Analog and Mixed-Signal Products

Analog Applications Journal

Third Quarter, 2002

TEXAS INSTRUMENTS

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Contents

To view past issues of the
***Analog Applications Journal*, visit the Web site**
www.ti.com/sc/analogapps

Introduction

Analog Applications Journal is a collection of analog application articles designed to give readers a basic understanding of TI products and to provide simple but practical examples for typical applications. Written not only for design engineers but also for engineering managers, technicians, system designers and marketing and sales personnel, the book emphasizes general application concepts over lengthy mathematical analyses.

These applications are not intended as “how-to” instructions for specific circuits but as examples of how devices could be used to solve specific design requirements. Readers will find tutorial information as well as practical engineering solutions on components from the following categories:

- Data Acquisition
- Amplifiers: Op Amps

Where applicable, readers will also find software routines and program structures. Finally, *Analog Applications Journal* includes helpful hints and rules of thumb to guide readers in preparing for their design.

Adjusting the A/D voltage reference to provide gain

By Russell Anderson (Email: anderson_russ@ti.com)
Senior Applications Engineer

The purpose of providing gain in the analog-to-digital converter (ADC) is to boost the signal and thereby improve the signal-to-noise ratio (SNR). Even in ADCs that have internal programmable gain amplifiers (PGAs), some noise also gets amplified so that a doubling of the SNR is not achieved for each doubling of the programmable gain. This article shows how changing the reference voltage can achieve gain even in those ADCs that have no PGA.

The reference voltage defines the range of voltage inputs and therefore the size of the LSB. For an N-bit converter, the LSB is defined as

$$\frac{\text{Full-scale voltage}}{2^N}.$$

The full-scale voltage range may not always be V_{REF}; depending on the particular ADC, it could be $2V_{REF}$ or $\pm V_{REF}$. But regardless of how the output codes are mapped, they are directly influenced by the value of V_{REF}. A smaller voltage for V_{REF} will mean that all the output codes are constrained to a smaller voltage range and that the LSB will be a smaller voltage.

The observed noise in the output codes comes from two sources: (1) analog noise sources (random and synchronous) and (2) quantization noise.

When the bit size is large, as in an 8- or 12-bit ADC, the dominant noise source is expected to be quantization noise. But for high-resolution converters of 24 bits, the quantization noise could be the smallest noise source. Even when quantization noise is not the major source, reducing it to contribute even less to the overall result can still be useful.

If we look at the RMS noise as a function of the reference voltage for the ADS1253 (Figure 1), we notice that, as a percentage of full-scale, the smallest noise is achieved with the largest V_{REF}. This is because, as we increase V_{REF}, we are increasing the LSB; therefore, the dominant noise source starts to be influenced more by quantization noise and less by other noise sources. Or at least, the other noise sources do not increase as fast and therefore contribute less to the larger voltage range and larger LSB size.

If we convert the graph in Figure 1 to a graph of the amplitude of the noise in volts (Figure 2), we can see that the total noise is actually increasing as quantization noise becomes a more significant factor.

We can now see that the lowest noise occurs at a V_{REF} of 1 V. This can be a significant factor as we examine the voltage range of the input signal.

Figure 1. ADS1253 RMS noise vs. V_{REF} in ppm of full-scale

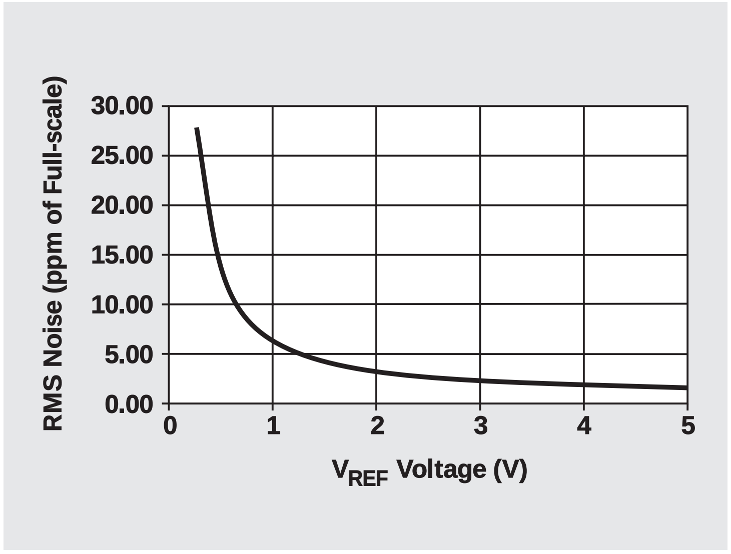
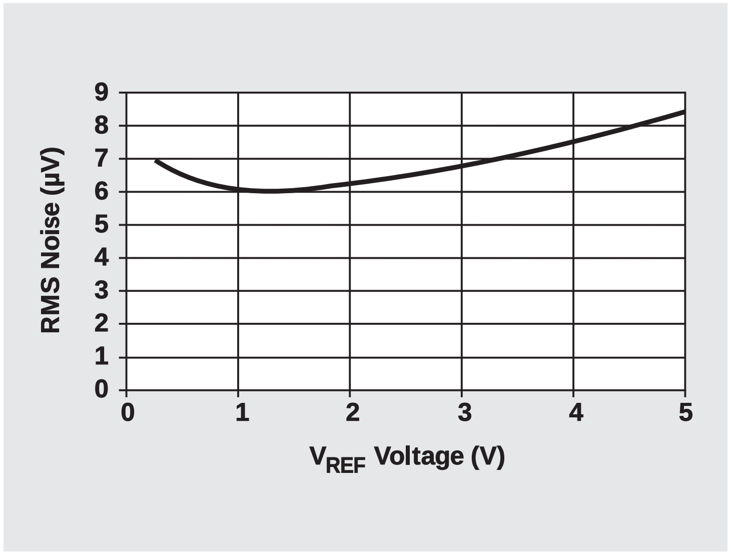


Figure 2. ADS1253 RMS noise vs. V_{REF} as µV



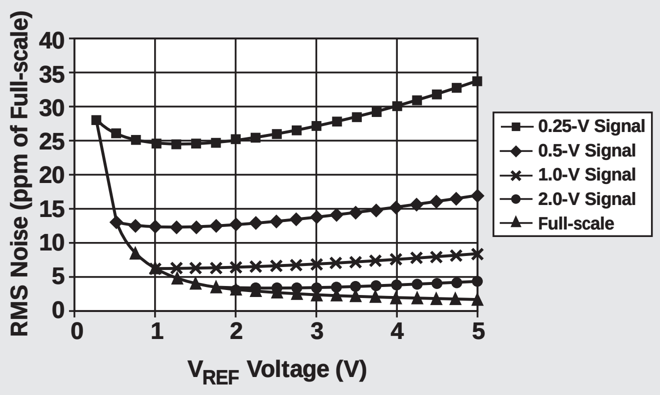
Let’s start by looking at the case of a V_{REF} of 5 V (Figure 3). If the input signal is smaller than the full-scale range of 10 V (–5 V to +5 V), then it would be more useful to plot the noise versus the signal full-scale value. For example, a noise of 5 ppm with a 5-V full-scale is 10 ppm if the full-scale range is only 2.5 V. It is the magnitude of the noise relative to the signal amplitude that holds our main interest.

The lowest line in Figure 3 is noise versus V_{REF}. The other lines are noise versus signal voltage.

As can be seen with these different full-scale signals, the best noise performance is achieved when the V_{REF} matches the full-scale range of the input signal, although there doesn’t appear to be any benefit to have a V_{REF} lower than 1 V.

With a lower reference voltage, more of the output codes are used with the smaller range of input voltages. This is the same as providing gain in the ADC. Normally the reference is 5 V; but with a V_{REF} of 2.5 V the effect is a gain of 2, and a V_{REF} of 1.25 V would be equivalent to a gain of 4. A gain of 5 seems to be the maximum useful gain, with a V_{REF} of 1 V.

