Section 31. 10-bit 2 Msps Analog-to-Digital Converter (ADC)

HIGHLIGHTS

This section of the manual contains the following major topics:

31.1 Introduction

The 10-bit 2 Msps Analog-to-Digital Converter (ADC) has these key features:

- 10-bit resolution
- Successive Approximation (SAR) conversion
- 2 Msps conversion speed at 5V
- 1 Msps conversion speed at 3V
- Up to 12 analog input pins
- Up to 5 Sample and Hold (S/H) circuits (4 dedicated, 1 shared)
- Unipolar Inputs
- ±1 Least Significant bit (LSb) maximum Differential Non Linearity (DNL)
- ±1 LSb maximum Integral Non Linearity (INL)
- No operation during Sleep mode

A block diagram of the 10-bit 2 Msps ADC module is shown in Figure 31-1. The 10-bit 2 Msps ADC is designed for applications that require low latency between the request for conversion and the resultant output data. Typical applications include the following:

- AC/DC power supplies
- DC/DC converters
- Power factor correction

Up to five inputs can be sampled simultaneously, and up to 12 inputs can request conversion at a time. If multiple inputs request conversion, the 10-bit 2 Msps ADC converts them sequentially starting with the lowest order input.

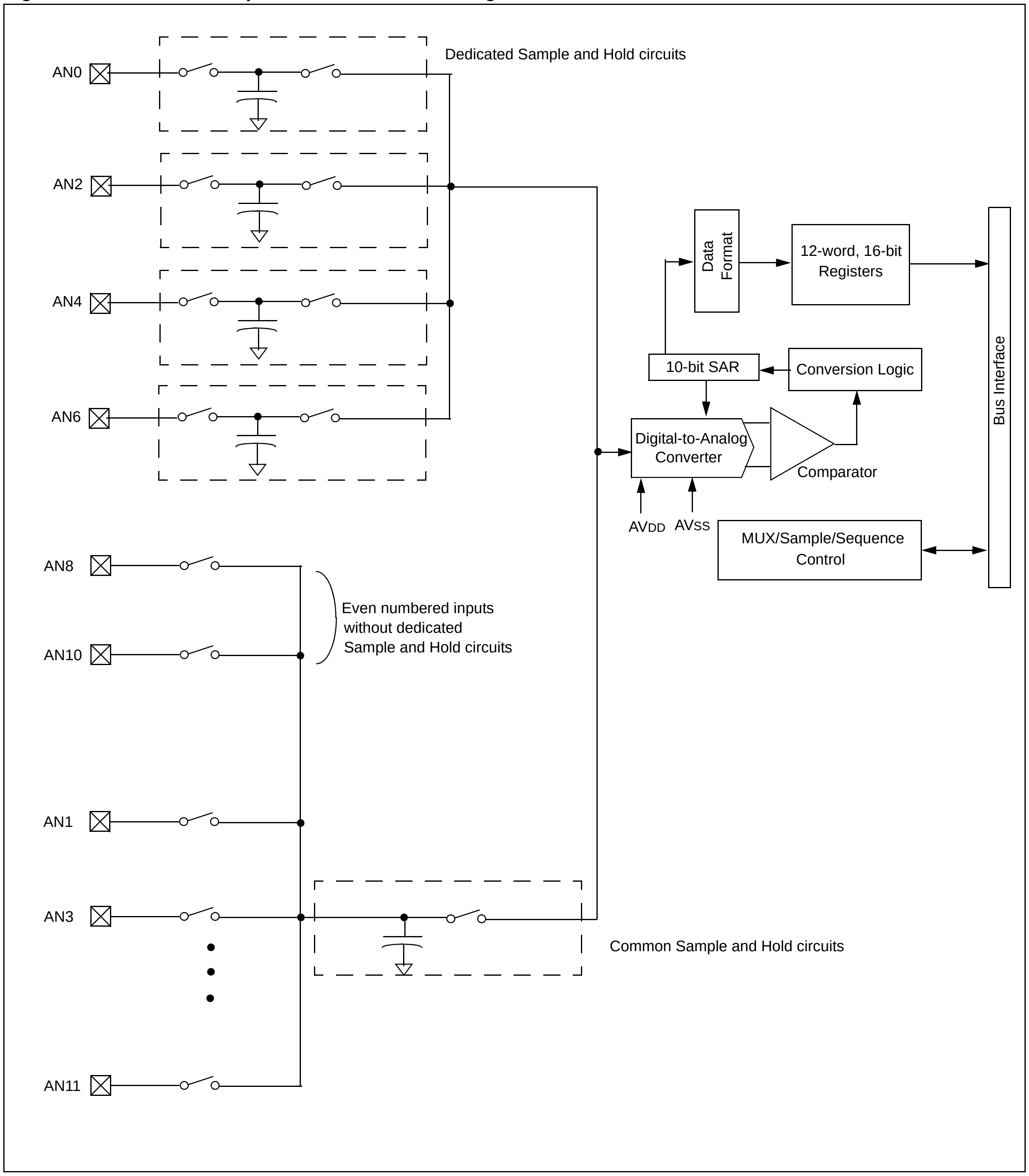
The actual number of analog input pins and the external voltage reference input configuration will depend on the specific device. Refer to the device data sheet for further details.

Power supply applications that use PWM technology typically require synchronization between analog data sampling and the PWM outputs. The design of this 10-bit 2 Msps ADC provides each pair of analog inputs (AN1, AN0), (AN3, AN2), (…) the ability to specify its own trigger source from among 16 different trigger sources. This capability allows the ADC to sample and convert analog inputs that are associated with PWM generators operating on independent time bases. The high-speed operation of this ADC module allows for data on demand.

In addition, the peripheral interface includes several hardware features to improve real-time performance in a typical DSP-based application:

- Result alignment options
- Automated sampling
- External conversion start control

Figure 31-1: 10-bit 2 Msps ADC Module Block Diagram


Dedicated Sample and Hold circuits
AN0
AN2
AN4
AN6
Data Format
12-word, 16-bit Registers
10-bit SAR
Conversion Logic
Bus Interface
Digital-to-Analog Converter
Comparator
AVDD
AVSS
MUX/Sample/Sequence Control
AN8
AN10
Even numbered inputs without dedicated Sample and Hold circuits
AN1
AN3
Common Sample and Hold circuits
AN11

31.2 Control Registers

The 10-bit 2 Msps ADC module has seven Control and Status registers. These registers are as follows:

- ADCON: ADC Control Register
- ADSTAT: ADC Status Register[1]
- ADBASE: ADC Base Register
- ADPCFG: ADC Port Configuration Register
- ADCPC0: ADC Pair Control Register 0
- ADCPC1: ADC Pair Control Register 1
- ADCPC2: ADC Pair Control Register 2

The ADCON register controls the operation of the ADC module. The ADSTAT register provides the Conversion Data Ready status bits for every ADC input channel pair. The ADBASE register provides a scheme to implement Pair Conversion Complete interrupts using a jump table. The ADPCFG register configures the ADC input channel as analog inputs or as digital I/O. The ADCPC0, ADCPC1 and ADCPC2 registers control the individual ADC input conversion pair options: trigger selection, interrupt enable and software trigger.

31.3 ADC Result Buffer

The 10-bit 2 Msps ADC module contains up to 12 data output registers (ADBUF0-ADBUF11) that store the A/D conversion results. These registers are 10 bits wide, but are read as 16-bit words, which are configured by the ADCON register as either Fractional or Integer format. Each analog input has a corresponding data output register. For example, AN0 conversion results are always stored in ADBUF0, AN1 conversion results are always stored in ADBUF1, and so forth.

The ADC write to ADBUFx registers is synchronous with the ADC clock. Reads of the buffer will always yield valid data assuming that the Data Ready interrupt has been cleared. If the buffer location has not been read by the software and the A/D needs to write to the buffer, the previous conversion results will be lost.

Note: The ADC result buffer is a read-only buffer.

Register 31-1: ADCON: ADC Control Register

R/W-0	U-0	R/W-0	U-0	U-0	R/W-0	U-0	R/W-0
ADON	—	ADSIDL	—	—	GSWTRG	—	FORM
bit 15							bit 8

R/W-0	R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-1	R/W-1
EIE	ORDER	SEQSAMP	—	—	ADCS<2:0>		
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15 **ADON:** ADC Operating Mode bit
1 = ADC module is operating
0 = ADC module is off

bit 14 **Unimplemented:** Read as ‘0’

bit 13 **ADSIDL:** Stop in Idle Mode bit
1 = Discontinue module operation when device enters Idle mode
0 = Continue module operation when device enters Idle mode

bit 12-11 **Unimplemented:** Read as ‘0’

bit 10 **GSWTRG:** Global Software Trigger bit
When this bit is set by the user application, it triggers conversions if selected by the TRGSRC<4:0> bits in the ADCPCx registers. This bit is cleared by the hardware when the first conversion has been completed.

bit 9 **Unimplemented:** Read as ‘0’

bit 8 **FORM:** Data Output Format bit
1 = Fractional (DOUT = `dddd dddd dd00 0000`)
0 = Integer (DOUT = `0000 00dd dddd dddd`)

bit 7 **EIE:** Early Interrupt Enable bit
1 = Interrupt is generated after first conversion is completed
0 = Interrupt is generated after second conversion is completed
Note: This control bit can only be changed while ADC is disabled (ADON = 0).

bit 6 **ORDER:** Conversion Order bit
1 = Odd numbered analog input is converted first, followed by conversion of even numbered input.
0 = Even numbered analog input is converted first, followed by conversion of odd numbered input.
Note: This control bit can only be changed while ADC is disabled (ADON = 0).

bit 5 **SEQSAMP:** Sequential Sample Enable bit
1 = Shared S/H is sampled at the start of the second conversion if ORDER = 0, or is sampled at the start of the first conversion if ORDER = 1.
0 = Shared S/H is sampled at the same time the dedicated S/H is sampled if the shared S/H is not currently busy with an existing conversion process. If the shared S/H is busy at the time the dedicated S/H is sampled, then the shared S/H will sample at the start of the new conversion cycle.

bit 4-3 **Unimplemented:** Read as ‘0’

bit 2-0 **ADCS<2:0>:** ADC Conversion Clock Divider Select bits
Configures the divisor used to determine ADC conversion frequency
000 = FADC/4
001 = FADC/6
010 = FADC/8
011 = FADC/10 (default)
100 = FADC/12
101 = FADC/14
110 = FADC/16
111 = FADC/18

Register 31-2: ADSTAT: ADC Status Register[1]

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	R/C-0 H-S	R/C-0 H-S	R/C-0 H-S	R/C-0 H-S	R/C-0 H-S	R/C-0 H-S
—	—	P5RDY	P4RDY	P3RDY	P2RDY	P1RDY	P0RDY
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown
C = Clear in software	H-S = Set by hardware		

bit 15-6	**Unimplemented:** Read as ‘0’
bit 5	**P5RDY:** Conversion Data for Pair 5 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.
bit 4	**P4RDY:** Conversion Data for Pair 4 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.
bit 3	**P3RDY:** Conversion Data for Pair 3 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.
bit 2	**P2RDY:** Conversion Data for Pair 2 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.
bit 1	**P1RDY:** Conversion Data for Pair 1 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.
bit 0	**P0RDY:** Conversion Data for Pair 0 Ready bit Bit set when data is ready in buffer, cleared when a ‘0’ is written to this bit.

Note 1: The PxRDY bit for each conversion pair must be cleared before the subsequent conversion can be detected. If the bit is not cleared, the conversion request is processed, but an interrupt is not generated.

Register 31-3: ADBASE: ADC Base Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADBASE<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0
ADBASE<7:1>							—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15-1 **ADBASE Register:** This register contains the base address of the user’s ADC Interrupt Service Routine jump table. This register, when read, contains the sum of the ADBASE register contents and the encoded value of the PxRDY Status bits.

The encoder logic provides the bit number of the highest priority PxRDY bits, where P0RDY is the highest priority, and P5RDY is the lowest priority.

Note: The encoding results are shifted left two bits, so bits 1-0 of the result are always zero.

bit 0 **Unimplemented:** Read as ‘0’

Register 31-4: ADPCFG: ADC Port Configuration Register

U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	PCFG11	PCFG10	PCFG9	PCFG8
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15-12 **Unimplemented:** Read as ‘0’

bit 11-0 **PCFG<11:0>:** A/D Port Configuration Control bits

1 = Port pin in Digital mode; port read input enabled; A/D input multiplexor connected to AVSS.
0 = Port pin in Analog mode; port read input disabled; A/D samples pin voltage.

Register 31-5: ADCPC0: ADC Pair Control Register 0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN1	PEND1	SWTRG1	TRGSRC1<4:0>				
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN0	PEND0	SWTRG0	TRGSRC0<4:0>				
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15 **IRQEN1:** Interrupt Request Enable 1 bit
1 = Enable IRQ generation when requested conversion of channels AN3 and AN2 is completed.
0 = IRQ is not generated.

bit 14 **PEND1:** Pending Conversion Status 1 bit
1 = Conversion of channels AN3 and AN2 is pending. Set when selected trigger is asserted.
0 = Conversion of channels AN3 and AN2 is complete.

bit 13 **SWTRG1:** Software Trigger 1 bit
1 = Start conversion of channels AN3 and AN2 (if TRGSRC1 = 00001). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND1 bit is set.

bit 12-8 **TRGSRC1<4:0>:** Trigger 1 Source Selection bits
Selects trigger source for conversion of analog channels AN3 and AN2.
00000 = No conversion enabled
00001 = Individual software trigger selected
00010 = Global software trigger selected
00011 = PWM special event trigger selected
00100 = PWM generator 1 trigger selected
00101 = PWM generator 2 trigger selected
00110 = PWM generator 3 trigger selected
00111 = PWM generator 4 trigger selected
01100 = Timer 1 period match
01101 = Timer 2 period match
01110 = PWM generator 1 current-limit ADC trigger
01111 = PWM generator 2 current-limit ADC trigger
10000 = PWM generator 3 current-limit ADC trigger
10001 = PWM generator 4 current-limit ADC trigger
10110 = PWM generator 1 fault ADC trigger
10111 = PWM generator 2 fault ADC trigger
11000 = PWM generator 3 fault ADC trigger
11001 = PWM generator 4 fault ADC trigger

bit 7 **IRQEN0:** Interrupt Request (IRQ) Enable 0 bit
1 = Enable IRQ generation when requested conversion of channels AN1 and AN0 is completed.
0 = IRQ is not generated.

bit 6 **PEND0:** Pending Conversion Status 0 bit
1 = Conversion of channels AN1 and AN0 is pending. Set when selected trigger is asserted.
0 = Conversion of channels AN1 and AN0 is complete.

bit 5 **SWTRG0:** Software Trigger 0 bit
1 = Start conversion of AN1 and AN0 (if TRGSRC0 = 00001). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND0 bit is set.

