MSC1210 debugging strategies for high-precision smart sensors

By Hugo Cheung (Email: cheung_hugo@ti.com)
Design Engineering Manager, High-Performance Analog, Data Acquisition Products

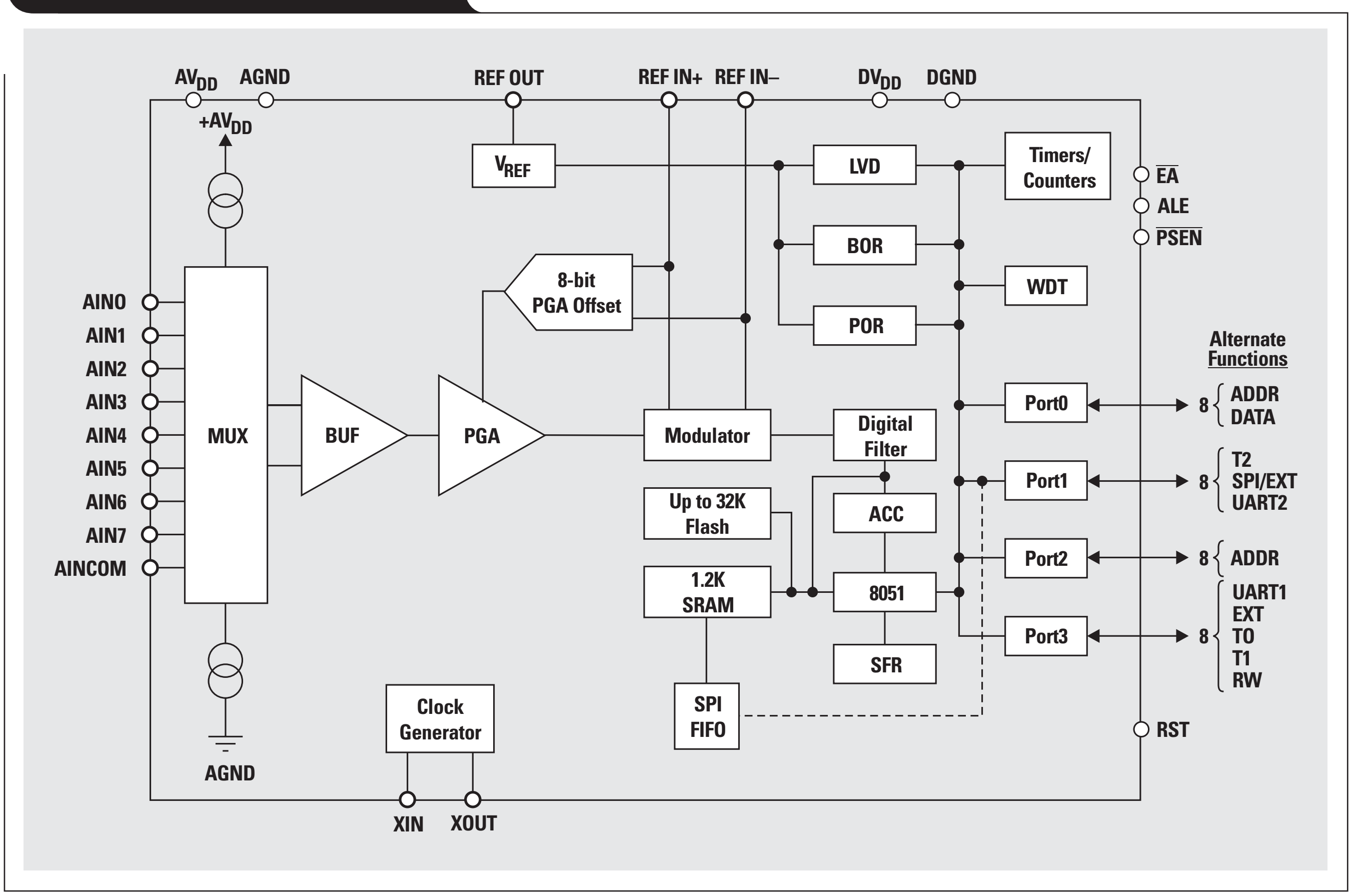
Introduction

The MSC1210 embeds an 8051 CPU, a 24-bit delta-sigma ADC, and high-performance peripherals to give a system on-chip solution for high-precision data acquisition systems (Figure 1). The MSC1210 therefore provides an excellent solution for implementing high-precision “smart sensors.” For high-precision requirements on industrial smart sensor applications working at a signal range lower than 100 nV, efficient debugging of code without sacrificing analog performance raises critical issues. This article discusses the issues involved in smart sensor development, suggests debugging strategies including integrated development environment (IDE) simulators, and compares simulators with in-system debuggers (ISDs).

