



EMUL166-PC

Getting Started Manual

Version 2.5

Contents

Chapter 1 The Nohau EMUL166: The Hardware Parts	4
Some Background: the Nohau EMUL166-PC Emulator	4
The History of the EMUL166 Emulator:	4
The Two-board EMUL166 Emulator Design:	5
Emulator Pod Jumpers, LEDs, and Test Points:	8
Emulator LED's:	8
Emulator Pin-outs (in numerical order)	9
Edge Jumpers (in physical order)	10
Default Jumper Settings of Emulator POD -167N Rev C	12
Trace Board Jumpers and Plugs	12
Trace User Input Connector: (J3)	12
 Chapter 2 The Nohau EMUL166: The Software Parts	 13
The Nohau Universal User Interface Seehau	13
Configuring the Emulator Software Seehau	13
Important Software and Hardware Notes:	15
Starting the Emulator and Seehau	15
Problems ?	15
Shutting down Seehau:	16
Seehau Commands	17
Macro Construction	17
Macro Execution	17
Command Format	18
Command Examples	18
Command Groups:	19
Quick Saving a Configuration Menu using the Apply Button	19
Creating New Buttons	20
 Chapter 3 The Nohau EMUL166: Timer.abs Example	 21
Running the example program TIMER.ABS	21
Demonstrations of Seehau Features	22
Watching Data in Real-time with the Shadow RAM	22
Graphical Display of Data in Real-Time Using a Gauge	23
Graphical Display of Data in Real-Time Using a Graph	24
Setting Breakpoints	24
Program Performance Analysis (PPA)	25
 Chapter 4 The Nohau EMUL166: Trace and Triggers	 27
Trace and Trigger Overview	27
Trace Window Display	27
Trigger Example on an Address and Data Qualifier.	28
Trace Filter Example	30

Chapter 5 The Nohau EMUL166: Connecting to your Target	31
Clocks, Oscillators and Crystals	31
Crystal Capacitors	31
EMUL166 Oscillators	31
Connecting to Your Target Hardware	32
Conclusions	32
 Chapter 6 Debugging with the Siemens C167 E2 Bondout Microcontroller	 33
Introduction	33
Internal or External Mode	33
What the E2 Bondout Offers	33
 Note: Nohau will be making some enhancements to the trace capture and display. When this is completed, the screen shots described here will no longer match. This chapter will be rewritten at that time to incorporate the latest chnges and will be posted on the web. The software updates will also be available free of charge on the Nohau website.	
Special Bondout Buses	34
IA Bus: Instruction Address	34
ID Bus: Instruction Data	34
OA: Operand Address	34
OD Bus: Opcode Data	34
XBUS	35
Misc.	35
Instruction Pipeline	36
Pipeline Prefetch & Flushes	36
Jump Cache	36
Jump Cache Hit	36
Two for the Price of One:	37
Triggering on Internal Data	37
Two Chip Emulation	37
Single Chip Mode	37
Emulation Accuracy	38
Conclusion	38

Chapter 1 The Nohau EMUL166: The Hardware Parts

Some Background: the Nohau EMUL166-PC Emulator

The History of the EMUL166 Emulator:

The EMUL166-PC emulator is the second generation Nohau emulator for the Siemens C166 family of micro-controllers. The first model consisted of a two board design for the emulator and two trace boards in a sandwich design. One trace board was for internal memory and the other external memory. The bottom emulator board contained the Siemens E (E1) bondout controller and C167CR and C163 controllers that were used to provide the CAN and SSP X-peripherals. This emulator operated at a maximum speed of 25 MHz.

A later version contained the Siemens E2 bondout and since this controller contains a CAN controller, the C167CR was not needed. The C163 remained to provide the SSP serial port. The speed was increased to 33 MHz. The upper emulator board contained the emulator logic including two FPGAs that provided great flexibility in updating the firmware. These two emulator boards have been redesigned into a single board and it uses the E2 bondout. The C163 has been removed in favour of a daughterboard arrangement to accommodate X peripherals not found on the Siemens bondout controller.

The two trace boards allowed tracing on both internal and external memory areas and were optional. The external trace contained a trigger in and trigger out connectors plus 8 bit data inputs that were recorded in the trace memory. These boards have been redesigned into the single board emulator available today.

The older Nohau emulators are similar in functionality to the newer two board solution. Some features have been added and the weight reduced. The older emulators will operate with the new Seehau user interface as well as the older Windows software, but the two board model works only with Seehau. Both emulators operate with the pls fast-view66 debugger which provides additional features. This manual will work with any EMUL166 emulator although some jumper settings may be different. The differences between the various models occur with the clock prescaler jumpers and the PC/AUX jumper. These jumpers are not on all models.

Figure 1 shows the four board emulator connected to the Keil C167CR evaluation board. The EPC cable is also shown. The Emulator Parallel Cable (EPC) connects to the LPTx port of the host PC. An ISA card and cable is also available (LC-ISA). Various adapters are available to connect to virtually any target system. The EMULST10 emulator also supports the C166 and ST10 families up to 50 MHz. This emulator has advanced trace and trigger facilities for power users. This manual mentions the C166 family: the SAB-80C166 part itself is not supported by Nohau emulators. All other derivatives are fully supported.

Figure 1



The Two-board EMUL166 Emulator Design:

The current EMUL166 emulator consists of an emulator pod board with an optional trace board. The emulator board contains the Siemens E2 bondout controller and various logic in the form of advanced CPLDs. This board will operate as an emulator but without the trace, trigger or shadow RAM features. An optional second board, the trace memory, plugs on top of the emulator pod board and provides these features. The current maximum speed of both boards is 33 MHz. The photo on the front cover of this document shows the complete emulator in its case. The emulator is configured and operated by the Nohau user interface, Seehau. Seehau also contains the firmware for the CPLDs of the emulator. Firmware updates are easy and complete with each new release of Seehau. There are no separate firmware loaders necessary for Nohau products.

The emulator board has various jumpers, LEDs and test points to configure the emulator hardware to your specifications. Figure 2 is a photo of the top of the emulator board and shows some of the user jumpers, connectors, other features and the E2 bondout controller. Figure 3 shows the features in a line drawing.

Figures 4 and 5 show the trace board. The trace board does not have any user facilities available as they are all reflected on the emulator board or are software configured by Seehau. Do not change any jumpers on the trace board. The four jumpers that exist are used to configure various trace and trigger memory sizes. Since the memory chips are soldered in place, there is no need to change these jumpers which may have zero ohm resistors installed. Trace options are all software configured by the Seehau interface.

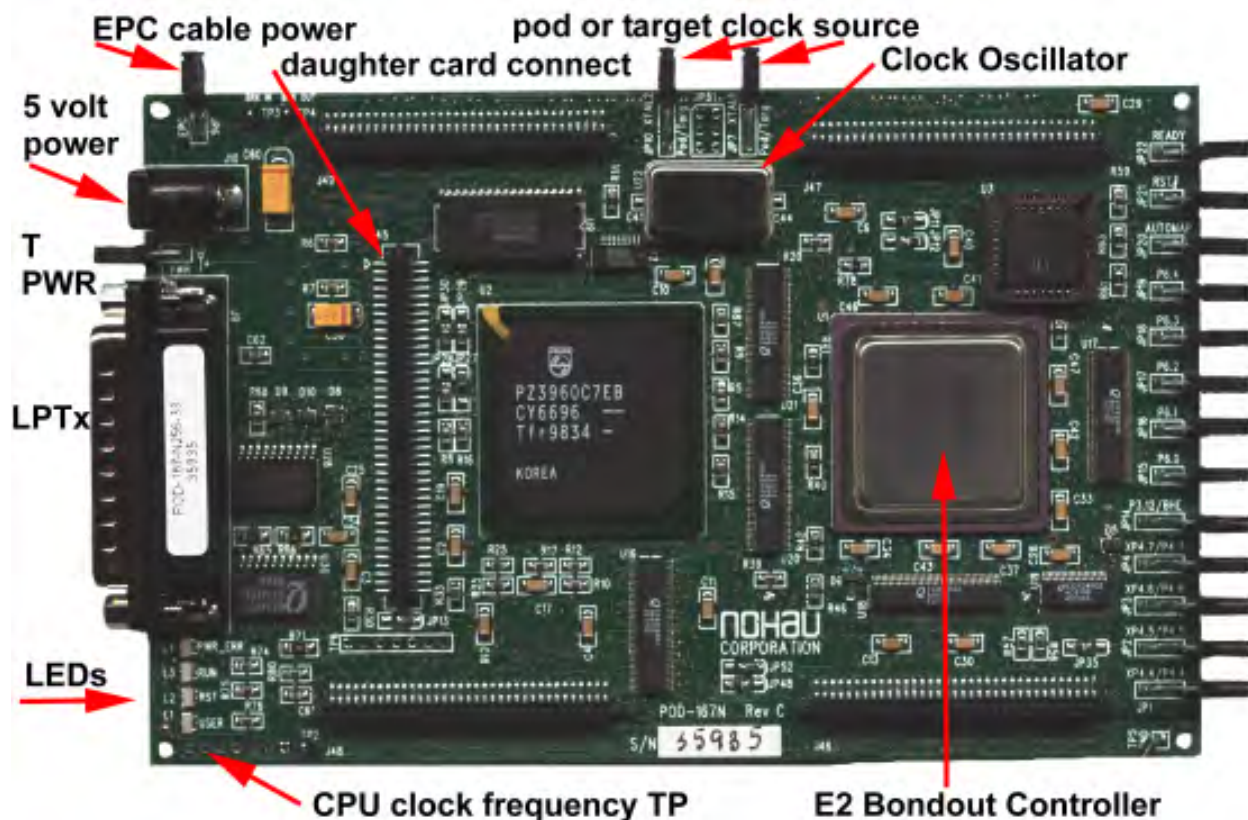


Figure 2
Emulator Board

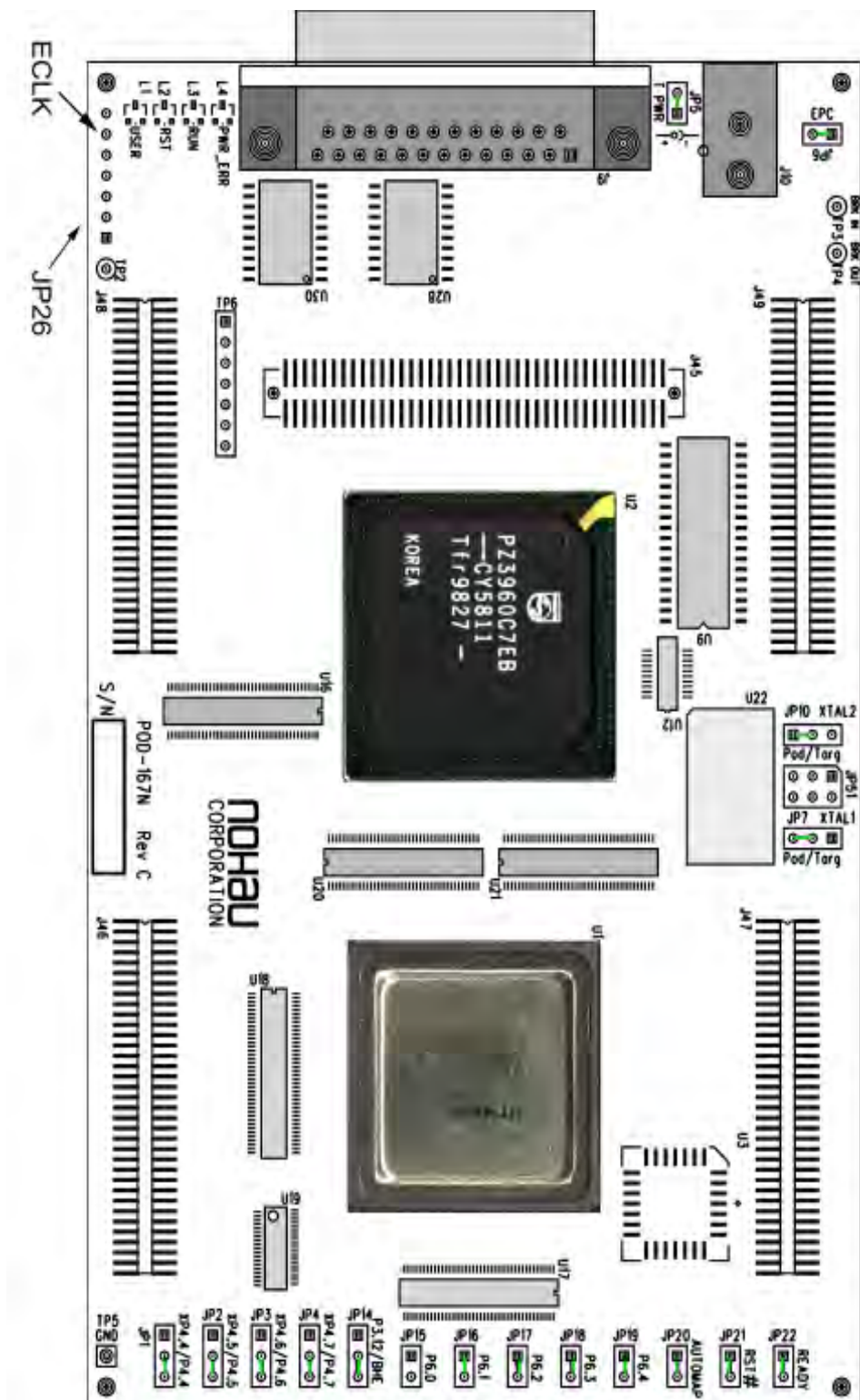


Figure 3
Emulator Board

On the bottom of the emulator pod are 5 connectors in an industry standard configuration designed to connect the bondout emulation controller to the target hardware. This connection can be direct with the appropriate connectors designed on the target board, or through various adapters available from Nohau. Figure 6 is a photo of the bottom of the emulation board with the bottom case installed. Note the 4 large connectors (J5 to J8) and a single small one (JP46). JP46 routes XBUS peripheral signals from future derivatives to be sent to the target. Maximum system speeds are maintained by not having these signals switched by logic. These signals will normally be routed physically by the adapter or the target board.

