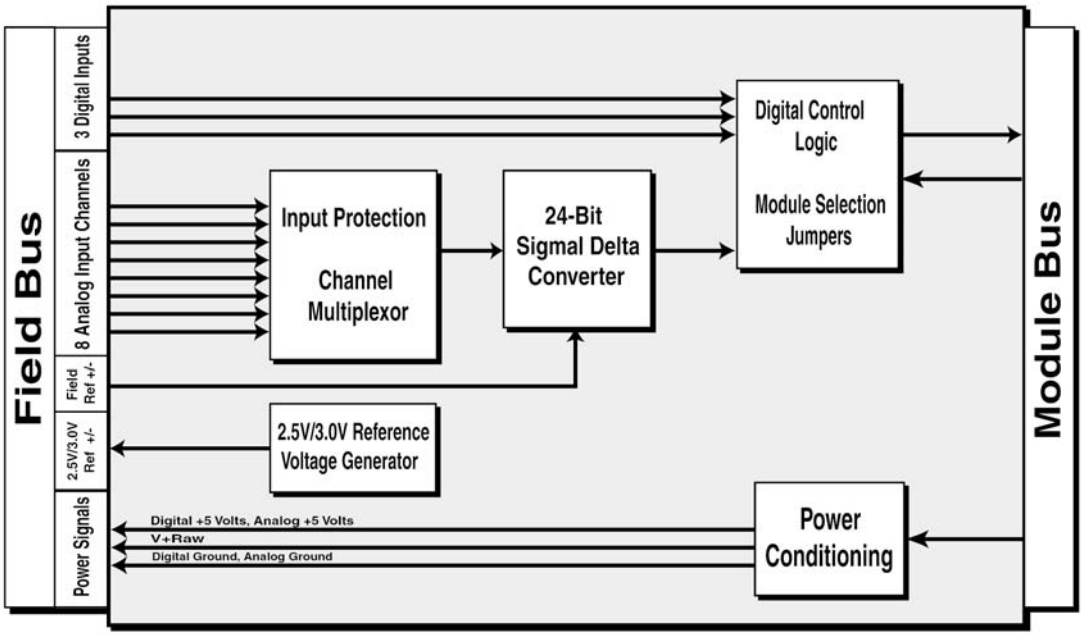
24/7 Data Acquisition Module Users Guide
Version 1.6



Block Diagram of the 24/7 Data Acquisition Module

September, 2002

Table of Contents

Introduction

The 24/7 Data Acquisition Module gives you the ability to amplify and sample low level signals at various frequencies directly from transducers using a sigma-delta technique. Its state-of-the-art analog to digital converter, Analog Devices' AD7714, provides 24 bits of resolution with no missing codes performance. The 24/7 Data Acquisition Module performs all signal conditioning and conversion for a system of up to seven input channels that are configurable in 10 different ways. Specifications are summarized in the following table.

24-7 Data Acquisition Wildcard Specifications	
Input Channels	4 fully differential or 7 pseudo-differential (ie, with a common signal rtn)
Resolution	24-bits (0 – 16,777,216 counts), sigma-delta conversion
Programmable Gain	Precisely 1, 2, 4, 8, 16, 32, 64 or 128, software selectable, corresponding to full scale input ranges of 20 mV to 2.5 V for unipolar conversions, and ± 20 mV to ± 2.5 V for differential conversions.
Input Voltage Range	Quasi-Bipolar Range: ± 2.5V with each input within -0.03 V to +5.03 V True Bipolar Range: ± 20 mV Common Mode: -30 mV to 5.03 V unbuffered, 50 mV to 3.5V buffered
Monotonicity	24 bits at up to 60 Hz data rate
Linearity	0.0015% FS at up to 60 Hz data rate
Noise Rejection	DC Common Mode: > 90 dB Normal Mode at 50/60 Hz: >100 dB Common Mode at 50/60 Hz: >150 dB
Noise and Accuracy	2.7 μV rms effective input noise, 21 bits max effective resolution
Sample Rate	Programmable to 1010 samples/sec
Digital Filter	Integral digital filter sets a programmable high frequency cutoff. Filter cut-off, resolution and conversion rate are mutually determined, for example as, Data Rate / Filter Cutoff / Effective Resolution 10 Hz / 2.6 Hz / 21.5 bits 30 Hz / 7.9 Hz / 20 100 Hz / 26.0 Hz / 18.5 500 Hz / 131.0 Hz / 13 Conversion rate can be chosen to place filter notches at noise harmonics. A data rate of 10 samples/sec excludes noise at 50 and 60 Hz and their harmonics. Filter settling time is <4 conversion periods.
Voltage Reference	2.5 V precision reference for use internally and externally 3.0 V precision power source available for sensor excitation
Input Protection	to ± 2000 V electrostatic discharge and ± 70 V continuous
Calibration Modes	Automated Full Scale and Zero Self Calibration and System Calibration

High-level software routines allow you to initialize, calibrate, configure, and control the 24/7 Data Acquisition Module. Once commanded to start converting, the analog to digital converter continually samples and converts at a fixed rate. You can read

the output once, or store each sequential conversion to a memory buffer. You must sample one channel at a time; changing channels requires restarting the conversion process.

This document describes, step by step, how to use the 24/7 Data Acquisition Module. It will show you how to:

- Power up and connect to the module;
- Use and configure the available hardware; and,
- Call the pre-coded software drivers to initialize, calibrate, configure, and obtain samples from the module.

The last sections of this document provide an example code listing, a glossary, and the complete schematics of the 24/7 Data Acquisition Module. If after reading this document you have further questions, please consult the data sheet for the AD7714 that is provided with the distribution diskette or located at http://www.analog.com.

Please take time to read through the example programs that illustrate how to effectively use the pre-coded software drivers. There are many subtleties involved with using the 24/7 Data Acquisition Module that are highlighted in the example code.

Connecting to the 24/7 Data Acquisition Module

To start using the 24/7 Data Acquisition Module, follow these simple steps:

1. Connect the Module Carrier Board to the QED Board as outlined in the "Module Carrier Board Users Guide".
2. With the power off, connect the 24/7 Data Acquisition Module to the Module Carrier Board by connecting the Module Bus of the 24/7 Data Acquisition Module (labeled Module Bus) to Module Port 0 of the Module Carrier Board (located below the QED Digital I/O header and to the right of Module Port 1). Caution: The 24/7 Data Acquisition Module and Module Carrier Board can be permanently damaged if the connection is done incorrectly.
3. Be sure the jumper shunt for J1 is not installed on the 24/7 Data Acquisition Module. With the jumper shunt not installed, the reference voltage for the analog to digital converter is set to 2.5 volts.
4. Be sure the jumper shunts for J2 and J3 are not installed on the 24/7 Data Acquisition Module. With both jumper shunts not installed, the address of the module is set to 0.
5. Turn on the power to the QED Board; this automatically powers the Module Carrier Board and the 24/7 Data Acquisition Module.
6. Connect Field 3 (Pin 18) to AGND (Pin 17).
7. Connect Field 2 (Pin 16) to a known voltage between 0 and 2.5 Volts.

8. Run the demonstration program as described in the readme.txt file on the distribution diskette. This program:
   - Initializes the module
   - Configures the module to use the on-board reference voltage
   - Calibrates and configures channel CH_2_3
   - Takes one 24 bit bipolar sample with a gain of one
   - Prints the voltage to the terminal
   - Returns a success flag.

Once the hardware and software are properly configured, you can obtain high resolution samples with the pre-coded high level driver routines.

Hardware

The 24/7 Data Acquisition Module consists of: the AD7714, a high resolution signal conditioning analog to digital converter; two low-power, low-dropout regulators that supply power for the digital and analog circuitry; a high precision voltage reference; 12 precision analog switches to protect the analog inputs of the analog to digital converter; three general purpose digital inputs; and seven analog input channels.

On-Board Reference

The 24/7 Data Acquisition Module has an on-board 2.5 V voltage reference that is accurate to +/- 0.005 V. This reference is available for use by sensors requiring a reference or producing a proportionate output. The reference voltage is available on pins 7 (REF+) and 13 (REF-) on H2 of the 24/7 Data Acquisition Module. To use the on-board reference voltage for analog to digital conversions call `Use_Onboard_Ref`. You can also use your own external reference voltage by connecting an external voltage to pins 9 (FDREF+) and 11 (FDREF-) of H2 and calling `Use_External_Ref`. Be sure that FDREF+ is always greater than FDREF- for correct operation of the analog to digital converter.

The recommended reference voltage for the analog to digital converter is 2.5 V. The 24/7 Data Acquisition Module is functional with external reference voltages from 2.5 volts down to 1 volt but with degraded performance as the output noise will, in terms of LSB size, be larger. If jumper J1 is installed, the reference voltage that appears on the REF+ output pin is changed to 3.0 V, which exceeds the specified reference input voltage by 0.5 volt. DO NOT USE THIS 3.0 VOLTS AS A REFERENCE VOLTAGE! That is, if jumper J1 is installed do not call `Use_Onboard_Ref` or connect REF+ and REF- to FDREF+ and FDREF-. The 3.0 V option is useful for powering external sensors and transducers and provides up to 10 mA of current. Figure 1 shows a typical application of 3.0 V option with an RTD.

