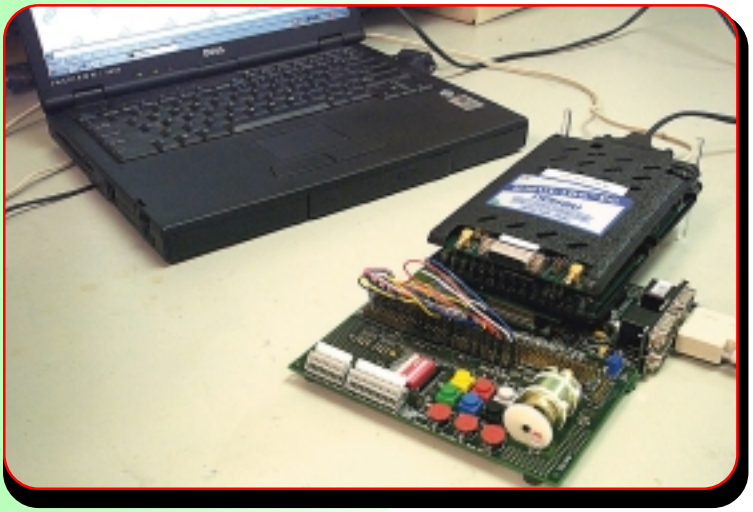
Case Studies

on connecting to C166 and ST10 target systems.

using....

Nohau In-Circuit Emulators



August 22, 2001

Nohau
275 E. Hacienda Avenue
Campbell, California 95008
Email: sales@nohau.com

Tel: (888) 886-6428
Tel; (408) 866-1820
Fax: (408) 378-7869
Web: www.nohau.com

Nohau Case Studies: Getting an Emulator to Run On A Real Life Target

by Robert Boys August, 2001 Version 3.1

Purpose:

This article illustrates three case studies of actual issues concerning the connection of the Nohau EMUL166 in-circuit emulator to real customer targets. The purpose is to share the problems and solutions that each of these people faced in order to help you get your emulator connected to your target. While these cases are based on the Infineon C166 family, the problems are similar for other semiconductor manufacturers.

We examined a few situations with our customers and found some common problems and solutions that were common to nearly everyone. The purpose here is not to blame or shift responsibility but to help our customers to meet both of our aims: to get your development project completed as soon as possible with the minimum amount of grief for everyone involved. Nohau is interested in being fairly frank in this discussion since we really need to get to the root of the problems.

Emulators are complex yet interesting devices and somewhat mysterious to most engineers and when something is not working it is most natural to blame the emulator first and most. To blame the emulator is detrimental to resolving the real root problem which is necessary to get your projects done with all dispatch as possible.

We should note that not all of our customers have this type of trouble. Most of our customers are able to get their targets running in a satisfactory manner with an appropriate amount of effort. These cases here were selected because they illustrate an interesting range of problems and not because they are representative of most situations.

The Nohau technical support team is in an interesting position since they see the issues faced by a wide range of embedded developers around the world. They see issues from the viewpoints of many people. There are other engineers having similar problems as you and we at Nohau hope this article will help you get your target up and running.

A) The Signs:

The stories are similar. This is what a customer having difficulty getting their target and emulator running together tends to say to our technical support engineers:

1) The target runs with the real chip installed but not with your emulator connected.
2) The emulator will not run in the target but works fine stand-alone.
3) Your competitor's emulator works in the target. (there is a reason for this - see "*The Explanation of This*" on page 7)
4) There is nothing wrong with our target. We believe the problem lies with the emulator. We have very little faith in the emulator and therefore believe it to be causing all the trouble.
5) The emulator is a piece of junk and we want our money back. (not true: we will show this)

B) In addition: these are a few facts that tend to be common to the situation:

1) The person is usually the software engineer, not hardware and did not design the board.
2) He or she has spent a great deal of time to get this working and is frustrated to varying degrees.
3) The pressure is on to get going with the development of their software, and time is now short mostly due to 2). This engineer has spent so much time trying to get this working, he or she has started to go "around in circles" and in fact is now getting nowhere. *We wish engineers would call us sooner.*
4) The hardware engineer is available but generally feels his/her part of the development is completed at this stage. The software and hardware people are usually in slightly different departments.