Register 31-5: ADCPC0: ADC Pair Control Register 0 (Continued)

bit 4-0	**TRGSRC0<4:0>:** Trigger 0 Source Selection bits Selects trigger source for conversion of analog channels AN1 and AN0. `00000` = No conversion enabled `00001` = Individual software trigger selected `00010` = Global software trigger selected `00011` = PWM Special Event Trigger selected `00100` = PWM generator 1 trigger selected `00101` = PWM generator 2 trigger selected `00110` = PWM generator 3 trigger selected `00111` = PWM generator 4 trigger selected `01100` = Timer 1 period match `01101` = Timer 2 period match `01110` = PWM generator 1 current-limit ADC trigger `01111` = PWM generator 2 current-limit ADC trigger `10000` = PWM generator 3 current-limit ADC trigger `10001` = PWM generator 4 current-limit ADC trigger `10110` = PWM generator 1 fault ADC trigger `10111` = PWM generator 2 fault ADC trigger `11000` = PWM generator 3 fault ADC trigger `11001` = PWM generator 4 fault ADC trigger

Register 31-6: ADCPC1: ADC Pair Control Register 1

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN3	PEND3	SWTRG3	TRGSRC3<4:0>				
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN2	PEND2	SWTRG2	TRGSRC2<4:0>				
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15 **IRQEN3:** Interrupt Request Enable 3 bit
`1` = Enable IRQ generation when requested conversion of channels AN7 and AN6 is completed.
`0` = IRQ is not generated.

bit 14 **PEND3:** Pending Conversion Status 3 bit
`1` = Conversion of channels AN7 and AN6 is pending. Set when selected trigger is asserted.
`0` = Conversion is complete.

bit 13 **SWTRG3:** Software Trigger 3 bit
`1` = Start conversion of AN7 and AN6 (if TRGSRC3 = `00001`). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND3 bit is set.

bit 12-8 **TRGSRC3<4:0>:** Trigger 3 Source Selection bits
Selects trigger source for conversion of analog channels AN7 and AN6.
`00000` = No conversion enabled
`00001` = Individual software trigger selected
`00010` = Global software trigger selected
`00011` = PWM special event trigger selected
`00100` = PWM generator 1 trigger selected
`00101` = PWM generator 2 trigger selected
`00110` = PWM generator 3 trigger selected
`00111` = PWM generator 4 trigger selected
`01100` = Timer 1 period match
`01101` = Timer 2 period match
`01110` = PWM generator 1 current-limit ADC trigger
`01111` = PWM generator 2 current-limit ADC trigger
`10000` = PWM generator 3 current-limit ADC trigger
`10001` = PWM generator 4 current-limit ADC trigger
`10110` = PWM generator 1 fault ADC trigger
`10111` = PWM generator 2 fault ADC trigger
`11000` = PWM generator 3 fault ADC trigger
`11001` = PWM generator 4 fault ADC trigger

bit 7 **IRQEN2:** Interrupt Request Enable 2 bit
`1` = Enable IRQ generation when requested conversion of channels AN5 and AN4 is completed.
`0` = IRQ is not generated.

bit 6 **PEND2:** Pending Conversion Status 2 bit
`1` = Conversion of channels AN5 and AN4 is pending. Set when selected trigger is asserted.
`0` = Conversion is complete.

bit 5 **SWTRG2:** Software Trigger 2 bit
`1` = Start conversion of AN5 and AN4 (if TRGSRC2 = `00001`). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND2 bit is set.

Register 31-6: ADCPC1: ADC Pair Control Register 1 (Continued)

bit 4-0	**TRGSRC2<4:0>:** Trigger 2 Source Selection bits Selects trigger source for conversion of analog channels AN5 and AN4. `00000` = No conversion enabled `00001` = Individual software trigger selected `00010` = Global software trigger selected `00011` = PWM special event trigger selected `00100` = PWM generator 1 trigger selected `00101` = PWM generator 2 trigger selected `00110` = PWM generator 3 trigger selected `00111` = PWM generator 4 trigger selected `01100` = Timer 1 period match `01101` = Timer 2 period match `01110` = PWM generator 1 current-limit ADC trigger `01111` = PWM generator 2 current-limit ADC trigger `10000` = PWM generator 3 current-limit ADC trigger `10001` = PWM generator 4 current-limit ADC trigger `10110` = PWM generator 1 fault ADC trigger `10111` = PWM generator 2 fault ADC trigger `11000` = PWM generator 3 fault ADC trigger `11001` = PWM generator 4 fault ADC trigger

Register 31-7: ADCPC2: ADC Pair Control Register 2

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN5	PEND5	SWTRG5	TRGSRC5<4:0>				
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IRQEN4	PEND4	SWTRG4	TRGSRC4<4:0>				
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as ‘0’	
-n = Value at POR	‘1’ = Bit is set	‘0’ = Bit is cleared	x = Bit is unknown

bit 15 **IRQEN5:** Interrupt Request Enable 5 bit
1 = Enable IRQ generation when requested conversion of channels AN11 and AN10 is completed.
0 = IRQ is not generated.

bit 14 **PEND5:** Pending Conversion Status 5 bit
1 = Conversion of channels AN11 and AN10 is pending. Set when selected trigger is asserted.
0 = Conversion is complete.

bit 13 **SWTRG5:** Software Trigger 5 bit
1 = Start conversion of AN11 and AN10 (if TRGSRC5 = 00001). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND5 bit is set.

bit 12-8 **TRGSRC5<4:0>:** Trigger Source Selection 5 bits
Selects trigger source for conversion of analog channels A11 and A10.
00000 = No conversion enabled
00001 = Individual software trigger selected
00010 = Global software trigger selected
00011 = PWM special event trigger selected
00100 = PWM generator 1 trigger selected
00101 = PWM generator 2 trigger selected
00110 = PWM generator 3 trigger selected
00111 = PWM generator 4 trigger selected
01100 = Timer 1 period match
01101 = Timer 2 period match
01110 = PWM generator 1 current-limit ADC trigger
01111 = PWM generator 2 current-limit ADC trigger
10000 = PWM generator 3 current-limit ADC trigger
10001 = PWM generator 4 current-limit ADC trigger
10110 = PWM generator 1 fault ADC trigger
10111 = PWM generator 2 fault ADC trigger
11000 = PWM generator 3 fault ADC trigger
11001 = PWM generator 4 fault ADC trigger

bit 7 **IRQEN4:** Interrupt Request Enable 4 bit
1 = Enable IRQ generation when requested conversion of channels AN9 and AN8 is completed.
0 = IRQ is not generated.

bit 6 **PEND4:** Pending Conversion Status 4 bit
1 = Conversion of channels AN9 and AN8 is pending. Set when selected trigger is asserted.
0 = Conversion is complete.

bit 5 **SWTRG4:** Software Trigger 4 bit
1 = Start conversion of AN9 and AN8 (if TRGSRC4 = 00001). If other conversions are in progress, then conversion is performed when the conversion resources are available. This bit is reset when the PEND4 bit is set.

Register 31-7: ADCPC2: ADC Pair Control Register 2 (Continued)

bit 4-0	**TRGSRC4<4:0>:** Trigger Source Selection 4 bits Selects trigger source for conversion of analog channels AN9 and AN8. `00000` = No conversion enabled `00001` = Individual software trigger selected `00010` = Global software trigger selected `00011` = PWM special event trigger selected `00100` = PWM generator 1 trigger selected `00101` = PWM generator 2 trigger selected `00110` = PWM generator 3 trigger selected `00111` = PWM generator 4 trigger selected `01100` = Timer 1 period match `01101` = Timer 2 period match `01110` = PWM generator 1 current-limit ADC trigger `01111` = PWM generator 2 current-limit ADC trigger `10000` = PWM generator 3 current-limit ADC trigger `10001` = PWM generator 4 current-limit ADC trigger `10110` = PWM generator 1 fault ADC trigger `10111` = PWM generator 2 fault ADC trigger `11000` = PWM generator 3 fault ADC trigger `11001` = PWM generator 4 fault ADC trigger

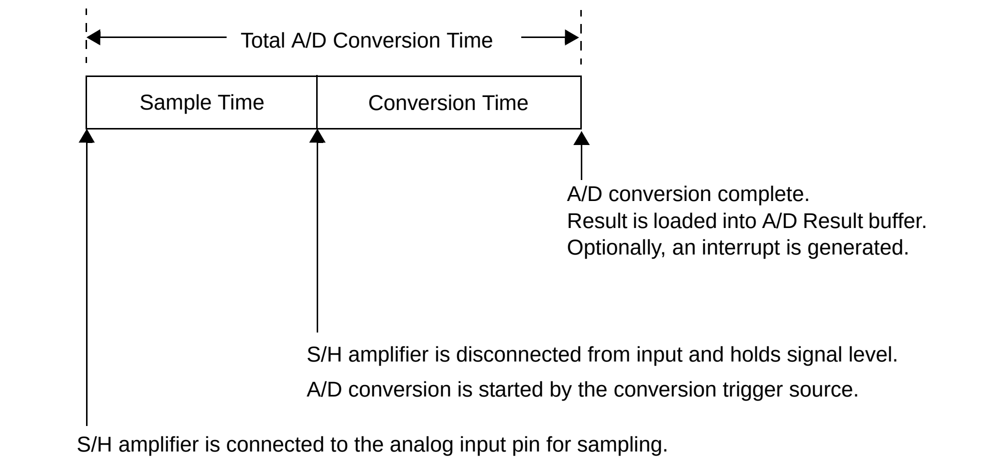
31.4 A/D Sample and Conversion

Figure 31-2 shows a basic conversion sequence. Sample time is the time that the S/H circuit is connected to the analog input pin. Conversion time is the time required for the ADC to convert the voltage held by the S/H circuit.

A total of 12 clock cycles (TAD) are required to convert one A/D channel. The ADC requires 2 TAD to sample the input and 10 TAD for the conversion process. Since the ADC module performs conversion on a pair of channels, a complete A/D conversion process takes 24 clock cycles. When the conversion is complete, the result is loaded into one of 12 A/D Result buffers (ADBUF0… ADBUF11) and the channel is ready to be sampled again. Optionally, a CPU interrupt can be generated at that time. The A/D clock value can be set by selecting ADCS bits in the ADCON register. When operating at the maximum conversion rate of 2 Msps per channel, the sampling period is 2 * 41.6 nsec = 83.3 nsec.

Every ADC pair specified in the A/D Convert Pair Control (ADCPCx) registers initiates a sample operation when the selected trigger event occurs. The conversion of the sampled value occurs as resources become available. If a newer trigger event occurs for a specific channel before a previous sample and convert request for the channel has been processed, the newer request is ignored. To avoid this condition, the conversion rate capability of the module must not be exceeded.

Figure 31-2: A/D Sample and Conversion Sequence


Total A/D Conversion Time
Sample Time
Conversion Time
A/D conversion complete.
Result is loaded into A/D Result buffer.
Optionally, an interrupt is generated.
S/H amplifier is disconnected from input and holds signal level.
A/D conversion is started by the conversion trigger source.
S/H amplifier is connected to the analog input pin for sampling.

31.5 ADC Conversion Pair

As shown in the ADC module block diagram (Figure 31-1), analog input voltage is measured by Sample and Hold (S/H) circuits (also called S/H channels). The ADC module on dsPIC30F SMPS devices has up to five S/H channels and up to 12 analog inputs (AN0-AN11), depending on the specific device. The first four even numbered analog inputs (AN0, AN2, AN4 and AN6) have dedicated S/H channels. The remaining analog inputs are multiplexed to the remaining (common) S/H channel. The common S/H channel serves up to a maximum of 8 analog inputs. The exact number of analog inputs and S/H channels depends on the device type. Refer to the device data sheet for more information.