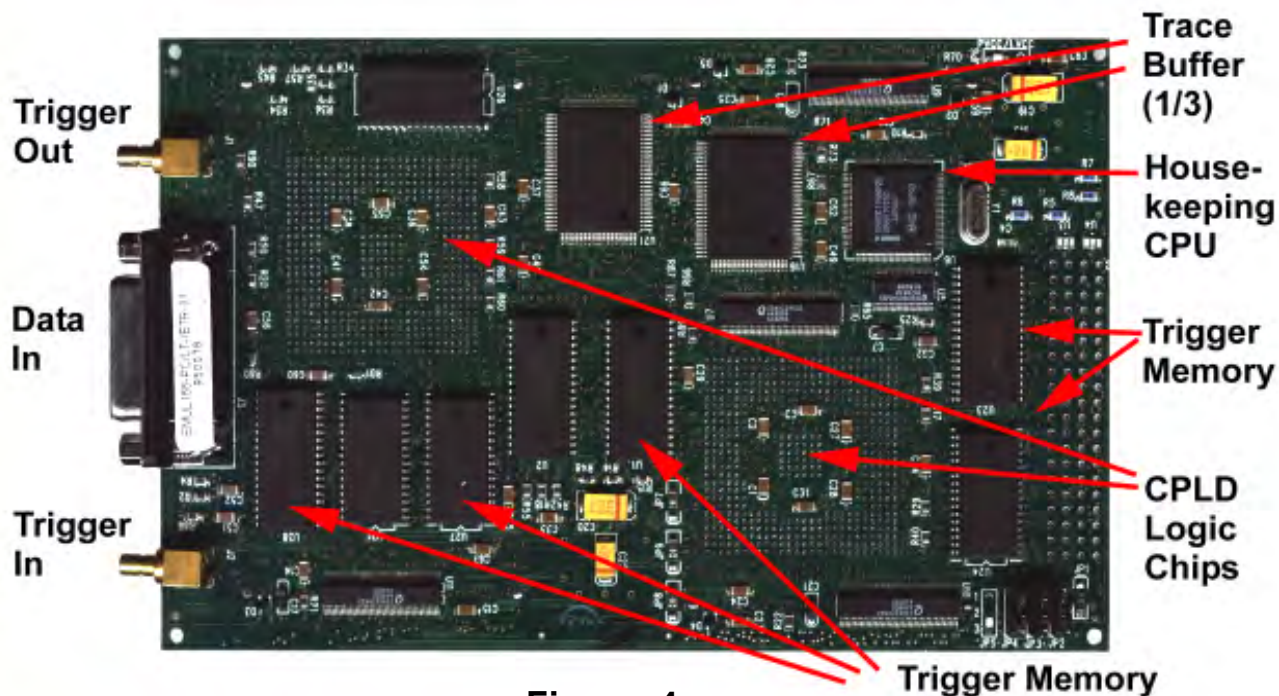


Figure 4
Trace Board

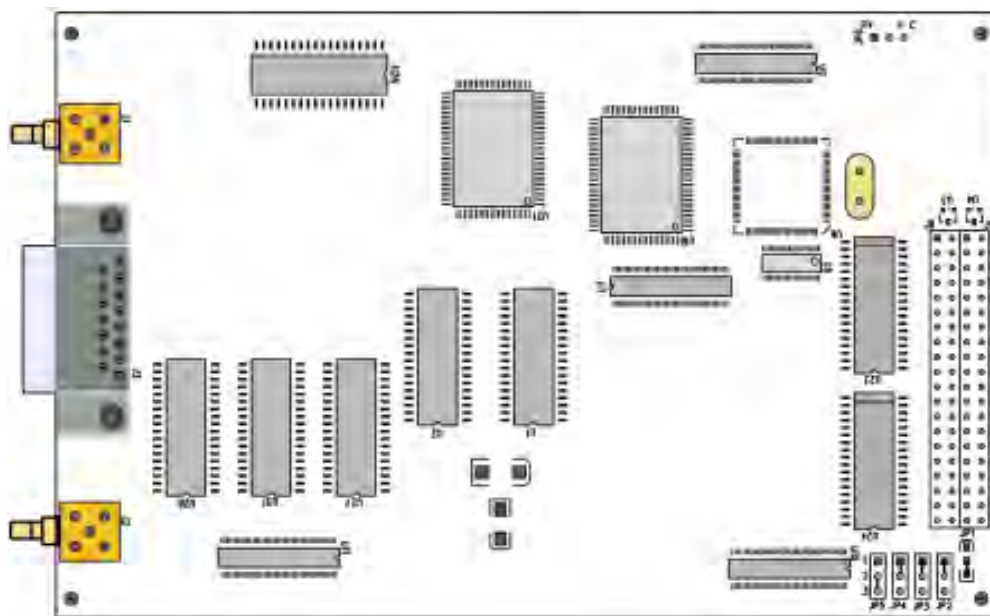


Figure 5
Trace Board

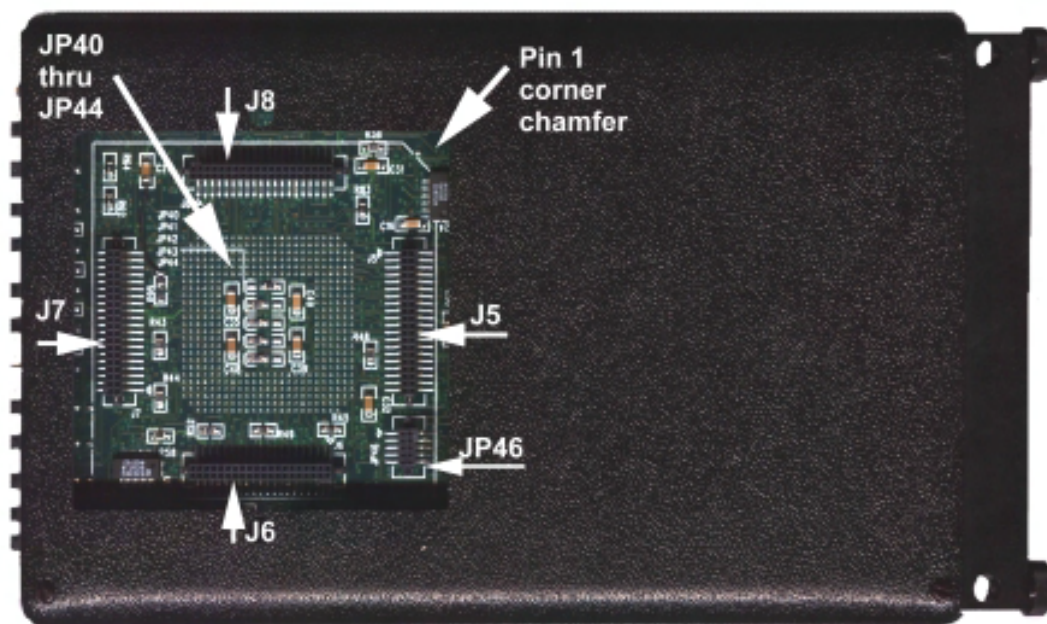


Figure 6
Target Connectors

Emulator Pod Jumpers, LEDs, and Test Points:

This section is based on the POD167 Revision C. This information is clearly marked on the upper side of the board between the trace board connectors J48 and J46. This is the edge opposite the two clock jumpers. This is shown just above the serial number location in Figure 3. Figure 2 and 3 show the location of the various jumpers and test points. Jumper information is also available in the Seehau on-line help. There are no user jumpers for the trace board.

Emulator LED's:

Normal operation with emulation stopped has all LEDs L1 to L4 off. With a user program running, L3 (RUN) will go on. The red RESET LED L2 lights when Seehau is performing a system reset.

L1 USER

This is a yellow LED the user can affect by applying standard TTL levels to TP2. TP2 is active low - grounding TP2 turns on L1.

L2 RST - RESET

This red LED indicates the application of a reset signal by the Seehau software. The emulator will be in the reset state when powered with 5 volt supply but without a cable connected to the PC connector J9.

You can reset the emulator by selecting Reset Chip in the Run menu or clicking on the RESET icon on the toolbar. The commands `run_reset` and `run_fullreset` will also reset the emulator. These commands can be entered in the command dialog box or through a Macro.

L3 RUN

This green LED indicates whether the Nohau monitor program or a User program is running. L3 illuminated means the user program is running.

L4 PWR_ERR - Power Error.

If this LED lights, the correct 5 volts supply is not connected. This can occur if the 5 volt regulated supply is not connected to J10 and the PC communications connector J9 is connected. L4 is a red LED.

Emulator Pin-outs (in numerical order)

TP1:

Not present.

TP2:

This input activates LED L1 USER. TP2 is active low: this action turns L1 on.

TP3: Break In

This input will cause a break in program execution. TP3 is usually connected to TP4 Break Out from another EMUL166 emulator. This feature is used to synchronize more than one emulator in a multi-processor system. Contact Nohau technical support at support@nohau.com for more information.

TP4: Breakout:

This output indicates if the emulator is in monitor mode or is running user code. It is connected to the RUN LED L3. This output is usually used for customers who have multiple emulators connected to the target board. It provides a means to synchronize them together. TP4 is high when the Nohau monitor is running.

TP5: GND

The emulator ground connection and a black ground wire with a clip is soldered to this point. Connect this clip to a secure ground (or earth) connection on your target hardware. Do this before connecting the emulator processor pins to the target.

TP6:

This is for Nohau testing. There is no user use for this test point.

JP7 and JP10:

XTAL1 and XTAL2 - These jumpers determine whether the on-board pod clock (U22) or the target clock is used for emulation. These jumpers must be changed in tandem as a pair. The default is set to POD (the upper positions) and U22 provides the clock signal to the POD controller. The Pod will supply the clock to the target if you jumper all three of the pins together. In this case, make sure the target clock is disconnected.

JP51:

Reserved for PLL clock and oscillator board. See chapter 5 for more information.

JP5: T PWR

+5 volts is supplied from the pod power supply through JP5 to the bondout controller. During normal operation with a target board, 5 volt power for the bondout controller U1 will be supplied by the target and JP5 must be removed. The pod power supply must still be connected in order to supply power for the rest of the emulator.

JP5 is used for standalone operation i.e. without a target board. JP5 will send power to the target if left jumpered. If the target has power, this can cause a conflict so JP5 must be removed. If the target has no internal power supply, the pod is able to supply a small amount, less than 500 ma, to power the target. This may be useful for very small targets with low current consumption.

Note: 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 bondout controller can damage it. Always power up the pod first, and power it down last. Do not leave the Pod powered up without the target power applied for more than 2 minutes. This situation stresses the I/O circuitry of the bondout chip and will cause permanent damage.

Do not leave the pod powered without the SeeHau software running it. The pod will be in an uninitialized state with unpredictable results. There are no reports of this causing damage.

Power the emulator by switching the AC mains power on after connecting the 5 volt plug to J10. Turn on transients caused by the 5 volt plug bouncing may lock up the bondout and CPLD chips. It is convenient to switch AC power on and off using a power bar rather than by inserting the DC plug into J10.

JP26:

This unpopulated connector is used for testing the pod by Nohau. Pin 6 (ECLK) can be used to measure the CPU clock frequency. The C166 family has a clock prescaler and a PLL on chip providing scaling of oscillator U22. This pin is the second from the end of the Pod board closest to L1, the USER LED.

Pin 7 (EMUL#) indicates if the Pod is in USER or MONITOR mode. This pin is closest to the edge of the board. This signal is connected to the RUN LED. The rest of the pins provide no useful information.

JP6:

EPC PWR - JP6 supplies the EPC cable (Emulator Parallel Cable) with 5 volts. If this special cable is not used, remove JP6. The LC-ISA board does not need power from the pod and JP6 must be removed.

JP13:

This 8 pin connector is used by Nohau to program a serial EEPROM. It has no user use.

J45:

This connector will connect to a small daughterboard to supply XBUS peripherals not supported by the bond-out controller. This allows the emulation of derivatives not yet introduced and prevents obsolescence of the emulator. Some of these signals will be sent to the target through JP46 on the underside of the Pod. See Figure 6. JP1 through JP4 control the path of signals from J45 or from the bondout controller U1.

J46, J47, J48 and J49:

The optional Trace memory board connects to these four connectors.

J5, J6, J7 and J8:

Connects the bondout controller signals to the target. This is an industry standard connector and can be directly connected to appropriate target boards or through various adapters available from Nohau.

JP46:

This 10 pin connector is used to send XBUS peripheral signals from the optional daughterboard to the target. This connector and J5 through J8 are shown in Figure 6.

JP40 through JP44: (see Figure 6)

These five jumpers have zero ohm resistors soldered in them. These jumpers direct logic flow for the five C166 chip selects. Normally, you do not need to change these jumpers. Contact Technical Support.

Edge Jumpers (in physical order)**JP22: READY:**

JP22 connects the READY input signal between the target and the POD controller. The READY signal is used to terminate a bus cycle and is useful when external devices have a long or indeterminate access times. READY is activated in certain address ranges as determined by the BUSCON registers.

If the READY signal is not asserted for a bus cycle that it is activated for in BUSCON, the processor will stop. Only a reset or a watchdog timeout can restore operation at this point (or the assertion of READY). Since the bondout controller on the pod is used for some housekeeping when emulation is not running, suspending it can cause Seehau to stop responding. Remove JP2 to prevent this. The default is connected.

JP21: RST#

JP21 connects the Pod and target reset pins together. The RST is the master reset pin. A target peripheral with the ability to assert this signal may not be desirable. Remove the jumper on JP21 to prevent the Pod controller from being reset from the target. A reset on the Pod controller will not be passed to the target board if this jumper is not connected. JP21 is connected in by default.