5) Often the hardware engineer is confident the hardware works fine and the software engineer is confident that there are no bugs in the software to cause these problems.
6) Sometimes it is hard to get the customer's code, schematic and hardware for security reasons. As time goes on, this usually gets easier to accomplish as the situation gets more desperate.
7) The customer wanted to use the emulator for some important debugging situations to enhance the quality and completeness of their work. The emulator is an essential component of their tool chain.

C) From our viewpoint we see these axioms:

1) We have many customers successfully using the emulator in similar configurations.
2) We have a great deal of experience and information to share with customers having difficulty connecting to their target and we have technical support departments spread around the world to accomplish this.
3) Nearly all of the time the fault lies not with the emulator but with operator error, adapter failure and target defects in this order. To find out the rest of the stories: please read on!
4) There is a document included with the emulator Getting Started Manual and on the Nohau website called ***Connecting the EMUL166 & EMUL-ST10 Emulators to your target system.*** Most of the engineers in this case study have read it. This document is appended at the end of this article.
5) Microcontrollers have become more complex with more specialized peripherals. The configuration and initialization of these devices is becoming more critical and difficult to accomplish.
6) The emulator and target system is essentially untested. Nohau has tested the emulator with various targets and application code: but not together with *your* code and *your* target. There might be some problems that we will have to solve together.

Case Study #1

Background:

The customer experienced all of A) except for 3). This might have shown their problems not to be with the Nohau emulator as you shall now read about. In B) they experienced all except for 6). They were rather frustrated at their inability to get their target going and were willing to send their target and schematic to Nohau in California. This was the situation that precipitated the writing of the document listed above.

The Target:

The target used an Infineon C167CR chip and had the four standard QUAD Connect connectors around it to connect the emulator to the target. It had an 8 bit external FLASH device and an 8 bit RAM device. The FLASH device was connected to chip select CS0 and the RAM device to CS1. The target has other various peripheral chips including a buzzer and a LCD display. The regulated 5 volt power supply regulator was built on the target board.

The Symptoms:

When power was applied to the target without the emulator connected, a test program located in the FLASH was executed at RESET causing a message to be displayed on the LCD display and the buzzer would sound at some interval. So it was confirmed: the target appeared to function normally stand-alone.

Connecting the Emulator:

The emulator was connected using the port P0 initial setup configuration and the clock jumpers were set to use the target clock. All memory was mapped to the target with the appropriate emulator jumper settings (AUTOMAP off and P6.0 on). The Nohau software Seehau was started which promptly crashed. Seehau is not really "crashing", but exiting properly with a fatal error. There is a difference and we will discuss this under "Seehau Crashes". Several more attempts were made with the same results. The emulator was removed from the target and it ran fine stand-alone after the jumpers were reset for internal emulator clock and the AUTOMAP jumper was installed. It was then confirmed that the emulator indeed failed to work in the target system. So, we set to the task of finding out why. We learned quite a bit as we continued.

The Clock Signal:

The emulator bondout chip which replaces the target CPU needs a clock signal to operate. If none is present Seehau will exit with an error because it timed out waiting for the bondout to start which cannot because it has no clock. If the CPU is set for PLL operation by port P0, the bondout will oscillate at a natural frequency of about 3.5 MHz. This effect cannot always be counted on.

We measured the clock signal with an oscilloscope on the emulator ECLK pin (see Getting Started Manual) and found nothing there! We switched the clock jumpers to select the internal emulator crystal and were subsequently able to start the emulator. The ECLK now measured 20 MHz which is correct.

In this case, there was some reason the oscillator on the target would not start the bondout CPU. We decided to wait until later to determine the cause of this problem. We used the emulator clock for now.

The Address and Data Bus Configuration:

We assumed that the customer had usable code in the FLASH device since the target was able to start by itself and run. We used the emulator source window to try and look at the disassembled code but saw nothing but nonsense. With the 166, you soon get to recognize the standard start-up sequence generated by either the Keil or Tasking compiler. The first instruction is usually a jump at address 0 to the start of the user code: often at 200 hex. Nothing similar to this was to be seen.

The emulator has a window that will identify the settings of port P0. This port's levels are sensed by the CPU at reset time to determine the target bus settings that must be used to start the system. This includes clock configuration, bus width and if it is multiplexed or not. Pullup resistors on P0 are used to select the various configurations. The emulator has a window that displays the setting of P0 as seen by the emulator at RESET and it said the target system reported a 16 bit data bus and a multiplexed address and data bus. This was clearly not so. With only one 8 bit FLASH and one 8 bit RAM devices, this had to be an 8 bit bus. A quick look at the schematic indicated that the target used demultiplexed address and data busses.