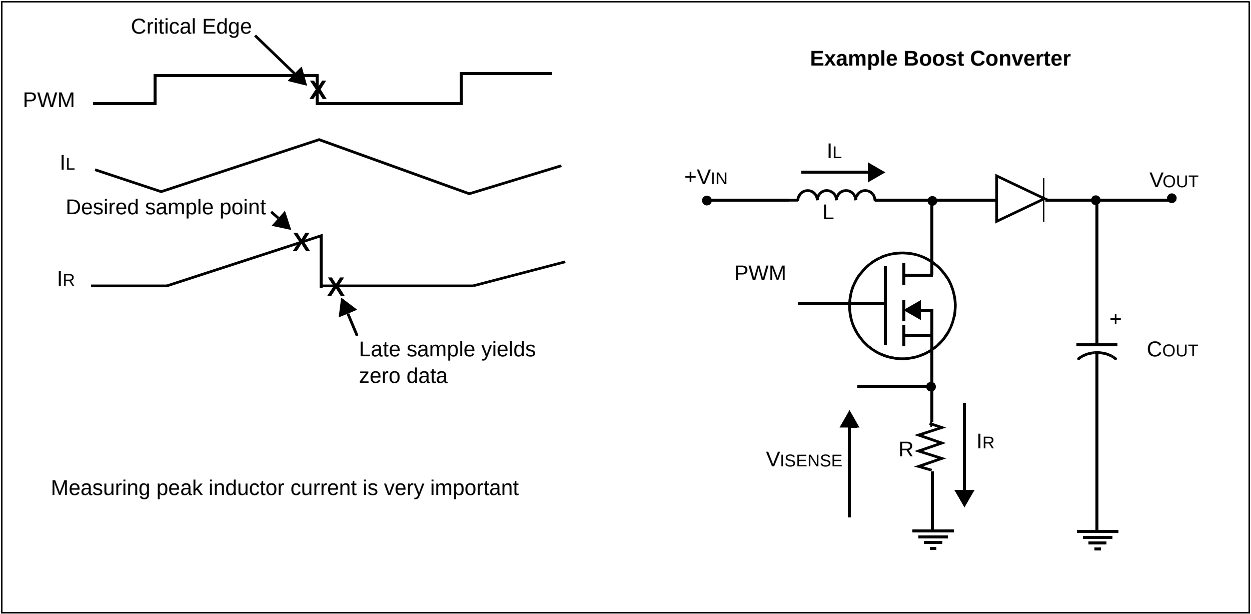
The ADC module converts analog inputs in pairs. A conversion pair is a combination of an even and odd input, such as inputs AN0 and AN1, AN2 and AN3, and so on. A pair can be served by just the common S/H amplifier (for example, inputs AN9 and AN8) or by both the common and the dedicated S/H channels (for example, inputs AN1 and AN0). The technique of using conversion pairs is particularly useful in power conversion applications where there is a need to measure voltages and currents for each PWM control loop. The ADC module enables the sample and conversion process of each conversion pair to be precisely timed relative to the PWM signals.

Figure 31-3 shows a typical power conversion application where the current through inductor L is sensed by monitoring the voltage (V_{SENSE}) across resistor R in series with the power transistor that charges the inductor. The PWM signal enables the transistor, which allows inductor L to charge to a desired current level. The longer the PWM signal is on, the longer the inductor charges, as shown in the timing diagram. Therefore, the inductor current is at its maximum at the end of the PWM signal. Often, this is the point where the current and voltage measurements need to be taken. If the sampling of the resistor voltage occurs slightly later than the desired sample point, the data read will be zero. This is not acceptable in most applications. Interaction with the PWM module lets the user choose when, during the PWM period, the ADC takes a sample. The dedicated sample and hold circuit insures that a sample is taken for a particular channel at the designated time. The S/H circuit will never be busy storing a sample for another channel.

Each conversion pair can be configured to be triggered from the following sources:

- Individual or global software triggers
- PWM Special Event trigger
- PWM Generator trigger
- Timer 1/Timer 2 period match
- PWM Generator Current Limit ADC trigger
- PWM Generator Fault ADC trigger

Figure 31-3: Power Conversion Application Example



31.6 ADC Module Clock

The A/D Conversion Clock Divider Select (ADCS<2:0>) bits in the ADCON<2:0> register specify the clock divisor value for the ADC clock generation logic. There are two ADC clock dividers. The high-speed clock divider operates when the PLL is operating and locked. The low-speed ADC clock divider is operational any time the PLL is not operating.

The input to the clock divider is a PLL-generated clock signal providing 240 MHz at 30 MIPS**,** when the PLL is operating. This high-frequency clock provides the needed timing resolution to generate the 24-MHz ADC clock signal required to process two ADC conversions in 1 microsecond.

Note: The ADCS<2:0> bits (ADCON<2:0>) cannot be changed while the ADC module is on (ADON = `1`). Results are unpredictable if this is attempted.

If the PLL is inactive, the input to the clock divider operates at twice the processor MIPS. For example, if the processor is operating at 7.5 MIPS, the input to the ADC clock divider would be 15 MHz.

Note: The 10-bit 2 Msps ADC module is rated for a maximum speed of 2 Msps. Although it may be possible to configure the ADC module for a speed greater than 2 Msps, results would be unpredictable. This feature enables the user to operate the ADC module at maximum speed even though the CPU is operating at less than 30 MIPS.

31.6.1 ADC Clock Setup – PLL ON

When the PLL is active, the input frequency to the ADC clock divider (FADC) is given by Equation 31-1.

Equation 31-1: ADC Input Frequency Calculation with PLL On

$$F_{ADC} = F_{OSC} \times 4$$

With PLL on, FOSC refers to the PLL output clock frequency, which is equal to the EXT or FRC clock frequency multiplied by four. The FRC clock source is further affected by the FRC Oscillator Tuning (TUN<5:0>) bits (OSCTUN<5:0> register) in the Oscillator module. The ADCS<2:0> bits (ADCON<2:0> register) define the clock divider value to be applied to FADC.

Consider the following example:

FOSC = 15 MHz * 4 = 60 MHz (Fast RC Oscillator (FRC) with PLL mode (FNOSC<1:0> = `01`) selected in the Oscillator Selection Configuration (FOSCSEL) register and Maximum Frequency (TUN<3:0> = `0111`) selected in the Oscillator Tuning (OSCTUN<3:0>) register.)

FADC = FOSC * 4
= 60 MHz * 4
= 240 MHz

Total conversion time = sample time (2TAD) + conversion time (10TAD), where TAD is given by Equation 31-2 with the ADC Conversion Clock Divider Select bits set to FADC/10 (ADCS<2:0> = `011`) in the A/D Conversion register (ADCON<2:0>).

Equation 31-2: TAD Calculation

$$T_{AD} = \frac{1}{((F_{ADC}) \div 10)} = \frac{1}{(240 \div 10)} = 41.66\ \mu\text{sec}$$

Total conversion time = (2 * 41.66 µsec) + (10 * 41.66 µsec) = 0.5 µsec (2 Msps).

For a conversion pair, this results in a total conversion time of 1 µsec.

31.6.2 ADC Clock Setup – PLL OFF

With PLL off, F_{OSC} refers to the EXT or FRC clock frequency. The FRC clock source is tuned by the TUN<3:0> bits (OSCTUN<3:0> register). The ADCS<2:0> bits (ADCON<2:0> register) define the clock divider value to be applied to F_{ADC}.

Consider the following case:

F_{OSC} = 15 MHz (Fast RC Oscillator (FRC) with PLL mode (FNOSC<1:0> = `00`) selected in the Oscillator Selection Configuration (FOSCSEL) register and Maximum Frequency (TUN<3:0> = `0111`) selected in the Oscillator Tuning (OSCTUN<3:0>) register.)

$F_{ADC} = F_{OSC}$ = 15 MHz

Total conversion time = sample time ($2T_{AD}$) + conversion time ($10T_{AD}$), where T_{AD} is calculated with the ADC Conversion Clock Divider Select bits set to $F_{ADC}/10$ (ADCS<2:0> = `011`) in the A/D Conversion register (ADCON<2:0>).

With the clock divider set to $F_{ADC}/10$ (ADCS<2:0> = `011`):

$T_{AD} = 1/(F_{ADC} \div 10) = 1/(15 \div 10) = 0.67$ µsec.

Total conversion time = (2 * 0.67) + (10 * 0.67) = 8.04 µsec (0.12 Msps).

For a conversion pair, this results in a total conversion time of 16.08 µsec.

31.7 ADC Module Reference Voltage

The ADC module does not feature external reference voltage pins. Voltage is determined from the AV_{DD} and AV_{SS} pins. The positive reference voltage is AV_{DD} and the negative reference voltage is AV_{SS}. Refer to the Electrical Characteristics in the device data sheet for specific information on the maximum and minimum values for AV_{DD} and AV_{SS}.

31.8 Configuring Analog Port Pins

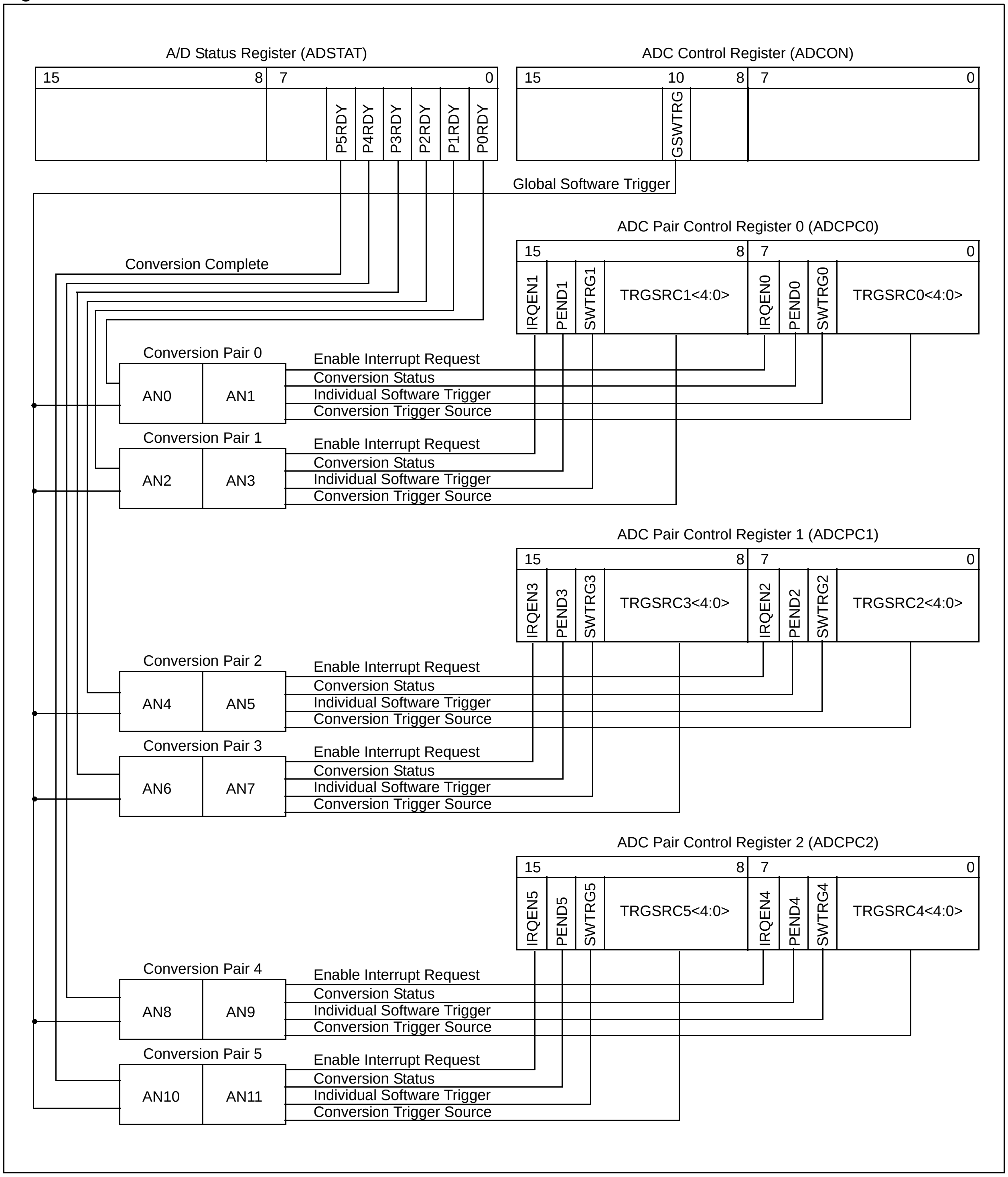
The ADC Port Configuration (ADPCFG) and Data Direction (TRISx) registers control the operation of the A/D port pins. A port pin, when configured for analog input, should have the corresponding Data Direction (TRIS) bit set (input). If the TRIS bit is cleared (output), the analog channel converts the digital output voltage.

The analog port pin should also have the corresponding A/D Port x Configuration Control (PCFGx) bit cleared. This configures the port to disable the digital input buffer.

31.9 Conversion Control

The ADC module has up to three ADC Pair Control registers (ADCPC0, ADCPC1 and ADCPC2), as shown in Figure 31-4. The upper and lower bytes of these registers control the conversion of an analog input pair by specifying the trigger source. These registers also enable the generation of interrupts upon completion of the requested conversion and provide interaction with user software.

Figure 31-4: ADC Conversion Pair Control


A/D Status Register (ADSTAT)
15 8 7 0
P5RDY P4RDY P3RDY P2RDY P1RDY P0RDY
ADC Control Register (ADCON)
15 10 8 7 0
GSWTRG
Global Software Trigger
ADC Pair Control Register 0 (ADCPC0)
IRQEN1 PEND1 SWTRG1 TRGSRC1<4:0> IRQEN0 PEND0 SWTRG0 TRGSRC0<4:0>
Conversion Complete
Conversion Pair 0
AN0 AN1
Conversion Pair 1
AN2 AN3
Enable Interrupt Request
Conversion Status
Individual Software Trigger
Conversion Trigger Source
ADC Pair Control Register 1 (ADCPC1)
IRQEN3 PEND3 SWTRG3 TRGSRC3<4:0> IRQEN2 PEND2 SWTRG2 TRGSRC2<4:0>
Conversion Pair 2
AN4 AN5
Conversion Pair 3
AN6 AN7
ADC Pair Control Register 2 (ADCPC2)
IRQEN5 PEND5 SWTRG5 TRGSRC5<4:0> IRQEN4 PEND4 SWTRG4 TRGSRC4<4:0>
Conversion Pair 4
AN8 AN9
Conversion Pair 5
AN10 AN11

The analog pin conversion pairs are independently triggered for conversion. If multiple conversion pairs are triggered at the same time, the conversion control logic applies a fixed priority to determine when individual conversions are initiated. The highest priority analog pair is AN0 and AN1 (Conversion Pair 0). The lowest priority pair is AN10 and AN11 (Conversion Pair 5).

Each conversion pair has an associated Pending Conversion Status (PENDx) bit in the ADCPCx registers. This status bit indicates when a conversion has been requested but not yet finished. The PENDx bit is set when a trigger request for a conversion has been received and is cleared when both conversions for an input pair have been completed.

