The Software Engineer's Guide to In-Circuit Emulation

How to increase your debugging skills!

ICE Technology Nohau Brand Embedded Systems Tools

Introduction

Software debuggers and target monitors offer economical debugging capabilities sufficient for many applications. In-circuit emulators offer additional advanced real-time debugging facilities. There are major differences between each of these options in terms of debugging power, real-time operation, and non-intrusiveness. Some debuggers, such as those from Keil, PLS and ChipTools can do advanced tasks when interfaced to an emulator.

With these debuggers, one can load, single-step and run programs. Software breakpoints can be set and memory can be examined. Code Coverage and Performance Analysis provide statistical information. Software debuggers do a good job, but have some shortcomings that can be crucial in detecting and fixing some of the more elusive bugs or in special circumstances.

Interesting "what-if" Scenarios

What if you are debugging code that is in ROM? Or even trickier: ROM inside the chip. What if you need the serial port that debuggers commonly use to communicate with the controller? What if you want to detect a certain situation: such as a write of a certain value to an external peripheral and if another variable equals 6, and then stop the execution? Or record these bus cycles? What if a bug causes your program to jump off into never-land and you need to know what was happening just before this event? Hardware Breakpoints, a non-intrusive connection, Conditional Triggers and Trace Memory are solutions to these problems and more. You need an emulator.

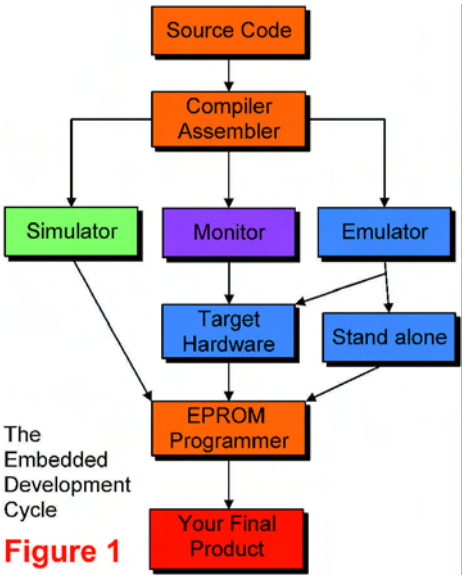
This article will explore some advantages offered by emulators and how you can produce better code by using one.

The Development Cycle

The typical microprocessor development project begins with a C compiler producing an object file from your source code. This object code will contain the physical addresses and some debugging information. This object code can be executed and debugged using a software simulator, a target monitor or an in-circuit emulator. A most undesirable method is to program an EPROM and simply run the target system.

The program is debugged by setting breakpoints to halt execution at selected instruction locations. When execution is halted, the memory and register contents are examined for clues to help find bugs.

The debugged object code is re-compiled removing the debug information and producing a file in a standard format such as Intel HEX. This file will be stored in the final product's nonvolatile memory such as EPROM or FLASH. This process is illustrated below in Figure 1.



The Embedded Development Cycle

Figure 1

Why do we need emulators?

There are some cases where an emulator is needed to resolve difficult to find bugs. In all cases an emulator will pay for itself by providing you with decreased debugging time, ease of system integration, increase in reliability and better testing procedures. Often, designers use both an emulator and software debugger during different project stages, especially in larger design teams.

Software simulators and debuggers offer only a few features beyond breakpoints such as displaying port contents and code coverage. There are no means to detect events or conditions and then act on them, and certainly not in real-time. There are also no means to record controller bus cycles to determine what actually happened to the program flow. If your microcontroller has on-board EPROM or FLASH memory, and is running in single chip mode: only an emulator can debug this scenario without serious intrusion and consumption of controller resources.

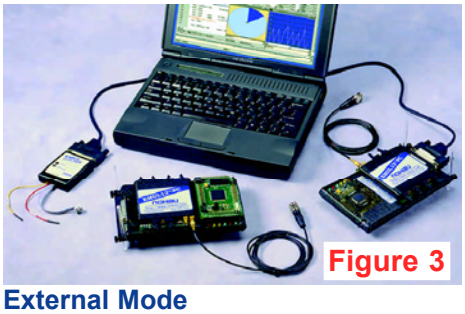
Figure 2

In-circuit emulators can easily do these tasks and more for you. Emulators are the bridge between software and hardware. At some point in time, you have to run your program in real hardware. An emulator will easily help you accomplish this. This article examines how emulators will help you with your debugging sessions.