Figure 3. RMS noise vs. signal and V_{REF} voltage



Related Web sites

analog.ti.com
www.ti.com/sc/device/ADS1253

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

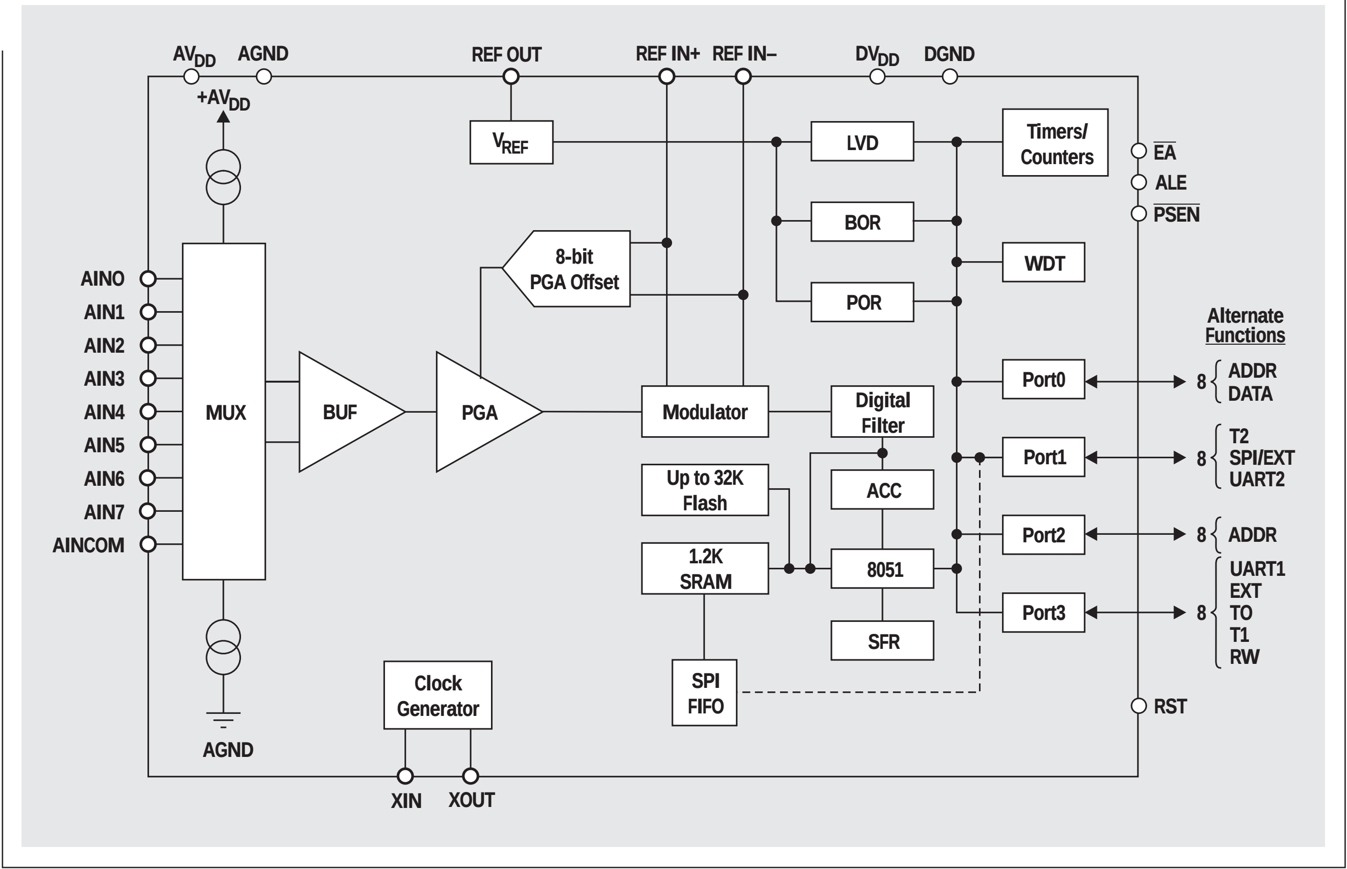
Introduction

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

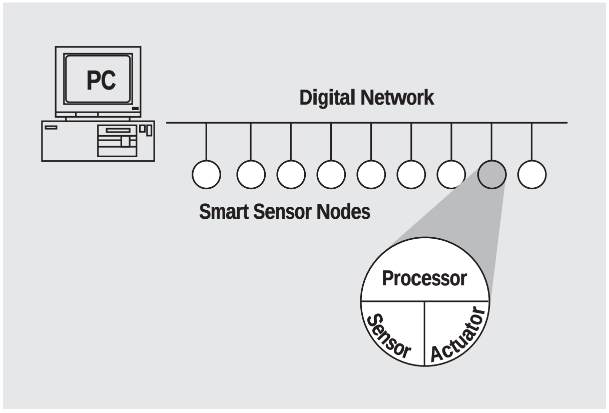


MSC1210 for smart sensors

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



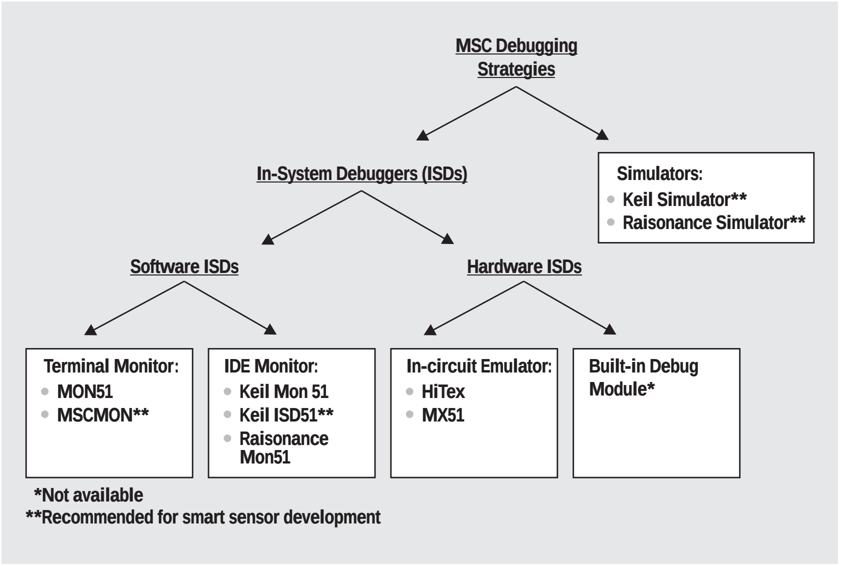
Smart sensors code development

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.

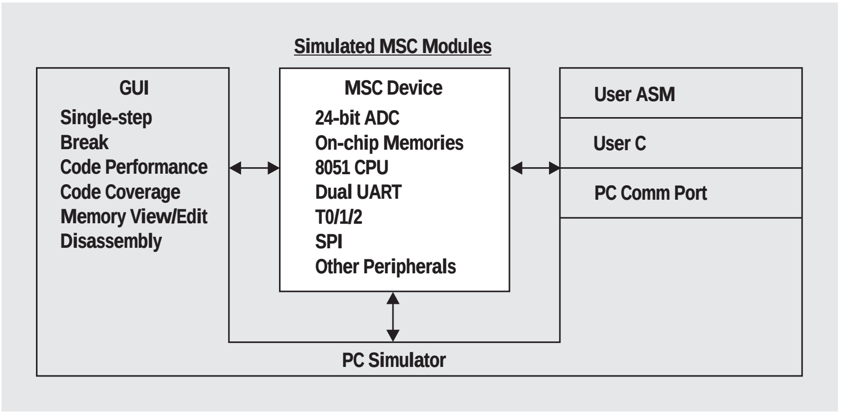
Figure 3. MSC debugging strategies tree



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

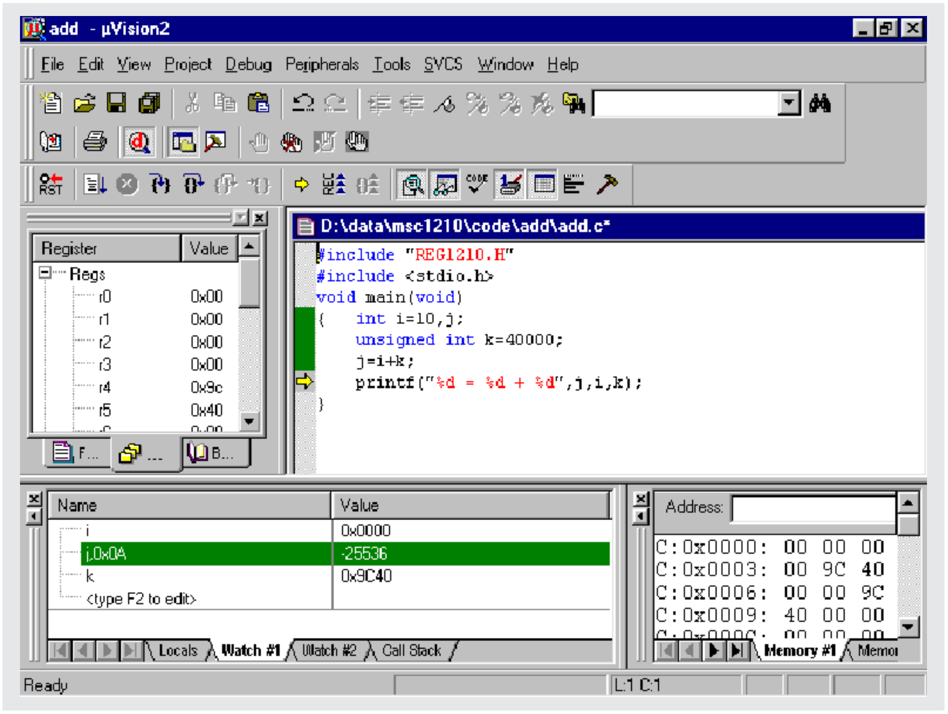


as device simulators in the same environment. Commonly available simulators of IDEs simulate 8051 devices within Microsoft® Windows®. UNIX platform simulators are not as common as the Windows version.

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


add - µVision2
File Edit View Project Debug Peripherals Tools SVCS Window Help
D:\data\msc1210\code\add\add.c*
Locals
Watch #1
Watch #2
Call Stack
Address:
Memory #1
Memor
Ready
L:1 C:1

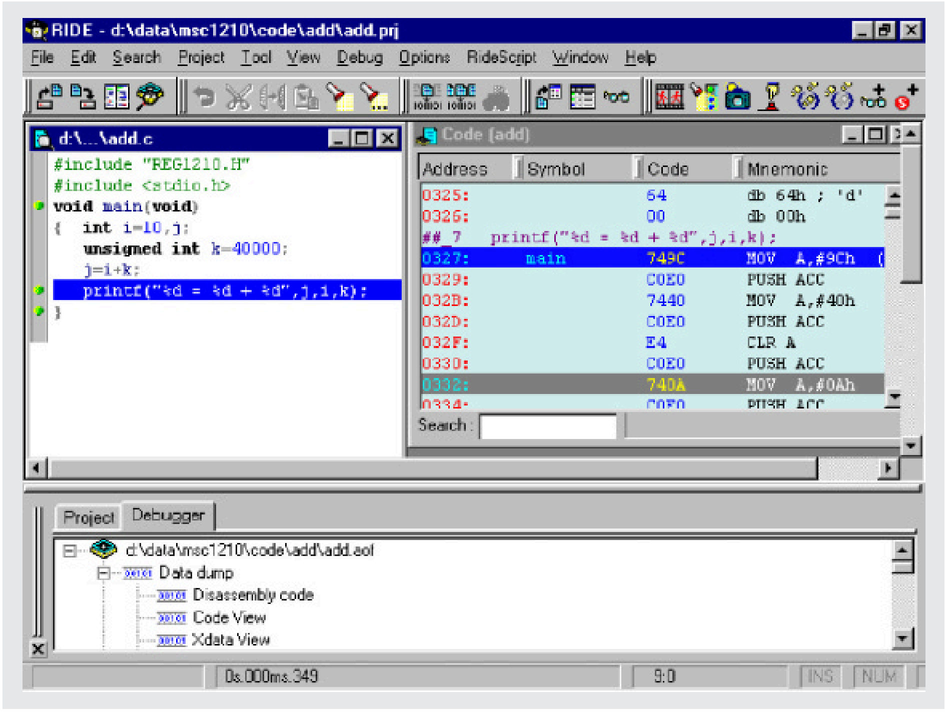
```
#include "REG1210.H"
#include <stdio.h>
void main(void)
{   int i=10,j;
    unsigned int k=40000;
    j=i+k;
    printf("%d = %d + %d",j,i,k);
}
```

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x9c
r5	0x40

Name	Value
i	0x0000
j,0x0A	-25536
k	0x9C40
<type F2 to edit>	

```
C:0x0000:  00 00 00
C:0x0003:  00 9C 40
C:0x0006:  00 00 9C
C:0x0009:  40 00 00
```

Figure 6. Raisonance IDE simulator


RIDE - d:\data\msc1210\code\add\add.prj
File Edit Search Project Tool View Debug Options RideScript Window Help
d:\...\add.c
Code (add)
Search :
Project
Debugger
d:\data\msc1210\code\add\add.aof
Data dump
Disassembly code
Code View
Xdata View
0s.000ms.349
9:0
INS
NUM

```
#include "REG1210.H"
#include <stdio.h>
void main(void)
{   int i=10,j;
    unsigned int k=40000;
    j=i+k;
    printf("%d = %d + %d",j,i,k);
}
```

```
Address    Symbol     Code    Mnemonic
0325:                 64      db 64h ; 'd'
0326:                 00      db 00h
##_7  printf("%d = %d + %d",j,i,k);
0327:      main       749C    MOV  A,#9Ch  (
0329:                 C0E0    PUSH ACC
032B:                 7440    MOV  A,#40h
032D:                 C0E0    PUSH ACC
032F:                 E4      CLR A
0330:                 C0E0    PUSH ACC
0332:                 740A    MOV  A,#0Ah
0334:                 C0E0    PUSH ACC
```

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

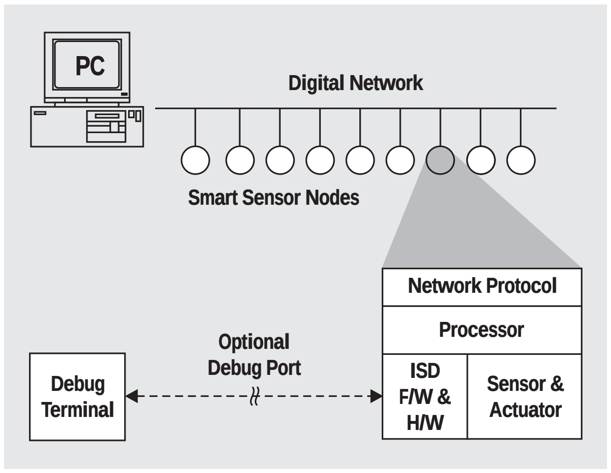
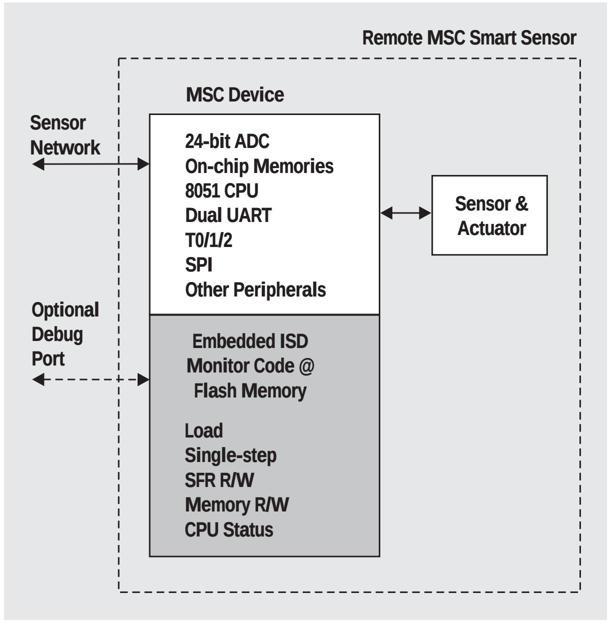


Figure 8. ISD monitor conceptual block diagram



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 bootloader 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.