The emulator can be set to ignore P0 and use values selected by the user. We did this and selected an 8 bit data bus demultiplexed.

At this point we could still not see what looked like valid instructions in the source window. This is a "show stopper". If we are unable to see the proper contents of the FLASH, the emulator will never be able to run. We left the port P0 override activated since we knew that often there are more than one problem in the system. Why port P0 does not show the proper levels at RESET is important, but the emulator will override this for now and we can return to it later. Crucial is why the emulator is unable to see the FLASH contents.

The Emulator to Target Connectors:

We decided a more detailed visual inspection was in order. We should have done this in the beginning. The emulator connected to the target through four Samtec DIP connectors of 40 pins each. I noticed that all of the inside connectors (all 160 of them) were not soldered to the board! Only the outside rows were. No matter what, the emulator, or any emulator, had no chance of working like this. Only half of the pins were connected! Even assuming a few of them would make touch contact through pressure from the soldered pins and that not all pins are needed for operation since many would be unused ports and so on: the chance was very high that at least one extremely essential pin would be unconnected.

We had the pins soldered in the Nohau lab. We tried the target system stand-alone to make sure nothing was changed and it was not. It still worked by itself. We then connected the emulator to the target.

Adapt Mode:

The C167CR controller mounted on the target must be disabled when using the emulator. This is done by putting the target chip into the "adapt" mode. This tri-states the target controller and allows the bondout controller to take its place. This is done by ensuring P0L.1 is low during RESET with a pull-down resistor. The emulator contains such a resistor with a value of 10 Kohms which is normally sufficient.

There were some problems and this voltage measured around 1 volt which was marginal. We suspected the target chip might be trying to start causing serious conflicts with the bondout controller in the emulator. The target had some logic circuitry that tended to pull this port pin high. Adding another 10 Kohm resistor (for 5 K total) gave a voltage reading of about 0.4 volts.

IT WORKS! (well, sort of)

This time we could see what looked like good disassembled code in the source window. We pressed GO and the LCD screen lit up and the buzzer buzzed in a fashion similar to the stand-alone operation. Unfortunately, the target only worked for about 15 seconds and then stopped.

More Checking:

We looked into the clock problem again to see if the problem was here. We have seen problems where the clock drops from 20 to 5 MHz and this usually causes the program to stop functioning normally. We checked the clock connections between the header on the target and the target crystal and found no connection. Something was not right here.

Conclusions:

At this point we had decisive proof that the target board traces were incorrect for the clock and we were suspicious of other traces. There was ample proof that there were likely other problems with the hardware. We never did check why port P0 was sending the wrong signals to the emulator.

It was very clear that the reason the emulator did not work was the unsoldered connections plus more. We documented the problems we found and returned the target system and the emulator to the customer for their hardware designer to correct his issues.

After this they were able to successfully complete their software project using the Nohau emulator. They apparently decided it was not junk after all. They gained faith that the emulator was indeed working correctly. We did not have to change or repair any Nohau software or hardware.

Case Study #2

Background:

The customer experienced all of A) except for #3 and #4. They had some problems with the CAN port with the target CPU running. But nothing worked with the emulator connected. For B), the target was obviously a prototype so the hardware people must have been still working on it. The software engineer responsible came in to Nohau with the target, schematics and code since they were a local company. The entire system was setup in the Nohau main conference room in Campbell.

The Target:

The target contained a C164CI and the main problem was the CAN channel was not working properly even with just the target chip operating. Most of the time the emulator would not start up but on occasion it would. As stated above, the target was a prototype. It had two RAM chips, some analog circuits and a CAN physical layer interface. It had parts half soldered on and blue wires running all over and not all were connected at both ends. On some of the chips, not all pins were connected to the circuit board.

The customer needed the emulator working in order to debug their CAN software to find out why it did not work properly.

The emulator was connected through a commercial adapter using a solder down adapter that was soldered in place of the CPU. They had other targets to demonstrate that the target worked without the emulator. We proceeded to look at these issues in this order.