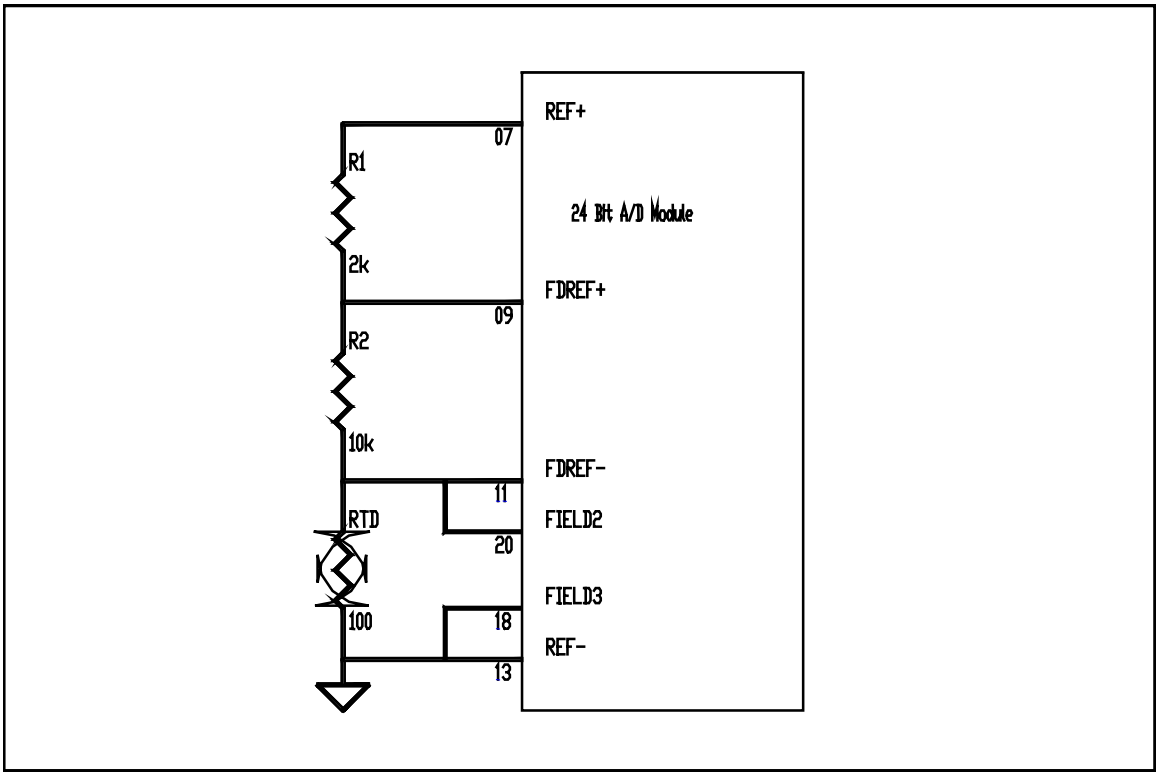


Figure 1: Typical Application of the 3.0 V Option

Digital Inputs

The 24/7 Data Acquisition Module has three TTL compatible CMOS digital inputs I17 (pin 5), I18 (pin 6), and I19 (pin 3) on the Field Bus header (H2). You can read the digital inputs by calling `Read_Digital_Input`.

Analog Inputs

The 24/7 Data Acquisition Module features four true differential, or seven pseudo-differential, overvoltage-protected, high-impedance analog input channels. "Pseudo Differential" operation indicates that multiple channels share the same signal return, FIELD7 (pin 10 of H2). The 10 different input options are outlined in Table 1.

The analog inputs are protected from electrostatic discharges of up to ±2000 volts and continuous voltages of up to ±70 volts by the resistors and analog switches in series with the analog inputs of the analog to digital converter. You can connect your transducer signals to the Field Bus (H2 on the 24/7 Data Acquisition Module) either through a ribbon cable or a screw terminal expansion board that breaks out the signals to screw terminal blocks. Shielding the connecting wires is highly recommended for optimal performance.

Associated Constant	Analog Input +	Analog Input -	Type
CH_0_7	FIELD0 (Pin 24)	FIELD7 (Pin 10)	Pseudo Differential
CH_1_7	FIELD1 (Pin 22)	FIELD7 (Pin 10)	Pseudo Differential
CH_2_7	FIELD2 (Pin 20)	FIELD7 (Pin 10)	Pseudo Differential
CH_3_7	FIELD3 (Pin 18)	FIELD7 (Pin 10)	Pseudo Differential
CH_4_7	FIELD4 (Pin 16)	FIELD7 (Pin 10)	Pseudo Differential
CH_5_7	FIELD5 (Pin 14)	FIELD7 (Pin 10)	Pseudo Differential
CH_6_7	FIELD5 (Pin 14)	FIELD7 (Pin 10)	Pseudo Differential
CH_0_1	FIELD0 (Pin 24)	FIELD1 (Pin 22)	Fully Differential
CH_2_3	FIELD2 (Pin 20)	FIELD3 (Pin 18)	Fully Differential
CH_4_5	FIELD4 (Pin 16)	FIELD5 (Pin 14)	Fully Differential
CH_6_7	FIELD6 (Pin 12)	FIELD7 (Pin 10)	Fully Differential

Table 1: Analog Input Connection Options

The gain range of each channel is 1 to 128 allowing the input full scale range to be user-selectable from 0-20 mV to 0-2.5V for unipolar signals, or a genuine bipolar range of +/- 20 mV, or a quasi-bipolar range of +/- 2.5 V. In all cases, the common mode range of the inputs is nominally 0-5V (precisely -30 mV to 5.03 V).

Components and transducers with high output impedances connected to the analog inputs will introduce gain errors in the analog to digital converter. Tables XII and

XIII in the data sheet show the allowable impedance for no 16 and 20 bit gain errors.

The 2.2k protection resistors and the analog switches add additional impedance and capacitance to the inputs, decreasing the amount of allowable impedance listed in Tables XII and XIII. To allow higher source impedances than those listed in Tables XII and XIII, the analog to digital converter has a buffered mode that buffers the input signal through a unity gain amplifier. The buffered mode operation provides zero gain error at the price of a small dc offset voltage of 2.2 μV. This offset voltage is generated by a 1 nA offset current from the buffer amplifier through the 2.2 kΩ protection resistor that is in series with the inputs to the analog to digital converter. To calculate the maximum offset voltage your transducer will generate, use Equation 1.

$$OffsetVoltage(\mu V)=2.2\mu V+TransducerSourceImpedance*1nA$$

Equation 1: Calculation of the Offset Voltage in Buffered Mode

To use the buffered mode call `Buffer_On` and to stop using the buffered mode call `Buffer_Off`. In buffered mode, the input voltage range is limited to +50 mV to +3.5 V.

Our measurements of the offset are provided in the following table:

Uncalibrated measured input offset voltage in microvolts					
BUFFER	CHANNEL	GAIN 1	GAIN 8	GAIN 16	GAIN 128
BUFFER OFF	1-2	1.8	3.8	3.1	3.5
	1-2	-0.3	3.1	3.7	2.8
	1-2	4.5	3.1	3.	3.1
	3-4	-9.5	0.22	-0.17	-0.5
	3-4	0.3	0.26	-0.2	-0.6
	3-4	-1.5	0	-0.3	-0.36
	5-6	0.6	-1.2	-1.6	-1.4
	5-6	-6.6	-1.2	-1.2	-1.9
	5-6	-1.2	-2.	-1.1	-1.2
BUFFER ON	1-2	189.	374.	374.	374.
	1-2	187.	374.	374.	374.
	1-2	190.	374.	373.	373.
	3-4	185.	373.	373.	374.
	3-4	187.	372.	373.	373.
	3-4	184.	373.	373.	373.
	5-6	186.	373.	373.	373.
	5-6	183.	374.	374.	373.
	5-6	182.	373.	374.	374.

This table was generated by shorting all of the inputs of the AD7714 chip to analog ground. Note that the offset variation is larger than the noise for each channel

especially at a gain of 1. Also note that with the buffer on, the offset becomes quite large. This offset can be zeroed out using the "System Offset Calibration".

Converter Noise and Effective Resolution

The AD7714 data sheet provides the effective resolution of a conversion as a function of gain and sampling frequency in Table Ia (for unbuffered mode) and Table III (for buffered mode) of the data sheet. The noise voltages shown on the data sheet result from noise generated internally to the AD7714 chip itself – approximately 1.5 microvolts rms at a 10 Hz sample rate and Gain of 1. The additional circuitry attached to the inputs of the chip contributes additional noise, effectively doubling the noise at low gains. Our own measurements of the rms noise show that it is increased to approximately 2.7 microvoltes at a 10 Hz sample rate and Gain of 1. The noise contribution of the additional circuitry diminishes at greater gains so that the noise figure is dominated by the internally generated noise.

Our own measurements of the 24-7 Wildcard Board's output noise are shown in the following table. Each cell of the table below was calculated by first performing a self calibration and then averaging 100 samples in bipolar mode at 10 Hz. Three sets of measurements were taken for each channel at each gain and the results averaged.

Measured RMS noise in microvolts					
BUFFER	CHANNEL	GAIN 1	GAIN 8	GAIN 16	GAIN 128
BUFFER OFF	1-2	3.0	0.64	0.38	0.25
	1-2	2.5	0.66	0.38	0.28
	1-2	2.7	0.69	0.40	0.29
	3-4	2.5	0.72	0.35	0.31
	3-4	2.7	0.60	0.37	0.23
	3-4	2.5	0.65	0.39	0.29
	5-6	2.7	0.61	0.42	0.34
	5-6	3.0	0.67	0.52	0.34
	5-6	2.6	0.67	0.42	0.34
BUFFER ON	1-2	3.0	0.44	0.38	0.27
	1-2	2.5	0.43	0.34	0.31
	1-2	2.6	0.41	0.36	0.32
	3-4	2.6	0.43	0.42	0.28
	3-4	2.6	0.44	0.33	0.30
	3-4	2.4	0.40	0.38	0.25
	5-6	2.5	0.39	0.34	0.24
	5-6	2.5	0.44	0.38	0.23
	5-6	2.2	0.47	0.42	0.26

It should be noted that chip's data sheet defines noise and resolution using rms noise voltages, not peak-to-peak output noise numbers. Peak-to-peak noise numbers can be up to 6.6 times the rms numbers while effective resolution numbers based on peak-

to-peak noise can be 2.5 bits below the effective resolution based on rms noise as quoted in the data sheet's tables.

Module Select Jumpers

The Module Select Jumpers, labeled J2 and J3, select a 2-bit code that sets a unique address (0-7) on the module port of the Module Carrier Board. Each module port on the Module Carrier Board can accommodate up to 4 modules. Module Port 0 on the Module Carrier Board provides access to modules 0-3 while Module Port 1 provides access to modules 4-7. Two modules on the same port cannot have the same address (jumper settings). Table 2 shows the possible jumper settings and the corresponding addresses.

Module Port	Module Address	Installed Jumper Shunts
0	0	None
	1	J2
	2	J3
	3	J2 and J3
1	4	None
	5	J2
	6	J3
	7	J2 and J3

Table 2: Jumper Settings and Associated Addresses

Tips For Measuring Small Voltages

When measuring small voltages with the 24-Bit A/D Module you should use twisted pairs of wires between the signals and the analog inputs of the Module. Try to keep the wires as short as possible to reduce the noise that will be coupled into the A/D converter. You should also be aware of the thermocouple effect that can add microvolts of temperature dependent offset to a signal. It is difficult to completely prevent thermocouple effects, but a few precautions can minimize them. Whenever possible use twisted pair wires of the same type extending from the voltage to be measured and terminated at the field I/O connector.