What exactly is an emulator?

"An emulator is a computer that engineers use to design other computers" is the most basic definition . Emulmulators simply replace the microcontroller in your target system. The emulator behaves exactly like the processor with the added benefit of allowing you to view data and code inside the processor and control the running of the CPU.

Figure 2 shows a Nohau emulator for the C166 and ST10 microcontroller families. Figure 3 shows the Nohau emulators for the Motorola HC12 family. These emulators are compact handheld emulators that go anywhere you and your laptop can go. They are all powered with a 5 volt supply.



External Mode

External mode is when the program memory and perhaps some data memory is located externally to the controller. This is the classic situation where the target board contains a microcontroller, some EPROM or FLASH, RAM and some type of peripheral chips. The address and data busses are available to access this memory and are the only means for an emulator to gather information and control the CPU. Therefore information about internal registers is not available in real-time. The address and data busses may not be used as general I/O ports. Production chips are effective for emulating this mode as are the special emulation chips.

Internal or Single-chip Mode

Internal mode is when program and data memory is located in the controller chip in the form of FLASH or EPROM. This method is becoming preferred for embedded designs due to its low cost. The address and data busses are not accessible to the user. These busses are then available as I/O ports. All program execution occurs in the internal ROM. This mode requires a bondout, Enhanced Hooks, JTAG or BDM chip for emulation.

Nohau emulators generally run these chips in external mode and use special circuitry to reconstruct the I/O ports. This process is transparent to the user. BDM and JTAG chips are special situations.

It is possible to embed a monitor kernel in the ROM, coexisting with the program code. The user can activate the kernel with a switch connected to an I/O pin. This will allow communication with a debugger via a serial port. Of course, some system resources need to be reserved for this scheme but it is still useful.

Bondout Chips

Bondout chips allow single chip emulation by providing extra pins connected to internal CPU nodes. Infineon C166, ST Microelectronics ST10 and Intel 196 and x86 emulators generally use bondout chips for effective emulation. The Nohau EMUL166-PC and EMUL-ST10 are good examples. It is possible to make emulators for these families using production parts but performance is usually lacking in internal accesses.

The Infineon C167 and the ST 167 and 168 microcontrollers use Ports 0 through 4 for address and data busses in external mode. Which ports are used depends on the bus configuration as programmed at CPU initialization. The bus configuration can be changed “on-the-fly” while the processor is running. From 16 to 40 port bits can be used as address/data lines. These then cannot be used for general purpose I/O ports.

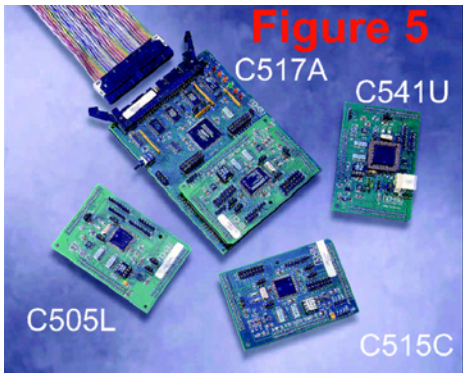
If the microcontroller is a version that contains internal ROM that contains program code and data, it can be run in internal or single-chip mode. Ports 0 through 4 can be used as general purpose I/O pins. The emulator bondout controller allows operation in this mode. The target controller will be disabled and the bondout controller will take its place.

It will use the emulator RAM to simulate the ROM. All ports (serial and parallel) will be free for use by the target system. The internal CPU busses will be made available to the emulator providing high performance and visibility. The Destination, Source and Value fields in Figure 4 are examples of internal busses made visible. A bondout chip can emulate both internal and external modes and uses no resources from the controller. Not all emulators are capable of this.

Hooks & Enhanced Hooks Chips

Hooks and Enhanced Hooks chips take advantage of unused cycle times on various pins to provide the address and data busses. Hooks and Enhanced Hooks chips are used with the Infineon C500 family, Philips, and Intel 8051 parts for both internal and external modes. Interestingly, these chips are also standard production chips. This increases emulation accuracy since the emulation chips are the same used in regular production.