The Adapter:

The E2 bondout chip has the chip selects (CS0-CS4) coming out as alternate functions on Port 6 (P6.0 to P6.4). This is the same as the C167 which the bondout is very similar to. The C164 has the chip selects (CS0-CS3) on Port 4 (P4.3 to P4.0). If Port 4 is needed as a parallel port on the target, the adapter must pass these port signals straight through. If you want to use the chip selects on your target, these must be routed from Port 6 on the bondout to where Port 4 is. The adapter supplies jumpers to accomplish this. The customer needed to have chip select 0 and 1 come from the bondout chip to the target.

You guessed it: the adapter jumpers were connected to put Port 4 of the bondout to the target and not to the appropriate pins of Port 6. There was no chance for the RAM devices to be properly selected. The cause of this problem is that the adapter manufacturer (not Nohau) provided no documentation and Nohau customer support failed to notice this issue. Setting the adapter jumpers correctly solved this problem.

RAM Chip Enable:

During a conversation with the customer, we learned that an unused RAM chip was disconnected from the target by having its enable pin disconnected and left floating. The address and data pins of this RAM chip were still connected to the target busses. We felt this could cause bus contention if the enable pins went active, so the customer removed the RAM chip entirely from the target board.

Bus Width:

During a visit to the customer's site by a Nohau engineer, the emulator port 0 P0 override was set to 16 bit data bus. This was in error since the schematics showed an 8 bit bus. The emulator override was turned off.

Bondout Power Supply:

The bondout needs a 5 volt supply. This can be provided by either the target or the emulator. In stand-alone operation, the bondout obviously must be powered by the emulator since no target exists. Installation of the T PWR jumper JP5 accomplishes this.

If a target exists, Nohau recommends powering the bondout with the target 5 volt supply unless the target consumes very little power (< .3 amps or so). Leave JP5 off to power from the target. It is important to make sure the target is never supplying power when the emulator is not. Proper sequencing of the two supplies is therefore necessary. The two pins of JP5 then provide handy test points to check the target and emulator power supplies.

In this case, JP5 was not connected so the bondout was supplied from the target. A voltage check at the pins of JP5 showed 5 volts from the emulator and only 3.5 volts from the target. More inspection showed the adapter was defective and the target 5 volts was not coming up to the bondout.

The 4.4 volts came from the bondout being supplied from its I/O pins, mainly the CAN receive line. Not a very satisfactory arrangement. It did explain why the emulator would function on occasion since the CAN line did raise the bondout to 4.4 volts which must have been just enough to get it started sometimes.

This was the show stopper and a couple of wires from the target supply to the top of the adapter solved this problem. After this fix, the emulator started normally.

Other Noticed Problems:

While we were working on this target a few additional items were discovered and reported to the customer. These issues did not seem to prevent the emulator from working properly:

1) The customer's target contained an opto-isolator for the CAN interface. The bondout signal would drive this LED by pulling the LED low. The other side of the LED was connected directly to VCC. This would be considered an undesirable circuit, because no resister would limit the current flow.

2) The customer's pod would not verify the contents of the loaded code. Immediately after loading and performing a verification, the pod would show a verify error at address 203. This problem could have been detected by performing the verification after the load. This problem was resolved when the RAM chip with the floating enable pin was removed.

Connecting the EMUL166 & EMUL-ST10 Emulators to your target system.

The engineer was asked if he read this document and he said he did. We reviewed it with him in an attempt to determine why it did not help him find his problem. One of the instructions says to get the emulator running stand-alone and then connect it to the target and without changing any settings, try and start it.

If the target is in reasonably good shape (i.e. no address or data lines shorted) the emulator will start normally. The idea here is that the target must not create such an abnormal situation as to stop the bondout from starting.

This software engineer said that when he got to this step the emulator quit working. Rather than trying to determine why it failed under these conditions, he continued on to the next step. If he had recognized the seriousness of not being able to get the emulator to work with this step and tried to find out why it failed at this point - he probably would have discovered the biggest problem which was the bondout voltage being low.

He was asked to tell us if this step's assumptions were correct once he had his target working properly. He agreed and later reported that the emulator starts using this step. We will strengthen the language in the application note as appropriate.

The Explanation of This: Nearly all Nohau emulators connect the emulation processor pins directly to the target system. This is done without gates or other logic to isolate the two systems to prevent serious timing and compatibility problems. Our customers tell us they prefer this method in order to provide the most accurate emulation possible. They prefer for their target to see the genuine CPU pin characteristics rather than those of a LS244 for example.