To accurately measure small voltages, use the System Offset Calibration. The different calibration options will be covered in detail in the Software Section. The System Offset Calibration requires you to short the field inputs of the module together (at the sensor) during the calibration sequence to remove any offsets between the field inputs of the module and the inputs of the A/D Converter. Offsets can be as great as 100 microvolts if a System Offset Calibration is not preformed.

Software

The following section describes the high-level software routines that initialize, calibrate, configure, and command the 24/7 Data Acquisition Module. Once the 24/7 Data Acquisition Module is commanded to start converting, it continually samples and converts at a fixed rate. You can read the output once or store each sequential conversion to a memory buffer. Only one channel is sampled at a time; selecting a new channel requires restarting a conversion.

The software routines use global variables to store the current module and channel information. Thus all of the 24/7 Data Acquisition Module software routines are non-re-entrant and only a single task in a multitasking application can use a 24/7 Data Acquisition Module.

Please take time to read through the example programs. The examples illustrate how to effectively use the pre-coded software drivers. Many subtleties involved with using the 24/7 Data Acquisition Module are highlighted in the example code.

The following steps and corresponding routines are required to obtain a sample from the 24/7 Data Acquisition Module:

1. Initialize the 24/7 Data Acquisition Module using `Init_AD24`.
2. Specify the reference voltage by calling `Use_Onboard_Ref` or `Use_External_Ref`.
3. Calibrate, configure, and start converting samples from a channel with `Start_Conversion`.
4. Get one or more samples by executing `AD24_Sample` or `AD24_Multiple`.

Each step is described in detail in the next four sections.

Initializing the 24/7 Data Acquisition Module

To initialize the 24/7 Data Acquisition Module, call `Init_AD24`. This configures the module for communication via the QED Board's Serial Peripheral Interface (SPI), starts the QED Board's Timeslicer, and performs a reset (`Reset_AD24`) that disconnects all of the analog inputs from the AD7714 and resets its digital filter, analog modulator, and registers.

Specifying The Reference Voltage

Once the module is initialized, you must choose what type of reference voltage to use. To use the on-board 2.5 V reference voltage, simply call `Use_Onboard_Ref` and make sure the jumper shunt on J1 is not installed. If an external reference voltage is required, connect the reference voltage to FDREF+ (pin 9 on H2) and FDREF- (pin 11 on H2) and call `Use_External_Ref`. These two routines connect the specified reference voltage to the reference input on the analog to digital

converter. Be sure to only use 1.0 to 2.5 volts as an external reference voltage and be sure that FDREF+ is more positive with respect to FDREF-.

Starting A Conversion

After you initialize the 24/7 Data Acquisition Module and specify the reference voltage, you can calibrate, configure, and start converting samples using `Start_Conversion`. An alternative routine, `Start_Conv_With_Values`, allows you to forgo the calibration process and use already determined calibration coefficients instead. Each of these routines are discussed in turn.

Calibrating and Starting a Conversion

To use any of the channels listed in Table 1, the channel must first be calibrated. You should also perform a calibration whenever the temperature or supply voltage changes significantly, or whenever you change the gain, sample frequency, or polarity options. `Start_Conversion` calibrates the analog to digital converter with the specified options and calibration type and initiates continuous sampling and conversion. `Start_Conversion` requires seven parameters to configure the sample frequency, gain, calibration type, polarity, resolution, burnout current option, and channel: each of which is discussed in detail in the following paragraphs. `Start_Conversion` also performs extensive error checking on each parameter to ensure that each option is valid. Table 3 lists the possible error codes returned by `Start_Conversion`.

Error Code	Description
-1	Start Conversion Successful
1	Invalid Gain
2	Invalid Frequency
3	Invalid Calibration Type
4	Invalid Channel
5	Invalid Synchronization Option
6	Invalid Burnout Current Option
7	Invalid Resolution
8	Invalid Polarity
9	Timeout Error

Table 3: Error Codes Returned By Start Conversion

Starting A Conversion with Predetermined Calibration Coefficients

Once you calibrate a channel using `Start_Conversion`, you do not have to repeat the calibration unless the temperature or power supply voltage changes. To start a conversion without a calibration use `Start_Conv_With_Values`. This routine allows you to quickly switch back and forth between channels without

recalibrating the channel each time. `Start_Conv_With_Values` requires nine parameters: the full scale calibration values, zero scale calibration values, sample frequency, gain, polarity, resolution, burnout current option, synchronization option, and channel.

To call `Start_Conv_With_Values` you need valid calibration coefficients. You can obtain these coefficients by calling `Read_Zero_Cal` and `Read_FS_Cal` *after* `Start_Conversion` is called. Be sure to save *all* seven parameters that specify the channel information (such as the gain, resolution, and sample frequency) along with the calibration values. See the example program in the Example Code Section for an example application using `Start_Conv_With_Values` to obtain and store 10 samples to an array from four channels.

No error checking is done in `Start_Conv_With_Values` to minimize the time required to switch from one channel to another and because error checking would already have been done by `Start_Conversion`.

The following sections describe the parameters required by `Start_Conversion` and `Start_Conv_With_Values`.

Calibration Options

The calibration option is only required by `Start_Conversion` and not by `Start_Conv_With_Values`. The 24/7 Data Acquisition Module has two basic calibration options, a self calibration and a system calibration. A self calibration performs a calibration that sets the analog to digital converter's zero point and gain by internally shorting its inputs. A system calibration performs the same calculations as a self calibration but uses voltages at the analog field inputs for the zero and full scale points. All together there are seven calibration options that are variations on a self or system calibration.

Figure 2 displays the different calibration options and their relationship to the two different calibration types. The left side of the figure shows the four different types of self calibration options while the right side shows the three different types of system calibration options. The bottom level contains the four basic calibration options while the next level shows the higher-level calibration options that simply call the basic calibration options in different ways.

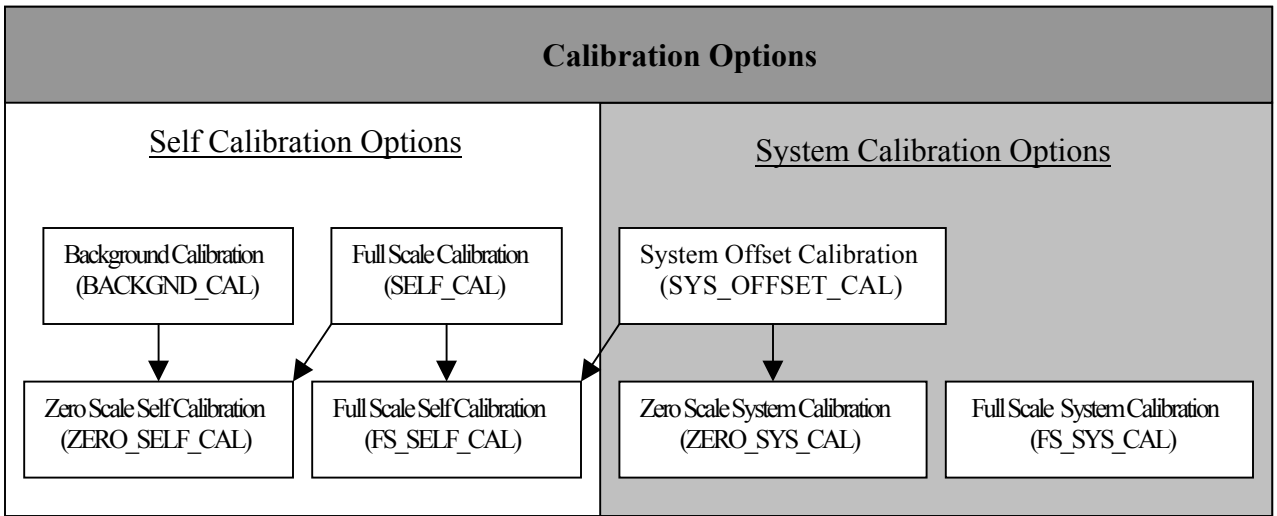


Figure 2: Graphical Representation of the Different Calibration Options

Table 4 shows a summary of the calibration options and the associated constants that you must pass to `Start_Conversion` as well as their execution times. Each calibration option is discussed in the following sections.

Calibration Type	Duration of Calibration	Associated Constant
Full Self Calibration	10 x sample period*	SELF_CAL
Zero Scale Self Calibration	7 x sample period*	ZERO_SELF_CAL
Full Scale Self Calibration	7 x sample period*	FS_SELF_CAL
Background Calibration	6 x sample period	BACKGND_CAL
Zero Scale System Calibration	5 x sample period*	ZERO_SYS_CAL
Full Scale System Calibration	5 x sample period*	FS_SYS_CAL
System Offset Calibration	10 x sample period*	SYS_OFFSET_CAL

*The duration of the calibration operations are rounded up from the times listed in Table XI of the data sheet to account for the pipe line delay as specified in the data sheet.

Table 4: Calibration Options and Their Duration

Self Calibration Options

A self calibration sets the analog to digital converters zero point and gain by internally shorting its inputs. A *Full Self Calibration* performs both *Zero Scale Self Calibration*, that sets the analog to digital converter's zero point, and a *Full Scale Self Calibration*, that sets the analog to digital converter's gain. Either of these, a *Zero Scale Self Calibration* or a *Full Scale Self Calibration*, may also be performed separately from a *Full Self Calibration*. However, a *Full Scale Self Calibration* should not be performed unless the analog to digital converter has valid zero-scale coefficients. The zero-scale coefficients are loaded by calling `Start_Conv_With_Values`, performing a *Full Self Calibration*, or performing a *Zero Scale Self Calibration*. You can perform a self-calibration in separate steps after a *Full Self Calibration* for additional offset or gain calibrations. Calibrating one of the parameters, either offset or gain, does not affect the other parameter.

A *Background Calibration* interleaves the calibration procedure with the normal conversion sequence. In background calibration mode, the analog to digital converter provides continuous zero-scale self-calibrations; it does not provide any full-scale calibrations. When invoked, the background calibration mode performs a *Zero Scale Self Calibration* after every sample, reducing the sampling frequency by a factor of six. Its advantage is that the analog to digital converter is continually performing offset calibrations and automatically updating its zero-scale calibration coefficients. As a result, the effects of temperature drift, supply sensitivity, and time drift on zero-scale errors are automatically removed. Because the background calibration does not perform full-scale calibrations, a *Full Self Calibration* should be performed before starting a background calibration. Removal of the offset drift in this mode leaves gain drift as the only source of error not removed. Typical gain

drift with temperature is 0.2 ppm/degree C. The synchronization option discussed below should not be used when a *Background Calibration* is operating.

