MicroConverter Product FAQs

MicroConverter GENERAL:

Q: I have some **fast logic edges** on some of the digital circuitry I plan to connect to the MicroConverter. Will these signals impact the analog performance of the chip?

A: Signals with rise and fall times of less than 5ns or so, when applied directly to a MicroConverter's digital input pins, can potentially feed through and degrade analog performance. A simple solution to this problem is found in the form of a series resistor. A series resistor of around 200Ω will slow an edge sufficiently that it won't affect the MicroConverter's analog performance.

Q: When using the on-chip **watchdog timer**, the datasheet says I must set both **refresh bits** to reset the timer. Do I need to set the two bits in any particular order? Do I need to clear each bit to zero after setting it to one?

A: To refresh the watchdog timer, you need only to *set* the two bits, WDR1 & WDR2. A read from these bits will always return zeros. The order in which you set the bits must be WDR1 first, immediately followed by WDR2. For an example of this, look at the code 'WDtimer.asm' in the example code directory at www.analog.com/microconverter.

Q: How do I use the on-chip **temperature sensor** feature of the MicroConverter?

A: The temperature sensor on each MicroConverter outputs a voltage which is *inversely* proportional to chip temperature. At 25°C, this voltage is approximately 600mV. As temperature changes, the voltage changes at a rate of –3mV/°C. So as operating temperature *rises*, the temperature sensor output voltage *decreases*. And of course the ADC is used to convert the voltage on the temperature sensor to a digital value.

Look for a tech note to appear on the MicroConverter home page (www.analog.com/microconverter) that describes how to obtain the best possible absolute accuracy from the MicroConverter temperature sensor. And remember, the on-chip temperature sensor is measuring *chip* temperature, not ambient temperature, so don't forget to factor self-heating into the equation if you wish to determine the temperature of the ambient air.

Q: Do I have to enter "NOP" instructions to wait for the **Flash/EE erase and program times** to elapse before performing the next data Flash/EE operation?

A: Absolutely not. All timing for access to the data Flash/EE space is taken care of for you in hardware. When you perform a Flash/EE *erase* or *program* command, the MCU core will not move on to the next machine cycle until the Flash/EE operation is complete. Effectively this means that a Flash/EE erase or program command will only take one machine cycle, but that this machine cycle is stretched out over a 20ms period (for a Flash/EE *erase* command) or a 250µs period (for a Flash/EE *program* command) rather than taking only a microsecond or so for a normal machine cycle (depending on master clock frequency).

Q: What are the **ETIM registers** used for?

A: Flash/EE erase and program timing is derived from the master clock. When using a master clock frequency of 11.0592MHz, you needn't write to the ETIM registers at all. However, when operating at other master clock frequencies (F_{CLK}), you must change the values of ETIM1 and ETIM2 to avoid degrading data Flash/EE endurance and retention. ETIM1 and ETIM2 form a 16bit word, ETIM2 being the high byte and ETIM1 the low byte. The value of this 16bit word must be set as follows to ensure optimum data Flash/EE endurance and retention.

ETIM2,ETIM1 = 100µs•F_{CLK}

ETIM3 should always remain at its default value of 201dec / C9 hex.

Q: What's this "**power-on configuration routine**" that I've heard about?

A: Every MicroConverter product features a "power-on configuration routine" that runs every time you apply power to the chip or reset it. Basically, this is a small piece of code that is executed prior to the execution of your code. It is used to configure some of the on-chip peripherals such as the ADC and Flash/EE memory with factory optimized calibration and timing parameters. Some of these you can see (for example, default ADC offset and gain calibration registers will be different from one chip to the next) and others you can't (for example, ADC linearity coefficients are not visible to your code). If you were to skip the power-on configuration routine (there are ways to do this) then ADC accuracy and Flash/EE endurance could be sub-optimal.

The power-on configuration routine is stored in a hidden area of program ROM. The address where the routine begins is FF00 hex. Therefore, when you "soft-launch" your code from the serial downloader or debugger applications, be sure to start from address FF00 rather than 0000.

Although the power-on configuration routine is "mapped" to addresses FF00 through FFFF hex, this does not interfere with external code memory which shares these addresses. Up to 64K bytes of *your* code can be executed from an external PROM chip (or 8K from internal Flash/EE and 54K from external PROM). The power-on configuration routine shares the same *address* as the top 256 bytes of external PROM, but is only executed when power-on or reset events occur.

To ensure adequate power-on timing in your applications, the ALE output is automatically disabled during the execution of the power-on configuration routine. ALE only begins toggling when the first line of *your* code is executed.