This is the reason why sometimes our emulator will not work in a given target while one of our competitor's will. They have buffers between the bondout and the target. A defective target may stop the Nohau from operating while the competitor's probably will keep on running. This is because they are not really connecting to the target except when specifically addressing a memory area on this target.

The result is that defective targets tend to show up earlier using the Nohau emulator and much later in the design stages with our competitor's. Our experience is that most designers prefer to know about problems early in the design stages when changes are easier to make. They also are not keen on rewriting portions of their supposedly bug free software designed on a defective target when problems are discovered. Hopefully they are before shipment to customers.

Conclusions:

The main problems were the missing bondout power and the jumpers for the chip selects being set in error. Neither Nohau's software or the pod hardware had to be modified to be able to run the customer's target. A systematic checking of the system displayed the problems clearly.

One thing we learned was that there may be more than one problem that can cause the emulator to not start. If finding an error did not fix the problem, setting it back the way it was will only reintroduce this error and prevent the emulator from starting. Finding the next and last error will not get the emulator running because the last error you discovered is still there. Fix the errors as you continue your troubleshooting techniques.

Once the emulator comes to life, it is a good idea to continue looking for additional problems that may cause problems while you are familiar with the system. This paid off for the customer in this case. I told this engineer I would be writing this and asked if there is anything he thinks I should mention that he thinks would help. He said "be more patient".

Case Study #3

Background:

This customer returned the Nohau emulator four times! He was not able to get it going in his target although it worked fine stand-alone. During all this time, he had also been evaluating a competitor's product and did not find it very satisfactory for his needs. After the fourth time, he purchased the competitive model since it would start up. He preferred the Nohau hardware, software and technical support: if only we could get it to work. We asked for one more chance to find out why he was having so much trouble since it did not make any sense. He agreed.

The Target:

The target used a C164CI and was designed to accommodate the Nohau ADP-164-EVAL adapter. The C164CI ran in single-chip mode therefore there is no on-board RAM or ROM. The various I/O ports and the CAN port were connected to various peripheral chips. There is no external busses activated at all in this case. We were told there was nothing unusual about the circuitry. We never did see the target or its schematic as the problem was solved before it got this far.

The Symptoms:

The emulator worked fine stand-alone in single-chip mode. When the customer tried to run the emulator connected to the target it timed out and Seehau exited.

Nohau technical support had worked with the customer but there was no solution. The fact the C164CI was in single-chip mode seemed to confuse the issue when it should not have. The emulator displays information slightly different due to the 32 bit emulation of the ROM.

The ROM Emulation:

The C164CI has 64 K of internal OTP ROM. This ROM is connected to the C164 CPU via an internal 32 bit bus to speed up execution. The opcodes are fetched in 32 bit chunks rather than 16 for external memory arrangements. The emulator provides emulation RAM 32 bits wide that connects to a special 32 bit bondout bus with the E2 bondout chip. This gives us access to the user code for trace display and since it is 32 bits wide, the user program can run at full speed.

One problem is that an accidental write to the code will be ignored by the real OTP ROM while it will be modified in the emulation RAM. There is no access security since Nohau believes the customer must have unimpeded access to this code area for easy modifications. It is easy to confirm if the loaded user code is intact with the Seehau "Verify Loaded Code" menu selection.

We did not suspect any problems with the single-chip emulation but paid a great deal of attention to it.

Other checks:

The customer had checked many things including the clock speed, power connections and jumper settings. He apparently was doing everything correctly. There was some confusion concerning the switching of the chip select signals from port 6 to port 4 but this was quickly cleared up. See Case 2 for details.

The RESET Jumper:

During a conversation with the customer about his RESET circuitry, the suspicion was raised that perhaps the RESET time was excessive and timing the emulator out. This had been experienced on a RIGEL C167CR board in the past. The customer was asked to pull the RESET jumper from the emulator and try again. The customer screamed "IT WORKS !" and everyone breathed a sigh of relief.

What Happened:

Every microcontroller needs a RESET signal of some sort. Generally speaking, a CPU can be held in RESET for eternity and it will sit there waiting for the RESET to come off. When this happens the CPU will start operating from a known state in the usual manner.