System Calibration Options

A system calibration performs the same calculations as a self calibration but uses voltages at the analog field inputs for the zero and full scale points. Full system calibration requires a two-step process, a *Zero Scale System Calibration* followed by a *Full Scale System Calibration*. A *Full Scale System Calibration* should not be performed unless the analog to digital converter has valid zero-scale coefficients. The zero-scale coefficients are loaded by calling `Start_Conv_With_Values` or by performing a *Zero Scale System Calibration*. The input voltages used for both system calibrations must be applied to the 24/7 Data Acquisition Module before the calibration is initiated and remain stable until the calibration is complete. In unipolar mode, the system calibration is performed between the two endpoints of the transfer function; in the bipolar mode, it is preformed between midscale (zero differential voltage) and positive full scale. You can perform a system-calibration in separate steps for additional offset or gain calibrations. Calibrating one of the parameters, either offset or gain, does not affect the other parameter.

The *System Offset Calibration* is a variation of both the system calibration and self-calibration. In this case, the zero-scale point is determined with a *Zero Scale System Calibration* and the full-scale calibration is performed with a *Full Scale Self Calibration*. The zero-scale point must be applied to the analog field inputs before the calibration is initiated and remain stable until the calibration is complete.

Whenever you use a system calibration mode, there are limits on how much the offset and span can be adjusted. The positive full-scale calibration limit is $\geq$ 1.05 x VREF/GAIN. This allows the input range to go 5% above the nominal range so that the digital to analog converter will still operate correctly with a positive full-scale voltage beyond the nominal. For more information on span and offset limits, see page 25 of the data sheet.

Frequency Option

The 24/7 Data Acquisition Module supports a wide range of sample frequencies from 4.8 Hz to over 1.01 kHz. The sample frequency is specified as an integer (*frequency integer* in this document) from 19 to 4000 to `Start_Conversion` or `Start_Conv_With_Values`. Equation 2 shows the relationship between the frequency integer and the sample frequency and Table 5 lists some of the available sample frequencies.

$$SampleFrequency(Hz) = \frac{2457600\,Hz}{128 * n} = \frac{19200}{n}$$

where n is the frequency integer whose value ranges from 19 - 4000

Equation 2: Calculation of the Sample Frequency from the Frequency Integer

The sample period is equal to one over the sample frequency.

Integer n	Sample Frequency in Hertz
4000	4.800
3999	4.801
3998	4.802
3840	5.000
1920	10.000
384	50.000
320	60.000
192	100.000
160	120.000
96	200.000
64	300.000
48	400.000
40	480.000
32	600.000
30	640.000
24	800.000
21	914.286
20	960.000
19	1010.526

Table 5: Available Sample Frequencies

Changing the sampling frequency as well as the gain impacts resolution. Table 6, taken from the manufacturers datasheet, shows the effect of the sampling frequency and gain on the effective resolution.

Sample Frequency in Hz	Accuracy as an Effective Resolution In Bits			
	Gain of 1	Gain of 8	Gain of 32	Gain of 128
10	22.5 (21.5)*	21.5	20.0	18.0 (16.5)*
60	20.0	19.5	18.0	16.0
240	15.5	15.5	15.5	14.5
960	11.0 (10.5)*	11.0	10.5	10.5 (11.0)*

* Values in parentheses indicate our own measurements of the effective resolution.

Table 6: Effect of Gain and Sample Frequency on Effective Resolution

In Table 6, the sample frequencies are rounded to the nearest whole number and values in parentheses indicate our own measurements of the effective resolution. The analog to digital converter internally adds approximately 1 μV of noise on its

analog inputs. This limits the accuracy of the converter at different sample frequencies. The table shows that the other components on the module (including the analog switches and protection circuitry) add an additional 1 μV of noise to the analog to digital converter.

The 24/7 Data Acquisition Module has a built in low pass filter that has a first notch filter frequency equal to the sample frequency and a –3dB frequency equal to about one quarter of the sample frequency (0.262 x sample frequency). At the sample frequency and integer multiples of the sample frequency, the filter attenuates input signals by more than 100 dB. For example, if 10 Hz is used as the sample frequency, there will be notches at 50 Hz and 60 Hz that will significantly attenuate all differential and common-mode noise.

Gain Options

The gain option is specified as a constant to `Start_Conversion` or `Start_Conv_With_Values`. Valid constants are GAIN_1, GAIN_2, GAIN_4, GAIN_8, GAIN_16, GAIN_32, GAIN_64, or GAIN_128. Saturation will occur if the absolute value of the input signal multiplied by the gain is greater than 2.5 volts.

Bipolar/Unipolar Option

Passing the UNIPOLAR or BIPOLAR constant to `Start_Conversion` or `Start_Conv_With_Values` sets the polarity option for the module. The 24/7 Data Acquisition Module accepts either unipolar or bipolar input voltage ranges. Bipolar input ranges do not imply that the part can handle negative voltages below –20 mV on its analog inputs! The input channels are arranged in pairs with an Analog In + and Analog In -. As a result, the voltage to which the unipolar and bipolar signals on the Analog In + input are referenced is the voltage on the respective Analog In - input. For example, if Analog In – is +2.5 V and the module is configured for unipolar operation with a gain of 2 and a reference voltage of +2.5 V, the input voltage range on the Analog In + input is +2.5 V to +3.75 V. If Analog In – is +2.5 V and the module is configured for bipolar mode with a gain of 2 and a reference voltage of +2.5 V, the analog input range on the Analog In + input is +1.25 V to +3.75 V (i.e., 2.5 V +/- 1.25 V).

Resolution Option

The 24/7 Data Acquisition Module has two resolution settings: 16 and 24 bit. The 16 bit resolution option is used when the constant WORD_16BIT is passed to `Start_Conversion` or `Start_Conv_With_Values`. The 24 bit option is used when the constant WORD_24BIT is passed to `Start_Conversion` or `Start_Conv_With_Values`. Changing the resolution only changes the data size; it does not affect the conversion process.

Burnout Current Option

The analog to digital converter contains two 1 μA currents, one source current from the analog power supply to the current Analog In + and one sink from Analog In – to analog ground. The currents are either both on or both off depending on the constant, BO_ON or BO_OFF, that is passed to `Start_Conversion` or `Start_Conv_With_Values`. You can use these currents to see if a transducer has burned out or gone open-circuit before attempting to take measurements on that channel. If the transducer becomes an open-circuit with the burnout currents on, the input voltage will be measured as full scale. If the transducer becomes a short circuit with the burnout currents on, the input voltage will be measured as zero.

Synchronization Option

The synchronization option allows you to reset the modulator and the digital filter of the analog to digital converter without affecting any other settings such as gain, channel, polarity, or calibration values. This allows conversions to be started at a precisely known time. To start a synchronous conversion, pass FSYNC_ON to `Start_Conv_With_Values` or call `Sync`, which synchronizes the conversion through a hardware pin. A valid sample will be available 3 sample periods (3/sample frequency) later. For example, if the sample frequency is 10 Hz, the channel has already been calibrated, and `Sync` is called, the next sample will be available after 300 ms (3 / 10 Hz). The synchronization option should not be used when a Background Calibration is operating.

Channel Option

Table 1 lists the options available for channel selection.

Obtaining A Sample

Once the 24/7 Data Acquisition Module has been initialized, calibrated, and configured, you can take one or more samples using `AD24_Sample` or `AD24_Multiple`. Irrespective of the resolution option chosen, the samples are returned as 32-bit (4 byte) values. `AD24_Sample` acquires a single sample by polling the analog to digital converter. If a sample is not available within a timeout period, an error is returned. The timeout period is calculated by Equation 3.

$$Timeout(TimesliceCounts) = \frac{NumberOfSamples + CalibrationDelay}{SampleFrequency * TimeslicePeriod}$$

Equation 3: Calculation of the Timout Value

The calibration delay is based on the calibration option and is listed in Table 4 as the "Duration of Calibration". If a timeout occurs, the timeout error flag is placed in the

least significant byte of the returned 32-bit value. A 32-bit value is used to allow an error flag to be returned with data regardless of the resolution (16 or 24 bits). See Figure 3 for a diagram of the data format.

WWWWWWWW XXXXXXXX YYYYYYYY ZZZZZZZZ

Figure 3: 32 Bit Value Returned By AD24_Sample

For a16 bit sample, WWWWWWWW is the most significant byte, XXXXXXXX is the least significant byte, YYYYYYYY is 0, and ZZZZZZZZ is the timeout flag. For a 24 bit sample, WWWWWWWW is the most significant byte, XXXXXXXX is the next significant byte, YYYYYYYY is the least significant byte, and ZZZZZZZZ is the timeout flag.

To obtain multiple samples, you can use `AD24_Multiple`. This routine acquires up to 8192 samples and stores the samples as sequential 32 bit values in memory within a timeout period or a false flag is returned. This routine sets up an interrupt service routine to obtain the multiple samples. The interrupt service routine (ISR) runs at more than twice the sample frequency to eliminate clock variations between the 24/7 Data Acquisition Module and QED Board and to guarantee that a sample is not missed even if the ISR is delayed up to 1/2 a sample period (or one full ISR period). Equation 4 shows how the ISR period is calculated.

$$ISRPeriod = \frac{SamplePeriod - InterruptLatencyDelay}{2}$$

Equation 4: Calculation of the ISR Period

The interrupt latency delay is composed of 200 μs for the time to update the analog to digital converter's data register with sample values, 10 μs of interrupt latency delay, and 50 μs to read the data ready line when the ISR is entered. You can also use this equation to calculate the maximum time the ISR can be delayed or the SPI used without missing a sample. The 24/7 Data Acquisition Module uses the SPI to communicate with the QED Board. You can create other tasks or routines that use the SPI as long as they do not use the SPI longer than one ISR period. For example to guarantee that a sample is not missed at a sample rate of 10 Hz, the maximum time another interrupt can take or routine can use the SPI is approximately 50 ms (0.04987 = ((1/10) - 0.00026) / 2).