MicorConverter DACs:

Q: When a MicroConverter DAC is disabled, what does it's output look like? Is there a **DAC "tri-state"** feature?

A: MicroConverter DACs default to a disabled state when the chip is powered up. In this state, the output appears as a high impedance node to the rest of the world. This is analogous to a "tri-state" type output in digital logic terms. You can place the DAC output into this high impedance state whenever you like simply by disabling the DAC again.

Q: The MicroConverter DACs have excellent DNL performance, but how can I get **better absolute accuracy** out of a DAC output?

A: You can improve absolute accuracy (INL, offset, and gain errors) of a MicroConverter DAC by feeding the DAC output back to an ADC input, and using a software routine to close the loop. In this configuration, you're basically using the DAC output only as a servo drive to increase or decrease the voltage until you read the correct voltage via the ADC. Absolute accuracy of the voltage generated is then equal to the ADC absolute accuracy, as defined by its offset error, gain error, and INL specifications.

Q: Some MicroConverters have two DACs. How can I **update the DACs simultaneously** to the new voltage?

A: MicroConverters with two DACs feature a *synchronous update* bit (in DACCON) which can be used to ensure that both DAC outputs step to their new values at the same instant. To use this feature, clear the bit to 0

before reloading new values to both DAC registers. Once new data has been loaded into both DAC registers, set the bit to 1 again. Both DAC outputs will then simultaneously step to their new values.

ADuC812 GENERAL:

Q: Port 1 on the ADuC812 is primarily used for analog inputs. Can I use **Port 1 pins for digital I/0**? If so, how?

A: Port 1 pins can be used as analog inputs or as digital inputs, but not as digital outputs. The default configuration for all Port 1 pins is as analog inputs. In this state, the Port 1 register contains FFhex (all ones). To configure any Port 1 pin as a *digital* input, simply clear the corresponding bit to zero in the Port 1 register (P1).

Q: Does the ADuC812 feature a static MCU core? **How *slow* can I run the master clock?**

A: Yes, the ADuC812 feature a static 8051 core. The core will operate at master clock frequencies right down to DC. However, the ADC clock is derived from the master clock, and ADC performance will degrade at very slow clock speeds. The minimum ADC clock frequency for reliable ADC performance is 400KHz. The ADC clock is equal to the master clock divided by 1, 2, 4, or 8 (selected by ADCCON1), so the minimum *master* clock frequency for reliable ADC operation is also 400KHz.

Q: **How *fast* can I run the master clock?**

A: The standard 8051 based MicroConverter core is capable of running at master clock speeds much greater than the specified maximum of 16MHz without any errors. However, running the ADuC812 master clock very much faster than this specification can degrade analog performance. Therefore, we do not recommend the use of crystal or clock oscillator frequencies of greater than 16MHz.

Q: Will I need to implement an external **reset generator** circuit for my ADuC812 design? If so, can I use a simple RC circuit, or must the circuit include a Schmidt trigger or some other active circuitry?

A: The ADuC812 features an internal power-on reset generator. However, if your power supply rise times are longer than 100ms, or if there are conditions where power could be removed and re-applied within less than 1 second, then you must implement an external power-on reset generator. The simplest way to do this is to use a small (SOT23) reset generator chip (eg. ADM810). If you prefer an RC circuit, you must use a Schmidt trigger to prevent multiple resets during power cycling.

Q: Somebody once told me that the ADuC812 featured **security bits** to restrict access to Flash/EE code and data space. Why can't I find any reference to that feature in the ADuC812 data sheet?

A: Those security bits exist in the ADuC812, but they don't function correctly. They are therefore not offered as a feature of the chip. They exist at page 160 (A0hex) of the Flash/EE data space, whereas the documented space runs from pages 0 thru 159 (00 – 9F hex). Writing to the undocumented page 160 should be avoided since it can result in accidentally locking the chip. See the ADuC812 errata sheet for details.

Q: I would like to serially download new code to the MicroConverter using my (____) host machine. What is the **serial download protocol** used to reprogram the MicroConverter?

A: MicroConverters can be serially reprogrammed in your system using Analog Devices' DOS/Windows based 'DownLoad' program to communicate through the chip's UART. Alternatively, you can in-system-reprogram

the MicroConverter from any other host processor using the same serial download protocol used by the 'DownLoad' application. Simply follow the below steps:

ADuC812 SERIAL DOWNLOAD:

1. To enable the serial download mode, power the chip up (or reset it) with the PSEN pin pulled low through a 1K resistor.