In this case, the target RESET had a fairly large capacitor (10 mfd) which caused a RESET time of about 1 second. The time constant of the RESET signal had no consequence to the customer and this had been arbitrarily chosen. This had big consequences to the emulator.

The emulator contains a bondout microcontroller which substitutes for the target controller. This bondout closely resembles a C167CR plus some additional internal busses "bonded out" for emulator use. This controller could be used as a stand-alone controller. It contains an identical RESET pin to the C166 family. It also can be held in RESET indefinitely.

The emulator on start-up must run some code on this controller to gain control of it. This is called monitor code and when this code is being run the emulator is said to be in "Monitor Mode". When user code is being run the emulator is said to be in "User Mode". The emulator hardware has the ability to switch between these modes very quickly. This is done with the triggers, START, STOP and Single Step icons in Seehau.

The monitor code is what the bondout starts running immediately after it comes out of RESET. Control is passed to Seehau where the user can then start the debugging process. The emulator does not wait forever for the RESET to go inactive. At some point in time, the Seehau software must inform the customer that there is something wrong with the system in that it is unable to acquire control of the bondout chip and therefore times out. This is exactly what was happening in this case and how it was fixed so readily.

Please note that this issue is only when the emulator is started from a cold start. Once the monitor code is running and Seehau has acquired control, you can hold the RESET on for as long as you wish.

The emulator RESET jumper simply disconnected the target RESET line from the bondout controller. If there is a need to reset the bondout from the target, the customer may have to adjust the RESET hold time to a value acceptable to the emulator and this is normally not a problem.

Conclusion:

The customer tested the emulator for a few days, comparing the Nohau to the competition and kept the Nohau. They were a good source of additional feedback to the emulator designers and we are grateful for their help. They successfully completed their project and are using the emulator on new projects. You will find the suggestion to remove the RESET jumper in the hints to connecting to target systems.

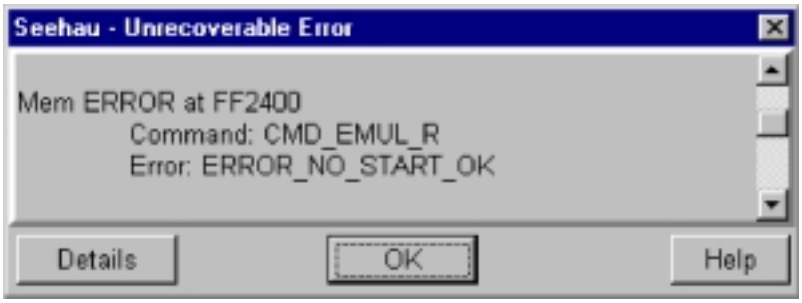
Summary:

We have presented three cases that we feel are relevant to embedded designers. There is one part left concerning software crashes and then a reprint of the application note concerning connecting to targets. We have focused on the Infineon C166 family since it is very popular and we take advantage of any opportunity to increase awareness of getting these projects running as soon as possible. We did not choose the C166 because it has more or less problems than other microcontrollers. We could easily have presented interesting cases about the Motorola HC16 or HC12, the Philips XA or Intel 196 and maybe someday we will. We could likely write a small book on our experiences with the 8051.

Seehau Crashes:

There are various ways the Seehau software can exit or crash and not all result from Nohau bugs. It is important to recognize this to reduce frustration and to determine the cause of the crash faster.

1) **BUGS:** Ok, we have to admit that there are probably a few bugs in our software. We track these down using information provided by internal Nohau testers and users, our reps and our customers. These failures appear as access violations reported by Windows. As we repair these problems, these errors become more rare. At this point, we suspect most "crashes" are from the Windows operating environment rather than from Seehau.
2) **Emulator to PC Connection Issues:** This screen shot shows what we get when the Seehau software is started without an emulator connected to it. Note that error indication is displayed. We are working on making it more readable. Nothing different can be expected as no software can function without its hardware connected. We suppose we could put Seehau into demo mode when this happens but this will only confuse the issue and delay the resolution of the problem. Similar problems can occur if the emulator power connection is not made or the communications cable is not installed. These errors are



not caused by bugs in the software no more than a car is at fault for not starting with no gasoline in it.

3) **Emulator Configuration:** Nohau provides for jumpers on the emulator body for certain configurable items rather than by using software settings. This is done for two important reasons.

