



EMUL-S12D-PC

Getting Started Guide

for the Freescale DP256

Version 2.6
(For use with SeeHau Version 3.1101A and later)

Contents

Chapter 1: Nohau EMUL-S12D-PC: The Hardware Parts	3
Introduction - The S12D Emulator for the DP256 Family	3
The Nohau HC12 “B” and “D” Emulator Families	3
The Trace Board and the Seehau Debugger	4
Nohau HC12 Family Emulator Hierarchy	5
Line Drawings of the Motherboard and CPU Module Jumper Settings	8
CPU Module CPU-MCS912DP256-xx Jumper Settings	10
The Nohau PLL Clock Source and How It Works	11
Clock Jumpers	11
Clock Jumpers to CPU Module Map	12
Voltage Jumpers	13
Voltage Jumpers to CPU Module Map	14
Getting the PLL to Work	15
Chapter 2: The Software Parts	20
The Nohau User Interface Seehau	20
Seehau: A General Introduction	20
Connecting the Emulator to your PC	20
Software Installation	20
Directory Structure	21
Configuring the emulator software Seehau.....	21
Important Software and Hardware Notes:	22
Starting the Emulator and Seehau	23
Problems ?	24
Shutting down Seehau	25
Selecting the Startup Macro	25
Chapter 3: The Example program and Data Display	26
The Example Program and Data Display:	26
Loading the example program BarTime.695	26
Viewing Data in Real-time with the Shadow RAM	27
Graphical Display of Data: Gauge	28
Graphical Display of Data: Graph	28
Data Hints	29
Watch Window	29
Inspect/Watch Window	29
Setting Hardware and Software Breakpoints	31

Chapter 4: Commands and Macros	33
Seehau Commands and Macros	33
Macro Construction	33
Macro Execution	33
Command Format	34
Saving Macros	34
Quick Saving a Configuration Menu using the Apply Button	34
Creating New Buttons	35
Sax Basic Interpreter	36
Chapter 5: The Trace and Triggers	38
Trace and Triggers	38
Trace and Trigger Overview	38
Trace Memory and Trigger Mechanism Board	38
Trace Board Jumpers	38
Trace Definitions	40
Trace Configuration Menu	40
Includes for Triggers and Filter	42
Trace Window Display	43
Trace Examples	44
Trigger Example on an Address and Data Qualifier	44
Trace Filter Example	47
COP Watchdog and RESET Sequences	47
OSEK RTOS Support	47
Chapter 6: Modifying the Register Window	48
Modifying INITRM:	48
Chapter 7: The Banking Example	49
Banking and the Freescale Next Generation Microcontrollers:	49
The Banking Example Program	49
Running the Banking Example	49
Showing Banking Pages in the Trace Window	51
Setting a Bank Aware Trigger	52

Chapter 1: Nohau EMUL-S12D-PC: The Hardware Parts

Introduction - The S12D Emulator for the DP256 Family

The EMUL-S12D-PC “S12D” full emulator supports the HCS12 MC9S12DP256 and other family members as well as the H256 family and is shown in Figure 1. Figure 2 shows the emulator with the Nohau Flex Cable adapter system connected to the Freescale DP256 evaluation board. The DP256 chip supports the rest of the DP256 family since it is the superset part. Another family chip such as the DJ can be installed on the personality module.

This emulator provides CMOS input and output levels for the IO pins. The Nohau HC12 emulator is a full feature emulator and is not a BDM emulator although Nohau provides one. It does not use the two HC12 hardware breakpoints as it has its own internal hardware breakpoint system. The Nohau hardware breakpoints are unlimited in number and no-skid. Nohau emulators are Made in the USA.

The HCS12 parts are specified to 25 MHz and the current Nohau emulator body runs to a maximum of 16 or 24 MHz bus speed depending on the model. The trace and personality modules are available in 25 MHz versions. Nearly all customers use a clock frequency of 24 MHz as it provides easily divisible frequencies for internal modules such as serial and CAN communications. Nohau will be redesigning the current 24 MHz emulator into a full 25 MHz version for those customers using the full 25 MHz clock. Contact Nohau USA for the status of the full 25 MHz emulator.

The basic Nohau HC12 emulator consists of an emulator motherboard, a CPU personality board, power supply, debugger software (Seehau) and a communications cable. You can run this system stand-alone without any target hardware. Add a target adapter and you can run in your target board. Add an optional trace card and you can trigger and record CPU instructions and their bus operations.

Figure 4 is the Nohau roadmap showing support for the various Freescale HC12 family members. For devices not listed, contact sales@icetech.com or your local rep. Nohau will support future HCS12 derivatives.

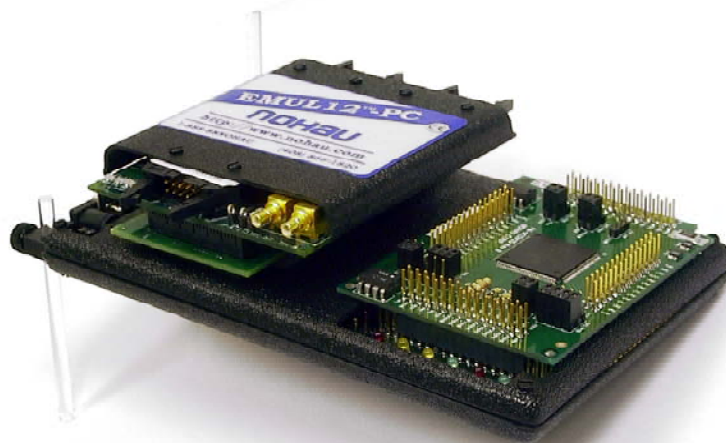
The Nohau HC12 “B” and “D” Emulator Families

Nohau makes three full emulator pods or motherboards. They are the “B”, “D” and “S12D” models. See the Nohau HC12 price list for more information on these emulator. Nohau also has a BDM emulator.

The EMUL12-PC “D” full emulator POD board supports the 68HC912D60, DA128, DG128, DT60A, DT128A, DA128A and DG128A microcontrollers. The part number is EMUL12-D/512-16. The “D” emulator looks similar to Figure 1. Support for the parts listed above is accomplished by changing the small economical daughtercard and selecting the appropriate controller in the software configuration menu.

The EMUL12-PC “B” full emulator POD board supports the 68HC912B32, BC32, BD32 and BE32 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.

Figure 1
Emulator “S12D”
POD Board



The Trace Board and the Seehau Debugger

The trace is optional and can be added later. The trace provides 128 K frames and can be configured for your needs. It is available in speeds (bus) of 12, 16 and 25 MHz. The triggers can be configured and the trace buffer can be viewed without stopping the emulation of the user program. Seehau, the Nohau 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. Shadow RAM permits viewing of data in various numeric or graphical formats in real-time without stealing CPU cycles. The trace card is shown in Figure 3. Figures 5 and 6 show the emulator pod board with no covers. Figures 7 and 8 are line drawings of the emulator and the CPU personality module.

This document as well as a hardware manual is available as a pdf file on the website: www.icetech.com
Application Notes are also available on the web.

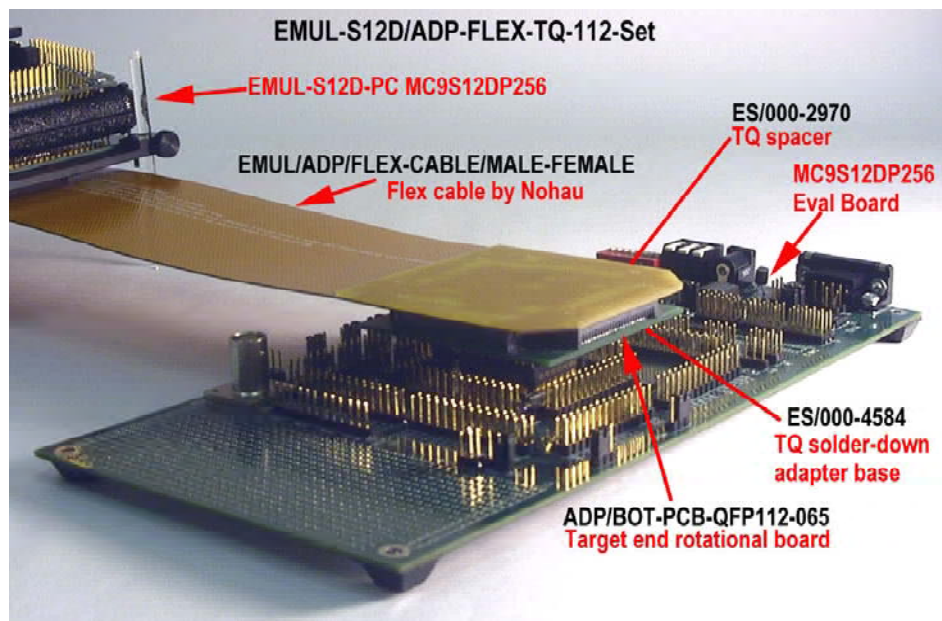
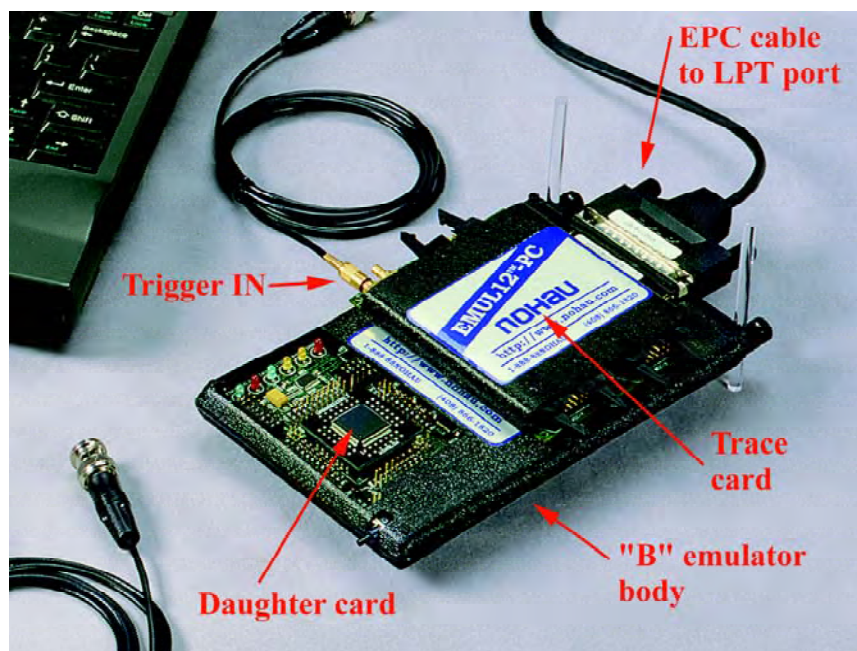