Smart sensors

Process control instrumentation relies upon high-precision analog sensor signals for the monitoring of control devices. The sensor signals are translated to the 4- to 20-mA analog signal standard, which has long been a standard for industrial process control. With today’s advanced technology, computers are used to monitor and control a system of instruments that connect clusters of sensors from a central point. The sensors are integrated with high-precision analog-to-digital converters and high-performance processors to produce smart sensors. Smart sensors replace 4- to 20-mA wiring with a digital network that is more accurate and more reliable, with simpler interconnections. The smart sensors also integrate distributed control functions that improve overall system performance and lower equipment cost.

Figure 1. MSC1210 block diagram

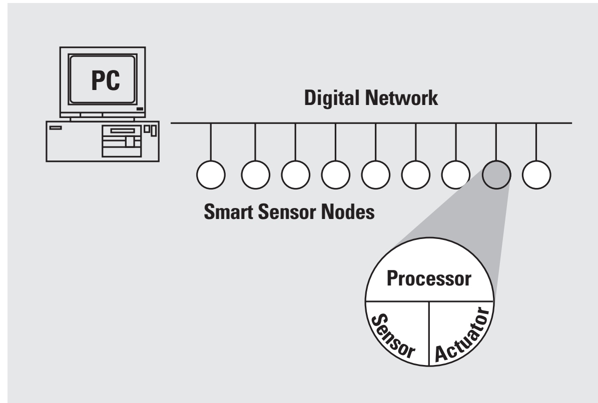


MSC1210 for smart sensors

The MSC1210 (Figure 2) contains many features that are required by smart sensors, including:

- High-precision ADC: Over 22 effective number of bits
- Embedded sensor signal conditioning circuit: Input buffer, PGA, offset DAC, gain and offset calibration functions
- Low power consumption to reduce power network requirements: Under 4 mW
- Enhanced CPU: 4 machine cycles per instruction 8051 core
- Embedded memory: Program (32KB) and data (1.2KB)
- High-performance communication channels: SPI with deep FIFO, dual UARTs
- Robust industrial environment circuits: Low-voltage detect, brownout detect, watchdog timer, wide operating conditions (power supply 2.7 to 5.25 V and operating temperature –40 to +85°C)

Figure 2. Smart sensor system



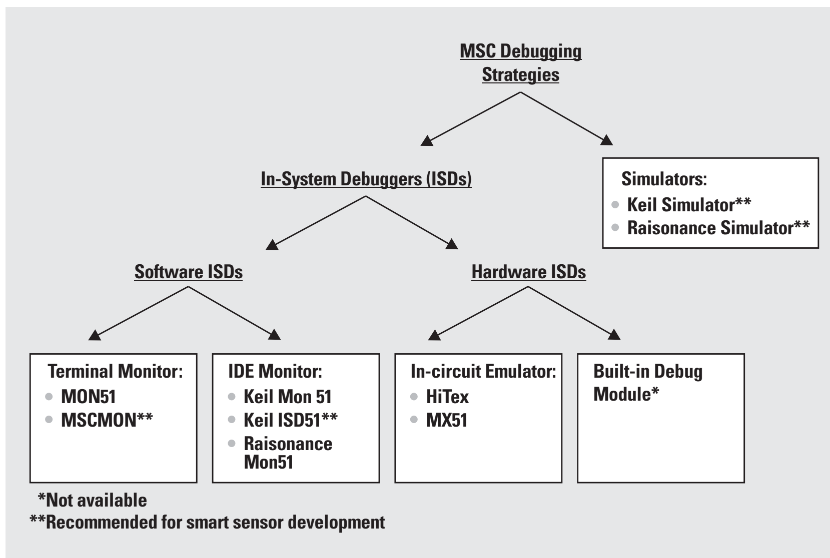
Smart sensors code development

Since a smart sensor is an integrated system with complicated sensor signal conversion, process control, and networking, the code development for smart sensors has to resolve problems such as the following:

- Development system effects on the analog signal precision
- Physical size of the target hardware
- Development host-to-target-system communication media
- Real-time control and networking timing
- Development system power source

The microsystem controller (MSC) provides various debugging strategies that meet the tough development requirements. Figure 3 depicts the available debugging strategies for MSC devices. The strategies range from simulation-based to in-system debugging. In-system debuggers (ISDs) are further divided into software-based and hardware-based. Among the available strategies, the Keil and Raisonance simulators, the MSCMon terminal monitor, and the Keil ISD51 IDE monitor are suitable for smart sensor code development.