All Infineon C500 production chips have Enhanced Hooks support built in. Figure 5 is the Nohau Enhanced Hooks pod with the C517A daughterboard installed. Shown are other daughterboards. These pods connect to the standard Nohau EMUL51 emulator. Production chip emulators can emulate these 8051 parts only in external mode so are not popular.



Standard Production Chips

The emulator needs information from, and also needs to control the target controller. This is done via the address and data busses and various control signals such as interrupt and read and write pins. Since internal information is not normally presented on these busses during run time, it is not available in real-time. The emulation must be halted, the internal registers must be read and stored and emulation continued. This results in unacceptable intrusion times. These types of emulators make excellent tools if the access restrictions are acceptable.

Advantages are the same emulation chip as production, easy changing of same family processors and low cost emulators.

Trace_1

Frame	Address	Destination	Source	Value	Opcode	Instr		Symbol
-31	0:	200			FA000002	JMPS	_CSTART_TASK	==> :
-28	200:	210	F600	F600	E60800F6	MOV	FE10h,#F600h	==> _CSTART_TASK:
-27	204:				CC00	NOP		
-25	206:	A00	4500	4500	E6F00045	MOV	R0,#4500h	
-23	20A:				CA006805	CALLA	cc_UC,main	
-20	568:				A55AA5A5	DISWDT		==> main:
-18	56C:	30C	4AE	4AE	E606AE04	MOV	FF0Ch,#4AEh	
-16	570:	218	50	50	E60C5000	MOV	FE18h,#50h	
-14	574:	314	4AE	4AE	E68AAE04	MOV	FF14h,#4AEh	
-12	578:				B54AB5B5	EINIT		
-10	57C:	A08	5100	5100	E6F40051	MOV	R4,#5100h	
-8	580:	212	A08	5100	F6F41202	MOV	timer,R4	

Frames 32(-32:-1);Trig.Count 0

Figure 4

BDM: Background Debug Mode

This is a Motorola scheme using a dedicated serial port to gain access to a special debug module inside the microcontroller. This module operates parallel to the microcontroller and generally uses no resources. It has access to internal registers and memory and can control the CPU. Some models have two hardware breakpoints for ROM operation. Figure 6 is the Nohau HC12 BDM emulator.

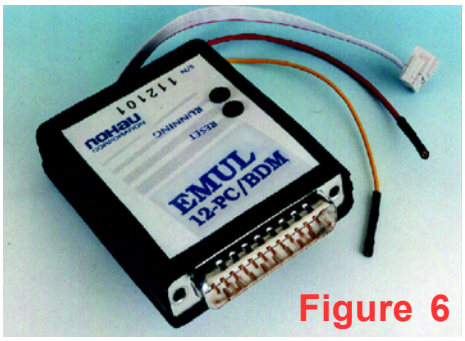


Figure 6

A BDM emulator does not have a trace memory, triggers or emulation memory. A BDM emulator is sometimes used in conjunction with a full emulator. It can program FLASH and EEPROM into the HC12 controller in your target system.

You can load a program and single-step or run the target processor. Source code will be visible for HLL debugging. You can set software breakpoints (hardware in HC12) and view memory in real-time.

Variable, arrays and structures are viewed in a variety of formats. The emulator runs at full clock speed. A BDM emulator is more robust than a monitor, but with less features than a full emulator.

JTAG and OCDS Debugging

The higher speeds and complexity of some microprocessors has encouraged chip makers to incorporate debugging facilities on-chip. The BDM is one example. The JTAG interface was designed to facilitate testing circuit board connections between large chips. This is a serial connection and signals can be sent to and read from I/O ports. This is also called the JTAG Boundary Scan because the serial data scans the outside I/O pins of the chip. A pattern sent from one chip to the next can be then compared to check if the circuit board connections are intact. Since the JTAG port is unused during normal chip operation, and since it runs in parallel to the controller's CPU, it can access an on-chip debugging module in real-time similar to the BDM interface. This debug module is called OCDS (On-Chip Debugging Support) in the newer Infineon microcontrollers such as the C161U and the UTAH. This module has direct access to the CPU core. Manufacturers such as Motorola also use this approach in the PowerPC and some 68K. The ARM7 also uses a JTAG accessed debugging module. Nohau supports the Infineon OCDS and ST Microelectronics OCE protocols. It is possible to build a full emulator using a combination of the OCDS and standard trace and trigger hardware giving high performance.