JP20: AUTOMAP:

Determines whether memory mapping is controlled by software with Seehau or the chip select pins setup contained in the ADDRSEL1 through ADDRSEL4 registers. The MEM Map Config window shown in Figure 20 is in the Config menu. AUTOMAP determines whether the numerical addresses specified in the window or the chip selects selected are passed to the target board. If JP20 is connected, then software mapping is activated. Since there are no entries in the mapping window, all memory is mapped to the emulator. If JP20 is removed, then mapping is determined by the chip select jumpers and the BUSCON and ADDRSEL registers. The default is JP20 connected.

JP15 through JP19:

The controller provides 5 programmable chip select pins (CS0-CS4) that are output on parallel port 6 if it is programmed as an Alternate Function. These are pins P6.0 through P6.4. Recall that if these pins are not used as chip selects, they can be used as general purpose I/O ports.

JP15 through JP19 connect these 5 pins to the target. They create the mapping signals from the chip selects as in previous Nohau C166 emulators. The default is JP16 through JP19 connected. JP15 not connected.

JP14: P3.12 or BHE:

If you configure P3.12 as an I/O pin, place JP14 on the P3.12 side (upper position). If you configure P3.12 as either BHE or WRH, place JP14 on the BHE side (lower position). The default is BHE (lower position).

In the process of mapping sections of memory to the target, the RD, WR and BHE signals are not passed to the target directly. The mapping function of the emulator switches off the pod memories as required. However, as the signals map memory access to the pod, it gates off the RD, WR and BHE signals from the bondout controller to the target.

When P3.12 is configured as a standard I/O pin, you would not want it gated off by the pod logic. When the jumper shunt is in the P3.12 position, it passes directly from the bondout controller to the target. When the jumper shunt is in the BHE position, P3.12 gates off and pulls high during memory accesses to the pod as required. The default is BHE connected (or the lower position).

JP1 through JP4:

XP4.4/P4.4 through XP4.7 through P4.7: These jumpers select whether the upper 4 bits of Port 4 supplied to the target come from the bondout controller or from a daughterboard connected to J45.

The XBUS is an internal representation of the external bus. This bus is used internally to connect various peripherals to the CPU. These peripherals look like external devices to the CPU even though they are on-chip and are accessed as such. Internal peripherals are also on the XBUS.

The bondout controller U1 used for emulation contains most peripherals. If a certain Xbus Peripheral is available on the bondout chip on the pod, then this peripheral will be used to provide the peripheral to the target. For those derivatives that use peripherals not on the bondout controller, a daughterboard connected to J45 will contain a regular series controller that contains this XBUS peripheral. JP1 through JP4 must be set so these signals are sent from the daughterboard to the target. (XP4.4 through XP4.7 - the upper positions) The default is all jumpers are set to P4.x (the lower positions).

Default Jumper Settings of Emulator POD -167N Rev C

This list is for stand alone operation without a target connected. Do not change any trace board jumpers.

POD board

JP7 and JP10 upper position (Pod). This selects the emulator clock and not the target clock.

JP5 on. This is T PWR and supplies 5 volts to the bondout controller.

JP6 off (on if using the EPC cable)

The 13 jumpers on the edge of pod board are all connected except for JP15 (P6.0). If a jumper has 3 pins (JP16 to JP20), the position is lower or close to the circuit board.

Trace Board Jumpers and Plugs

JP2 thru JP5 these jumpers may have zero ohm resistors installed. Please do not change these jumpers. They configure the trace and trigger memory size and since you cannot change the memory chips

JP6 PVCC/LVCC This jumper is used to select the operating voltage of the trace board. PVCC is for 5 volts and this jumper must not be changed.

JP6 selects power voltages and your pod may not survive. There is no warranty for this easily identified condition.

Trace User Input Connector: (J3)

The trace board contains a 15 pin D shell connector. Refer to Figure 4 for its location. This connector provides 8 user input signals to be recorded in the trace memory under the Misc. column. These are TTL level inputs. The Trigger In (J2) is also available on this connector. A special socket with clips is available from Nohau as part number TR167 Probe. The pin assignments are as follows:

Name	Pin	Lead Colour of Trace Probe Cable Assembly
External In 0	2	black
External In 1	3	brown
External In 2	4	red
External In 3	5	orange
External In 4	6	yellow
External In 5	7	green
External In 6	8	blue
External In 7	9	violet
Vcc 5 volt	1	not connected in Trace Probe cable assembly
Trigger In	11	red clip -white wire
Trigger Out	13	green clip -white wire
Ground	15	grey

Chapter 2 The Nohau EMUL166: The Software Parts

The Nohau Universal User Interface Seehau

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

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

Configuring the Emulator Software Seehau.

The commands and data used to configure Seehau when it is started is contained in the file startup.bas. The file startup.bas is an ASCII file in the default directory c:\nohau\Seehau16x\Macro. This file is created by the Seehau Configuration program using user supplied information about the emulator and its environment. It is not created when Seehau is initially installed. The file seehau.ini in the c:\nohau\Seehau16x directory specifies this startup file.

The configuration can be started by clicking on the Seehau Config icon on your desktop. If you start Seehau itself and startup.bas does not exist, Seehau will start the configuration program. This is a good method to totally reconfigure the emulator software. Simply delete or rename startup.bas and you can create a new one.

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

You can access the emulator and trace setup windows from within Seehau under the Config menu item in the main window. Click on Emulator or Trace for the appropriate setup desired. Note that more detailed setup options are available this way.

This manual assumes you are able to load the Seehau software. Make sure this is done now. If you have trouble loading from diskettes, copy them to a temporary directory on your hard disk and install from there.

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.

- 1) Click on the Seehau Config icon on your desktop. You do not need the emulator connected at this time.
- 2) A blank Figure 1 will open. You will now configure the emulator. You will need to know what interface connector you are using: EPC or the LC-ISA card.
- 3) Change the settings as indicated. Make sure you always select an E2 selection unless you have an E1 bondout controller on the original emulator series. The EPC cable and the LC ISA card are the only appropriate choices for the EMUL166. Figure 1 shows the settings used if you have are using the EPC cable. The EPC cable is distinguished by a male/female connector on the end that plugs into your PC parallel port. Figure 2 shows the settings for the LC-ISA card. The Emulator Board Address dialog box is for the address of the internal communication link from your computer. For the ISA card, the most common address is 200. This setting can be changed on the board. If you are using the EPC (Enhanced Parallel Cable), this address is not applicable. The EPC cable normally uses address 378 which represents the LPT1 port on your PC. It can use other LPT addresses as well.

When all the information has been entered in Figure 1 or 2, pressing Next will open up Figure 3. It is possible to operate the emulator remotely via a TCP/IP stack. Contact Nohau Technical Support at support@nohau.com or (408) 866-1820.

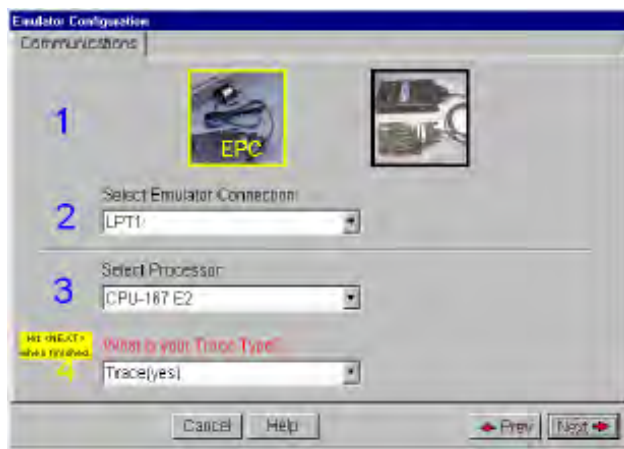


Figure 1

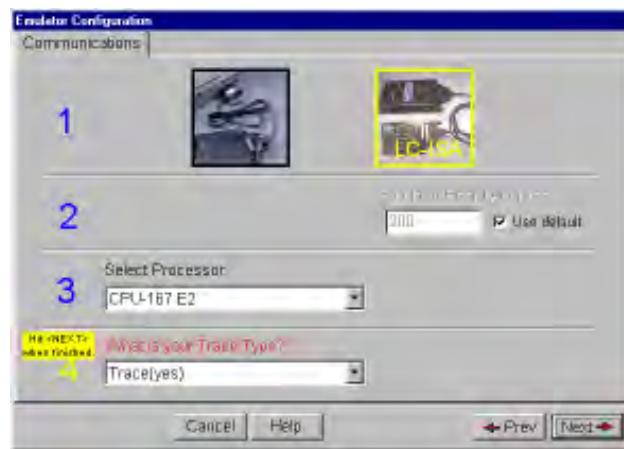


Figure 2

uP Clock: The internal CPU clock. This is usually the oscillator or crystal multiplied by 4 if the controller is in PLL mode. If you have a 5 MHz crystal installed, the CPU frequency will be 20 MHz. This setting is used only for the calculation of the Trace timestamp. It has no effect on the operating speed of the emulation controller.

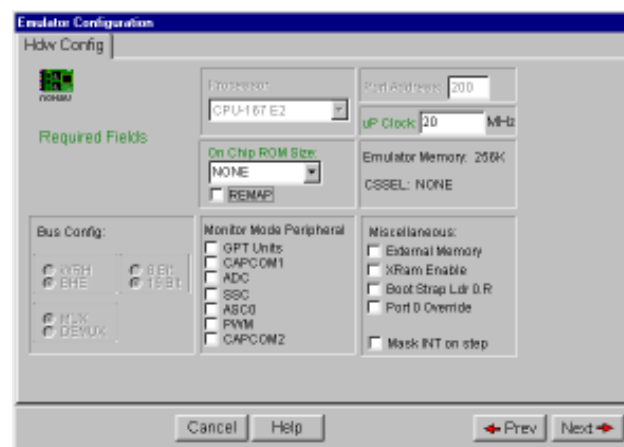
Processor: This is set to C167-E2 or C167CR-E2. This can be changed only through the configuration program or be editing startup.bas. Only early models of EMUL166 have an E1 bondout.

ON Chip ROM size: this tells Seehau the size of the on chip ROM or FLASH memory in your controller. We do not have any ROM here since we have no target so set this to NONE. (default)

- 4) Confirm your settings are the same as in Figure 3. Set the frequency to 20 MHz. This entry only effects the trace timestamp. It does not effect any other part of the emulator.
- 5) There are plenty of interesting configuration settings here. See the online help for more information.
- 6) Click on Next to enter the data. When Seehau will ask if you are done, click on Yes.
- 7) Seehau configuration program creates Startup.bas and Seehau is now configured to run your emulator.
- 9) The Seehau configuration program shuts down. Seehau is ready to run the emulator.

If all this completed without any errors, you are ready to run the Seehau User Interface after you have connected and powered up the emulator EMUL166-PC. Do not connect the power supply to the emulator yet. The emulator is powered by 5 volts regulated at the standard power supply socket J10. The center pin is positive. The emulator jumper settings will be the default for this part of the manual.

Figure 3



Important Software and Hardware Notes:

Always use Uninstall to unload any existing version of Seehau from your computer before loading in another copy. Do not simply delete the Seehau files and/or folders. Do not install Seehau on top of an existing copy. Seehau will not allow this action. Under My Computer select Control panel, then Add/Remove Programs. After Uninstall has run, you might want to delete any files and folders that Uninstall did not delete. Save your personal macros and source files first. Pay particular attention to startup.bas in the Macro directory. You may not want to use your old copy of this file so make sure it is erased.

Do not leave the emulator powered for extended periods of time without Seehau active. The pod will be in an uninitialized state with unpredictable results. There are no reports of damage resulting from this.

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 bondout controller will damage it. Power up the pod first, and power it down last.

Starting the Emulator and Seehau

- 1) The two clock jumpers JP7 and JP10 must be in the upper position i.e. "POD".
- 2) T Pwr (JP5) must be connected. This jumper supplies power to the bondout CPU. This jumper is located between the PC connector and the power supply socket.
- 3) If you are using the EPC, the EPC power jumper (JP6) must be connected to send power to the EPC. This jumper is located beside the power socket opposite side of the T Pwr jumper. If you are using the ISA card, this jumper must be left open or the power from the ISA will conflict with your power supply.
- 4) All the 13 jumpers on the end edge of the board must be connected except for JP15 (P6.0 or CS0). For the 5 that have dual pins, the position is lower or close to the PC board. All these jumpers may not absolutely be needed in order for this example to work.
- 5) Connect the cable between the PC and the 25 pin connector on the emulator board.
- 6) It might be a good idea to double check your settings.
- 7) Plug in the Nohau 5 volt power supply.
- 8) The green RUN and perhaps the red RST LEDs will illuminate with the EPC cable connected.
- 9) Double click on the Seehau 16x icon on your PC desktop.
- 10) Seehau will configure itself from startup.bas and may present this message asking if you want to reset the emulator. The green RUN LED is probably on: Click on Yes. This gives you the opportunity to reconnect to a running emulator if you had shut down Seehau previously. "Mac" will be displayed in red at the bottom of the main window while startup.bas is running.
- 11) The red Reset LED will then light and then both it and the Run led will go out. The Seehau interface will finish loading itself. Position and size the main window to your preference. You can open up new windows. They are found in the New menu item on the main Seehau window.

Problems ?

Problems are usually from the EPC PWR, T PWR or the clock incorrectly connected. Review instructions 1, 2 and 3. Make sure the cable to the PC is correctly connected. If are using the EPC, try it without a printer connected. Do not have the emulator connected to a target system or adapter. You should have a 5 MHz oscillator module installed to get a 20 MHz CPU frequency. Set your PC parallel port for "Output Only" in the BIOS setup. It may not work well in ECP or other modes. Turn on the 5 volt supply by connecting the 5 volt cable to J10 first, then connect the mains AC power. This creates fewer turn-on transients.

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

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

I set my windows up as in Figure 4. I resized the existing windows and opened up the Trace window.