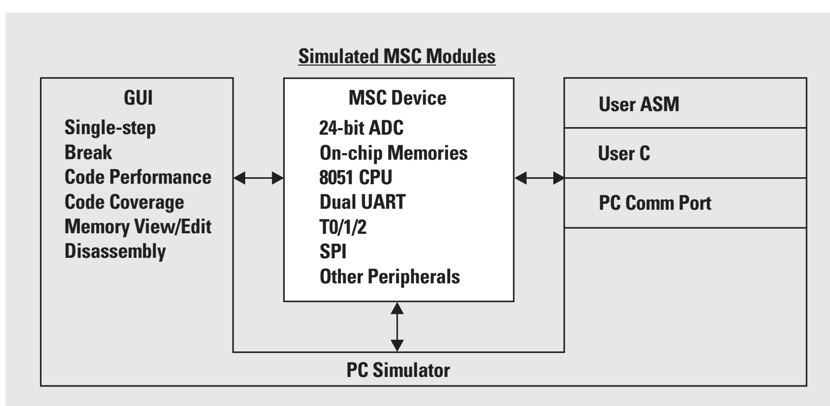
Figure 3. MSC debugging strategies tree



IDE simulator for initial smart sensor coding

The integrated development environment (IDE) is a set of development tools integrated in a user-friendly GUI suite. Development tools integrated into the same environment shorten the code development cycle and reduce code errors, which also enhances code quality. IDEs provide tools such as editors, assemblers, compilers, linkers, project management, and revision control, as well

Figure 4. IDE simulator conceptual block diagram



as device simulators in the same environment. Commonly available simulators of IDEs simulate 8051 devices within Microsoft® Windows®. UNIX platform simulators are not as common as the Windows version.

Simulators enable users to simulate code execution without actual target hardware. Users can verify algorithms and timing, and simulate peripherals, interrupts, and I/O. This is important because it allows the user to start code development and evaluate system performance even before the target hardware is available. Figure 4 depicts the conceptual block diagram of the IDE simulator for the MSC device. Users can perform disassembly, break point, memory watch/modify, code execution trace, and peripheral monitoring. Simulators also support code coverage tools that “mark” the code that has been executed. Simulators also provide performance analyzer tools that record the execution time for functions so that the user can profile code performance. However, the most common simulator operation is code stepping. Simulators support single-step with “step-into” a target function or “step-over” the function. The machine cycle count is accurate in the simulators; thus execution time can be easily evaluated for inefficiencies.

Common PC Windows IDE simulators include the Keil debugger (Figure 5) and the Raisonance debugger (Figure 6). The Keil IDE user’s manual is a good reference for the detailed simulator operations. The Raisonance IDE has debugging features similar to the Keil IDE. See “Related Web sites” at the end of this article.