Each conversion pair has an associated Conversion Data for Pair x Ready (PxRDY) bit in the A/D Status (ADSTAT) register. These status bits indicate that a conversion is completed and the data is ready in its buffer. The PxRDY bit for each conversion pair must be cleared before the subsequent conversion can be detected. If the bit is not cleared, the conversion request is processed, but an interrupt is not generated.

31.10 ADC Triggers

Each conversion pair can be triggered from a different source. The trigger source is configured by the Trigger Source Selection (TRGSRCx<4:0>) bits in the ADCPCx registers. A conversion pair can be triggered from the following sources:

- Individual Software Trigger
- Global Software Trigger
- PWM Special Event Trigger
- PWM Generator Trigger
- Timer Triggers
- PWM Generator Current Limit Trigger
- PWM Generator Fault ADC Trigger

Table 31-1 lists the trigger source configuration options.

Table 31-1: A/D Conversion Trigger Source Configurations

TRGSRCx<4:0> Bit Configuration	Trigger Source for Associated Analog Channels
`00000`	No conversion enabled
`00001`	Individual software trigger (triggers one conversion pair)
`00010`	Global software trigger (triggers all selected conversion pairs)
`00011`	PWM Special Event trigger
`00100`	PWM Generator 1 trigger
`00101`	PWM Generator 2 trigger
`00110`	PWM Generator 3 trigger
`00111`	PWM Generator 4 trigger
`01100`	Timer 1 period match
`01101`	Timer 2 period match
`01110`	PWM Generator 1 current-limit ADC trigger
`01111`	PWM Generator 2 current-limit ADC trigger
`10000`	PWM Generator 3 current-limit ADC trigger
`10001`	PWM Generator 4 current-limit ADC trigger
`10110`	PWM Generator 1 fault ADC trigger
`10111`	PWM Generator 2 fault ADC trigger
`11000`	PWM Generator 3 fault ADC trigger
`11001`	PWM Generator 4 fault ADC trigger

31.10.1 Software Triggers

The conversion pairs can be configured for software triggers. Software triggers allow the application program to initiate the A/D conversion. The software can trigger conversion individually by setting the SWTRGx bit in the appropriate ADCPCx register or globally by setting the GSWTRG bit in the ADCON register.

A software trigger source is specified when TRGSRCx<4:0> is configured to '`00001`' for individual conversion or '`00010`' for global conversion. The individual software triggers are used in conjunction with the SWTRGx bits in the appropriate ADCPCx register. The global software trigger uses the GSWTRG bit in the ADCON register to trigger multiple conversions. In either case, the TRGSRCx bits in the corresponding ADCPCx registers must be configured to use one or the other software trigger.

31.10.1.1 Individual Software Trigger

An individual software trigger is specified by configuring TRGSRCx<4:0> = `00001` in the ADCPCx register that corresponds to that conversion pair. The conversion process is triggered when the software sets the SWTRGx bit in that register. If the ADC is not busy, the conversion is initiated immediately. If the ADC is busy, then the requested conversion will be performed when the resources become available. The SWTRGx bit is cleared when the requested conversion is initiated.

31.10.1.2 Global Software Trigger

A global software trigger is specified by configuring TRGSRCx<4:0> = `00010` for each conversion pair in the corresponding ADCPCx register. The conversion process is triggered for all properly configured conversion pairs when the software sets the GSWTRG bit in the ADCON register. In this case, the conversion pairs are converted in a fixed priority order. Conversion Pair 0 (AN0 and AN1) has the highest priority and Conversion Pair 6 (AN10 and AN11) has the lowest. As a result, the GSWTRG bit can be used to initiate multiple conversions by a single software instruction. This bit is cleared almost immediately by the hardware when the conversion starts.

31.10.2 PWM Special Event Trigger

When an ADC conversion pair is configured for a PWM special event trigger (TRGSRCx<4:0> = `00011`), conversion occurs when the value in the PTMR register of the PWM module matches the value in the SEVTCMP register. The SEVTPS bits in the PTCON register control the Special Event Trigger postscalar ratio. The Special Event trigger allows the user application to minimize the delay between the time when the A/D conversion results are acquired and the time when the duty cycle value is updated. The PWM generators must be configured for Phased mode for the Special Event Trigger to have a meaningful application. In Independent Time Base mode, the Master time base has no relationship to the local independent time bases.

31.10.3 PWM Generator Trigger

An ADC conversion pair can be configured to be triggered by any of up to four PWM generators. A trigger pulse from a PWM generator can be used to initiate a sample and conversion operation. A match value is loaded in the PWM Trigger Compare Value (TRIGx) register in the Power Supply PWM module. When the value of the local time-base counter matches the TRIGx value, the associated PWM generator generates a trigger. The Trigger Output Divider (TRGDIV<2:0>) bits in the PWM Trigger Control (TRGCONx) register in the Power Supply PWM module act as a postscalar for the ADC trigger. The Trigger Postscalar Start Enable Select (TRGSTRT<5:0>) bits in the TRGCONx register specify the start enable for the postscalar. The PWM Generator Triggers are typically used for implementing control loops.

31.10.4 Timer Period Match Trigger

An ADC conversion pair can be configured to be triggered by a timer period match. Either Timer 1 (TRGSRCx<4:0> = `01100`) or Timer 2 (TRGSRCx<4:0> = `01101`) can be used for this trigger event.

31.10.5 PWM Generator Current Limit Trigger

The PWM generators can be configured to generate triggers based on conditions of the external SFTLTx and IFLTx pins. These fault conditions are classified as Current-Limit Control and Fault Control. The Current-Limit Control Trigger source is controlled by the Current-Limit Control Signal Source Select (CLSRC) bits in the PWM Fault Current-Limit Control (FCLCONx) register in the Power Supply PWM module. The active level for the condition is controlled by the Current-Limit Polarity (CLPOL) bit in the FCLCONx register. When the condition is active, the PWM generator generates a Current-Limit Trigger. If an ADC conversion pair is configured to be triggered by a PWM generator Current-Limit Trigger, the PWM Current-Limit event will initiate a sample and conversion operation. This trigger is typically used to read the current and voltages when an external fault condition occurs.

31.10.6 PWM Generator Fault Trigger

The Fault Control Trigger Source is controlled by the Fault Control Signal Source Select (FLTSRC) bits in the FCLCONx register. The active level for the condition is controlled by the FLTPOL bit in the FCLCONx register. When the condition is active, the PWM generator generates a Fault Control Trigger. If an ADC conversion pair is configured to be triggered by a PWM generator Fault Control Trigger, the PWM Fault Control event will initiate a sample and conversion operation. This trigger is typically used to read the current and voltages when an external fault condition occurs.

31.10.7 External Triggering of Conversion Pairs

The ADC module allows an indirect method of triggering a conversion from an external source. Conversion pairs can be triggered externally via the PWM module. The Current Limit and Fault sources in the PWM module can be configured to be any of the following:

- External SFLT pins
- IFLT pins
- On-board Analog Comparator

A Conversion pair can be configured to be triggered by either the PWM Fault trigger or the PWM Current-Limit Trigger. An active level on the external SFTLTx, IFLTx or the CMPxA/B/C/D pins will cause a PWM Fault or PWM Current-Limit trigger in the PWM module, which will then trigger the ADC module.

31.11 ADC Module Interrupts

The ADC module features Global and Individual Pair interrupts. The Global interrupt (ADCInterrupt) is a logical OR of all the interrupt sources (conversion pairs) in the ADC module. This interrupt can be used if a common procedure needs to be executed when the ADC module completes a conversion. The Individual Pair interrupt (ADCPxInterrupt) is raised when processing of a conversion pair is complete. The Individual Pair interrupts can be used for exclusive processing of channel data.

31.11.1 Group Interrupt Generation

The ADC module provides a common or Group interrupt request that is the logical OR of all the enabled interrupt sources within the module. Each ADCPCx register has two IRQENx bits, one for each conversion pair (see Figure 31-4). If the IRQENx bit is set, an interrupt request is made to the interrupt controller when the requested conversion is completed. When an interrupt is generated by a conversion pair, an associated PxRDY bit in the ADSTAT register is set. The PxRDY bit must be cleared by the software before the next pair conversion request can be processed.

In the Interrupt Service Routine, the application can use the ADSTAT register to find the conversion pair that caused the interrupt. The group interrupt is useful for applications that use a common software routine to process ADC interrupts for multiple analog input pairs.

The application software must clear the IFS bit associated with the ADC interrupt before clearing the PxRDY bit. If the PxRDY bit for a conversion pair is cleared, the ADC module will process a conversion request for this conversion pair. If this happens and the IFS bit is cleared, the new interrupt is lost.

31.11.2 ADC Interrupt Vectoring Using the ADBASE Register

To speed up the evaluation of PxRDY bits in the ADSTAT register, the ADC module uses a read/write address base register, ADBASE. When read, the ADBASE register provides a sum of the contents of the ADBASE register plus an encoding of the PxRDY bits set in the ADSTAT register. If there are no PxRDY bits set, reading the ADBASE register will provide the value written to the register. The least significant bit of the ADBASE register is forced to zero, which ensures that all (ADBASE + PxRDY) results will be on instruction boundaries.

The PxRDY bits are binary-priority encoded, with P0RDY the highest priority and P5RDY the lowest priority. The encoded priority result is then shifted left two bit positions and added to the contents of the ADBASE register. The priority encoding yields addresses that are on two instruction word boundaries.

The user application typically loads the ADBASE register with the base address of a jump table that either contains the addresses of the appropriate ISRs or branches to the appropriate ISR. The encoded PxRDY values are set up to reserve two instruction words per entry in the jump table. The application software typically uses one instruction word to load an identifier into a W register, and the other instruction to branch to the appropriate ISR.

See Example 31-3 and Example 31-4 for implementation details on ADC interrupt vectoring using the ADBASE register.

31.11.3 Individual Pair Interrupts

The ADC module also provides Individual Pair Interrupt outputs, one for each conversion pair. These interrupts are configured via the IEC, IFS and IPC Interrupt Controller registers. The use of individual interrupts can reduce the problem of accidentally losing a pending interrupt while processing and clearing a current interrupt.

Using the group interrupt can require the ISR to determine which interrupt source generated the interrupt. For applications that use separate software tasks to process ADC data, a common interrupt vector can cause performance bottlenecks. The use of the individual pair interrupts can save many clock cycles compared to using the group interrupt to process multiple interrupt sources. The individual pair interrupts support the construction of application software that is responsive and organized on a task basis.

Regardless of whether an individual pair interrupt or the global interrupt is used to respond to an interrupt request from an ADC conversion, the PxRDY bits in the ADSTAT register function in the same manner. The use of the individual pair interrupts also enables the application to change the interrupt priority of individual ADC channels (pairs), as compared to the fixed priority structure of the group interrupt.

31.11.4 Early Interrupt Generation

The Early Interrupt Enable (EIE) control bit in the ADCON<7> register enables the generation of the interrupts after completion of the first conversion instead of waiting for the completion of both inputs of a conversion pair. Even though the second input will still be in the conversion process, the software can be written to perform some of the computations using the first data value while the second conversion is being completed.

The application software can be written to account for the conversion period of the second input before using the second data, or the application can poll the PEND bit in the ADCPCx register. The PEND bit is cleared automatically when conversion of the pair is complete.

Note: If EIE is enabled, the PxRDY bit is set after the first conversion is completed and an interrupt is generated. The PEND bit, however, is cleared only after the second conversion has been completed.

31.12 Conversion Request Conflict Resolution

If more than one pair conversion request is active at the same time, the ADC control logic will process the requests from highest to lowest priority, starting at conversion pair 0 (AN0 and AN1) and ending at conversion pair 5 (AN10 and AN11). If the application software specifies the same trigger source for multiple conversion pairs, the ADC module will process the requested conversions sequentially until the sequence has been completed.

The ADC module will not loop once triggered. Each sequence of conversion requires a new trigger or multiple triggers.

31.13 Sampling Sequence Control

The ADC module manages analog-to-digital conversion by providing dynamic control over how analog inputs to each conversion pair are sampled and converted. Sampling and conversion are independently controlled to ensure optimum use by the application.

Analog inputs can be sampled simultaneously or sequentially. The samples can then be converted in normal order (for example, AN0, then AN1) or reverse order (AN1, then AN0). This results in the following combinations:

- Simultaneous input sampling with normal conversion order
- Simultaneous input sampling with reverse conversion order
- Sequential input sampling with normal conversion order
- Sequential input sampling with reverse conversion order

31.13.1 Conversion Order

Normal or reverse conversion order is controlled by the Conversion Order (ORDER) bit in the A/D Control (ADCON<6>) register. If ORDER = `0`, the even numbered input in the conversion pair is converted first, followed by the odd numbered input. If ORDER = `1`, the odd numbered input in the conversion pair is converted first, followed by the even numbered input. This feature, when used in conjunction with the Early Interrupt Enable (EIE) bit (ADCON<7>), minimizes the time for acquiring feedback data in order to update the output of a control system.

31.13.2 Simultaneous and Sequential Sampling

Simultaneous and sequential sampling are controlled by the Sequential Sample Enable (SEQSAMP) bit (ADCON<5>). With simultaneous sampling, both inputs in the conversion pair are sampled at the same time. With sequential sampling, the sample event for the second conversion starts at the end of the conversion event of the first conversion. Conversion pairs that use both the Shared and the Dedicated S/H amplifiers are sampled in the following manner:

- If SEQSAMP = `0`, and the common (shared) S/H circuit is NOT busy, the shared S/H circuit samples its specified input at the same time as the dedicated S/H circuit. This action provides “simultaneous” sample and hold functionality.
- If SEQSAMP = `0`, and the shared S/H circuit is currently busy with a conversion in progress, the shared S/H circuit samples as soon as possible (at the start of the new conversion process for the pair).
- If SEQSAMP = `1`, the shared S/H circuit samples at the start of the conversion process for that input. For example, if the ORDER bit = `0`, the shared S/H circuit samples when conversion of the even numbered input is complete and the conversion of the odd numbered input must start. If ORDER = `1`, the shared S/H circuit samples along with the dedicated S/H circuit. The odd numbered input is converted first.