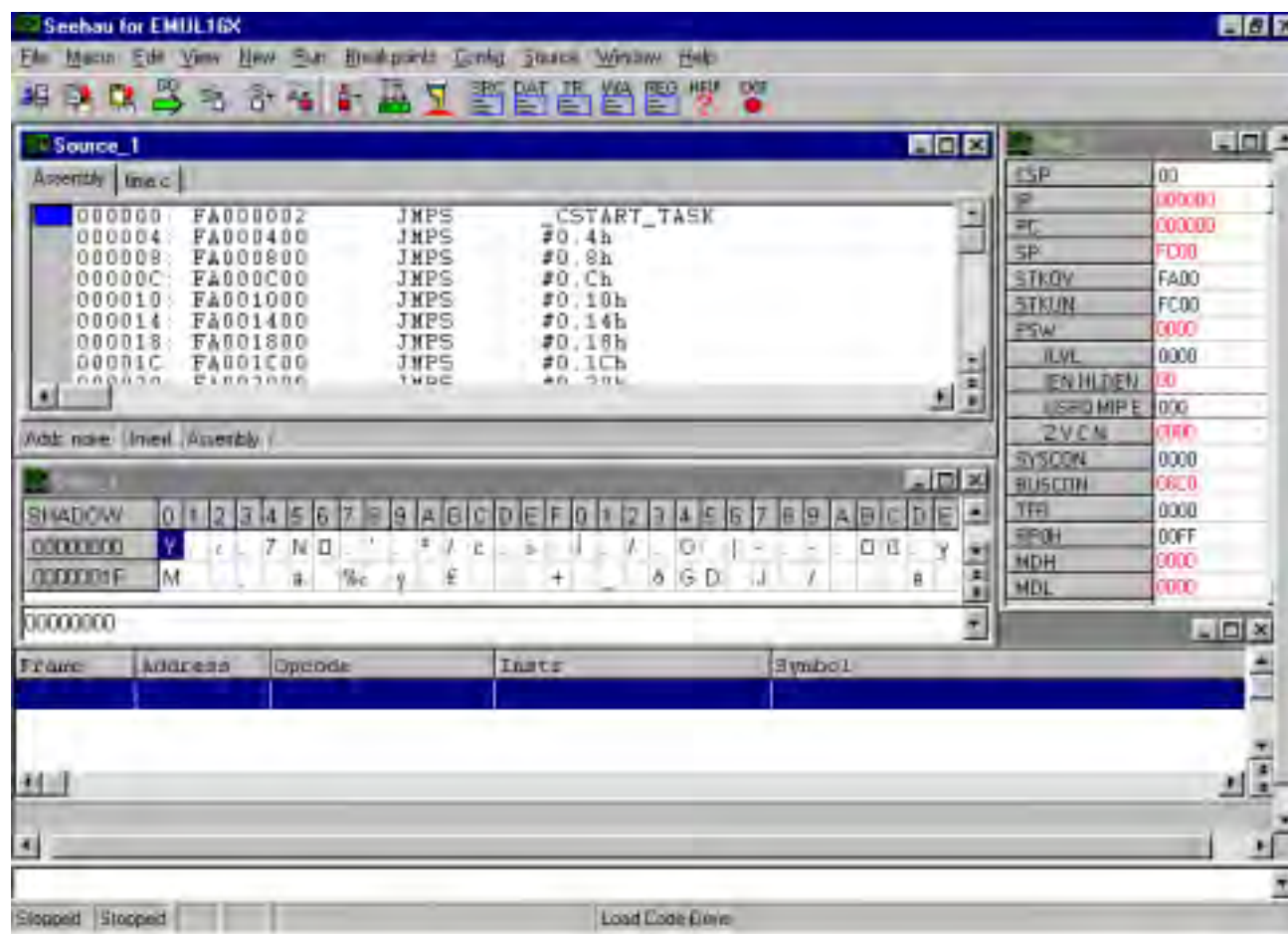
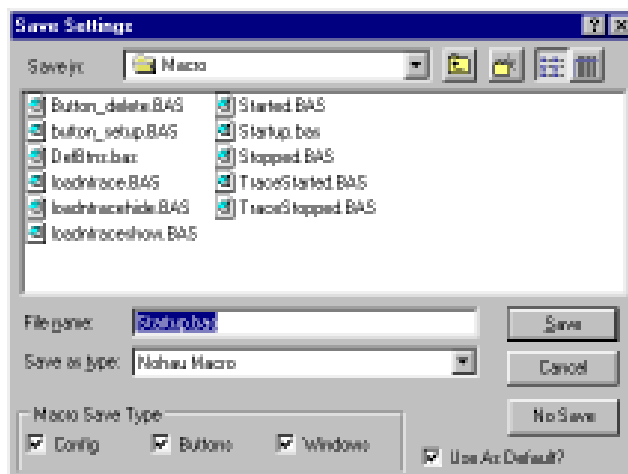


Figure 4

Shutting down SeeHau:

- 1) Click on the X in the upper right corner or select the menu item File, Exit. Figure 5 will appear.
- 2) You can save your setup in startup.bas or a macro file of your own naming.
- 3) If the box Use as Default? is enabled, this file will be used when SeeHau is started.
- 4) This macro will save those items enabled in the Macro Save Type area.
- 5) Select No Save and exit from SeeHau. This will enable a fresh start for the examples.

Figure 5



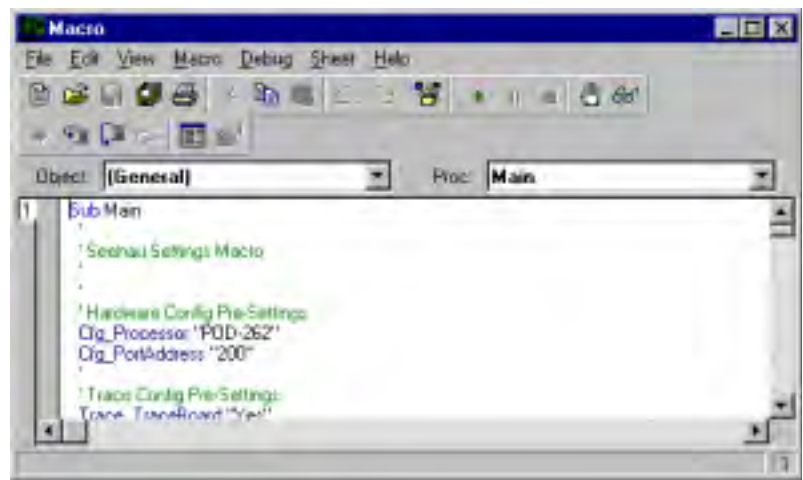
Seehau Commands

Seehau Commands are used to communicate and control the emulator hardware and the Seehau Software Debugger. These commands are used in the Seehau Macro facilities such as user macros, emulator startup configuration and for Seehau configuration items in general. Commands allow the Seehau debugger software to be a very flexible and powerful tool. Modifications and preference settings are easily made by the user. A Macro as defined here is a collection of commands in a text file that are executed by the Seehau debugger. Sax Basic is included in Seehau and permits powerful IF..Then statements to be used in your macros. If Seehau is used as an OLE Automation Server, the commands are executed from the automation controller.

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 6. The macro recorder and editor are found under the *Macro* menu. The resulting filename.bas files are simple text files and can be modified at any time with a text editor regardless of how the macro was originally constructed. These files are stored in the *nohau/seehau/macro* directory on your hard drive.

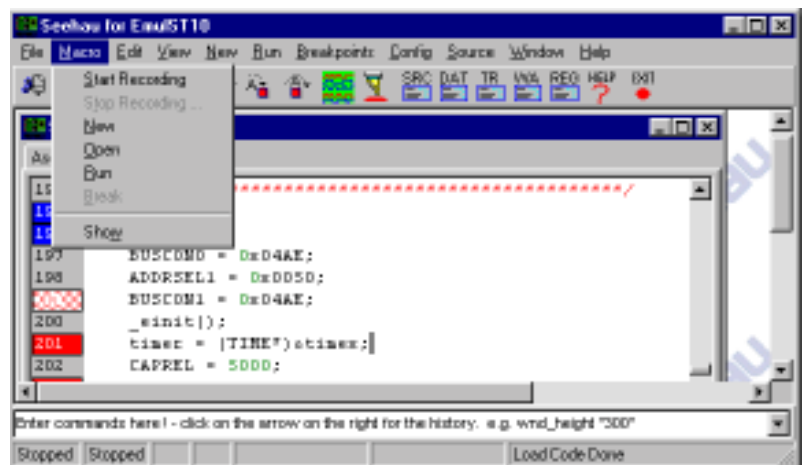
Figure 6



Macro Execution

Macros can be executed by Seehau during its startup phase or manually by the user using the *Macro/Run* command. Individual commands can be executed by direct entry in the main Seehau window. Figure 7 illustrates the dialog box at the bottom of the main window for direct entry. An example is shown. This box can be opened up to display a list of the recently used commands. This eliminates the need to retype most used commands. The box at the bottom of Figure 7 where the word “Done” is where Seehau puts the results of command operations. Not all commands display a result. Note the Macro menu item expanded at the top of the window.

Figure 7



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 online help.

Command Examples

An example of directly entering a command:

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

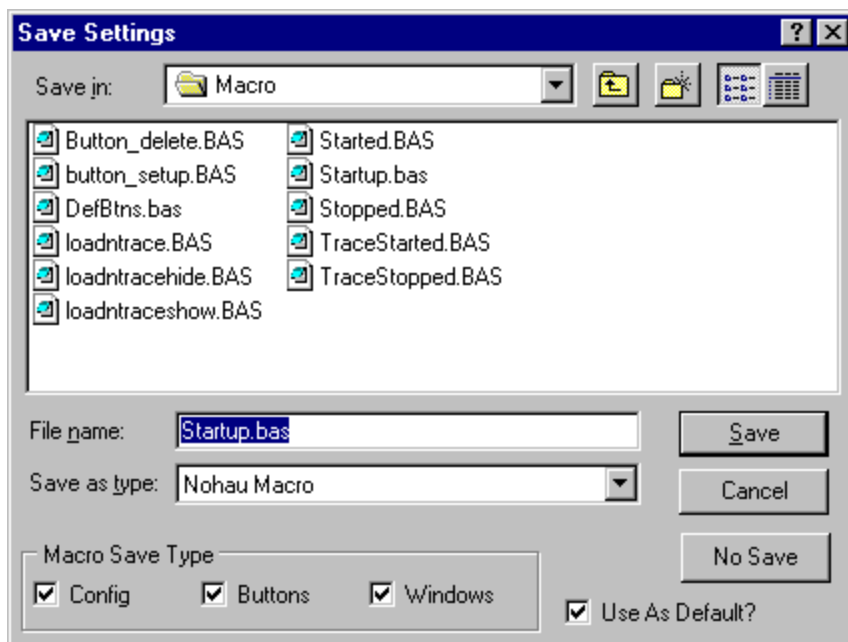
The open window or the one most recently selected by a macro command will have its height adjusted to 300 pixels. To modify the main window, you may have to type in this command first:

`Wnd_select "Seehau for Emul16x"`

This example uses the Emul 166 Emulator for the Siemens C166 family. Enter the appropriate fields for your system as indicated as the caption on the main window.

If you enter these commands in the file `/Macro/startup.bas`, they will be executed everytime you start up Seehau. You can also make your own Macro that saves Seehau configuration setup. You are given this opportunity when you leave Seehau. Figure 8 shows the options you can select:

Figure 8



Config:

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

Buttons:

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

Windows:

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

Use as Default ?:

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

Command Groups:

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

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

Quick Saving a Configuration Menu using the Apply Button

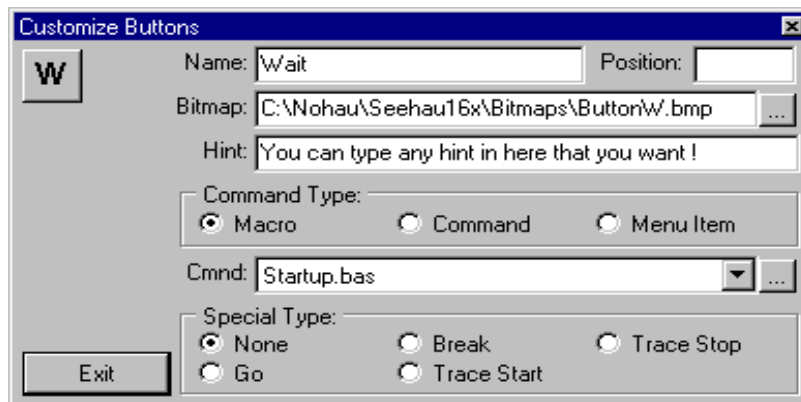
- 1) The configuration menu to be saved must be open and its Apply button must be visible.
- 2) In the Macro menu item, select Start Recording. The configuration window will disappear.
- 3) Re-open the configuration window from the appropriate menu item. Click on Apply. The settings associated with this 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 8. Enter a filename of your choosing and click on Save. The macro is ready to use and will accurately re-create your configuration settings.

Configuration menus are also saved when general Seehau settings are saved upon exit or through the Config, Save Settings menu.

Creating New Buttons

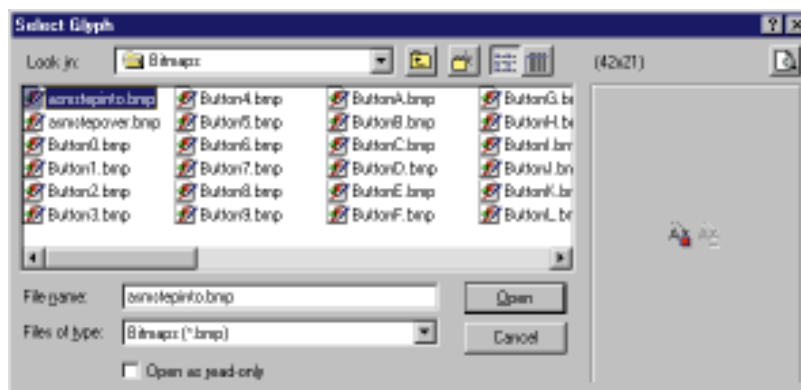
- 1) Buttons can be created and deleted on the icon bar in the main menu. A created button can be attached to a macro you have created. The icons themselves are Windows bitmaps (*.bmp) and are easily created with virtually any graphics program.
- 2) Open the menu Config, Buttons and an empty Figure 9 will open.