Figure 5. Keil IDE simulator

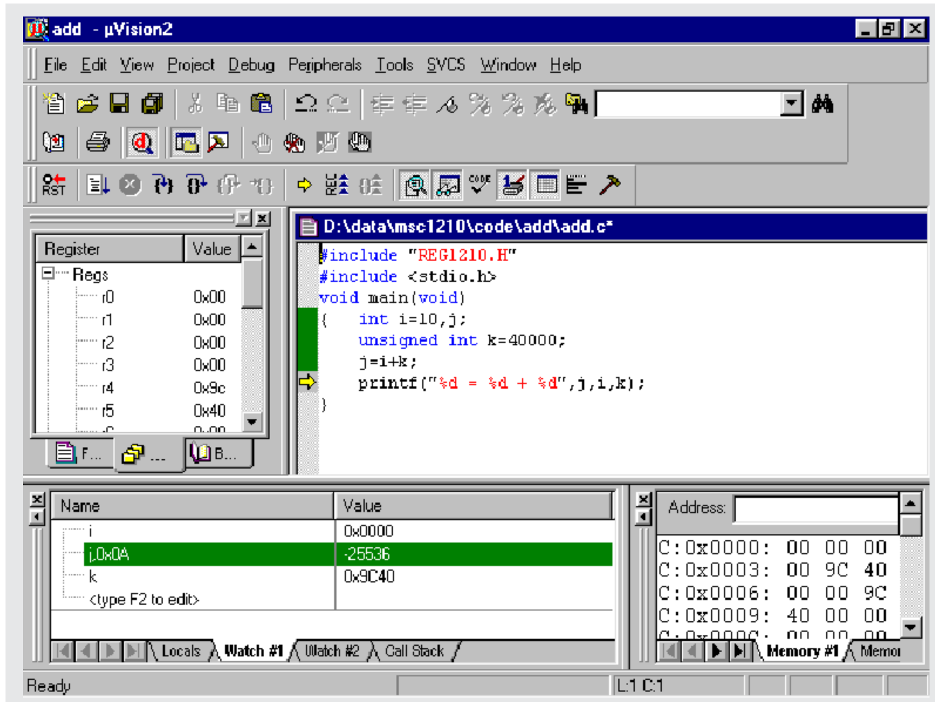
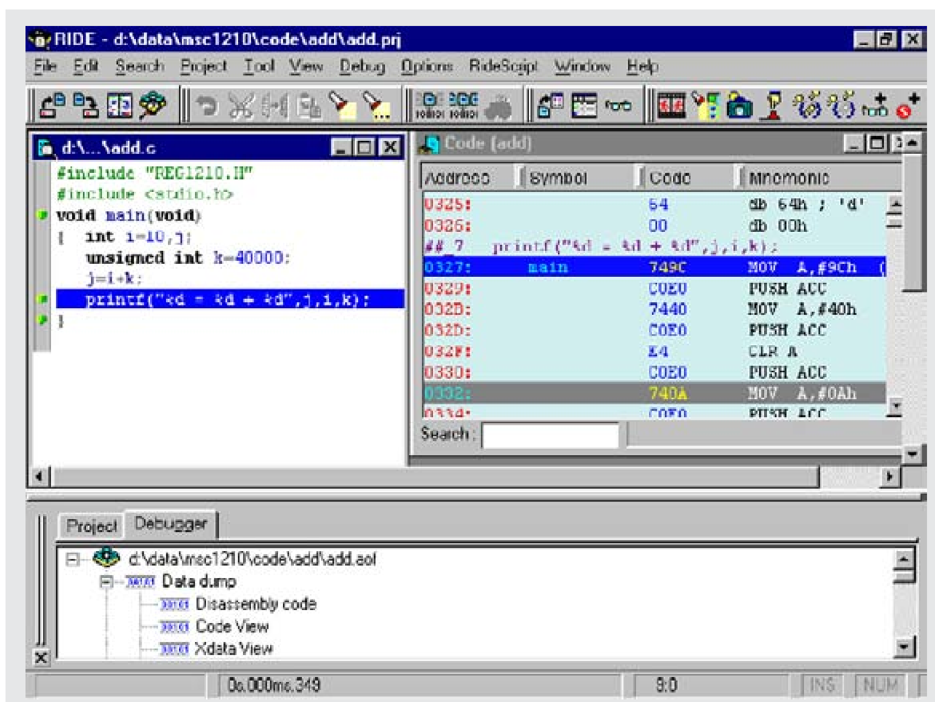


Figure 6. Raisonance IDE simulator



Advantages of using IDE simulators for smart sensor code developments

- Code simulation is a low-cost approach to code development, since no development hardware tools are required.
- Code development can be started before target system hardware is available.
- IDE simulators are best suited for initial smart sensor code development.

Disadvantages of using IDE simulators for smart sensor code development

- A precision analog signal cannot be simulated.
- Network timing and real-time interactions in process control are difficult to simulate.
- When code development reaches the stage that requires target hardware, debugging in the target system or ISD is needed.

Ad hoc debugging style is insufficient for smart sensor development

Instead of PC simulation, the ISD executes and debugs code in the actual target system. Ad hoc debugging—simply inserting the debug code wherever it is needed—is the easiest method. For example, simply add a `printf` statement and inspect the result. This style is good for simple code testing. However, when the code size increases, the number of `printf` statements will become unmanageable.