C Example Listing

```
#include </mosaic/allqed.h>
#include "library.c"

//---------------------------------------------------------------------
//                            Example 1
//---------------------------------------------------------------------

// This first sample routine demonstrates how to use Init_AD24,
// Use_Onboard_Ref, Start_Conversion, and AD24_Sample.  This routine
// takes 1 differential 24-bit bipolar sample at 10 Hz with a gain
// of 1 and the burnout options turned off and prints it out.  If an

// invalid option is specified or if a timeout occurs, an error flag

// is returned.  Error flags are:
//  INVALID_GAIN = 1, INVALID_FREQ = 2, INVALID_CAL = 3,
//  INVALID_CHANNEL = 4, INVALID_FSYNC = 5, INVALID_BO = 6,
//  INVALID_SIZE = 7, INVALID_POLARITY = 8, TIMEOUT_ERROR = 9
int Sample_Routine ( void )
{
  int flag = 0;
  ulong sample;
  float result;

  if( Init_AD24( MODULE0) )         // 24/7 Data Acquisition Module

  {                                 // is first module on the stack

    Use_Onboard_Ref();              // Use on-board reference

    // Start_Conversion must be called before getting a sample!
    flag = Start_Conversion(  SELF_CAL,
                              1920,       // 10 Hz -> 19200/10=1920
                              GAIN_1,
                              BIPOLAR,
                              WORD_24BIT, // 24 bit resolution
                              BO_OFF,
                              CH_0_1 );
    if( flag == -1 )
    {
      sample = AD24_Sample();              // Get sample
      if( sample != TIMEOUT_ERROR )        // If no timeout occurred
      {
        // Divide by 256 to remove timeout flag & convert to 24 bits
        // Subtract 8388608 (2^23) to remove the bipolar offset
        // Multiply by (5.00+/-0.01)/(2^24) to convert to volts
         // 5.00+/-0.01 is obtained by multiplying the reference
         // voltage by 2; i.e. (2.500+/-0.005)*2
         // Divide result by the gain for gains greater than 1
         // for example: for a gain of 8 divide result by 8
         // DO NOT DIVIDE BY GAIN_8! GAIN_8 != 8!
        result=(float)((sample/256)-8388608)*(0.0000002980);

        printf("\nResult = %2.4f \n",result);
```

```
      }
    }
  }
  return(flag);
}

//---------------------------------------------------------------------
//                            Example 2
//---------------------------------------------------------------------

// This second sample routine demonstrates how to use AD24_Multiple.

// This routine takes 10 samples from a single channel at 10 hz and
// stores the samples to the pad area.  Returns -1 if successful, 0
// if an invalid module number was passed to Init_AD24, returns 1-9
// if an invalid calibration coefficient was passed to
// Start_Conversion.
int Sample_Routine2 ( void )
{
  int flag = 0;
  EXTENDED_ADDR pad_buffer;        // 88 byte buffer in common RAM

  pad_buffer.sixteen_bit.page16 = 0x00;
  pad_buffer.sixteen_bit.addr16 = 0x8b24;


  if( Init_AD24( MODULE0 ) )       // 24/7 Data Acquisition Module
                                   // is first module on the stack
  {
    Use_Onboard_Ref();             // Use on-board reference

    // Start_Conversion must be called before getting a sample!
    flag = Start_Conversion(  SELF_CAL,
                              1920,       // 10 Hz -> 19200/10=1920
                              GAIN_1,
                              BIPOLAR,
                              WORD_24BIT, // 24 bit resolution
                              BO_OFF,
                              CH_0_1 );
    if( flag == -1 )          // Start_Conversion was successful

    {
      // Get 10 samples, store to RAM
      // The values raw conversion values can be shown by typing:
      // pad 40 dump
      // The values can be read from memory by using: FetchLong()
      flag = AD24_Multiple( 10, pad_buffer.addr32 );
    }
  }
  return(flag);
}

//---------------------------------------------------------------------
//                            Example 3
//---------------------------------------------------------------------

// The final example shows how to read 4 different channels at a
```

```
// fixed sample rate without performing a calibration before each
// sample and without using interrupts.  This example performs a
// Self Calibration on each channel, reads the calibration
// coefficients using Read_FS_Cal() and Read_Zero_Cal(), stores the
// calibration coefficients into a structure, loads the 24-Bit A/D
// with the stored calibration coefficients using
// Start_Conv_With_Values(), periodically samples each channel using

// AD24_Sample_NP(), and finally stores the samples into an array.

#define CH0       0              //  Constants for channels 0 - 3
#define CH1       1
#define CH2       2
#define CH3       3
#define SAMPLE_FREQ 320          // Constant corresponding to 60 hz
                                  // 19200 / 60 = 320 [See Table 5]
#define NUM_SAMPLES 40           // Total number of samples:
                                  // 10 samples for 4 channels
#define NUM_CHANNELS 4           // Num channels we are sampling

// Declare an array for samples & allocate memory for the samples
// from 4 sensors; each sample is 4 bytes.
ulong ad24_data[NUM_SAMPLES/NUM_CHANNELS][NUM_CHANNELS];


typedef struct                   // Config options for each channel
{
  ulong ad_zero_cal;             // 24-bit zero sacle cal val
  ulong ad_fs_cal;               // 24-bit full scale cal val
  int  ad_freq_int;              // Frequency Integer 19 - 4000.
  char ad_gain;                  // Gain 1 to 128.
  char ad_polarity;              // Bipolar or Unipolar mode.
  char ad_res;                   // Resolution: 16-bit or 24-bit.
  char ad_bo;                    // Burn out current on/off
  char ad_fsync;                 // Sync on/off.
  char ad_ch;                    // Channel.
} AD_CHANNEL;

typedef struct                   // Global structure.
{
  AD_CHANNEL ch0;
  AD_CHANNEL ch1;
  AD_CHANNEL ch2;
  AD_CHANNEL ch3;
  char   current_channel;        // Current channel being used.
  int    index;                  // Index into data array
} AD_INFO;

AD_INFO ad24_struct;             // Declare a global instance of
the
                                  // structure.

// Perform a Full Self Calibration on channel 0-1 for bipolar, unity

// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 0 of my_struct with calibration coefficients
// and settings.
```

```
int Init_CH0 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_0_1 );

  if( flag == -1 )
  {
    ad24_struct.ch0.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch0.ad_gain     = GAIN_1;
    ad24_struct.ch0.ad_polarity = BIPOLAR;
    ad24_struct.ch0.ad_res      = WORD_24BIT;
    ad24_struct.ch0.ad_bo       = BO_OFF;
    ad24_struct.ch0.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch0.ad_ch       = CH_0_1;
    ad24_struct.ch0.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch0.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 2-3 for bipolar, unity

// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 1 of my_struct with calibration coefficients
// and settings.
int Init_CH1 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_2_3 );

  if( flag == -1 )
  {
    ad24_struct.ch1.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch1.ad_gain     = GAIN_1;
    ad24_struct.ch1.ad_polarity = BIPOLAR;
    ad24_struct.ch1.ad_res      = WORD_24BIT;
    ad24_struct.ch1.ad_bo       = BO_OFF;
    ad24_struct.ch1.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch1.ad_ch       = CH_2_3;
    ad24_struct.ch1.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch1.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 4-5 for bipolar, unity

// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 2 of my_struct with calibration coefficients
// and settings.
int Init_CH2 ( void )
{
  int flag;
```

```
  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_4_5 );

  if( flag == -1 )
  {
    ad24_struct.ch2.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch2.ad_gain     = GAIN_1;
    ad24_struct.ch2.ad_polarity = BIPOLAR;
    ad24_struct.ch2.ad_res      = WORD_24BIT;
    ad24_struct.ch2.ad_bo       = BO_OFF;
    ad24_struct.ch2.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch2.ad_ch       = CH_4_5;
    ad24_struct.ch2.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch2.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 6-7 for bipolar, unity

// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 3 of my_struct with calibration coefficients
// and settings.
int Init_CH3 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_6_7 );

  if( flag == -1 )
  {
    ad24_struct.ch3.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch3.ad_gain     = GAIN_1;
    ad24_struct.ch3.ad_polarity = BIPOLAR;
    ad24_struct.ch3.ad_res      = WORD_24BIT;
    ad24_struct.ch3.ad_bo       = BO_OFF;
    ad24_struct.ch3.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch3.ad_ch       = CH_6_7;
    ad24_struct.ch3.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch3.ad_fs_cal   = Read_FS_Cal();
  }
    return(flag);
}

// This routine loads the 24-Bit A/D with the next set of
// calibration coefficients without performing a calibration.
void Load_Coefficients( int current_ch )
{

  switch( current_ch )
  {
    case CH0: Start_Conv_With_Values( ad24_struct.ch0.ad_fs_cal,
              ad24_struct.ch0.ad_zero_cal,
              ad24_struct.ch0.ad_freq_int,
              ad24_struct.ch0.ad_gain, ad24_struct.ch0.ad_polarity,
```

```
                ad24_struct.ch0.ad_res, ad24_struct.ch0.ad_bo,
                ad24_struct.ch0.ad_fsync, ad24_struct.ch0.ad_ch );
                break;
      case CH1: Start_Conv_With_Values( ad24_struct.ch1.ad_fs_cal,
                ad24_struct.ch1.ad_zero_cal,
                ad24_struct.ch1.ad_freq_int,
                ad24_struct.ch1.ad_gain, ad24_struct.ch1.ad_polarity,
                ad24_struct.ch1.ad_res, ad24_struct.ch1.ad_bo,
                ad24_struct.ch1.ad_fsync, ad24_struct.ch1.ad_ch );
                break;
      case CH2: Start_Conv_With_Values( ad24_struct.ch2.ad_fs_cal,
                ad24_struct.ch2.ad_zero_cal,
                ad24_struct.ch2.ad_freq_int,
                ad24_struct.ch2.ad_gain, ad24_struct.ch2.ad_polarity,
                ad24_struct.ch2.ad_res, ad24_struct.ch2.ad_bo,
                ad24_struct.ch2.ad_fsync, ad24_struct.ch2.ad_ch );
                break;
      case CH3: Start_Conv_With_Values( ad24_struct.ch3.ad_fs_cal,
                ad24_struct.ch3.ad_zero_cal,
                ad24_struct.ch3.ad_freq_int,
                ad24_struct.ch3.ad_gain, ad24_struct.ch3.ad_polarity,
                ad24_struct.ch3.ad_res, ad24_struct.ch3.ad_bo,
                ad24_struct.ch3.ad_fsync, ad24_struct.ch3.ad_ch );
                break;
    }
}

// This routine takes one sample, stores it to an array, then starts
// a conversion for the next channel.
void Get_Sample ( void )
{
// Get sample from the 24-Bit A/D, store to array
ad24_data[ad24_struct.index][ad24_struct.current_channel]=AD24_Sample_NP();

  // Increment channel number, did we sample all the channels yet?
  if( ad24_struct.current_channel++ >= NUM_CHANNELS - 1 )
  {
    // Init channel number to 0, we finished sampling all channels.
    ad24_struct.current_channel = 0;
    // Set varible to store next group of data
    ad24_struct.index++;
  }

  // Load coefficients for the next channel
  Load_Coefficients ( ad24_struct.current_channel );
}

// This routine repeatedly calls Get_Sample() every timeslice_ticks
// * 5ms.  Be sure other tasks do not take longer than
// timeslice_ticks * 5ms.
void Execute_So_Often ( uint num_times, ulong timeslice_ticks )
{
  ulong target_time;
  int i;

  for(i=0;i<num_times;i++)
  {
```

```
    target_time = TIMESLICE_COUNT + timeslice_ticks;
    Get_Sample();
    do
    {
      Pause();
    } while (TIMESLICE_COUNT<target_time);
  }
}

// This routine takes 10 samples from 4 sensors at 60 Hz.  All of
// the settings for each channel are stored in a global structure.
// All channels must have the same sampling rate!
int Sample_Routine3 ( void )
{
  int flag;

  flag = Init_AD24( MODULE0 );// 24/7 Data Acquisition Module is
                              // the first module on the stack

  Use_Onboard_Ref();          // Use on-board reference
  ad24_struct.current_channel = CH0;// Set ch0 as current channel
  ad24_struct.index = 0;      // Init array index number

  // Init structure;  Be sure to call Init_CH0 last since it is the
  // first channel sampled.
  Init_CH3(); Init_CH2(); Init_CH1(); Init_CH0();

  // Get 1 sample every 60 ms.  60 ms is the fastest we can call
  // Get_Sample() because the sample rate is 60Hz and the 24-Bit A/D
  // takes 3 clock cycles to obtain a sample when using
  // Start_Conv_With_Values().  This alone is 3/60 or 50 ms.  If a
  // full Self-Calibration was performed before each conversion, the
  // fastest rate you could sample one channel would be 10/60 or 167
  // ms.  This would amount to 666 ms for 4 channels or 1.5 Hz per
  // channel.
  Execute_So_Often( NUM_SAMPLES, (ulong)12 ); // 12*5ms = 60ms
  return(flag);

}
```