Figure 2
Emulator connected to the Freescale DP256 Eval Board

Figure 3
Emulator "B"
POD Board



Nohau HC12 Family Emulator Hierarchy

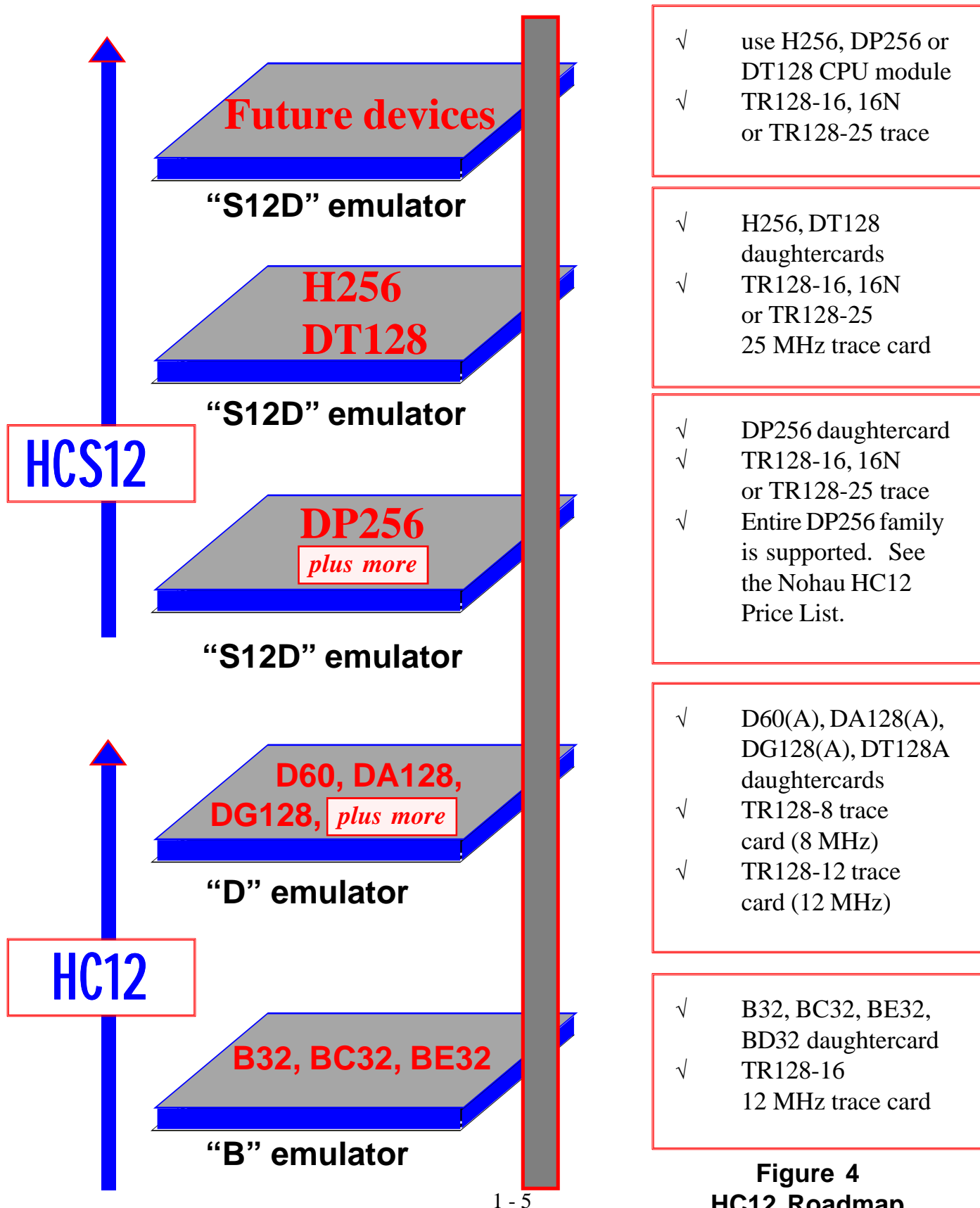


Figure 4
HC12 Roadmap

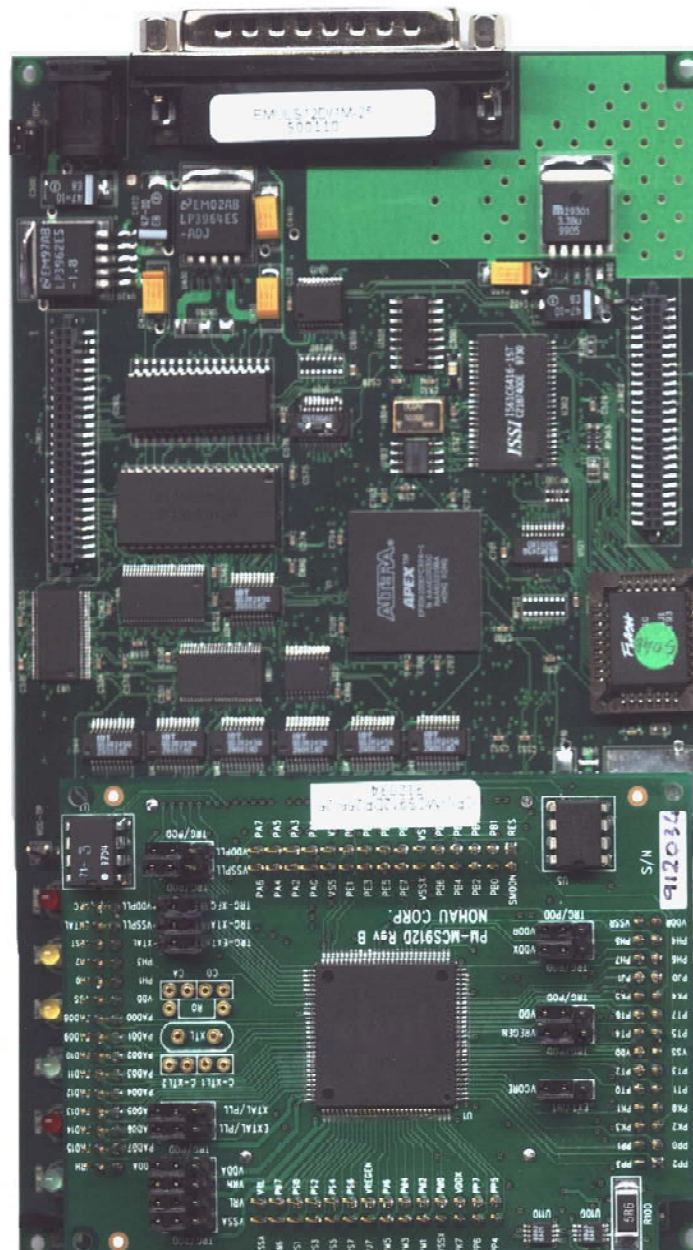


Figure 5
Emulator POD Board: Top View

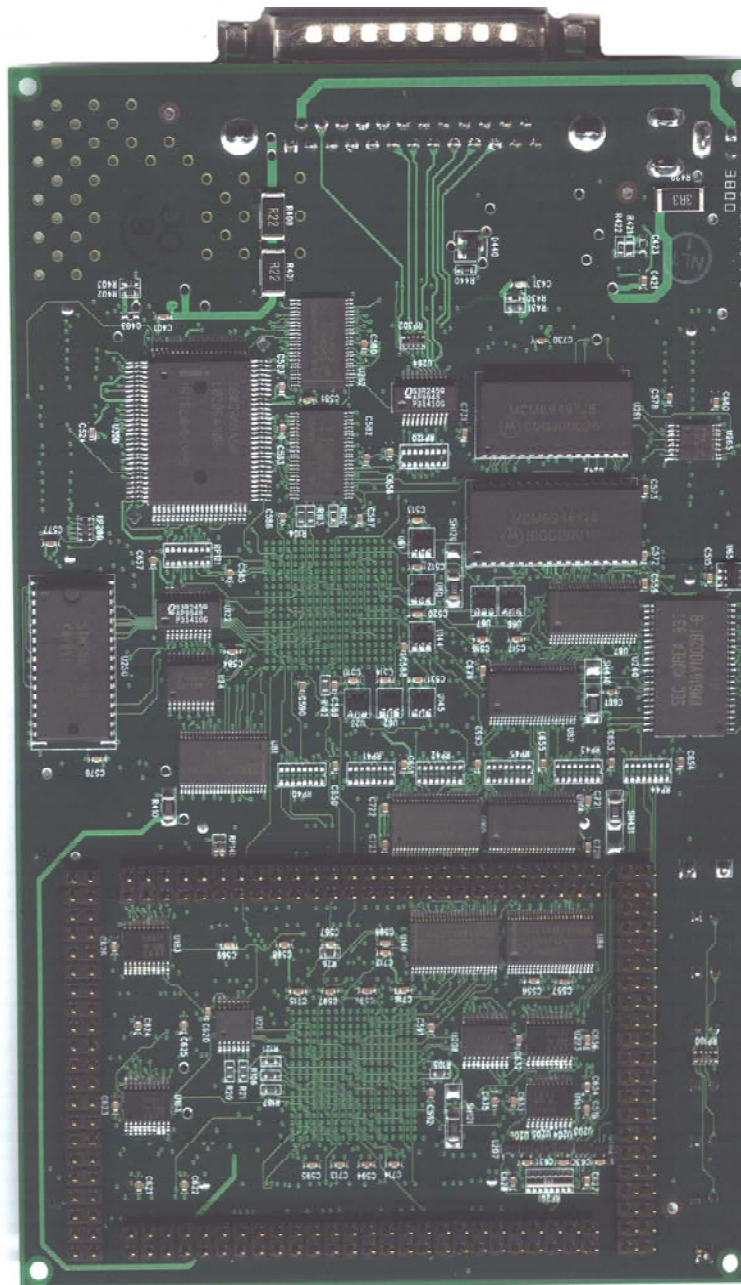


Figure 6
Emulator POD Board: Bottom

Line Drawings of the Motherboard and CPU Module Jumper Settings

Figure 7 shows the top view of the “S12D” emulation pod. Figure 8 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 section *CPU Module CPU-MCS912DP256-xx Jumper Settings* in this chapter for the meaning of the jumpers.

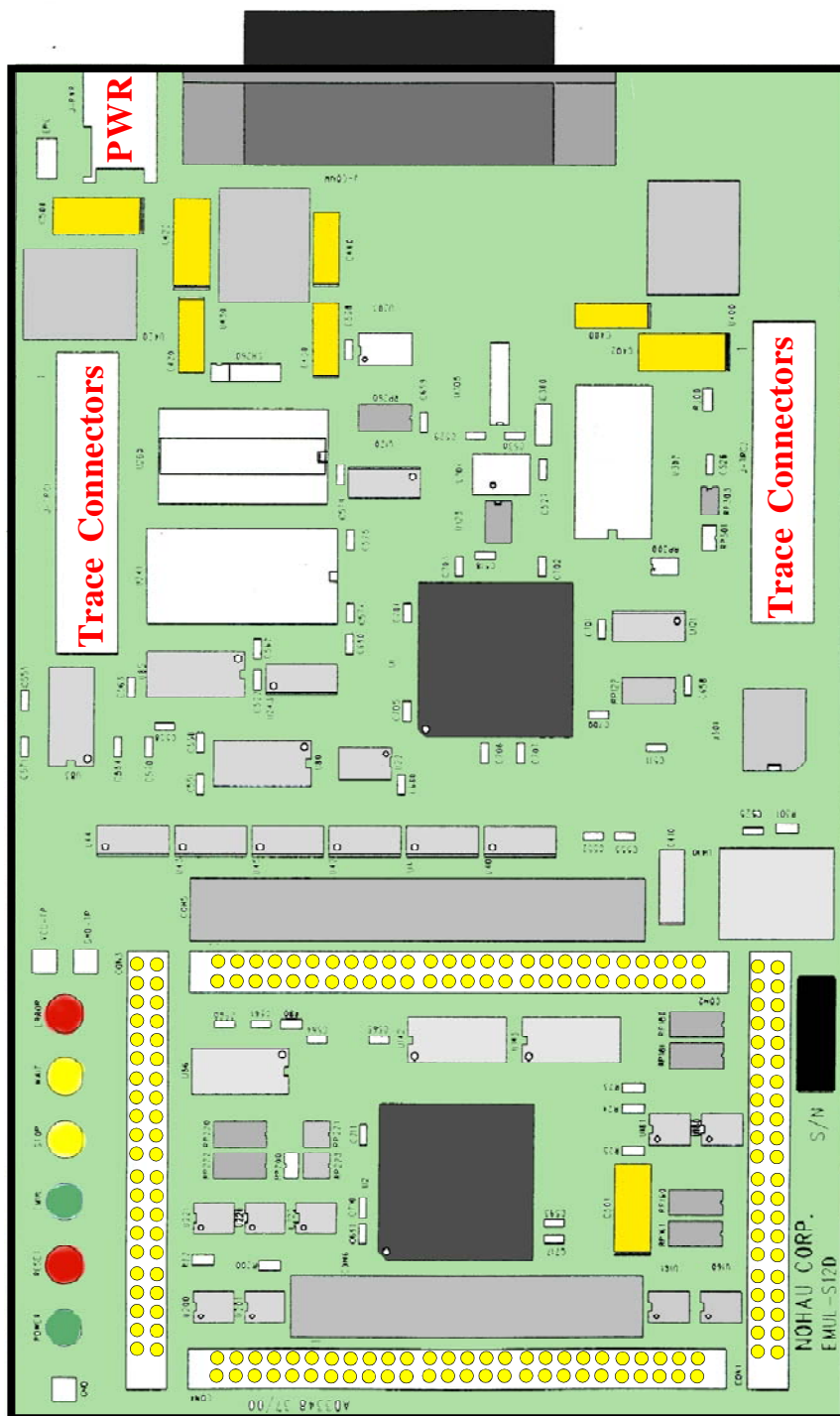


Figure 7
Line Drawing of the Emulator POD Board: Top View

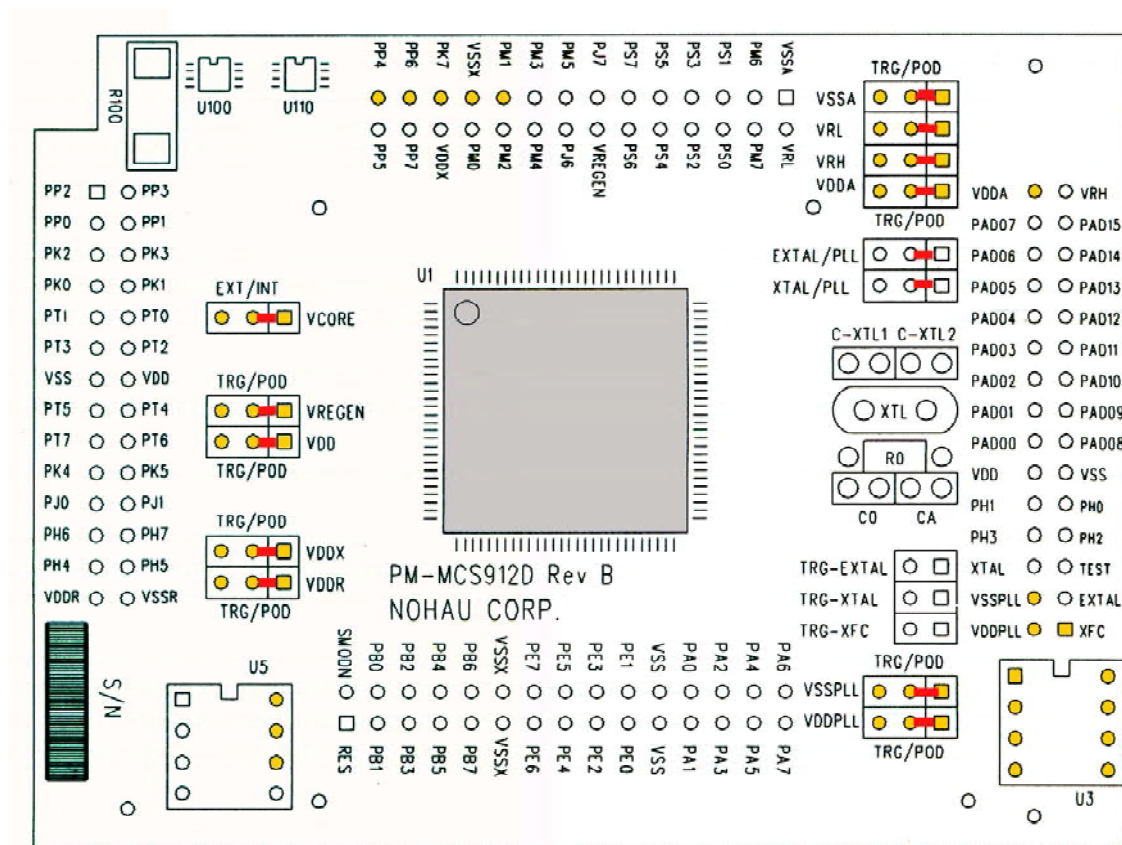


Figure 8
Line Drawing of the Emulator CPU Board: Expanded View

CPU Module CPU-MCS912DP256-xx Jumper Settings

This document provides jumper information on the DP256 personality card for the Nohau EMULS12D full emulator. This emulator supports all Motorola Next Generation processors and this personality card supports all MC9S12D parts including the MC9S12DP256. The part number for the module is CPU-MCS912DP256-16 or -25. The circuit diagrams enclosed are Copyright © ICE Technology.

- 1) The emulator will work stand-alone (no target connected) with the default settings.
- 2) These jumpers are named where possible to conform to the Motorola data sheet.
- 3) The jumpers are sorted into separate clock and voltage sections in this note.
- 4) "PLL" as in EXTAL/PLL denotes the Nohau internal PLL clock source and not the Motorola on-chip PLL clock. The Nohau PLL supplies a 2.5 VPP signal and the frequency is set in the Seehau software under the Config, Emulator window. By default the Nohau PLL source is used to supply EXTAL without the Motorola PLL activated and the resulting E clock or bus speed frequency will be 0.5 that is entered into the Seehau software.
- 5) The jumpers have a pin 1 which is indicated on the board silkscreen as a white line as indicated by the arrow on this sample. Pin 1 is also shown on the schematics. Jumpers can be set by the numbers or with the setting indicated on the silkscreen i.e. XTL/PLL or TRG/POD.



- 6) TRG means Target and the signal will connect to the target system if set to TRG or to the emulator hardware if set to the POD position. The POD position is normally jumper pins 1 and 2. The factory set default position is shown by a black bar such as seen on the jumper here.

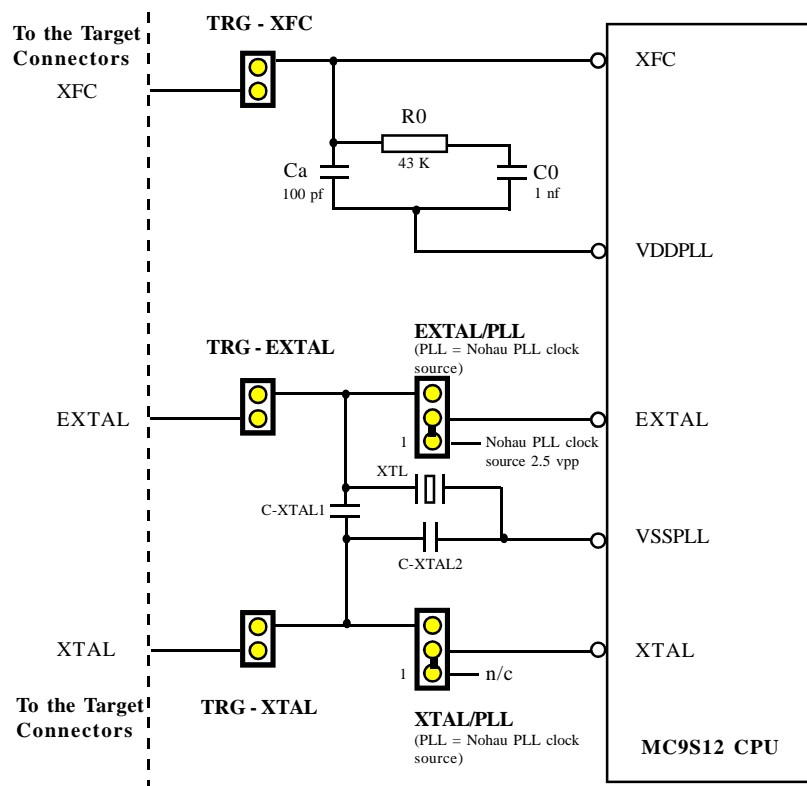


- 7) The gold header pins accessible on the top of this personality module connect directly to the target connectors and are labelled. These connectors do not necessarily connect directly to the appropriate pin on the DP256 controller. Recreated port pins will go through emulator circuitry and hence are not directly connected. The recreated ports are A, B, E and K and some additional control signals and are not directly connected. Peripheral modules on other ports are generally directly connected. Modules such as CAN, SPI and A/D are not affected or controlled by the emulator logic.

The on-board microcontroller is always operated in expanded wide mode even if another mode such as single-chip is being specified by the user. The pins on the on-board chip will show this expanded mode while the header pins will look like the user specified mode. Do not take signal tests directly from the chip as you may receive erroneous results.



Clock Jumpers



1) This set of jumpers is used to select either the Nohau PLL clock source or a user supplied crystal or external source is used to supply a clock signal to the emulation CPU on the emulator. In addition, jumpers are used to connect the XFC, EXTAL or XTAL pins of the CPU to the target board as desired by the user.

2) The default values of C0 is 1 nf (1000 pf) and Ca is 100 pf. R0 is 43 Kohm. These are the values as installed by Nohau. Contact your Motorola representative for the best values appropriate for your frequency of operation or refer to www.icetech.com/s12calc/pll_302.html for an S12 PLL circuit calculator supplied by Motorola.

The TRG - XFC jumper selects the PLL filtering components from the target system. The Nohau installed C0, R0 and Ca must be removed if this jumper is installed.

3) “PLL” as in EXTAL/PLL denotes the Nohau internal PLL clock source and not the Motorola on-chip PLL clock. The Nohau PLL supplies a 2.5 VPP signal and the frequency is set in the SeeHau software under the Config, Emulator window.

4) Nohau PLL clock source can be used to supply either the Motorola divide-by-2 clock or the Motorola PLL circuitry as determined by the setting of the PLLSEL bit in the CLKSEL register. If the Nohau PLL source is used to supply EXTAL without the Motorola PLL, the E clock or bus speed frequency will be 1/2 the value entered into the SeeHau software.

If the Motorola PLL is used, then the output frequency will be determined by the ratio of the SYNRR and REFDV registers. In this case the Nohau PLL will be OSCCLK in the Motorola PLL circuitry. The jumper EXTAL/PLL supplies the Nohau PLL clock source to the EXTAL pin of the CPU.

The Nohau PLL Clock Source and How It Works

The Nohau PLL clock source frequency is selected in the SeeHau software under Config, Emulator. The value entered in the box titled Clock (MHz) and can range from 3.125 MHz to the maximum frequency of the emulator which can be 16, 24 or 25 MHz. These values are the possible outputs of the PLL circuitry but may or may not be appropriate for your system configuration. Practical frequency settings could be narrower than these maximum ranges.

The value entered has a software resolution of 15 digits but this exceeds the capabilities of the PLL circuitry. The PLL will provide a frequency within +/- .05% of the value entered. SeeHau will return to the box the frequency it was able to select. If 3.125678 is entered and APPLY is clicked, 3.12568 is returned in the box and this signal is applied to the emulator circuitry and pin 1 of jumper EXTAL/PLL. This clock is 2.5 VPP.

The diagram illustrates the hardware configuration for the PM-MCS912D Rev B board, which is based on the MC9S12 CPU. The top portion shows the internal circuitry connecting the CPU pins to external components and target connectors. The bottom portion shows the physical layout of the board with various connectors and components.

MC9S12 CPU Pin Connections:

- XFC:** Connected to the TRG-XFC target connector.
- VDDPLL:** Connected to the TRG-P0D target connector.
- EXTAL:** Connected to the TRG-EXTAL target connector.
- VSSPLL:** Connected to the TRG-P0D target connector.
- XTAL:** Connected to the TRG-XTAL target connector.
- XTAL/PLL:** Connected to the TRG-XTAL target connector.

Internal Components:

- R0:** 43 K resistor.
- Ca:** 100 pf capacitor.
- C0:** 1 nf capacitor.
- XTL:** Crystal.
- C-XTAL1:** Capacitor.
- C-XTAL2:** Capacitor.

