ADuC816 FAQs

MICROCONVERTER GENERIC

LAYOUT AND DESIGN CONSIDERATIONS

Q: The MicroConverter 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 MicroConverter on a single ground plane. If you have two separate ground planes on your board, then place the MicroConverter on the quieter of the two (usually the analog plane) to obtain optimum ADC and DAC performance.

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.

MISCELLANEOUS

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: I heard that the MicroConverter can address up to **16Mbytes of external data memory**. Is this true and how is it done?

A: Yes, the MicroConverter can address up to 16Mbytes of external data memory. If access to more than 64 Kbytes of RAM is desired, a feature unique to the MicroConverter allows addressing up to 16 Mbytes of external RAM simply by adding an additional latch on the Port 2 address bus.
On a standard 8051 as with the MicroConverter Port 0 (P0) serves as a multiplexed address/data bus. It emits the low byte of the data pointer (DPL) as an address, which is latched by a pulse of ALE prior to data being placed on the bus by the MicroConverter (write operation) or the SRAM (read operation). Port 2 (P2) provides the data pointer page byte (DPP) to be latched by ALE, followed by the data pointer high byte (DPH). If no latch is connected to P2, DPP is ignored by the SRAM, and the 8051 standard of 64kByte external data memory access is maintained.

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' Windows based download program (WSD.exe) 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 WSD application. Refer to tech note uC004 available on the web or as part of the QuickStart™ Development System.

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, PTAT coefficients are not visible to your code). For the MicroConverter devices that run off a 32kHz crystal the power on configuration will also wait for the part to 'lock' before it starts to run user code.
The power-on configuration routine is stored in a hidden area of program ROM. The address where the routine begins is FF00 hex. 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 user code can be executed from an external PROM chip (or 8K from internal Flash/EE and 56K from external PROM).
The ALE output is automatically disabled during the execution of the power-on configuration routine so that you can tell when your code starts to execute. ALE only begins toggling when the first line of *your* code is executed.

Q: The *PSEN* pin is always an output on an 8051. How then can it be used as an input to **enter serial download mode?** What pulldown resistor should be used to be sure to enter serial download mode? Is the standard functionality of the *PSEN* pin affected?

A: On an 8051 the *PSEN* pin is always an output. It is used to access from external code memory. On the MicroConverter the *PSEN* pin is used to access external code memory and is used as an input to determine whether to run user code or run the on chip serial downloader.
The *PSEN* pin is normally configured as an output. When the RESET pin is pulled high, the *PSEN* pin becomes a digital input and the voltage at the *PSEN* pin is sampled every machine cycle. Once the RESET pin is released (low) the last sampled logic level of the *PSEN* pin is used to decide whether to run from user code or whether to run the on chip serial downloader.
To ensure that the *PSEN* pin is interpreted as a logic low it should be pulled low through a 1k resistor to ground. To ensure that the *PSEN* pin is interpreted as a logic high it can be left float as there is an internal pullup resistor.
Because the functionality of the *PSEN* pin on the MicroConverter only differs from that of an 8051 when RESET is pulled high (in this condition the *PSEN* pin on an 8051 is useless) the functionality of the *PSEN* pin on the MicroConverter is unaffected by its additional functionality.

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/250µs period for a Flash/EE *erase/program* command on the ADuC812 or over a 2ms/250µs period for a Flash/EE *erase/program* command for the ADuC816/ADuC824.

Q: What happens if the power supply falls during a **program or erase of the Flash/EE data memory?**

A: If the power supply falls below 2.7V during a Flash/EE program or erase instruction then the core cannot guarantee that the instruction will execute correctly.
Also, because the Flash/EE program or erase instruction takes much longer than a typical instruction (250us for a program and 20ms/2ms (ADuC812/(ADuC816 or ADuC824)) for an erase), the time required to respond to interrupts can be greatly increased if the interrupt occurs during a flash/EE program or erase instruction.
For instance, if a power-supply-monitor (PSM) interrupt occurs during a Flash/EE program or erase instruction, it will only be processed after the instruction is complete. In this way, the PSM can be used to indicate when power has dropped below a specified threshold during a Flash/EE program or erase instruction, thereby indicating that the instruction may not have executed correctly.

Q: Are there any **sockets available for the 52PQFP package**?

A: Yes, but only test or evaluation type sockets. They aren't suitable for production since they are ZIF types that cost many times the price of the MicroConverter products. They can, however, prove useful for test/programming heads or in some cases for debug/prototyping setups. A couple of different types include the below....

