The Software Engineer's Guide to In-Circuit Emulation
How to increase your debugging skills!

Nohau Corporation
51 East Campbell Avenue
Campbell, California 95008

© 2002 Nohau Corp
Version 1.2

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.

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.

Why do we need emulators ?

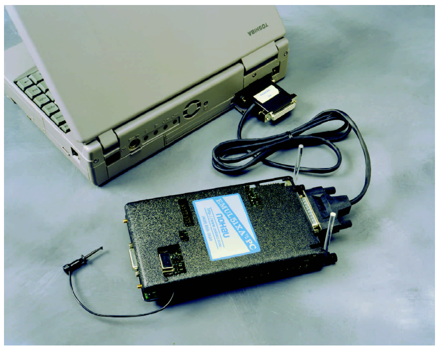
Software debuggers and monitors offer economical debugging capabilities sufficient for many designs. There are some cases where an emulator is needed to resolve difficult to find bugs. In all cases an emulator will pay for itself by providing you with decreased debugging time, ease of system integration, 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.

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 I have thought of. Emulators 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 computer and control the running of the CPU. Emulators are a cost effective method of debugging embedded software. Shown below is the new Nohau EMUL51XA-PC emulator.



Internal and External Modes

Internal or single-chip mode is when program and data memory is located in the controller chip in the form of FLASH or EPROM. The address and data busses are not available 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 or Hooks chip for effective emulation.

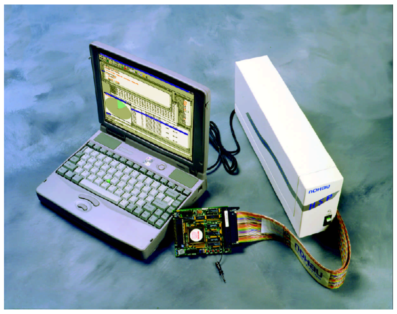
External mode is when the program memory and perhaps some data memory is located externally to the controller. The address and data busses are available to access this memory. Production, bondout or Hooks chips are effective for this mode. The address and data busses may not be used as general I/O ports. Nohau emulators use all three types of controllers for effective debugging.

Bondout, Hooks and Standard Production Chips

These terms refer to the emulation processor used by the emulator to replace your target controller. Bondout and Hooks chips allow single chip emulation. A bondout chip has extra pins connected to internal nodes. The Philips XA uses a bondout chip for emulation. Philips 8051 products use Hooks emulation. The Nohau EMUL51XA-PC and the EMUL51 provide support for both.

Hooks chips take advantage of unused cycle times on various pins to provide the address and data busses. Hooks chips are used with the Philips 8051 family for both internal and external modes. Interestingly, these chips are also standard production chips.

Many Philips 8051 production chips have Hooks support built in. This provides very accurate emulation since the production and emulation chips are exactly the same. Shown below is one model of Nohau Hooks pod and the HSP box.



Getting the Hardware Working

Simulators are great, but they can not take all the variables into account. The simulator designer has to think of everything: usually those items that come up only after the hardware is constructed. Items such as capacitance, timing, inductance, and chip versions. These are more important as CPU speeds increase.

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.

Emulators have all the features of software debuggers and monitors plus these benefits:

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.

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 external ROM memory since a TRAP can not be inserted. The ROM can be mapped into emulation memory to solve this issue or use hardware breakpoints.

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. The trace can be enabled by the triggers. This results in filtering where only those cycles of interest are recorded and others are discarded. Simulators and monitors do not posses trace memory. The screen below shows some features of Nohau emulators.

Conditional Triggers

These are extremely powerful and easy to use. They allow you to specify an action when some event happens. The trigger can include an 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

These 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 simulation depends on all the nuances of complex peripherals entered correctly.

Often, it seems that some problems only develop when the actual hardware is used. An emulator will help get your development finished faster by getting you to this point directly.

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.

Performance Analysis in Hardware

A debugger can only simulate this 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 may 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.

Connecting to the 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.