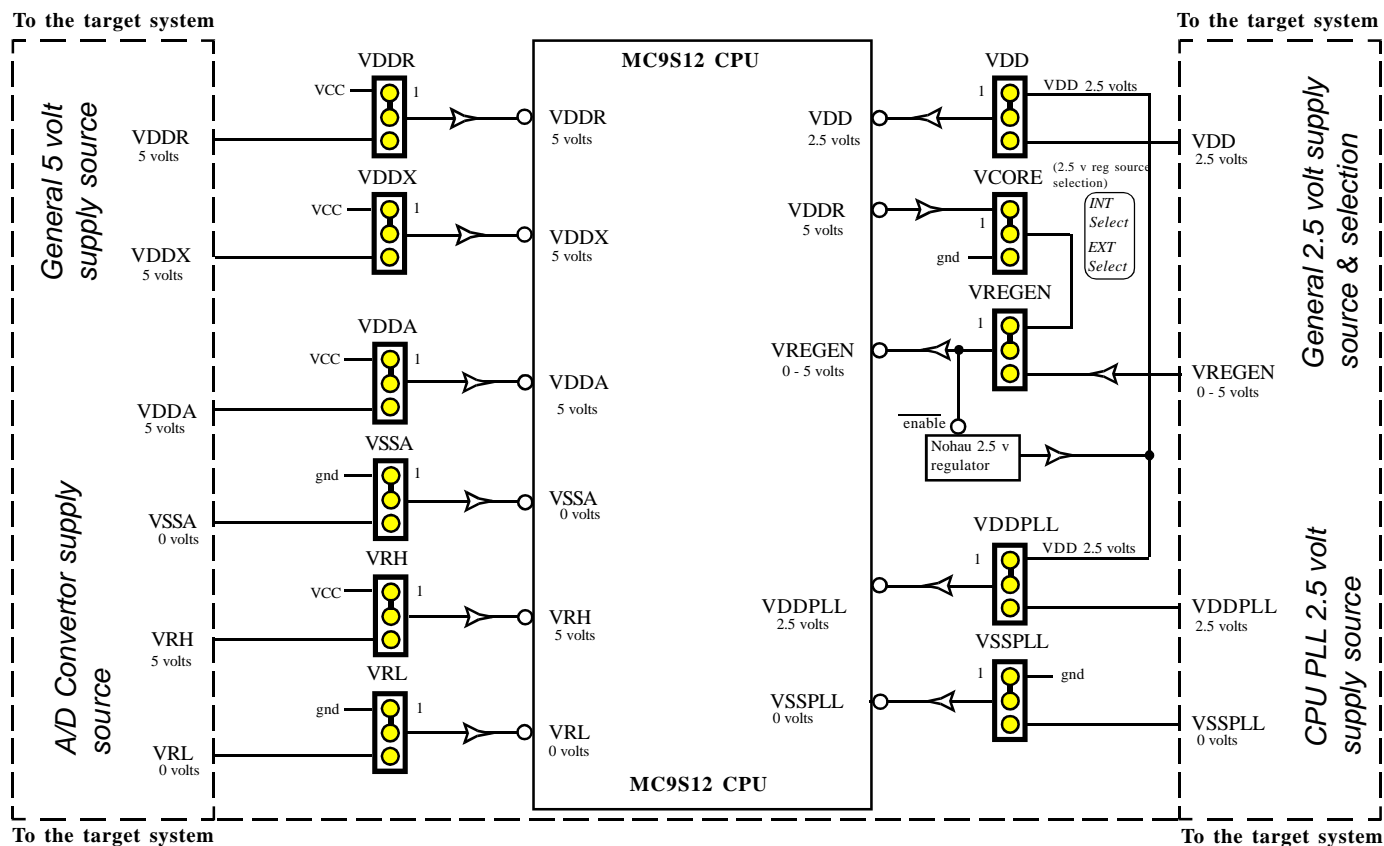
Board Components and Connectors:

- PP2, PP0, PK2, PK0, PT1, PT0, PT3, PT2, PT5, PT4, PT7, PT6, PK4, PK5, PJ0, PJ1, PH6, PH7, PH4, PH5, VDDR, VSSR:** Various pins on the left side of the board.
- R100, U100, U110, U5, S/N, U3:** Other components and connectors on the board.
- MC9S12 CPU (U1):** The central microcontroller chip.

Target Connectors:

- TRG-XFC:** Connects to the XFC pin.
- TRG-P0D:** Connects to VDDPLL and VSSPLL pins.
- TRG-EXTAL:** Connects to the EXTAL pin.
- TRG-XTAL:** Connects to the XTAL and XTAL/PLL pins.

Voltage Jumpers



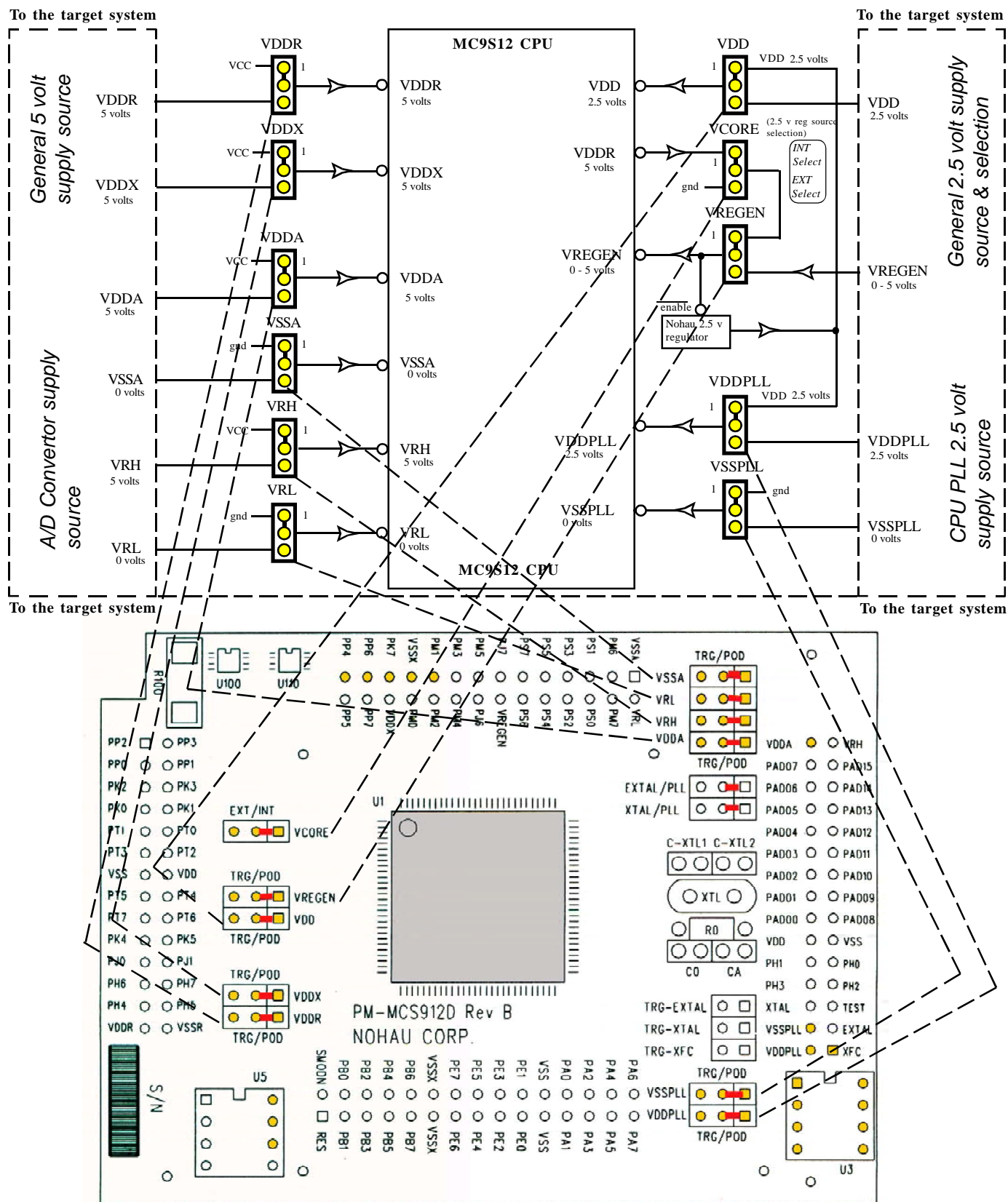
- “VCC”, “VDD”, “gnd” are supplied by the Nohau emulator POD. VCC is 5 and VDD is 2.5 volts.
- Upper positions are POD, lower are TRG (target). These are marked on the CPU module. VCORE is the exception and is marked accordingly on this document and on the module.
- ➤ indicates the direction of the effect, not current or signal electron flow.
- VDD is the 2.5 volt output of the CPU internal voltage regulator. Do not connect 5 volts to this point or the CPU will likely be destroyed. This is also true for VDDPLL.

All names in **BOLD** are jumpers and not CPU pins.

VCORE selects if the CPU internal 2.5V regulator will be turned on or off when the **VREGEN** jumper is in the POD position (default). If **VREGEN** jumper is in the TRG position, the target will determine the state of the internal regulator. If the CPU internal regulator is turned off, the Nohau external 2.5V regulator on the CPU-Module is automatically activated. This is used to power the VDD & VDDPLL pins of the CPU as selected by the **VDD** & **VDDPLL** jumpers (POD) or the target system if set to the TRG position.

The other jumpers determine whether voltage sources or grounds are supplied by the target system or the emulator system. The default position is to select the emulator sources.

Voltage Jumpers to CPU Module Map



Getting the Motorola PLL to Work

A) The Basics

The emulator as it is delivered from Nohau with the default jumper settings can be run using either the Motorola divide-by-2 clock or the Motorola PLL mechanism. Recall that the word “PLL” as indicated on the personality module or in this document refers to the Nohau PLL clock generator and not the Motorola CPU PLL. We will refer to the Motorola PLL with “CPU PLL”. Consult the Motorola datasheet for more information on their Clock and Reset Generator (CRG) module.

The default is the divide-by-2 clock and you can switch it to CPU PLL by programming the SYNCR and REFDRV registers then activating the CPU PLL by setting the PLLSEL bit in the CLKSEL register. No jumper changes are necessary.

You can switch the emulator from CPU PLL to divide-by-2 with your code running on the emulator or by typing in the appropriate values into the specified register windows in Seehau while the emulator is in monitor mode (i.e. emulation is stopped).

B) Setting Up the Emulator

These instructions will construct the windows in the Seehau software to allow you to manipulate the CPU PLL circuitry. You will need an oscilloscope to view the oscillator waveform and measure its frequency.

- 1) Start Seehau in the normal fashion and right mouse click on a register window. The default Reg_1 window will do fine.
- 2) Select Add Special Register (SFR) and double click or press Enter. The SFR window as shown in Figure 1 opens up.

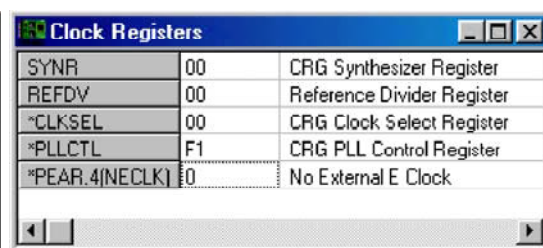
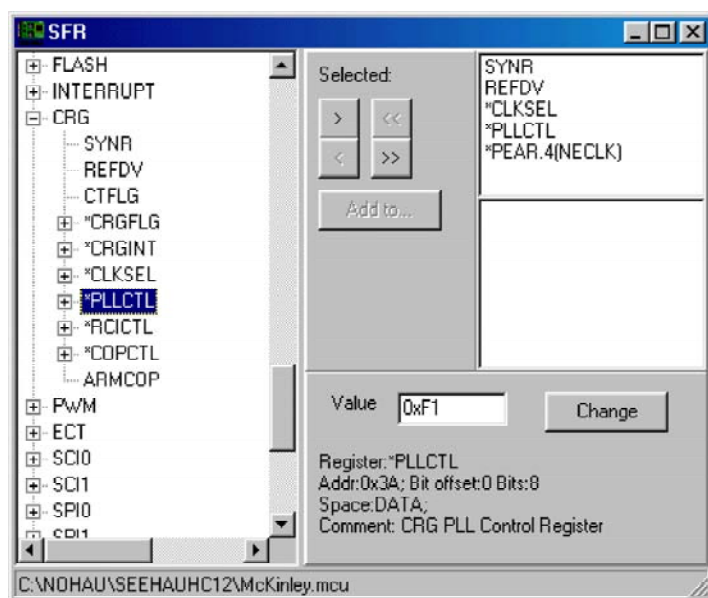


Figure 2

Figure 1

- 3) Double click on CRG or click on the “+” mark beside it. You might have to scroll down to view this.
- 4) In turn, highlight and click on the “>” symbol to place the register names in the window on the right as shown in Figure 1. You will find the NECLK=0 under I/O Control, *PEAR. NECLK is a bit in the PEAR register and is used to send the E Clock to the PE4 header pin on the top of the CPU module for easy access by an oscilloscope.
- 5) Click on Add To.. and select New Window. A window similar to Figure 2 will open up. It will probably be labelled Reg_2.
- 6) Right click on this new register window and select Show Description.
- 7) Right click on this register window again and select Change Caption. I entered Clock Registers and Figure 2 is the result.
- 8) In the main Seehau screen, select Config, and select Save Settings to save the entire workspace. Give it a name of your choice or use the default startup.bas. You can also select Save Window to save the register window if it is currently in focus to a macro that can be easily recalled with a button. See the Getting Started manual for more information.

C) Getting the Hardware Configured

- 1) Have Seehau operating but not running any user programs. The green GO icon should be visible.
- 2) Connect an oscilloscope to the header pin PE4 on the top of the Nohau personality board. This pin is readily accessible.
- 3) In the register window Figure 2, enter a 0 into Pear.4 (NECLK) and press Enter. This sends the E clock signal to PE4. This will now be visible on the oscilloscope. Entering a 1 will turn it off. In Single Chip mode, this bit defaults to a 1 (click on the Nohau RESET icon) and in Expanded modes it defaults to 0. We are using Single Chip but you can use Expanded if you prefer.
- 4) Select the Pear.4 (NECLK) line and right click on it. Select Change Attributes and check the Enable Reset Value. The default value of 0x0000 is already entered. Click on OK. Everytime you click on the Nohau RESET icon, the Pear.4(NECLK) bit will be set to zero.

You can do this later to the SYNRR and REFDV registers with your own default values. You must resave the settings under Config, Save Settings in order for these effects to be activated the next time you open Seehau.

- 5) Click on Config, Emulator in the main Seehau screen and note the value in the Clock(MHz) box. The default is 16 MHz and represents the value of the Nohau PLL clock source as fed through the EXTAL/PLL jumper to the EXTAL pin on the DP256 microcontroller.

The E clock frequency in the divide-by-2 mode (no CPU PLL) will be 1/2 the Nohau PLL frequency. The default E clock frequency displayed on your oscilloscope will be 8 MHz with an EXTAL frequency of 16 MHz.

The Nohau PLL or EXTAL frequency is labelled OSCCLK in the Motorola PLL specifications. It is readily accessible for monitoring on the EXTAL/PLL jumper pin 1. It is a 2.5 volt PP signal to correspond with the CPU core voltage. You can easily change this frequency by typing in a new value and pressing Enter or Apply. You can enter values from 3.125 KHz to 16, 24 or 25 MHz depending on the maximum frequency of the emulator. See page 2 for more Nohau PLL information.

D) Basic Motorola PLL Information

- 1) The Motorola PLL is selected by entering 80 in the CLKSEL register in the register window of Figure 2. A zero turns it off. This bit actually switches from the divide-by-2 circuitry to the CPU PLL. When the CPU PLL is off, the E clock will be OSCCLK divided by 2.
- 2) The frequency is determined by the formulae:
$$\text{PLLCLK} = 2 \times \text{OSCCLK} \frac{\text{SYNR} + 1}{\text{REFDV} + 1} \quad \text{or} \quad \text{ECLK} = \text{OSCCLK} \frac{\text{SYNR} + 1}{\text{REFDV} + 1}$$
- 3) PLLCLK does not show up externally on the CPU.
- 4) E clock is PLLCLK / 2 and is visible to the outside world on PE4 as described in 3) above.
- 5) The minimum OSCCLK frequency is from 1 MHz to the limit of the emulator whether it is 32, 48 or 50 MHz.
- 6) OSCCLK is the oscillator frequency as fed into the CPU EXTAL pin via the EXTAL/PLL jumper.
- 7) OSCCLK is the same frequency as entered in the Seehau software in the Config, Emulator dialog window.
- 8) E Clock is derived from the OSCCLK and is 1/2 the frequency of OSCCLK when in divide-by-2 mode. In CPU PLL mode it will be the ratio of SYNRR+1 and REFDV+1.
- 9) Register and frequency values used must be in appropriate ranges. It is possible to set the CPU PLL to provide frequencies outside the range of the CPU and/or the emulator. Erroneous microcontroller or emulator operation will result.
- 10) The CPU PLL filter values as determined by C0, Ca and R0 must be selected according to the frequencies synthesized. Contact your Motorola representative for these values or refer to www.icetech.com/s12calc/pll_302.html for an S12 PLL circuit calculator supplied by Motorola.
- 11) We will only program and pay attention to the SYNRR, REFDV registers and the PLLSEL bit in the CLKSEL register for this demonstration. This is to demonstrate the Motorola PLL with the Nohau emulator functions as described and provides a good reference point for your software design. Consult the Motorola datasheet for information on other clock registers. The Nohau emulator supports all the clock registers even if they are not mentioned here.