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.

Code Listing 2: Minimum-configuration MSC monitor—MSCMon

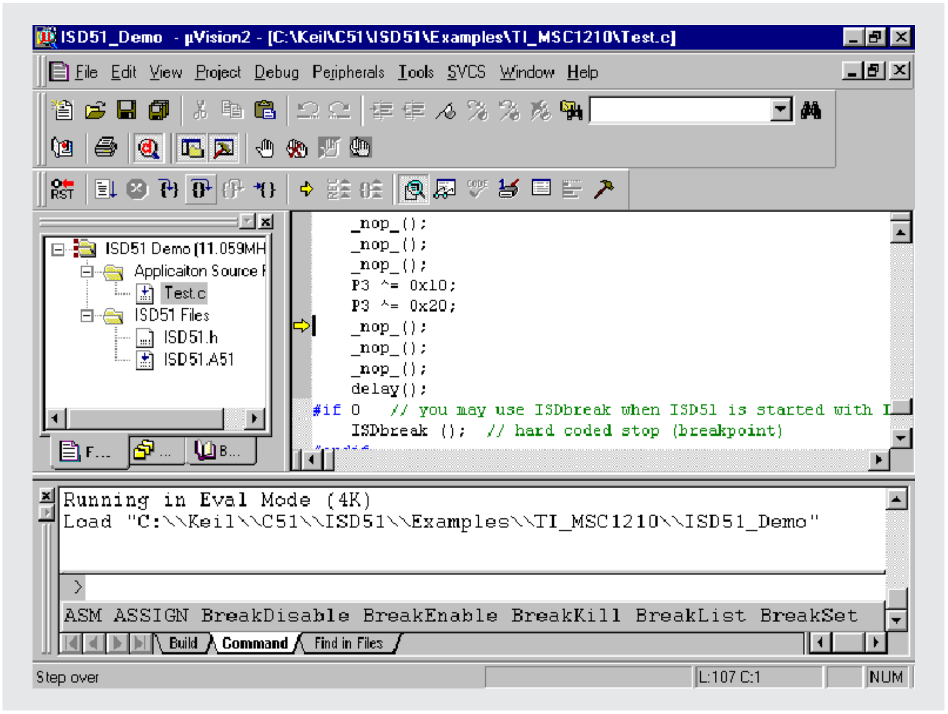
```
$include (reg1210.inc)
            CSEG     at 807FH
            ; HCR0: PML=0 RSL=1 to protect 0~1000H
            db         0BFH       Flash
            CSEG       at 0000H   ; Monitor code start
            LCALL      autobaud
            mov        R6, #high greet
            mov        R7, #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
```

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


ISD51_Demo - µVision2 - [C:\Keil\C51\ISD51\Examples\TI_MSC1210\Test.c]
File Edit View Project Debug Peripherals Tools SVCS Window Help
ISD51 Demo (11.059MH
Applicaiton Source F
Test.c
ISD51 Files
ISD51.h
ISD51.A51
F...
B...
Build
Command
Find in Files
Step over
L:107 C:1
NUM

```
    _nop_();
    _nop_();
    _nop_();
    P3 ^= 0x10;
    P3 ^= 0x20;
    _nop_();
    _nop_();
    _nop_();
    delay();
#if 0   // you may use ISDbreak when ISD51 is started with I
    ISDbreak ();  // hard coded stop (breakpoint)
```

```
Running in Eval Mode (4K)
Load "C:\\Keil\\C51\\ISD51\\Examples\\TI_MSC1210\\ISD51_Demo"

>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet
```

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. "MX51, AX51," www.hitex.com/products.html?axmx51.html~content	—
7. Russell Anderson, "Debugging Using the MSC1210 Boot ROM Routines," Application Report	sbaa079

Related Web sites

analog.ti.com
www.ti.com/sc/device/MSC1210
www.hitex.com
www.keil.com/demo/
www.raisonance.com/download/index.php

Using direct data transfer to maximize data acquisition throughput

By Mark Buccini (Email: m-buccini@ti.com)
MSP430 Applications Manager

Introduction

Increasing real-time throughput in MCU-based data acquisition systems is a challenge faced in many new applications. By definition, real-time applications must sample, digitize, transfer, and process acquired data before subsequent samples are made. Adequate CPU reserves must also be kept between samples to allow digital signal processing of the data. When acquisition rates exceeds 10 kSPS, engineers are often forced to make compromises due to the extreme overhead placed on the MCU. The basic task of moving acquired data becomes dominant and often unmanageable. Increasing CPU clock speeds or using specialized digital signal processors (DSPs) is often the only solution to meeting high data acquisition rates. This article offers alternatives that maximize system performance using single-chip mixed-signal MCU solutions with direct data transfer control.

System example using an external ADC

A typical multi-chip MCU-based data acquisition system is presented in Figure 1. The system includes an MSP430F123 MCU and a TLV1549 10-bit ADC. A low-power, 32-kHz watch crystal is used for the auxiliary clock (ACLK); and the MCU's internal digitally controlled oscillator (DCO) at its default frequency of ~1 MHz is used for the CPU's master clock (MCLK). The system samples a sensor signal with 10 bits of accuracy and transfers the acquired output code to on-chip RAM at 8192 samples per second. After 20 ADC samples have been acquired, the MCU will process the data stored in RAM. An application with such requirements could be a DTMF detection system.

The interface between the ADC and the MCU is glueless, using the MCU's integrated USART in SPI mode. To support the 8192-sample rate, the MCU's 16-bit Timer_A uses the ACLK with capture compare register 0 (CCR0) configured to trigger an interrupt service routine (CCR0_ISR) at the required sampling rate.

Inside the `Mainloop`, the MCU is normally in low-power mode 3 (LPM3). CCRO_ISR wakes the CPU, and an ADC sample is made. For each CCR0_ISR, software enables the ADC by resetting the ADC conversion start $\overline{\text{CS}}$. Data are exchanged between the ADC and MCU as two 8-bit bytes for each 10-bit conversion. The software starts and times a conversion by writing two dummy bytes back-to-back to the MCU's USART transmit buffer (TXBUF). The software then polls the USART receive interrupt flag (RXIFG) to indicate the receipt of a data byte from the ADC to the USART's receive buffer (RXBUF). Two bytes are received, packed, and transferred to MCU RAM with software and a pointer register R4. $\overline{\text{CS}}$ is reset, disabling the ADC. After 20 samples have been acquired, the `Mainloop` breaks from LPM3; and the data are processed as shown in Code Listing 1.

Figure 1. External ADC MCU data acquisition system

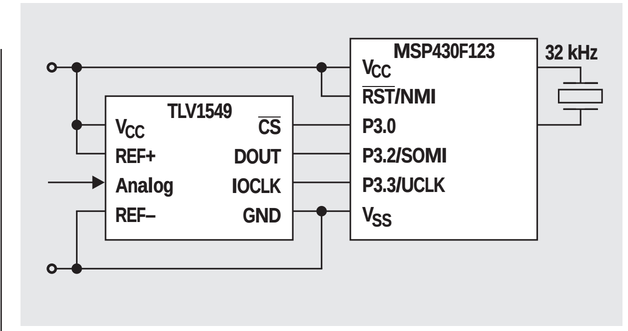
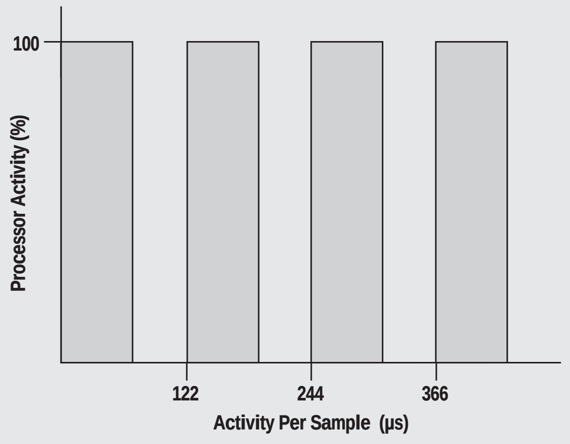


Figure 2. External ADC with MCU data acquisition system activity



For each external ADC measurement, 16 I/O clocks are required. The MCU's USART transfers data at half of the applied clock—the DCO, the same clock source used for the CPU MCLK. Thus 16 I/O clocks are equivalent to 32 MCLKs. The CCR0_ISR overhead and transfer of the ADC conversion code to memory takes additional cycles, for a total of 83 CPU MCLKs for each sample. With 8192 samples per second, an external ADC, and a 1-MHz MCLK, the CPU overhead is calculated as follows:

$$\text{CPU}_{\text{overhead (external ADC)}} = \frac{83 \times 8192}{1000000} = 0.68$$

The CPU is loaded 68%, as shown in Figure 2. The Timer_A CCR0_ISR and two CPU registers are also required.

Code Listing 1: MSP430F123/TLV1549 software example

```
#include  "msp430x12x.h"
;******************************************************************************
;   MSP-FET430P120 Demo - USART0 SPI Interface to TLV1549 10-bit ADC
;
;   M.Buccini - Texas Instruments, Inc - July 2002
;******************************************************************************
;------------------------------------------------------------------------------
            ORG     0F000h                  ; Program Reset
;------------------------------------------------------------------------------
RESET       mov.w   #0300h,SP               ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
SetupP3     bis.b   #0Ch,&P3SEL             ; P3.2,3 SPI option select
            bis.b   #09h,&P3DIR             ; P3.3,0 output direction
SetupSPI    bis.b   #USPIE0,&ME2            ; Enable USART0 SPI
            bis.b   #CKPH+SSEL1+SSEL0+STC,&UTCTL0 ; SMCLK, 3-pin
            bis.b   #CHAR+SYNC+MM,&UCTL0    ; 8-bit SPI Master
            mov.b   #02h,&UBR00             ; SMCLK/2 for baud rate
            clr.b   &UBR10                  ; SMCLK/2 for baud rate
            clr.b   &UMCTL0                 ; Clear modulation
            bic.b   #SWRST,&UCTL0           ; **SWRST**
SetupTA     mov.w   #TASSEL0+TACLR,&TACTL   ; ACLK, clear TAR
SetupC0     mov.w   #CCIE,&CCTL0            ; CCR0 interrupt enabled
            mov.w   #4-1,&CCR0              ; CCR0 counts to 4
            bis.w   #MC0,&TACTL             ; Start Timer_a in upmode
            eint                            ; Enable interrupts
                                            ;
Mainloop    clr.w   R4                      ; Clear pointer
Meas1549    bis.w   #LPM3,SR                ;
            bic.b   #01h,&P3OUT             ; Enable TLV1549, /CS reset
            mov.b   #00h,&TXBUF0            ; Dummy write to start SPI
            mov.b   #00h,&TXBUF0            ;
L1          bit.b   #URXIFG0,&IFG2          ; RXBUF ready?
            jnc     L1                      ; 1 = ready
            mov.b   &RXBUF0,R5              ; R5 = 00|MSB
            swpb    R5                      ; R5 = MSB|00
L2          bit.b   #URXIFG0,&IFG2          ; RXBUF ready?
            jnc     L2                      ; 1 = ready
            mov.b   &RXBUF0,R6              ; R6 = 00|LSB
            add.w   R6,R5                   ; R6 = MSB|LSB
            bis.b   #01h,&P3OUT             ; Disable TLV1549, /CS set
            mov.w   R5,0200h(R4)            ;
            incd.w  R4                      ;
            cmp.w   #040,R4                 ; R4 = 20 words?
            jne     Meas1549                ;
;         ** SIGNAL PROCESSING HERE
            jmp     Mainloop                ; Again
                                            ;
TA0_ISR     mov.w   #GIE ,0(SP)             ; System active on reti
            reti                            ;
                                            ;
            ORG     0FFFEh                  ; MSP430 RESET Vector
            DW       RESET                  ;
            ORG     0FFF2h                  ; Timer_A0 Vector
            DW       TA0_ISR                ;
            END
```

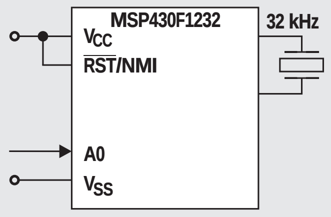
Using an internal ADC

To miniaturize applications and reduce system cost, an ADC is commonly integrated into an MCU. This integration provides a more compact system on-chip solution with the twofold benefit of reduced board space and the elimination of a serial port required for communicating between the MCU and ADC. An example of an MCU with an integrated 10-bit ADC is the MSP430F1232, as shown in Figure 3.