Enplas:

OTQ-52-0.65-01:	"open-top" type
FPQ-52-0.65-01A:	"clam-shell" type

(sockets mount on through-hole pattern, useful for test/programming heads.)

Ironwood Electronics:

CA-QFE52SB-L-Z-T-01:	socket module
SF-QFE52SB-L-01:	board header

(board header solders to 52PQFP footprint, and socket adapter plugs into it to accept MicroConverter package. also provides test point connections to all 52 pins. useful for debug/prototyping setups.)

ADuC816 Specific

ADCs General

Q: The datasheet says the ADuC816's ADCs are calibrated in the factory, will I ever need to **calibrate the ADCs**? How long does a calibration cycle take?

A: At power-on, factory calibration coefficients (for both the primary and auxiliary ADC) are automatically downloaded to the calibration registers in the ADuC816 SFR space. These calibration registers will be overwritten if any of the four calibrations are initiated by writing to the mode bits in ADCMODE. Both ADCs are chopped meaning that an internal offset calibration should never be required. Also, because factory 5V/25°C gain calibration coefficients are automatically present at power-on, an internal full-scale calibration will only be required if the part is being operated at 3V or at temperatures significantly different from 25°C.
During an ADC calibration the ADC automatically converts on the slowest update rate (5Hz). Since two ADC conversions are required (ADC input chopping is used to minimize system offset) each calibration takes 0.374s.

Q: I want to overwrite the **ADC factory calibration coefficients.** Do I have to do a calibration every time I run my software.

A: Every time the ADuC816 is reset, the factory calibrated calibration coefficients are downloaded to the calibration registers in the SFR space. Hence these SFRs must be overwritten with your individual calibration coefficients. To avoid having to do a calibration every time that the ADuC816 powers up it is often easier to write the calibration coefficients to the Flash/EE data memory space. Then in your own initialization the coefficients can simply be read from the flash/EE data memory and the calibration process does not have to be repeated.

Q: Where's the **ADC DMA mode** gone, I see no reference to it in the ADuC816 datasheet?

A: Since for the ADuC816 the ADC conversion time is so long in comparison to the time it takes to write an ADC conversion result into external data memory the ADC DMA mode is not of any significant advantage and was therefore removed for the ADuC816.

Q: What's the **difference between the INL spec and the noise spec** on the ADC?

A: Any given ADC conversion result has both an INL error and a noise error. The INL specification on the ADuC816 gives accuracy to 1 LSB). Depending on the range and the update rate this ADC conversion result will have a varying amount of noise on it.
The noise which exists on this ADC conversion result is random and is equally as likely to be positive as negative. Hence by doing more averaging this noise error can be reduced. Since the INL will stay constant for every ADC conversion (given the same DC input) then software averaging will not remove the INL error.

ADuC816 PRIMARY ADC:

Q: The **primary ADC is differential**. Can I convert on a single-ended input voltage by grounding one of the inputs.

A: No, the ADC inputs for the primary ADC cannot be within 100mV of the rails. Hence one of the primary ADC inputs cannot be grounded. To convert on a single-ended input voltage either bias the signal up to a voltage so that the input signal is never within 100mV of the rails or use the Auxiliary ADC which features single-ended inputs.

Q: The **ADC noise tables** on page 33 of the ADuC816 datasheet show that the **effective resolution depends on the range and update rate** selected. Why?

A: The ADuC816 features Sigma Delta ADCs. Sigma Delta ADCs use a method of averaging (using a sinc^3 filter) to obtain the very high accuracy ADC conversion results. The more averaging that is done the more accurate the ADC conversion is going to be. With high update rates the ADC gets to perform less averaging in hardware on the chip. Hence the resolution will not be as good. With slower update rates more averaging is done on the chip and the resolution is improved.
The resolution of the ADC also depends on the input voltage range. Noise added to an input signal with the ADC converting on the 2.56V range will affect the conversion result much less (128 times less) than the same amount of noise being added to a signal converting on the 20mV range. Hence converting on a larger signal will produce an ADC result accurate to many more bits.

Q: Is it possible to **switch OFF the internal buffer** of primary ADC? Some of the AD77xx family have this particular feature on board. Can I **turn chopping off** like on AD77xx parts so as to increase throughput time?

A: No, it is not possible to turn off the internal buffer of the primary ADC. The auxiliary ADC provides an unbuffered input if required. It is also not possible to turn ADC chopping off on the ADuC816.

Q: I am changing ranges and I am seeing **much more than 2uV gain mis-match.** Why?