   First, software settings require some sort of logic circuitry that will introduce capacitance, inductance and delays to the important bondout/target connection. Everybody wants this connection to be as direct as possible and jumpers are the best method to do this. Some features are software driven (such as some types of software mapping) because this is the only practical method to do them. We cannot have hundreds of jumpers to provide memory mapping down to one byte for example.

   Second, it is much easier to determine at a glance all the jumper settings than to go through all the potential software settings to determine your present configuration. A photo of an emulator and its jumpers are much faster and less confusing to confirm that by using software. This is especially useful during technical support sessions.

4) **Defective Target Hardware:** This is a common issue. The idea with an emulator is to mimic the target processor as accurately as possible. The semiconductor and emulator manufacturers spend a great deal of effort to ensure this goal. The first point in 3) above is a good example. We want the emulator to not work in a target system if there is a fatal design flaw in it. Examples are shorted address and data busses or poor clock circuits. It is generally better to find these problems sooner than later. Because the target chip will operate is not a guarantee that the target hardware is functional and we have illustrated this in this article.

   An interesting story concerns an 8051 derivative. The hardware engineers discovered that a peripheral chip on their target could consistently be operated just outside the 8051 bus specification from the chip manufacturer. This peripheral was staying on the bus longer than it should have. This helped them with a critical bus timing issue (it concerned the ALE strobe) and simplified their design. The target appeared to work properly with this timing modification.

   Later in the development the software engineers were faced with a particularly nasty bug that caused their system to stop working with a seemingly random pattern. This bug did not occur often and with no predictability, was impossible to find. They decided to purchase an emulator to help solve this problem and purchased a Nohau EMUL51 emulator.

   They had a great deal of trouble getting the emulator to start - it simply would not except for perhaps once in a while. The story was the same: the target works stand-alone (except for the intermittent bug) and your emulator is no good. The Nohau rep worked with them and in conversations heard about the timing issue with the ALE strobe. He recommended they modify the target to remove this "feature" and the emulator started up. The emulator was telling them *exactly* what it is supposed to: that there was something wrong with the target!

   Of course, once the target was modified to operate within the specs of the microcontroller, the intermittent problem magically vanished. This emulator certainly paid for itself.

5) **Compiler Software Initialization Configuration:** Most microcontrollers have many registers that need to be configured to allow the operation desired. This includes system, bus and peripheral configurations. If these are not initialized correctly, the controller will not function as expected. Many times this initialization is handled by the compiler according to user entered information. The initialization routines generated are normally executed immediately after RESET.

   It is possible that a setting can be so wrong as to make the system non-functioning. An example is if the system is 16 bit data bus and the controller is set to 8. The controller will crash as soon as the User Mode is entered and it is possible Seehau will exit with an error.

   Many Nohau emulators are capable of overriding such settings to help track down the cause faster than recompiling the user code.

There are more scenarios, but the message is made. Not all crashes are the result of software, firmware or hardware bugs of the emulator system.

Who Helped:

Joe Pennese	Nohau Representative Northern California
Jorgen Anderson	VP Engineering at Nohau
Scott Axlund	ST10 Technical Support Engineer at Nohau

Many other Nohau personnel for tiny tidbits of information here and there. Can't list them all.

Our anonymous customers some of whom I thanked personally.

For comments and suggestions about this document contact Debbie Richberger at debbier@nohau.com

Connecting to Your Infineon or STMicroelectronics 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 need to have the complications of your target buses making things harder to resolve. Make things simple at first.

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. Nohau makes a set of useful isolation boards. Please call your local rep or Nohau direct.

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 of the Getting Started Manual for the EMUL166-PC. 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 target clock pins are correct. JP10 and JP7 should be in the lower position. Please note that the microcontroller documentation specifies the two crystal capacitors to be of different values. Having the same value for both can create some startup problems when the added distance of the emulator adapters is added.

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. This can damage the bondout.

8) Consider purchasing an evaluation board to serve as a reference design. Phytec 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 boards are like this. The early C161 board does not use the Siemens bootstrap mode. The C167 board uses CS0 for the FLASH and CS1 for the RAM and this setup is very easy to use. Adapters are very inexpensive for these boards. See www.phytec.com.

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. Stuck bits, flakey values and croswed bits are very common and will obviously stop the system from working properly.

continued.....

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 byte, this nearly always means adapter problems or crossed data lines on your target.