Figure 9



- 3) Open the bitmap browser by clicking on the ... button at the right side of the dialog box.
- 4) Figure 10 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 10.
- 5) Highlight ButtonW.bmp. Click on Open and its image will appear as in Figure 10.

Figure 10



- 6) In the Hint: dialog box shown in Figure 9, 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 Cmd: box will change according to the Command Type setting.
- 8) In the Cmd: dialog box, select startup.bas. You can choose the down arrow to open the Macro directory or use the browser by clicking on the ... button.
- 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 9.
- 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.

Chapter 3 The Nohau EMUL166: Timer.abs Example

Running the example program TIMER.ABS

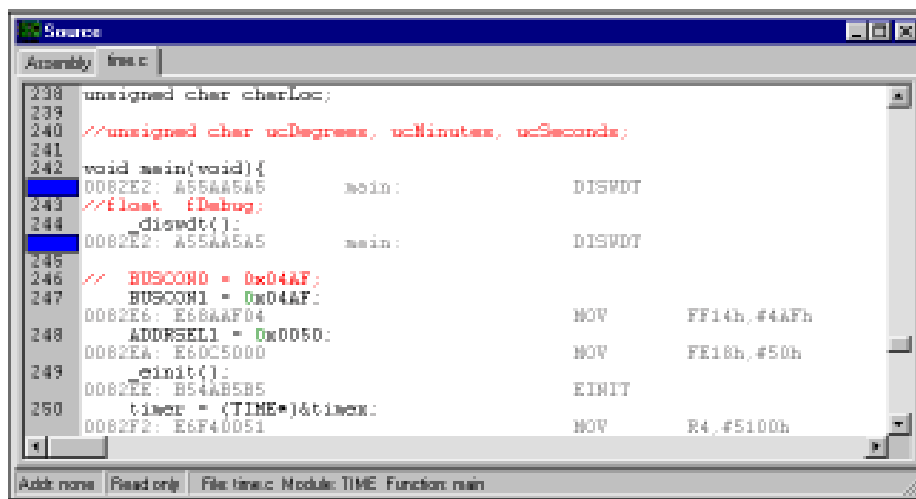
- 1) Nohau provides a small example program called time.abs. It is found in the c:\nohau\seehaul6x\examples default directory. The source code, time.c is also present in this directory.
- 2) Start the emulator as per the instructions in the preceding chapter *Starting the Emulator and Seehau*.
- 3) Resize the windows as in Figure 4 in Chapter 2 but do not add the Trace window.
- 4) Open the menu item File, Load Code or press CTRL L.
- 5) A window similar to Figure 1 opens up. Click on the arrow at the end of the “Files of type” box and select ABS Files. Your window will now look like Figure 1.

Figure 1



- 6) Highlight Time.abs and click on Open. You can also double-click on Time.abs. Time.abs will load into the emulator. The source window will show a JMPS at the reset vector (00000). The jump is to _CSTART. This is at memory location 200 and you may scroll there to see it.
- 7) Click on the Step Into icon and the program will run to the start of MAIN.
- 8) Note that the time.c tab appears on the Source window. You can easily switch between assembly and source language by clicking on these tabs.
- 9) Right click on the source window with time.c tab selected and select Mixed Mode. You will see assembly code mixed in with the appropriate source lines as in Figure 2. Note the program counter (IP) indicated by the blue blocks at the start of MAIN. Click on a line number to set or unset a breakpoint.
- 10) Remove the Mixed Mode from the Source window so only the C source code remains.
- 11) Click on the Source Step Into icon repeatedly and the program counter will advance through the CPU initialization code. Note that where there is assembly code only, the steps are at that level.
- 12) Click on Config, Save Settings and click on Save to save your setup. You may rename your startup file.

Figure 2



Demonstrations of Seehau Features

Watching 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. This example assumes you have completed all the steps so far in this manual and that Time.abs is still loaded in your emulator.

- 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 5000 and press Enter.

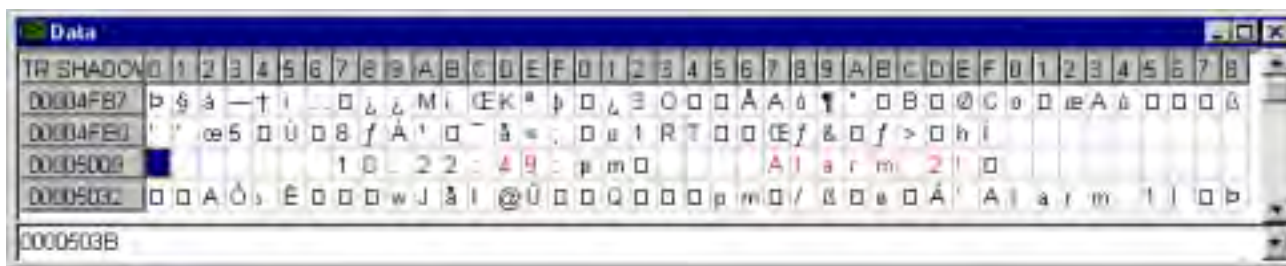
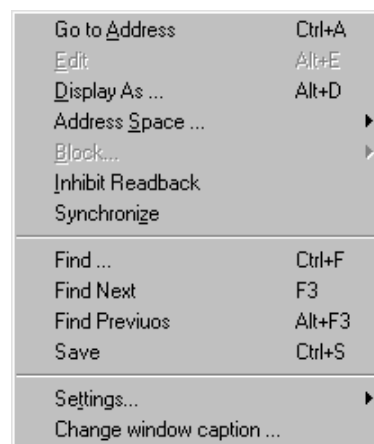


Figure 3

Figure 4



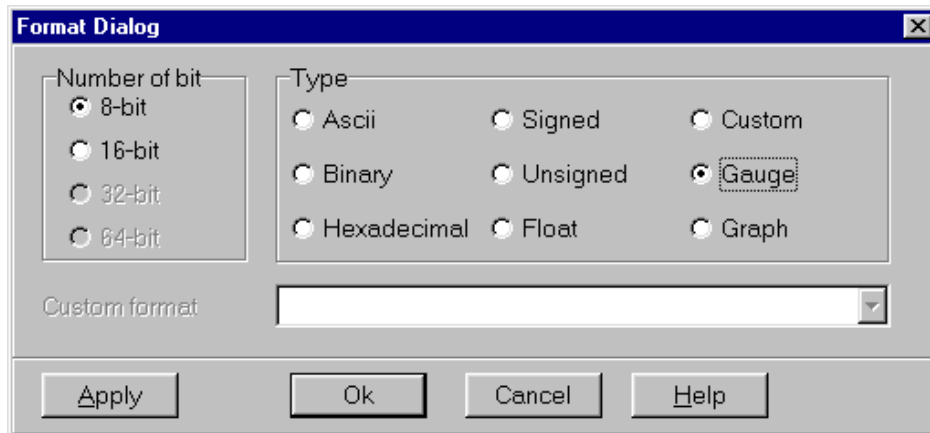
- 3 Right mouse click on the Data window and Figure 4 opens. The idea is to change the Data window to display the data in ASCII format and get data from the Trace Shadow RAM in real-time.
- 4 Select Display As ... and select ASCII. Note the viewing options available in this window. Click OK.
- 5 Right mouse click again and select Address Space ... and select TR SHADOW.
- 6 The address at the bottom represents where the mouse is pointing to. The box highlighted in blue at location 5009 in Figure 3 is the last location where you clicked the mouse on. Data in red indicates that which has been modified by the last emulation run. You will not see ASCII data shown if time.c has not been appropriately initialized at these locations by Time.c running for at least a few seconds..
- 7 Click on the GO icon or press F9. The program time.c will run.
- 8 The time will be updated in real-time. No CPU cycles are stolen to accomplish this. The clock seconds will increment at one second intervals.
- 9 The Shadow RAM will be frozen in its current state when the trace memory is halted. This gives a useful indication of the state of the memory contents when the trace was stopped. The trace is usually stopped because of a trigger becoming true but can also be stopped and started manually in real-time.

Graphical Display of Data in Real-Time Using a Gauge

Seehau has the capability of displaying data in many different formats. The Shadow RAM allows the real-time viewing of data from emulation or target memory and many internal registers. Nearly any register and memory location that is written to will have the write data recorded in the Shadow RAM. Shadow RAM is superior to Dual Port RAM in that data from outside the emulator (i.e. on the target) can be seen in real-time. This example assumes you have Timer.c loaded as per the instructions in this chapter.

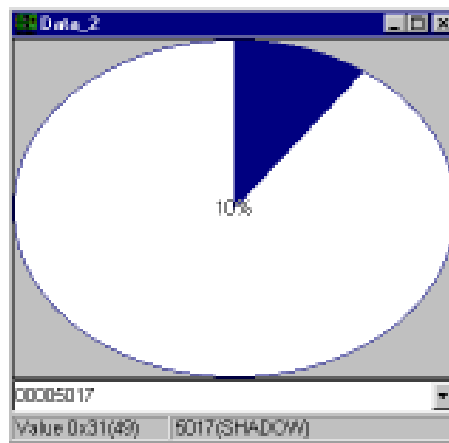
- 1) Open a new Data window. Note that you can have multiple windows of each type open at the same time.
- 2) Right click on the Data window and select Display As..., Figure 5 is displayed, then select Gauge. Note the different types of data display offered. Select OK to close this window.

Figure 5



- 3) Right click on the Data window and select Address Space and click on Shadow to access the Shadow RAM.
- 4) In the dialog box at the bottom of the Data window enter 5017.
- 5) Resize the data window so the display is proportioned correctly as in Figure 6.

Figure 6



- 6) Click on the GO icon to start Time.c
- 7) Note the blue area in Figure 6 changes each second. We can adjust the display to make it more readable.
- 8) Right click on this data window and select Settings, Gauge. In the Gauge Options window, enter 48 in MIN and 58 in Max. The data can have values between hex 30 and 39 which is the ASCII representation of the numbers 0 through 9. This corresponds to decimal 48 to 57. We use 58 rather than 57 as expected so 48 and 57 do not occupy the same space. Click on OK. Note we did this without stopping the emulation. The display will look better now.
- 9) Note that we can change the refresh rate under Settings, Timer and other Gauge formats such as Needle in the Gauge Options window. Colors can also be changed here. Experiment as you wish.

Graphical Display of Data in Real-Time Using a Graph

- 1) In addition to the gauge display mode, Seehau can also display your data as a graph. The data collected can be saved to a file. Right click on the graph to see the local menu. Click anywhere to close it.
- 2) With a data window selected, right click and select Display As ... In the Format Dialog box, select Graph and click on Ok to close this window. You can use the gauge window from the last example or open a new data window. A blank Figure 7 will appear. Size it as appropriate.
- 3) Click on the GO icon to start timer program unless it is already running.
- 4) The graph will start to collect data and scroll to the side.
- 5) Right click and select Settings, Graph and enter 40 for the Y Axis Min and 65 for the Y Axis Max. Click OK. This will improve the way the data is displayed in this window.
- 6) Right click and select Settings, Color. Change the foreground to red and the background to grey. Click OK to close this window.
- 7) The display will be similar to Figure 7.
- 8) Shut down Seehau without saving.

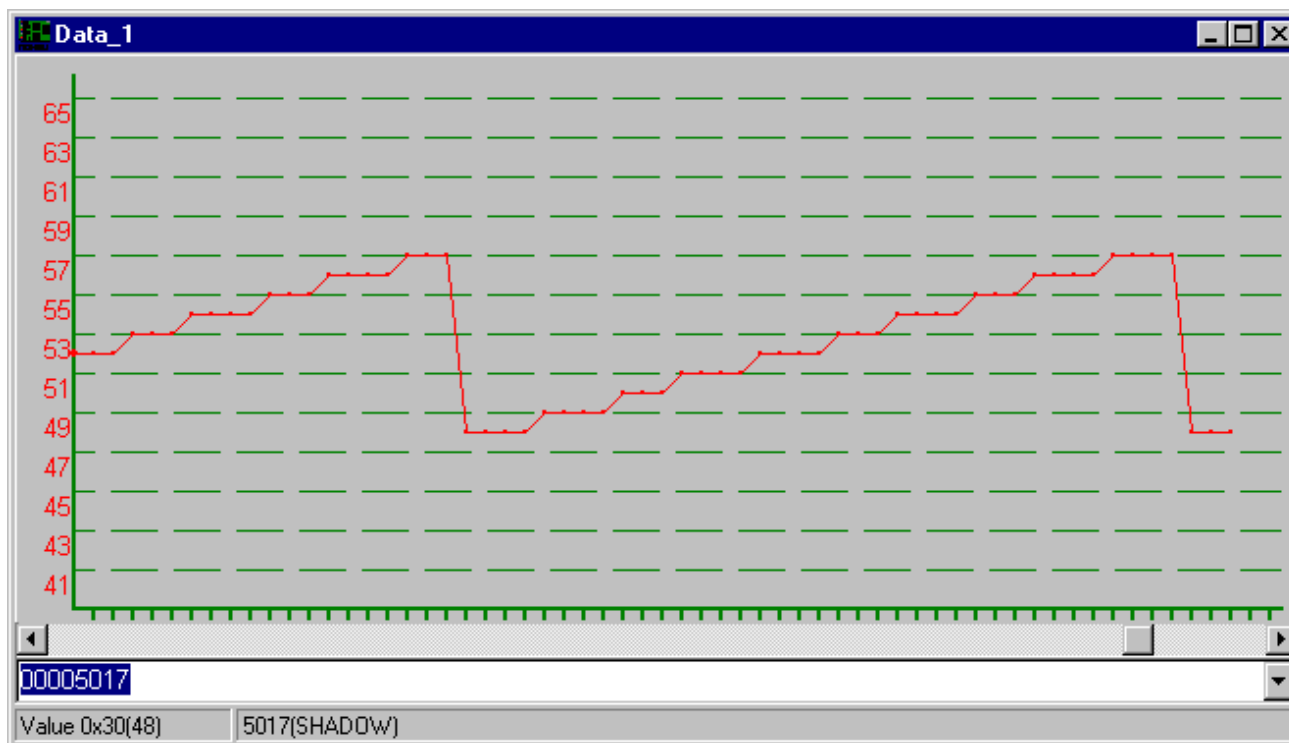


Figure 7

Setting Breakpoints

The EMUL166 has both software and hardware breakpoints. These breakpoints are no-skid in that they do not execute the instruction they are placed on.