The MSP430F1232 eliminates the external 10-bit ADC, using the integrated ADC10 instead. CCR0_ISR is again used to wake the CPU from LPM3 in the `Mainloop`, as shown in Code Listing 2. The `Mainloop` manages the acquisition of 20 samples, storing the output code from the ADC10 memory buffer (ADC10MEM) to RAM. Only 40 MCLKs are required for each sample. With the internal ADC10 and a 1-MHz MCLK, the CPU overhead to support the 8192-sample rate is calculated as follows:

$$\text{CPU}_{\text{overhead (internal ADC)}} = \frac{40 \times 8192}{1000000} = 0.33$$

Figure 3. MSP430F1232 MCU data acquisition system



Code Listing 2: MSP430F1232 ADC10 software example

```
#include  “msp430x12x2.h”
;******************************************************************************
;   MSP-FET430P120 Demo - ADC10 Sample A0 20x, AVcc, TA0 ISR
;
;   M.Buccini - Texas Instruments, Inc - July 2002
;******************************************************************************
;------------------------------------------------------------------------------
            ORG     0F000h                  ; Program Reset
;------------------------------------------------------------------------------
RESET       mov.w   #0300h,SP               ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop watchdog timer
SetupADC10  mov.w   #ADC10SHT_2+ADC10ON,&ADC10CTL0 ; 16x
            bis.b   #01h,&ADC10AE           ; P2.0 ADC10 option select
SetupTA     mov.w   #TASSEL0+TACLR,&TACTL   ; ACLK, clear TAR
SetupC0     mov.w   #CCIE,&CCTL0            ; CCR0 interrupt enabled
            mov.w   #4-1,&CCR0              ; CCR0 counts to 4
            bis.w   #MC0,&TACTL             ; Start Timer_a in upmode
            eint                            ; Enable interrupts
                                            ;
Mainloop    clr.w   R4                      ; Clear pointer
MeasADC10   bis.w   #ENC+ADC10SC,&ADC10CTL0 ; Start sampling/conversion
            bis.w   #LPM3,SR                ;
            mov.w   &ADC10MEM,0200h(R4)     ;
            incd.w  R4                      ;
            cmp.w   #040,R4                 ; R4 = 20 words?
            jne     MeasADC10               ;
;         ** SIGNAL PROCESSING HERE
            jmp     Mainloop                ; Again
                                            ;
TA0_ISR     mov.w   #GIE ,0(SP)             ; System active on reti
            reti                            ;
                                            ;
            ORG     0FFFEh                  ; MSP430 RESET Vector
            DW      RESET                   ;
            ORG     0FFF2h                  ; Timer_A0 Vector
            DW      TA0_ISR                 ;
            END
```

The CPU is loaded 33%, as shown in Figure 4. Using an internal ADC reduces the CPU overhead by 50% compared to when an external ADC is used. The CPU overhead reduction is accomplished by eliminating the software and delay associated with servicing the serial port communication between the MCU and the external ADC. The free serial port can be used for other features or completely eliminated in cost-sensitive applications.

Data transfer controller

In addition to ADC10, the MSP430F1232 contains a data transfer controller (DTC). The DTC provides the capability of automatically transferring conversion code from the ADC10 output buffer memory (ADC10MEM) directly to on-chip memory—without CPU intervention, as shown in Figure 5. The start address (ADC10SA) of the transfer sequence can point anywhere in the MCU's memory. The total number of conversions is configured with the ADC10DTC1 control register. The DTC can transfer ADC10 output code in a single-block sequence or a dual-block sequence, both with or without repeating, and both with or without interrupt capability. In the example, the DTC transfers code into a single block, repeating with interrupt.

The DTC does not require the CPU to transfer ADC10MEM to memory, but it does require one memory data bus (MDB) clock, which is the same as the MCLK used by the CPU. When an MDB clock is required for transfer, the DTC will halt any CPU activity for exactly one clock. This halt is to prevent memory conflict between the DTC and CPU. In effect, the DTC "steals" one MCLK from

Figure 4. MSP430F1232 ADC10 data acquisition system activity

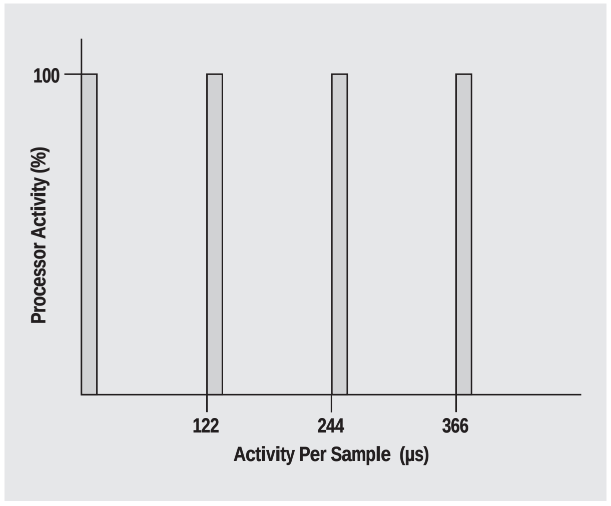
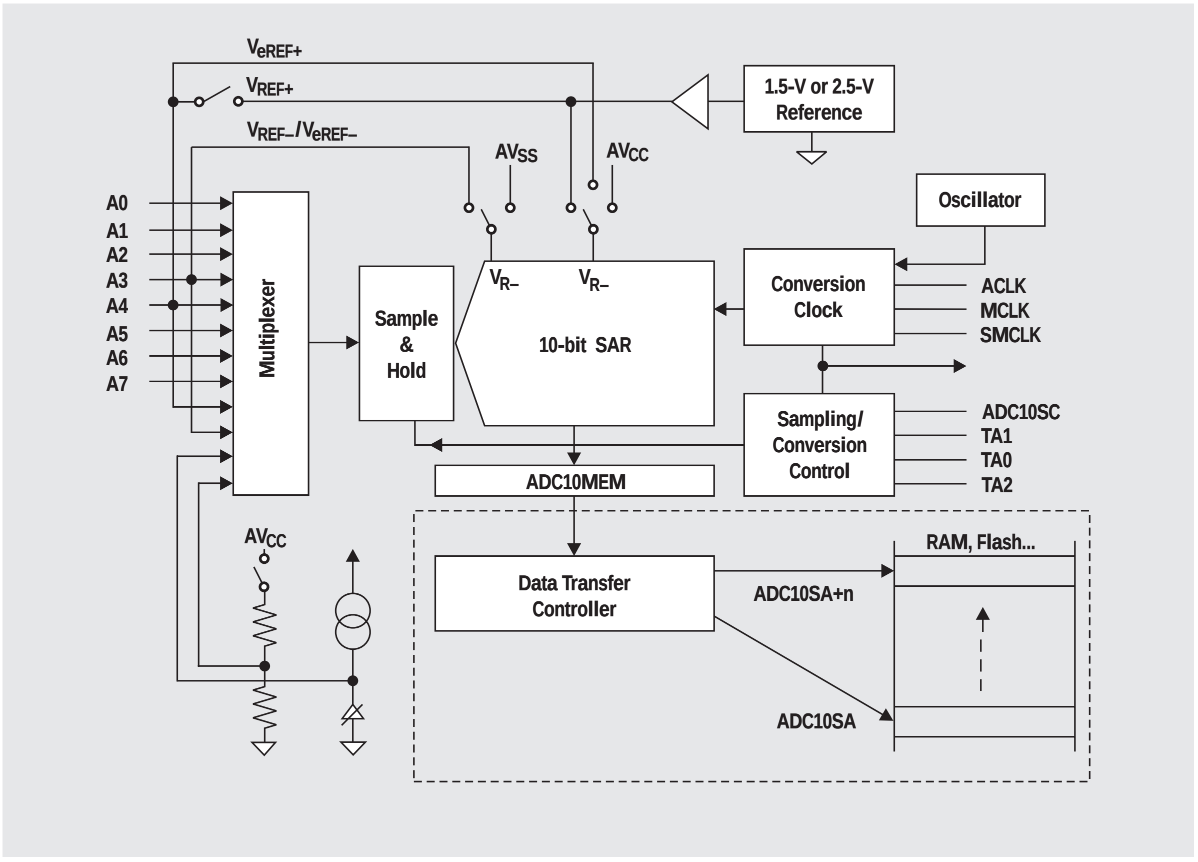


Figure 5. MSP430F1232 ADC10 block diagram



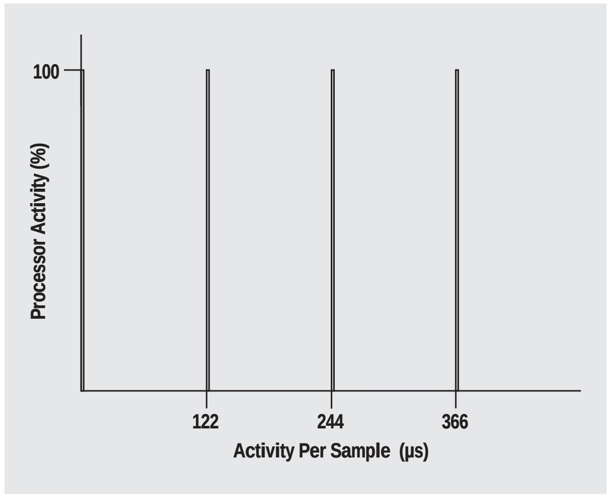
the CPU for each DTC transfer. This may be interpreted as CPU overhead. If the CPU is not active and using the MDB, the effect of the DTC is of no consequence.

With 8192 samples per second, a 1-MHz CPU MCLK, and use of the DTC, the CPU overhead is calculated as follows:

$$\text{CPU}_{\text{overhead (internal ADC with DTC)}} = \frac{1 \times 8192}{1000000} = 0.008$$

The CPU is loaded less than 1% while sustaining an 8192-sample rate, as shown in Figure 6. The DTC allows over 99% of the CPU resources to be available for digital signal processing and control. In the example, the DTC will set the ADC10 interrupt after a block of 20 samples has been transferred. The function of R4, used as a loop counter in the previous two examples, is no longer required. The DTC and ADC10 interrupt service routine (ADC10_ISR) will wake the CPU from LPM3 in `Mainloop` only after a complete block of 20 samples has been taken and transferred automatically to RAM, as shown in Code Listing 3.