E) Basic Nohau Emulator Information

- 1) You can change the CPU PLL registers on-the-fly while running your program by entering the values into a properly set data window. The Motorola BDM is used to update these values and is described on the next page.
- 2) You can change them when the emulator is stopped by entering new values into a register window.
- 3) You can also change them in your program code. An example is given later in this document.
- 4) You may change your CPU PLL frequency as often as you want and the full Nohau emulator will track these changes. It will make any modifications to its operating environment as needed and in real time without stealing machine cycles.
- 5) There is an issue of the LOCK bit not correctly indicating the CPU PLL is out of lock after writing to one of the PLL registers. There is a simple workaround which consists of a short time delay before you test the LOCK bit. This is an issue only when you are switching register values with your own code. It is not an issue if you are entering register values into the emulator manually because of the inherent time delay.

F) Programming Methods for the Motorola PLL Registers

The standard method of programming the CPU PLL registers (see Figure 2) for the purposes of this article is the following:

- 1) Enter an appropriate value into the Seehau software for OSCLK. 8 MHz is used in this application note.
- 2) Make sure the CLKSEL register contains 0. This turns the CPU PLL off by setting bit PLLSEL off.
- 3) Program the appropriate values into SYNRR and REFDV. We will use in this note values will be in the area of 1, 2, 3 or 4.
- 4) Enter 80 into CLKSEL. This sets bit PLLSEL on. If the values above are in range, the E Clock will change to the new frequency.
- 5) To change the CPU frequency again, simply write a 0 to CLKSEL, enter new values to SYNRR and REFDV as needed and write an 80 to CLKSEL to activate the CPU PLL again.

G) Changing the CPU PLL via the Seehau Register Window (see Figure 2)

- 1) Make sure the emulator is setup as described previously under *Getting the Hardware Configured*.
- 2) Enter 8 into the Clock MHz box under Config, Emulator in the main Seehau window. This sets OSCLK to 8 MHz. The frequency as measured on ECLK (PE4) will be 4 MHz if the CPU is in divide-by-2 mode.
- 3) Enter 0 into the Pear 4 NECLK register. This sends the E clock to Port E.4. (PE4)
- 4) Enter 0 into the CLKSEL register and either press Enter or click on another register. CPU PLL is now ensured to be off.
- 5) Enter 1 into SYNRR and 3 into REFDV. The frequency of PLLCLK will be $(2)(8)(1+1)/(3+1) = 8$ MHz. ECLK will be 4 MHz.
- 6) Enter 80 into CLKSEL and press Enter. 4 MHz will be available on the scope which is the same as before.
- 7) To test if the CPU is in PLL mode, grab the PLL filter components with your fingers and an increased fuzziness in the scope display will result. If the PLL filter components are very close to certain values, this effect may be very small. If the CPU is in divide-by-2 mode, doing this will have reduced or no effect. This is called the SRTest after its discoverer.
- 8) Enter 0 into the CLKSEL register and press Enter to disable the CPU PLL. The SRTest will fail. The CPU PLL is off.
- 9) Enter 4 into REFDV and then 80 into CLKSEL and press Enter. ECLK will now equal $(8)(1+1)/(4+1) = 3.2$ MHz. Confirm this with your oscilloscope. Try the SRTest. It will indicate the CPU PLL is running properly.
- 10) Enter 0 into the CLKSEL register and press Enter to disable the CPU PLL. The SRTest will fail.
- 11) Enter 3 into SYNRR and then 80 into CLKSEL and press Enter. ECLK will now equal $(8)(3+1)/(4+1) = 6.4$ MHz.
- 12) Reset the emulator by clicking on the RESET icon.

If the waveforms become fuzzy, this likely is the result of the PLL filter components C0, Ca and R0 not being correct for this frequency of operation. Consult Motorola for correct values or refer to www.icetech.com/s12calc/pll_302.html for an S12 PLL circuit calculator supplied by Motorola. This example worked with the Nohau default values.

H) Software Control of the Motorola PLL

The Motorola PLL can be controlled directly by software and switching on-the-fly is possible. There are a few issues with the main one being the LOCK bit problem. The LOCK bit (bit 3 in the CRGFLG register) indicates that the PLL is within a certain range of the desired frequency and is used to detect if it is allowable to enable the CPU PLL so it will become the system clock.

The problem is the setting and clearing the LOCK bit is delayed after the SYNRR and REFDV is written to. Normally the SYNRR, REFDV and perhaps other registers will be initialized by your code and then the PLLSEL bit is set to enable the CPU PLL. The code must wait until the LOCK bit is set indicating the PLL is stable before it can be used in the system and before program execution continues. Normally this would be accomplished by testing the state of the LOCK bit before setting PLLSEL.

The mentioned delay will result in the LOCK bit still asserted even if the PLL is not locked if tested immediately after setting the SYNRR, REFDV or PLLCTL. A successful workaround is to insert a small time delay after configuring SYNRR, REFDV or PLLCTL but before setting the PLLSEL bit. The LOCK bit can be tested and when it indicates the CPU PLL is stable and locked, it is safe to enable it. Assembly code and C source examples are given.

I) Nohau Test Program: Switches the CPU PLL On and Off Continuously

A Nohau test program that switches the CPU PLL clock on and off is included in the Seehau software release. It is in the \Examples\DP256\Tstpll directory in the installed software. The asm code is commented.

It loads to address 400H so you need to have the EEPROM turned off to run it (check the Disable EEPROM check-box).

Open a RUN-TIME-DATA window to point to 7F0H. This shows whether continuance BDM communication is maintained during the rapid changes in the BDM communication rate when the PLL is selected and deselected.

Enter 8 MHz clock speed into the Config, Emulator window which then creates 4 MHz & 16 MHz bus speeds at Non-PLL and PLL modes accordingly.

The program selects and deselects the PLL to be used as the system clock many times in an endless loop. It has a ratio of 4:1 between PLL mode to non-PLL mode. It increments two counters at 7F0H & 7F8H every time PLL or Non-PLL mode becomes active. This will be visible in the data window.

J) Motorola Assembly Example

This example is taken from a application note by Gordon Doughman. Go to www.motorola.com/mcu and search for AN2153.

“Shown below is a code snippet from one of my up coming app notes. To obtain 25 MHz operation the reference clock must be an integer multiple of 25 MHz. So, if you are using the 16 MHz oscillator on the evaluation board, the value for OscClk would be changed to 16000000. The value for fEclock would be changed to 25000000. The only way to obtain a 25 MHz E-clock with a 16 MHz oscillator is to have a reference frequency of 1 MHz, so RefClock should be set to 1000000. For 25 MHz operation, a 5 MHz crystal/oscillator would be a better choice.”

```
OscClk:      equ 8000000          ; oscillator clock frequency.
fEclock:     equ 24000000         ; final E-clock frequency (PLL).
RefClock:    equ 8000000         ; reference clock used by the PLL.
REFDVVal:    equ (OscClk/RefClock)-1 ; value for the REFDV register.
SYNRVal:     equ (fEclock/RefClock)-1 ; value for the SYNRR register.
    if OscClk>12800000
FCLKDIVVal:  equ (OscClk/200000/8)+FDIV8 ; value for the FCLKDIV register.
    else
FCLKDIVVal:  equ (OscClk/200000)         ; value for the FCLKDIV register.
    endif

    ldab     #REFDVVal            ; set the REFDV register.
    stab     REFDV
    ldab     #SYNRVal            ; set the SYNRR register.
    stab     SYNRR
    nop
    nop                          ; nops required for bug in initial silicon.
    nop
    nop
    nop
    brclr    CRGFLG,#LOCK,*       ; wait here till the PLL is locked.
    bset     CLKSEL,#PLLSEL       ; switch the bus clock to the PLL.
```

K) C Code Example

The **FOR loop** for (i =0 ; i <4 ; i++); provides a delay to allow the LOCK bit to be cleared so testing it will make sense. Remove this line and the program will not work correctly.

```
CLKSEL.Data.Byte &= ~PLLSEL;      // disable PLL
SYNR.Data.Byte = 0x3;
REFDV.Data.Byte = 0x01;
for (i =0 ; i <4 ; i++);          // small delay
while (!(CRGFLG.Data.Bit.B3))
CLKSEL.Data.Byte = PLLSEL;        // enable PLL
```



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, ME, NT, XP and 2000Pro.

Seehau has the capability to run over a TCP/IP stack. You can easily control the emulator over the Internet from anywhere including for Nohau Technical Support. SeeHau is also an OLE Automation server. 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. 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 such as C++, Java, Delphi or Visual Basic.

Connecting the Emulator to your PC

There are three methods available to connect the emulator to your PC. One is the EPC (Emulator Parallel Cable) cable in Chapter 1 Figure 3. This connects to a LPTx printer port. This provides the most portable method of connection and is perfect for laptops. The second is an 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. If you have trouble with your ISA card try 210. The third is the USB cable, it connects to the USB connector on the appropriately equipped PC and to the emulator with the standard 25 pin D shell connector.

Software Installation

Seehau comes on a CD ROM and installs easily. Install the software to your PC hard drive in the manner required by the version of Windows you are using. This document uses Windows 98 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 onto your desktop. SeeHau Config is used to construct startup.bas which is the startup macro file. It is an ASCII file hence can be edited with any editor. It can also include BASIC language commands. You can modify many configuration items when SeeHau is running under the Config, emulator menu selection. You are unable to modify the controller emulated this way. You must run the config or reconfig utility or manually edit the startup.bas file to accomplish this.



Figure 1

Always use the Windows Uninstall utility to unload any existing version of SeeHau from your computer before loading in another copy. SeeHau will detect an existing copy of itself and allow you to uninstall it. You can manually uninstall it under My Computer select Control panel, then Add/Remove Programs. Uninstall will not delete any files and folders that were originally installed. Save your personal macros and source files if needed for backup. 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. Select Config, environment and check "Use Startup Dialog?" to choose it at startup time.

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@icetech.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.

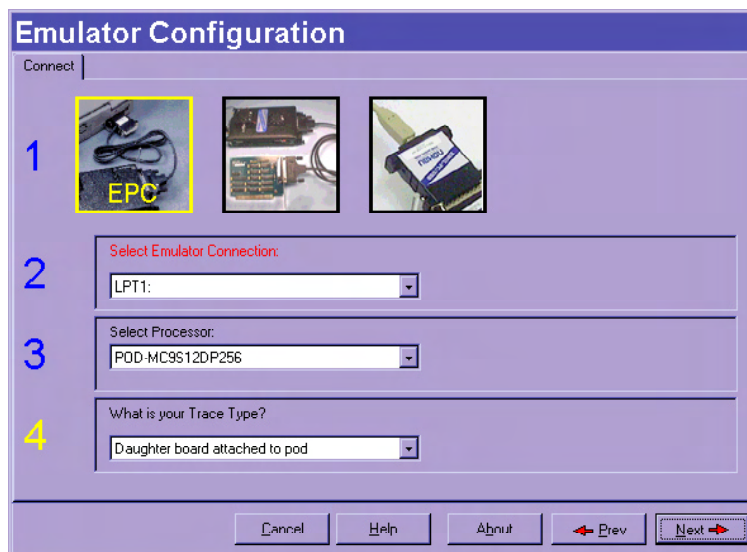
Note you do not need to have the emulator connected to the PC to run the Seehau Config icon. You do need the emulator connected and with its jumpers properly set for the Seehau regular executive to operate properly. For more information on macros see the Macro Section in this manual.

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, the LC-ISA card or the USB.
- 3) Confirm the settings as indicated. Select POD-MC9S12DP256. The EPC cable and the LC ISA card are the only appropriate choices until the USB board is ready. Figure 2 shows the settings used if you are using the EPC cable. Choose the correct LPT port. The EPC cable is distinguished by a male/female connector on the end that plugs into your PC parallel port. Figure 3 shows a portion of the window if the ISA card is selected. The settings for the LC-ISA card are similar. The Emulator Board Address dialog box that will appear 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, try address 210. 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. This setting changes the Nohau PLL clock frequency. A setting of 16 gives an 8 MHz bus speed. Setting it to 32 provides a 16 MHz bus speed.

Processor: This will be set to POD-MC9S12DP256. This personality card will provide support for other DP256 family members or the specific family member can be installed on the card. Please contact Nohau.

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

Miscellaneous: Select with a check mark Disable Internal EEPROM and Disable ROM to substitute emulation RAM for these modules. This simplifies loading test programs. Select Mask Interrupt on Step so your single-stepping runs straight through interrupt routines in their entirety.

All other settings will be the default values. You can change some of these settings in the main Seehau menu later by clicking under Config, Emulator. You will see more options in this menu such as memory mapping control and how to control the COP Watchdog timer.

- 6) Click on Finish and the 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 EMUL-S12D 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@icetech.com or 650.375.0409 or your local Nohau representative.



Figure 3

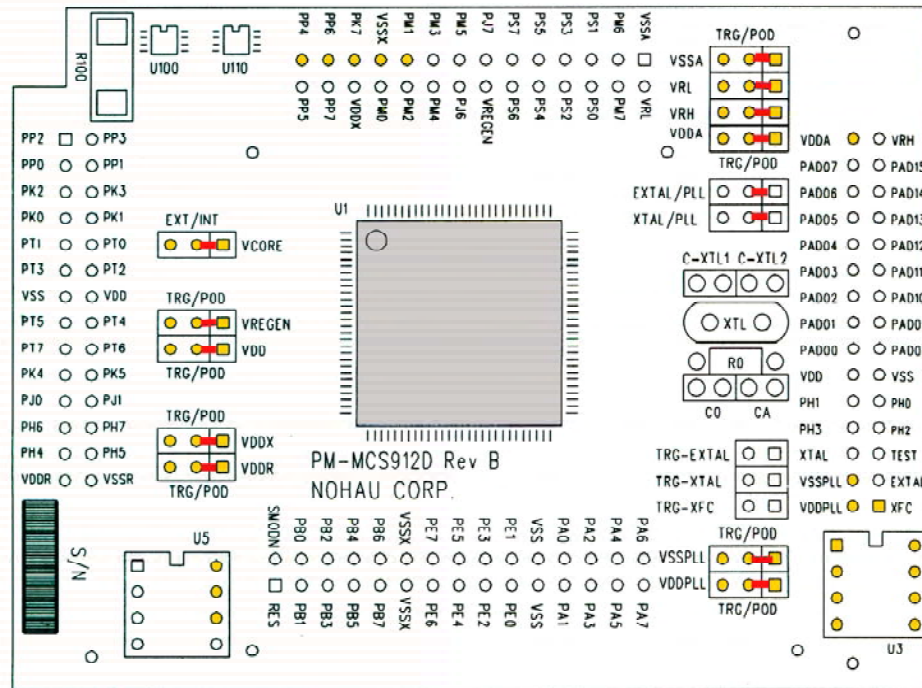
Figure 4

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. Seehau will prompt you if necessary. Then reinstall a fresh or updated version of Seehau. The latest version of Seehau is always available on the Nohau website and is without charge.

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.

- 1) The jumpers on the daughtercard must all be on their default positions as shown below with the red lines. All the jumpers except for TRG-EXT, TRG-XTAL and TRG-XFC are jumpered on the side marked POD. TRG-EXT, TRG-XTAL and TRG-XFC are OFF and VCORE is set at INT.V. TRG means Target.
- 2) There are two capacitors and one resistor in positions CO, CA and RO. They are not needed for this tutorial but can be installed. They are the CPU internal PLL filter components. RO is 43K, CO is 1 nf (labelled 103) and CA is 100 pf (labelled 101).



- 2 - 23

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 do not need a crystal installed. The emulator PLL is used for the clock frequency.

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@icetech.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.

The program counter gets reset to 181 but this may vary depending on the emulator. You can force it to another value by right clicking on the source window and selecting Go To Address, entering a value and clicking on OK. You can set 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 desired and click on the Program Counter box. Note you can also preset the Stack Pointer in this manner. Normally the emulator will select the reset value depending on the contents of FFFE. (reset vector)

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.

Please call or email Nohau Technical Support or your local rep if you experience difficulty and need help.

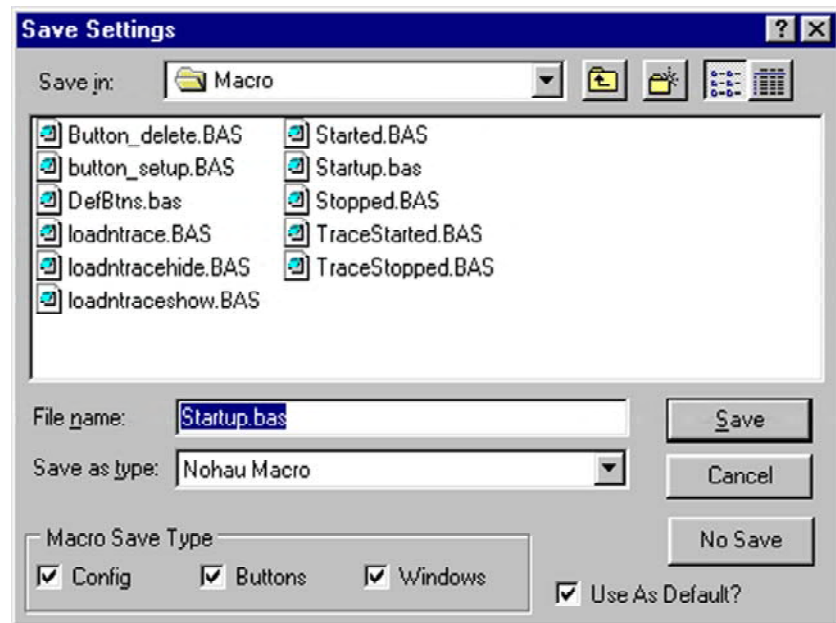
Figure 5

Shutting down Seehau

- 1) Click on the X in the corner, press ALT X or select the menu item File, Exit. Figure 6 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? in the Config, Environment menu is checked, 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.