To set a software breakpoint, click on a line number. This area will turn red.

To set a hardware breakpoint, click on a line number with the Alt key depressed. A crosshatch red box will appear. Breakpoint information can be viewed in the View, Breakpoints menu item. Click again to remove the breakpoints.

Program Performance Analysis (PPA)

Seehau has a Performance Analyzer which provides statistical information on your program execution.

Information on both code and data accesses are provided and this information can be saved to a file. Figure 8 is an example of a PPA window with sample data from the timer program. Data can be displayed as a bar graphs (as shown) or as numbers. The PPA window is configurable in the local menu which is accessible with a right mouse click on the PPA window.

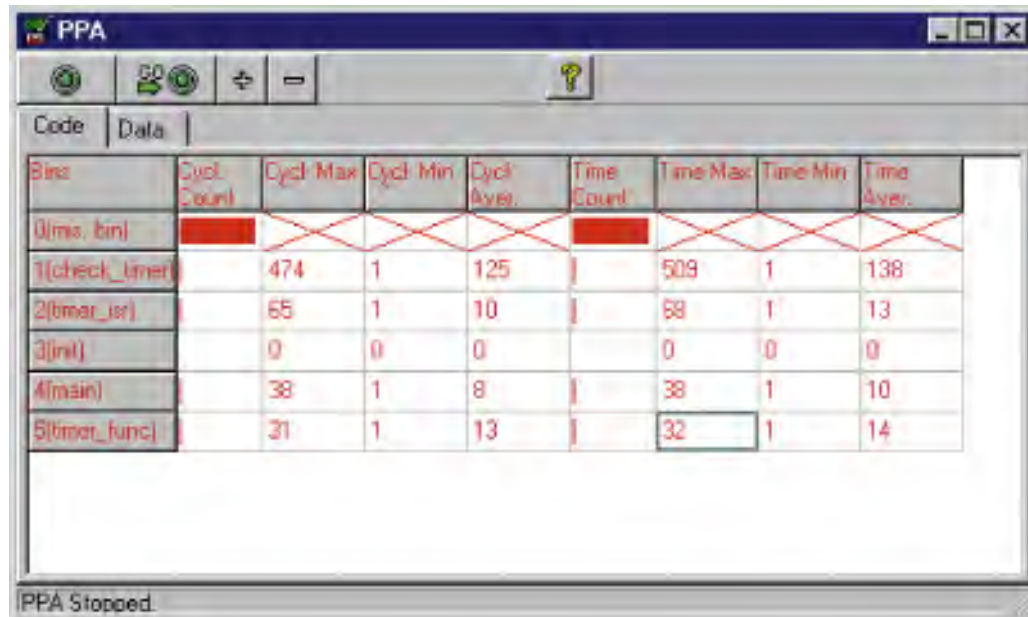
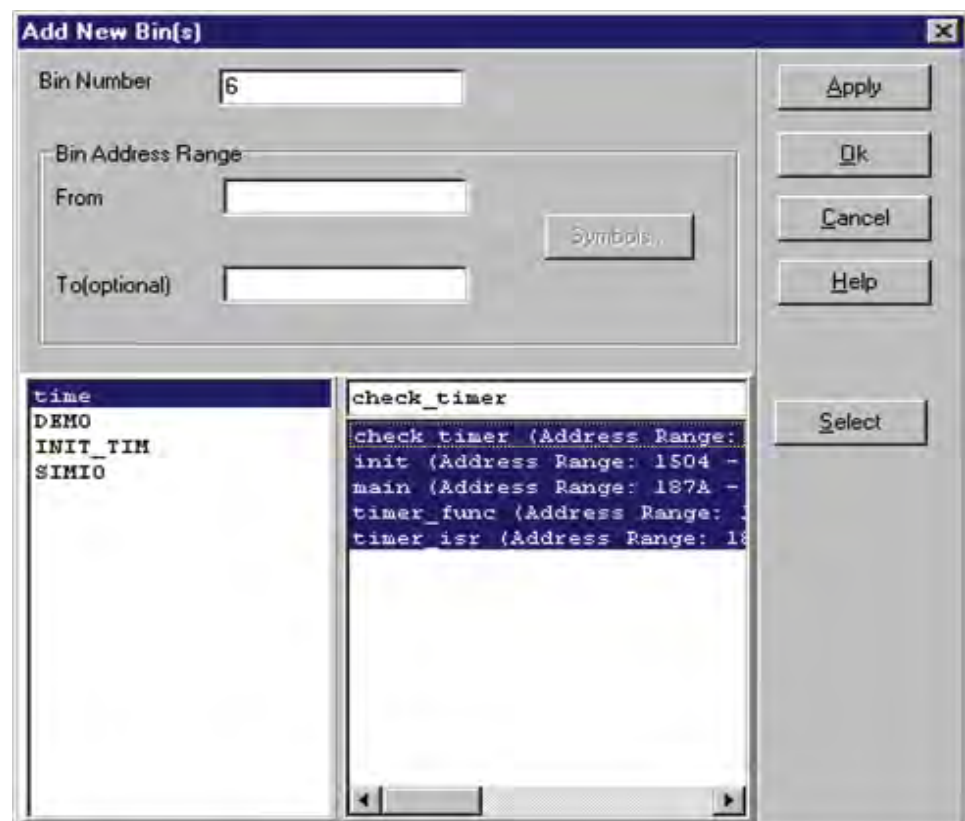


Figure 8

- 1) Load the timer.c program in the usual fashion. Click on the step into source to come to the start of main. This ensures the symbols will be available to the PPA.
- 2) Select the PPA screen in the Tools, PPA in the main SeeHau window. An empty Figure 8 will appear.
- 3) Right mouse click on this window to open the local menu and select Add Bin. You can also click on the + button in the PPA window.
- 4) An empty Figure 9 will appear. This is where you select modules, functions or address ranges for the bins where the information will be stored.
- 5) Click on the Symbols box to activate the browse feature.
- 6) Double click on `time` to bring up the module names from the symbol browser.
- 7) Hold the Ctrl button and click on the indicated names to select them. You can use the shift key also.
- 8) Click on Apply and OK. Each of these functions will now be listed as in Figure 8.
- 9) Click GO on the main icon bar to start the emulation.
- 10) Click on the Start icon in the PPA window to start the collection of data. You could also click on the GO button in the PPA window to start the emulation and the PPA at the same time. The GO button turns red and this now becomes the STOP button. It will stop both the emulation and the PPA process.
- 11) The data will appear in a few seconds and will update periodically. Click on the Code and Data tabs to view the different types of PPA data.
- 12) Right click on the PPA window and select Show Bar Graphs.
- 13) Right click and select Colors and right click on the red square to replicate the same color scheme as Figure 8.
- 14) Click on Stop in the PPA window to stop the collection of data. See 10) for the location of the STOP button.

The emulation must be running in order to start the PPA. If you stop emulation while the PPA is running, it will take some time for this action to be recognized.

Figure 9



Chapter 4 The Nohau EMUL166: 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 25 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 target 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.


Trace Window Display

The fields displayed in the trace window can be selected depending on your needs. Figure 1 shows the trace local window. The meaning of these fields is explained in the on-line Help files. The XDATA, ID_IA and OD_OA busses are internal CPU busses brought out by the bondout controller. These fields are available to the user. The emulator uses these busses to access the internal areas of the controller.

Figure 1



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 also stops providing a snapshot of the system memory.

With the timer program loaded and the trace window open, run timer for a few seconds and then stop the trace with the trace icon. This is the icon with TR indicated on it.  It will turn green.

The trace window will be filled with frames. Try activating certain features in Figure 1 to see the effects.

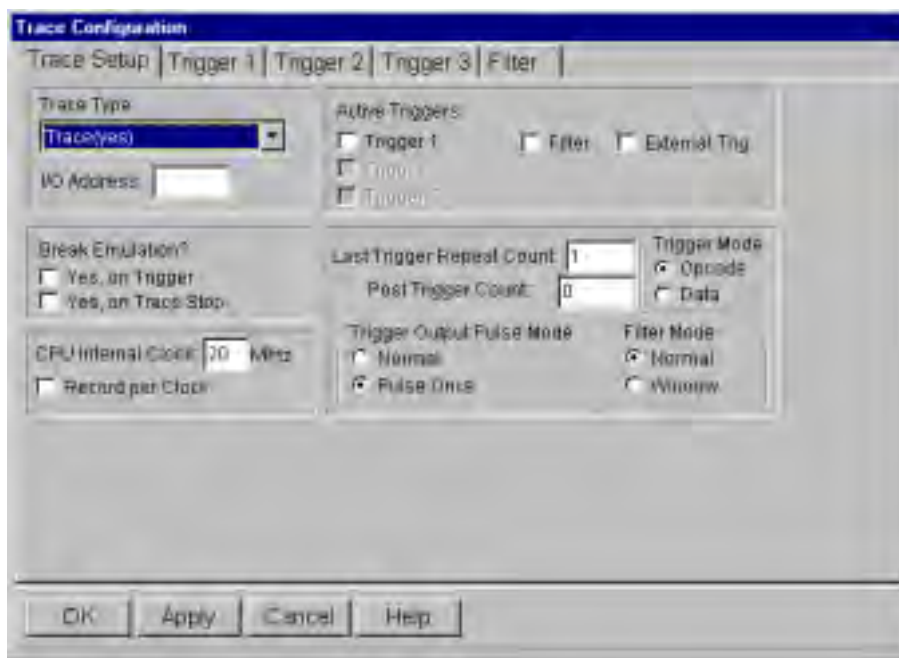
The trace memory is where the frames are stored. The triggers have the power to stop the trace recording, and/or emulation and to control what is recorded in the trace memory (filtering).

Trigger Example on an Address and Data Qualifier.

Triggers are probably 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 5017 with the ASCII value of 34.

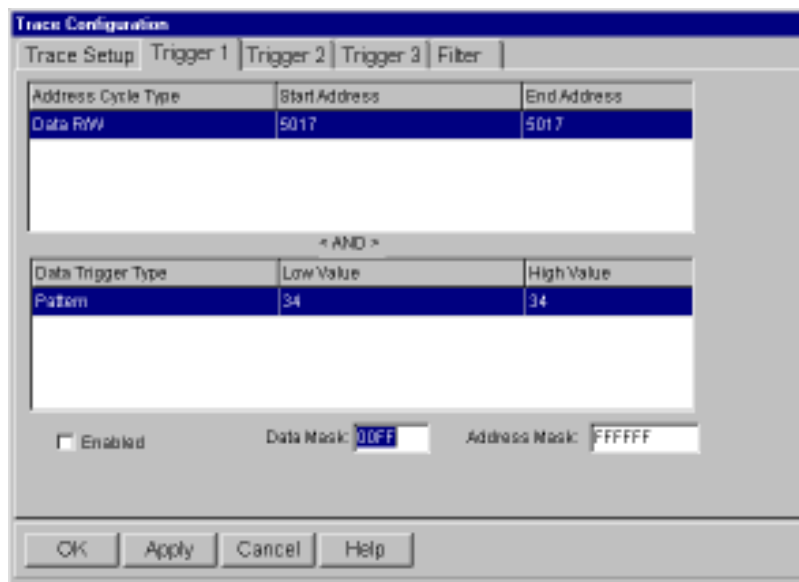
- 1) Setup the emulator to run the timer example as described in Chapter 3. Make sure the data window is set to view ASCII data as in Figure 3 in Chapter 3. This is not needed for the trigger to work but for your inspection of the 5017 memory location. Open the trace window and position the windows so 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. The timestamp timings are derived from the value in CPU Internal Clock box. Note the tabs at the top showing the three Triggers and the Filter as well as the Trace Setup.

Figure 2



- 3) Activate the Yes, on Trigger box. When a trigger occurs, the emulation will be stopped.
- 4) Click on the Trigger 1 tab. A blank Figure 3 opens up. The values of the qualifiers are inserted here.

Figure 3



- 5) Right click on the Address Cycle Type window and select Add. The Edit Trigger Qualifier dialog box in Figure 4 opens up. Fill in the fields as shown. Make sure you select Data R/W. Press OK.
- 6) Repeat for the Data Trigger Type and enter the value of 34 in each field as in Figure 5. Click OK.
- 7) In the Trace Configuration menu as shown in Figure 3, enter 00FF in the Data Mask field. Note you can use this field to enter a “don’t care” condition on a bit by bit basis. i.e. 000F will only use the lower nibble of the data qualifier. This can also be done in the address mask field to mask off address ranges.