No target or CPU resources used

Monitor kernels typically need about 10K ROM and 10-20 bytes RAM and a free communication port. A good emulator uses none of these. The emulator should be invisible to the target. Better emulators are and in addition do not steal CPU cycle time for ordinary housekeeping duties.

Getting the Hardware Working

Simulators are great, but they can not take all the variables into account. A simulator designer has to think of everything: the big problems are usually those items that come up after the hardware is constructed. Items like capacitance, timing, inductance, and chip versions. These become more important as CPU speeds increase. It is a very difficult task to replicate the pipeline found in many microcontrollers today and is best done with real hardware.

Target monitors are considerably better in that they run on real hardware. But the target system must be a complete working system in order to get the monitor kernel to run. Not so with an emulator. An emulator will run with no hardware at all or incomplete sections. A target monitor can be installed in the final target ready to be activated at any time for debugging. This is useful for test and repair purposes. Figure 7 is the Nohau EMUL-ST10-PC emulator connected to the Phytec C167CR evaluation board. Evaluation boards are an economical and practical method of shortening your product development time. It is also a useful reference design.

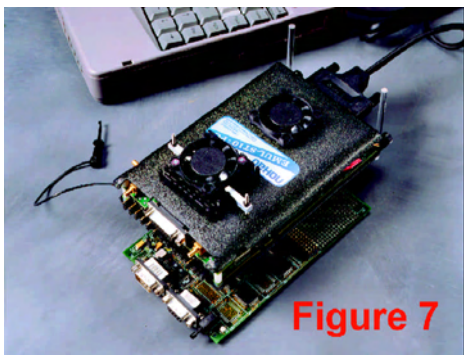
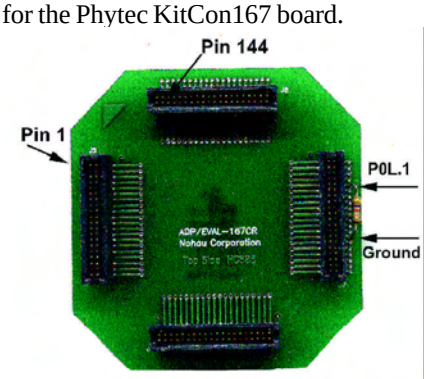


Figure 7

Connecting to Your Target System

This is easy. Most issues will be handled by the board designer in conjunction with your emulator representative. Connection to the target is a two step process.

First, the adaptation method must be chosen. Solder-down and socket methods are preferred. Clip-over adapters are handy but expensive. Nohau provides the Delta probe. If you need to access the target in a hard-to-access area, consider Nohau's Flex Cable. Figure 8 is the adapter for the Phytec KitCon167 board.



This view will be visible when the adapter is connected to the target.
The emulator plugs into this side.
Figure 8

Second, the software and jumper settings on the emulator must be correctly set to match the target board and the software initialization routines. This is easy to do and here is where good technical support counts. Usually the default settings work.

To connect the Nohau EMUL166 to the Phytec C167CR board, only four sockets need to be soldered on it and one jumper (AutoMap) needs to be removed on the emulator. In the EMUL-ST10, two chip selects need to be activated with simple mouse clicks. Load the code, and the board immediately goes into operation! The board behaves as before, but now you have access to the internal workings of the system. You also have powerful trace and trigger features as already described. The same scenario works for other boards as well as your custom prototype.

Hardware Breakpoints

A software breakpoint is created by inserting a 2 byte TRAP instruction which will divert normal program flow to the debugger. The program may crash if the program counter lands on the second byte. Nohau hardware breakpoints use comparators to detect accesses to a location and no code memory contents are modified. Breaks on regions need hardware breakpoints. Software breaks are still useful and Nohau provides both types.

Software breakpoints are useless with ROM memory since a TRAP can not be inserted. Only hardware breakpoints function in ROM systems.