Forth Example Listing

```
\ ----------------------------------------------------------------------
\                          Example 1
\ ----------------------------------------------------------------------

hex

FF constant LSB_MASK                \ Constant to isolate 8 least
                                    \ significant bits of an integer
decimal

\ This first sample routine demonstrates how to use Init_AD24,
\ Use_Onboard_Ref, Start_Conversion, and AD24_Sample.  This routine
\ takes 1 differential 24-bit bipolar sample at 10 Hz with a gain of 1
\ and the burnout options turned off and prints it out.  If an invalid
\ option is specified or if a timeout occurs, an error flag is
\ returned.  Error flags are: INIT_ERROR = 0,
\           INVALID_GAIN = 1, INVALID_FREQ = 2, INVALID_CAL = 3,
\           INVALID_CHANNEL = 4, INVALID_FSYNC = 5, INVALID_BO = 6,
\           INVALID_SIZE = 7, INVALID_POLARITY = 8, TIMEOUT_ERROR = 9
: Sample_Routine ( -- flag | flag = success )
false locals{ &flag }

  MODULE0 Init_AD24                 \ 24/7 Data Acquisition Module is
                                    \ the first module on the stack
  if
    Use_Onboard_Ref                 \ Use on-board reference
    SELF_CAL
    1920                            \ 10 Hz -> 19200 / 10 = 1920
    GAIN_1
    BIPOLAR
    WORD_24BIT                      \ 24 bit resolution
    BO_OFF
    CH_2_3
    Start_Conversion                \ This must be called before
                                    \ getting a sample
    to &flag
    &flag -1 =
    if
      AD24_Sample                   \ Get sample
      over LSB_MASK AND             \ Just look at 8 LSB of the sample
      TIMEOUT_ERROR = not           \ Does it equal to timeout flag?
      if                            \ If no timeout occurred,
        -8 DSCALE                   \ Shift sample to get 24 bits
                                    \ Convert to volts
        din 8388608 d- dflot        \ Subtract off 0x800000 because
                                    \ of the bipolar conversion
        0.0000002980 f*             \ Multiply by (5.00+/-0.01)/(2^24)

       \ 5.00+/-0.01 is obtained by multiplying the reference
       \ voltage by 2; i.e. (2.500+/-0.005)*2

        \ 1.0 f/                    \ Divide by the gain if necessary
                                    \ DO NOT DIVIDE BY GAIN_1!
```

```
                                    \ GAIN_1 != 1, GAIN_2 != 2, ...
        f.                          \ Print out result
        true to &flag               \ Return true
      endif
    endif
  endif
  &flag
;

\ ------------------------------------------------------------------
\                         Example 2
\ ------------------------------------------------------------------

\ This second sample routine demonstrates how to use AD24_Multiple.
\ This routine takes 10 samples from a single channel at 10 hz and
\ stores the samples to the pad area.  Returns TRUE if successful,
\ FALSE if a timeout occurred in AD24_Multiple, an invalid calibration
\ coefficient was passed to Start_Conversion, or an invalid module
\ number was passed to Init_AD24.

: Sample_Routine2 ( -- flag )

  MODULE0 Init_AD24                 \ 24/7 Data Acquisition Module is

                                    \ the first module on the stack
  if
    Use_Onboard_Ref                 \ Use on-board reference
    SELF_CAL
    1920                            \ 10 Hz -> 19200 / 10 = 1920
    GAIN_1
    BIPOLAR
    WORD_24BIT                      \ 24 bit resolution
    BO_OFF
    CH_0_1
    Start_Conversion                \ This must be called before
                                    \ getting a sample
    if                              \ If a conversion was
                                    \ successfully started,
      10 pad AD24_Multiple          \ Get 10 samples, store to pad
    else
      FALSE                         \ Invalid calibration coefficient
    endif
  else
    FALSE                           \ Invalid module number
  endif
;

\ ------------------------------------------------------------------
\                         Example 3
\ ------------------------------------------------------------------

\ The final sample routine uses the timeslice clock to obtain 10
\ samples from 4 different sensors at 60 Hz without using interrupts.
\ This routine uses a global structure to contain the settings and
\ calibration coefficients of each channel.
```

```
0   constant CH0                    \ Constants for channels 0 - 3
1   constant CH1
2   constant CH2
3   constant CH3
320 constant SAMPLE_FREQ            \ freq int corresponding to 60 hz
                                    \ 19200 / 60 = 320 [See Table 5]
40  constant NUM_SAMPLES            \ Total number of samples:
                                    \ 10 samples for 4 channels
4   constant NUM_CHANNELS           \ Num channels we are sampling

array: my_data                      \ Declare an array for samples

structure.begin: ad_channel         \ Config options for each channel
  double-> +ad_zero_cal             \ 24-bit zero scale cal val
  double-> +ad_fs_cal               \ 24-bit full scale cal val
  int->  +ad_freq_int               \ Frequency Integer 19 - 4000.
  byte-> +ad_gain                   \ Gain 1 to 128.
  byte-> +ad_polarity               \ Bipolar or Unipolar mode.
  byte-> +ad_res                    \ Resolution: 16-bit or 24-bit.
  byte-> +ad_bo                     \ Burn out current on/off
  byte-> +ad_fsync                  \ Sync on/off.
  byte-> +ad_ch                     \ Channel.
structure.end

structure.begin: ad_info            \ Global structure.
  ad_channel struct-> +ch0
  ad_channel struct-> +ch1
  ad_channel struct-> +ch2
  ad_channel struct-> +ch3
  byte->  +current_channel          \ Current channel being used.
  int->   +index                    \ Index into data array
structure.end

ad_info v.instance: my_struct       \ Declare a global instance of
                                    \ the structure in variable area.

: Init_CH0 ( -- flag )
  \ Perform a Full Self Calibration on channel 0-1 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_0_1
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch0 +ad_freq_int  !
    GAIN_1        my_struct +ch0 +ad_gain     c!
    BIPOLAR       my_struct +ch0 +ad_polarity c!
    WORD_24BIT    my_struct +ch0 +ad_res      c!
    BO_OFF        my_struct +ch0 +ad_bo       c!
    FSYNC_OFF     my_struct +ch0 +ad_fsync    c!
    CH_0_1        my_struct +ch0 +ad_ch       c!
    Read_Zero_Cal my_struct +ch0 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch0 +ad_fs_cal   2!
    true
  else
    false                       \ Invalid calibration coefficients CH0
```

```
  endif
;

: Init_CH1 ( -- flag )
  \ Perform a Full Self Calibration on channel 2-3 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_2_3
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch1 +ad_freq_int  !
    GAIN_1        my_struct +ch1 +ad_gain     c!
    BIPOLAR       my_struct +ch1 +ad_polarity c!
    WORD_24BIT    my_struct +ch1 +ad_res      c!
    BO_OFF        my_struct +ch1 +ad_bo       c!
    FSYNC_OFF     my_struct +ch1 +ad_fsync    c!
    CH_2_3        my_struct +ch1 +ad_ch       c!
    Read_Zero_Cal my_struct +ch1 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch1 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH1
  endif
;

: Init_CH2 ( -- flag )
  \ Perform a Full Self Calibration on channel 4-5 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_4_5
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch2 +ad_freq_int  !
    GAIN_1        my_struct +ch2 +ad_gain     c!
    BIPOLAR       my_struct +ch2 +ad_polarity c!
    WORD_24BIT    my_struct +ch2 +ad_res      c!
    BO_OFF        my_struct +ch2 +ad_bo       c!
    FSYNC_OFF     my_struct +ch2 +ad_fsync    c!
    CH_4_5        my_struct +ch2 +ad_ch       c!
    Read_Zero_Cal my_struct +ch2 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch2 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH2
  endif
;

: Init_CH3 ( -- flag )
  \ Perform a Full Self Calibration on channel 6-7 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_6_7
  Start_Conversion
  -1 =
  if
```

```
    SAMPLE_FREQ          my_struct +ch3 +ad_freq_int  !
    GAIN_1         my_struct +ch3 +ad_gain     c!
    BIPOLAR        my_struct +ch3 +ad_polarity c!
    WORD_24BIT     my_struct +ch3 +ad_res      c!
    BO_OFF         my_struct +ch3 +ad_bo       c!
    FSYNC_OFF      my_struct +ch3 +ad_fsync    c!
    CH_6_7         my_struct +ch3 +ad_ch       c!
    Read_Zero_Cal my_struct +ch3 +ad_zero_cal 2!
    Read_FS_Cal    my_struct +ch3 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH3
  endif
;

: Do_So_Often (word_xcfa \ ud -- | ud is in ticks of timeslice clock)

\ This word calls another routine periodically, with a fixed time
\ interval between calls of ud ticks of the timeslicer clock.  The
\ routine is designated by word.xcfa and it should return only a flag
\ on the stack. If the flag is true it will continue to be repeatedly
\ executed; as soon as it returns with a false flag this routine stops
\ calling it and returns immediately.  The word.xcfa is called at times

\ 0, ud, 2*ud, 3*ud, etc.. measured in units of timeslicer clock ticks.

\ If the execution time of word.xcfa is greater than ud ticks of the
\ timeslicer then it is just repeatedly called as rapidly as possible.
\ With a 5 msec timeslicer period the interval between calls can be up
\ to 248 days with a resolution of 5 msec.  Because we depend on the
\ timeslicer clock that clock should not be stopped or reset while this
\ routine is running.  To prevent unnoticed rollover if this routine is
\ interrupted by another task, the other task should not take longer
\ than 248 days; that is, control must return to this routine at least
\ once every 248 days.  Also word.xcfa should not take longer than 248
\ days to execute either.  That should generally not be a problem.

\ If another task has control when ud ticks are done and it is time to
\ call word.xcfa then the call to word.xcfa will be delayed until this
\ routine regains control.  However, as long as the other routine and
\ the word.xcfa routine together don't take longer than ud then all
\ subsequent timing will still occur at integer multiples of ud; there
\ is no cumulative timing error.

\ There is a PAUSE which may be removed if you don't want any other
\ tasks to have a chance at machine time.

locals{ d&time_interval x&word_xcfa | d&target_time d&start_time
        d&elapsed_time }
  timeslice.count 2@ to d&start_time \ get the start time
  begin
    d&time_interval to d&target_time
    x&word_xcfa execute              \ execute the user's word
  while           \ we stop repetitively calling the user's word
                  \ when it returns with a false flag
                  \ D&Target.Time and D&Elapsed.Time are measured from
```

```
    D&Start.Time
    begin
      pause
      timeslice.count 2@ 2dup
      d&start_time d- to d&elapsed_time to d&start_time
      d&elapsed_time d&target_time
      du<
    while         \ We readjust the start and target times to maximize
                  \ the time available to other tasks before we
                  \ experience a rollover.  This way the rollover
                  \ horizon is always pushed out to the maximum count.
      d&target_time d&elapsed_time d- to d&target_time
    repeat
    d&start_time d&target_time d+ d&elapsed_time d- to d&start_time
  repeat
;

\ This routine takes one sample, stores it to an array, then starts a
\ conversion for the next channel.
: Get_Sample ( -- flag | done? )
my_struct +current_channel c@          \ Get current channel
my_struct +index            @          \ Get current index
locals{ &index &ch | x&struct_base }

  AD24_Sample_NP                       \ Get sample from a/d
  &index &ch my_data 2!                \ Store to array

  &ch 1 + NUM_CHANNELS <
  if
    &ch 1 +                            \ Increment channel number
    dup
    my_struct +current_channel c!      \ Store to structure
    to &ch                             \ Store to local
  else
    0 my_struct +current_channel c!    \ Roll over channel
    0 to &ch                           \ Roll over local
    &index 1 + my_struct +index !      \ Increment index
  endif

  my_struct &ch ad_channel * xn+       \ Get base address of struct
  to x&struct_base                     \ store to local

  x&struct_base +ad_fs_cal   2@        \ Get settings for next channel
  x&struct_base +ad_zero_cal 2@
  x&struct_base +ad_freq_int  @
  x&struct_base +ad_gain     c@
  x&struct_base +ad_polarity c@
  x&struct_base +ad_res      c@
  x&struct_base +ad_bo       c@
  x&struct_base +ad_fsync    c@
  x&struct_base +ad_ch       c@
  Start_Conv_With_Values

  &ch 1 + &index 1 + * NUM_SAMPLES >=  \ Index and channel start at 0
  if
    false                              \ Done sampling
```

```
  else
    true                                \ Keep going
  endif
;

\ This routine takes 10 samples from 4 sensors at 60 Hz.  All of the
\ settings for each channel are stored in a global structure.  All
\ channels must have the same sampling rate!
: Sample_Routine3 ( -- flag )

  \ Allocate memory for 10 samples from 4 sensors; each sample is
  \ 4 bytes.
  NUM_SAMPLES NUM_CHANNELS / NUM_CHANNELS 2 4 ' my_data dimensioned


  MODULE0 Init_AD24                     \ 24/7 Data Acquisition Module is

                                        \ the first module on the stack
  if
    Use_Onboard_Ref                     \ Use on-board ref for samples

    CH0 my_struct +current_channel c! \ Set ch0 as the current channel
    0   my_struct +index !              \ Init array index number
    Init_CH1                            \ Init global structure
    Init_CH2 or
    Init_CH3 or
    Init_CH0 or                         \ Init ch 0 last since it will be

                                        \ the first channel to be sampled

    \ Get 1 sample every 60 ms.  60 ms is the fastest we can call
    \ Get_Sample because the sample rate is 60Hz and the 24-Bit A/D
    \ takes 3 clock cycles to obtain a sample when using
    \ Start_Conv_With_Values.  This alone is 3/60 or 50 ms.  If a
    \ full Self-Calibration was performed before each conversion, the
    \ fastest rate you could sample one channel would be 10/60 or 166
    \ ms.  This would amount to 666 ms for 4 channels or 1.5 Hz per
    \ channel.
    cfa.for get_sample 12 0 do_so_often \ 12 * 5ms = 60 ms
  endif
;
```