Smart sensor in-system debuggers

The ISD development environment embeds debugging support within the smart sensor (Figure 7). The developing code will process real system input from the sensors and provide instantaneous system response instead of system simulation. Therefore, system-level issues such as sensor accuracy, control system stability, or sensor network throughput can be resolved. As shown in Figure 3, there are two categories of ISDs—software-based and hardware-based. Software-based ISDs are further divided into terminal-based and IDE-based. Terminal-based debugging includes a general-purpose monitor and an on-chip debugger. IDE-based debugging includes a source-level monitor and Flash-enabled ISD. Hardware-based ISDs are further divided into an in-circuit emulator (ICE) and a built-in debugger module (BDM).

Since smart sensors are compact systems, most of them cannot accommodate external memory devices that occupy extra board space, increase power consumption, and increase system cost. Therefore, the resources to provide an ISD setup must be included within the MSC1210. Those resources include ISD code space, a CPU time-to-processing ISD routine, and an ISD port.

Software ISDs that require external memory (Figure 3), such as the general-purpose MON51, Keil MON51, and Raisonance MON51, are not the best candidates for smart sensor development. Neither are hardware ISDs that require an extra ICE connection bus, consume more ICE bus power, and have higher system noise created by the ICE bus. ICEs are not recommended for in-system debugging. The ICE and ISD with external memory should be used only as an intermediate development setup.

Figure 7. Smart sensor ISD configuration

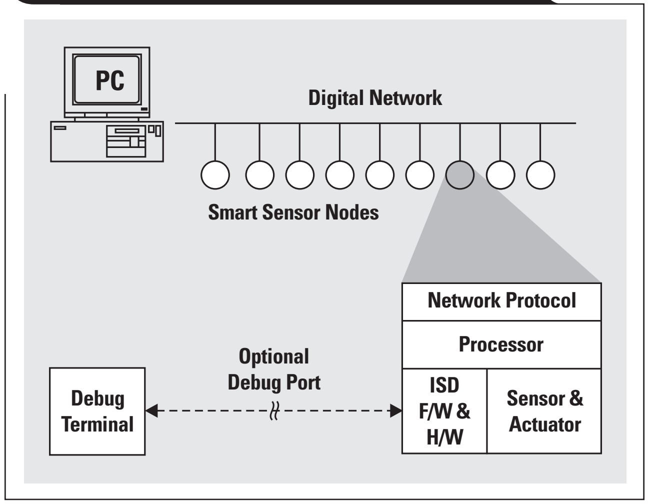
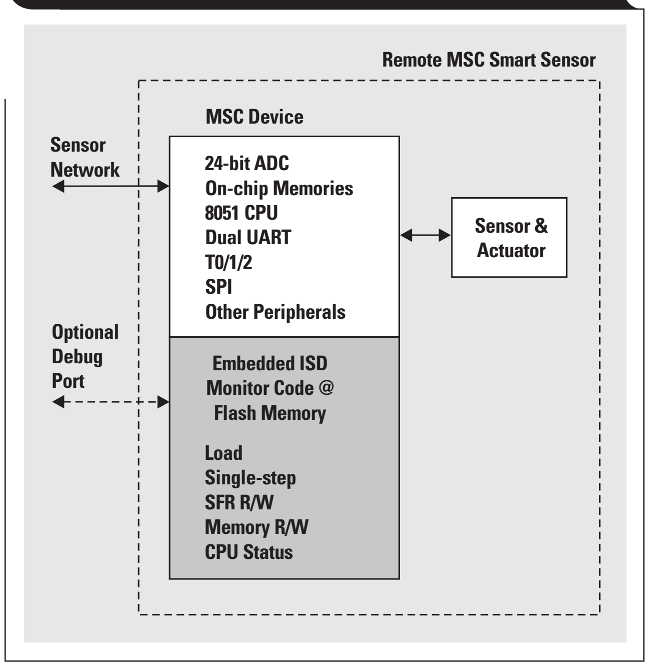


Figure 8. ISD monitor conceptual block diagram



Software ISD debugging style is ideal for smart sensor development

The embedded monitor for remote target systems such as smart sensors is an ISD software monitor that resides in code memory. The ISD monitor program (Figure 8) acts as an interpreter between the user code and the debug terminal program. The monitor allows code to be downloaded into the target system memory from the debug terminal (such as PC terminal programs using the on-chip UART) and then allows debugging functions—such as memory or SFR read/modify, CPU status request, user routine calls, single-step, or break—to be performed.