2. Once in this mode, the micro's UART will immediately transmit "ADuC812 krl" at a baudrate of 9600 (based on 11.0592MHz master clock). This is a single transmission and can be used by the host to verify that the micro is in serial download mode.

3. The micro will then wait to receive an Intel standard hex file through its UART. It receives this file on a per record basis, recognizing each record by a colon start character ":" (ASCII value 3Ahex).

4. At the end of each record, the micro will transmit an ACK (everything OK) or an NACK (everything not OK). ACK is 06hex and NACK is 15hex, i.e. industry standard UART ACK and NACK. When the host receives an ACK, it should move on to the next record (beginning with a ":"). If the host receives an NACK, it can abort the current download and restart it by sending a 'soft reset' to the micro. This is accomplished simply by sending an exclamation point "!" (ASCII value 21hex) to the micro. Assuming the micro is still in serial download mode, it will respond with the "ADuC812 krl" string as in step number 2 above, and the host can begin sending again from beginning of the hex file.

5. Once the entire hex file has been sent, the program can be launched by toggling reset or power cycling the chip, *without* the pulldown on PSEN.

6. Alternatively, the host can force the code to execute by sending a start address. This is done by sending a semicolon character ";" (ASCII value 3Bhex) followed by a 4 digit hexadecimal start address. in most cases, this start address should be "FF00" in order to execute the power-on configuration code. among other things, this code downloads the factory calibration coefficients. It then wraps around to begin executing user code at address 0000hex.

Q: The ADuC812 has separate pins for **analog and digital grounds**. Should I connect these to two separate ground planes on my board?

A: No. At least, not unless the two ground planes are connected together very near the chip. In all other situations, it is best to keep the ADuC812 on a single ground plane. If you have two separate ground planes on your board, then place the ADuC812 on the quieter of the two (usually the analog plane) to obtain optimum ADC and DAC performance.

Q: The ADuC812 has separate pins for **analog and digital power supplies**. Can I run from a split supply, for example a 3.3V DV_{DD} and a 5V AV_{DD}? What other implications should I be aware of about the supplies?

A: No. You cannot run the ADuC812 from split supplies. The absolute maximum rating for the difference between DV_{DD} and AV_{DD} is ±0.3V. For this reason, it is best to run the ADuC812 from a single supply. The most that should separate these supplies (and only if necessary) is a small value ferrite bead (0805 chip size) or a small value (1 or 2 Ω) resistor. Of course, decouple each supply pin to the ground plane with a 0.1µF chip cap connected with short trace lengths, and make sure there are larger electrolytic reservoir capacitors decoupling the supply to ground somewhere on the board. If you do choose to split the supplies with a ferrite bead or a small resistor, make sure there are local chip caps *and* an electrolytic on *each* side of the split.

Q: How do I calculate the **total power consumed** by the ADuC812 under a given set of conditions?

A: Below is a table to allow simple calculation of the ADuC812's total current draw. All of the below are typical values. Simply add the current draw of all enabled peripherals to the core (normal mode) current consumption at your given master clock frequency (M_{CLK}).

	V_{DD}=5V	V_{DD}=3V
core (normal mode):	1.6µs•M_{CLK}+5mA	0.8µs•M_{CLK}+1.5mA
core (idle mode):	0.75µs•M_{CLK}+5mA	0.25µs•M_{CLK}+1.5mA
ADC:	1.3mA	1.0mA
DAC:	250µA	200µA
voltage reference:	200µA	150µA

ADuC812 VOLTAGE REFERENCE:

Q: I'm using the ADuC812's **internal voltage reference**. What should I do with the V_{REF} and C_{REF} pins?

A: Decouple both to ground using 0.1µF chip capacitors with short trace lengths.

Q: I would like to **drive other circuitry with the internal voltage reference** of the ADuC812. Should I take this voltage from the V_{REF} pin or the C_{REF} pin? What are the current source/sink capabilities of these pins?

A: Use the V_{REF} pin. The C_{REF} pin is an internal node within the buffer. It's voltage won't be equal to V_{REF}. With regard to the ability of the V_{REF} pin to sink and source current, it really has none. It is effectively 2.5V with a nominally 50KΩ source impedance. You must buffer the voltage on this pin if you wish to use the voltage to drive other circuitry. Of course, decouple the V_{REF} and C_{REF} pins with 0.1µF chip capacitors to ground just as you would normally.

Q: Can I use an **external voltage reference** with the ADuC812?

A: Yes. Simply drive the V_{REF} pin with your external voltage reference. This will look like a 50KΩ load to 2.5V. Depending on the voltage reference you choose, you may or may not want to capacitively decouple the V_{REF} node. The C_{REF} pin, however, must be decoupled to ground using a 0.1µF chip capacitor as in any other configuration.