Conversion pairs that use the shared S/H circuit for both the input channels will always sample sequentially. The SEQSAMP bit is useful for some applications that want to minimize the time from a sample event to the conversion of the sample.

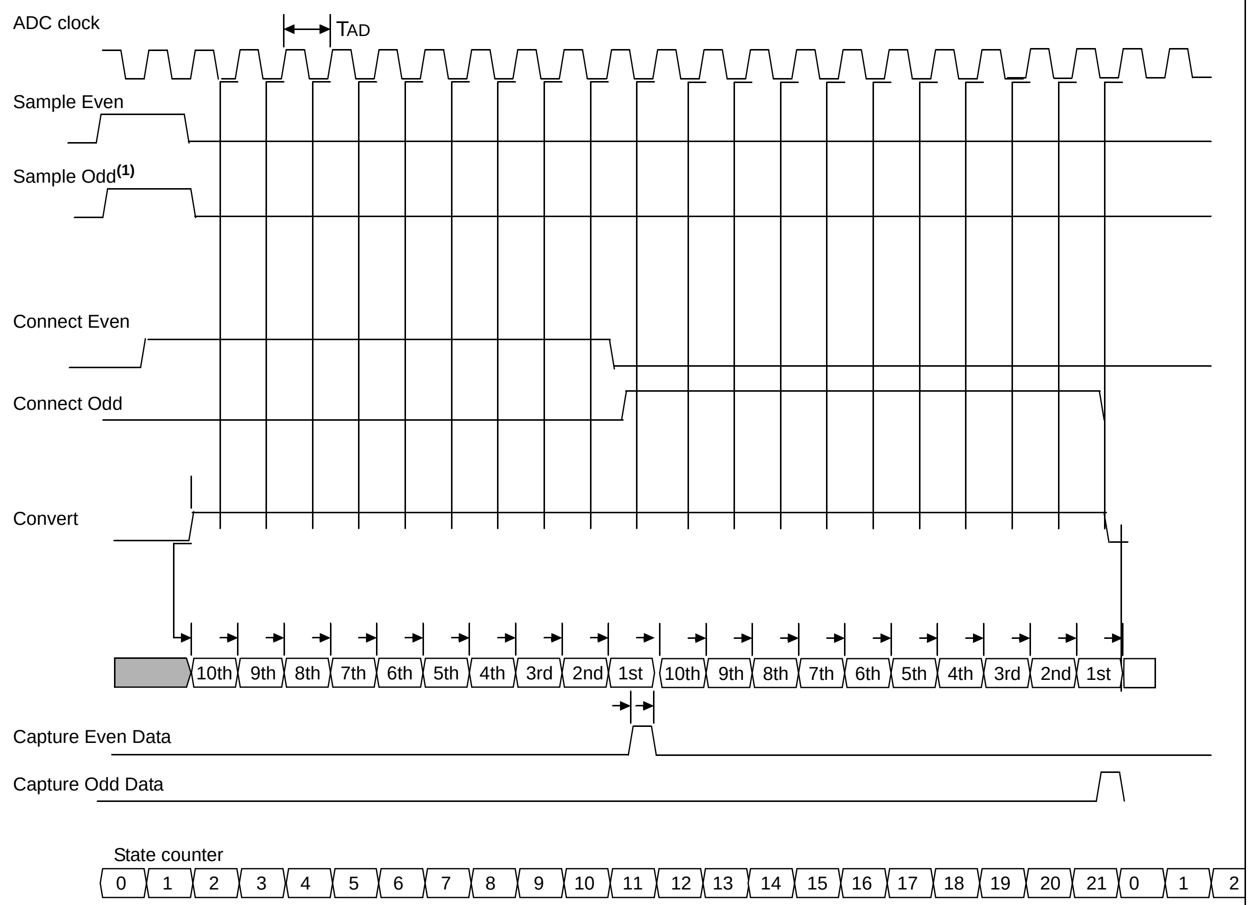
Note:	Irrespective of the ORDER and SEQSAMP bit values, the dedicated S/H circuits always sample at time of trigger.

31.14 ADC – Sample and Conversion Timing

The S/H circuits assigned to the input pins have their own timing logic, which is triggered when an external sample and convert request (from PWM or TMR) is made. The S/H circuits have a fixed two-clock data sample period. When the sample has been acquired, the ADC control logic is notified of a pending request. The conversion is performed as the conversion resources become available.

The ADC module always converts a pair of analog input channels, so a typical conversion process requires 24 clock cycles. Figure 31-5 shows the timing sequence for a sample and conversion operation when ORDER = `0` and SEQSAMP = `0`. It also assumes that the converter is not busy with another conversion. The ADC module will sample both channels at nearly the same time. A conversion process involves sampling the channel, connecting the channel to the converter and storing the conversion result at the end of conversion.

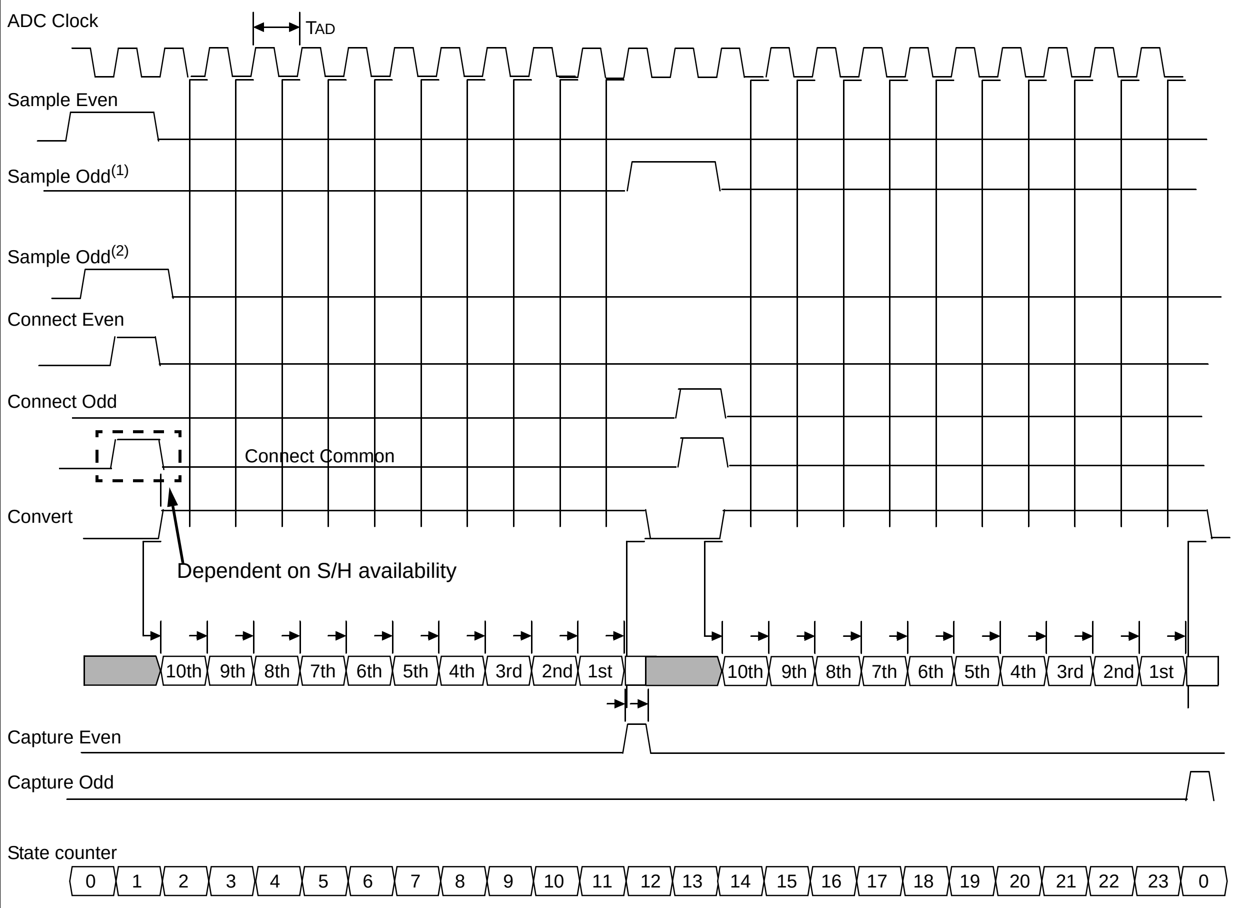
Figure 31-5: Detailed Conversion Sequence Timings, ORDER = `0` and SEQSAMP = `0`



Note 1: The Sample Odd event occurs at this time if the Sample Even input is connected to a dedicated S/H circuit. Otherwise, it occurs after conversion of the even event.

Figure 31-6 shows detailed timing of a conversion when sequential sampling is enabled. The diagram clarifies the functioning of the sequential sampling feature. If a channel has a dedicated S/H circuit, it will always sample when triggered. A channel that uses the common S/H circuit will try to sample when the previous conversion is complete. If the common S/H circuit is busy (a conversion is taking place), the ADC logic will wait for it to be freed before starting a sample operation.

Figure 31-6: Detailed Conversion Sequence Timings, ORDER = `0` and SEQSAMP = `1`



Note 1: For all analog input pairs that do not have dedicated S/H circuits, the common S/H circuit samples the input at the start of the first and second conversions. Therefore, the samples are sequential, not simultaneous.

2: For all analog input pairs that have dedicated S/H circuits, the common S/H circuit samples the input at the start of the first conversion so that both samples (odd and even) are nearly simultaneous.

31.15 Sequential Sampling and Pair Conversion Order

This section describes the behavior of the ADC module when multiple conversion pairs request conversion simultaneously. The SEQSAMP and ORDER bits in the ADCON register affect the sampling and conversion sequence of the individual channels. Consider an example where two conversion pairs (AN1, AN0) and (AN3, AN2) are triggered at the same time. The sampling and conversion sequence is affected by the following:

- The SEQSAMP bit.
- The ORDER bit.
- The current allocation of resources in the ADC module. Channels that have a dedicated S/H circuit will be activated immediately when triggered, irrespective of the SEQSAMP bit settings. The SEQSAMP bit affects channels that use the shared (common) S/H circuit.
- The fixed priority of conversion when multiple conversions are requested. Conversion pair AN1, AN0 has the highest priority (will be processed first), conversion pair AN3, AN2 will be processed next, and so on.

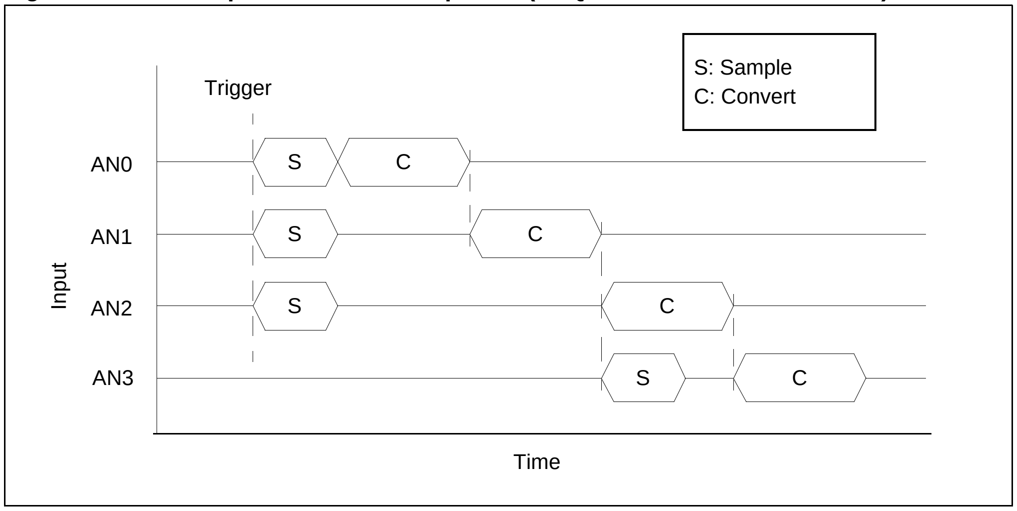
In the preceding example, channels AN0 and AN2 have dedicated S/H circuits, while channels AN1 and AN3 have shared S/H circuits. Hence channels AN0 and AN2 will always sample when triggered, irrespective of the SEQSAMP and ORDER bits.

31.15.1 SEQSAMP = `0` AND ORDER = `0`

Figure 31-7 shows the sequence of sampling and conversion events when conversion pair AN1, AN0 and conversion pair AN3, AN2 are triggered at the same time. Channels AN0, AN1 and AN2 are sampled at the same time. Since ORDER = `0`, the even numbered channel in a conversion pair is converted first. AN0 is converted first. AN1 is converted next. When the conversion of AN1 is complete, the following occurs:

- The shared S/H circuit is released and AN3 can be sampled.
- The AN2 conversion process is initiated.
- When channel AN2 conversion is complete, the conversion of channel AN3 is initiated.