Selecting the Startup Macro

Figure 6



When Seehau is started, the file seehau.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 Seehau is started Figure 7 will appear. You can select the desired macro file.

Figure 7



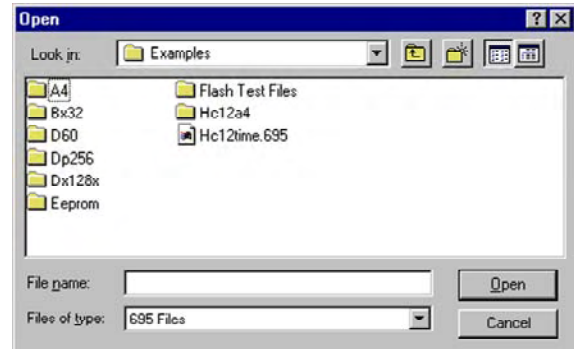
Chapter 3: The Example program and Data Display

The Example Program and Data Display:

Loading the example program BarTime.695

- 1) Nohau provides a small example program called BarTime.695. This file, source code and other compiler output files, are found in the c:\Nohau\SeehauHC12\Examples\Dp256 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.
- 4) Figure 1 opens up. Double click on the Dp256 directory in this directory. This shows all the files in this directory with the file extension of .695. You can select other file formats in the dialog box Files of Type.

Figure 1





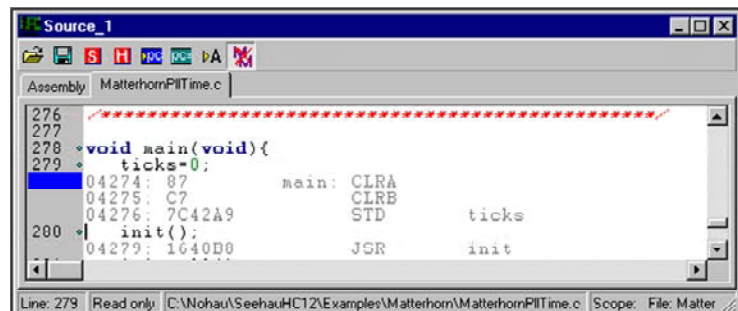
- 5) Highlight BarTime.695 and click on Open. You can also double-click on this file to load it. BarTime.695 will load into the emulator. This file is located in IEEE 695 format. The source window will show a CLRA instruction at 2000 as pointed to by the reset vector (at FFFE). The JSR at 2011 is to 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. You can confirm the loaded code is not corrupted under File, Verify Loaded Code.
- 6) Click on the Step Into icon  (or press F7) and the program will run to the start of MAIN.
- 7) Note that the BarTime.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 BarTime.c tab selected and select Mixed Mode or click on the MM icon . Select *At PC Line*. You will see assembly code and source lines similar to Figure 2. The addresses will be in the 2000 range. Note the program counter (PC) is indicated by the blue block.
- 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. If you click on Assembly Step Into, the Source window is switched to Mixed Mode.

Figure 2



Viewing Data in Real-time with the Shadow RAM

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

- 1) Open a Data window either by clicking on the Data icon or selecting New, Data.
- 2) A window similar to Figure 3 will appear. The data will be in hex: not ASCII as shown. Resize it as needed. In the address box at the bottom, highlight the existing address and type *show* and press Enter.
- 3) Right mouse click on the Data window and Figure 3 shows. You can change the Data window to display the data in ASCII format and retrieve it from the Shadow RAM without slowing the emulation.
- 4) Select Display As ... and select ASCII. Note the viewing options available in this window. Click OK. Note you can select these items from the margin display areas in the Data window.

Figure 3

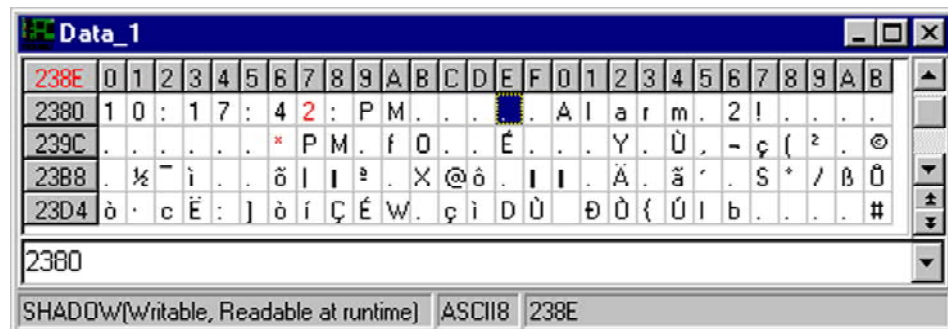
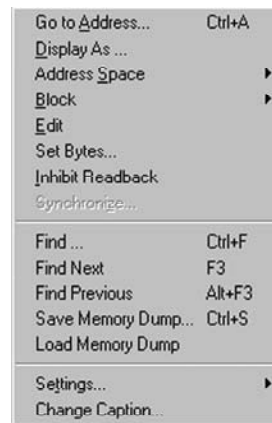


Figure 4



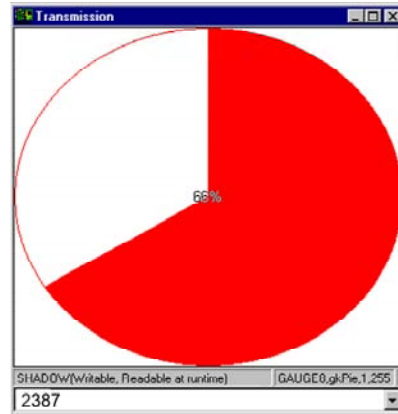
- 5) Right mouse click again and select Address Space ... and select SHADOW.
- 6) The address at the top left corner represents where the mouse is pointing to. The box highlighted in blue at location 238E in Figure 3 is the last location where the mouse was positioned when this screen shot was made. Data in red indicates that data which has been modified by the last instruction executed. You will not see ASCII data if BarTime.695 has not initialized the memory at these locations.
- 7) Click on the GO icon or press F9. The program BarTime.695 will run.
- 8) The time will be updated in real-time. No CPU cycles are stolen to accomplish this. You can modify the contents in the Shadow RAM memory without intrusion into real-time by double clicking and typing a new value in. The CPU will look for a free cycle, then inject your new data value into the RAM. If no free cycle is found in 128 cycles, then it will inject the value in by stealing a memory write cycle from the CPU.
- 9) If you select Run Time Data the HC12 BDM feature will be used to provide real-time display (usually) in the data window. You can modify this data in the same fashion as in the Shadow RAM. The Run Time data is the HC12 BDM feature. The Shadow RAM is a Nohau feature and never steals CPU cycles.

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 2387.

Figure 5

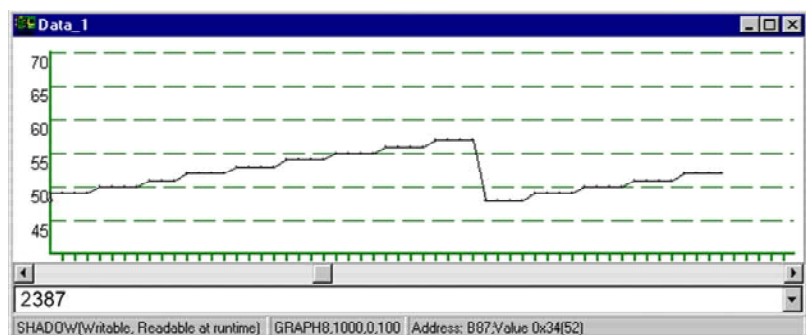


- 5) The gauge will display the value of 2387 in terms of its percentage of FF hex.
- 6) The range is not very suitable. Note that the 2387 values range from 30 to 39 hex which corresponds to 48 to 57 decimal. These represent the ASCII characters 0 through 9.
- 7) Right click on this Data window and select Setting, Gauge. Enter the range of 48 and 57 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 in the Setting, Graph menu 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



Data Hints

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.

Stop emulation and place the cursor over an interesting name in the Source window. I selected the word “timer” in timer.hour (in line 152) and Figure 7 was displayed.

Figure 7

```
timer = UnnamedStruct0 {...}
hour:  unsigned char '\n' 10 (0xa)
min:    unsigned char '!' 17 (0x11)
sec:    unsigned char '!' 23 (0x17)
am_pm:  unsigned char[10] 0x23a3 "PM"
status: unsigned char[10] 0x23ad ""
```

Watch Window

The Nohau Watch window is replaced with the combined Inspect/Watch window. This window is found under the New menu or CTRL-1.

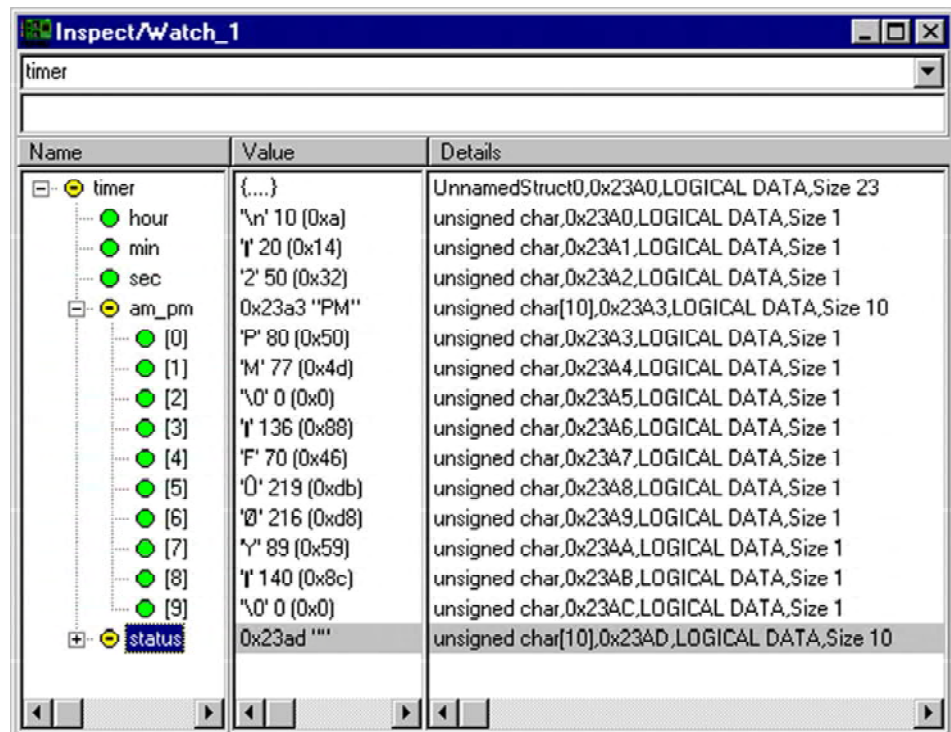
Inspect/Watch Window

An Inspect/Watch window is used to view the contents of variables or structures. These values can be updated in real-time if desired. The contents of the variable or of an element can be modified but 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

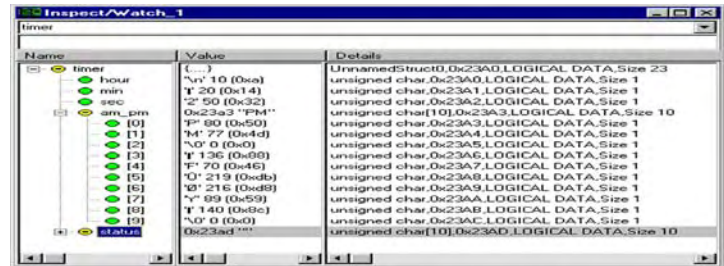
- 1) Select the New, Inspect menu item. A window similar to Figure 8, but empty, will open.
- 2) Enter *timer* in the top data entry bar. Click on the small yellow icon ☺ or the + sign beside it. The contents of the structure time will expand and display similar to Figure 8. You can also right click on the name in the Source window and select Debug Windows, Add to Inspect to add a element.
- 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.
- 4) Right click on this window and select Update During Runtime. The elements will now be updated while the emulation is running when you click on GO.

Figure 8



Modifying the Data in the Inspect Window

- 1) Click on the hour element and then a right mouse click on it. Select Modify Value from the menu.
- 2) Enter a new value and press Enter. The new value will be entered into the Inspect window.
- 3) If the element you want to change is being updated you can click once to select the line and then once more on the changing data numbers to enter the modify mode. Change the data as described above.

Figure 9***Viewing more members of the structure***

If you double click on a structure element name or click on the yellow icon, this element will be expanded. If the element is a structure within a structure, this new structure will be displayed.

Updating the Data in Real-Time

Right mouse click on the Inspect window to open the local menu and select Update During Runtime. The Inspect window will use the Shadow RAM facility to update the values as they are modified by CPU writes.

Saving Inspect Windows

You can save any number of Inspect windows with all your selections as a Nohau macro and recall them easily.

- 1) With the Inspect window in focus (i.e. activated) open the Config, Save Window settings. Specify a filename of your choosing and the bas extension will be added to it if you do not.
- 2) Close the Inspect window.
- 3) Recall your Inspect window open the Macro menu item and select Run. Choose the appropriate macro filename and your Inspect window will be recalled.

Other Inspect Features

Right mouse click on the Inspect window to open the local menu to see other options available. Locate in Data Window is particularly useful.

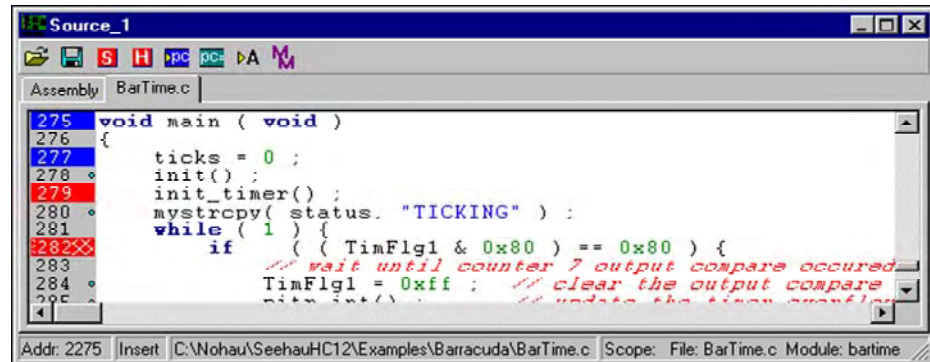
You can display more than one item in the Inspect window at the same time.

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 BarTime.695 is still loaded in your emulator. If not, please reload it at this time.

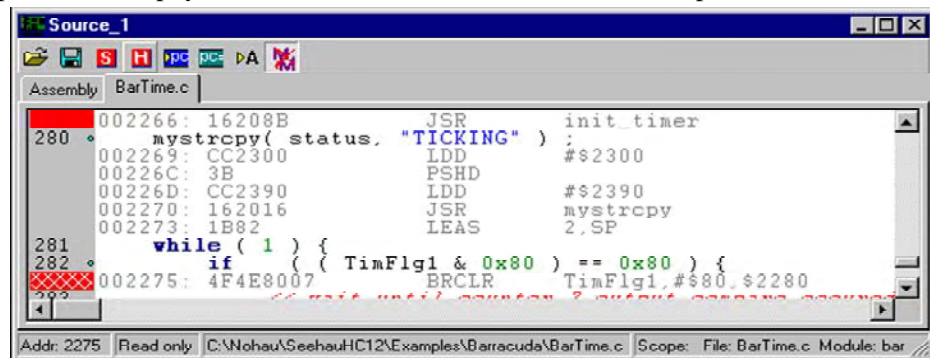
- 1) Click on the Reset icon. Select the BarTime.c tab in the Source window. You may need to click on the Step Into icon to activate this tab.
- 2) Scroll to lines 279 and 282. Note the two small green dots beside each line number. This indicates that there is at least one assembly instruction that a breakpoint can be set on.
- 3) Click on the number 279. The 279 turns red indicating a software breakpoint has been set.
- 4) Click on the number 282 while holding down the ALT key. The 282 turns a cross-hatched red indicating a hardware breakpoint has been set. Figure 10 shows the resulting screen.

Figure 10



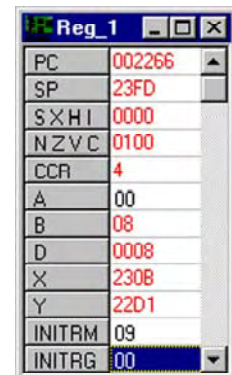
- 5) Right click on the Source window and a local menu will open up. Select Mixed Mode and At Top Line. Or click on the MM icon. Scroll if needed so the breakpoints at lines 279 and 282 appear as in Figure 11.
- 6) Note the software breakpoint is set to physical address 002266 and the hardware breakpoint to 002275.


Figure 11




- 7) Click on GO. The PC will advance to 2266 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 2266. The instruction at the breakpoint (JSR = Jump To Subroutine) has certainly not been executed.

Figure 12



- 8) Click on Assembly Step Into  and the JSR instruction is executed. The PC in the Register window is 208B which is the address for init_timer. This is an important feature. Jumps where a breakpoint is set on will not occur. Had such an instruction been executed, the program flow could have been diverted somewhere else with no indication where it had come from. It could have been anywhere.

A Trace memory board would indicate the instruction that caused such a jump. Open the trace buffer window by clicking on the trace icon  and the JSR instruction at address 2266 is clearly shown in Figure 13. This is a single and elementary example of the power of the trace memory. It has shown you where the program came from and this will assist you in debugging this if it is a problem.

- 9) Click on GO and the emulator will run to line 282 (address 2275) and will stop before executing this line. An instruction where a hardware breakpoint is set to will not be executed when the breakpoint stops program execution.
- 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.
- 11) Click on the breakpoints in the Source window to remove them. The blue block appears which is the program counter position when no breakpoint is activated there. You could also select Delete All in the Breakpoint window.