Sensor network as debug port

The debug port connection to the debug terminal creates a wiring problem for the compact remote smart sensor. An alternative is to communicate debug commands and responses and even to load user code between the sensor and terminal via the sensor network. However, there are so many sensor network standards being used that a custom ISD monitor code including sensor network processing is needed.

Downloading user code through sensor network or debug port

Programming microcontroller-embedded Flash memory with serial/parallel Flash programming operations is a function commonly available for many microcontrollers, including MSC1210. User code may be downloaded to remote sensors through debug port UART0 by putting the MSC1210 remote sensor in Flash programming mode. When a debug port is not feasible, or the number of sensor network connections must be minimized, user code may be downloaded during normal user operation via the sensor network. The MSC1210 has a program Flash memory self-update capability. In other words, when executing the embedded ISD monitor *Load User Code* command, the monitor program that resides in the MSC1210's embedded Flash memory will download the user code and store it in the same Flash memory.

MSC1210 IAP Flash memory

While the user code is being downloaded, the embedded Flash memory is busy performing write or erase operations, during which CPU execution from the Flash memory is not possible. The MSC1210 has an embedded 2KB boot-loader ROM that provides Flash write/erase routines. The ROM routines are used for in-application programming (IAP) of the embedded Flash memory. The MSC1210 user application code, such as the ISD monitor program, calls the ROM routines to program the Flash memory.

ISD monitor—embedded MSCMon

The ISD monitor may be modified with general-purpose monitor programs that are available free from the Internet. Many of them are tested for MSC1210, such as MonPlus by Steve Kemplin, PaulMon by Paul Stoffregen, and Ultramon by an unknown author.

MSCMon—an on-chip MSC monitor program

Besides the Flash IAP routines, the embedded ROM also includes other supporting routines (see Code Listing 1) for the monitor program or applications. Using the boot-loader debug subroutines, such as `autobaud`,

Code Listing 1: BootROM subroutine prototypes

```
void put_string(char code *string); // print a string to SBUF0
//erase a program or data memory page
char page_erase (int faddr, char fdata, char fdm);
//write a program or data memory byte, sel program/data with MXWS bit
char write_flash (int faddr, char fdata);
// write a program or data memory byte with 3 retries
char write_flash_chk (int faddr, char fdata, char fdm)
// Write A to @DPTR, select program/data with MXWS bit
// ASM write_flash_byte ;
char faddr_data_read(char); // read a HW config memory byte
char data_x_c_read(int addr, char fdm); // read a xdata or code byte
void tx_byte(char c); //transmit c to SBUF0
void tx_hex(char); // transmit hex of c to SBUF0
void putok(void); // transmit "ok" to SBUF0
char rx_byte(void); // receive a byte from SBUF0
char rx_byte_echo(void); // receive and echo a byte from SBUF0
char rx_hex_echo(void); // receive two hex digits from SBUF0
// receive and echo four hex digits from SBUF0, return R6:R7 as int type
int rx_hex_word_echo(void);
// receive four hex digits from SBUF0, return R6:R7 as int type
int rx_word_echo(void);
// receive and echo four hex digits from SBUF0, return R7:R6 as stack order
int rx_hex_word_echo(void);
void autobaud(void); // SBUF0 auto baud rate setup with T2
void putspace4(void); // print 4,3,2,1 ASCII Space, Carriage Return
void putspace3(void);
void putspace2(void)
void putspace1(void)
void putcr(void);
// ASM cmd_parser ; Command parser entry point
// ASM monitor_isr ; Monitor ISR entry point
```

`put_string`, or `cmd_parser`, gives the user the lowest-overhead debugging setup—MSCMon (see Code Listing 2).

The MSC-embedded debug subroutines support not only the basic debugging commands such as *S-SingleStep*, *B-Break*, and *Q-Continue*; they also support Flash memory commands such as *CP-CodePageErase*, *CW-CodeWrite*, *XP-XDataPageErase*, and *L-LoadFlash*. The MSCMon translates into very low debug overhead; the minimum-configuration MSCMon needs only 29 bytes of Flash code space. In addition, since user code is stored in Flash memory, the downloaded code can be used for the final application. The complete command listing is shown in Figure 6. Typically, the MSC monitor is resident in Flash memory, and the *Load User Code* command is used to download user code. Since the memory requirement for the MSC monitor is very low, the user code may be linked with the MSCMon.