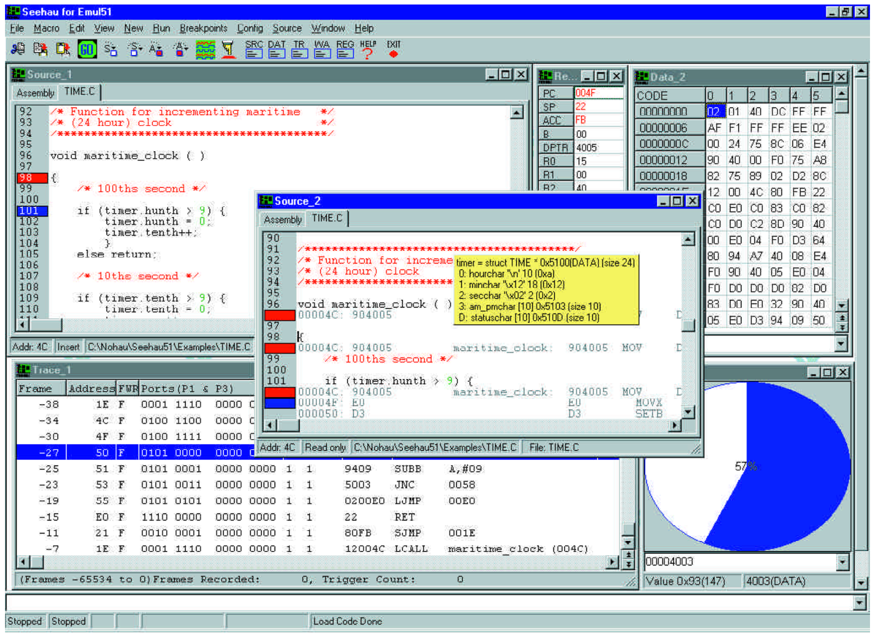
First, the adaptation method must be chosen. Solder-down and socket methods are preferred. Clip-over adapters are both expensive and unreliable. The emulator manufacturer should be involved at the start of the board design.

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 EMUL51XA to the Philips XTEND-G3 evaluation board: the emulator is plugged into the G3 socket and memory mapped in the Seehau software with simple mouse clicks. Load the code and it immediately goes into operation!

Conclusion

This article has provided information about In-Circuit Emulators and the benefits that accrue to you, the designer.


Seehau for Emul51
File Macro Edit View New Run Breakpoints Config Source Window Help
GO SRC DAT TR WA REG HELP EXIT
Source_1
Assembly TIME.C
Source_2
Re...
Data_2
Trace_1
57%

```
/* Function for incrementing maritime    */
/* (24 hour) clock                       */
/****************************************/

void maritime_clock ( )
{
    /* 100ths second */

    if (timer.hunth > 9) {
        timer.hunth = 0;
        timer.tenth++;
        }
    else return;

    /* 10ths second */

    if (timer.tenth > 9) {
        timer.tenth = 0;
```

Addr: 4C | Insert | C:\Nohau\Seehau51\Examples\TIME.C

```
/*******************************************/
/* Function for increme
/* (24 hour) clock
/************************
void maritime_clock ( )
00004C: 904005
{
00004C: 904005          maritime_clock:   904005  MOV
    /* 100ths second */

    if (timer.hunth > 9) {
00004C: 904005          maritime_clock:   904005  MOV
00004F: E0                                E0      MOVX
000050: D3                                D3      SETB
```

```
timer = struct TIME * 0x5100(DATA) (size 24)
 0: hourchar '\n' 10 (0xa)
 1: minchar '\x12' 18 (0x12)
 2: secchar '\x02' 2 (0x2)
 3: am_pmchar [10] 0x5103 (size 10)
 D: statuschar [10] 0x510D (size 10)
```

Addr: 4C | Read only | C:\Nohau\Seehau51\Examples\TIME.C | File: TIME.C

PC	004F
SP	22
ACC	FB
B	00
DPTR	4005
R0	15
R1	00
R2	40

CODE	0	1	2	3	4	5
00000000	02	01	40	DC	FF	FF
00000006	AF	F1	FF	FF	EE	02
0000000C	00	24	75	8C	06	E4
00000012	90	40	00	F0	75	A8
00000018	82	75	89	02	D2	8C
	12	00	4C	80	FB	22
	C0	E0	C0	83	C0	82
	C0	D0	C2	8D	90	40
	00	E0	04	F0	D3	64
	80	94	A7	40	08	E4
	F0	90	40	05	E0	04
	F0	D0	D0	D0	82	D0
	83	D0	E0	32	90	40
	05	E0	D3	94	09	50

```
Frame  Address FWR Ports(P1 & P3)
  -38      1E  F   0001 1110  0000 0
  -34      4C  F   0100 1100  0000 0
  -30      4F  F   0100 1111  0000 0
  -27      50  F   0101 0000  0000 0
  -25      51  F   0101 0001  0000 0000  1  1   9409    SUBB   A,#09
  -23      53  F   0101 0011  0000 0000  1  1   5003    JNC    0058
  -19      55  F   0101 0101  0000 0000  1  1   0200E0  LJMP   00E0
  -15      E0  F   1110 0000  0000 0000  1  1   22      RET
  -11      21  F   0010 0001  0000 0000  1  1   80FB    SJMP   001E
   -7      1E  F   0001 1110  0000 0000  1  1   12004C  LCALL  maritime_clock (004C)
(Frames -65534 to 0)Frames Recorded:     0, Trigger Count:     0
```

00004003
Value 0x93(147) | 4003(DATA)

Stopped | Stopped | Load Code Done