A: You should not see much more than 2uV gain matching. i.e. as you changes the range from one range to another the difference in an ADC results between one range and the next should typically correspond to less than 2uV.
However if you have performed a system offset calibration then you will have changed the offset calibration registers (OF0H/M). Since the ADCs use chopping there is no need for an offset calibrated in the factory. Hence the default offset calibration coefficients are 8000h. Once a system offset calibration is performed the OF0H/M get overwritten by values that when subtracted with a zero input will produce a zero reading. When the range is changed, however, the offset calibration registers still apply to the previous range.
e.g. if on the 20mV range a system offset of 10uV existed producing an offset calibration coefficient of 8021h. Now imagine that the range is changed to the 40mV range. The OF0L register still reads 8021h. This corresponds to an offset of 20uV on the 40mV range. On the 2.56V range the offset would correspond to 1.28mV.

There are two ways to prevent large offset gain errors from occurring when the ranges are changed. The first way is to calibrate the system offset at each of the ranges and store the different calibration results into flash/EE data memory. Then whenever the gain is changed the appropriate calibration coefficients can be downloaded from flash/EE data memory. The second way is to calibrate the offset at the 20mV range. Then whenever you change the range you should appropriately change the OF0H/M SFRs so that the difference between OF0H/M and 8000h is halved every time the range is increased one level as shown in the following expression:

$$\mathrm{OF0H/M}_{\mathrm{new\ range}} = 8000\mathrm{h} + (\mathrm{OF0H/M}_{\mathrm{20mV\ range}} - 8000\mathrm{h}) / 2^{\mathrm{RN}}$$

Q: I'm using the ADuC816 but now require **more than 16 bits of resolution**. For this reason I would like to use the **ADuC824**. Is this device fully compatible with the ADuC816?

A: Yes, the ADuC824 is a drop in replacement for the ADuC816 for those applications that require more resolution. The only hardware difference between the ADuC816 and the ADuC824 is that the ADuC824 has 24 bits of resolution in the primary ADC instead of the 16 bits of resolution of the ADuC816. All other peripherals are identical including the PLL circuit which means all timings of the ADuC816 and the ADuC824 will be identical.
The improved resolution means that the ADuC824 contains three more ADC SFRs, ADC0L, OF0L and GN0L. Because, all the SFRs of the ADuC816 are located in the same locations for the ADuC816 and the ADuC824 (the three extra ADuC824 SFRs use Reserved locations on the ADuC816) then software on the ADuC816 will be 100% compatible with that on the ADuC824.

ADUC816 AUXILIARY ADC:

Q: Why is there a second ADC on the ADuC816 and **what is the Auxiliary ADC used for**?

A: Since some applications require the use of a single ended signal, the Auxiliary ADC is added to the ADuC816. Often this is used for temperature compensation, to measure the battery voltage, or indeed to measure any single ended voltage.
Because the two ADCs are separate simultaneous ADC conversions can be done on both the primary and auxiliary ADCs. This means that for example the temperature can be measured at the same time as another signal and the ADC conversion does not have to be waited for.

Q: What are the implications of having **unbuffered Auxiliary ADC**?

A: Because the auxiliary ADC is unbuffered external circuitry placed at these inputs can affect the ADC conversion result. The average analog input current for the auxiliary ADC inputs (125nA/V) is much higher than that of the primary ADC. Also this unbuffered input path provides a dynamic load to the driving source. Therefore, resistor/capacitor combinations on the input pins can cause dc gain errors depending on the output impedance of the source that is driving the ADC inputs.

ADuC816 Voltage Reference

Q: What are the implications of having **unbuffered differential reference inputs**?

A: The unbuffered reference input provides a high impedance, dynamic load. Because the input impedance of each reference input is dynamic, resistor/capacitor combinations on these inputs can cause dc gain errors depending on the output impedance of the source that is driving the reference inputs. Reference voltage sources, like those recommended (AD780) will typically have low output impedances and therefore decoupling capacitors on the REFIN(+) and REFIN(-) inputs would be recommended. Deriving the reference input voltage across an external resistor, as in many ratiometric configurations, will mean that the reference input sees a significant external source impedance. External decoupling on the REFIN(+) and REFIN(-) pins would not be recommended in this type of circuit configuration.

Q: The ADuC816 features an internal voltage reference but I have been recommended to use an external voltage reference. Is this correct? **If so why is the internal reference present**?