Code Listing 2: Minimum-configuration MSC monitor—MSCMon

```
$include (reg1210.inc)
            CSEG     at 807FH
            ; HCR0: PML=0 RSL=1 to protect 0~1000H
            db        0BFH      Flash
            CSEG      at 0000H  ; Monitor code start
            LCALL     autobaud
            mov       R6, #high greet
            mov       R7, #low greet
            lcall     _put_string
            LJMP      cmd_parser
greet:      db        0ah,0dh,"MSC Monitor",0
            CSEG      at 033H
            ; any other AuxInt handler
            LJMP      monitor_isr
```

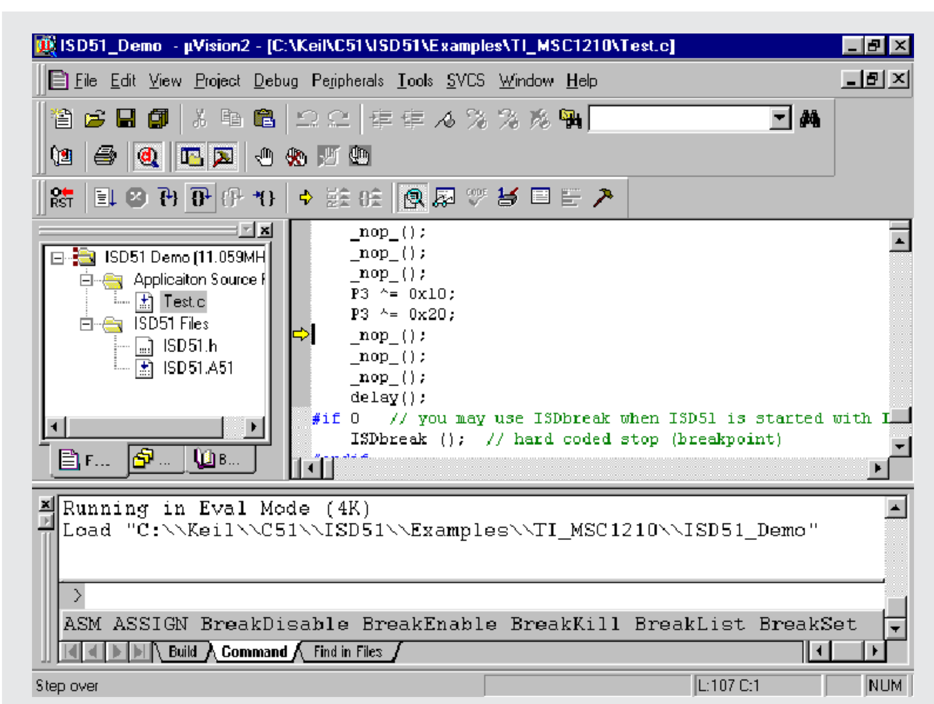
The MSCMon requires a PC debug terminal such as Hyper-term, Tera-term, Procomm, or Telix. A user sensor network program is needed when a debug port and debug terminal are not available.

Assembly-level and source-level debugging

Typical debug terminal programs using RS-232 ports have no information about the source code or monitor operation. Source-level debugging functions such as single-step require a symbol table and machine-code-to-source-code relationship. Therefore, only assembly-level debugging is supported. However, PC IDE programs have special handshaking with monitor programs for extra source code information that a general-purpose monitor or MSCMon does not have. These sophisticated source-level interfaces, when combined with the debugging commands, provide a user-friendly environment.

The Keil ISD51 monitor is a Flash memory resident program that communicates with the IDE to achieve source-level debugging. The IDE GUI greatly enhances the debugging efficiency (Figure 9). At any time, the user can monitor the CPU register and any memory contents. Since

Figure 9. Keil ISD51 source-level debugging



it is source-level, the debugging quality is much better than with assembly-level monitors. Because of the small size requirement for the ISD51, it is compiled and downloaded together with user code in the target system. The MSC1210 has a built-in hardware break point that detects the break address with hardware. The ISD51 fully utilizes the hardware break-point function, improving the execution performance over that of the software by a hundredfold.

Although the Keil ISD51 is a very attractive tool, an RS-232 debug port is needed. Again, the setup for the ISD monitor to communicate with the custom sensor network has to be independently developed.