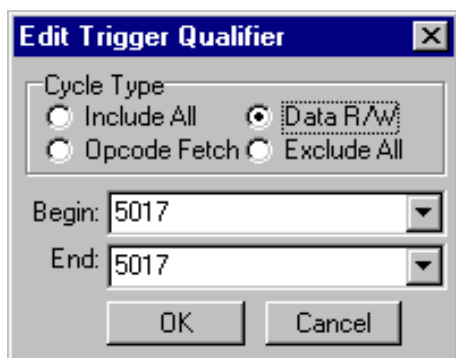


Figure 4

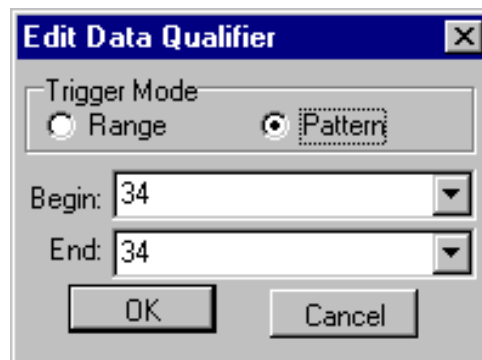


Figure 5

- 8) Click on OK to enter the data.
- 9) If the emulation and trace were running during this time, temporarily stop and start the trace to register the trigger data. Note that emulation will not be slowed or stopped during trigger setting or trace stopping.
- 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 6.

Frame	Address	XData	Opcode	Inst	Symbol
-7	470:	F207	07F23000	ADDB	RL1, #30h
-6	472:	0030	0030 -- oo2		
-5	474:	2CB9	B92C	MOVB	[R12], RL1
-4	476:	C1DB	08C1	ADD	R12, #1h
-3	478:	F3E7	E7F23A00	MOVB	RL1, #3Ah
-2	47A:	003A	003A -- oo2		
-1	5017:	34	34 -- w1		
0	47C:	2CB9	B92C	MOVB	[R12], RL1
1	47E:	00CB	CB00	RET	
2	480:	90B8	B890	MOV	[-R0], R9 ==> check_time:

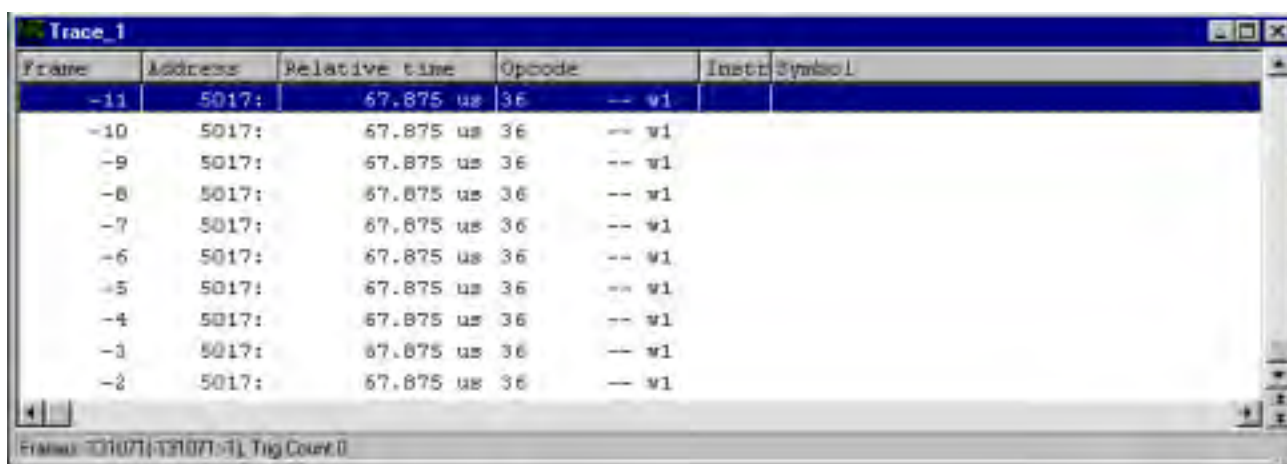
Figure 6

- 12) The arrow at B is the write of 34 to address 5017. w1 means a 1 byte write bus cycle. The instruction that did write this is at arrow A.
- 13) The instructions at Frame -4 and -3 are prefetches to the CPU pipeline. The fact that the instruction fetch at A is followed later by the write at B clearly shows the pipeline effect. Frame -2 is the subsequent word fetch corresponding to the first fetch shown in Frame -3. oo2 signifies a 2 byte fetch or a 2 byte pipeline flush. Frame -2 is repeated inside Frame -3 where it was disassembled.. Frame -2 is for reference to the bus cycle that actually occurred. It is placed in Frame -2 for readability.

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 5017 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 Figures 3, 4 and 5. 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 7 will be displayed. Note that only byte writes of 34 to location 5017 are recorded.



Frame	Address	Relative time	Opcode	Instr	Symbol
-11	5017:	67.875 us	3E	-- w1	
-10	5017:	67.875 us	3E	-- w1	
-9	5017:	67.875 us	3E	-- w1	
-8	5017:	67.875 us	3E	-- w1	
-7	5017:	67.875 us	3E	-- w1	
-6	5017:	67.875 us	3E	-- w1	
-5	5017:	67.875 us	3E	-- w1	
-4	5017:	67.875 us	3E	-- w1	
-3	5017:	67.875 us	3E	-- w1	
-2	5017:	67.875 us	3E	-- w1	

Figure 7

Note the timestamp. It shows that each write was 67 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.

Chapter 5 The Nohau EMUL166: Connecting to your Target

Clocks, Oscillators and Crystals

The maximum frequency of the EMUL166 is 33 MHz. This is the ECLK signal located on the emulator board. The most used frequency is 20 MHz. This is usually supplied by a 5 MHz crystal or oscillator which is then multiplied by 4 to obtain the required 20 MHz. See the Siemens User manual for more information on clocks.

Crystal Capacitors

If your application uses a crystal please note that Siemens specifies the capacitors are two different values. While most crystal oscillators use capacitors of the same value, stability is increased with different values. Siemens have an application note on this subject named AP242003 and it can be found on the Infineon web at <http://www.infineon.com/products/ics/34/index3.htm>. Note that many Phytex evaluation boards use same value capacitors and you may need to add an additional capacitor to ensure adherence to the Siemens specification. Suggested values at 5 MHz are 22 pf (input) and 47 pf (output) with a resistor value of 390 ohms.

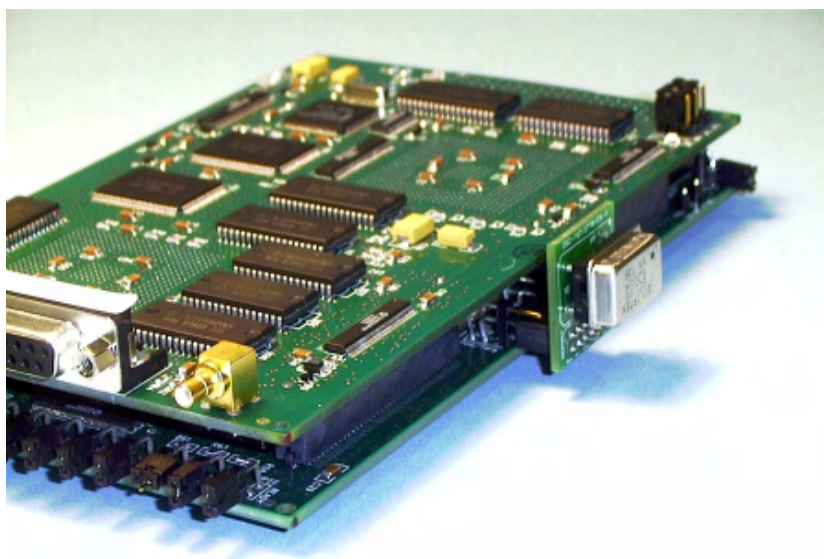
EMUL166 Oscillators

Earlier models of Nohau emulators had a plug in oscillator package of either 20 or 5 MHz. Later models have this oscillator soldered in for reliability. Nohau offers a simple adapter board so you can easily change oscillator values. This is shown in Figure 1 and both oscillator and crystal versions are offered.

This adapter connects to JP51, JP7 and JP10. The adapter will plug into the lower set of jumpers i.e. target.

Some versions may have a PLL IC and the ability to send the clock to the target. Contact Nohau Technical Support for the latest information. support@nohau.com

Figure 1



Connecting to Your Target Hardware

Connecting an emulator to a target system can require some careful work. The Nohau EMUL166 mimics the C166 family as closely as possible. There are a few issues that can cause particular problems. Here are some useful hints:

- 1) Try the emulator in stand-alone mode first. Get it to work this way first. You do not have the complications of your target buses to make things harder to resolve.
- 2) If you connect the emulator to the target without making any changes to your jumper or software settings: the emulator should still work. If it does not, check for shorted or incorrect signal lines on the target. Get someone else to check your board layout. Many problems are finally traced to crossed lines.
- 3) Once you have the emulator working from its internal memory - you can switch to the target memory.
- 4) Remove the RESET and READY jumpers. If your reset stays on too long, the emulator will never run.
- 5) Measure the CPU frequency at the ECLK pad on the emulator. See Figure 3 in Chapter 1. Make sure it is what you expect it to be. If your program crashes, make sure the frequency is not changing unexpectedly.
- 6) Most users prefer to bring the target's clock up to the bondout controller. Make sure your clock pins are correct. JP10 and JP7 should be in the lower position.
- 7) If you are supplying power to the target and the emulator separately, make sure JP6 T-PWR is not connected. Make sure you use the proper power sequence to protect the bondout. Never allow the target to have power without the emulator being powered up.
- 8) Consider purchasing an evaluation board to serve as a reference design. Phytex boards are good choices but you should consider that some do not use the Siemens chip selects but rather a GAL device for addressing. The C161 and C164 are like this. The early C161 does not use the Siemens bootstrap mode. Adapters are very inexpensive for these boards. The C167 board uses CS0 for the FLASH and CS1 for the RAM and this setup is very easy to use.
- 9) To test target RAM, use the data window to change a memory location to different values such as FF or 00. The emulator must return the same value that you entered. You can use any erroneous bit patterns to help determine where the error is.
- 10) Design small test programs that you can send to Nohau technical support. Please include the source and any compiler project files if these are used. The ability to replicate the problem is important and helps a great deal.
- 11) The reset location (00 0000) will normally contain a valid jump instruction after the user code has been loaded. If it does not, make sure you are in the correct address space such as external or single-chip mode and that you are not accessing some ROM.
- 12) If you have some RAM on your target and you have this mapped to your emulator: you should be able to write values to this RAM from within a Data window and get the same value returned. You should be able to do this on adjacent bytes and also on a whole word. If you do not get the same value returned, you are either in some sort of ROM, nonexistent or defective RAM or a setting could be incorrect.

If you can write correctly to one byte yet not to the next, this nearly always means adapter problems or crossed data lines on your target..

Conclusions

This explains a portion of the power of Seehau and Nohau C166 family emulators. We have not explored the RTOS support, code coverage, compiling and editing abilities of Seehau.

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

For applicable Application Notes see www.nohau.com under Technical Publications.

Chapter 6 Debugging with the Infineon E2 Bondout Microcontroller

Introduction

There are several possible methods to emulate a microcontroller and each method has its benefits and drawbacks. Infineon chose the bondout controller method for its C166 family and Enhanced Hooks for its C500. This article will focus on the E2 bondout that emulates all C166 family members except the C166 itself. The C166 has its own bondout.

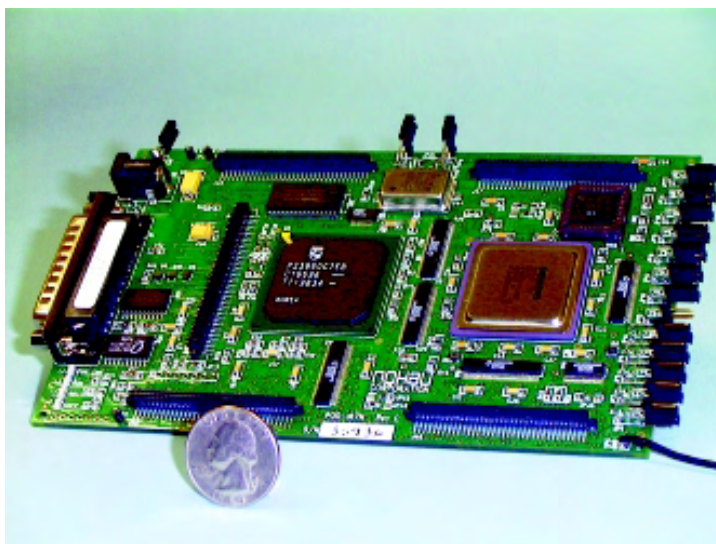
Nearly all emulators made for the C166 family utilize the E2 bondout although there are a few that use standard production chips. The E2 replaces the older E (or E1). The Nohau EMUL166 is shown below without its case. The E2 bondout is on the right side and has a silver top.

Internal or External Mode

The C166 can operate in either single-chip or external mode. In External mode, address and data buses are used to access external memory via ports P0, P1 and P4.

In single-chip, the instructions are fetched and data is accessed from on-board ROM or FLASH and RAM. There are no data and address buses to the outside world that an emulator can use. P0, P1 and P4 can then be used as general purpose I/O.

External mode allows an emulator to see operations since the buses are visible. The emulation must be interrupted in order to read internal registers and RAM. Real-time operation is therefore lost. A Bondout or Enhanced Hooks fixes these problems.



What the E2 Bondout Offers

The E2 bondout operates in both external and single-chip modes. It has extra external buses to provide the emulator access to internal resources and information even when the address and data buses are not available as in single-chip mode. These buses are called IA, ID, OA, OD and XBUS. Most emulators use these buses to obtain detailed internal information and they are also displayed in the Nohau trace memory. Nohau displays the signals it derives from these buses.

You can trigger on and record in trace memory internal events in real-time. These internal events are impossible to see without a special emulation chip such as a bondout or Enhanced Hooks.

For more information see www.nohau.com under Technical Publications.

The E2 offers internal triggering, ROM simulation, and detailed trace recording all without intrusion into the emulation.

Note: Nohau will be making some enhancements to the trace capture and display. When this is completed, the screen shots described here will no longer match. This chapter will be rewritten at that time to incorporate the latest changes and will be posted on the web. The software updates will also be available free of charge on the Nohau website.