Figure 6. MSP430F1232 ADC10 with DTC data acquisition system activity



Code Listing 3: MSP430F1232 ADC10 with DTC software example

```
#include  "msp430x12x2.h"
;*****************************************************************************
;   MSP-FET430P120 Demo - ADC10 Sample A0 20x, AVcc, TA0 Trigger, DTC DCO
;
;   M.Buccini - Texas Instruments, Inc - July 2002
;*****************************************************************************
;-----------------------------------------------------------------------------
            ORG     0E000h                  ; Program Start
;-----------------------------------------------------------------------------
RESET       mov.w   #0300h,SP               ; Initialize stackpointer
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
SetupADC10  mov.w   #SHS_2+CONSEQ_2,&ADC10CTL1 ; TA0 trigger
            mov.w   #ADC10SHT_2+ADC10ON+ADC10IE,&ADC10CTL0;
            mov.b   #ADC10CT,&ADC10DTC0     ; Continuos
            mov.b   #020,&ADC10DTC1         ; 20 conversions
            bis.b   #001h,&ADC10AE          ; P2.0 ADC10 option select
SetupTA     mov.w   #TASSEL0+TACLR,&TACTL   ; ACLK, clear TAR
SetupC0     mov.w   #OUTMOD_4,&CCTL0        ; CCR0 toggle
            mov.w   #2-1,&CCR0              ; PWM Period
            mov.w   #0200h,&ADC10SA         ; Data buffer start
            bis.w   #ENC,&ADC10CTL0         ; Sampling and conversion ready
            bis.w   #MC0,&TACTL             ; Start Timer_a in upmode
            eint                            ; Enable interrupts
                                            ;
Mainloop    bis.w   #LPM3,SR                ; LPM3, ADC10 ISR will force exit
;         ** SIGNAL PROCESSING HERE
            jmp     Mainloop                ; Again
                                            ;
ADC10_ISR   mov.w   #GIE,0(SP)              ; System active on reti
            reti                            ;
                                            ;
            ORG     0FFFEh                  ; MSP430 RESET Vector
            DW      RESET                   ;
            ORG     0FFEAh                  ; ADC10 Vector
            DW      ADC10_ISR               ;
            END
```

DTC channel scanning

Some systems, such as an electricity meter, require channel scanning, mixing alternate samples of voltage and current. Using software and an internal or external ADC to switch channels adds at least 10 MCLK cycles of overhead for each sample. At an 8192-sample rate with a 1-MHz MCLK, software-based channel scanning adds an additional 0.08% CPU overhead. DTC provides the ability to scan a mix of different channels (i.e., A2, A1, A0, A2, A1, A0...) automatically without any CPU resources. If channel scanning is useful in an application, the benefits of using a DTC are magnified.

SOC-introduced phase error

High-performance data acquisition systems must take into account several factors in addition to the sampling speed discussed so far. In many applications, an analog sensor output must be sampled at very exact intervals to detect small phase changes. For example, stimuli may be presented to sensor input, with a delayed response measured at an output. For demonstration purposes, a repeating 10-kHz triangle signal from 0 to 3 V is shown in Figure 7. The rate of change of the triangle signal is calculated as follows:

$$\text{Signal}(dv/dt) = 3\text{ V}/100\ \mu\text{s} = 30\text{ mV}/\mu\text{s}$$

With a 10-bit ADC in a 3-V system, the ADC step size is the ADC reference (assumed to be the supply) divided by 1024.

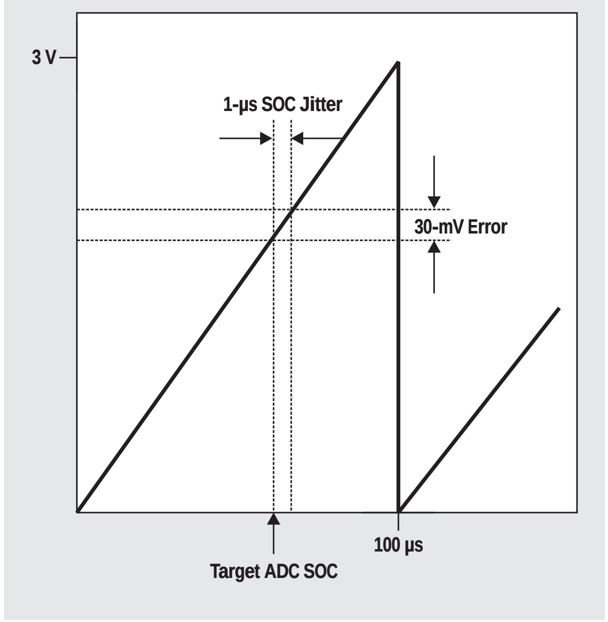
$$\text{ADC10(step)} = 3\text{ V}/1024 = 3\text{ mV}$$

If only software is used to trigger an ADC start of conversion (SOC), some uncertainty or jitter will always be present. Other always-present CPU activity such as servicing an interrupt or UART handler causes this uncertainty. Unless all other CPU activity is ceased prior to a software-initiated SOC—very impractical for modern embedded applications—some uncertainty with software-triggered SOC will always be possible.

For example, using the example system with an MCLK of 1 MHz (1-µs clock), the effect of just one clock of SOC uncertainty will introduce 30 mV of ADC uncertainty or jitter error. This 30 mV of uncertainty error is equivalent to 10 10-bit steps. The 10-bit ADC output code with 1 µs of SOC jitter is reduced to an accuracy of a 7-bit ADC!

When an ADC10 is used, a CCRx output can automatically trigger ADC10 SOC with the precision of the used timer clock—all interrupt and latency uncertainty associated with software-driven SOC is removed completely from the system. The ADC10 SOC will be triggered with perfect timing, using the CCRx hardware asynchronously and regardless of other CPU, software, or system activity.

Figure 7. ADC sampling uncertainty error



DTC system throughput performance increase

Using DTC in data acquisition applications relieves the CPU from the burden of transferring ADC output code to memory. The DTC in itself reduces CPU loading by a factor of 40× compared with using an internal ADC10, and by a factor of greater than 80× compared with using an external ADC.

In the example using an external ADC, the MCU MCLK would need to be increased to 4 MHz to acquire the ADC's maximum conversion rate of 38 kSPS—and the CPU would be loaded 79%. At the same 4-MHz MCLK and with the MSP430F1232's internal ADC10, 100 kSPS can be acquired at 100% CPU loading; and, with the DTC, 235 kSPS are possible at only 6% CPU loading. See Figure 8.

MCUs with DTC-enabled ADCs allow well over an order-of-magnitude reduction in CPU loading in data acquisition systems. CPU performance reserves can be focused on differentiated digital signal processing instead of on the basic process of sampling and moving data.

References

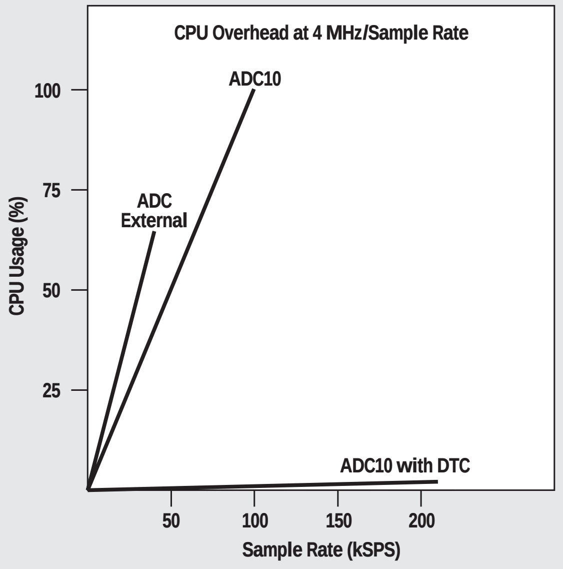
For more information related to this article, you can download an Acrobat Reader file at www-s.ti.com/sc/techlit/*litnumber* and replace *"litnumber"* with the **TI Lit. #** for the materials listed below.

Document Title	TI Lit. #
1. "MSP430x1xx Family," User's Guide	slau049
2. "MSP430x11x2, MSP430x12x2 Mixed Signal Microcontroller," Data Sheet	slas361
3. "MSP430x12x Mixed Signal Microcontroller," Data Sheet	slas312
4. "TLV1549C, TLV1549I, TLV1549M 10-Bit Analog-to-Digital Converters with Serial Control," Data Sheet	slas071

Related Web sites

analog.ti.com
www.ti.com/sc/device/*partnumber*
Replace *partnumber* with MSP430F123, MSP430F1232 or TLV1549

Figure 8. Data acquisition CPU overhead

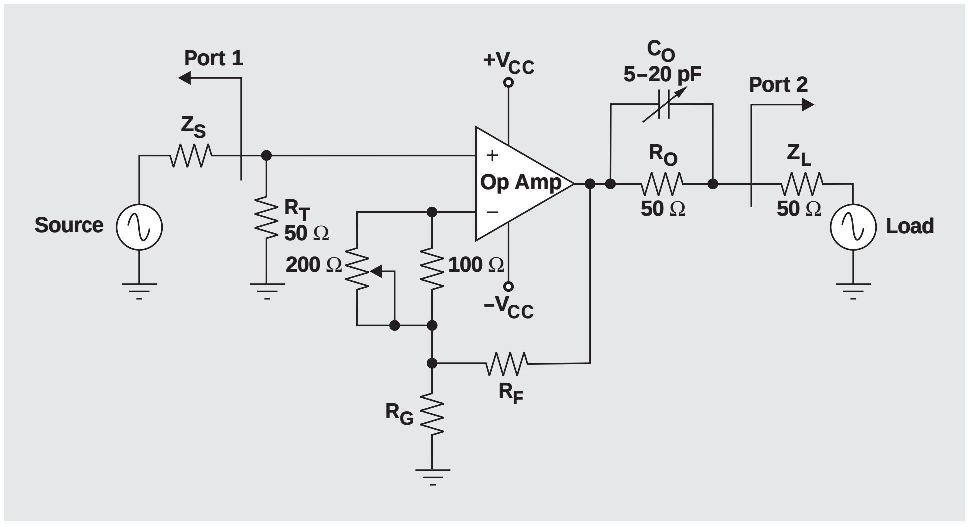


Using high-speed op amps for high-performance RF design, Part 2

By Bruce Carter (Email: r-carter5@ti.com)
Advanced Linear Products, Op Amp Applications

This is Part 2 in a two-part series devoted to the topic of using op amps for RF design. Part 1 focused on how to form an RF stage from op amps and the scattering parameters. Part 2 focuses on other RF specifications and some of the finer points of RF design.

Figure 1. Frequency-response peaking



Frequency-response peaking

Current-feedback amplifiers allow an easy resistive trim for frequency peaking that has no impact on the forward gain. This frequency-response flatness trim has the same effect in non-inverting and inverting configurations.

Figure 1 shows this adjustment added to an inverting circuit. This resistive trim inside the feedback loop has the effect of adjusting the loop gain and, hence, the frequency response without adjusting the signal gain, which is still set by R_F and R_G.

Values for R_F and R_G must be reduced to compensate for the addition of the trim potentiometer, although their ratio and, hence, the gain should remain the same. The adjustment range of the pot, combined with the lower R_F value, ensures that the frequency response can be peaked for slight variations in the current-feedback amplifier parameters.

–1-dB compression point

The –1-dB compression point is defined as the output power, at a fixed input frequency, where the amplifier's actual output power is 1 dBm less than expected. Stated another way, it is the output power at which the actual amplifier gain has been reduced by 1 dB from its value at lower output powers. The –1-dB compression point is the way RF designers talk about voltage rails.

Op amp designers and RF designers have very different ways of thinking about voltage rails, which are related to the requirements of the systems they design. An op amp designer—interfacing op amps to data converters, for example—takes great pains not to hit the voltage rail of the op amp, thus losing precious codes. An RF designer, on the other hand, is often concerned with squeezing the last half decibel out of an RF circuit. In broadcasting, for example, a very slight increase in decibels means a lot more coverage. More coverage means more audience and more advertising dollars. Therefore, slight clipping is acceptable, as long as resulting spurs are within FCC regulations.

Standard ac-coupled RF amplifiers show a relatively constant –1-dB compression power over their operating frequency range. For an operational amplifier, the maximum output power depends strongly on the input frequency. The two op amp specifications that serve a similar purpose to –1-dB compression are V_{OM} and slew rate.

At low frequencies, increasing the power of a fixed frequency input will eventually drive the output "into the rails"—the V_{OM} specification. At high frequencies, op amps will reach a limit on how fast the output can transition (respond to a step input)—the slew rate limitation. The op amp slew rate specification is divided by two, because of the matching resistor used at the output.

As is the case for op amps used in any other application, it is probably best to avoid operation near the rails, as the inevitable distortion will produce harmonics in the RF signal that are probably undesirable for FCC testing. That said, if harmonics are still at an acceptable level at the –1-dB compression point, it can be a very useful way to boost power to a maximum level out of the circuit.

