byte wr
-132	000B87	51.480 us	34	-- cpu byte wr
-130	000B87	51.480 us	34	-- cpu byte wr
-128	000B87	51.480 us	34	-- cpu byte wr
-126	000B87	51.480 us	34	-- cpu byte wr
-124	000B87	51.480 us	34	-- cpu byte wr
-122	000B87	51.480 us	34	-- cpu byte wr
-120	000B87	51.480 us	34	-- cpu byte wr
-118	000B87	51.480 us	34	-- cpu byte wr
-116	000B87	51.480 us	34	-- cpu byte wr
-114	000B87	51.480 us	34	-- cpu byte wr
-112	000B87	51.480 us	34	-- cpu byte wr
-110	000B87	51.480 us	34	-- cpu byte wr
-108	000B87	51.480 us	34	-- cpu byte wr
-106	000B87	51.480 us	34	-- cpu byte wr
-104	000B87	51.480 us	34	-- cpu byte wr
-102	000B87	51.480 us	34	-- cpu byte wr
-100	000B87	51.480 us	34	-- cpu byte wr
-98	000B87	51.480 us	34	-- cpu byte wr
-96	000B87	51.480 us	34	-- cpu byte wr
-94	000B87	51.480 us	34	-- cpu byte wr
-92	000B87	51.480 us	34	-- cpu byte wr
-90	000B87	51.480 us	34	-- cpu byte wr
-88	000B87	51.480 us	34	-- cpu byte wr
-86	000B87	51.480 us	34	-- cpu byte wr
-84	000B87	51.480 us	34	-- cpu byte wr
-82	000B87	51.480 us	34	-- cpu byte wr
-80	000B87	51.480 us	34	-- cpu byte wr
-78	000B87	51.480 us	34	-- cpu byte wr
-76	000B87	51.480 us	34	-- cpu byte wr
-74	000B87	51.480 us	34	-- cpu byte wr
-72	000B87	51.480 us	34	-- cpu byte wr
-70	000B87	51.480 us	34	-- cpu byte wr
-68	000B87	51.480 us	34	-- cpu byte wr
-66	000B87	51.480 us	34	-- cpu byte wr
-64	000B87	51.480 us	34	-- cpu byte wr
-62	000B87	51.480 us	34	-- cpu byte wr
-60	000B87	51.480 us	34	-- cpu byte wr
-58	000B87	51.480 us	34	-- cpu byte wr
-56	000B87	51.480 us	34	-- cpu byte wr
-54	000B87	51.480 us	34	-- cpu byte wr
-52	000B87	51.480 us	34	-- cpu byte wr
-50	000B87	51.480 us	34	-- cpu byte wr
-48	000B87	51.480 us	34	-- cpu byte wr
-46	000B87	51.480 us	34	-- cpu byte wr
-44	000B87	51.480 us	34	-- cpu byte wr
-42	000B87	51.480 us	34	-- cpu byte wr
-40	000B87	51.480 us	34	-- cpu byte wr
-38	000B87	51.480 us	34	-- cpu byte wr
-36	000B87	51.480 us	34	-- cpu byte wr
-34	000B87	51.480 us	34	-- cpu byte wr
-32	000B87	51.480 us	34	-- cpu byte wr
-30	000B87	51.480 us	34	-- cpu byte wr
-28	000B87	51.480 us	34	-- cpu byte wr
-26	000B87	51.480 us	34	-- cpu byte wr
-24	000B87	51.480 us	34	-- cpu byte wr
-22	000B87	51.480 us	34	-- cpu byte wr
-20	000B87	51.480 us	34	-- cpu byte wr
-18	000B87	51.480 us	34	-- cpu byte wr
-16	000B87	51.480 us	34	-- cpu byte wr
-14	000B87	51.480 us	34	-- cpu byte wr
-12	000B87	51.480 us	34	-- cpu byte wr
-10	000B87	51.480 us	34	-- cpu byte wr
-8	000B87	51.480 us	34	-- cpu byte wr
-6	000B87	51.480 us	34	-- cpu byte wr
-4	000B87	51.480 us	34	-- cpu byte wr
-2	000B87	51.480 us	34	-- cpu byte wr
0	000B87	51.480 us	34	-- cpu byte wr
2	000B87	51.480 us	34	-- cpu byte wr
4	000B87	51.480 us	34	-- cpu byte wr
6	000B87	51.480 us	34	-- cpu byte wr
8	000B87	51.480 us	34	-- cpu byte wr
10	000B87	51.480 us	34	-- cpu byte wr
12	000B87	51.480 us	34	-- cpu byte wr
14	000B87	51.480 us	34	-- cpu byte wr
16	000B87	51.480 us	34	-- cpu byte wr
18	000B87	51.480 us	34	-- cpu byte wr
20	000B87	51.480 us	34	-- cpu byte wr
22	000B87	51.480 us	34	-- cpu byte wr
24	000B87	51.480 us	34	-- cpu byte wr
26	000B87	51.480 us	34	-- cpu byte wr
28	000B87	51.480 us	34	-- cpu byte wr
30	000B87	51.480 us	34	-- cpu byte wr
32	000B87	51.480 us	34	-- cpu byte wr
34	000B87	51.480 us	34	-- cpu byte wr
36	000B87	51.480 us	34	-- cpu byte wr
38	000B87	51.480 us	34	-- cpu byte wr
40	000B87	51.480 us	34	-- cpu byte wr
42	000B87	51.480 us	34	-- cpu byte wr
44	000B87	51.480 us	34	-- cpu byte wr
46	000B87	51.480 us	34	-- cpu byte wr
48	000B87	51.480 us	34	-- cpu byte wr
50	000B87	51.480 us	34	-- cpu byte wr
52	000B87	51.480 us	34	-- cpu byte wr
54	000B87	51.480 us	34	-- cpu byte wr
56	000B87	51.480 us	34	-- cpu byte wr
58	000B87	51.480 us	34	-- cpu byte wr
60	000B87	51.480 us	34	-- cpu byte wr
62	000B87	51.480 us	34	-- cpu byte wr
64	000B87	51.480 us	34	-- cpu byte wr
66	000B87	51.480 us	34	-- cpu byte wr
68	000B87	51.480 us	34	-- cpu byte wr
70	000B87	51.480 us	34	-- cpu byte wr
72	000B87	51.480 us	34	-- cpu byte wr
74	000B87	51.480 us	34	-- cpu byte wr
76	000B87	51.480 us	34	-- cpu byte wr
78	000B87	51.480 us	34	-- cpu byte wr
80	000B87	51.480 us	34	-- cpu byte wr
82	000B87	51.480 us	34	-- cpu byte wr
84	000B87	51.480 us	34	-- cpu byte wr
86	000B87	51.480 us	34	-- cpu byte wr
88	000B87	51.480 us	34	-- cpu byte wr
90	000B87	51.480 us	34	-- cpu byte wr
92	000B87	51.480 us	34	-- cpu byte wr
94	000B87	51.480 us	34	-- cpu byte wr
96	000B87	51.480 us	34	-- cpu byte wr
98	000B87	51.480 us	34	-- cpu byte wr
100	000B87	51.480 us	34	-- cpu byte wr
102	000B87	51.480 us	34	-- cpu byte wr
104	000B87	51.480 us	34	-- cpu byte wr
106	000B87	51.480 us	34	-- cpu byte wr
108	000B87	51.480 us	34	-- cpu byte wr
110	000B87	51.480 us	34	-- cpu byte wr
112	000B87	51.480 us	34	-- cpu byte wr
114	000B87	51.480 us	34	-- cpu byte wr
116	000B87	51.480 us	34	-- cpu byte wr
118	000B87	51.480 us	34	-- cpu byte wr
120	000B87	51.480 us	34	-- cpu byte wr
122	000B87	51.480 us	34	-- cpu byte wr
124	000B87	51.480 us	34	-- cpu byte wr
126	000B87	51.480 us	34	-- cpu byte wr
128	000B87	51.480 us	34	-- cpu byte wr
130	000B87	51.480 us	34	-- cpu byte wr
132	000B87	51.480 us	34	-- cpu byte wr
134	000B87	51.480 us	34	-- cpu byte wr
136	000B87	51.480 us	34	-- cpu byte wr
138	000B87	51.480 us	34	-- cpu byte wr
140	000B87	51.480 us	34	-- cpu byte wr
142	000B87	51.480 us	34	-- cpu byte wr
144	000B87	51.480 us	34	-- cpu byte wr
146	000B87	51.480 us	34	-- cpu byte wr
148	000B87	51.480 us	34	-- cpu byte wr
150	000B87	51.480 us	34	-- cpu byte wr
152	000B87	51.480 us	34	-- cpu byte wr
154	000B87	51.480 us	34	-- cpu byte wr
156	000B87	51.480 us	34	-- cpu byte wr
158	000B87	51.480 us	34	-- cpu byte wr
160	000B87	51.480 us	34	-- cpu byte wr
162	000B87	51.480 us	34	-- cpu byte wr
164	000B87	51.480 us	34	-- cpu byte wr
166	000B87	51.480 us	34	-- cpu byte wr
168	000B87	51.480 us	34	-- cpu byte wr