A: Yes, we do recommend that an external voltage reference is used for the applications that require good ADC accuracy. Typically the internal voltage reference is only 13-14 bit accurate and will not be useful for most applications. The WASP (available as part of the QuickStart™ Development System) allows you to compare the noise performance of the ADuC816 with an internal and an external voltage reference.
The internal voltage reference is often used with the auxiliary ADC where the very high accuracy might not be required. Often the primary ADC is used with a ratiometric voltage reference which will be useless for another measurement. In this case the internal reference can be used for the second measurement. The internal voltage reference is also necessary for the internal temperature sensor.

Q: The ADC result when using the **internal reference** is totally different than when **using a 2.5V external reference**. Why?

A: The internal reference for the ADCs is 1.25V and not 2.5V (as with the ADuC812). This means that when converting on the particular voltage ranges as selected by ADC0CON that the ranges are actually half of those specified (they are specified assuming a 2.5V reference). The internal reference is not recommended for high accuracy ADC conversion. Instead a voltage reference such as the AD780 is recommended. Although the internal reference for the ADC is only 1.25V the internal reference for the DAC is still 2.5V

Q: What are the **minimum and maximum external voltage references** allowed by the ADuC816.

A: The minimum voltage reference allowed between REFIN+ and REFIN- is 1V. The maximum voltage reference is AV_{DD}.

Q: The **reference input is differential**. Can I use a **single ended reference**?

A: Yes, REFIN- can be grounded if a single ended reference is being used.

UART SERIAL PORT

Q: How do I **configure the ADuC816 UART serial port at 9600 baud**?

A: The UART serial port on the ADuC816 can be configured for 9600 baud using the following piece of code (assuming PLLCON=3):

```
MOV     RCAP2H,#0FFh   ; config UART for 9830baud
MOV     RCAP2L,#-5     ; (close enough to 9600baud)
MOV     TH2,#0FFh
MOV     TL2,#-5
MOV     SCON,#52h
MOV     T2CON,#34h
```

Note: if using a remote RUN command to run the code instead of a using a hard reset then the UART is automatically setup for 1200 baud. If you would like to reconfigure the UART serial port for a different baud rate then the following piece of code should be added to your code before you configure the UART.

```
MOV     SCON,#00h
MOV     T2CON,#00h
```

Q: Does changing the PLL divider value via the **CD bits in PLLCON affect the baud rate**?

A: Yes, changing the CD bits does affect the baud rate. The output of the PLL divider is the core clock frequency which the baud rates are obtained from. See the ADuC816 datasheet (pg 59 REV.A) for more details.

Q: What is the **highest baud rate** obtainable by the UART?

A: The highest baud rate obtainable by ADuC816 is 393kbaud. In terms of PC baud rates (integer divisions of 115200) the highest baud rate obtainable by the ADuC816 is 57600 baud. This is obtained by clearing the CD bits in PLLCON and using reload values of –1 and –7 in RCAP2H and RCAP2L respectively.

MISCELLANEOUS

Q: Will I need to implement an external **reset generator** circuit for my ADuC816 design

A: You must implement external POR (power-on reset) circuitry to drive the RESET pin of the ADuC816. Your circuit must hold the RESET pin asserted (high) whenever the power supply (AVDD & DVDD) is below 2.5V. Furthermore, VDD must remain above 2.5V for at least 10ms before the RESET signal is de-asserted (low). The external POR circuit must be operational down to 1.2 volts or less. See the ADuC816 datasheet (pg 63 REV.A) for more details.

Q: I want to **drive LEDs directly** from some digital outputs what pins should I use?

A: The high drive port pins are P1.0 and P1.1. Both of these port pins are tested to sink 10mA and keep the V_{OL} below 0.4V which is easily capable of sinking enough current for an LED.

Q: Port 1 on the ADuC816 is primarily used for analog inputs. Can I use **Port 1 pins for digital I/0**? If so, how?

A: Pins 1.2 to P1.7 inclusive are analog input/digital input pins. They have no digital output capabilities. The default configuration for these six Port 1 pins is as analog inputs. In the default state, the Port 1 register contains FFh. To configure any of these six Port 1 pins as a *digital* input, simply clear the corresponding bit to zero in the Port 1 register (P1).
P1.0 and P1.1 are digital I/O pins and have a pull-up configuration as described below for Port 3. Both P1.0 and P1.1 have an increased current sink capability of 10 mA.

Q: What **external clocks** can be used with the ADuC816? Can the **PLL be bypassed**?