Special Bondout Buses

Five buses available on the bondout are optionally visible in the trace display as shown in Figure 1. The descriptions of the buses are general in nature and are not true for all cycles. Not all buses are shown.



Frame	Address	Relative time	Misc.	XData	Opcode	Instr	Symbol
-56	4C8:	100 ns	0 00 7853	0217	0217	-- oo2	
-55	4CA:	100 ns	0 00 7863	D800	00D8	ADD	D13,D8
-54	4CC:	100 ns	0 00 7875	00CA	CA00AC02	CALLA	cc_strcmp
-53	4CE:	100 ns	0 00 7806	02AC	02AC	-- oo2	
-52	4D0:	100 ns	0 00 7816	4048	4048	-- oo2	
-51	4D2:	100 ns	0 00 7820	1B3D	1B3D	-- oo2	
-50	2AC:	100 ns	1 00 7910	0600	0D06	JMPB	cc_UC,6h ==> strcmp:
-49	2AC:	100 ns	1 00 7820	2CA9	2CA9	-- oo2	

Frames: 1343-1343-1; Trig Count: 0

Figure 1

IA Bus: Instruction Address

ROM Instruction Address Bus. This 16 bit bus shows the address of instruction transfers on the internal ROM bus. This is visible only in single-chip mode and is not shown in the Figures.

ID Bus: Instruction Data

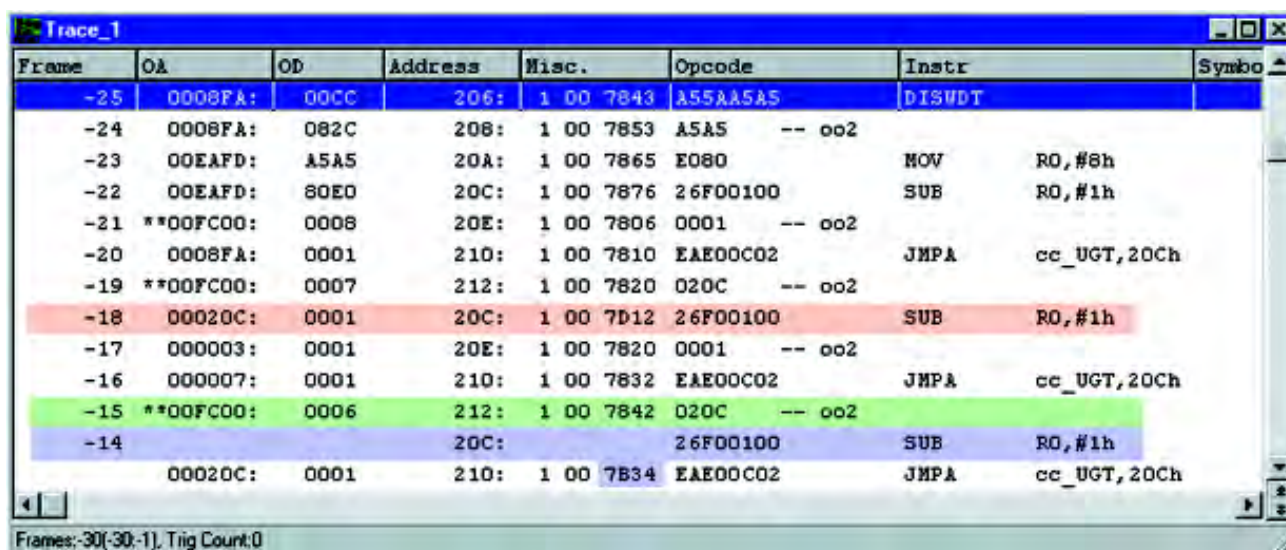
ROM Instruction Data bus. This 32 bit bus displays instruction and data over the ROM bus. Effective only for single-chip and is not shown in the figures..

OA: Operand Address

This 18 bit bus displays read and write addresses. This and the OD bus are shown in the second screen shot. The two * characters shown in the OA field signify valid write cycles that were executed. This field can be used to distinguish between executed opcodes and pipeline flushes. This is shown in Figure 2.

OD Bus: Opcode Data

This 16 bit bus displays the data word of an opcode. Opcodes of injected instructions (into the pipeline) are visible here. The OD and OA buses are very useful. This is shown in Figure 2.



Frame	OA	OD	Address	Misc.	Opcode	Instr	Symbol
-25	0008FA:	00CC	206:	1 00 7843	A55AA5A5	DISWDT	
-24	0008FA:	082C	208:	1 00 7853	A5A5	-- oo2	
-23	00EAFD:	A5A5	20A:	1 00 7865	E080	MOV	RO,#8h
-22	00EAFD:	80E0	20C:	1 00 7876	26F00100	SUB	RO,#1h
-21	**00FC00:	0008	20E:	1 00 7806	0001	-- oo2	
-20	0008FA:	0001	210:	1 00 7810	EAE00C02	JMPA	cc_UGT,20Ch
-19	**00FC00:	0007	212:	1 00 7820	020C	-- oo2	
-18	00020C:	0001	20C:	1 00 7D12	26F00100	SUB	RO,#1h
-17	000003:	0001	20E:	1 00 7820	0001	-- oo2	
-16	000007:	0001	210:	1 00 7832	EAE00C02	JMPA	cc_UGT,20Ch
-15	**00FC00:	0006	212:	1 00 7842	020C	-- oo2	
-14			20C:		26F00100	SUB	RO,#1h
	00020C:	0001	210:	1 00 7B34	EAE00C02	JMPA	cc_UGT,20Ch

Frames: 30(30-1); Trig Count: 0

Figure 2

XBUS

The XBUS is an internal representation of the external bus. This 16 bit bus displays the data on the internal XBUS as well as data writes to external memory. This field is shown as XDATA in Figure 2.

Misc.

This field displays bit flags that Nohau derives from the bondout controller pins. These signals are useful to determine certain situations the controller is in. Examples include jump taken, jump cache hit, R/W, trigger event, operand size, and instruction or data. Eight I/O signals and the pipeline instruction counter are also displayed in this field. This field is shown in Figure 2. Note Frame -25. The Misc. field consists of this string:

The “1”

The first digit indicates an internal condition of the bondout. This field is used for Nohau development and is usually not visible. This bit field has no use for the user.

The “00”

The second byte is displayed in hex and represents the state of the 8 bits of user data on the input connection J3. This is the D shell connector on the trace board. In this example the byte is at 00.

The “78”

The next hex word (7843) is divided into 2 bytes. The 8 most significant bits (78) represents 8 status signals coming from the bondout controller.

MSB

Trigger	A trigger occurred during this frame.	
Instruction	1 = instruction fetch	0 = data fetch
Size	1 = 2 byte fetch	0 = 1 byte fetch
R/W	1 = read cycle	0 = write cycle
I EXT FETCH	The word fetch was internal (0) or external (1) memory space.	
JC Load	The Jump Cache was loaded with this target instruction (1).	
JC Hit	The Jump Cache contains the target instruction which will be fetched from the cache.	
Jump Taken	The jump or call was taken (1).	

LSB

Examples:

“78” means Instruction fetch, 2 byte, Read from external memory space.

“x9” means a jump taken.

“xB” means a cache hit.

“xD” means a cache load.

The “43”

The 4 lowest bits (3) are the prefetch queue counter contents and is assigned for executed instructions. The next higher 4 bits (4) is the prefetch queue counter and represents word fetches into the pipeline. In this example, the word being fetched is assigned the number 4. The word being executed is that which was assigned the number 3 in some previous frame. The word can be an opcode or data.

If the fetch or execute digit does not change as the 6 in Frames -2 and -22 in Figure 2, the counter was not advanced because the described operation did not take place on one of the lines. Fetches and executions do not necessarily occur on every frame all the time.

This is useful for seeing where a given word was fetched and subsequently executed unless it was flushed from the pipeline. You can match the fetch with the execution to follow the exact bus activity.

Instruction Pipeline

The C166 family has a 4 stage instruction pipeline that can be demonstrated in the trace memory. The four stages are fetch, decode, execute and write back. If an instruction is fetched, but not executed because of a change in program flow (usually from a jump or call), these instruction(s) are flushed from the pipeline and new ones loaded.

These flushed instructions are tracked in order that there is no false triggering on them. This tracking is displayed in the misc. field. Unexecuted instructions must not be used as trigger qualifiers.

It is important to be able to detect pipeline effects since there is a definite delay from when an instruction is fetched to when it performs its operation. The E2 provides enough information to the emulator in order to decode this data.

Pipeline Prefetch & Flushes

The screen shot shown in Figure 1 resulted from this code sequence being fetched or executed:

The ADD at 4CA is executed and the CALLA is taken. Program flow continues at *strcmp* at address 2AE.

Frame -54 (green) is the fetch of the CALLA opcode. It is obviously executed at frame -50 with the appearance of JMPR instruction, 400 nsec later. Since each frame records one word, Frame -54 actually contains the first word of the CALLA, namely CA00. The AC02 is put there for readability. Frame -53 contains the second word 02AC. The oo2 means a 2 byte opcode fetch. Remember, the C166 is Little Endian so the data came off the bus in this order and Nohau swaps it for convenience and readability.

Frame -52 and -51 are prefetches of the CMP (4840) at 4D0 and the JMPR (3D1B) at 4D2 ! These are flushed instructions that were prefetched, but not used as the program flow changed as a result of the CALLA instruction. Note all this information is present in the Nohau trace plus a lot more. Interestingly, Frame -49 is also a flushed instruction.

Jump Cache

The C166 family has a jump cache mechanism to speed conditional loops. The branch target instruction is stored in the jump cache and each time it is needed, it is fetched from here and not from slower program memory. This branch target instruction is injected directly into the decode stage of the pipeline. The trace screen shown in Figure 2 was derived from this code segment:

The Branch Target Instruction is the SUB at 20C. The DISWDT is the disable watchdog timer. R0 is initialized with 8 and is decremented by 1 by SUB. JMPA tests R0 and jumps back to SUB until R0 equals zero. In this case, a trigger was set for R0 to equal 5, then halt emulation. This action is not visible in this trace window.

Jump Cache Hit

Siemens states that when the cache jump instruction passes through the decode stage of the pipeline for the first time, the branch target instruction is fetched and stored in the jump cache. This is visible in the trace window as described here:

When the second nibble of the Misc. word contains a D (i.e. 7D12), this means the instruction is placed into

the jump cache. When this nibble equals B, this signals a jump hit. This instruction is then taken from the cache and injected directly into the pipeline.

Frame -20 contains the first jump and Frame -14 contains the cached SUB. The 7D12 and the 7B34 indicate this. Frame -14 actually has the 7B34 on the following line. Note there is no data in either the OA or OD fields. This is because no fetch occurred.

Two for the Price of One:

The pipeline can allow the CPU to perform two operations at the same time and this is also visible in the Nohau trace. Frames -21, -19 and -15 have two events at the same time.

Note the ** in the OA column which indicates a valid write cycle. Where is this write ? It is to memory FC00 which is equal to R0 because the CP register (Context Pointer) points to FC00. This is the base of the register bank. Recall R0 is being decremented by SUB and you can see this in the OD column on Frames -19 (R0=7) and -15 (R0=6). Frame -21 (R0=8) is the initialization by the MOV at address 20A. This is the first event.

During this time there is also a fetch of the data part of an opcode. In each of Frames -21, -19 and -15, under the Opcode column is the second half of the opcode fetch of the frame before it.

The Opcode value for Frame -19 contains the data or second word for the JMPA at Frame -20. It is 020C swapped to 0C02.

Triggering on Internal Data

The E2 bondout allows triggering on internal memory areas such as RAM, SFRs, ROM and XRAM. Programs can be executed in the internal RAM, and these instruction fetches can cause triggers. Triggering can start or stop the trace recording, provide an external output to trigger an oscilloscope or logic analyzer or halt emulation. This is an important feature. The E2 bondout provides a way to peek inside the controller to increase your debugging power.

In the example below, the trace recording was halted when R0 was changed to 5. This is impossible to without a bondout chip or by replicating the CPU in the emulator hardware (which is not easy).

A trigger can have up to 50 addresses ANDed with 50 data values or ranges: and there are three triggers in the Nohau EMUL166 ! Plus a Filter to specify what cycles are recorded in the trace memory.

Two Chip Emulation

The E2 bondout contains two CAN modules (hence can support the C164) and 3 XRAM modules of 2K, 4K and 6K. Peripherals that are not on the E2 such as the C163 SSP are supported with a C163 on a daughterboard in Emulation mode. This turns off the 163 CPU but the XBUS peripherals are still active and available to the E2 through the external bus. New derivatives can be supported in this manner. Other peripherals such as the RTC, timers, CAPCOM and PWM are included in the E2 and are accessed normally.

Single Chip Mode

The E2 bondout supports internal ROM or FLASH devices from 32K to 512K. Remapping of this ROM to segment 1 is fully supported. The ROM is simulated with fast RAM in the emulator. There are menu items in the emulator software to inform the E2 it is in ROM mode, how much ROM it must simulate and if remapping is needed.

The user program must set the ROMEN bit and perhaps the ROMS1 bit in the SYSCON register during the initialization period. This is easy to do. These values can be manually poked in with the emulator for small experiments.

The data path for the ROM is 32 bits and this will be reflected accurately in the trace memory in the Opcode column. All ports are available for the user: none are used for address and data buses. The appropriate information is obtained from the IA and ID buses.

Emulation Accuracy

The E2 directly supports the C167, C164 and C161. It can further support the C165 and C163 and other derivatives except for the SAB 80C166 itself.

The E2 is not an exact copy of any of these controllers so some differences will occur. The CPU core is exactly the same so this reduces differences substantially.

Conclusion

Experience has shown the E2 to accurately emulate C166 family members up to its design speed of 33 MHz. A well designed In-Circuit Emulator can substantially reduce your debugging time and grief by using the power of the E2 bondout from Siemens. Nohau is committed to providing the user with all E2 features as technically feasible.