Two-tone, third-order intermodulation intercept

When two closely spaced signals are present in the RF bandwidth being amplified, sum and difference frequencies are created. These sum and difference frequencies are intermodulation harmonics. They are undesirable and may lead to problems with FCC testing of the system.

The problem with these harmonics is that they increase in amplitude three times as fast as the fundamentals. Figure 2 shows a theoretical system with a beginning fundamental signal level of 0 dBm, with intermodulation harmonics at –60 dBm. As the amplitude of the fundamentals is increased, the harmonics increase at three times the rate of the fundamentals, leading to an eventual intercept of the two lines at +30 dBm.

The typical RF circuit cannot be adjusted to +30 dBm, so the third-order intermodulation intercept is theoretical. The third-order intermodulation intercept should be as high as possible so that the intermodulation harmonics will be proportionally lower at any real-world fundamental level.

Figure 3 illustrates the two-tone intermodulation intercept in a different way. The fundamental frequencies at f_O–Δf and f_O+Δf are shown amplified in 10-dBm steps in Figures 3(b), 3(c), and 3(d). While the fundamentals are raised a total of 30 dBm, the third-order harmonics at f_O–3Δf and f_O+3Δf increase 90 dBm, eventually attaining the same amplitude as the fundamentals! Clearly, this must be avoided in a working RF system if spurs are to be rejected at all.

Figure 2. Two-tone, third-order intermodulation intercept

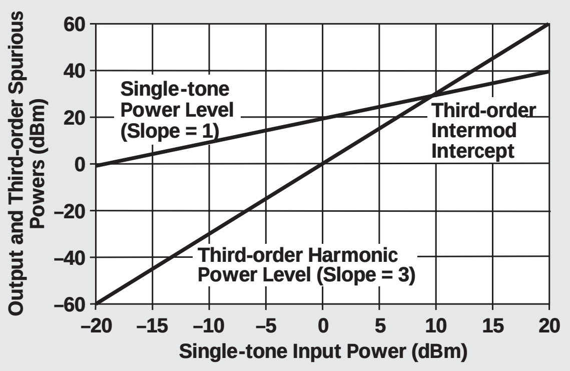
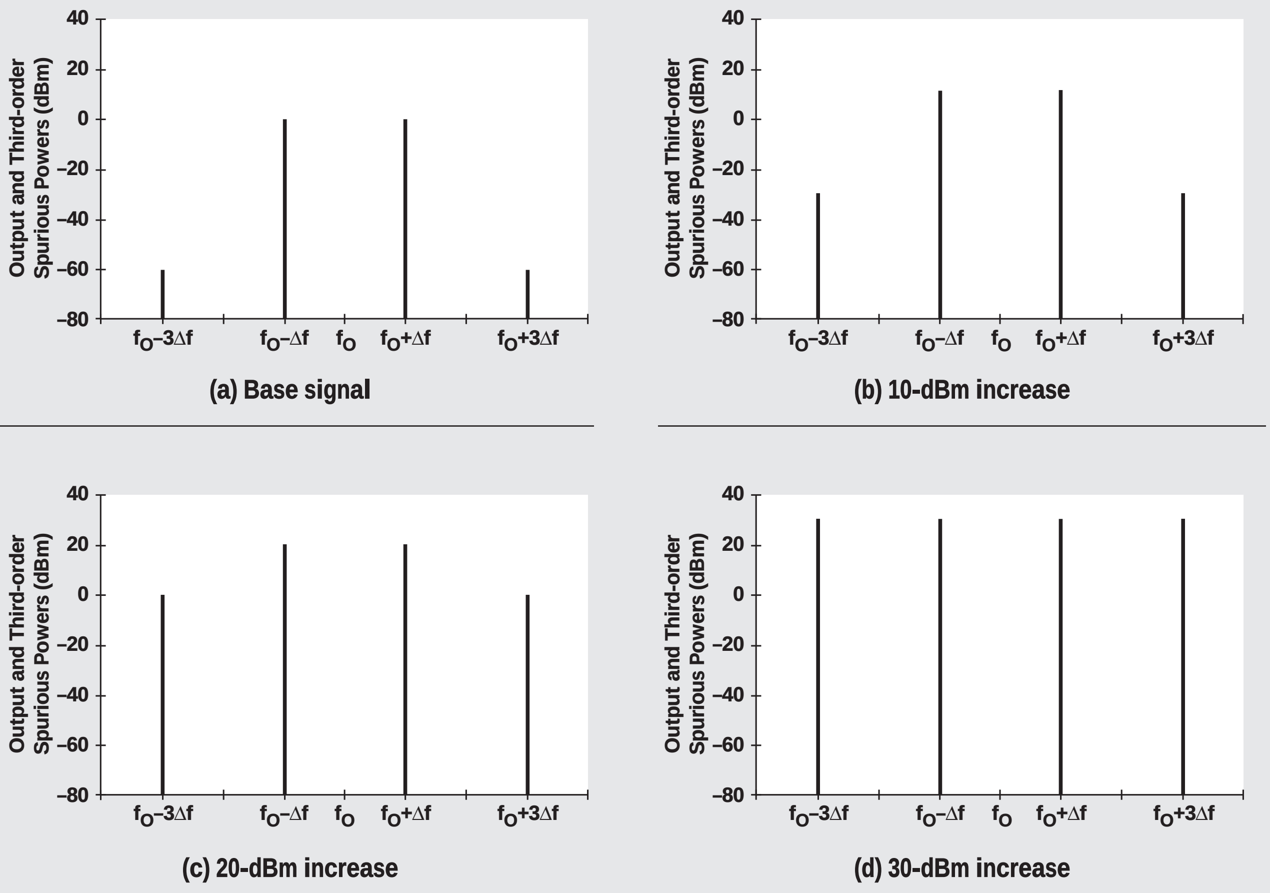


Figure 3. Two-tone, third-order intermodulation intercept amplitudes



Noise figure

The RF noise figure is the same thing as op amp noise, when an op amp is the active element. There is some effect from thermal noise in resistors used in RF systems, but the resistor values in RF systems are usually so small that their noise can be ignored.

Noise for an op amp RF circuit is dependent on (1) the bandwidth being amplified and (2) gain.

This example uses the 11.5 nV/$\sqrt{\text{Hz}}$ op amp. The application is a 10.7-MHz IF amplifier. The signal level is 0 dBV, and the gain is unity.

Figure 4 is extrapolated from real data. The 1/f corner frequency, in this case, is much lower than the band-width of interest. Therefore, the 1/f noise can be completely discounted (assuming that filtering removes any noise that would cause the amplifier or data converter to saturate).

For narrow bandwidths, noise may be quite low! The noise for various bandwidths is shown in Table 1.

Figure 4. Noise bandwidth

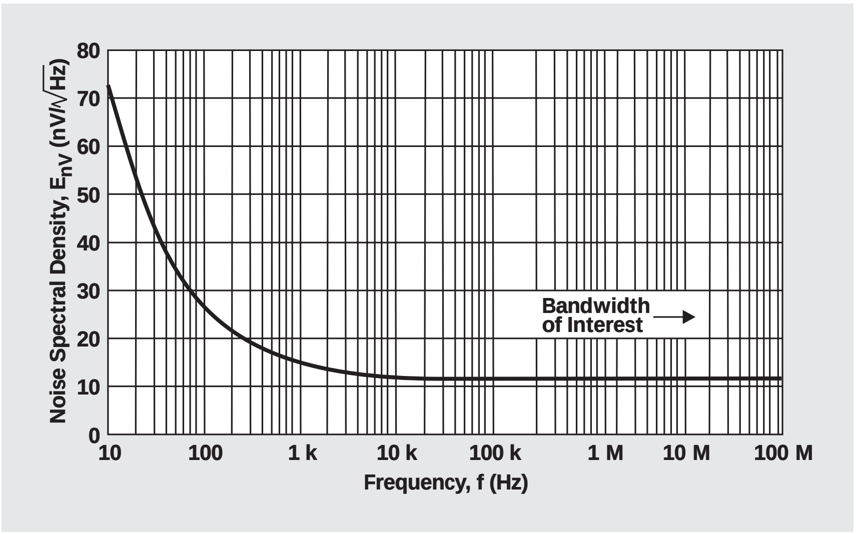


Table 1. Noise for various bandwidths

BANDWIDTH (kHz)	EQUIVALENT INPUT NOISE (µV)	SIGNAL-TO-NOISE RATIO (dB)
280	6.09	–104.3
230	5.52	–105.2
180	4.88	–106.2
150	4.45	–107.0
110	3.81	–108.4
90	3.45	–109.2

Obviously, there is a slight advantage in reducing the bandwidth. A lower-noise op amp, however, will provide the greatest benefit.

Noise is amplified by the gain of the stage. Therefore, if a stage has high gain, care must be taken to find a low-noise op amp. If the gain of a stage is low, the noise will not be amplified as much, and a less expensive op amp may be suitable.

Conclusion

Op amps are suitable for RF design, provided that the cost can be justified. They are more flexible to use than discrete transistors, because the biasing of the op amp is independent of the gain and termination. Current-feedback amplifiers are more suitable for high-frequency, high-gain RF design because they do not have the gain/bandwidth limitations of voltage-feedback op amps.

Scattering parameters for RF amplifiers constructed with op amps are very good. Input and output VSWRs are good because the effects of termination and matching resistors can be controlled independently of stage biasing. Reverse isolation is very good because the RF stage uses an op amp consisting of dozens or hundreds of transistors instead of a single transistor. Forward gain is very good with a current-feedback amplifier.

Special considerations apply to RF design that do not normally apply to op amp design—the phase linearity, the –1-dB compression point (as opposed to voltage rails), the two-tone, third-order intermodulation intercept, peaking, and noise bandwidth. In just about every case, the performance of an RF stage implemented with op amps is better than one implemented with a single transistor.

Related Web site

analog.ti.com

FilterPro™ low-pass design tool

By John Bishop (Email: j-bishop1@ti.com)
Applications Specialist, High-Performance Linear Products

Introduction

Although low-pass filters are vital in modern electronics, their design and verification can be tedious and time-consuming. The FilterPro program aids in the design of low-pass filters implemented with the multiple feedback (MFB) and Sallen-Key topologies. This article is an introduction to the use and capabilities of FilterPro.

History of FilterPro

In 1991 Burr-Brown released a version of FilterPro as a DOS application by Bruce Trump and R. Mark Stitt. When TI purchased Burr-Brown in 2000, the idea of updating some of Burr-Brown's tools for customer use was proposed, including writing a Windows® version of FilterPro. The source code was written in Q-Basic, and the best path for the upgrade seemed to be Visual Basic®. A new operator interface was developed, and the original computational subroutines were able to be used nearly verbatim.

The major difference between the original FilterPro for DOS and FilterPro for Windows is that all menu-selected windows available in the old version are visible on a single form of the new version. In addition, the new version displays the circuits schematically instead of referring to schematics in the application note.

Easy design of low-pass filters

Once the FilterPro program is started, several parameters must be entered to design a low-pass filter. The cutoff frequency, number of poles, filter type, and filter configuration are the main inputs. Because there are instances where the Sallen-Key filter topology is a better choice, the user can specify either MFB or Sallen-Key topology.

An ideal low-pass filter would completely eliminate signals above the cutoff frequency and perfectly pass signals below it (in the pass-band). In real filters, various trade-offs are made in an attempt to approximate the ideal. Some filter types are optimized for gain flatness in the pass-band, some trade off gain variation (ripple) in the pass-band for steeper roll-off, and still others trade off both flatness and rate of roll-off in favor of pulse-response fidelity. FilterPro supports the three most commonly used all-pole filter types: Butterworth, Chebyshev, and Bessel. Figures 1 and 2 are examples of filters designed by FilterPro that use two of these filter types.