Glossary

FORTH: **AD24_Multiple** (u/xaddr -- flag | u = num samples, xaddr = start_xaddr)
C: int **AD24_Multiple** (uint num_sample, xaddr starting_xaddr)
Acquires u samples from the 24 bit analog to digital (A/D) converter and stores the samples as sequential 32 bit values starting at the specified xaddr within a timeout period or a false flag is returned. Calls `Pause` while waiting for the samples. The timeout period is calculated using the following equation:

$$Timeout(TimesliceCounts) = \frac{NumberOfSamples + CalibrationDelay}{SampleFrequency * TimeslicePeriod}$$

`Init_AD24`, `Use_Onboard_Ref` or `Use_External_Ref`, and `Start_Conversion` or `Start_Conv_With_Values` must be called prior to this routine to initialize, configure, and calibrate the 24/7 Data Acquisition Module.

The maximum number of samples is limited to 8192 because this routine can only store data on a single page. The number of samples is clamped so the routine does not store results past a page boundary. No error flag is returned if an invalid number of samples is specified.

This routine uses an interrupt service routine (ISR) to obtain samples from the A/D. The ISR runs at more than twice the frequency specified to `Start_Conversion` or `Start_Conv_With_Values`. This is done to eliminate clock variations between the A/D and QED clocks and to guarantee that a sample is not missed even if the ISR is delayed up to 1/2 a sample period or one full ISR period. The following equation is used to calculate the sampling period, the maximum acceptable delay time of the ISR, and the maximum time a routine or task can use the SPI:

$$ISRPeriod = \frac{SamplePeriod - InterruptLatencyDelay}{2}$$

The INTERRUPT_LATENCY_DELAY is composed of 200 µs for the time to update the A/D data register with sample values, 10 µs of interrupt latency delay, and 50 µs to read the data ready line when the ISR is entered. The INTERRUPT_LATENCY_DELAY is a constant specified in TCNTs. DO NOT CHANGE THE RESOLUTION OF TCNT!

This routine is not re-entrant. If using in a multitasking system, be sure only one task calls this routine or be sure that separate tasks do not use the AD7714 at the same time.

`AD24_Multiple` takes 1.60 ms to setup the ISR and do the error checking on the number of samples before checking the data ready line.

FORTH: **AD24_Sample** (-- ud | ud = conversion)
C: ulong **AD24_Sample** (void)
Acquires and places on the stack a single sample from the 24 bit analog to digital (A/D) converter within a timeout period or a false flag is returned. The timeout period is calculated using the following equation:

$$Timeout(TimesliceCounts) = \frac{NumberOfSamples + CalibrationDelay}{SampleFrequency * TimeslicePeriod}$$

`Init_AD24`, `Use_Onboard_Ref` or `Use_External_Ref`, and `Start_Conversion` or `Start_Conv_With_Values` must be called prior to this routine to initialize, configure, and calibrate the 24/7 Data Acquisition Module.

This routine disables interrupts, then checks the data ready line (/DRDY) to see if a sample is ready. If the sample is ready, gets the sample and restores interrupts. If the data is not ready, re-enables interrupts, then calls `Pause` before checking the data ready line again. If the sample is not ready before the timeout value, the routine returns a 32 bit value with TIMEOUT_ERROR in the least significant byte to indicate a timeout has occurred. If the data is ready, this routine returns the conversion with 0 stored in the least significant byte. See Figure 3 for a graphical representation of the returned data format for 16 and 24 bit resolutions.

Do not use this routine in an interrupt service. See Example 3 for an example of periodic use.

This routine is not re-entrant. If using in a multitasking system, be sure only one task calls this routine or be sure that separate tasks do not use the AD7714 at the same time.

`AD24_Sample` takes 1.18 ms to read the data ready line and will obtain a sample within 1.38 ms. The variation between the two times is due to the 200 µs time when the data ready line is high when the AD7714 is updating its output register. The execution time of this routine is 1.64 ms.

FORTH: **AD24_Sample_NP** (-- ud | ud = conversion)
C: ulong AD24_Sample_NP (void)
Identical to `AD24_Sample` except does not disable interrupts and does not call `Pause`.

Do not use this routine in an interrupt service. See Example 3 for an example of periodic use.

This routine is not re-entrant. If using in a multitasking system, be sure only one task calls this routine or be sure that separate tasks do not use the AD7714 at the same time.

FORTH: **Buffer_Off** (--)
C: void **Buffer_Off** (void)
Turns the buffered input off. This is the default state after power ups and resets.

FORTH: **Buffer_On** (--)
C: void **Buffer_On** (void)
Allows high source impedances to be used at the analog field inputs by buffering the input signal through an amplifier. This results in a small DC offset voltage developed across the source impedance but not a gain error since the offset current is fixed at 1 nA. The offset voltage can be calculated using Equation 1. With the buffer on, the input voltage is also limited to the range 50 mV - 3.5 V.