13) Pay special attention to the Port0 configuration resistors at start-up. Many problems result from incorrect settings. Do not trust the target schematic diagram - ohm out the connections by hand in case there are layout errors. Make sure the RPOH register (visible in the register window) is what you expect.

    If you click on the Seehau REST icon with the RESET jumper installed, the RESET signal will be sent to the target and reset it. The RP0H register will contain the upper byte of Port 0. If you get random values in RP0H when you cause this reset, suspect that something on the target is driving the Port 0 pins during this time. Usual causes are floating Read, Write, CSx and BHE pins. At RESET, these pins are tri-stated in the bondout chip and your decoding logic needs to ensure no devices on your target are activated at this time and specifically that they do not drive PORT 0 which is the address and/or data lines in various external modes. Appropriate 10 Kohm pullup or pulldown resistors can be used.

14) Make sure your target is in bootstrap mode or not - whichever you expect. The EMUL166 emulator has an icon labeled "BSL" in the Seehau software to indicate whether bootstrap mode is selected or not according to what the emulator sees on Port0. The ST10 does not have this icon. Call Nohau technical support if you need help. The bootstrap mode is set with Port0 bit 4 (POL.4) being low at RESET.

15) Make sure the chip on the target board is in Adapt Mode. Consult the Infineon or STMicroelectronics datasheet for more information. The emulator uses a 10 Kohm pulldown resistor on PORT0 POl.1 and under normal circumstances this is sufficient. In cases where the target circuitry provides pullup resistance on this pin, an additional external 10 K resistor may be needed. Erratic operation will result from the target CPU being marginally in Adapt mode.

15) If you find a problem - fix it before you move one to the next hint. Ignoring a problem will not help you get your system working. Your problem will not get fixed until each anomaly is corrected. Do not assume a problem is insignificant to your situation. These are valuable clues that must be attended to.

16) Please remember that there are many legitimate reasons that the emulator will not work in the target and the real chip will. This is an important clue and I believe this document clearly illustrates the importance of understanding the appropriate issues that cause this effect.

One more for the road:

Thanks for Larry Claassen for relaying this gem ...

Background:

The customer used a Nohau EMUL166 to successfully complete the project and had delivered several thousand units to customers. The project was successful except they were never able to get the target to run from the target FLASH devices using the emulator. The real chip (as usual) worked fine.

The Target:

The target used an Infineon C167CR chip and had an two 8 bit external FLASH devices and two 8 bit RAM devices operating on a 16 bit bus. The FLASH devices were connected to chip select CS0 and the RAM devices to CS1 in the standard manner. The target has other various peripheral chips. This target was used in a low level critical area in the automotive industry.

The Problem:

During initial system development, the code was developed successfully using the emulation memory. The target ran perfectly. A problem arose when trying to run the code using the target FLASH devices. The target program would not run reliably using the emulator but ran fine when using the real C167CR chip. They were never able to resolve this difficulty and assumed the emulator had a defect or a bug. The emulator is capable of running programs from target FLASH devices.

The company did their final testing using the real chip and released the product to the market. Several thousand units were sold, mostly in the USA. The emulator was trying to tell them something.

Some time later, they wanted to upgrade the software and they setup their debugging system including the emulator. They soon experienced the same problems as before when running the code from the target FLASH devices. One of the engineers remembered this problem and was prepared to dismiss it again but another decided to look into the matter further.

The Investigation:

The engineer looked at the problem not as something wrong with the emulator but as a potential problem with their target or code. She first looked at various issues such as the address and data buss structures and then decided to focus on the start-up code since this is where these are first configured. She compared the SYSCON, BUSCON and ADDRSEL registers obtained from the emulator with the design specifications and the compiler start-up macro. She quickly discovered that while the hardware designer specified one wait state to be used with the FLASH device, this was not implemented in the compiler macro and hence not in the resulting object code. When she changed the emulator's settings in the bits MTCT in BUSCON0 to provide one waitstate, the emulator successfully and reliably ran the code from the FLASH memory.

The Solution:

Subsequent investigation revealed that the bus timings with the real chip at zero waitstates was marginal and a significant amount of design risk existed. A product recall was needed. They were also looking for faster FLASH devices and if they could not find suitable replacements, they would have to recompile and retest all their existing code. The emulator was telling them they had a problem and they simply needed to trust it a little more. The emulator uncovered the problem promptly so they could deal with it on their own schedule.