Figure 31-7: Sample Conversion Sequence (SEQSAMP = `0` and ORDER = `0`)



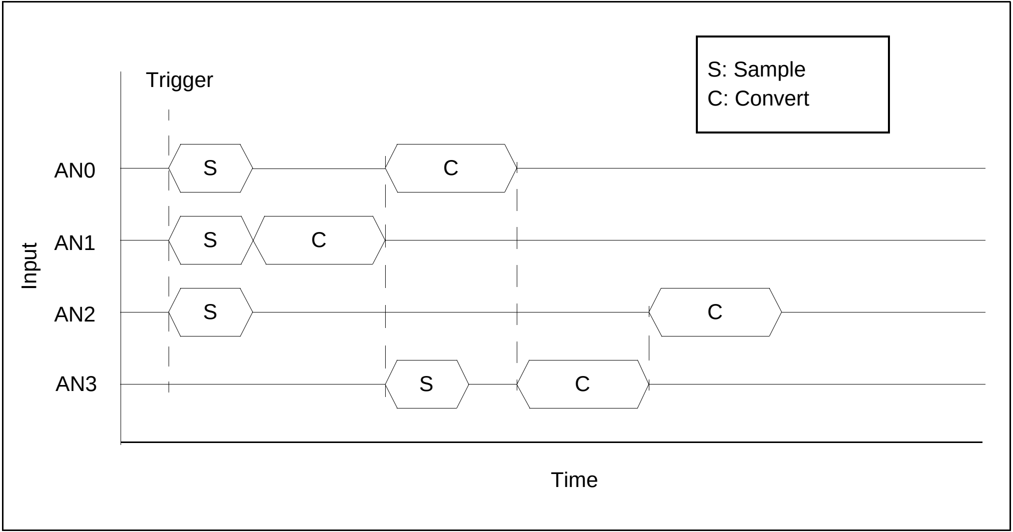
31.15.2 SEQSAMP = 0 and ORDER = 1

Figure 31-8 shows the sequence of sample and conversion events when SEQSAMP = 0 and ORDER = 1. Channels AN0, AN1 and AN2 are sampled simultaneously. In this case, the odd numbered channel in a conversion pair is converted first. When the conversion of AN1 is complete, the following occurs:

- The shared S/H circuit is released and AN3 can be sampled.
- The AN0 conversion process is initiated.

When the conversion of AN0 is complete, AN3 is converted (odd channel first since ORDER = 1), followed by the conversion of AN2.

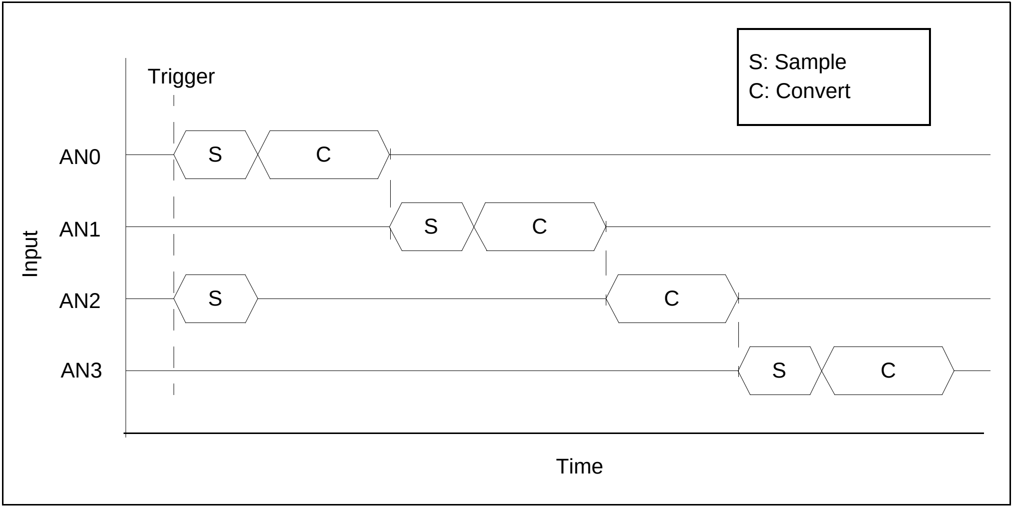
Figure 31-8: Sample Conversion Sequence (SEQSAMP = 0 AND ORDER = 1)



31.15.3 SEQSAMP = 1 and ORDER = 0

Figure 31-9 shows the sequence of sample and conversion events when SEQSAMP = 1 and ORDER = 0. Channels AN0 and AN2 are sampled at the same time since these are not affected by the SEQSAMP bit. When the conversion of AN0 is complete, AN1 is sampled and converted. Note that AN2 is not converted, even though the sampling process is complete. This is due to the priority assigned to the conversion of AN1 and AN0. When conversion of AN1 is complete, AN2 is converted. AN3 is then sampled and converted.

Figure 31-9: Sample Conversion Sequence (SEQSAMP = 1 and ORDER = 0)



31.15.4 SEQSAMP = 1 and ORDER = 1

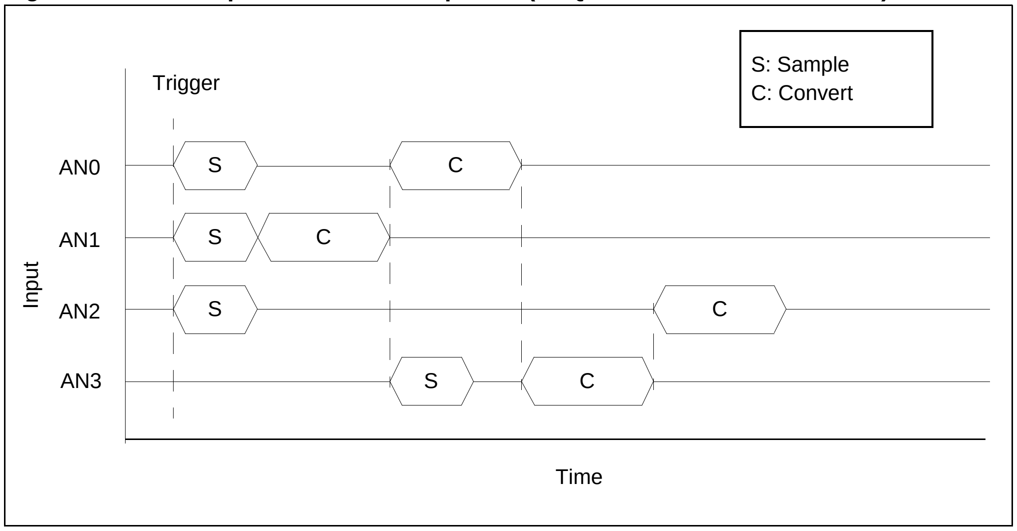
Figure 31-10 shows the sequence of sample and conversion events when SEQSAMP = 1 and ORDER = 1. Channels AN0 and AN2 are sampled at the same time, since these inputs are not affected by the SEQSAMP bit. Channel AN1 is also sampled, since ORDER = 1. This also dictates that AN1 will be converted first.

When the conversion of AN1 is complete, the following occurs:

- The dedicated S/H circuit is released, and AN3 is sampled.
- The converter is now free to convert AN0.

When AN0 conversion is complete, AN3 is converted. When the conversion of AN3 is complete, AN2 conversion is initiated. Note that although AN2 is sampled when the conversion trigger occurs, it is converted last.

Figure 31-10: Sample Conversion Sequence (SEQSAMP = 1 and ORDER = 1)



31.16 A/D Module Configuration

Follow this process to configure an A/D conversion:

1. In the ADPCFG register, define which port pins are analog inputs to the ADC.
2. In the TRIS register, define the ports for the analog input pins as inputs.
3. Using the ADCS<2:0> bits in the ADCON register, select the A/D conversion clock.
4. Using the FORM bit in the ADCON register, specify how the conversion results should be presented.
5. Using the SEQSAMP and ORDER bits in the ADCON register, set up the sampling sequence and conversion order.
6. Set up conversion pair Interrupts and Trigger Sources in the corresponding ADCPC registers.
7. If ADC interrupt is desired, configure the ADC interrupt priority and clear the related status flag in the IFS register. Set the related IEC bit.
8. Clear the ADSTAT register.
9. Turn the A/D module on (ADCON1<15>).

31.17 ADC Sampling Requirements

The total sampling time for the ADC is a function of the internal amplifier settling time and the holding capacitor charge time. To minimize the effects of pin leakage currents on the accuracy of the ADC, the maximum recommended source impedance (Rs) is 100 ohms. After the analog input channel is selected, the sampling function must be completed before the conversion can start.

The minimum sample time is 39.2 nsec. The recommended sample time is 44.8 nsec. For more details, refer to the electrical characteristics for the specific device.

31.18 Reading the A/D Result Buffer

The ADBUF registers are 10 bits wide, but the data is automatically formatted to one of two selectable formats when a read from the buffer is performed. The FORM bit (ADCON<8>) selects the format. The formatting hardware provides a 16-bit result on the data bus for all of the data formats. Figure 31-11 shows the data output formats that can be selected using the FORM bit.

Figure 31-11: A/D Output Data Formats

ADBUF Contents:							d09	d08	d07	d06	d05	d04	d03	d02	d01	d00
Read to Bus:																
Integer	0	0	0	0	0	0	d09	d08	d07	d06	d05	d04	d03	d02	d01	d00
Fractional (1.15)	d09	d08	d07	d06	d05	d04	d03	d02	d01	d00	0	0	0	0	0	0

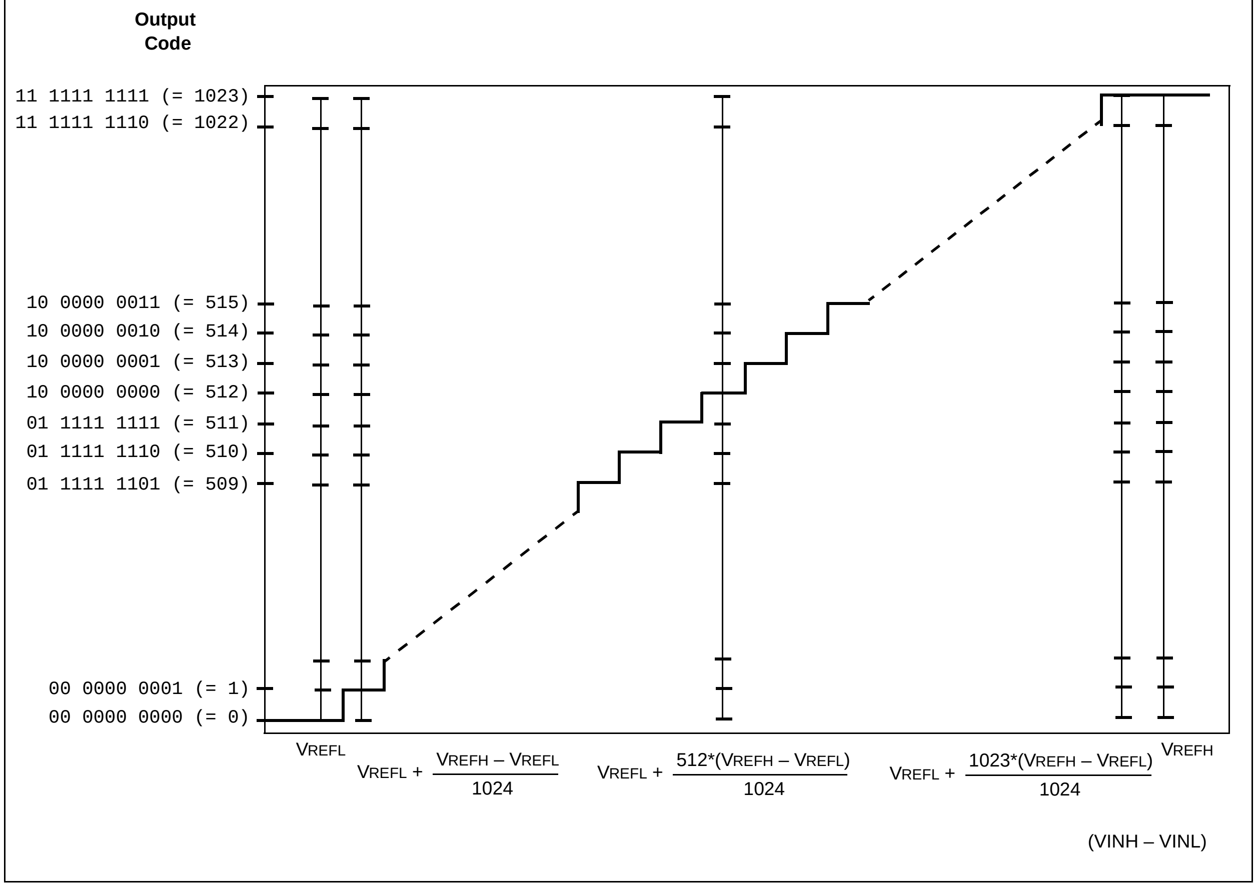
31.19 Transfer Function

The ideal transfer function of the ADC is shown in Figure 31-12. The difference of the input voltages (VINH – VINL) is compared to the reference (VREFH – VREFL).

Note: VREFH and VREFL are connected to AVDD and AVSS respectively.

- The first code transition occurs when the input voltage is (VREFH – VREFL/2048) or 0.5 LSb.
- The '`00 0000 0001`' code is centered at (VREFH – VREFL/1024) or 1.0 LSb.
- The '`10 0000 0000`' code is centered at (512*(VREFH – VREFL)/1024).
- An input voltage less than (1*(VREFH – VREFL)/1024) converts as '`00 0000 0000`'.
- An input greater than (1023*(VREFH – VREFL)/1024) converts as '`11 1111 1111`'.

Figure 31-12: A/D Transfer Function



31.20 Connection Considerations

Since the analog inputs employ ESD protection, they have diodes to V_{DD} and V_{SS}. This requires that the analog input be between V_{DD} and V_{SS}. If the input voltage exceeds this range by greater than 0.3V (either direction), one of the diodes becomes forward biased and it can damage the device if the input current specification is exceeded.

An external RC filter is sometimes added for anti-aliasing of the input signal. The R component should be selected to ensure that the sampling time requirements are satisfied. Any external components connected (via high-impedance) to an analog input pin (capacitor, Zener diode, etc.) should have very little leakage current at the pin.

31.20.1 ADC Operation During CPU Idle Mode

The Stop in Idle Mode (ADSIDL) bit in the A/D Control (ADCON<13>) register selects whether the module stops or continues on Idle. If ADSIDL = `0`, the module will continue normal operation when the device enters Idle mode. If the A/D interrupt is enabled in the Interrupt module, the device will wake up from Idle mode when the A/D interrupt occurs. Program execution will resume at the A/D Interrupt Service Routine if the A/D interrupt priority is greater than the current CPU priority. Otherwise, execution will continue from the instruction after the `PWRSAV` instruction that placed the device in Idle mode.