Q: How do I **enable and disable the internal voltage reference** of the ADuC812? How long after enabling the internal voltage reference should I allow for **reference voltage settling time**?

A: The ADuC812's internal voltage reference is automatically enabled whenever you enable the ADC (via ADCCON1) or either DAC (via DACCON). Once enabled, the voltage reference will require 65ms to turn on and settle to an accurate value.

ADuC812 ADC:

Q: Can I connect a **high impedance analog source** directly to one of the ADuC812's inputs? Or must I buffer the signal first?

A: The main limitation with high impedance sources is the input leakage current of the ADuC812's analog inputs, which is typically ±1µA. This current flowing through a source impedance of 610Ω will generate 610µV of error. With a 2.5V reference, this is 1LSB (or 2.5V/4096). Therefore, a source impedance of greater than 610Ω can potentially generate measurable DC errors.

Q: Some feature lists talk about the "**self calibration**" capabilities of the ADuC812. How do I make use of this feature?

A: "Self calibration" of the ADuC812 involves the use of a software routine soon to appear in a tech note on the MicroConverter home page (www.analog.com/microconverter).

Q: What **acquisition time** is required by the ADuC812? How do I choose the proper values for the acquisition time select bits in ADCCON1?

A: In nearly all cases, an acquisition time of 1 ADC clock (ADCCON1.2=0, ADCCON1.3=0) will provide plenty of time for the ADuC812 to acquire its signal before switching the internal track&hold amplifier in to hold mode. The only exception would be a high source impedance analog input, but these should be buffered first anyway since source impedances of greater than 610Ω can cause DC errors as well.

Q: How do I choose the **conversion time** in the ADuC812's ADCCON1 register? And what are these **ADC clock divide bits** anyway?

A: The ADuC812's successive approximation ADC is driven by a divided down version of the master clock. To ensure adequate ADC operation, this ADC clock must be between 400KHz and 4MHz., and optimum performance is obtained with ADC clock between 400KHz and 3MHz. Frequencies within this range can easily be achieved with master clock frequencies from 400KHz to well above 16MHz with the four ADC clock divide ratios to choose from. For example, with a 12MHz master clock, set the ADC clock divide ratio to 4 (i.e. $ADC_{CLK} = M_{CLK} / 4 = 3MHz$) by setting the appropriate bits in ADCCON1 (ADCCON1.5=1, ADCCON1.4=0).

The total ADC conversion time is 15 ADC clocks, plus 1 ADC clock for synchronization, plus the selected acquisition time (1, 2, 3, or 4 ADC clocks). For the example above, with a 1 clock acquisition time, total conversion time is 17 ADC clocks (or 5.67µs for a 3MHz ADC clock).

Q: How do I determine the ADuC812's **sample rate in continuous conversion mode**?

A: In continuous conversion mode, a new conversion begins each time the previous one finishes. The sample rate is then simply the inverse of the total conversion time described above. In the example above, the continuous conversion mode sample rate would be 176.5KHz.

Q: What is the ADuC812's **aperture delay** in hardware CONVST mode? And what **aperture uncertainty** can I expect?

A: In hardware CONVST mode an external logic input is used to trigger ADC conversions. The *aperture delay* of the ADuC812 is the time from the rising edge of that external trigger to the moment when the sample & hold amplifier goes into hold mode. This time is equal to the acquisition time (selected via ADCCON1) plus a synchronization time of between 0.5 and 1.5 ADC clock periods.

When the CONVST trigger is asynchronous to the ADC clock, this results in an *aperture uncertainty* of 1 ADC clock period. This aperture uncertainty can be avoided by synchronizing the external CONVST signal with the ADC clock. Since the ADC clock is simply a divided down master clock it is not available to you directly. Therefore, to synchronize your CONVST signal with the ADC clock, you must synchronize it with a *divided* master clock, where the divide ratio is a direct multiple of the divide ratio used to generate the ADC clock (selected in ADCCON1).

Q: What happens if the ADuC812 receives a **second CONVST trigger** (software SCONV, Timer2 trigger, or hardware CONVST trigger) during a conversion that it hasn't yet completed?

A: The second CONVST trigger will be ignored. In this situation, the second trigger event is lost. To avoid this loss of samples, make sure your software checks the state of the ADC busy flag (ADCCON3.7) before starting a conversion. In timer driven or hardware driven conversion modes, make sure the timer overflow rate or the input edge rate is more that the ADC conversion time plus acquisition time (controlled by ADCCON1).