A: The PLL circuit on the ADuC816 will lock to 384 times the input clock as long as the clock frequency is within the frequency range 32.768kHz ± 20%. The ADC incorporates 50Hz and 60Hz rejection filters which assume a 32.768kHz input frequency. Using an input clock that differs from 32.768kHz will cause these rejection filters to move.
Although it is possible to use the ADuC816 with an input clock other than 32.768kHz it is recommended that you use a 32.768kHz clock/crystal to preserve the ADC functionality.
There is no mode of operation on the ADuC816 that allows the PLL to be bypassed. Hence the ADuC816 cannot be driven directly with an oscillator other than 32.768kHz +/-20%.

Q: The ADuC816 features **security bits** to restrict access to Flash/EE code and data space. Can I accidentally write to them in my code and prevent the part from entering serial download mode?

A: The ADuC816 facilitates three modes of Flash/EE program memory security. These modes can be independently activated, giving different levels of security to your code as described in the ADuC816 datasheet (pg 39 REV.A)
The security bits exist at page 160 (A0hex) of the Flash/EE data space, whereas the documented space runs from pages 0 thru 159 (00 – 9F hex). This page is, however, locked out from user code making it impossible for your code to accidentally set the security bits. The security bits can be set after parallel programming the ADuC816 or directly after a serial download by following the serial download protocol as described in uC004 (ver 2.0). This protocol is implemented in our Windows serial downloader (WSD) available as part of the QuickStart development system.

Q: How do I use the on-chip **temperature sensor** feature of the ADuC816?

A: The temperature sensor on the ADuC816 is converted by the auxiliary ADC to output a digital code to represent the chip temperature. The auxiliary ADC must be configured in bipolar mode using the internal voltage reference in order for the temperature sensor to produce sensible results.
The temperature sensor is calibrated in the factory at 25°C so that the ADC conversion result is 8000h at 0°C with 256 counts per °C.
e.g. An ADC conversion result of 9680h represents a temperature of 22.5°C.
An ADC conversion result of 6756h represents a temperature of –24.66°C.
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: I have **run out of flash/EE program memory** as my program has grown too big, what are my options

A: The ADuC816 contains 8k of flash/EE program memory. This memory is expandable off chip up to 64kBytes of program memory. However Analog Devices is currently designing the ADuC816B2 which will contain 62kBytes of Flash/EE program memory on chip. The internal RAM and flash/EE data memory will also be expanded.

Q: I want to initiate the **serial downloader through the SPI port**, how can I do this?

A: It is not possible on the ADuC816 to write to the 8k of flash/EE program memory from the 8k space itself. It is only possible to write to this space from the on chip serial downloader. Hence it is only possible to serially download to the ADuC816B2 using the UART serial port.
However in the ADuC816B2 because of its larger flash/EE program memory it will be possible to serially download to the flash/EE program memory. To download to this space you will have to write your own serial downloader in the first 8k of the flash/EE program memory to download to the rest of the flash/EE program memory. The downloader can use any means to download, parallel, SPI, I2C, UART or any other protocol you wish.

Q: I just want to **initiate a sample conversion every 10 hours**, what is my **lowest power consumption** configuration in this mode?

A: The ADuC816 can be programmed into power down mode by setting PCON.1. This powers the whole chip down into a low power mode. The Time Interval Counter (TIC) on the ADuC816 is the only peripheral which can remain powered up while the chip is powered down. The TIC allows you to wake the chip up from powerdown mode every particular interval.
Therefore if you only wish to initiate a sample conversion every 10 hours the chip can be left in power down mode for the 10 hours. Once the 10 hours has elapsed the chip will wake itself up allowing you to perform an ADC conversion, wait for the result, and the go back to power down mode. In this mode the average current will typically be about 8uA for a 3V supply.

Q: Why does it take the ADuC816 **so long to power on**?

A: Before the user code executes, the ADuC816 waits for the crystal to oscillate. This can take anywhere from under 1ms to over 300ms. See the ADuC816 datasheet (pg 6 REV.A) for typical figures.
When reset, the ADuC816 must first execute a "power on configuration routine". The power on configuration routine first waits for the crystal to oscillate, and then for the PLL to lock to its frequency (about 1ms typically). Once the PLL has gained lock the rest of the power on configuration routine which downloads the calibration coefficients to particular SFRs is executed. This typically takes about 2ms. When the power on configuration is finished the ADuC816 will start to execute your code starting at address 0.

Q: The ADuC816 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: Yes, You may run the ADuC816 from split supplies. See the ADuC816 datasheet (pg 63 REV.A) for more details.