If ADSIDL = `1`, the module will stop in Idle mode. If the device enters Idle mode in the middle of a conversion, the conversion is aborted. The converter will not resume a partially completed conversion on exiting from Idle mode.

31.21 Effects of a Reset

A device Reset forces all registers to their Reset state. This forces the ADC module to be turned off and any conversion in progress to be aborted. All pins that are multiplexed with analog inputs will be configured as analog inputs. The corresponding TRIS bits will be set.

The values in the ADCBUF registers are not initialized during a Power-on Reset. ADCBUF0… ADCBUFB will contain unknown data.

31.22 ADC Module – Bit Setting and Clearing Responsibility

The ADC module registers feature a set of bits that are set/cleared automatically by hardware and a set of bits which need to be set or cleared by the software. The following list describes these bits.

- The Software Trigger (SWTRGx) bit (ADCPCx register) is set by the software and cleared by the hardware when the pair conversion is initiated.
- The Conversion Pending (PENDx) Bit (ADCPCx register) is set by the hardware when a conversion pair is triggered, and cleared by the hardware when the pair conversion is complete.
- The Pair Conversion Complete Status (PxRDY) bit (ADSTAT register) is set by hardware when the conversion pair causes an interrupt. This happens when pair conversion is complete or, if the Early Interrupt Enable (EIE) bit (ADCON<7>) is set, when the first channel within a pair has been converted. The PxRDY bit must be cleared in software for the pair to generate subsequent interrupts.
- The Global Software Trigger (GSWTRG) bit (ADCON,10>) is set by the software to trigger a conversion and cleared by the hardware when the conversion process is initiated.

31.23 ADC – Code Examples

This section provides code examples for various configurations of the ADC module. The method of employing the ADBASE register is described using both assembly and C language. The timing calculations in all the examples assume that the internal FRC clock has been selected with the high-range option and that PLL has been enabled. The TUN bits are assumed to be ‘`0000`’, resulting in FOSC value of 14.55 MHz. The examples have been coded for the dsPIC30F2020 which has eight ADC channels.

31.23.1 Pair Conversion Trigger by Timer

Example 31-1 configures Pair 0 to be triggered when Timer 1 period matches. With a FOSC value of 14.55 MHz, the device core will operate at FCY = 29.1 MHz. Timer 1 increments at intervals of 34.36 nsec. The input clock to the ADC module is FADC = 232.8 MHz. With ADCS = 5, the ADC module will operate at 16.62 MHz. This will result in a channel conversion time of approximately 722 nsec. A conversion pair would require 1.44 µsec for its processing. In the example, the Timer 1 period is set to trigger a conversion every 3.43 µsec.

31.23.2 Early Interrupt Generation with (EIE)

Example 31-2 configures Pair 0 to be converted when triggered by software. As the EIE bit is set, the ADC interrupt occurs when channel AN0 conversion is completed. The end of conversion for the channel AN1 is checked for by polling the PEND bit in ADCPC0.

31.23.3 Vectored Interrupt From ADBASE – Code Example in ‘C’

Example 31-3 demonstrates the use of the ADBASE register to implement Individual Pair Conversion ISRs. In this code example, the jump table is an array containing the pointers to ISRs for each conversion pair. The ADBASE register is initialized with the address of `jumpTable`. When a conversion is complete, ADBASE will point to a location in the jump table. The function pointer stored in this location points to an ISR corresponding to the pair that caused the interrupt. For example, if Pair 1 caused an interrupt, ADBASE will point to `jumpTable[2]`. This location contains a pointer to the function `ConvPair1Handler`, which is the service routine for Pair 1 Conversion.

All of the four conversion pairs are configured to be triggered by software.

31.23.4 Vectored Interrupt From ADBASE – Code Example in Assembly

Example 31-4 also demonstrates Pair Conversion Interrupt handling using the ADBASE register. The ADBASE register is initialized with the address of `jumpTable`. When a conversion is complete, ADBASE will point to a location in the jump table. The function pointer stored in this location points to an ISR corresponding to the pair that caused the interrupt. For example, if Pair 2 caused an interrupt, ADBASE will point to `jumpTable[2]` + 8. This location contains a pointer to the function `handlePair2`, which is the service routine for Pair 2 Conversion.

31.23.5 Pair Conversion Interrupt With PWM Trigger.

Example 31-5 demonstrates triggering of a conversion pair interrupt using the PWM module. The TRGCON and TRIG registers in the PWM module are configured to cause a trigger every 0.214 µsec from the start of the PWM cycle. Pair 0 of the ADC module is configured to be triggered by a trigger from PWM Generator #1. The conversion process takes 1.44 µsec before causing the ADC interrupt.

Example 31-1: Interrupt Triggered by Timer 1

```
#include <p30F2020.h>

#defineTIMER_PERIOD  0x0064                   /* Set the timer period for 3.43 usec*/

int main(void)
{
    /* Set up the ADC Module */

    ADCONbits.ADSIDL = 0;                     /* Operate in Idle Mode*/
    ADCONbits.FORM = 0;                       /* Output in Integer Format*/
    ADCONbits.EIE = 0;                        /* No Early Interrupt*/
    ADCONbits.ORDER = 0;                      /* Even channel first*/
    ADCONbits.SEQSAMP = 1;                    /* Sequential Sampling Enabled*/
    ADCONbits.ADCS = 5;                       /* Clock Divider is set up for Fadc/14*/


    ADPCFG = 0xFFFC;                          /* AN0 and AN1 are analog inputs.*/
    ADSTAT = 0;                               /* Clear the ADSTAT register*/
    ADCPC0bits.TRGSRC0 = 0xC;                 /* Trigger conversion on Timer 1 Period Match*/
    ADCPC0bits.IRQEN0 = 1;                    /* Enable the interrupt*/

    /* Set up Timer1 */

    T1CON = 0;                                /* Timer with 0 prescale*/
    TMR1 = 0;                                 /* Clear the Timer counter*/
    PR1 = TIMER_PERIOD;                       /* Load the period register*/

    /* Set up the Interrupts */

    IFS0bits.ADIF = 0;                        /* Clear AD Interrupt Flag*/
    IPC2bits.ADIP = 4;                        /* Set ADC Interrupt Priority*/
    IEC0bits.ADIE = 1;                        /* Enable the ADC Interrupt*/

    /* Enable ADC and Timer */
    ADCONbits.ADON = 1;                       /* Start the ADC module*/
    T1CONbits.TON = 1;                        /* Start the Timer*/

    while(1);

    }

void __attribute__ ((__interrupt__)) _ADCInterrupt(void)
{
    /* AD Conversion complete interrupt handler */

    int channel0Result, channel1Result;

    IFS0bits.ADIF = 0;                        /* Clear ADC Interrupt Flag*/
    channel0Result = ADCBUF0;                 /* Get the conversion result*/
    channel1Result = ADCBUF1;
    ADSTATbits.P0RDY= 0;                      /* Clear the ADSTAT bits*/
    }
```

Example 31-2: Early Interrupt Using ADCON<EIE>

```
#include <p30F2020.h>

int main(void)
{
    int channel1Result;

    /* Set up the ADC Module */

    ADCONbits.ADSIDL = 0;                   /* Operate in Idle Mode*/
    ADCONbits.FORM = 0;                     /* Output in Integer Format*/
    ADCONbits.EIE = 1;                      /* Enable Early Interrupt*/
    ADCONbits.ORDER = 0;                    /* Even channel first*/
    ADCONbits.SEQSAMP = 1;                  /* Sequential Sampling Enabled*/
    ADCONbits.ADCS = 5;                     /* Clock Divider is set up for Fadc/14*/

    ADPCFG = 0xFFFC;                        /* AN0 and AN1 are analog inputs.*/
    ADSTAT = 0;                             /* Clear the ADSTAT register*/
    ADCPC0bits.TRGSRC0 = 0x1;               /* Trigger conversion on Timer 1 Period Match*/
    ADCPC0bits.IRQEN0= 1;                   /* Enable the interrupt*/

    ADCONbits.ADON = 1;                     /* Start the ADC module*/

    /* Set up the Interrupts */

    IFS0bits.ADIF = 0;                      /* Clear AD Interrupt Flag*/
    IPC2bits.ADIP = 4;                      /* Set ADC Interrupt Priority*/
    IEC0bits.ADIE = 1;                      /* Enable the ADC Interrupt*/

    ADCPC0bits.SWTRG0 = 1;                  /* Trigger the Conversion Pair 0*/

    while (1)
    {
        while(ADCPC0bits.PEND0);            /* Wait for the 2nd conversion to complete*/
        channel1Result = ADCBUF1;           /* Read the result of the second conversion*/
        ADCPC0bits.SWTRG0 = 1;              /* Trigger another conversion*/

    }
}

void __attribute__ ((__interrupt__)) _ADCInterrupt(void)
{
    /* AD Conversion complete early interrupt handler */

    int channel0Result;

    IFS0bits.ADIF = 0;                      /* Clear ADC Interrupt Flag*/
    channel0Result = ADCBUF0;               /* Get the conversion result*/
    ADSTATbits.P0RDY = 0;                   /* Clear the ADSTAT bits*/


}
```

Example 31-3: Vectored Interrupt From ADBASE Register – ‘C’ Code

```
#include <p30F2020.h>


#defineTIMER_PERIOD  0x0064                      /* Set the timer period for 3.43 usec*/
#define CONVERSION_PAIRS 4                       /* Number of conversion pairs*/

void ConvPair0Handler (void);                    /* Declare the pair conversion handlers*/
void ConvPair1Handler (void);
void ConvPair2Handler (void);
void ConvPair3Handler (void);


void (*jumpTable[CONVERSION_PAIRS * 2 -1])(void);

int main(void)
{
    /* Set up the ADC Module */

    jumpTable[0] = &ConvPair0Handler;            /* Set up the jump table*/
    jumpTable[2] = &ConvPair1Handler;
    jumpTable[4] = &ConvPair2Handler;
    jumpTable[6] = &ConvPair3Handler;

    ADCONbits.ADSIDL = 0;                        /* Operate in Idle Mode*/
    ADCONbits.FORM = 0;                          /* Output in Integer Format*/
    ADCONbits.EIE = 0;                           /* No Early Interrupt*/
    ADCONbits.ORDER = 0;                         /* Even channel first*/
    ADCONbits.SEQSAMP = 1;                       /* Sequential Sampling Enabled*/
    ADCONbits.ADCS = 5;                          /* Clock Divider is set up for Fadc/14*/


    ADPCFG = 0xFFF0;                             /* AN0, AN1,AN2 & AN3 are analog inputs.*/
    ADSTAT = 0;                                  /* Clear the ADSTAT register*/

    ADCPC0bits.TRGSRC0 = 0xC;                    /* Pair 0 conversion on Timer 1 Period Match*/
    ADCPC0bits.IRQEN0 = 1;                       /* Pair 0 Enable the interrupt*/
    ADCPC0bits.TRGSRC1 = 0xC;                    /* Pair 1 conversion on Timer 1 Period Match*/
    ADCPC0bits.IRQEN1 = 1;                       /* Pair 1 Enable the interrupt*/
    ADCPC1bits.TRGSRC2 = 0xC;                    /* Pair 2 conversion on Timer 1 Period Match*/
    ADCPC1bits.IRQEN2 = 1;                       /* Pair 2 Enable the interrupt*/
    ADCPC1bits.TRGSRC3 = 0xC;                    /* Pair 3 conversion on Timer 1 Period Match*/
    ADCPC1bits.IRQEN3 = 1;                       /* Pair 3 Enable the interrupt*/

    ADBASE =(int)( &jumpTable[0]);               /* Initialize ADBASE with the starting address */
                                                 /* of the jump table*/

    /* Set up Timer1 */

    T1CON = 0;                                   /* Timer with 0 prescale*/
    TMR1 = 0;                                    /* Clear the Timer counter*/
    PR1    = TIMER_PERIOD;                       /* Load the period register*/

    /* Set up the Interrupts */

    IFS0bits.ADIF = 0;                           /* Clear AD Interrupt Flag*/
    IPC2bits.ADIP = 4;                           /* Set ADC Interrupt Priority*/
    IEC0bits.ADIE = 1;                           /* Enable the ADC Interrupt*/
```

Example 31-3: Vectored Interrupt From ADBASE Register – 'C' Code (Continued)

```
    /* Enable ADC and Timer */
    ADCONbits.ADON = 1;                       /* Start the ADC module*/
    T1CONbits.TON = 1;                        /* Start the Timer*/

    while(1);

    }

void __attribute__ ((__interrupt__)) _ADCInterrupt(void)
{
    /* AD Conversion complete interrupt handler */

    IFS0bits.ADIF = 0;                        /* Clear ADC Interrupt Flag*/

    ( (void (*)()) *((int *)ADBASE))();       /* Call the corresponding handler*/

}

void ConvPair0Handler (void)
{
    int channel0, channel1;

    ADSTATbits.P0RDY = 0;                     /* Clear the ADSTAT bits*/
    channel0 = ADCBUF0;                       /* Read channel 0 conversion result*/
    channel1 = ADCBUF1;                       /* Read channel 0 conversion result*/

}

void ConvPair1Handler (void)
{
    int channel2, channel3;

    ADSTATbits.P1RDY = 0;                     /* Clear the ADSTAT bits*/
    channel2 = ADCBUF2;                       /* Read channel 2 conversion result*/
    channel3 = ADCBUF3;                       /* Read channel 3 conversion result*/
}

void ConvPair2Handler (void)
{
    int channel4, channel5;

    ADSTATbits.P2RDY = 0;                     /* Clear the ADSTAT bits*/
    channel4 = ADCBUF4;                       /* Read channel 4 conversion result*/
    channel5 = ADCBUF5;                       /* Read channel 5 conversion result*/
}

void ConvPair3Handler (void)
{
    int channel6, channel7;

    ADSTATbits.P3RDY = 0;                     /* Clear the ADSTAT bits*/
    channel6 = ADCBUF6;                       /* Read channel 6 conversion result*/
    channel7 = ADCBUF7;                       /* Read channel 7 conversion result*/
}
```