Figure 13

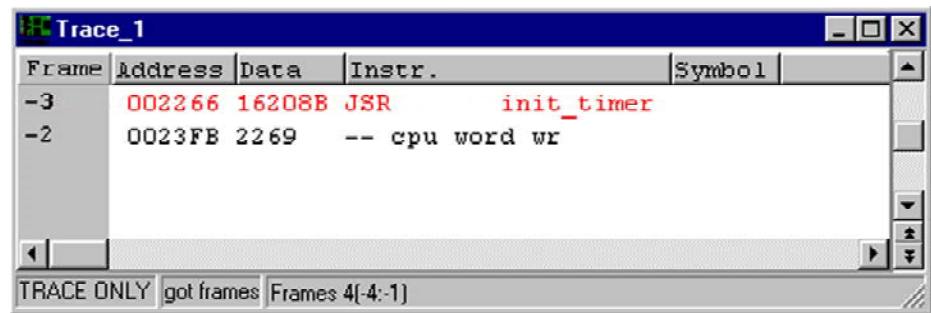
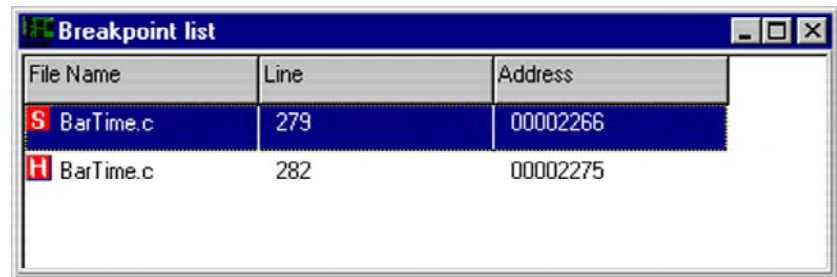


Figure 14



Chapter 4: Commands and Macros

Seehau Commands and Macros

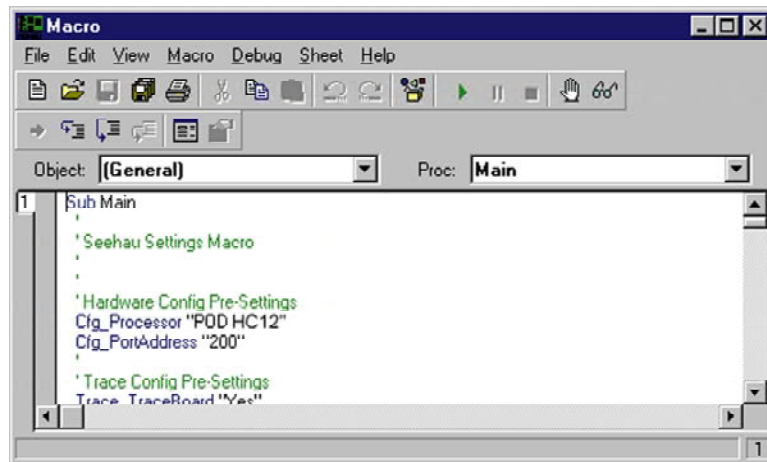
Seehau Commands are used to communicate and control the emulator hardware and the Seehau Software Debugger. These commands are used in the Seehau Macro facilities such as user macros, emulator startup configuration and for Seehau configuration items in general. Commands allow the Seehau debugger software to be a very flexible and powerful tool. Command listing and descriptions are found in the Seehau online help. Most macro operations are done transparently and do not require user intervention.

A Macro as defined here is a collection of commands in a text file that may include SAX Basic commands that are executed by the Seehau debugger. Seehau contains a full debugging environment for designing and working with Seehau macros. You can use as many of the macro facilities as you please. Modifications and preference settings are easily made by the user.

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 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.

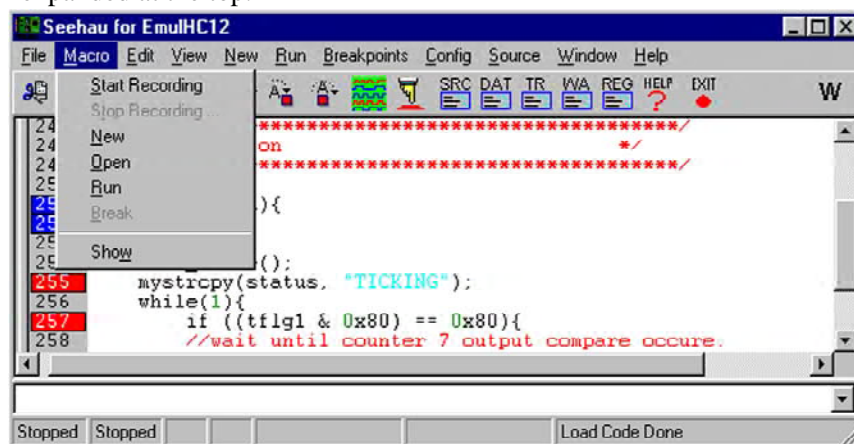
Figure 1



Macro Execution

Macros can be executed by Seehau during the startup phase or manually by the user using the *Macro/Run* command. The box at the bottom of Figure 2 where the word "Load Code Done" is where Seehau puts the results of command operations. In this case it was the operation resulting from loading the sample code timer. 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 Seehau is case insensitive here. An example of a command is: *Cfg_PortAddress "378"*. This command tells Seehau that the emulator hardware is connected to the PC port at hex 378 (LPT1). Descriptions of various fields (if applicable) are contained in the Seehau on-line help.

Saving Macros

You can save many different session settings either when you close Seehau or by selecting the Config, Save Settings menu. Figure 3 will open up.

The specified file will be the one used by Seehau on start-up. *Startup.bas* is the initial default. If *startup.bas* is not present, Seehau will go to the Config program allowing you to specify your system. How to select the start-up file is explained in Chapter 2.

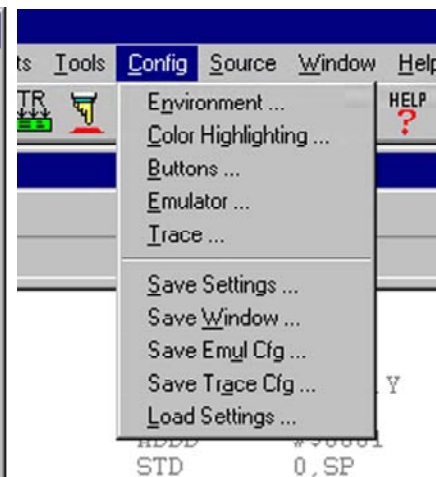
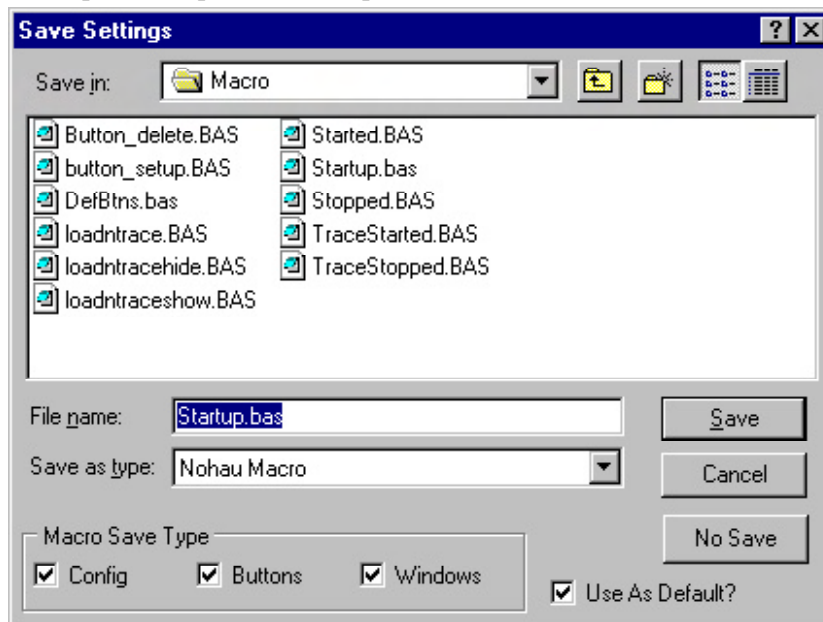


Figure 4

Figure 3

Figure 4 shows the options available under the Config menu. You can save the entire Seehau session (Save Settings), the current window in focus (Save Window), the emulator configuration as set in Config, Emulator (Save Emul Cfg) and the trace configuration as set under Config, Trace (Save Trace Cfg). These will all be saved as a macro and you can save as many of these as you have disk space.

Quick Saving a Configuration Menu using the Apply Button

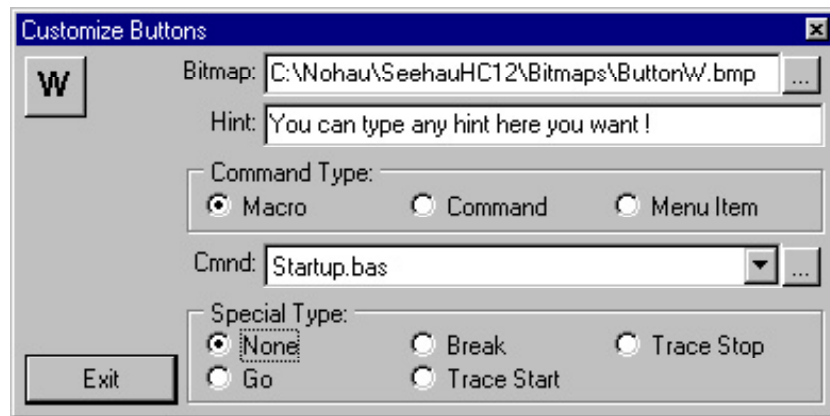
This technique is useful for saving things not covered by the Save Emul Cfg and Save Trace Cfg menu items.

- 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

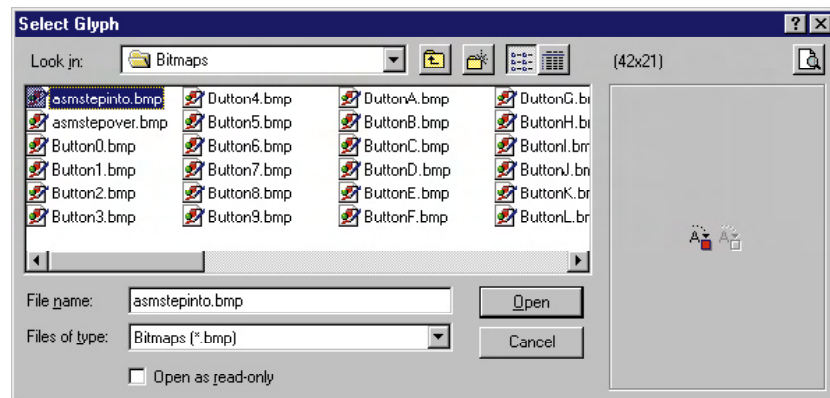
- 1) Buttons can be created and deleted on the icon bar in the main menu. A created button can be attached to a macro you have created. The icons themselves are Windows bitmaps (*.bmp) and are easily created with virtually any graphics program.
- 2) Open the menu Config, Buttons and an empty Figure 5 will open.
- 3) Open the bitmap browser by clicking on the ... button at the right side of the dialog box.
- 4) Figure 6 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 6.

Figure 5



- 5) Highlight ButtonW.bmp. Click on Open and its image will appear as in Figure 6 on the right side.
- 6) In the Hint: dialog box shown in Figure 5, 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.
- 7) 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 Cmdnd: box will change according to the Command Type setting.

Figure 6



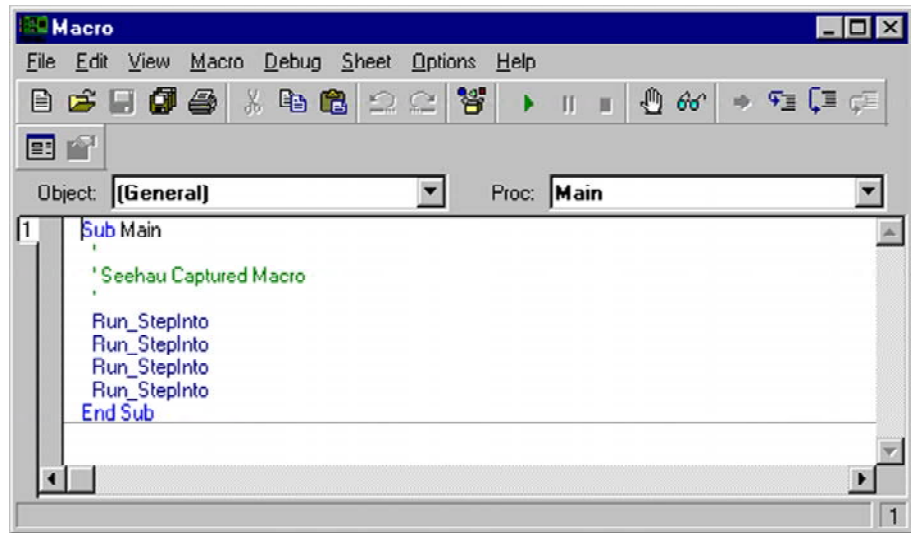
- 8) In the Cmdnd: dialog box, select startup.bas. You can choose the down arrow to open the Macro directory or use the browser by clicking on the ... button.
- 9) Click and drag the W icon to the main window to the right of the EXIT icon. The W icon will be placed at this point. Click on Exit in the Customize Buttons window shown in Figure 5.
- 10) Confirm that if you place the cursor on the W icon the hint you entered appears.
- 11) Click on the W icon in the main window and the startup.bas sequence will be executed.
- 12) 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.

Sax Basic Interpreter

Seehau has an integral Basic language interpreter called SAX Basic. It is a Visual Basic for Applications (VBA) compatible interpreter. Sax Basic is embedded into the regular Seehau command macros and is useful for creating your own dialog boxes. Here is a simple example.

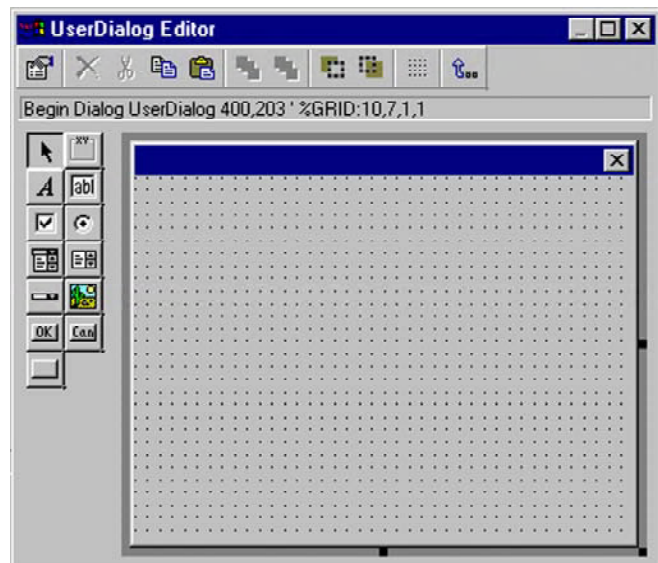
- 1) Load the timer program as before.
- 2) Open the Macro menu and select Start Recording.
- 3) Click on the Source Step Into icon 4 times.
- 4) Open the Macro menu and select Stop Recording.
- 5) When the macro save window opens give it a name. I called it captured.bas.
- 6) Open the macro window and click on Open and select the macro you just created. Figure 7 will appear.

Figure 7



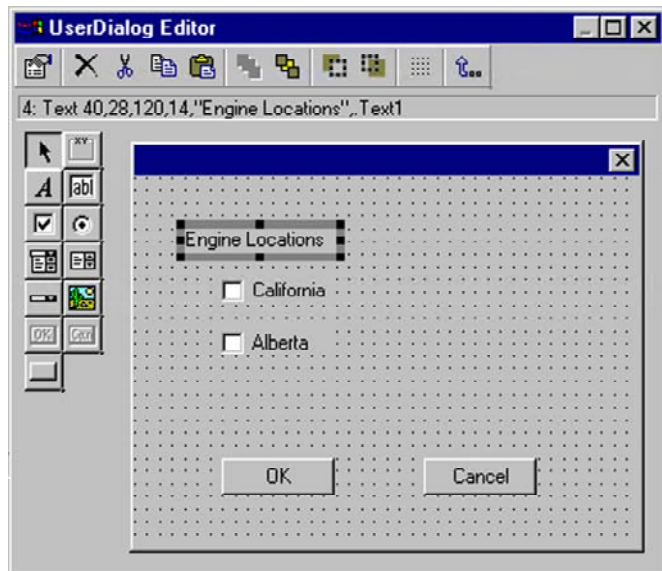
- 7) You can see the start of the macro which is constructed as a subroutine, the four Step Into commands and the end of the subroutine. Note the green triangle on the toolbar just beneath the word Help. This the GO button. The square to its left is the STOP button.
- 8) Place the cursor at the end of the second Run_StepInto line and press Enter to create an empty line.
- 9) Click on Edit and select UserDialog. Figure 8 will open up. This is where you will construct an example dialog window using the integrated Basic interpreter.
- 10) Hold the cursor on the toolbar on the left side and hints will be displayed giving the name of the button. We will use the OK, Cancel, Text and Checkbox buttons.

Figure 8



- 11) Selecting in turn the OK, Cancel, Text and Checkbox buttons, place one of each in the UserDialog Editor window similar to Figure 9. Do not add any others at this time.