Figure 9

- 2) Deselect the Trigger 1 checkbox. This disables the settings we placed in Trigger 1.
- 3) Deselect the Break on Emulation? checkbox.
- 4) Click on the Filter tab.
- 5) Enter the data in the same fashion as in Figure 3. Except this time in the Filter tab menu.
- 6) Set the Data Mask to 00FF as before.
- 7) Click on OK to enter the data.
- 8) Start the emulation as before and let it run for a few seconds and stop emulation.
- 9) Figure 9 will be displayed. Note that only byte writes of 34 to location 5017 are recorded.

Note the timestamp. It shows that each write was 51 usec apart. There are many other combinations of triggers and filters you can use. In each of the Address Cycle and Data Trigger Type fields you can enter 50 qualifiers.

This filter mode has been the Normal mode. This mode records qualifiers within a range of addresses OR data values. If you set this to record a function, any functions called by this function will not be recorded.

With the Window mode, Trigger 1 is used to start the trace recording and Trigger 2 is used to stop the recording. Any called functions from within a specified function will be recorded in this case. In Figure 2, click on the Window check box under Filter Mode and the triggers will turn to start and stop types.

Conclusions

This explains a portion of the power of Seehau and Nohau HC12 family emulators. We have not explored the Performance Analysis, RTOS support and editing abilities of Seehau.

For more information, contact your local Nohau representative or Nohau at www.nohau.com.