Figure 1. Response vs. frequency of even-order (4-pole), 3-dB-ripple Chebyshev filter

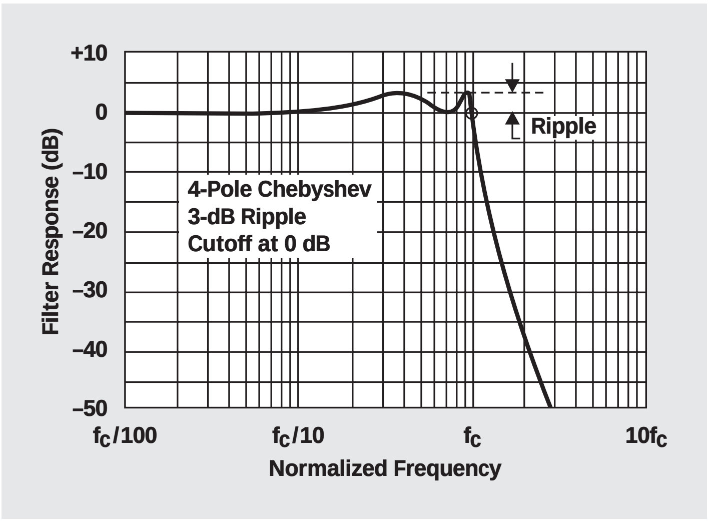
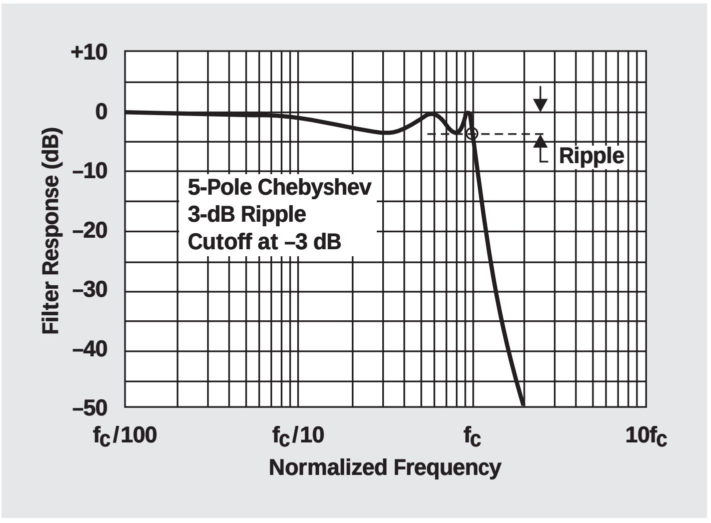


Figure 2. Response vs. frequency of even-order (5-pole), 3-dB-ripple Chebyshev filter



Filter circuits

Even-order filters designed with FilterPro consist of cascaded sections of complex pole pairs. Odd-order filters contain an additional real-pole section. The program automatically places lower-Q stages ahead of higher-Q stages to prevent op amp output saturation due to gain peaking. The program can be used to design filters up to tenth order.

Complex pole-pair circuit

The choice of a complex pole-pair circuit depends on performance requirements. FilterPro supports the two most commonly used active pole-pair circuit topologies. Figures 3–5 show the three different pole-pair schematics.

Using the FilterPro program

With each data entry, the program automatically calculates filter performance and values for all filter components. This allows you to use a "what if" spreadsheet-type design approach. For example, you can quickly determine, by trial and error, how many poles are needed for a given roll-off.

Computer requirements

The operating system required for FilterPro for Windows should be Windows 95, NT 3.5, or newer. The display should be configured for at least 800 × 600. It is helpful, but not required, to have a printer (capable of printing a screen dump) available either locally or on a network.

Installation

To install FilterPro on your computer, go to analog.ti.com and, under AMPLIFIERS AND COMPARATORS, click on Engineer Design Utilities. Download FilterPro and then run the setup.exe program from your hard drive.

Getting started

The first time you use the program, you may want to double-click on the FilterPro icon on the desktop. Another way is to select Start, Programs, and FilterPro. The start-up screen shows default values for a 3-pole, 1-kHz Butterworth filter. Figure 6 shows a 9-pole MFB design with a Chebyshev response and a cutoff frequency of 100 kHz. Notice that the ripple is .001 dB. If a higher ripple were entered, the response would be different. For a different filter design, click on the radio buttons and/or enter different values in the Settings frame as follows:

1. Under Circuit Type, choose the pole-pair circuit: Sallen-Key or MFB.
2. Under Filter Type, select Bessel, Butterworth, or Chebyshev.
3. For the Chebyshev filter type, enter the ripple amount in the Ripple box at upper right: 0.0001 dB to 10 dB.
4. In the Poles box, enter the desired number of poles: 1 to 10 (minimum of 2 for Bessel or Chebyshev).
5. In the Cutoff Freq. box, enter the filter cutoff frequency: 1 MHz to 100 MHz.
6. If you want to view the gain/phase response of the current filter design at a particular frequency (the default value is 10 times the cutoff frequency), enter the frequency of interest in the Response Freq. box. The gain/phase values are displayed in the f_n, Q, and Response fields at the lower right of the screen.
7. If you want to change the resistor scaling, enter a value in the R1 Seed box.
8. If you want to change the gain of a section, enter the desired value in the appropriate Gain boxes under Optional Entry. The default value for gain is 1.0 V/V in each section.
9. If you want to choose your own capacitor values, enter them in the appropriate C1 or C2 boxes under Optional Entry.

Figure 3. MFB complex pole-pair section (gain = –R2/R1)

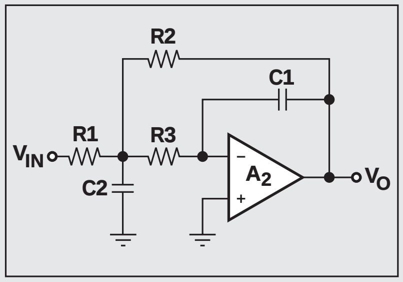


Figure 4. Sallen-Key complex pole-pair section (gain = 1)

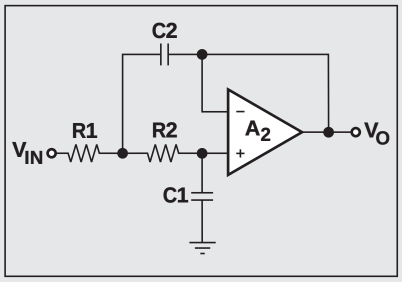
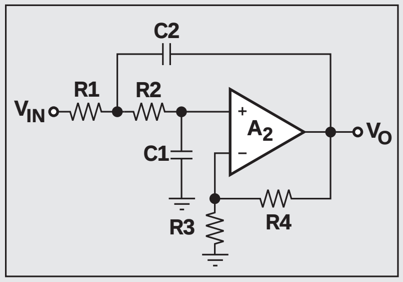


Figure 5. Sallen-Key complex pole-pair section (gain = 1+ R4/R3)



10. If you want to design with standard 1% resistors instead of exact resistors, click the "1% Resistors" check box.

On-screen prompts to the left of the response graph will guide you in program use. Refer to this article for more detail, if needed.

Program features

To print results

To print results, select the Print menu at the top of the screen. It will display a dialog box that can be used to print the screen. When printing is started, the normal screen size will be increased to include a table containing sensitivity data or component values not shown on the schematic in Figure 6. The larger screen will then be captured and sent to the printer. If the screen is not fully visible due to position or size, only what is visible will be printed.

Sensitivity

Sensitivity is the measure of the vulnerability of a filter's performance to changes in component values. The important filter parameters to consider are natural frequency (f_n) and Q.

f_n sensitivity for both MFB and Sallen-Key

Sensitivity of f_n to resistor, capacitor, and amplifier gain variations is always low for both the Sallen-Key and MFB filter topologies.

$$S_R^f = S_C^f = \pm 0.5\%/\% \text{ and}$$

$$S_K^f = 0, \text{ where}$$

S_R^f, S_C^f, and S_K^f = sensitivity of f_n to resistor, capacitor, and gain variations, respectively.

Q sensitivity

For the MFB topology, sensitivities to Q are also always low, but sensitivities for the Sallen-Key topology can be quite high—exceeding $2KQ^2$. K is the variable used here for op amp gain. At unity gain, the Sallen-Key Q sensitivity to resistor and capacitor variations will always be low. Unfortunately, however, the sensitivity of the unity-gain Sallen-Key pole pair to op amp gain can be high.

Q sensitivity for MFB pole pair

$$S_C^Q = \pm 0.5\%/\%,$$

$$S_R^Q = \pm \frac{R2 - R3 - KR3}{2(R2 + R3 + KR3)} \text{ (MFB complex pole pair), and}$$

$$S_K^Q = \pm \frac{KR3}{R2 + R3 + KR3} \text{ (MFB complex pole pair).}$$

Notice, by inspection, that S_R^Q is always less than ±0.5%/%, and S_K^Q is always less than 1.0%/%.

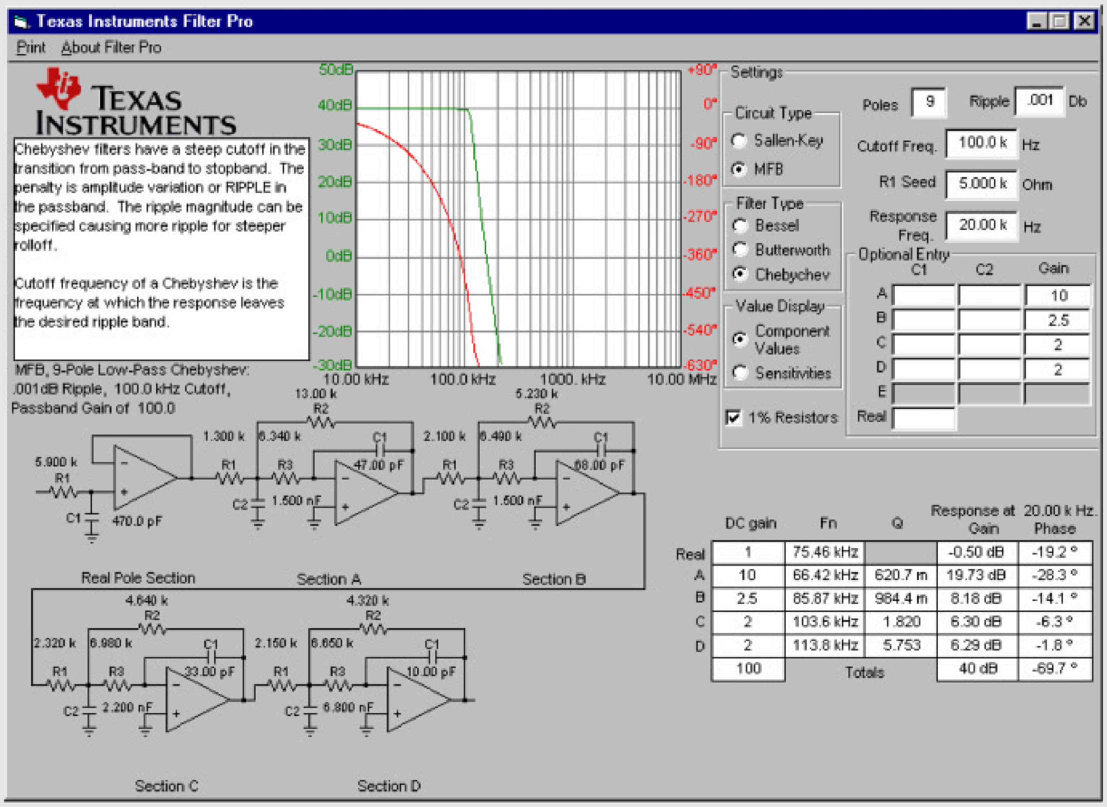
Q sensitivity for Sallen-Key pole pair (gain = 1)

$$S_C^Q = \pm 0.5\%/\% \text{ and}$$

$$S_R^Q = \pm \frac{R1 - R2}{2(R1 + R2)} \text{ (Sallen-Key complex pole pair).}$$

Therefore, S_R^Q is always less than ±0.5%/%.

Figure 6. Screen display of FilterPro showing a 9-pole MFB filter (gain = 40 dB)



Chebyshev filters have a steep cutoff in the transition from pass-band to stopband. The penalty is amplitude variation or RIPPLE in the passband. The ripple magnitude can be specified causing more ripple for steeper rolloff.