Figure 9




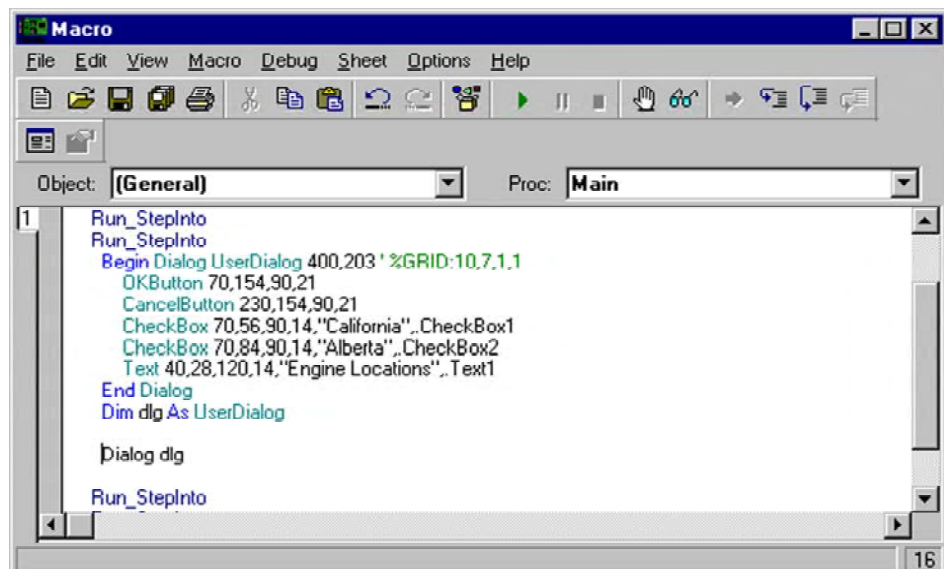
- 12) To add the text as shown, double click on each of the boxes and an entry box will open allowing you to enter the sample text in. In this window are << and >> buttons allowing you to click from one box to the next saving time.
- 13) Save this box by clicking on the Save & Exit icon.  The macro will now look similar to Figure 10.
- 14) Note the new Basic instructions that have been added to the original four Step-Into instructions. These instructions will display the dialog box in this example but any information you enter into it at run-time will be lost. Additional Basic instructions need to be added to accomplish this. The Basic Help in Figure 10 gives the syntax used and outlines can be cut and pasted into the macro editor window. The specific data such as variable names can then be entered reducing the amount of typing. See “Begin Dialog” in the Help files.
- 15) Click on the green GO triangle and the macro will run executing the first two Step Into commands and then stop and display the dialog box you created and will wait for your input. Click on OK in the dialog box and Seehau will execute the two remaining Step Into commands.

Figure 10



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. This description is for the 12, 16 and 25 MHz trace boards.

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.

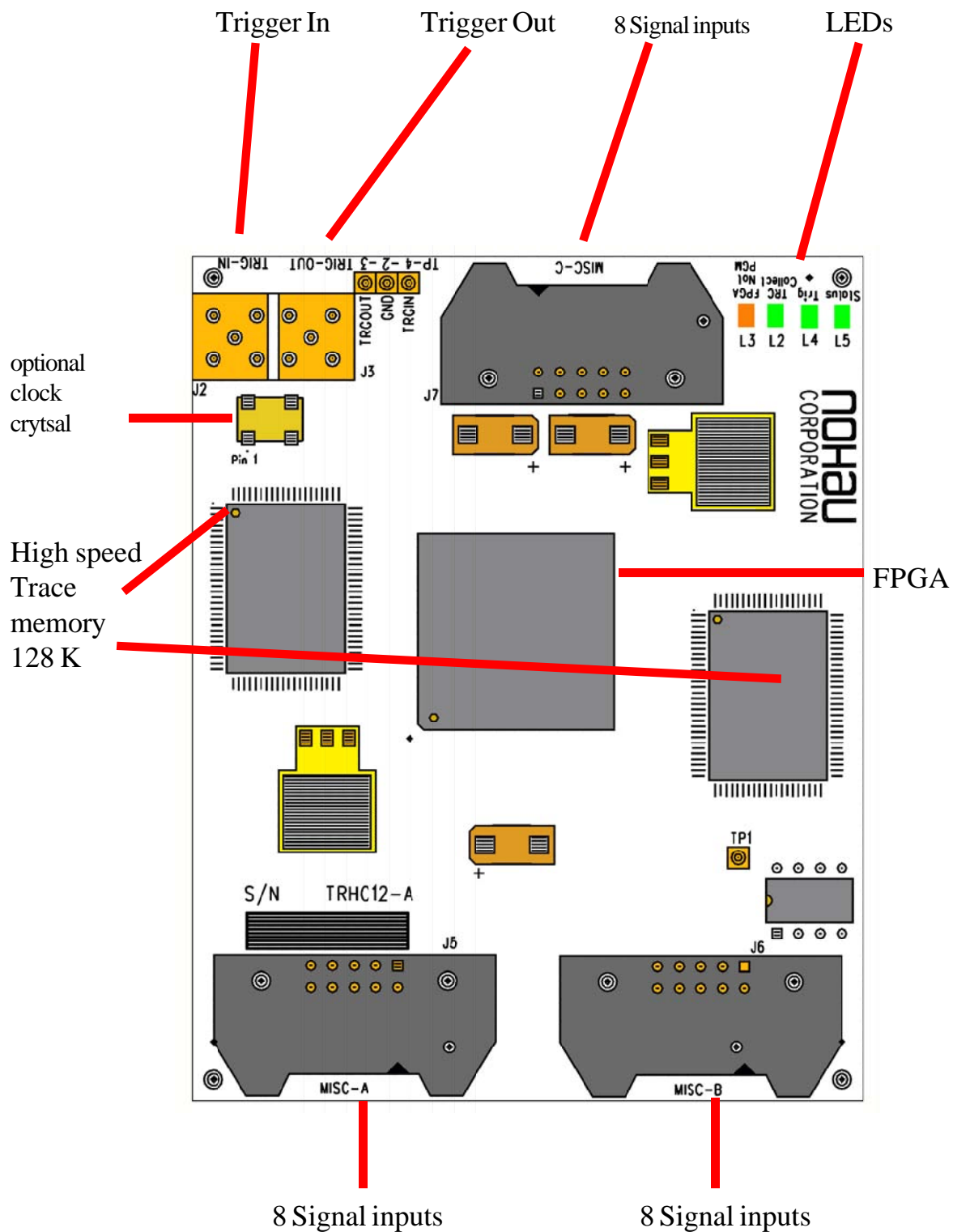


Figure 1
Trace Board

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.

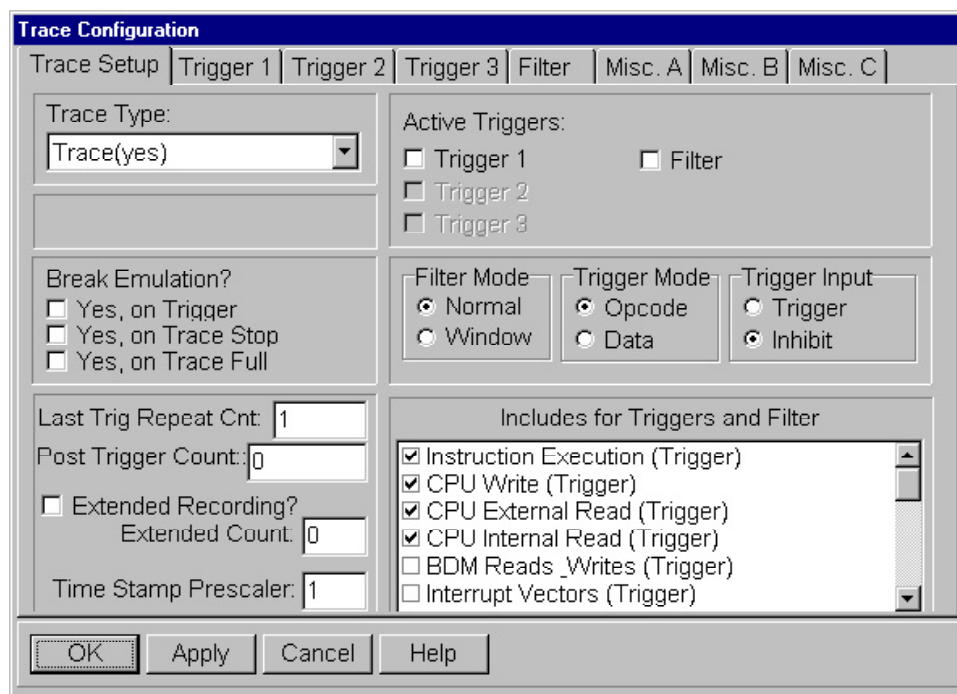
Break Emulation?

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:

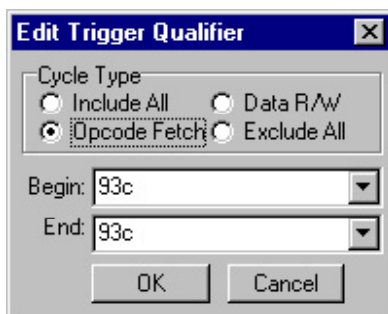
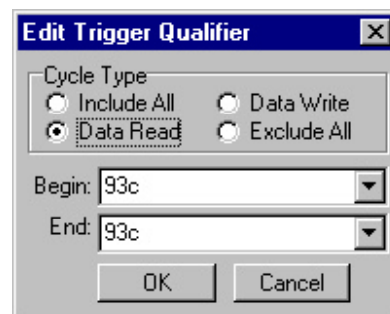
This specifies the options available in the dialog boxes that are available when qualifiers are entered in the trigger and filter menus. Select a Trigger or Filter tab and right click on the Address Cycle Type window and select ADD to see these menus that are shown in Figures 3 and 4.

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, 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 reset transition state is the start-up code the emulator executes that initializes the HC12 CPU immediately after RESET. This happens in the first 1000 cycles after RESET. The Reset Transition State checkbox must be activated in the Config. Emulator, in the Miscellaneous field and this is the default. When the Reset Transition State is active, it will be recorded in the trace memory if the Reset Transition Filter box is checked in the trace configuration Includes section. If the Reset Transition box is unchecked, the emulator will immediately execute the user code without any Nohau initialization sequences. Certain limitations will apply. Contact Nohau Tech Support.

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 BarTime.695 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.

Note: If you see “instruction exe word” (or byte) in the trace window: the emulator frequency is running too fast for the trace card. Reduce the clock frequency under Config, Emulator or purchase a faster trace card. It is normal to see “instruction exe” at the end of the trace buffer on occasion.

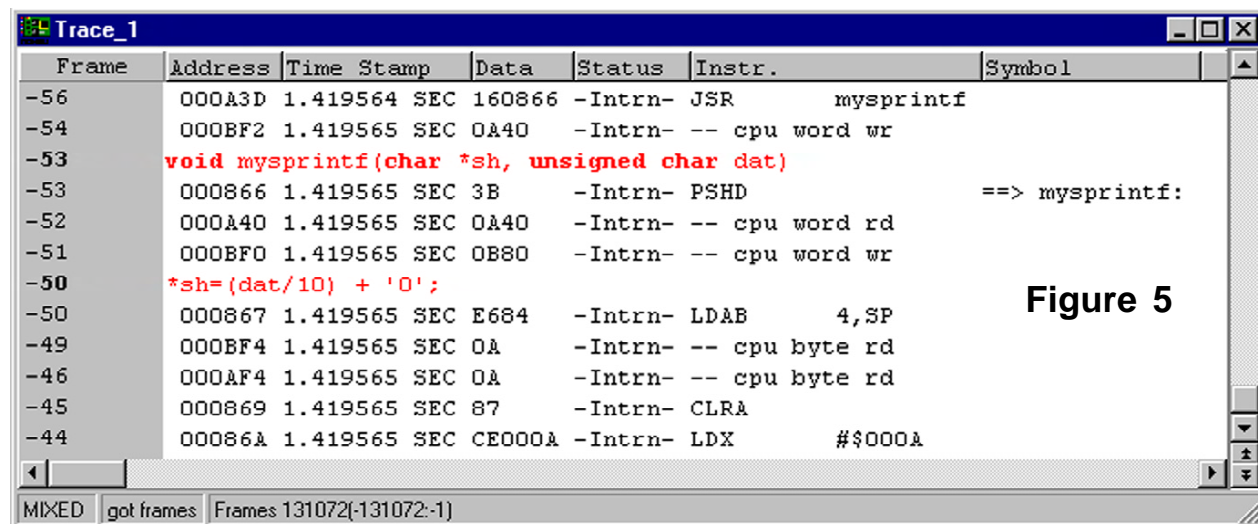


Figure 5

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 2387 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. In the "Includes for Triggers and Filters" window, check the box for "Fetch Cycles (Filter). Instruction fetches (not execution) will now be recorded in the trace buffer.
- 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 with 2387 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		
Trace Setup Trigger 1 Trigger 2 Trigger 3 Filter Misc. A Misc. B Misc. C		
Address Cycle Type	Start Address	End Address
Data R/W	2387	2387
< AND >		
Data Trigger Type	Low Value	High Value
Range	34	34
<input checked="" type="checkbox"/> Enable Trigger Data Mask: <input type="text" value="FF"/>		
<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 since this is a byte write.
- 8) In the Trace Setup tab, 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. See the note below.
- 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.

Note:

You can set and modify triggers, start, stop and view the trace and enable and disable the triggers in real time without stealing any cycles from the emulation of your target code.

In order to effect any changes you have made to the trigger settings, you must start and stop the trace but as mentioned, not the emulation.

If you click on the “Break Emulation?” box in the Trace Setup window, you might have to stop emulation for this feature to be activated. If you do this while emulation is stopped, the break will be correctly activated.

You can alert the emulator hardware to check for trace trigger breaks and hardware breakpoints by setting a dummy hardware breakpoint to an address never accessed (FFFE is suggested). Use the Breakpoint, Setup menu, right click and select Add New or hit the Insert key. Enter FFFE in the Address block and check the Hardware box. You can change the trace configuration and you do not have to stop the emulation to activate it. You just need to stop and start the trace recording.

Frame	Address	Time Stamp	Data	Instr.	Symb
-10	002082	999.976278	mSEC 306B	-- word fetch	
-8	00207F	999.976389	mSEC 1815	IDIVS	
-7	002084	999.976417	mSEC 70C6	-- word fetch	
-5	002081	999.977139	mSEC CB30	ADDB #\$30	
-4	002086	999.977167	mSEC 3A6B	-- word fetch	
-3	002083	999.977194	mSEC 6B70	STAB 1,Y+	
-2	002088	999.977222	mSEC 4031	-- word fetch	
0	002387	999.977278	mSEC 34	-- cpu byte wr	
1	002085	999.977306	mSEC C6	instruction exe start	

TRACE ONLY got frames Frames:131072(-131069:2), Trig Count:1

Figure 8

- 12) Note the trigger point at Frame 0. This is the byte cycle of the write of 34 to address 2387.
- 13) The instruction that caused this write was the STAB executed at Frame -3.
- 14) This STAB was fetched to the pipeline at Frame -10 (the 6B) and Frame -7 (the 70).
- 15) Frames -2 and -4 one half of -7 are flushed instructions. They represent the instructions starting at address 2086 and 2088. Frame 1 represents an opcode execution but is incomplete because of the emulation halt. This instruction represented here will be executed once emulation is started again and properly recorded in the trace memory. Note that instructions on the odd addresses are correctly displayed in the trace.
- 16) Change the filter Includes to see their effect on the trace window. The Free Cycles is interesting. You must rerun the program to refresh the trace.
- 17) Right click on the trace window and activate some of the columns such as the timestamp.

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 2387 in the trace memory.

- 1) Open the Trace Configuration menu in the Config menu item. Figure 2 will open.
- 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 2387 are recorded.

Note the timestamp. It shows that each write was 26.056 or 26.083 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.

Frame	Address	Time Stamp	Data	Instr.	Symbol
-18	002387	26.056	uSEC 34	-- cpu byte wr	
-16	002387	26.083	uSEC 34	-- cpu byte wr	
-14	002387	26.056	uSEC 34	-- cpu byte wr	
-12	002387	26.056	uSEC 34	-- cpu byte wr	
-10	002387	26.056	uSEC 34	-- cpu byte wr	
-8	002387	26.083	uSEC 34	-- cpu byte wr	
-6	002387	26.056	uSEC 34	-- cpu byte wr	
-4	002387	26.056	uSEC 34	-- cpu byte wr	
-2	002387	26.056	uSEC 34	-- cpu byte wr	

TRACE ONLY got frames Frames:38333(-38333:-1), Trig Count:0

Figure 9

COP Watchdog and RESET Sequences

The Nohau HC12 emulators operate through COP Watchdog resets and RESET sequence and will record this in the trace. You can also trigger on these events for debugging. For information regarding tracing on these COP Watchdog and RESET sequences, see the Nohau Application note available on the web at www.icetech.com/appnotes/cop.pdf or through Nohau Technical Support or your local rep. This appnote was also packaged with the emulator.

OSEK RTOS Support

Nohau can provide an OSEK RTOS debugger that operates with Seehau. Contact Nohau or your local Nohau representative for more information .

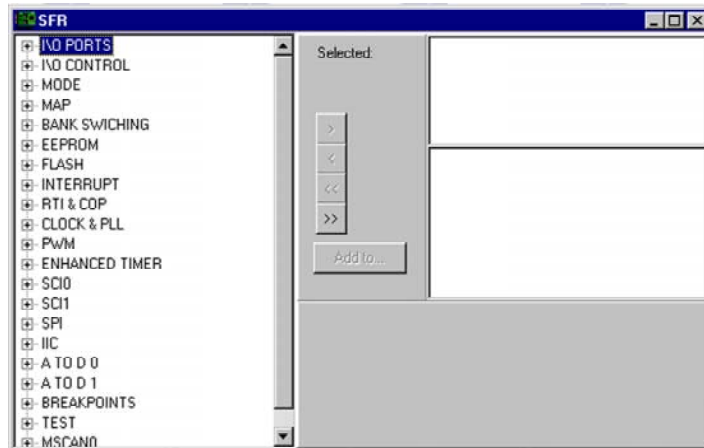
Chapter 6: Modifying the Register Window

Modifying INITRM:

This chapter shows you how to add the register INITRM to a Register Window to allow the modification of this register to move the internal RAM to a location specified by you at emulator reset time.