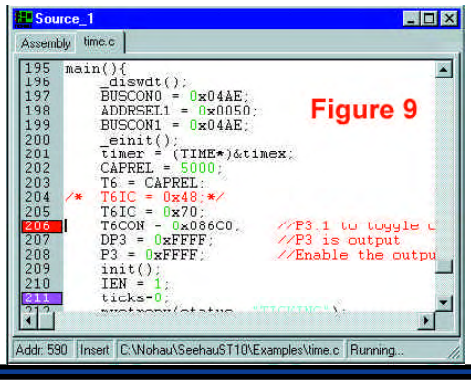
Trace Memory

The Trace records each processor cycle along with a timestamp and optionally external signal levels. The trace can record all code fetches and will distinguish between instructions that are cancelled in the pipeline and those successfully executed. False triggering is therefore avoided on unexecuted instructions. Simulators and monitors do not have trace memory.

Trace Memory: An example

The trace window shown in Figure 4 is from the EMUL-ST10-PC and results from the code shown in Figure 9 which is the source code window. Other emulators are similar. This recording was unfiltered and represents all executed instructions. Instructions entering the pre-fetch queue, then cancelled, are not erroneously classified as been executed. They can be usually shown in the trace if desired.

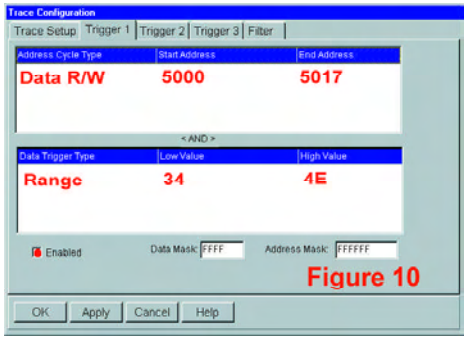
Note the fields available. The trace can be filtered with the triggers so that only specified cycles are recorded. This saves time searching for bugs. You can trigger on specific addresses, data values, and qualifying them as reads or writes and many other similar qualifiers. The trace memory is a powerful tool, displaying events as they really happened.



Conditional Triggers

These are extremely powerful and easy to use. They allow you to specify an action when some event happens. The Trigger Configuration window shown in Figure 10 is a basic entry. If a data R/W cycle with a value from 34 to 4E and in the address range from 5000 to 5017 occurs, a trigger event will be created.

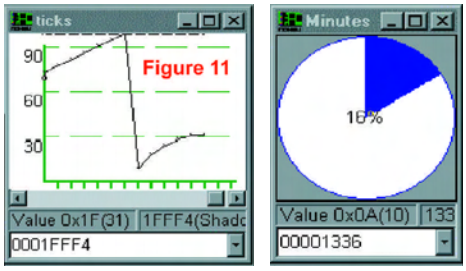
The trigger can include address, data, clock cycles and external signals. These can trigger a break, start/stop the trace capture, record a timestamp or many other things determined by the emulator's capabilities. This powerful tool is found only in emulators.



Actual Memory and I/O Ports

Ports and memory can be viewed from real hardware parts and not simply a software simulation. It is possible to wire your favorite peripheral chip to the bottom of the emulator pod and access it. Accurate software simulation depends on all the nuances of complex peripherals entered correctly and is hard to achieve.

Often, it seems that problems only develop when the program is run on the actual hardware. How often does it seem that things depend so much on the rise or fall time of some input signal? Or the routing of a certain wire? An emulator will help get your development finished faster by getting you to this point directly.



Data can be viewed in various forms such as the graphical types shown in Figure 11, or in several handy numerical formats.

Since the emulator has its own internal RAM which can be substituted for ROM, you can debug and modify the program code and data easily in ROM systems.

In the same fashion, memory not yet installed on the target can be substituted by the emulator. The size and address of this RAM is selectable. The granularity is 2 bytes on the EMUL166-PC allowing mapping around any external peripheral.

Performance Analysis in Hardware

A debugger can only simulate Performance Analysis and it does a good job. The emulator goes one step further by doing the analysis on the real hardware increasing accuracy. Once again, using the actual hardware will show problems that might not be evident in software simulation. Spurious interrupts and other functions that unexpectedly consume CPU resources can cause serious performance problems and can be difficult to find. Performance Analysis can easily find such problems. Figure 12 shows a PPA display.

Conclusion

This article has provided information about In-Circuit Emulators and the benefits that accrue to you, the designer. You will be better able to select various components of the cycle depending on your needs. See www.icetech.com for more information on emulators and how they can help you debug your projects faster.