FORTH: **Init_AD24** (b -- flag | b = module number)
C: int **Init_AD24** (char module_number)
Initializes the 24/7 Data Acquisition Module by starting the timeslicer, turning on the SPI, disconnecting the analog field inputs from the analog to digital converter, and resetting the analog to digital converter. Returns a false flag if the module number is not valid. Valid modules are MODULE0, MODULE1, MODULE2, MODULE3, MODULE4, MODULE5, MODULE6, and MODULE7. This routine does not change the SPI settings. The SPI settings are changed before each read or write operation to the analog to digital converter and then restored when the operation is complete. This module is fully compatible with any routine that uses the SPI unless the SPI resource is "gotten" and not "released".

FORTH: **Read_Digital_IO** (-- nibble)
C: char **Read_Digital_IO** (void)
Reads the 3 digital I/O lines on the 24/7 Data Acquisition Module. The data is returned in the least significant nibble. The least significant bit of the returned byte indicates the status of the data ready line as shown in the following figure. The other three bits are uncommitted.

Bit 3	Bit 2	Bit 1	Bit 1
I19 (Pin 3 of H2)	I18 (Pin 6 of H2)	I17 (Pin 5 of H2)	\DRDY

FORTH: **Read_FS_Cal** (-- ud | ud = fs cal value)
C: ulong **Read_FS_Cal** (void)
Reads the full scale calibration information from the analog to digital converter for the current channel and settings. `Start_Conversion` must be called

before attempting to read the full scale calibration information or invalid calibration information will be returned.

This routine checks the data ready line before reading the calibration coefficient. It checks the data ready line to make sure that a calibration is not occurring. If a calibration is occurring, this routine will wait up to 10 sample periods (the delay associated with a Full Self Calibration). If the data ready line does not return low within the delay time, it will return a timeout error.

FORTH: **Read_Zero_Cal** (-- ud | ud = zero cal value)
C: ulong **Read_Zero_Cal** (void)
Reads the zero scale calibration information from the analog to digital converter for the current channel and settings. `Start_Conversion` must be called before attempting to read the zero calibration information or invalid calibration information will be returned.

This routine checks the data ready line before reading the calibration coefficient. It checks the data ready line to make sure that a calibration is not occurring. If a calibration is occurring, this routine will wait up to 10 sample periods (the delay associated with a Full Self Calibration). If the data ready line does not return low within the delay time, it will return a timeout error.

FORTH: **Reset_AD24** (--)
C: void **Reset_AD24** (void)
Resets the digital filter, the analog modulator, and the on-chip registers of the analog to digital converter, disconnects all analog field inputs (not including the reference inputs), and turns the input buffer off. `Start_Conversion` or `Start_Conv_With_Values` must be executed after this routine is called to start another conversion.

FORTH: **Software_Reset** (--)
C: void **Software_Reset** (void)
Resets the serial interface of the analog to digital converter to a known state. This routine does not reset contents of any registers but if the interface was lost, it is advisable to set up all registers again using `Start_Conversion` or `Start_Conv_With_Values`. This routine does not disconnect the analog field inputs from the analog to digital converter or turn the buffer off.

FORTH: **Start_Conv_With_Values** (ud1\ud2\u\b1\b2\b3\b4\b5\b6 --)
C: void **Start_Conv_With_Values** (ulong fs_cal_val, ulong zero_cal_val, uint freq_int, char gain, char pol, char res, char bo, char fysnc, char ch)
Starts a conversion without a calibration. This routine loads the calibration registers with the specified calibration values and options and starts a conversion. `Init_AD24`, `Use_Onboard_Ref` or `Use_External_Ref`, and `Start_Conversion` must be called prior to this routine to initialize, configure, and calibrate the 24/7 Data Acquisition Module. This routine can be

used only if a calibration has already been performed on the specified channel with the specified settings and the calibration values were recorded. This will save up to 10 x sample rate because no calibration is performed. No error checking is done on any of the parameters to allow fast execution when switching between channels. The parameters and their allowed values are:

Parameter (Forth/C)	Description	Valid Values
ud1/fs_cal_val	Full Scale Calibration Value	Value returned by Read_FS_Cal
ud2/zero_cal_val	Zero Scale Calibration Value	Value returned by Read_Zero_Cal
u/freq_int	Frequency Integer	19-4000*
b1/gain	Sample Gain	GAIN_1, GAIN_2, GAIN_4, GAIN_8, GAIN_16, GAIN_32, GAIN_64, GAIN_128
b2/pol	Signal Polarity	UNIPOLAR, BIPOLAR
b3/res	Sample Resolution	WORD_16BIT, WORD_24BIT
b4/bo	Burnout Current Option	BO_ON, BO_OFF
b5/fsync	Synchronization Option	FSYNC_ON, FSYNC_OFF
b6/ch	Channel Number	CH_0_7, CH_1_7, CH_2_7, CH_3_7, CH_4_7, CH_5_7, CH_0_1, CH_2_3, CH_4_5, CH_6_7.
flag	Error Flag	INVALID_GAIN=1, INVALID_FREQ=2, INVALID_CAL=3, INVALID_CHANNEL=4, INVALID_FSYNC=5, INVALID_BO=6, INVALID_SIZE=7, INVALID_POLARITY=8, TIMEOUT_ERROR=9

*The sample frequency in Hertz is calculated by dividing 19200 by the frequency integer number.

This routine is not re-entrant. If using in a multitasking system, be sure only one task calls this routine or be sure that separate tasks do not use the AD7714 at the same time.

This routine takes 2.34 ms to start a conversion and 7.00 ms to execute.

FORTH: **Start_Conversion** (b1\u\b2\b3\b4\b5\b6 -- flag)
C: int **Start_Conversion**(char cal_type, uint freq_int, char gain, char pol, char res, char bo, char ch)

Starts a conversion by configuring the specified channel with the specified options. `Init_AD24` and `Use_Onboard_Ref` or `Use_External_Ref` must be called prior to this routine to initialize the 24/7 Data Acquisition Module. Error checking is done for each option and an error flag is returned if an invalid parameter is used. The parameters and their allowed values are:

Parameter (Forth/C)	Description	Valid Values
b1/cal_type	Calibration Type	SELF_CAL, ZERO_SELF_CAL, FS_SELF_CAL, BACKGND_CAL, ZERO_SYS_CAL, FS_SYS_CAL, SYS_OFFSET_CAL
u/freq_int	Frequency Integer	19-4000*
B2/gain	Sample Gain	GAIN_1, GAIN_2, GAIN_4, GAIN_8, GAIN_16, GAIN_32, GAIN_64, GAIN_128
b3/pol	Signal Polarity	UNIPOLAR, BIPOLAR
b4/res	Sample Resolution	WORD_16BIT, WORD_24BIT
b5/bo	Burnout Current Option	BO_ON, BO_OFF
b6/ch	Channel Number	CH_0_7, CH_1_7, CH_2_7, CH_3_7, CH_4_7, CH_5_7, CH_0_1, CH_2_3, CH_4_5, CH_6_7.
Flag	Error Flag	INVALID_GAIN = 1, INVALID_FREQ = 2, INVALID_CAL = 3, INVALID_CHANNEL = 4, INVALID_FSYNC = 5, INVALID_BO = 6, INVALID_SIZE = 7, INVALID_POLARITY = 8, TIMEOUT_ERROR = 9

*The sample frequency in Hertz is calculated by dividing 19200 by the frequency integer number.

This routine is not re-entrant. If using in a multitasking system, be sure only one task calls this routine or be sure that separate tasks do not use the AD7714 at the same time.

This routine takes 3.08 ms to start a conversion and 5.08 ms to execute.

FORTH: **Stop_Conversion** (--)
C: void **Stop_Conversion** (void)

Disconnects the 24 bit analog to digital converter from its analog field inputs including the voltage references. Called in `Init_AD24` and `Reset_AD24`.

FORTH: **Sync** (--)
C: void **Sync** (void)
Resets the modulator and digital filter of the analog to digital converter without affecting any of the setup conditions. This allows you to start gathering samples from the analog input from a known point in time (once the routine is executed). 3 sample periods must elapse before the next valid sample is available. See pages 26 and 27 of the data sheet for more information.

FORTH: **Use_External_Ref** (--)
C: void **Use_External_Ref** (void)
Connects pins 9 (FDREF+) and 11 (FDREF-) as the 24 bit analog to digital converter's reference and disconnects the on-board reference voltage.

FORTH: **Use_Onboard_Ref** (--)
C: void **Use_Onboard_Ref** (void)
Connects the on-board 2.5 volt reference to the 24 bit analog to digital converter and disconnects the external reference voltage.

Appendix A: 24/7 Data Acquisition Module Pinouts

The pinouts of all of the connectors on the 24/7 Data Acquisition Module are presented below. To locate the connectors on the board, consult the white silk-screened labels located near each connector. Pin 1 and pin 24 are identified on each side and on each connector.

Note that some signals have compound names to suggest multiple functions. For example, SEL0/XMIT+ on the Module Bus is a signal that can be used as a module select line or the RS485 positive transmit line. The underlined signals are used by the 24/7 Data Acquisition Module while the signals with an asterisk are not.

Module Bus

GND	-	1	2	-	+5V*
*/IRQ	-	3	4	-	V+RAW
SEL1/XMIT-	-	5	6	-	SEL0/XMIT+
MOSI/XCV-	-	7	8	-	MISO/XCV+
/RESET	-	9	10	-	SCK
/MOD.CS	-	11	12	-	16MHZ*
E	-	13	14	-	R//W*
/OE	-	15	16	-	/WE
AD7	-	17	18	-	AD6
AD5	-	19	20	-	AD4
AD3	-	21	22	-	AD2
AD1	-	23	24	-	AD0

Field Bus

GND	-	1	2	-	+5V
I19	-	3	4	-	V+RAW
I17	-	5	6	-	I18
REF+	-	7	8	-	VAN
FDREF+	-	9	10	-	FIELD7
FDREF-	-	11	12	-	FIELD6
REF-	-	13	14	-	FIELD5
AGND	-	15	16	-	FIELD4
AGND	-	17	18	-	FIELD3
AGND	-	19	20	-	FIELD2
AGND	-	21	22	-	FIELD1
AGND	-	23	24	-	FIELD0

Appendix B: 24/7 Data Acquisition Module Schematics