- 1) Right click on a Register window. Select Add Special Register. Figure 1 opens up.
- 2) Double click on MAP or click on its + sign to open this section up.
- 3) Highlight *INITRM and click on the ">" button to place INITRM in the box on the right side of Figure 1.
- 4) Click on "Add To..." and select Reg_1 or an appropriate register.

Figure 1

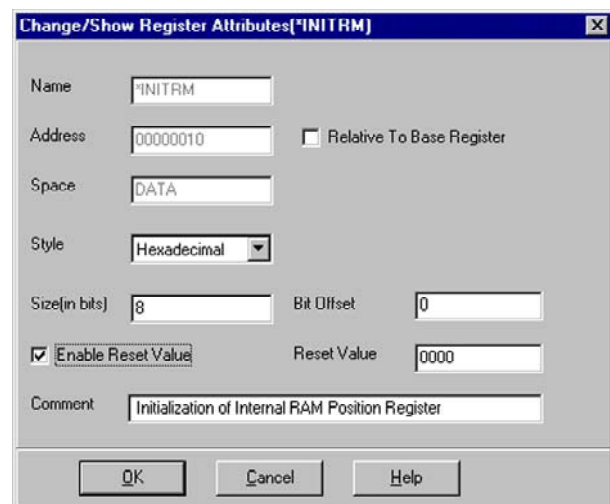


- 5) *INITRM will be entered in the specified register window.
- 6) Click on the X in the SFR Add window to close it.
- 7) Highlight to data number on the *INITRM register in the Register window and right click in the Register window.
- 8) Figure 2 will open up.
- 9) Click on the "Enable Reset Value" and enter 0 in the Reset Value box as shown in Figure 2. Click on OK.
- 10) Everytime the CPU is reset, INITRM will be loaded with zero.
- 11) You can configure register windows to your own preferences.

If you make a register name yourself and specify its physical address, you can select Shadow RAM or Run Time data to update this window while the emulation is running. In the box Space, specify which is desired.

If the Special Function lists are used as in the above example, this box is grayed out and no selection is possible.

Figure 2



Chapter 7: The Banking Example

Banking and the Motorola Next Generation Microcontrollers:

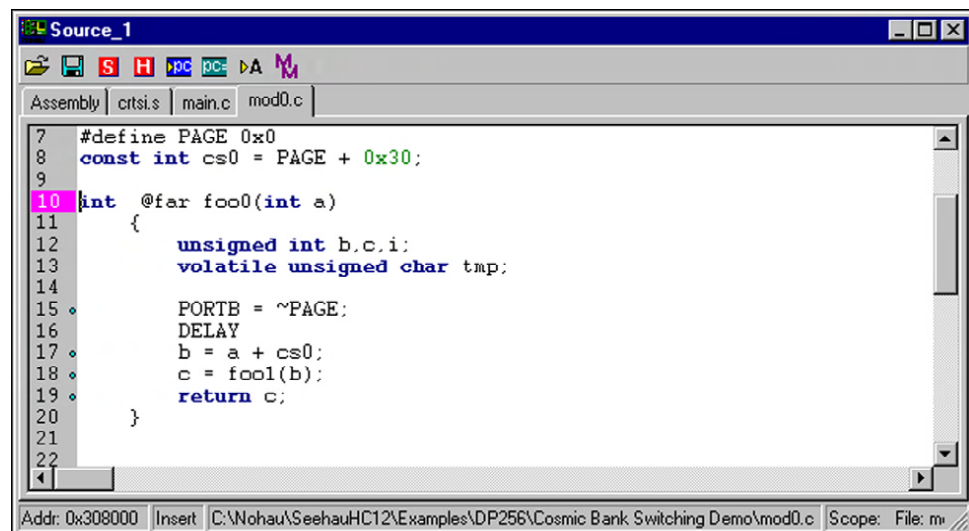
The Banking Example Program

Nohau provides a small banking example program called test.elf. It is also available as a IEEE 695 as test.695. These files, the source code and other compiler output files are in the `c:\Nohau\SeehauHC12\Examples\DP256\Cosmic Bank Switching Demo` default directory. This chapter shows you how to view the register PPAGE in a Register Window to show the program in various banks. You will also be shown how to display the page information in the trace window. The PPAGE register is already installed in the Register window by default in the Nohau Seehau software.

Running the Banking Example

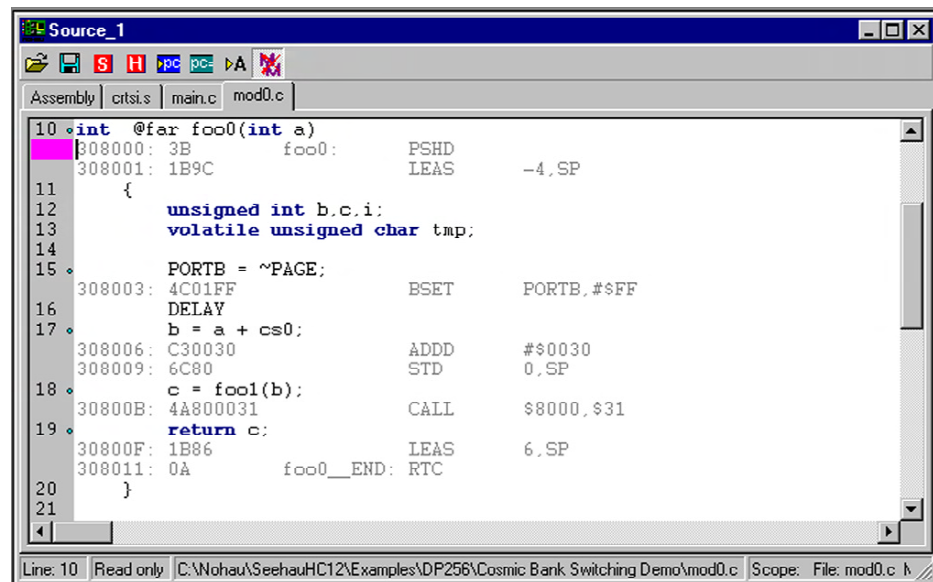
- 1) Start the emulator with a default startup.bas file. If you have set any configuration items in Seehau that might interfere with this demonstration, create a new startup.bas file by running the HC12 Config program.
- 2) Click on File, Load Code. In the Load window that will appear, select All Files, ELF Files or 695 depending on your preference to view the files. The .elf is the Motorola ELF-DWARF debugging file format.
- 3) Select test.elf or test.695 and click on OK. This file will be loaded into the emulator memory.
- 4) Open the Breakpoint dialog box under Breakpoints, Select and right mouse click and select Add New or press the INS key.
- 5) Enter the address foo0 and click on OK. Select either hardware or software breakpoints. Click on GO to run the emulation and the program will halt in the module mod0.c. Figure 1 shows the Source window.
- 6) This is the Cosmic example code where the program moves between the banks of memory. Line 18 is where the program switches from one bank to the next. After this particular line is executed, the next one that switch banks will be `c = foo2(b)`. Line 19 is where it will return successively to lower bank numbers.
- 7) Note the PPAGE register in the Register window is set to 30. This is the first page for the DP256 family. Seehau is compatible with the Motorola banking convention.
- 8) Note that a new tab opened up in the Source window called MOD0.c which is the source file for this part of the code. As you enter each module, its tab will be opened up. You can easily select between modules with simple mouse clicks.
- 9) To force the tabs to be opened without actually running that part of the program, right mouse click in the Source window and select File, Open File. Select as many as you want displayed and click on OK. All the tabs for the source files you selected will be placed in the Source window. You can also close any tabs you want in a similar fashion.

Figure 1



- 10) Click on the Mixed Mode icon in the Source window or right click and select Mixed mode. Choose At PC Line. Figure 2 opens up. The Source window now consists of the C source code with the instructions created by the compiler inserted below the appropriate C line.
- 11) Note that the machine address is the standard 4 hex digits (64Kbytes) with a two hex digit page number appended to the front of it. Line 10 is really address 8000 in page 30 or 308000. PPAGE in the Register window is equal to 30 at this time. The page number is also similarly displayed in the trace buffer. Seehau breakpoints and triggers are bank aware for maximum debugging power.
- 12) Note Line 18 is `CALL $8000, $31` which is the Motorola convention to switch to another bank. In this case the switch will be to bank 31.

Figure 2



- 13) Click on source Step Into or Assembly Step Into according to your preference until Line 18 is executed. PPAGE will change to 31 indicating the program is now executing from Page 31. The physical address at Line 10 will now be 318000. It is clear that unless breakpoints and triggers are bank aware, false triggering and breaking will happen whenever an address is encountered, regardless of which bank it is in.
- 14) Continue clicking on Source Step Over and observe PPAGE changing at the source line `c = foox(b);`. To get to page 3D faster, enter a breakpoint at `fooD` and click on GO. Please note that this entry is case sensitive.
- 15) Note that when in Page 3E and page 3F, the PPAGE register still reads 3D. This is correct. This compiled example accesses Page 3E and 3F at their fixed addresses i.e. 4000-7FFF and C000-FFFF. PPAGE will not be changed for these accesses unless they are accessed through the page window at 8000-DFFF. The Motorola databook has two excellent diagrams of the memory map under Resource Mapping. The "MC9S12DP256 Memory Paging" diagram is very useful.
- 16) Click on Source Step Into and observe PPAGE decreasing back to Page 30 as the code `return c;` is executed in the various pages.
- 17) You can step to a page of your choice or click on its tab and set a breakpoint. Run the program and note the breakpoints are page aware. Remove all breakpoints. You can use either hardware or software breakpoints.
- 18) There is a banking example version designed to be programmed into the FLASH memory. There is an example Nohau macro included which will load the demo program for FLASH into the emulator RAM. An example user startup file with sources is included. This is found in the Nohau HC12 Examples\Cosmic DP256 Best Practices Bank Switching Demo. An information sheet is located in this directory as a .doc file.

Figure 3

Frame	Address	Data	Instr.	Symbol
-35	3B8007	EC84	LDD 4,SP	
-34	003F91	0247	-- cpu word rd	
-32	3B8009	C3003B	ADDD #003B	
-30	3B800C	6C80	STD 0,SP	
-29	003F8D	0282	-- cpu word wr	
-20	3B800E	4A80003C	CALL \$8000,\$3C	
-27	000030	3B	-- cpu byte rd	
-26	000030	3C	-- cpu byte wr	
-25	003F8B	8012	-- cpu word wr	
-24	003F8A	3B	-- cpu byte wr	
-22	3C8000	3B	PSHD	
-21	003F88	0282	-- cpu word wr	
-20	3C8001	1B9C	LEAS -4,SP	
-17	3C8003	C6F3	LDAB #F3	
-16	3C8005	5B01	STAB _PORTB	
-14	000001	F3	-- cpu byte wr	
-13	3C8007	EC84	LDD 4,SP	
-12	003F88	0282	-- cpu word rd	
-10	3C8009	C3003C	ADDD #003C	
-8	3C800C	6C80	STD 0,SP	
-7	003F84	02BE	-- cpu word wr	
-6	3C800E	4A80003D	CALL \$8000,\$3D	
-5	000030	3C	-- cpu byte rd	
-4	000030	3D	-- cpu byte wr	
-3	003F82	8012	-- cpu word wr	
-2	003F81	3C	-- cpu byte wr	

Showing Banking Pages in the Trace Window

- 1) Delete all breakpoints and one at fooD (3D8000) unless it is already present from the previous exercise. Click on GO and the program will stop on this breakpoint.
- 2) Open the Trace window. A window similar to Figure 3 will open. The colour was changed to red for visibility but the other trace settings are the defaults. The red frames indicate that there is source code available. Right click on the trace window and select Display Mode and select Mixed. The source code and the associated instructions executed will be displayed. To see code as it is fetched, turn on the Fetch Cycles (Filter) in the Trace Configuration Includes section.
- 3) Note the addresses are 16 bits wide plus an extra byte at the beginning indicating the page number. This example shows the instructions being executed in pages 3B and 3C. There are some reads and writes in the uppaged memory. The last instruction executed was a CALL to page 3D at Frame 6 and this fact will be indicated by the contents of the PPAGE register.
- 4) Open the Trace configuration window under Config, Trace in the main window. You can enter various Includes such as the Fetch and Filter Filters. Click on GO to refresh the trace window.
- 5) Right mouse click on the trace window and select the timestamp and click GO to refresh the display. The timestamp will be displayed either in Relative or Accumulated time depending on the setting of the relative checkbox.
- 6) Remove all the features you activated or the screens in the next section will not match exactly.

Setting a Bank Aware Trigger

The triggers are bank aware. Triggers allow you to specify an event such as a instruction execute, fetch and interrupts as set in the Includes section (Triggers) in the trace configuration menu in Config, Trace. You can further set your event or events on data reads and data writes.

- 1) Open the Trace configuration menu and click on the Trigger 1 tab. An empty Figure 4 will open up.

Figure 4

The **Trace Configuration** dialog box has tabs for Trace Setup, Trigger 1, Trigger 2, Trigger 3, Filter, Misc. A, Misc. B, and Misc. C. The **Trigger 1** tab is active. It contains two tables. The first table, **Address Cycle Type**, has columns for Address Cycle Type, Start Address, and End Address. The second table, **Data Trigger Type**, has columns for Data Trigger Type, Low Value, and High Value. Below the tables is a checkbox for **Enable Trigger** (checked) and a **Data Mask** field set to **FFFF**. At the bottom are buttons for **OK**, **Apply**, **Cancel**, **Help**, **Refresh**, and **Clr Trigger1**.

Address Cycle Type	Start Address	End Address
Opcode Fetch	foo9	foo9

< AND >

Data Trigger Type	Low Value	High Value

☒ Enable Trigger Data Mask:

OK **Apply** **Cancel** **Help** **Refresh** **Clr Trigger1**

- 2) Right mouse click on the box under Address Cycle Type (where the words Opcode Fetch will be) and an empty Figure 5 opens up.
- 3) Enter foo9 in both boxes as shown in Figure 5 and select Opcode Fetch and click on OK.
- 4) Click on the Trace Setup tab in Figure 4 and in the Break Emulation ? box, select Yes, on Trigger. Click on OK to apply these values and close the window.
- 5) Click on GO to run the emulation and the emulator will run and then quickly halt.
- 6) The trace window will display frames similar to Figure 6.
- 7) Note the trace trigger point at Frame 0 which is the execution of the instruction PSHD at address 398000 which is in page 9. The PPAGE register equals 39.
- 8) The program counter equals 398009 which is a skid of 4 words. A trigger is by definition something that has already happened so does not stop before the instruction is executed.

Figure 5

The **Trigger Qualifier** dialog box has a **Cycle Type** section with radio buttons for **Include All**, **Opcode Fetch** (selected), **Data R/W**, and **Exclude All**. Below is the **Address Range** section with **Start** and **End** dropdown menus, both set to **foo9**. At the bottom are **OK** and **Cancel** buttons.

Cycle Type

☐ Include All ☐ Data R/W
☒ Opcode Fetch ☐ Exclude All

Address Range

Start: ▼
End: ▼

OK **Cancel**

Figure 6

Frame	Address	Time Stamp	Data	Instr.	Sync
-32	378009	367 nSEC	C30037	ADDD #0037	
-30	37800C	251 nSEC	6C80	STD 0,SP	
-29	003FB1	135 nSEC	019C	-- cpu word wr	
-28	37800E	116 nSEC	4A800038	CALL \$8000,\$38	
-27	000030	19 nSEC	37	-- cpu byte rd	
-26	000030	116 nSEC	38	-- cpu byte wr	
-25	003FAF	116 nSEC	8012	-- cpu word wr	
-24	003FAE	135 nSEC	37	-- cpu byte wr	
-22	388000	482 nSEC	3B	PSHD	
-21	003FAC	135 nSEC	019C	-- cpu word wr	
-20	388001	116 nSEC	1B9C	LEAS -4,SP	
-18	388003	251 nSEC	C6F7	LDAB #F7	
-17	388005	116 nSEC	5B01	STAB _PORTB	
-14	000001	154 nSEC	F7	-- cpu byte wr	
-13	388007	96 nSEC	EC84	LDD 4,SP	
-12	003FAC	19 nSEC	019C	-- cpu word rd	
-10	388009	367 nSEC	C30038	ADDD #0038	
-8	38800C	251 nSEC	6C80	STD 0,SP	
-7	003FA8	135 nSEC	01D4	-- cpu word wr	
-6	38800E	116 nSEC	4A800039	CALL \$8000,\$39	
-5	000030	19 nSEC	38	-- cpu byte rd	
-4	000030	116 nSEC	39	-- cpu byte wr	
-3	003FA6	116 nSEC	8012	-- cpu word wr	
-2	003FA5	135 nSEC	38	-- cpu byte wr	
0	398000	482 nSEC	3B	PSHD	
1	003FA3	135 nSEC	01D4	-- cpu word wr	

TRACE ONLY | got frames | Frames:529(527:1). Trig Count:1

Note the execution halted when the first instruction in foo9 was executed at Frame 0. As specified in the trace configuration window, this execution is in page 39.

The program counter equals 398009 which is a skid of 4 words. A trigger is by definition something that has already happened so does not stop before the instruction is executed.

Breakpoints, both hardware and software, do not skid. They do not execute the instruction they are set to.