Conclusion

Since the ISD monitor tools reside in the smart sensor, the target smart sensor does not compromise analog performance from the code-development setup. The MSCMon and Keil ISD51 monitors have low code-space overhead. If code space is sensitive in some applications, a larger-memory version of the MSC1210 family of devices can be used during prototyping, then switched to a lower-memory option for the production unit. The ISD monitors do not need extra hardware for debugging, which implies low development cost.

References

For more information related to this article, you can download an Acrobat Reader file at www-s.ti.com/sc/techlit/*litnumber* and replace *"litnumber"* with the **TI Lit. #** for the materials listed below.

Document Title **TI Lit. #**

1. "Precision Analog-to-Digital Converter (ADC) with 8051 Microcontroller and Flash Memory," MSC1210 Data Sheet sbas203
2. "ISD51 In-System Debugger," www.keil.com/c51/isd51.htm —
3. "Installing and Using the Keil Monitor-51," Application Note 152, www.keil.com/appnotes/files/apnt_152.pdf —
4. Russell Anderson, "Programming the MSC1210," Application Report sbaa076
5. "MSC1210 Precision ADC with 8051 Microcontroller and Flash Memory Evaluation Module," User's Guide sbau073
6. Russell Anderson, "Debugging Using the MSC1210 Boot ROM Routines," Application Report sbaa079

Related Web sites

analog.ti.com
www.ti.com/sc/device/MSC1210Y2
www.hitex.com
www.keil.com/demo
www.raisonance.com/download/index.php

IMPORTANT NOTICE

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products	
Amplifiers	**amplifier.ti.com**
Data Converters	**dataconverter.ti.com**
DSP	**dsp.ti.com**
Interface	**interface.ti.com**
Logic	**logic.ti.com**
Power Mgmt	**power.ti.com**
Microcontrollers	**microcontroller.ti.com**
Applications	
Audio	**www.ti.com/audio**
Automotive	**www.ti.com/automotive**
Broadband	**www.ti.com/broadband**
Digital control	**www.ti.com/digitalcontrol**
Military	**www.ti.com/military**
Optical Networking	**www.ti.com/opticalnetwork**
Security	**www.ti.com/security**
Telephony	**www.ti.com/telephony**
Video & Imaging	**www.ti.com/video**
Wireless	**www.ti.com/wireless**

TI Worldwide Technical Support

Internet

TI Semiconductor Product Information Center Home Page
support.ti.com

TI Semiconductor KnowledgeBase Home Page
support.ti.com/sc/knowledgebase

Product Information Centers

Americas			
Phone	+1(972) 644-5580	Fax	+1(972) 927-6377
Internet/Email	support.ti.com/sc/pic/americas.htm		
Europe, Middle East, and Africa			
Phone			
Belgium (English)	+32 (0) 27 45 54 32	Netherlands (English)	+31 (0) 546 87 95 45
Finland (English)	+358 (0) 9 25173948	Russia	+7 (0) 95 7850415
France	+33 (0) 1 30 70 11 64	Spain	+34 902 35 40 28
Germany	+49 (0) 8161 80 33 11	Sweden (English)	+46 (0) 8587 555 22
Israel (English)	1800 949 0107	United Kingdom	+44 (0) 1604 66 33 99
Italy	800 79 11 37		
Fax	+(49) (0) 8161 80 2045		
Internet	support.ti.com/sc/pic/euro.htm		
Japan			
Fax			
International	+81-3-3344-5317	Domestic	0120-81-0036
Internet/Email			
International	support.ti.com/sc/pic/japan.htm		
Domestic	www.tij.co.jp/pic		
Asia			
Phone			
International	+886-2-23786800		
Domestic	Toll-Free Number		Toll-Free Number
Australia	1-800-999-084	New Zealand	0800-446-934
China	800-820-8682	Philippines	1-800-765-7404
Hong Kong	800-96-5941	Singapore	800-886-1028
Indonesia	001-803-8861-1006	Taiwan	0800-006800
Korea	080-551-2804	Thailand	001-800-886-0010
Malaysia	1-800-80-3973		
Fax	886-2-2378-6808	Email	tiasia@ti.com
Internet	support.ti.com/sc/pic/asia.htm		ti-china@ti.com

C011905

Mailing Address: Texas Instruments
Post Office Box 655303
Dallas, Texas 75265