Cutoff frequency of a Chebyshev is the frequency at which the response leaves the desired ripple band.

MFB, 9-Pole Low-Pass Chebyshev:
.001dB Ripple, 100.0 kHz Cutoff,
Passband Gain of 100.0

Settings

Circuit Type: ( ) Sallen-Key [✓ MFB]

Filter Type: ( ) Bessel ( ) Butterworth [✓ Chebychev]

Value Display: [✓ Component Values] ( ) Sensitivities

[✓] 1% Resistors

Poles: 9 Ripple: .001 Db

Cutoff Freq.: 100.0 k Hz

R1 Seed: 5.000 k Ohm

Response Freq.: 20.00 k Hz

Optional Entry

	C1	C2	Gain
A			10
B			2.5
C			2
D			2
E			
Real			

	DC gain	Fn	Q	Response at 20.00 k Hz. Gain	Phase
Real	1	75.46 kHz		-0.50 dB	-19.2 °
A	10	66.42 kHz	620.7 m	19.73 dB	-28.3 °
B	2.5	85.87 kHz	984.4 m	8.18 dB	-14.1 °
C	2	103.6 kHz	1.820	6.30 dB	-6.3 °
D	2	113.8 kHz	5.753	6.29 dB	-1.8 °
	100	Totals		40 dB	-69.7 °

$S^2 < S_K^Q < 2Q^2S$ (Sallen-Key complex pole pair), where

S_R^Q, S_C^Q, and S_K^Q = sensitivity of Q to resistor, capacitor, and gain variations (%/%), respectively.

K = op amp gain. For the circuit in Figure 3, K = R2/R1. For the circuit in Figure 4, K = 1.0. For the circuit in Figure 5, K = 1 + R4/R3.

Note: FilterPro always selects component values so that unity-gain Sallen-Key S_K^Q will be closer to Q^2 than to $2Q^2$. However, it will allow you to design Sallen-Key pole pairs with high sensitivities (high Qs and gain >> 1). You must make sure that sensitivities to component variations do not make these designs impractical. A feature in the display allows you to view the f_n and Q sensitivities of filter sections to resistor and capacitor variations.

Using the sensitivity display feature

To use the Sensitivity display option, click on the Sensitivity radio button in the Settings frame (see Figure 6). The schematic shows sensitivity of f_n (S^f) and Q (S^Q) to each component for each filter section.

Rather than displaying the derivative with respect to component variations, the program calculates the f_n and Q change for a 1% change in component values. This gives a more realistic sensitivity value for real-world variations.

Using the seed resistor setting

The Seed Resistor setting allows you to scale the computer-selected resistor values to match the application. Move the cursor to the R1 Seed field and enter your *seed* resistor value. The default value of 10 kΩ is suggested for most applications.

When the circuit is in a power-sensitive environment (battery power, solar power, etc.), the value can be increased to decrease power consumption. Some high-speed op amps require lower feedback resistance, so their seed resistor value should be decreased.

Higher resistor values—e.g., 100 kΩ—can be used with FET-input op amps. At temperatures below about 70°C, dc errors and excess noise due to op amp input bias current will be small. Remember, however, that noise due to the resistors will be increased by $\sqrt{n}$ where n is the resistor increase multiplier.

Lower resistor values—e.g., 50 Ω—are a better match for high-frequency filters using the OPA620 or OPA621 op amps.

Using the capacitor option

Compared to resistors, capacitors with tight tolerances are more difficult to obtain and can be much more expensive. The capacitor fields (C1 and C2 boxes under Optional Entry, shown in Figure 6) allow you to enter actual measured capacitor values. In this way, an accurate filter response can be achieved with relatively inexpensive components. Prompts on the left of the screen advise minimum/maximum capacitor entry limits. With each capacitor entry, the program will select the exact or closest standard 1% resistor values as before.

Unless capacitor entries are made, FilterPro selects capacitors from standard E6 values (six values per decade). When values other than E6 are used (E12, measured, etc.), then the appropriate values should be entered.

Input capacitance compensation—Sallen-Key only

If the common-mode input capacitance of the op amp used in a Sallen-Key filter section is more than approximately C1/400 (0.25% of C1), it must be considered for accurate filter response. You can use the capacitor entry fields to compensate for op amp input capacitance by simply adding the value of the op amp common-mode input capacitance to the actual value of C1. The program then automatically recalculates the exact or closest 1% resistor values for accurate filter response. No compensation for op amp input capacitance is required with MFB designs.

Capacitor selection

Capacitor selection is very important for a high-performance filter. Capacitor behavior can vary significantly from ideal, introducing series resistance and inductance, which limit Q. Also, nonlinearity of capacitance versus voltage causes distortion.

Common ceramic capacitors with high dielectric constants, such as “high-K” types, can cause errors in filter circuits. Recommended capacitor types are: NPO ceramic; silver mica; metallized polycarbonate; and, for temperatures up to 85°C, polypropylene or polystyrene.

Using the f_n and Q displays

To aid in selection of the op amp, a feature in FilterPro allows you to view pole-pair sections f_n and Q. The f_n and Q information is also useful when troubleshooting filters by comparing expected to actual responses of individual filter sections.

Op amp selection

It is important to choose an op amp that can provide the necessary dc precision, noise, distortion, and speed.

Op amp bandwidth

In a low-pass filter section, maximum gain peaking is very nearly equal to Q at f_n (the section’s natural frequency). So, as a rule of thumb:

- *For an MFB section:* Op amp bandwidth should be at least $100 \times \text{Gain} \times Qf_n$.
- *For high-Q Sallen-Key sections:* A higher op amp bandwidth is required.
- *For a Sallen-Key section:* For Q > 1, op amp gain-bandwidth should be at least $100 \times \text{Gain} \times Q^3f_n$. For $Q \leq 1$, op amp gain-bandwidth should be at least $100 \times \text{Gain} \times f_n$.
- *For a real-pole section:* Op amp bandwidth should be at least $50f_n$.

Although Q is formally defined only for complex poles, it is convenient to use a Q of 0.5 for calculating the op amp gain required in a real-pole section.

For example, a unity-gain, 20-kHz, 5-pole, 3-dB ripple Chebyshev MFB filter with a second pole-pair f_n of 19.35 kHz and a Q of 8.82 needs an op amp with a unity-gain bandwidth of at least 17 MHz. On the other hand, a 5-pole Butterworth MFB filter with a worst-case Q of 1.62 needs only a 3.2-MHz op amp. The same 5-pole Butterworth filter implemented with a Sallen-Key topology would require an 8.5-MHz op amp in the high-Q section.

Op amp slew rate

For adequate full-power response, the slew rate of the op amp must be greater than $\pi V_{O\ p-p} \times$ Filter Bandwidth. For example, a 100-kHz filter with 20-V_{p-p} output requires an op amp slew rate of at least 6.3 V/ms. Texas Instruments offers an excellent selection of op amps that can be used for high-performance active filters. The Web site mentioned earlier (analog.ti.com) can help you select an appropriate op amp for your application.

Full-power bandwidth

The full-power bandwidth parameter of the op amp should be at least the maximum signal frequency to be passed.

Conclusion

Using FilterPro for Windows, a designer can quickly and accurately develop low-pass filters for many different applications without the need for complex calculations.

Reference

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. "FilterPro™ MFB and Sallen-Key Low-Pass Filter Design Program," Application Report	. . .sbfa001

Related Web site

analog.ti.com

Index of Articles

Data Acquisition

Title	Issue	Page
Aspects of data acquisition system design	August 1999	1
Low-power data acquisition sub-system using the TI TLV1572	August 1999	4
Evaluating operational amplifiers as input amplifiers for A-to-D converters	August 1999	7
Precision voltage references	November 1999	1
Techniques for sampling high-speed graphics with lower-speed A/D converters	November 1999	5
A methodology of interfacing serial A-to-D converters to DSPs	February 2000	1
The operation of the SAR-ADC based on charge redistribution	February 2000	10
The design and performance of a precision voltage reference circuit for 14-bit and 16-bit A-to-D and D-to-A converters	May 2000	1
Introduction to phase-locked loop system modeling	May 2000	5
New DSP development environment includes data converter plug-ins (PDF - 86 Kb)	August 2000	1
Higher data throughput for DSP analog-to-digital converters (PDF - 94 Kb)	August 2000	5
Efficiently interfacing serial data converters to high-speed DSPs (PDF - 80 Kb)	August 2000	10
Smallest DSP-compatible ADC provides simplest DSP interface (PDF - 120 Kb)	November 2000	1
Hardware auto-identification and software auto-configuration for the TLV320AIC10 DSP Codec — a “plug-and-play” algorithm (PDF - 105 Kb)	November 2000	8
Using quad and octal ADCs in SPI mode (PDF - 94 Kb)	November 2000	15
Building a simple data acquisition system using the TMS320C31 DSP (PDF - 235 Kb)	February 2001	1
Using SPI synchronous communication with data converters — interfacing the MSP430F149 and TLV5616 (PDF - 182 Kb)	February 2001	7
A/D and D/A conversion of PC graphics and component video signals, Part 1: Hardware (PDF - 191 Kb)	February 2001	11
A/D and D/A conversion of PC graphics and component video signals, Part 2: Software and control	July 2001	5
Intelligent sensor system maximizes battery life: Interfacing the MSP430F123 Flash MCU, ADS7822, and TPS60311	First Quarter, 2002	5
SHDSL AFE1230 application	Second Quarter, 2002	5
Synchronizing non-FIFO variations of the THS1206	Second Quarter, 2002	12
Adjusting the A/D voltage reference to provide gain	Third Quarter, 2002	5
MSC1210 debugging strategies for high-precision smart sensors	Third Quarter, 2002	7
Using direct data transfer to maximize data acquisition throughput	Third Quarter, 2002	14

Power Management

Title	Issue	Page
Stability analysis of low-dropout linear regulators with a PMOS pass element	August 1999	10
Extended output voltage adjustment (0 V to 3.5 V) using the TI TPS5210	August 1999	13
Migrating from the TI TL770x to the TI TLC770x	August 1999	14
TI TPS5602 for powering TI’s DSP	November 1999	8
Synchronous buck regulator design using the TI TPS5211 high-frequency hysteretic controller	November 1999	10
Understanding the stable range of equivalent series resistance of an LDO regulator	November 1999	14
Power supply solutions for TI DSPs using synchronous buck converters	February 2000	12
Powering Celeron-type microprocessors using TI’s TPS5210 and TPS5211 controllers	February 2000	20
Simple design of an ultra-low-ripple DC/DC boost converter with TPS60100 charge pump	May 2000	11
Low-cost, minimum-size solution for powering future-generation Celeron™-type processors with peak currents up to 26 A	May 2000	14
Advantages of using PMOS-type low-dropout linear regulators in battery applications (PDF - 216 Kb)	August 2000	16
Optimal output filter design for microprocessor or DSP power supply (PDF - 748 Kb)	August 2000	22

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 55 32
Finland (English)	+358 (0) 9 25173948
France	+33 (0) 1 30 70 11 64
Germany	+49 (0) 8161 80 33 11
Israel (English)	1800 949 0107
Italy	800 79 11 37
Netherlands (English)	+31 (0) 546 87 95 45
Spain	+34 902 35 40 28
Sweden (English)	+46 (0) 8587 555 22
United Kingdom	+44 (0) 1604 66 33 99
Fax	+(49) (0) 8161 80 2045
Email	epic@ti.com
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
Australia	1-800-999-084
China	108-00-886-0015
Hong Kong	800-96-5941
Indonesia	001-803-8861-1006
Korea	080-551-2804
Malaysia	1-800-80-3973
New Zealand	0800-446-934
Philippines	1-800-765-7404
Singapore	800-886-1028
Taiwan	0800-006800
Thailand	001-800-886-0010
Fax	886-2-2378-6808
Email	tiasia@ti.com
Internet	support.ti.com/sc/pic/asia.htm

A070802

SLYT039