Example 31-4: Vectored Interrupt From ADBASE Register – Assembly Code

```
    .include"p30F2020.inc"
    .global __reset
    .global__ADCInterrupt
    .equiv STACK_BASE,0x0900
    .equiv STACK_LIM,0x09F0
    .equiv CHAN_CFG,0xFF00                   ; The input channel config word for ADPCFG

    .section jumpTable,bss
    handlePair0: .space 4                    ; Allocate 4 bytes for storing address of each pair
    handlePair1: .space 4                    ; handler
    handlePair2: .space 4
    handlePair3: .space 4

    .text
__reset:

    MOV #STACK_BASE,w15                      ; Initialize the stack pointer and the stack limit
    MOV #STACK_LIM,w0
    MOV w0, SPLIM

    CLR ADCON                                ; Configure the ADC module
    BCLR ADCON, #13                          ; IDLE = 0, Operate in IDLE mode
    BCLR ADCON, #8                           ; FORM = 0, Output in Integer format
    BCLR ADCON, #7                           ; EIE   = 0, No Early Interrupt
    BCLR ADCON, #6                           ; ORDER= 0, Even channel first
    BSET ADCON, #5                           ; SEQSAMP= 1, Sequential Sample Enabled
    BSET ADCON, #2                           ; ADCS = 5, Divide Fadc by 14
    BCLR ADCON, #1
    BSET ADCON, #0

    MOV #.startof.(jumpTable), w0            ; Initialize the ADBASE register with the
    MOV w0, ADBASE                           ; Jump table address

    MOV #handle(Pair0Handler), w0            ; Store the address of the pair handlers
    MOV w0,handlePair0                       ; in the jump table.
    MOV #handle(Pair1Handler), w0
    MOV w0,handlePair1
    MOV #handle(Pair2Handler), w0
    MOV w0,handlePair2
    MOV #handle(Pair3Handler), w0
    MOV w0,handlePair3

    MOV #CHAN_CFG,w0                         ; Configure AN0-AN7 as analog input channels
    MOV w0, ADPCFG

    CLR ADSTAT                               ; Clear all conversion status bits
    CLR ADCPC0                               ; Set up Pair 0 and Pair 1 for software trigger
    CLR ADCPC1                               ; Set up Pair 2 and Pair 3 for software trigger

    BSET ADCPC0, #15                         ; IRQEN1 = 1, Pair 1 Interrupt Enabled
    BSET ADCPC0, #8                          ; TRGSRC1 = 1, Individual Software Trigger
    BSET ADCPC0, #7                          ; IRQEN0 = 1, Pair 0 Interrupt Enabled
    BSET ADCPC0, #0                          ; TRGSRC0 = 1, Individual Software Trigger

    BSET ADCPC1, #15                         ; IRQEN3 = 1, Pair 3 Interrupt Enabled
    BSET ADCPC1, #8                          ; TRGSRC3 = 1, Individual Software Trigger
    BSET ADCPC1, #7                          ; IRQEN2 = 1, Pair 2 Interrupt Enabled
    BSET ADCPC1, #0                          ; TRGSRC2 = 1, Individual Software Trigger
```

Example 31-4: Vectored Interrupt From ADBASE Register – Assembly Code (Continued)

```
    BSET ADCON, #ADON                    ; Turn the ADC module ON.

    BCLR IFS0,#ADIF                      ; Clear the AD interrupt Status Flag
    BSET IEC0,#ADIE                      ; Enable the AD interrupt
    BSET IPC2,#ADIP2                     ; AD Interrupt Priority = 4
    BCLR IPC2,#ADIP1
    BCLR IPC2,#ADIP0

    BSET ADCPC0,#5                       ; Trigger Pair 0
    BSET ADCPC0,#13                      ; Trigger Pair 1
    BSET ADCPC1,#5                       ; Trigger Pair 2
    BSET ADCPC1,#13                      ; Trigger Pair 3

again:

    BRA again                            ; Loop here.


__ADCInterrupt:

    BCLR IFS0,#ADIF                      ; Clear the AD interrupt flag
    MOV ADBASE, w1                       ; Get the Jump Table location
    MOV [w1],w0                          ; Contents of this location is the
    GOTO w0                              ; address of the Pair Handler ISR

Pair0Handler:

    BCLR ADSTAT, #0                      ; Clear Pair 0 conversion status bit
    MOV ADCBUF0, w2                      ; Read channel 0 conversion result
    MOV ADCBUF1, w3                      ; Read channel 1 conversion result
    RETFIE                               ; Return from interrupt

Pair1Handler:

    BCLR ADSTAT, #1                      ; Clear Pair 1 conversion status bit
    MOV ADCBUF2, w2                      ; Read channel 2 conversion result
    MOV ADCBUF3, w3                      ; Read channel 3 conversion result
    RETFIE                               ; Return from interrupt

Pair2Handler:

    BCLR ADSTAT,#2                       ; Clear Pair 2 conversion status bit
    MOV ADCBUF4, w2                      ; Read channel 4 conversion result
    MOV ADCBUF5, w3                      ; Read channel 5 conversion result
    RETFIE                               ; Return from interrupt

Pair3Handler:

    BCLR ADSTAT, #3                      ; Clear Pair 3 conversion status bit
    MOV ADCBUF6, w2                      ; Read channel 6 conversion result
    MOV ADCBUF7, w3                      ; Read channel 7 conversion result
    RETFIE                               ; Return from interrupt

    .end
```

Example 31-5: Interrupt Triggered by PWM

```
#include "p30F2020.h"

int main(void)
{
    PTPER = 2048;                              /* PWM Period = 2.199 usec @ 29.1 MIPS */
                                               /* Refer to PWM section for more details*/

    /* Initialize PWM Generator 1 */

    IOCON1bits.PENH = 1;                       /* PWM Module controls High output*/
    IOCON1bits.PENL = 1;                       /* PWM Module controls Low output*/
    IOCON1bits.POLH = 0;                       /* High Output Polarity is active High*/
    IOCON1bits.POLL = 0;                       /* Low Output Polarity is active High*/
    IOCON1bits.PMOD = 1;                       /* Independent output mode*/
    IOCON1bits.OVRENH = 0;                     /* High Output Override disabled*/
    IOCON1bits.OVRENL = 0;                     /* Low Output Override disabled*/

    TRGCON1bits.TRIGDIV = 0;                   /* Trigger on every event*/
    TRGCON1bits.TRGSTRT = 0;                   /* Start the counting at the start*/

    TRIG1 = 200;/                              * Trigger event at 0.214 usec from */
                                               /* start of the PWM cycle*/

    PWMCON1bits.FLTSTAT = 0;                   /* Clear Fault Interrupt flag*/
    PWMCON1bits.CLSTAT = 0;                    /* Clear Current Limit Interrupt flag*/
    PWMCON1bits.TRGSTAT = 0;                   /* Clear PWM Trigger Interrupt flag*/
    PWMCON1bits.FLTIEN = 0;                    /* Disable Fault Interrupt*/
    PWMCON1bits.CLIEN = 0;                     /* Disable Current Limit Interrupt*/
    PWMCON1bits.TRGIEN = 0;                    /* Disable Trigger Interrupt*/
    PWMCON1bits.ITB = 0;                       /* Time base is read from PTMR*/
    PWMCON1bits.MDCS = 0;                      /* Duty cycle is read from PDC*/
    PWMCON1bits.DTC = 2;                       /* No Dead Time*/
    PWMCON1bits.XPRES = 0;                     /* No external reset for PTMR*/
    PWMCON1bits.IUE = 0;                       /* Immediate update to PDC*/

    PDC1 = 128;                                /* Start with a Ton value of 0.137usec*/
    PHASE1 = 0;                                /* No staggering*/

    /* Initialize the ADC */

    ADCONbits.ADSIDL = 0;                      /* Operate in Idle Mode*/
    ADCONbits.FORM = 0;                        /* Output in Integer Format*/
    ADCONbits.EIE = 1;                         /* Enable Early Interrupt*/
    ADCONbits.ORDER = 0;                       /* Even channel first*/
    ADCONbits.SEQSAMP = 1;                     /* Sequential Sampling Enabled*/
    ADCONbits.ADCS = 5;                        /* Clock Divider is set up for Fadc/14*/

    ADPCFG = 0xFFFC;                           /* AN0 and AN1 are analog inputs.*/
    ADSTAT = 0;                                /* Clear the ADSTAT register*/
    ADCPC0bits.TRGSRC0 = 0x4;                  /* Trigger conversion on PWM#1 Trigger*/
    ADCPC0bits.IRQEN0 = 1;                     /* Enable the interrupt*/

    ADCONbits.ADON = 1;                        /* Start the ADC module*/

    /* Set up the Interrupts */

    IFS0bits.ADIF = 0;                         /* Clear AD Interrupt Flag*/
    IPC2bits.ADIP = 4;                         /* Set ADC Interrupt Priority*/
    IEC0bits.ADIE = 1;                         /* Enable the ADC Interrupt*/

     PTCON = 0x8000;                           /* Enable PWM Module*/

    while(1);


}
```

Example 31-5: Interrupt Triggered by PWM (Continued)

```
void __attribute__((__interrupt__)) _ADCInterrupt()
{
    int channel0Result;

    IFS0bits.ADIF = 0;                            /* Clear ADC Interrupt Flag*/
    ADSTATbits.P0RDY = 0;                         /* Clear the ADSTAT bits*/
    channel0Result = ADCBUF0;                     /* Get the conversion result*/

    /* Update the Duty cycle with value read from AN0 */
    /* PDC value will be such that 7F >= PDC1 >= 3F0 */

    PDC1 = (channel0Result >= 0x03F0) ? 0x03F0 :
             ((channel0Result <= 0x007F) ? 0x007F : channel0Result);

}
```

31.24 Special Function Registers Associated with the ADC

Table 31-2 lists ADC Special Function registers, including their addresses and formats.

All unimplemented registers, and/or unimplemented bits within a register, read as zeros.

Table 31-2: ADC Register Map

File Name	ADR	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset States
ADCON	0300	ADON	—	ADSIDL	—	—	GSWTRG	—	FORM	EIE	ORDER	SEQSAMP	—	—	ADCS<2:0>			0009
ADPCFG1	0302	—	—	—	—	PCFG11	PCFG10	PCFG9	PCFG8	PCFG7	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0	0000
Reserved	0304	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
ADSTAT	0306	—	—	—	—	—	—	—	—	—	—	P5RDY	P4RDY	P3RDY	P2RDY	P1RDY	P0RDY	0000
ADBASE	0308	ADBASE<15:1>															—	0000
ADCPC0	030A	IRQEN1	PEND1	SWTRG1	TRGSRC1<4:0>					IRQEN0	PEND0	SWTRG0	TRGSRC0<4:0>					0000
ADCPC1	030C	IRQEN3	PEND3	SWTRG3	TRGSRC3<4:0>					IRQEN2	PEND2	SWTRG2	TRGSRC2<4:0>					0000
ADCPC2	030E	IRQEN5	PEND5	SWTRG5	TRGSRC5<4:0>					IRQEN4	PEND4	SWTRG4	TRGSRC4<4:0>					0000
Reserved	0310 – 031E	—-	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
ADCBUF0	0320	—	—	—	—	—	—	ADC Data Buffer 0										xxxx
ADCBUF1	0322	—	—	—	—	—	—	ADC Data Buffer 1										xxxx
ADCBUF2	0324	—	—	—	—	—	—	ADC Data Buffer 2										xxxx
ADCBUF3	0326	—	—	—	—	—	—	ADC Data Buffer 3										xxxx
ADCBUF4	0328	—	—	—	—	—	—	ADC Data Buffer 4										xxxx
ADCBUF5	032A	—	—	—	—	—	—	ADC Data Buffer 5										xxxx
ADCBUF6	032C	—	—	—	—	—	—	ADC Data Buffer 6										xxxx
ADCBUF7	032E	—	—	—	—	—	—	ADC Data Buffer 7										xxxx
ADCBUF8	0330	—	—	—	—	—	—	ADC Data Buffer 8										xxxx
ADCBUF9	0332	—	—	—	—	—	—	ADC Data Buffer 9										xxxx
ADCBUF10	0334	—	—	—	—	—	—	ADC Data Buffer 10										xxxx
ADCBUF11	0336	—	—	—	—	—	—	ADC Data Buffer 11										xxxx
Reserved	0338 – 037E	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000

Legend: x = unknown

Note: All interrupt sources and their associated control bits may not be available on a particular device. Refer to the device data sheet for details.

31.25 Design Tips

Question 1: ***How can I optimize the system performance of the ADC?***

Answer: Make sure you are meeting all timing specifications. If you are turning the module off and on, there is a minimum delay before the module can process a conversion request. Set up the ORDER and SEQSAMP bits so that you obtain the best combination of sample and conversion times. The ADC speed is limited to 2 Msps. If you try to trigger a conversion on a pair that is currently being processed, that trigger is not processed.

Question 2: ***Do you know of a good reference on A/D conversions?***

Answer: A good reference for understanding A/D conversions is the "*Analog-Digital Conversion Handbook*," third edition, published by Prentice Hall (ISBN 0-13-03-2848-0).

Question 3: ***Does the ADC module always require 24 ADC clock cycles to complete a conversion?***

Answer: The ADC module tries to best utilize the available resources for completing a conversion. Two channels can be sampled simultaneously, but only one can be converted at a time. In such a case, the pair conversion time would be 22 clock cycles. 24 clock cycles is the time required for conversion when ORDER = `0` and SEQSAMP = `0,` assuming that the sample and convert circuits are available.

31.26 Revision History

Revision A

This is the initial released revision of this document. (January 2007)

NOTES: