World's Fastest Embedded Interleaved 12-bit ADC Using PIC32MZ and PIC32MK Families

Interleaved ADC Performance Graph

Figure 1.

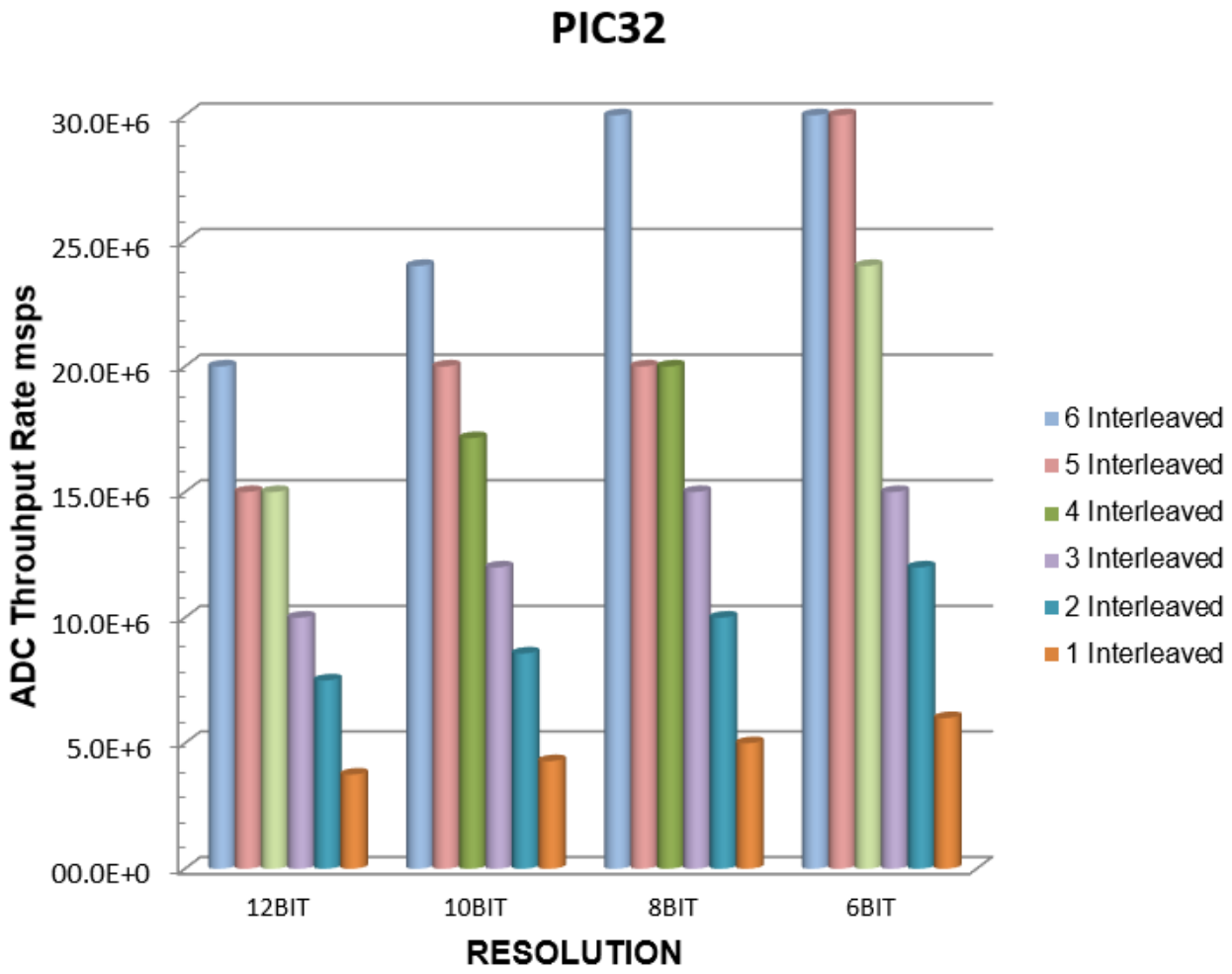


Table of Contents

1. Introduction

The PIC32MZ and PIC32MK device families have an advanced Class_1 12-bit ADC with features that enable them to be interleaved such that the composite ADC through-put rate can far exceed any individual ADC through-put rate. This is possible because each family has anywhere from 5-6 separate Class_1 ADC's that have independent trigger sources. By interleaving, (i.e. staggering), the trigger events and connecting the target analog signal to multiple interleaved ADC inputs in parallel, it's possible to achieve the rates listed below in the respective product family tables. Complete lists of ADC selectable high speed throughput rates are provided in Table 3-5 and Table 3-6 to assist users with proper timing and setup. It has been assumed that the user is utilizing the recommended optimum settings.

Table 1-1. PIC32MZ Family Max ADC Throughput Rate Summary

# of Interleaved ADC Possible	ADC TAD(1)(min) = 20 ns (50 Mhz max spec as of 7/1/18)			
	12-bit (max) msps	10-bit (max) msps	8-bit (max) msps	6-bit (max) msps
1	3.125 msps	3.57143 msps	4.16667 msps	5.00 msps
2	6.25 msps	7.14286 msps	8.33333 msps	10.00 msps
3	8.333333 msps	10.00 msps	12.50 msps	12.50 msps
4	12.50 msps	14.28571 msps	16.66667 msps	20.00 msps

Table 1-2. PIC32MZ Settings Assumptions

SYSCLK	200 Mhz (5 ns)
ADC / Timer3 PB3DIV<PBDIV>	2:1 ratio
ADCCON3<CONCLKDIV>	SYSCLK/2
ADCxTIME<ADCDIV> (ADC TAD CLOCK)	50 Mhz (20 ns) (1)
ADCxTIME<SAMC>	3 TAD
VDD=AVDD	> 2.5v
CFGCON< IOANCPEN> & ADCCON1<AICPMPEN>	0

Note: x = Respective interleaved ADC(s) utilized.

Table 1-3. PIC32MZ Possible ADC Interleaved Grouping Combinations

PIC32MZ	ADC0	ADC1	ADC2	ADC3	ADC4	ADCxTrigger Combinations	12-bit Interleaved Combined Throughput Rate(TAD = 50 Mhz)	12-bit Interleaved Combined Throughput Rate(TAD = 63 Mhz)
Group 1	X	X	---	---	---	OC5 & TMR3	6.25 msps	7.875 msps (1)
	---	---	X	X	---	OC1 & TMR5	6.25 msps	7.875 msps(1)
Group 2	X	X	X	---	---	OC1, OC3 & TMR5	8.333333 msps	10 msps(1)
	---	---	---	X	X	OC5 & TMR3	6.25 msps	7.875 msps(1)
Group 3	X	X	X	X	---	OC1, OC3, OC5, TMR3	12.5 msps(1)	15.75 msps(1)

Note:

1. In the current data sheet, ADC TAD spec (DS60001320), it has been mentioned as 20 ns (50Mhz) spec. However, the PIC32MZ EF family is currently being evaluated to test the feasibility of increasing the ADC TAD clock from 50 Mhz to 63 Mhz in support of CPU 252MHz operation. Refer to the current device data sheet, ADC TAD spec (DS60001320), which is available for download from the Microchip web site and then see Table 3-5 and Table 3-6 accordingly to the ADC throughput rates based on number of interleaved ADC groupings.

Table 1-4. PIC32MK Family Max ADC Throughput Rate Summary

# of Interleaved ADC Possible	ADC TAD(min) = 16.667-ns (60-Mhx max)			
	12-bit (max) msps	10-bit (max) msps	8-bit (max) msps	6-bit (max) msps
1	3.75 msps	4.28571 msps	5.00 msps	6.00 msps
2	7.50 msps	8.57143E+6	10.0 msps	12.00 msps
3	10.00 msps	12.00 msps	15.00 msps	15.00 msps
4	15.00 msps	17.14286 msps	20.00 msps	24.00 msps
5 (2)	---(1)	20.00 msps	---(1)	30.00 msps
6 (3)	20.00 msps	24.00 msps	30.00 msps	---(1)

Note:

1. No improvement, same msps, from previous throughput rate with one less interleaved ADC.
2. Up to five interleaved ADCs are possible on PIC32MKxxGPxx family.
3. Up to six interleaved ADCs are possible on PIC32MKxxMCxx motor control variants.

Table 1-5. PIC32MK Optimum Settings Assumptions

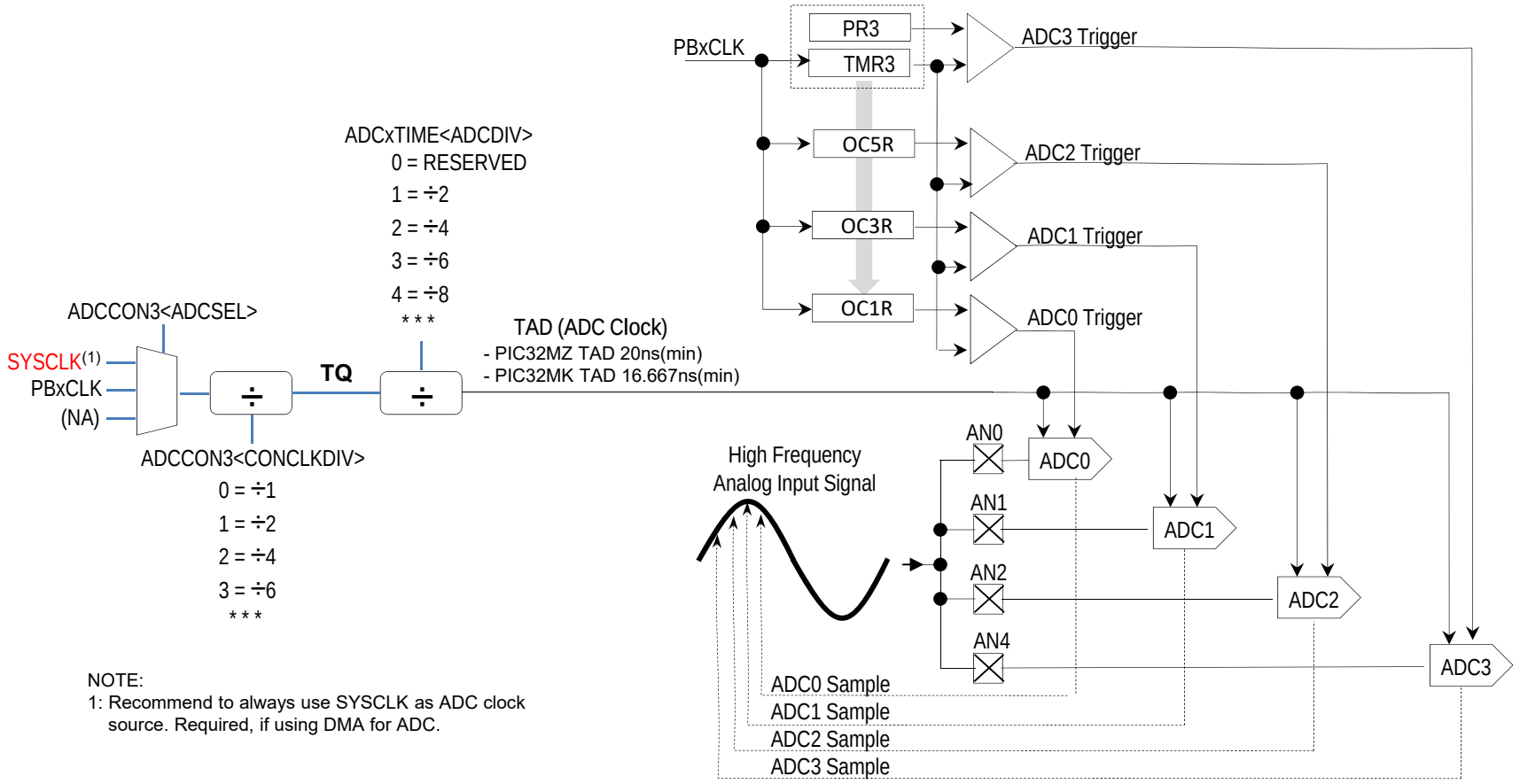
SYSCLK	120 Mhz (8.333333ns)
Timer3 PB2DIV<PBDIV>	120 Mhz (8.333333ns)
ADC PB5DIV<PBDIV>	120 Mhz (8.333333ns)
ADCCON3<CONCLKDIV>	120 Mhz (8.333333ns)
ADCxTIME<ADCDIV> (ADC TAD CLOCK)	60 Mhz (16.666666ns)
ADCxTIME<SAMC>	3 TAD
VDD=AVDD	> 2.5v
CFGCON< IOANCPEN> & ADCCON1<AICPMPEN>	0

Note: x = Respective interleaved ADC utilized.

Table 1-6. PIC32MK Possible ADC Interleaved Grouping Combinations

PIC32MK	ADC0	ADC1	ADC2	ADC3	ADC4	ADC5	ADCxTrigger Combinations	12-bit Interleaved ADC Combined Throughput rate (TAD = 60-Mhz)
Group 1	X	X	---	---	---	---	OC1 & TMR5	7.5 msps
	---	---	X	X	---	---	OC4 & TMR3	7.5 msps
	---	---	---	---	X	X	PWM Trig 1 & 2	7.5 msps
Group 2	X	X	X	---	---	---	OC1, OC2 & TMR3	10 msps
	---	---	---	X	X	X	PWM Trig 1, 2 & 3	10 msps
Group 3	X	X	---	---	---	---	OC1 and TMR3	7.5 msps
	---	---	X	X	X	X	PWM Trig 1, 2, 3 & 4	15 msps
Group 4	X	X	X	X	X		OC1,OC2,OC3,OC4 & TMR3	15 msps
Group 5	X	X	X	X	X	X	PWM Trig 1, 2, 3, 4, 5 & 6	20 msps

Figure 1-1. PIC32MZ Typical Four Interleaved ADC and ADC Trigger Clock Flow Block Diagram



NOTE:
1: Recommend to always use SYSCLK as ADC clock source. Required, if using DMA for ADC.

Table 1-7. Recommended ADC Clock / Timing Settings

	RECOMMENDED SETTINGS	
	PIC32MZ	PIC32MK
SYSCLK (Mhz)=	200	120
ADC PBxDIV<PBDIV>=	1	0
Timer3 PBxDIV<PBDIV>=	1	0
ADCCON3<ADCSEL>=	SYSCLK	SYSCLK
ADCCON3<CONCLKDIV>=	1	0
TQ=	10ns(1)	8.333333 ns
ADCxTIME<ADCDIV>=	1	1
ADC CLOCK (TAD)=	20ns(1)	16.666667 ns

Note:

1. . In the current data sheet, ADC TAD spec (DS60001320), it has been mentioned as 20 ns (50 Mhz) spec. However, the PIC32MZ EF family is currently being evaluated to test the feasibility of increasing the ADC TAD clock from 50 Mhz to 63 Mhz in support of CPU 252MHz operation. Refer to the current device data sheet, ADC TAD spec (DS60001320), which is available for download from the Microchip web site and then see Table 3-5 and Table 3-6 Table accordingly to the ADC throughput rates based on number of interleaved ADC groupings.

2. PIC32MZ/MK SAR ADC Class Types

Table 2-1.

ADC Classes	Analog Input	ADC Trigger Event	
CLASS_1 (ADC0-ADCx)	Dedicated Independent SAR ADC	Independent per ADC module	Ends sampling, Initiates ADC Conversion Cycle
CLASS_2 (ADC7)	Shared SAR ADC7	Independent per ADC7 ANx input	Initiates ADC Sample Cycle
CLASS_3 (ADC7)	Shared SAR ADC7	Shared for multiple AD7 ANx inputs	Initiates ADC Sample Cycle

2.1 ADC Class_1 Functional description

Class_1 ADCs are denoted with a dedicated single analog input. (some pseudo alternate inputs). Once a Class_1 ADC is enabled, it has only two states: sampling or converting. A trigger event initiates a conversion if the hardware enforced ADCxTIME<SAMC> time has expired. This is different than the shared ADC7 module with CLASS_2 & CLASS_3 analog input channels where the trigger starts the sampling rather than the conversion like on the Class_1.

Table 2-2. Dedicated Class_1 ADCs by PIC32MZ/MK Family

Dedicated Class_1 ADC Modules	Dedicated Analog Inputs	PIC32MZxxxx		PIC32MKxxxx	PIC32MKxxGPxx	PIC32MKxxMCxx
		Alternate Analog Inputs ADCTRGMODE <SHxALT>	# Of Synchronized Trigger Sources	Alternate Analog Inputs ADCTRGMODE <SHxALT>	# Of Synchronized Trigger Sources Available	# Of Synchronized Trigger Sources Available
0	AN0	AN45 (2)	4	AN3, AN5, AN24 (2)	5	6+
1	AN1	AN46 (2)		AN4, AN7, AN0 (2)		
2	AN2	AN47 (2)		AN5, AN6, AN25 (2)		
3	AN3	AN48 (2)		AN0, AN8, AN28 (2)		
4	AN4	AN49 (2)		AN1, AN9, AN0 (2)		
5 (1)	AN5 (1)	----	----	AN2, AN6, AN25 (2)		

Note:

1. Only the PIC32MKxxxx family has six Class_1 ADCs, and all other families have only five Class_1 ADCs.
2. Referring to the following figure and the previous table, regardless of what alternate analog input is selected by ADCTRGMODE<SHxALT> for the Class_1 ADCs, all ADC control and results are handled by the ADCx module as if it were the native dedicated default analog input, in column two of the previous table. For example, If ADCTRGMODE<SH0ALT> = ‘0b11= AN24, (PIC32MKxxxx), from a software and silicon hardware control and results register perspective, the user application must initialize and read ADC results from ADC0 module as if AN24 were actually AN0. None of the AN24 registers control or results functions will affect ADC0 behavior. This means the ADCx module and logic has no idea where the analog input originates after the ANx analog mux in the following figure and treats all control, status, trigger and results as if it were the native dedicated input.

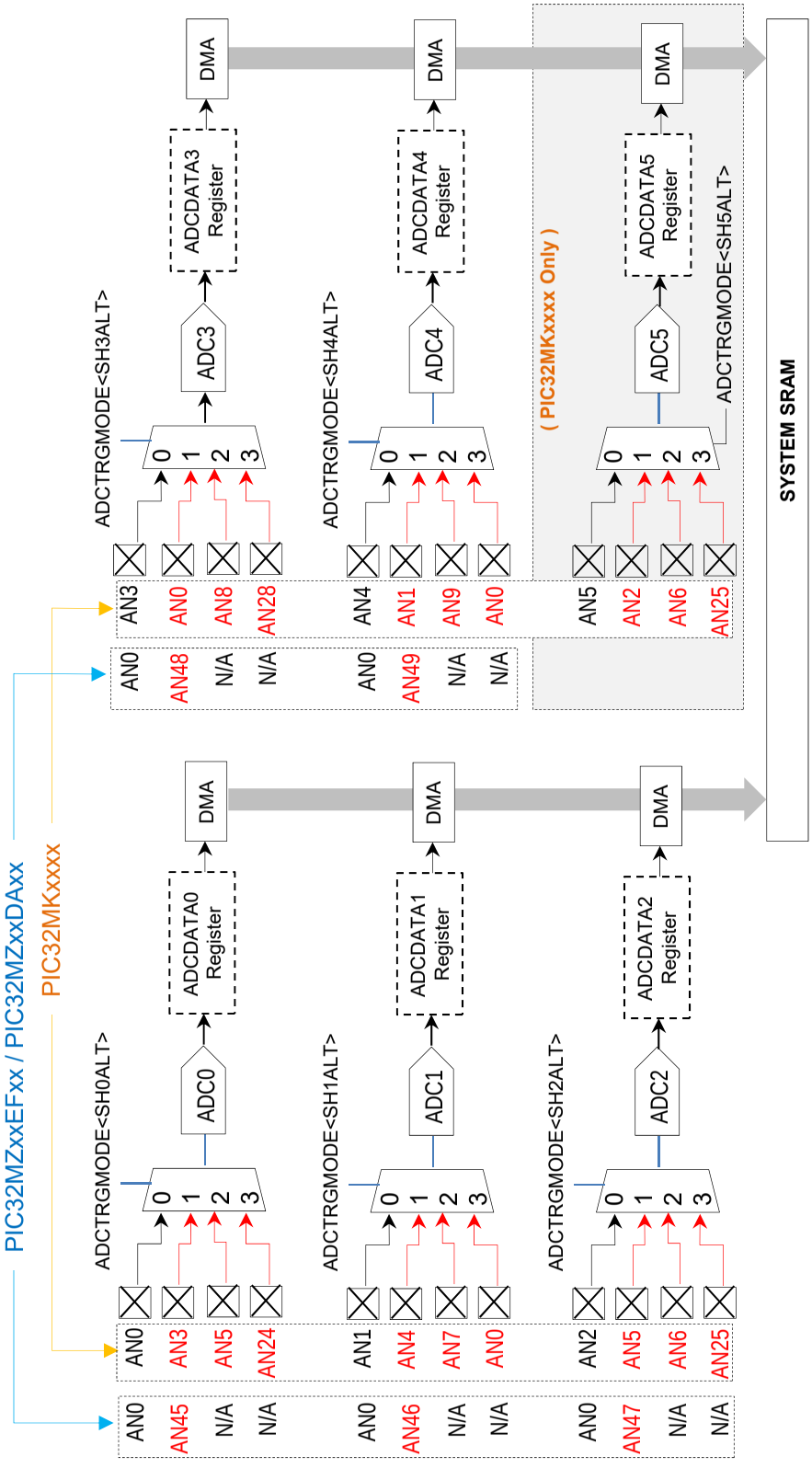


Figure 2-1. Alternate Dedicated CLASS_1 ADC Inputs

3. ADC Class_1 Triggers

Important requirements for high-speed interleaved ADC operation.

1. Always set the minimum sampling time of each Class_1 ADC equal to 3TAD as defined by ADCxTIME<SAMC>.
2. Ensure that the selected trigger source time base clock rate is twice faster than the ADC clock TAD rate. This will provide the highest timing resolution along with the most sampling rate options as defined in the tables given in the "Section 3.4 ADC Class_1 Triggers Key Learnings".
3. The selected trigger sources must be able to be mapped to a single synchronized time base. For example:
   - TMR3 & OCx (x=1,3,5 PIC32MZxxxx)
   - TMR3 & OCx (x=1-4 PIC32MKxxGPxx)
   - PTMRx & TRIGx (x=1-12 PIC32MKxxMCxx PWM)
   - PTMRx & STRIGx (x=1-12 PIC32MKxxMCxx PWM)
4. In the same group of interleaved ADCs, the user should not use two separate timer trigger sources, for example, ADC0, ADC1 and ADC2 are being used in an interleaved group.
   - Trigger sources can be any combination of valid ADC OCx edge triggered sources synchronized to a single TMRx. It is not allowed to use in the same group of interleaved ADCs OCx, for example, TMR3 & TMR5. The separate TMRx in different interleaved ADC groups is OK.
5. If using the DMA for the ADC, use SYSCLK as the ADC source clock, (i.e., ADCCON3<ADCSEL>).
6. Configure ADC for 512 TAD warm-up time (the ADCANCON bits.WKUPCLKCNT = 0x9) before enabling ADC.
7. Users software must wait for ADC BANDGAP ready, (ADCCON2 register), and ADC warm-up time, (ADCANCON register), after enabling ADC and before activating the ADC trigger sources.
8. ADC must always be initialized first before activating the ADC trigger sources.

⚠ CAUTION Failure to follow these eight recommendations will result in inaccurate ADC data acquisition and poor performance.

ADC interleaved trigger timing is a function of the peripheral clock of the user-selected trigger sources specifically the general purpose Timers, Output Compares and motor control PWMs, all of which can operate at a faster rate than the ADC TAD clock. Assuming the use of the max PBCLKx rate for the ADC trigger sources, (i.e. 2 * TAD frequency), whose peripheral clock frequency is controlled by their respective PBxDIV register, then the ADC trigger source period interval will be half of the ADC TAD period. (i.e., Trigger peripheral clock period = (TAD/2). Under this condition, the trigger intervals can be any multiple of a ½ TAD provided it's ≥ than the minimum TAD trigger interval as defined by Equation 2. This is important, as even though everything about the ADC is governed by the TAD clock periods, (i.e., fixed ADCxTIME<SAMC> sample time and the Conversion Time), the trigger sources are not. This allows a user to take advantage of one of the characteristics of a CLASS_1 ADC where the ADC continues to sample indefinitely even after the hardware enforced ADCxTIME<SAMC> sample time minimum (in TAD cycle times) until a trigger event. Therefore, the trigger peripheral clock frequency is twice the ADC TAD clock, it allowing sampling cycles effectively to be on multiples of ½ TAD boundaries that are ≥ the minimum timing interval as defined in Equation 2 and the tables given in the "Section 3. ADC Class_1 Triggers Key Learnings". This allows higher resolution of trigger intervals and access to more sample rates on 1/2 TAD boundaries instead of just TAD boundaries.

Before a trigger event or immediately after a conversion, a Class_1 ADC is always in the sampling state. A trigger event is persistent meaning that it will remain pending until after the hardware enforced sampling time defined by ADCxTIME<SAMC> (i.e. TAD sample time) has expired before allowing any trigger event to be processed. Once an enabled Class_1 ADC is triggered after the required ADCxTIME<SAMC>, the conversion cycle will immediately be initiated. The conversion half of the ADC cycle is defined by:

Equation 1: ADC Conversion Cycle time

ADC Conversion Cycle time = ((# bits resolution+1) * TAD)

TAD = ADC sample & conversion clock period

3.1 Class_1 ADC Hardware Trigger Timing Considerations

In the ADC edge selectable trigger sources (i.e. OCx, TIMRx and PWMx) in the ADCTRGx<TRGSRCy> register), mostly it has been assumed that the ADC trigger happens on the match condition, however, in reality, the ADC triggers happen on the next increment after a match.

The peripherals that act as the source of the ADC trigger must be enabled for the trigger to be active, but the corresponding I/O pin for the function does not, for example, the output compare peripheral functions are routed through the Peripheral Pin Select (PPS), matrix to the user defined I/O pin. In Output Compare when used as an ADC trigger that is not required. This leaves the I/O pin free to be assigned to any other appropriate alternate function by the users application even though the Output Compare module is enabled.

This is also true for even primary I/O pin functions (i.e., non-PPS) like PWM triggers in the PIC32MKxxMCxx family, for example, if a user wanted to use the PWMs as trigger sources for the ADC, and also did not want to use the PWM I/O pin functions, then they can clear all the appropriate IOCONx<PENH> = 0, (i.e., PWMxH), and IOCONx<PENL> = 0, (i.e., PWMxL) prior to enabling the PWM, (i.e., PTCON<PTEN> = 1).

Attention:
The Output Compare (OCx), ADC trigger is special in only one respect. The ADC trigger event only occurs on what would be a rising edge output compare signal. The Output Compare module mode for use with ADC trigger therefore must be:

OCxCON<OCM> = 0b101 = Initialize OCx low; generate continuous pulses with:

OC1R = TAD Trigger Timing Count and OC1RS = (OC1R+2)

This insures the OCx output starts low and generates an ADC trigger on the OCx rising edge on match+1 condition. OCx state then returns low (2) clocks later ready for the next ADC trigger event in the next interleaved sequence.

Note: Remember the Output Compare peripheral does not need to be mapped physically to a pin using the PPS for use as an internal ADC trigger source.

3.2 Class_1 Interleaved ADC Trigger Formulas

Equation 2: Minimum Interleaved ADC TAD Trigger Source Interval:

= ((SAMC Sample Time (min) + Conversion Time + ADJ) * TAD) / # of interleaved ADCs used

= (3TAD + (((#bits Resolution+1) + ADJ) * TAD)) / # of interleaved ADCs used

Important: ADJ term = A user selected whole number adjustment factor between 0-4 that must yield a *Minimum TAD Trigger interval* that yields any exact multiple of a ½ TAD for the result of Equation 2. The reason for ½ TAD is because the TMRx trigger clock source is assumed to be 2x faster or ½ ADC TAD clock period. Once the interleaved minimum trigger TAD is known, then any throughput rate including the minimum Trigger TAD in equation 2 in 0.5 TAD increments are possible. Refer to tables 3-5 through 3-7.

Equation 3: Interleaved ADC Throughput Rate Formula:

Interleaved ADC Throughput Rate = 1/(ADC TAD period * TAD Trigger Source spacing)

Note: *Trigger TAD spacing* value in Equation 3 must be ≥ Equation 2 Result, and an exact multiple of 0.5.

3.2.1 PIC32MZ / PIC32MK Interleaved ADCs Trigger Calculation Examples

EXAMPLE 1 of 4: (PIC32MZxxxx, 12-bit, 12.5 msps w/four Interleaved ADCs)

Trigger TAD spacing = (((Sample Time (min) + Conversion Time + *ADJ*) * TAD) / # of interleaved ADCs used)

= (((3TAD + (#bits Resolution+1) + *ADJ*) * TAD) / # of interleaved ADCs used)

= ((16 + *ADJ*) / 4)

= ((16 + 0) / 4) //ADJ=0 (user selected so equation result is an exact multiple of 0.5)

= 4.0 Trigger source TAD spacing

Interleaved ADC Throughput Rate = 1/(ADC TAD period * Trigger TAD spacing)

= 1/(20ns * 4.0)

= 12.50 msps (millions samples/sec)

Note: *ADJ* = A user selected whole number adjustment factor between zero and four that must yield any exact multiple of a ½ TAD for the result for *Trigger TAD spacing* provided it is ≥ Equation 2 minimum TAD Trigger Source spacing requirement.

EXAMPLE 2 of 4: (PIC32MZxxxx, 10bit 14.285714 msps w/four Interleaved ADCs)

Trigger TAD spacing (min) = (((Sample Time (min) + Conversion Time + *ADJ*) * TAD) / # of interleaved ADCs used)

= (((3TAD + (#bits Resolution+1) + *ADJ*) * TAD) / # of interleaved ADCs used)

= ((14 + *ADJ*) / 4)

= ((14 + 0) / 4) //ADJ=0 (user selected so equation result is an exact multiple of 0.5)

= 3.5 Trigger source TAD spacing

Interleaved ADC Throughput Rate = 1/(ADC TAD period * Trigger TAD spacing)

= 1/(20ns * 3.5)

= 14.285714 msps (millions samples/sec)

Note: *ADJ* = A user selected whole number adjustment factor between zero and four that must yield any exact multiple of a ½ TAD for the result for *Trigger TAD spacing* provided it is ≥ *Equation 2 minimum* TAD Trigger Source spacing requirement.

Table 3-1. PIC32MZ Trigger Source Initialization

x **= (2 * TAD Trigger Source spacing from Equation 3)**			
Peripheral Registers↓	**# Interleaved ADC**		
	2	**3**	**4**
OC1R=	(x – 1)	(x – 1)	(x – 1)
OC1RS=	(OC1R+2)	(OC1R+2)	(OC1R+2)
OC1CON=	0x800D	0x800D	0x800D
OC3R=	---	(2 * *x*) - 1	(2 * *x*) - 1
OC3RS=	---	(OC3R+2)	(OC3R+2)
OC3CON=	-	0x800D	0x800D
OC5R=	-	-	(3 * *x*) - 1
OC5RS=	-	-	(OC5R+2)
OC5CON=	-	-	0x800D
PR3=	(2 * *x*) - 1	(3 * *x*) - 1	(4 * *x*) - 1

Note: x = (2 * *TAD Trigger Source spacing* from Equation 3), *TAD Trigger Source spacing* value in equation 3 must be ≥ Equation 2 Result and an exact multiple of 0.5.

EXAMPLE 3 of 4: (PIC32MK, 12bit, 15msps with five Interleaved ADCs)

Trigger TAD spacing = (((Sample Time (min) + Conversion Time + *ADJ*) * TAD) / # of interleaved ADCs used)

= (((3TAD + (#bits Resolution+1) + *ADJ*) * TAD) / # of interleaved ADCs used)

= ((16 + *ADJ*) / 5)

= ((16 + 4) / 5) //ADJ=4 (user selected so equation result is an exact multiple of 0.5)

= 4.0 Trigger source TAD spacing

Interleaved ADC Throughput Rate = 1/(ADC TAD period * Trigger TAD spacing)

= 1/(16.666667ns * 4)

= 15.0 msps (millions samples/sec)

Note: *ADJ* = A user selected whole number adjustment factor between zero and four that must yield any exact multiple of a ½ TAD for the result for *Trigger TAD spacing* provided it's ≥ Equation 2 minimum TAD Trigger Source spacing requirement.

Table 3-2. PIC32MKxxxx Trigger Source Initialization (Five Interleaved ADC Example)

x = (2 * TAD Trigger Source spacing from Equation 3)				
Peripheral Registers↓	# Interleaved ADC			
	2	3	4	5
OC1R=	(x – 1)	(x – 1)	(x – 1)	(x – 1)
OC1RS=	(OC1R+2)	(OC1R+2)	(OC1R+2)	(OC1R+2)
OC1CON=	0x800D	0x800D	0x800D	0x800D
OC2R=	---	(2 * *x*) - 1	(2 * *x*) - 1	(2 * *x*) - 1
OC2RS=	---	(OC3R+2)	(OC3R+2)	(OC3R+2)
OC2CON=	-	0x800D	0x800D	0x800D
OC3R=	-	-	(3 * *x*) - 1	(3 * *x*) - 1
OC3RS=	-	-	(OC5R+2)	(OC5R+2)
OC3CON=	-	-	0x800D	0x800D
OC4R=	-	-	-	(4 * *x*) - 1
OC4RS=	-	-	-	(OC5R+2)
OC4CON=	-	-	-	0x800D
PR3=	(2 * *x*) - 1	(3 * *x*) - 1	(4 * *x*) - 1	(5 * *x*) - 1

Note: x= (2 * *TAD Trigger Source spacing* from Equation 3), *TAD Trigger Source spacing* value in equation 3 must be ≥ Equation 2 Result and an exact multiple of 0.5.

EXAMPLE 4 of 4: (PIC32MKxxMCxx, 12-bit, 20 msps with six Interleaved ADCs)

Trigger TAD spacing = (((Sample Time min + Conversion Time + *ADJ*) * TAD) / # of interleaved ADCs used)

= (((3TAD + (#bits Resolution+1) + *ADJ*) * TAD) / # of interleaved ADCs used)

= ((16 + *ADJ*) / 6)

= ((16 + 2) / 6) //ADJ=2 (user selected so equation result is an exact multiple of 0.5)

= 3.0 Trigger source TAD spacing

Interleaved ADC Throughput Rate = 1/(ADC TAD period * Trigger TAD spacing)

= 1/(16.666667ns * 3)

= 20.0 msps (millions samples/sec)

Note: *ADJ* = A user selected whole number adjustment factor between zero and four that must yield any exact multiple of a ½ TAD for the result for *Trigger TAD spacing* provided it is ≥ Equation 2 minimum *TAD Trigger Source spacing* requirement.

Table 3-3. PIC32MKxxMCxx Trigger Source Initialization (Six interleaved ADC Example)

x = (2 * TAD Trigger Source spacing from Equation 3)					
Peripheral Registers↓	# Interleaved ADC				
	2	3	4	5	6
TRIG1=	(x – 1)	(x – 1)	(x – 1)	(x – 1)	(x – 1)
TRIG2=	---	(2 * *x*) - 1	(2 * *x*) - 1	(2 * *x*) - 1	(2 * *x*) - 1
TRIG3=	---	---	(3 * *x*) - 1	(3 * *x*) - 1	(3 * *x*) - 1
TRIG4=	---	---	---	(4 * *x*) - 1	(4 * *x*) - 1
TRIG5=		---	---	---	(5 * *x*) - 1
TRIG6=	(2 * *x*) - 1	(3 * *x*) - 1	(4 * *x*) - 1	(5 * *x*) - 1	(6 * *x*) - 1
PTPER=	(2 * *x*) - 1	(3 * *x*) - 1	(4 * *x*) - 1	(5 * *x*) - 1	(6 * *x*) - 1

Note: x= (2 * *TAD Trigger Source spacing* from Equation 3), *TAD Trigger Source spacing* value in equation 3 must be ≥ Equation 2 result and an exact multiple of 0.5.

The various triggers sources must be able to be synchronized to the same time base. Therefore, the number of Class_1 ADCs and triggers that can be combined using the interleave technique depends on the available trigger sources defined in ADCTRGx<TRGSRC> that meet the requirement just described.

Consider the following Class_1 PIC32MZ and PIC32MK individual ADC available trigger sources as defined in ADCTRGx<TRGSRC>:

Table 3-4. Available ADC CLASS_1 Trigger Sources for Each Individual Class_1 ADCs

PIC32MZ EF Family	PIC32MZ DA Family	PIC32MK MCF Family	PIC32MK MCM Family
5 Class_1 ADCs	5 Class_1 ADCs	6 Class_1 ADCs	6 Class_1 ADCs
Register ADCTRGx<TRGSRCy>: ADCx Trigger Source for Conversion of Analog Input ANx			
11111 = Reserved	11111= Reserved	11111 = Reserved	11111 = Reserved
11110 = Reserved	11110= Reserved	11110 = PWM Generator 12 trigger	11110 = PWM Generator 12 trigger
11101 = Reserved	11101= Reserved	11101 = PWM Generator 11 trigger	11101 = PWM Generator 11 trigger
11100 = Reserved	11100= Reserved	11100 = PWM Generator 10 trigger	11100 = PWM Generator 10 trigger
11011= Reserved	11011= Reserved	11011 = PWM Generator 4 Current-Limit	11011 = PWM Generator 4 Current-Limit
11010= Reserved	11010= Reserved	11010 = PWM Generator 3 Current-Limit	11010 = PWM Generator 3 Current-Limit
11001= Reserved	11001= Reserved	11001 = PWM Generator 2 Current-Limit	11001 = PWM Generator 2 Current-Limit
11000= Reserved	11000= Reserved	11000 = PWM Generator 1 Current-Limit	11000 = PWM Generator 1 Current-Limit
10111= Reserved	10111= Reserved	10111 = PWM Generator 9 trigger	10111 = PWM Generator 9 trigger
10110= Reserved	10110= Reserved	10110 = PWM Generator 8 trigger	10110 = PWM Generator 8 trigger
10101= Reserved	10101= Reserved	10101 = PWM Generator 7 trigger	10101 = PWM Generator 7 trigger
10100= CTMU trip	10100= CTMU trip	10100 = CTMU trip	10100 = CTMU trip
10011= Reserved	10011= Reserved	**10011 = Output Compare 4 rising edge**	**10011 = Output Compare 4 rising edge**
10010= Reserved	10010= Reserved	**10010 = Output Compare 3 rising edge**	**10010 = Output Compare 3 rising edge**
10001= Reserved	10001= Reserved	**10001 = Output Compare 2 rising edge**	**10001 = Output Compare 2 rising edg**
10000= Reserved	10000= Reserved	**10000 = Output Compare 1 rising edge**	**10000 = Output Compare 1 rising edge**
01111= Reserved	01111= Reserved	**01111 = PWM Generator 6 trigger**	**01111 = PWM Generator 6 trigger**
01110 = Reserved	01110 = Reserved	**01110 = PWM Generator 5 trigger**	**01110 = PWM Generator 5 trigger**
01101 = Reserved	01101= Reserved	**01101 = PWM Generator 4 trigger**	**01101 = PWM Generator 4 trigger**
01100 = Comparator 2 (COUT)	01100 = Comparator 2 (COUT)	**01100 = PWM Generator 3 trigger**	**01100 = PWM Generator 3 trigger**
01011 = Comparator 1 (COUT)	01011= Comparator 1 (COUT)	**01011 = PWM Generator 2 trigger**	**01011 = PWM Generator 2 trigger**
01010 = OCMP5	**01010 = OCMP5**	**01010 = PWM Generator 1 trigger**	**01010 = PWM Generator 1 trigger**
01001 = OCMP3	**01001= OCMP3**	**01001 = Secondary PWM time base**	**01001 = Secondary PWM time base**
01000 = OCMP1	**01000 = OCMP1**	**01000 = Primary PWM time base**	**01000 = Primary PWM time base**
00111 = TMR5 match	**00111= TMR5 match**	**00111 = General Purpose Timer5**	**00111 = General Purpose Timer5**
00110 = TMR3 match	**00110 = TMR3 match**	**00110 = General Purpose Timer3**	**00110 = General Purpose Timer3**
00101 = TMR1 match	00101= TMR1 match	00101 = General Purpose Timer1	00101 = General Purpose Timer1
00100 = INT0 External interrupt	00100 = INT0 External interrupt	00100 = INT0	00100 = INT0
00011 = STRIG	00011= STRIG	00011 = Scan trigger	00011 = Scan trigger
00010 = Global level software trigger	00010 = Global level software trigger	00010 = Software level trigger	00010 = Software level trigger
00001 = Global software edge Trigger	00001= Global software edge Trigger	00001 = Software edge trigger	00001 = Software edge trigger
00000 = No Trigger	00000 = No Trigger 1 (COUT)	00000 = No trigger	00000 = No trigger
In both the PIC32MZxxEFxx and PIC32MZxxDAxx there are only four ADC trigger sources that can be synchronized to a common time base as highlighted in the previous table, which limits the maximum number of ADCs that can be interleaved even though there are five Class_1 ADCs available. In the case of the PIC32MKxxGPxx family there are five sources, and for the PIC32MKxxMCxx family motor control variants with PWM triggers, all six available PIC32MKxxxx Class_1 ADCs could be interleaved because any of the PWMx generator triggers can be synchronized with the PWM primary or secondary time base.			

3.3 ADC Class_1 Trigger Misconceptions

The minimum ADC Class_1 sampling time is 3 TAD. A common mistake that users frequently make is assuming that if they stagger the ADC triggers by the minimum TAD sample time listed in the data sheet that they can achieve the fastest interleaved throughput rate. Referring to the following figure you can see that assumption is only valid for the first series of samples, (four samples, ADC0-3). When the second OC1 trigger occurs for ADC0 it is still in the conversion process from the previous trigger event. Recall that in Class_1 ADCs that even though the trigger event is persistent it *will not* be acknowledged until any conversion cycle already in progress has completed plus the ADCxTIME<SAMC> sample time has expired for any given ADC. This means for example that even though the 2nd ADC0 Output Compare 1 trigger occurs at the end of the 12th ADC0 TAD conversion cycle, ADC0 will not actually be triggered until 4TAD later, (i.e., 13th conversion TAD + 3TAD sample time). This is confirmed by comparing the 2nd expected OC1 trigger event to the corresponding ACTUAL SAMPLE/CONV EVENT timing event. The SAMPLING ERROR timing represents this error relationship. The sample error is the same for all the interleaved ADCs. Notice also that the error is cumulative and compounds itself for every ADC0-3 series of samples. After just a few cycles the interleaved ADC throughput rate that the user thought was a constant [1/(3*TAD)] is actually not constant at all and in fact variable with an increasing phase shift every series of interleaved conversions constantly compounding the measurement error. To maintain a consistent ADC throughput sampling rate it is critical to follow equation 2 :

Equation 2: Minimum Interleaved ADC TAD Trigger Source Interval:

= ((SAMC Sample Time min + Conversion Time + *ADJ*) * TAD) / # of interleaved ADCs used

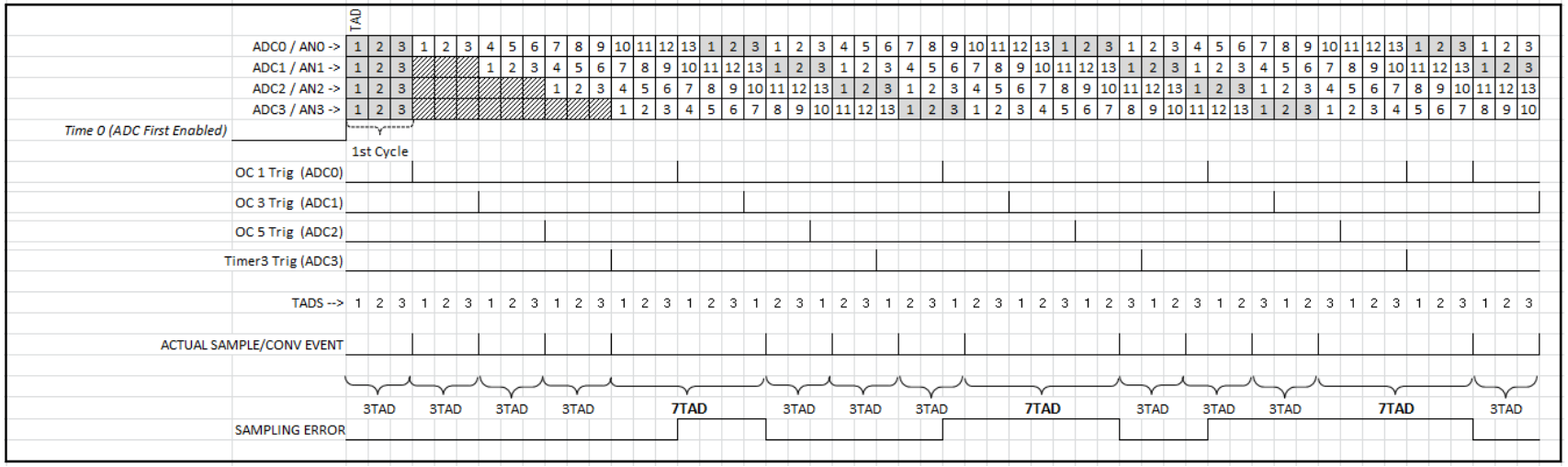
= (3TAD + (((#bits Resolution+1) + *ADJ*) * TAD)) / # of interleaved ADCs used

Important: *ADJ term* = A user selected whole number adjustment factor between 0-4 that must yield a *Minimum TAD Trigger interval* that yields any exact multiple of a ½ TAD for the result of Equation 2. The reason for ½ TAD is because the TMRx trigger clock source is assumed to be two times faster or ½ ADC TAD clock period. Once the interleaved minimum trigger TAD is known, then any throughput rate including the minimum Trigger TAD in equation 2 in 0.5 TAD increments are possible.

Note: Following these rules will insure a deterministic interleaved sampling rate as shown in Combined 10msps ADC0-3 Interleaved FFT Result which is also baked into the Interleaved ADC Throughput Rate tables in "Section 3. ADC Class_1 Triggers Key Leanings".

Another misconception is that users think they can use multiple timers as trigger sources in the same group of interleaved ADCs. (See ADC Class_1 Triggers: Important Rules and Requirements, rule #4.

Figure 3-1. Erroneous 12bit ADC Class_1 Trigger Example



3.4 ADC Class_1 Triggers Key Learnings

- All PIC32 ADC trigger sources whether timers, output compares or PWMs follow the same rules. The trigger and interrupt event is generated on match+1 count. Therefore, for timers the value should be PRx-1, for Output compares OCxR-1 and PWM's TRIGx-1 and/or STRIGx-1
- When using interleaved method, the available throughput rates so as to not create any sampling phase shift errors, it is critical that the user always set the minimum sampling time of each ADC = 3TAD as defined by ADCxTIME<SAMC> AND the staggered trigger intervals to only those defined in the tables in PIC32MZ, 12bit 12.5msps and 10bit 14.285714 msps Interleaved ADCs Trigger Calculation Examples based on the listed available throughput rates. ANYTHING else will cause non equidistant triggers from one interleaved ADC group sequence cycle to the next interleaved group cycle producing phase shifted results and therefore sampling rate errors.
- ADC Class_1 edge trigger event(s) are persistent until after the sampling time has expired as defined by ADCxTIME<SAMC> for a particular Class_1 ADC Module, at which time the trigger event is cleared and initiates a start of conversion sequence for the respective target Class_1 ADC.
- When using interleaved techniques, the available ADC throughput rates for no sampling phase shift errors, must be ≥ Equation 2 result and be an exact multiple of 0.5 assuming the user is using the recommended clock setting as described in PIC32MK Optimum Settings Assumptions and throughout this document.
- In the PIC32MK series, the PWM Timer, (i.e. PTMR), unlike the general purpose TIMERS, the PWM period timer hardware automatically rolls over to on a PTMR to PTPER match not on a match +1. This is important to note because if you ever set a TRIGx = PTPER the match and thus trigger would never happen as the PTMR hardware rolls over on PTPER-1. Example: PTPER=500, PTMR will only count from 0-499 before the hardware will roll it over back to 0x0000.
- The user SHOULD always set the minimum sampling time of each Class_1 ADC equal to 3TAD as defined by ADCxTIME<SAMC>.
- The user should insure that the selected trigger source time base clock rate is two times faster than the ADC clock TAD rate. This will provide the highest timing resolution along with the most sampling rate options as defined in Tables in PIC32MZ, 12bit 12.5msps and 10bit 14.285714 msps Interleaved ADCs Trigger Calculation Examplesand Class_1 ADC Trigger Event Behavior Questions and Answers.
- The user selected trigger sources have to be able to be mapped to a single synchronized time base to maintain equidistant periodic trigger separation between all interleaved ADC's spanning all ADC cycles. For example:
  - TMR3 & OCx (x=1,3,5 PIC32MZxxxx)
  - TMR3 & OCx (x=1-4 PIC32MKxxGPxx)
  - PTMRx & TRIGx (x=1-12 PIC32MKxxMCxx PWM)
  - PTMRx & STRIGx (x=1-12 PIC32MKxxMCxx PWM)
- If using the DMA for the ADC the user *MUST* use SYSCLK as the ADC source clock, (i.e. ADCCON3<ADCSEL>).
- User must configure ADC for 512 TAD warm-up time (ADCANCONbits.WKUPCLKCNT = 0x9) *BEFORE* enabling ADC.
- Users SW must wait for ADC BANDGAP ready, (ADCCON2 register), and ADC warm-up time, (ADCANCON register), *AFTER* enabling ADC and before activating the ADC trigger sources.
- ADC and DMA must always be initialized 1st *BEFORE* activating the ADC trigger sources.

3.5 Class_1 ADC Trigger Event Behavior Questions and Answers

1. What happens if the ADC Class_1 trigger event occurs during but before the expiration of ADCxTIME<SAMC>?
   Trigger event will remain pending until the expiration of the hardware enforced ADCxTIME<SAMC> sample time. Immediately after which it will be processed by the hardware and trigger the ADC conversion and clear any pending trigger event. If multiple trigger events occur before expiration of ADCxTIME<SAMC> only one will remain pending causing only one conversion.
2. What happens if the trigger event occurs sometime after the expiration of ADCxTIME<SAMC>?
   ADC hardware will enforce user ADCxTIME<SAMC> sample time but will continue sampling for an indefinite amount of time until a trigger event is eventually received, at which time it will immediately initiate the ADC conversion cycle.
3. What happens if a trigger or multiple trigger events occur during ADC conversion in progress?
   If multiple triggers are received, only one will remain pending. An ADC conversion already in process will continue undisturbed through the end of the conversion cycle, and through to the end of the next ADCxTIME<SAMC> sample time cycle before it will be acknowledged and processed by the hardware.

Table 3-5. PIC32MZ Interleaved ADC Throughput Rates for Devices with TAD Electrical Specifications ADC Clock Period = 20ns(min) (50Mhz)

Interleaved ADCs	TAD (min) 50Mhz	12 bit Mode		10 bit Mode		8 bit Mode		6 bit Mode	
		TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)
4	20ns(1)	4.0 (min)	12.5E+6	3.5 (min)	14.285714E+6	3 (min)	16.666667E+6	2.5 (min)	20.0E+6
		4.5	11.111111E+6	4	12.5E+6	3.5	14.285714E+6	3	16.666667E+6
		5.0	10.0E+6	4.5	11.111111E+6	4	12.5E+6	3.5	14.285714E+6
		5.5	9.090909E+6	5	10.0E+6	4.5	11.111111E+6	4	12.5E+6
		6.0	8.333333E+6	5.5	9.090909E+6	5	10.0E+6	4.5	11.111111E+6
3	20ns(1)	6.0 (min)	8.333333E+6	5.0 (min)	10.0E+6	4 (min)	12.5E+6	4 (min)	12.5E+6
		6.5	7.692308E+6	5.5	9.090909E+6	4.5	11.111111E+6	4.5	11.111111E+6
		7.0	7.142857E+6	6.0	8.333333E+6	5	10.0E+6	5	10.0E+6
		7.5	6.666667E+6	6.5	7.692308E+6	5.5	9.090909E+6	5.5	9.090909E+6
		8.0	6.25E+6	7.0	7.142857E+6	6	8.333333E+6	6	8.333333E+6
2	20ns(1)	8.0 (min)	6.25E+6	7.0 (min)	7.142857E+6	6 (min)	8.333333E+6	5 (min)	10.0E+6
		8.5	5.882353E+6	7.5	6.666667E+6	6.5	7.692308E+6	5.5	9.090909E+6
		9.0	5.555556E+6	8.0	6.25E+6	7	7.142857E+6	6	8.333333E+6
		9.5	5.263158E+6	8.5	5.882353E+6	7.5	6.666667E+6	6.5	7.692308E+6
		10.0	5.00E+6	9.0	5.555556E+6	8	6.25E+6	7	7.142857E+6
		10.5	4.761905E+6	9.5	5.263158E+6	8.5	5.882353E+6	7.5	6.666667E+6
		11.0	4.545455E+6	10.0	5.0E+6	9	5.555556E+6	8	6.25E+6
		11.5	4.347826E+6	10.5	4.761905E+6	9.5	5.263158E+6	8.5	5.882353E+6
		12.0	4.166667E+6	11.0	4.545455E+6	10	5.0E+6	9	5.555556E+6
		12.5	4.00E+6	11.5	4.347826E+6	10.5	4.761905E+6	9.5	5.263158E+6
		13.0	3.846154E+6	12.0	4.166667E+6	11	4.545455E+6	10	5.0E+6
		13.5	3.703704E+6	12.5	4.0E+6	11.5	4.347826E+6	10.5	4.761905E+6
		14.0	3.571429E+6	13.0	3.846154E+6	12	4.166667E+6	11	4.545455E+6
		14.5	3.448276E+6	13.5	3.703704E+6	12.5	4.0E+6	11.5	4.347826E+6
		15.0	3.333333E+6	14.0	3.571429E+6	13	3.846154E+6	12	4.166667E+6
		15.5	3.225806E+6	14.5	3.448276E+6	13.5	3.703704E+6	12.5	4.0E+6
		16.0	3.125E+6	15.0	3.333333E+6	14	3.571429E+6	13	3.846154E+6

Table Assumptions

- ADC Trigger TIMERx PBCLK = 2/TAD =100Mhz
- ADC Clock (TAD) = 50Mhz

Note:

1. Based on current data sheet DS60001320E the ADC TAD spec = 20ns. However, consideration is currently being evaluated to increase the PIC32MZ ADC TAD clock electrical spec from 50Mhz to 63Mhz, (15.873ns), resulting in a significant increase in ADC throughput rates. Check current data sheet on the Microchip website and see respective table 3-5 or 3-7 accordingly for ADC throughput rates based on number of interleaved ADCs and ADC TAD electrical specification.

Table 3-6. PIC32MK Interleaved ADC Throughput Rates with TAD Electrical Specifications, ADC Clock = 16.667ns(min) (60Mhz)

Interleaved ADCs	TAD (min) 60Mhz	12 bit Mode		10 bit Mode		8 bit Mode		6 bit Mode	
		TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)
6 (1)	16.666667ns	3 (min)	20.0E+6	2.5 (min)	24.0E+6	2 (min)	30.0E+6	2 (min)	30.0E+6
		3.5	17.142857E+6	3	20.0E+6	2.5	24.0E+6	2.5	24.0E+6
		4	15.0E+6	3.5	17.142857E+6	3	20.0E+6	3	20.0E+6
		4.5	13.333333E+6	4	15.0E+6	3.5	17.142857E+6	3.5	17.142857E+6
		5	12.0E+6	4.5	13.333333E+6	4	15.0E+6	4	15.0E+6
5	16.666667ns	4 (min)	15.0E+6	3 (min)	20.0E+6	3 (min)	20.0E+6	2 (min)	30.0E+6
		4.5	13.333333E+6	3.5	17.142857E+6	3.5	17.142857E+6	2.5	24.0E+6
		5	12.0E+6	4	15.0E+6	4	15.0E+6	3	20.0E+6
		5.5	10.909091E+6	4.5	13.333333E+6	4.5	13.333333E+6	3.5	17.142857E+6
		6	10.0E+6	5	12.0E+6	5	12.0E+6	4	15.0E+6
4	16.666667ns	4 (min)	15.0E+6	3.5 (min)	17.142857E+6	3 (min)	20.0E+6	2.5 (min)	24.0E+6
		4.5	13.333333E+6	4	15.0E+6	3.5	17.142857E+6	3	20.0E+6
		5	12.0E+6	4.5	13.333333E+6	4	15.0E+6	3.5	17.142857E+6
		5.5	10.909091E+6	5	12.0E+6	4.5	13.333333E+6	4	15.0E+6
		6	10.0E+6	5.5	10.909091E+6	5	12.0E+6	4.5	13.333333E+6
3	16.666667ns	6 (min)	10.0E+6	5 (min)	12.0E+6	4 (min)	15.0E+6	4 (min)	15.0E+6
		6.5	9.230769E+6	5.5	10.909091E+6	4.5	13.333333E+6	4.5	13.333333E+6
		7	8.571428E+6	6	10.0E+6	5	12.0E+6	5	12.0E+6
		7.5	8.0E+6	6.5	9.230769E+6	5.5	10.909091E+6	5.5	10.909091E+6
		8	7.5E+6	7	8.571428E+6	6	10.0E+6	6	10.0E+6
2	16.666667ns	8 (min)	7.5E+6	7 (min)	8.571428E+6	6 (min)	10.0E+6	5 (min)	12.0E+6
		8.5	7.058823E+6	7.5	8.0E+6	6.5	9.230769E+6	5.5	10.909091E+6
		9	6.666667E+6	8	7.5E+6	7	8.571428E+6	6	10.0E+6
		9.5	6.315789E+6	8.5	7.058823E+6	7.5	8.0E+6	6.5	9.230769E+6
		10	6.0E+6	9	6.666667E+6	8	7.5E+6	7	8.571428E+6
		10.5	5.714286E+6	9.5	6.315789E+6	8.5	7.058823E+6	7.5	8.0E+6
		11	5.454545E+6	10	6.0E+6	9	6.666667E+6	8	7.5E+6
		11.5	5.217391E+6	10.5	5.714286E+6	9.5	6.315789E+6	8.5	7.058823E+6
		12	5.0E+6	11	5.454545E+6	10	6.0E+6	9	6.666667E+6
		12.5	4.8E+6	11.5	5.217391E+6	10.5	5.714286E+6	9.5	6.315789E+6
		13	4.615385E+6	12	5.0E+6	11	5.454545E+6	10	6.0E+6
		13.5	4.444444E+6	12.5	4.8E+6	11.5	5.217391E+6	10.5	5.714286E+6
		>14	1/(TAD Trigger Spacing * TAD)	>13	1/(TAD Trigger Spacing * TAD)	>12	1/(TAD Trigger Spacing * TAD)	>11	1/(TAD Trigger Spacing * TAD)

Table Assumptions:

- ADC Trigger TIMERx PBCLK = 2/TAD =120Mhz
- ADC Clock = 60Mhz

Table 3-7. Interleaved ADC Throughput Rates for devices with TAD Electrical Specifications ADC Clock Period = 15.873ns(min) (63Mhz)

Interleaved ADCs	TAD (min) 63Mhz	12 bit Mode		10 bit Mode		8 bit Mode		6 bit Mode	
		TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)	TAD Trigger Source Spacing	ADC Combined Throughput Rate (msps) = 1 / (TAD * TAD Trigger Spacing)
4	15.873ns(1)	4.0 (min)	15.75E+6	3.5 (min)	18.0E+6	3 (min)	21.0E+6	2.5 (min)	25.2E+6
		4.5	14.0E+6	4	15.75E+6	3.5	18.0E+6	3	21.0E+6
		5.0	12.6E+6	4.5	14.0E+6	4	15.75E+6	3.5	18.0E+6
		5.5	11.454545E+6	5	12.6E+6	4.5	14.0E+6	4	15.75E+6
		6.0	10.5E+6	5.5	11.454545E+6	5	12.6E+6	4.5	14.0E+6
3	15.873ns(1)	6.0 (min)	10.5E+6	5.0 (min)	12.6E+6	4 (min)	15.75E+6	4 (min)	15.75E+6
		6.5	9.692308E+6	5.5	11.454545E+6	4.5	14.0E+6	4.5	14.0E+6
		7.0	9.0E+6	6.0	10.5E+6	5	12.6E+6	5	12.6E+6
		7.5	8.4E+6	6.5	9.692308E+6	5.5	11.454545E+6	5.5	11.454545E+6
		8.0	7.875E+6	7.0	9.0E+6	6	10.5E+6	6	10.5E+6
2	15.873ns(1)	8.0 (min)	7.875E+6	7.0 (min)	9.0E+6	6 (min)	10.5E+6	5 (min)	12.6E+6
		8.5	7.411765E+6	7.5	8.4E+6	6.5	9.692308E+6	5.5	11.454545E+6
		9.0	7.0E+6	8.0	7.875E+6	7	9.0E+6	6	10.5E+6
		9.5	6.631579E+6	8.5	7.411765E+6	7.5	8.4E+6	6.5	9.692308E+6
		10.0	6.30E+6	9.0	7.0E+6	8	7.875E+6	7	9.0E+6
		10.5	6.0E+6	9.5	6.631579E+6	8.5	7.411765E+6	7.5	8.4E+6
		11.0	5.727273E+6	10.0	6.3E+6	9	7.0E+6	8	7.875E+6
		11.5	5.478261E+6	10.5	6.0E+6	9.5	6.631579E+6	8.5	7.411765E+6
		12.0	5.25E+6	11.0	5.727273E+6	10	6.3E+6	9	7.0E+6
		12.5	5.04E+6	11.5	5.478261E+6	10.5	6.0E+6	9.5	6.631579E+6
		13.0	4.846154E+6	12.0	5.25E+6	11	5.727273E+6	10	6.3E+6
		13.5	4.666667E+6	12.5	5.04E+6	11.5	5.478261E+6	10.5	6.0E+6
		14.0	4.5E+6	13.0	4.846154E+6	12	5.25E+6	11	5.727273E+6
		14.5	4.344828E+6	13.5	4.666667E+6	12.5	5.04E+6	11.5	5.478261E+6
		15.0	4.2E+6	14.0	4.5E+6	13	4.846154E+6	12	5.25E+6
		15.5	4.064516E+6	14.5	4.344828E+6	13.5	4.666667E+6	12.5	5.04E+6
		16.0	3.9375E+6	15.0	4.2E+6	14	4.5E+6	13	4.846154E+6

Table Assumptions

- ADC Trigger TIMERx PBCLK = 2/TAD =126Mhz
- ADC Clock (TAD) = 63Mhz

Note:

1. Based on current data sheet DS60001320E the ADC TAD spec = 20ns. However, consideration is currently being evaluated to increase the PIC32MZ ADC TAD clock electrical spec from 50Mhz to 63Mhz, (15.873ns), resulting in a significant increase in ADC throughput rates. Check current data sheet on the Microchip website and see respective table 3-5 or 3-7 accordingly for ADC throughput rates based on number of interleaved ADCs and ADC TAD electrical specification.

4. PIC32MKxxxx 20msps Six ADC Interleaved Code Example

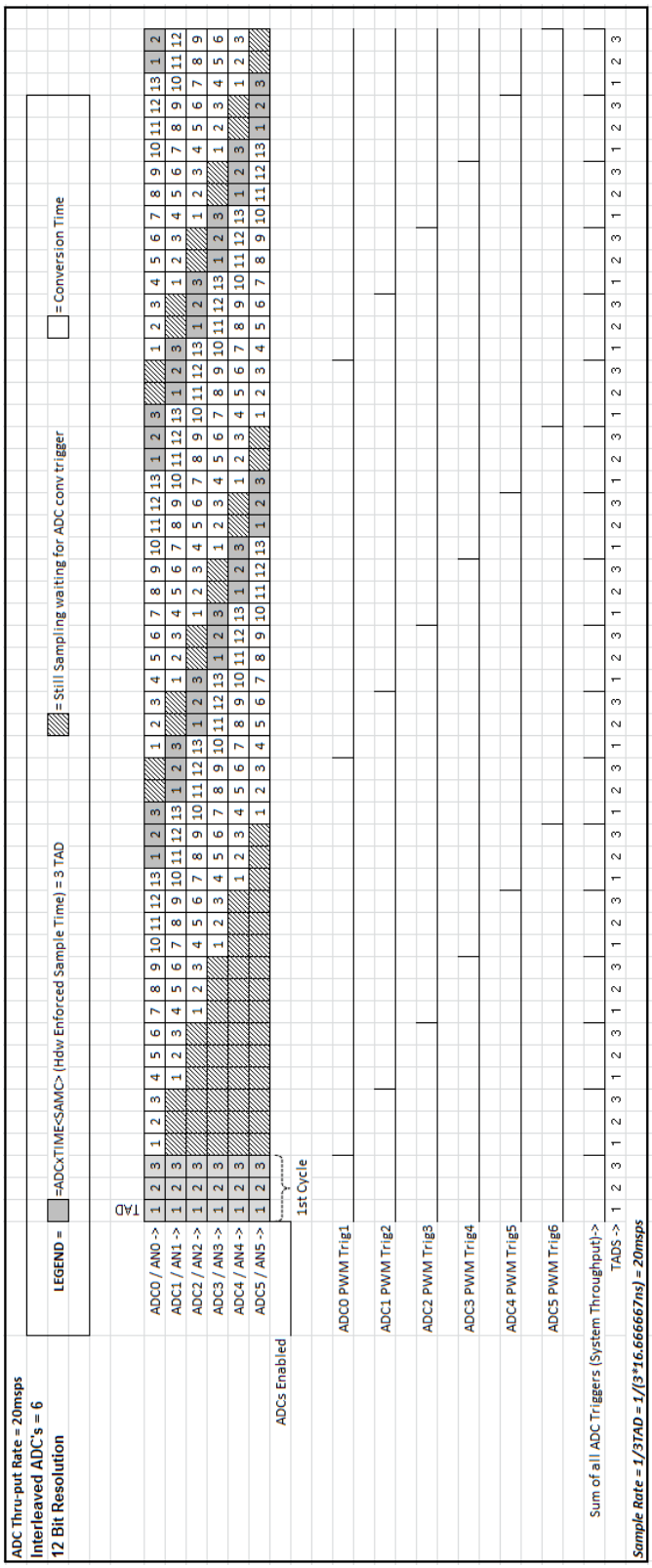


Figure 4-1. PIC32MKxxxx 20msps 6 ADC Interleaved Timing Diagram

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
// ************************************************************************************************
// PIC32MKxxMCFxx 20msps Interleaved ADC sample code example
//
// SUMMARY:
//    TARGET CPU: PIC32MKxxMCFxx
//    OSC TYPE = 24 MHz Clock oscillator (EC Mode)
//    SYSCLK=120 MHz
//    PBCLK = 120 MHz
//    ADC TAD = SYSCLK / 2 = 60 MHz = 16.666667ns
//    ADC SAMC=3 TAD
//    ADC SAMPLE RATE = 20msps
//    INTERLEAVED ADCS
//      ADC 0: 12bit mode, ADCTRG1<TRGSRC0> PWM Generator 1 trigger PTMR time base
//      ADC 1: 12bit mode, ADCTRG1<TRGSRC1> PWM Generator 2 trigger PTMR time base
//      ADC 2: 12bit mode, ADCTRG1<TRGSRC2> PWM Generator 3 trigger PTMR time base
//      ADC 3: 12bit mode, ADCTRG1<TRGSRC3> PWM Generator 4 trigger PTMR time base
//      ADC 4: 12bit mode, ADCTRG2<TRGSRC4> PWM Generator 5 trigger PTMR time base
//      ADC 5: 12bit mode, ADCTRG2<TRGSRC5> PWM Generator 6 trigger PTMR time base
//
//    interleaved ADC0/1/2/3/4/5 with only AN0/AN2/AN5 inputs, (20 msps combined throughput rate)
//
//   NOTE:
//     Motor Control PWM peripherals and hence PWM triggers only exists on
//     PIC32MKxxMCxx variants. If PWM used as ADC trigger sources then PWM motor control
//     functionality would be excluded. The two independent uses are exclusive of each other.
// ************************************************************************************************
//   NOTE: (FYI only)
//     A user could configure the PIC32MK to have multiple sets of
//     interleaved ADC's to measure multiple different analog input streams like:
//          ----------------------------  EXAMPLE  1  CONFIGURATION  ----------------------------
//       1) ADC0-ADC1 (2) ADCs with triggers OC1 and TMR5 @ 7.5msps(max) combined
//       2) ADC2-ADC3 (2) ADCs with triggers OC4 and TMR3 @ 7.5msps(max) combined
//       3) ADC4-ADC5 (2) ADCs PWM Generator 1 & PWM Generator 2 trigger PTMR time base  @ 7.5msps(max)
//          ----------------------------  EXAMPLE  2  CONFIGURATION  ----------------------------
//       4) ADC0-ADC2 (3) ADCs with triggers OC1, OC2 and TMR3 @ 10msps(max) combined
//       5) ADC3-ADC5 (3) ADCs with PWM Generator 1 / 2 / 3 trigger PTMR time base  @ 10msps(max) combined
//          ----------------------------  EXAMPLE  3  CONFIGURATION  ----------------------------
//       6) ADC0-ADC1 (2) ADCs with triggers OC1 and TMR3 @ 7.5msps(max) combined
//       7) ADC2-ADC5 (4) ADC's with PWM Generator 1 / 2 / 3 / 4  trigger PTMR time base  @ 15msps(max) combined
//          ----------------------------  EXAMPLE  4  CONFIGURATION  ----------------------------
//       8) ADC0-ADC5 (6) ADC's with PWM Generator 1/2/3/4/5/6 trigger PTMR time base  @ 20msps(max) combined
// ************************************************************************************************
//USER NOTE:
// Although the number of interleaved ADC's the user wishes to use is selectable, it is required
// "FOR THIS CODE ONLY" that those ADCx modules be sequential starting from AN0 to the last sequential
// ANx to be used for interleaving. In different code than below, any combination of interleaved ADC are allowed
//
// ************************************************************************
//       F I L E   I N C L U D E S
// ************************************************************************
#include <xc.h>
#include <sys/attribs.h>

// ************************************************************************
//       C O N F I G U R A T I O N   W O R D S
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
// ******************************************************************************
//
//------------------------------------------------------------
//                  DEVCFG0
//------------------------------------------------------------
#pragma config DEBUG =      OFF
#pragma config JTAGEN =     OFF
#pragma config ICESEL =     ICS_PGx2
#pragma config TRCEN =      OFF
#pragma config BOOTISA =    MIPS32
#pragma config FSLEEP =     OFF
#pragma config DBGPER =     PG_ALL
#pragma config SMCLR =      MCLR_NORM
#pragma config SOSCGAIN =   GAIN_2X
#pragma config SOSCBOOST =  ON
#pragma config POSCGAIN =   GAIN_LEVEL_3
#pragma config POSCBOOST =  ON
#pragma config EJTAGBEN =   NORMAL
#pragma config CP =         OFF

//------------------------------------------------------------
//                  DEVCFG1
//------------------------------------------------------------
#pragma config FNOSC =      SPLL
#pragma config DMTINTV =    WIN_127_128
#pragma config FSOSCEN =    OFF
#pragma config IESO =   OFF
#pragma config POSCMOD =    EC    //External 24Mhz Clock oscillator
#pragma config OSCIOFNC =   ON
#pragma config FCKSM =      CSECME
#pragma config WDTPS =      PS1048576
#pragma config WDTSPGM =    STOP
#pragma config FWDTEN =     OFF
#pragma config WINDIS =     NORMAL
#pragma config FWDTWINSZ = WINSZ_25
#pragma config DMTCNT =     DMT31
#pragma config FDMTEN =     OFF

//------------------------------------------------------------
//                  DEVCFG2
//------------------------------------------------------------
#pragma config FPLLIDIV =   DIV_3
#pragma config FPLLRNG =    RANGE_5_10_MHZ
#pragma config FPLLICLK =   PLL_POSC
#pragma config FPLLMULT =   MUL_60
#pragma config FPLLODIV =   DIV_4
#pragma config VBATBOREN = OFF
#pragma config DSBOREN =    ON
#pragma config DSWDTPS =    DSPS32
#pragma config DSWDTOSC =   LPRC
#pragma config DSWDTEN =    OFF
#pragma config FDSEN =      ON
#pragma config BORSEL =     HIGH
#pragma config UPLLEN =     OFF
//------------------------------------------------------------
//                  DEVCFG3
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
//------------------------------------------------------------
#pragma config USERID =     0x0ADC
#pragma config FUSBIDIO2 = OFF
#pragma config FVBUSIO2 = OFF
#pragma config PGL1WAY = ON
#pragma config PMDL1WAY = ON
#pragma config IOL1WAY = ON
#pragma config FUSBIDIO1 = OFF
#pragma config FVBUSIO1 = OFF
#pragma config PWMLOCK = OFF
//------------------------------------------------------------
//                  BF1SEQ0
//------------------------------------------------------------
#pragma config TSEQ =       0x0000
#pragma config CSEQ =       0xffff

// *****************************************************************************************************
//                                  U S E R     D E F I N E S
// Note: User must configure as required
// *****************************************************************************************************
#define TAD_TRIGGER_SOURCE_SPACING   3    // TAD trigger spacing value from ADC Table 3-6 w/12bit & 6 interleaved ADCs
#define INTERLEAVED_ADC_COUNT              6     // Number of interleaved ADC (Cannot be <2 or >6). Changing this value will
                                                 // automatically scale the number of interleaved ADC and triggers
utilized in this code.

// *****************************************************************************************************
// EQUATION #1: (Assumptions: INTERLEAVED_ADC_COUNT=6)
//    TAD_TRIGGER_SOURCE_SPACING(min)  =
//        = ((SAMC + (((#bits Resolution+1) + ADJ) * TAD)) / INTERLEAVED_ADC_COUNT)
//        = (3+12+1) + ADJ) * TAD)) / 6 ADCs)
//        = (16 + ADJ) * TAD)) / 6 ADCs
//        = (16 + 2) * TAD)) / 6 ADCs
//        = 3.0 TAD (minimum trigger spacing possible)
//
// NOTE: "ADJ" term = A user selected whole number adjustment factor between 0-4
//                    that must yield a Minimum TAD Trigger interval that equates to an exact
//                    multiple of a ½ TAD. This represents the minimum TAD_TRIGGER_SOURCE_SPACING
//                    and therefore the fastest throughput possible. Additional TAD_TRIGGER_SOURCE_SPACING
//                    and hence ADC Throughput Rates are possible in 1/2 TAD trigger spacing increments.
//                    (See examples 1-3 below)
//
//  NOTE: Once the minimum, TAD trigger spacing possible is determined, the user can configure for any
//        additional sampling frequency required if needed in 0.5 TAD trigger spacing increments.
//
//        Example 1: (INTERLEAVED_ADC_COUNT=6)
//          TAD_TRIGGER_SOURCE_SPACING(min) = 3
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 60Mhz / 3.0(min)
//                                          = 20msps
//
//        Example 2: (INTERLEAVED_ADC_COUNT=6)
//          TAD_TRIGGER_SOURCE_SPACING = 3.5
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 60Mhz / 3.5
//                                          = 17.142857msps
//
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
//        Example 3: (INTERLEAVED_ADC_COUNT=6)
//          TAD_TRIGGER_SOURCE_SPACING = 4.0
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 60Mhz / 4.0
//                                          = 15msps
//
// ***********************************************************************************************

// ***********************************************************************************************
//                              P R O G R A M    D E F I N E S
//                                     (User Do Not Change)
// Note:
//       TAD_TRIGGER_SOURCE_SPACING is multiplied by (2) because
//        PWM trigger source time base is 2x faster than an ADC TAD clock.
//        2 PWM clocks = 1 ADC TAD Clock
// ***********************************************************************************************
// -----------------------------------------------------------------------------------------------------------------
// NOTE: Remember that the peripheral trigger clock frequency is 2x TAD clock frequency.
//             This is why the 2x. The -1 is because the Si peripheral triggers are always
//             on (match+1), so to compensate for correct timing subtract 1.
//
#define ADC_TRIG        (TAD_TRIGGER_SOURCE_SPACING * 2)        //6
#define PWM1_ADC_TRIG   (ADC_TRIG - 1)           // 5, ADC0 PWM1 Trig (Note: ADC Trig on match +1)
#define PWM2_ADC_TRIG   ((2 * ADC_TRIG) - 1)   //11, ADC1 PWM1 Trig (Note: ADC Trig on match +1)
#define PWM3_ADC_TRIG   ((3 * ADC_TRIG) - 1)   //17, ADC2 PWM2 Trig (Note: ADC Trig on match +1)
#define PWM4_ADC_TRIG   ((4 * ADC_TRIG) - 1)   //23, ADC3 PWM3 Trig (Note: ADC Trig on match +1)
#define PWM5_ADC_TRIG   ((5 * ADC_TRIG) - 1)   //29, ADC4 PWM4 Trig (Note: ADC Trig on match +1)
#define PWM6_ADC_TRIG   ((6 * ADC_TRIG) - 1)   //35, ADC5 PWM5 Trig (Note: ADC Trig on match +1)

#define ADC_CHANNEL_BUFFER_LENGTH   (uint16_t) 128         // Allowed = 2n n=0-7: (128 Samples per ADC per Buffer. Each ADC has 2
buffers)
#define HALF_BUFFER_LENGTH          (uint16_t) (ADC_CHANNEL_BUFFER_LENGTH*INTERLEAVED_ADC_COUNT)
#define TOTAL_BUFFER_LENGTH         (uint16_t) (2*ADC_CHANNEL_BUFFER_LENGTH*INTERLEAVED_ADC_COUNT)
#define ADC_SAMPLE_OFFSET           (uint16_t) (ADC_CHANNEL_BUFFER_LENGTH*2) // Sample Offset between two sequential samples in
RAW ADC DMA Buffer
#define ADC_INTER_BUFFER_OFFSET     ADC_CHANNEL_BUFFER_LENGTH // Sample offset between Buffer A and Buffer B
#define DMA_BUFA_FULL  (0x00000001 << (INTERLEAVED_ADC_COUNT-1))
#define DMA_BUFB_FULL  (0x00010000 << (INTERLEAVED_ADC_COUNT-1))
#define DMA_BUFA_FULL_INT  (0x00000100 << (INTERLEAVED_ADC_COUNT-1))
#define DMA_BUFB_FULL_INT  (0x01000000 << (INTERLEAVED_ADC_COUNT-1))
#define DMA_INT_SRC  (DMA_BUFA_FULL_INT  |  DMA_BUFB_FULL_INT)
#define BUFFER_RPT_COUNT            1 //

#if ((INTERLEAVED_ADC_COUNT  <  2) | (INTERLEAVED_ADC_COUNT  >  6))
#error  INTERLEAVED_ADC_COUNT error, it is either less than 2 or greater than 6
#endif

// ***********************************************************************************************
//        F U N C T I O N     P R O T O T Y P E S
// ***********************************************************************************************
void init_PWM(void);
void init_ADC(void);

// ***********************************************************************************************
//        G L O B A L   V A R I A B L E S
// ***********************************************************************************************
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
unsigned int ADC_var=0;
char buffer_select = 0;
uint8_t buffer_a_index, buffer_b_index, buffer_index;

// *************************************************************************************************
// NOTE:
// Each ADC has 2 buffers. Each buffer size can be up to 128 samples (256bytes). Thus,
// total max buffer size per ADC is 256 samples (512 bytes). Since we are using 6 ADC
// channels in this example, the total DMA BUffer size =
// 128(samples per buffer)*2(buffer per adc)*6(number of adchannels) * BUFFER_RPT_COUNT
// = 1536 samples or 3072 bytes.
// *************************************************************************************************
uint16_t adc_dma_raw_buffer[TOTAL_BUFFER_LENGTH]; //Buffer where ADC Data is dumped by the DMA
uint16_t adc_data_rearranged_buffer[BUFFER_RPT_COUNT][TOTAL_BUFFER_LENGTH]; // Buffer where ADC samples are re-arranged in
sequentially
uint16_t source_index, destination_index, adc_count, index_offset, sample_count;

unsigned short  *adc0_ptr,*adc1_ptr,*adc2_ptr,*adc3_ptr,*adc4_ptr,*adc5_ptr;
unsigned short *dest_ptr, *end_ptr;


// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $                 M A I N       R O U T I N E                        $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
int main(void)
{
    // *************************************************************************************************
    // CPU Performance Optimization:
    // *************************************************************************************************
    register unsigned long tmp_cache;                 //KSEG0 cache enable
    asm("mfc0 %0,$16,0" :  "=r"(tmp_cache));
    tmp_cache = (tmp_cache & ~7) | 3;
    asm("mtc0 %0,$16,0" :: "r" (tmp_cache));

    CHECONbits.PFMWS=3;       //Flash wait states = 3 CPU clock cycles @ 120Mhz
    CHECONbits.PREFEN = 1;    //PREFEN
    PRISSbits.PRI7SS = 1;    //DMA Interrupt with priority level of 7 uses Shadow Set 1
    INTCONbits.MVEC = 1;    //Enable multi-vector interrupts
    __builtin_mtc0(12,0,(__builtin_mfc0(12,0) | 0x0001)); // Global Interrupt Enable

    SYSKEY = 0xAA996655;       // Write Key1 to SYSKEY
    SYSKEY = 0x556699AA;       //Unlock
    PB1DIVbits.PBDIV = 0;      //0 = 1:1 = SYSCLK/1 = 120Mhz
    PB2DIVbits.PBDIV = 0;      //0 = 1:1 = SYSCLK/1 = 120Mhz
    PB3DIVbits.PBDIV = 0;      //0 = 1:1 = SYSCLK/1 = 120Mhz
    PB4DIVbits.PBDIV = 0;      //0 = 1:1 = SYSCLK/1 = 120Mhz
    PB5DIVbits.PBDIV = 0;      //0 = 1:1 = SYSCLK/1 = 120Mhz
    PB6DIVbits.PBDIV = 3;      //0 = 4:1 = SYSCLK/4 = 30Mhz
    SYSKEY = 0x00000000;        //Lock

    // *************************************************************************************************
    // Initialize and enable ADC + ADC DMA then ADC PWM trigger sources
    // *************************************************************************************************
    init_ADC();                //Initialize and enable interleaved ADCs first before PWM ADC triggers are enabled

    while(1)
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    {
          // USERS CODE GOES HERE
    }
}

// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $                        F U N T I O N    S U B _ R O U T I N U E S                                  $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// ***********************************************************************************************
// Interleaved ADC PWM Trigger initialization:
//   - PWM output pins disabled
//   - PWMs 1-x = 50% duty cycle
//   - PWM Period Timer = 6x ADC TAD trigger spacing
//   - PWM ADC conversion triggers staggered in 1/6 increments
//  NOTE:
//   1): In this application, PWM functionality is exclusive to only the interleaved ADC trigger(s)
//   2): PWM when used as ADC trigger(s) must never be enabled before ADC initialization 100% complete
// ***********************************************************************************************
void init_PWM(void)
{
    PTPER = (INTERLEAVED_ADC_COUNT * ADC_TRIG);

    PDC1 = (PTPER >> 1);    //50% of period
    TRIG1 = PWM1_ADC_TRIG;    //PWM1 Trigger point

    PDC2 = PDC1;
    TRIG2 = PWM2_ADC_TRIG;    //PWM2 Trigger point

    if(INTERLEAVED_ADC_COUNT   >  2)
    {
        PDC3 = PDC1;
        TRIG3 = PWM3_ADC_TRIG;    //PWM3 Trigger point
    }

    if(INTERLEAVED_ADC_COUNT   >  3)
    {
        PDC4 = PDC1;
        TRIG4 = PWM4_ADC_TRIG;    //PWM4 Trigger point
    }

    if(INTERLEAVED_ADC_COUNT   >  4)
    {
        PDC5 = PDC1;
        TRIG5 = PWM5_ADC_TRIG;    //PWM5 Trigger point
    }

    if(INTERLEAVED_ADC_COUNT   >  5)
    {
        PDC6 = PDC1;
        TRIG6 = PWM6_ADC_TRIG;    //PWM6 Trigger point
    }

    IOCON1 = 0x00030000;    //Fault disabled, GPIO controls PWM1H/L pins
    IOCON2 = 0x00030000;    //Fault disabled, GPIO controls PWM2H/L pins
    IOCON3 = 0x00030000;    //Fault disabled, GPIO controls PWM3H/L pins
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    IOCON4 = 0x00030000;     //Fault disabled, GPIO controls PWM4H/L pins
    IOCON5 = 0x00030000;       //Fault disabled, GPIO controls PWM5H/L pins
    IOCON6 = 0x00030000;       //Fault disabled, GPIO controls PWM6H/L pins

    PTCONbits.PTEN = 1;
}

// ***************************************************************************************************
// init_ADC
// Interleaved ADC Initialization Function:
//    Used analog inputs if:
//       INTERLEAVED_ADC_COUNT  = 2   (Analog inputs, AN0 only)
//       INTERLEAVED_ADC_COUNT  = 3   (Analog inputs, AN0 & AN2 only)
//       INTERLEAVED_ADC_COUNT  = 4   (Analog inputs, AN0 & AN2 only)
//       INTERLEAVED_ADC_COUNT  = 5   (Analog inputs, AN0 & AN2 only)
//       INTERLEAVED_ADC_COUNT  = 6   (Analog inputs, AN0 & AN2 & AN5 only)
// ***************************************************************************************************
// ***************************************************************************
//  These steps are already done in this code example, this is just a reminder
//   to the user for any new original code development.
//
// CAUTION:(Required)
// 1) The user SHOULD always set the minimum sampling time of each Class_1 ADC
//      equal to 3TAD as defined by ADCxTIME<SAMC>.
// 2) The user should insure that the selected trigger source time base clock rate is
//      exactly 2x faster than the ADC clock TAD rate. This will provide the highest
//      timing resolution along with the most sampling rate options as defined in
//      Table 3-6 of this app note.
// 3) The user selected trigger sources have to be able to be mapped to a single
//      synchronized time base.
//      For example:
//        a) TMR3 & OCx (x=1,3,5 PIC32MZxxxx)
//        b) TMR3 & OCx (x=1-4 PIC32MKxxGPxx)
//        c) PTMRx & TRIGx ( x=1-12 PIC32MKxxMCxx PWM)
//        d) PTMRx & STRIGx ( x=1-12 PIC32MKxxMCxx PWM)
// 4) In a same group of interleaved ADC's a user "MUST NOT" use two separate timer trigger sources.
//     Example: ADC0, ADC1 and ADC2 are being used in an interleaved group.
//       Trigger sources can be any combination of valid ADC OCx edge triggered sources synchronized to a SINGLE TMRx.
//       It would not be allowed to use in the same group of interleaved ADCs OCx, TMR3 & TMR5 for example.
//       Separate TMRx in different interleaved ADC groups is OK.
// 5  If using the DMA for the ADC the user "MUST" use SYSCLK as the ADC
//     source clock, (i.e. ADCCON3<ADCSEL>).
// 6) User must configure ADC for 512 TAD warm-up time
//     (ADCANCONbits.WKUPCLKCNT = 0x9) "BEFORE" enabling ADC.
// 7) Users SW  must wait for ADC BANGAP and ADC warm-up time "AFTER"
//      enabling ADC and before activating the ADC trigger sources.
// 8) ADC must always be initialized 1st "BEFORE" activating the ADC trigger sources
//
// NOTE: Failure to follow these recommendations will result in inaccurate ADC
//             data acquisition and poor performance.
//***************************************************************************
void init_ADC(void)
{
    ADC0CFG = DEVADC0;        //Load ADC0 Calibration values
    ADC1CFG = DEVADC1;        //Load ADC1 Calibration values
    ADC2CFG = DEVADC2;        //Load ADC2 Calibration values
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    ADC3CFG = DEVADC3;        //Load ADC3 Calibration values
    ADC4CFG = DEVADC4;        //Load ADC4 Calibration values
    ADC5CFG = DEVADC5;        //Load ADC5 Calibration values
    ADC7CFG = DEVADC7;        //Load ADC7 Calibration values

    ADCANCONbits.WKUPCLKCNT = 0x9; // ADC Warm up delay = (512 * TADx)

    //***************************************************************************
    // If using the DMA the user MUST use SYSCLK as the ADC source clock, (ADCCON3<ADCSEL> = 3)
    //***************************************************************************
    ADCCON3bits.ADCSEL = 3;        // Select ADC input clock source = SYSCLK 120Mhz
    ADCCON3bits.CONCLKDIV = 0;     // Analog-to-Digital Control Clock (TQ) Divider = SYSCLK Divide by 1
    ADCCON1bits.FSSCLKEN = 1;       //Fast synchronous SYSCLK to ADC control clock is enabled

    //***************************************************************************
    // ADC0 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM1 Trigger
    //***************************************************************************
    ADC0TIMEbits.ADCDIV = 1;              // ADCx Clock Divisor, Divide by 2, 60Mhz
    ADC0TIMEbits.SAMC = 1;                 // 3 TAD (Hardware enforced sample time)
    ADC0TIMEbits.SELRES = 3;              //ADC0 12bit resolution mode
    ADC0TIMEbits.BCHEN=1;                 // ADC data saved in DMA system ram buffer
    ADCTRG1bits.TRGSRC0 = 10;           // Set AN0 to trigger from PWM Trigger 1
    ADCANCONbits.ANEN0 = 1;              // Enable ADC0 analog bias/logic
    ADCCON3bits.DIGEN0 = 1;               // Enable ADC0 digital logic

    //***************************************************************************
    // ADC1 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM2 Trigger
    //***************************************************************************
    ADC1TIMEbits.ADCDIV = 1;
    ADC1TIMEbits.SAMC = 1;                   // 3 TAD
    ADCTRGMODEbits.SH1ALT = 3;       // AN0 is the input to ADC1
    ADC1TIMEbits.SELRES = 3;              //ADC1 12bit resolution mode
    ADC1TIMEbits.BCHEN=1;                  // ADC data saved in DMA system ram buffer
    ADCTRG1bits.TRGSRC1 = 11;           // Set AN1 to trigger from PWM Trigger 2
    ADCANCONbits.ANEN1 = 1;              // Enable ADC1 analog bias/logic
    ADCCON3bits.DIGEN1 = 1;               // Enable ADC1 digital logic

    //***************************************************************************
    // ADC2 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM3 Trigger
    //***************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  2)
    {
        ADC2TIMEbits.ADCDIV = 1;
        ADC2TIMEbits.SAMC = 1;                // 3 TAD
        ADCTRGMODEbits.SH2ALT = 0;    // AN2 is the input to ADC2
        ADC2TIMEbits.SELRES = 3;            //ADC2 12bit resolution mode
        ADC2TIMEbits.BCHEN=1;               // ADC data saved in DMA system ram buffer
        ADCTRG1bits.TRGSRC2 = 12;       // Set AN2 to trigger from PWM Trigger 3
        ADCANCONbits.ANEN2 = 1;          // Enable ADC2 analog bias/logic
        ADCCON3bits.DIGEN2 = 1;           // Enable ADC2 digital logic
    }

    //***************************************************************************
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    // ADC3 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM4 Trigger
    //*****************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  3)
    {
        ADC3TIMEbits.ADCDIV = 1;
        ADC3TIMEbits.SAMC = 1;                          // 3 TAD
        ADCTRGMODEbits.SH3ALT = 1;   // AN0 is the input to ADC3
        ADC3TIMEbits.SELRES = 3;          //ADC3 12bit resolution mode
        ADC3TIMEbits.BCHEN=1;             // ADC data saved in DMA system ram buffer
        ADCTRG1bits.TRGSRC3 = 13;       // Set AN3 to trigger from PWM Trigger 4
        ADCANCONbits.ANEN3 = 1;          // Enable ADC2 analog bias/logic
        ADCCON3bits.DIGEN3 = 1;           // Enable ADC2 digital logic
    }

    //*****************************************************************************
    // ADC4 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM5 Trigger
    //*****************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  4)
    {
        ADC4TIMEbits.ADCDIV = 1;
        ADC4TIMEbits.SAMC = 1;                          // 3 TAD
        ADCTRGMODEbits.SH4ALT = 3;   // AN0 is the input to ADC4
        ADC4TIMEbits.SELRES = 3;          //ADC4 12bit resolution mode
        ADC4TIMEbits.BCHEN=1;             // ADC data saved in DMA system ram buffer
        ADCTRG2bits.TRGSRC4 = 14;       // Set AN4 to trigger from PWM Trigger 5
        ADCANCONbits.ANEN4 = 1;          // Enable ADC2 analog bias/logic
        ADCCON3bits.DIGEN4 = 1;           // Enable ADC2 digital logic
    }

    //*****************************************************************************
    // ADC5 Module setup:
    //    TAD = SYSCLK/2, 12bit, DMA enb, SAMC=3TAD, PWM6 Trigger
    //*****************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  5)
    {
        ADC5TIMEbits.ADCDIV = 1;
        ADC5TIMEbits.SAMC = 1;              // 3 TAD
        ADCTRGMODEbits.SH5ALT = 0;   // AN5 is the input to ADC5
        ADC5TIMEbits.SELRES = 3;          //ADC5 12bit resolution mode
        ADC5TIMEbits.BCHEN=1;             // ADC data saved in DMA system ram buffer
        ADCTRG2bits.TRGSRC5 = 15;     // Set AN5 to trigger from PWM Trigger 6
        ADCANCONbits.ANEN5 = 1;         // Enable ADC2 analog bias/logic
        ADCCON3bits.DIGEN5 = 1;           // Enable ADC2 digital logic
    }

    //*****************************************************************************
    // ADC DMA Master Configuration (PIC32MKxxxx Only)
    // 12 buffers @128 samples/buffer = (6 ping pong buffer A?s, 6 ping pong buffer B?s)
    //*****************************************************************************
    ADCCON1bits.DMABL = 7; // ADC Buffer length = 128 samples

    ADCDMAB = (unsigned int) &adc_dma_raw_buffer & 0x1FFFFFFF;
    ADCDSTATbits.DMAEN = 1;
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    ADCDSTAT = (ADCDSTAT | DMA_INT_SRC);    // Enb DMA Buff Full A&B Interrupt
    IFS3bits.AD1FCBTIF = 0;         // Clear DMA Buffer full interrupt flag
    IPC26bits.AD1FCBTIP = 1;        // Set DMA Buffer full interrupt priority
    IPC26bits.AD1FCBTIS = 1;         // Set DMA Buffer full interrupt sub-priority
    IEC3bits.AD1FCBTIE = 1;             // Enable DMA Buffer full interrupt

    ADCCON1bits.ON = 1;                // Turn the ADC on

    //*************************************************************************
    // Waiting for ADC warm-up time and ADC bandgap reference to stabilize
    //*************************************************************************
    while(!ADCCON2bits.BGVRRDY);    // Wait until the reference voltage is ready
    while(!ADCANCONbits.WKRDY0);     // Wait until ADC0 is ready
    while(!ADCANCONbits.WKRDY1);     // Wait until ADC1 is ready
    if(INTERLEAVED_ADC_COUNT  >  2) while(!ADCANCONbits.WKRDY2); // Wait until ADC2 is ready
    if(INTERLEAVED_ADC_COUNT  >  3) while(!ADCANCONbits.WKRDY3); // Wait until ADC3 is ready
    if(INTERLEAVED_ADC_COUNT  >  4) while(!ADCANCONbits.WKRDY4); // Wait until ADC4 is ready
    if(INTERLEAVED_ADC_COUNT  >  5) while(!ADCANCONbits.WKRDY5); // Wait until ADC5 is ready

    init_PWM();             //Initialize internal PWM interleaved triggers and
                                //configure PWM pins for general purpose I/O or alternate PPS
}

// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $     I N T E R R U P T   S E R V I C E    R O U T I N E S        $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// ***********************************************************************************************
//        D M A   I S R (Interrupt Service Routine)
// ***********************************************************************************************
void __attribute__((interrupt(ipl7auto), at_vector(_ADC_DMA_VECTOR), aligned(16))) isr ()
{
    ADC_var = ADCDSTAT;               // To preserve ADCSTAT reg after read

    //*************************************************************************
    // DMA Ping-Pong Buffer A full processing
    // Each ADC samples (1 / INTERLEAVED_ADC_COUNT) of the analog input signal. For
    // CPU processing of the interleaved ADC data it must be read from each of the
    // active interleaved ADC's dedicated ping-pong buffer A and rearranged in
    // sequential order in a CPU ADC result system SRAM buffer, for signal processing
    //*************************************************************************
    if(ADC_var & DMA_BUFA_FULL)       //Last sequential ADC RAM DMA ping-pong Buffer A full.
    {
        adc0_ptr = (unsigned short*)&adc_dma_raw_buffer; // Initialize ADC0 Source pointer to ADC0_BUFFER_A location in SRAM
        adc1_ptr = adc0_ptr + ADC_SAMPLE_OFFSET;    // Initialize ADC1 Source pointer to ADC1_BUFFER_A location in SRAM

        #if (INTERLEAVED_ADC_COUNT  >  2)
            adc2_ptr = adc1_ptr +ADC_SAMPLE_OFFSET;    // Init ADC2 Src ptr to ADC2_BUFFER_A
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  3)
            adc3_ptr = adc2_ptr +ADC_SAMPLE_OFFSET;    // Init ADC3 Src ptr to ADC3_BUFFER_A
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  4)
            adc4_ptr = adc3_ptr +ADC_SAMPLE_OFFSET;    // Init ADC4 Src ptr to ADC4_BUFFER_A
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  5)
            adc5_ptr = adc4_ptr +ADC_SAMPLE_OFFSET;    // Init ADC5 Src ptr to ADC5_BUFFER_A
        #endif

       //***************************************************************************
       // Initialize DMA destination pointer to base address where re-arranged
       // ADC data from Buffer A is stored in SRAM
       //***************************************************************************
       dest_ptr = (unsigned short*)&adc_data_rearranged_buffer[buffer_index][0];

       //***************************************************************************
       // Initialize DMA End address value for re-arranged ADC Data from Buffer A.
       //***************************************************************************
       end_ptr  = (unsigned short*)&adc_data_rearranged_buffer[buffer_index][0]+ HALF_BUFFER_LENGTH;

       do
        {
             *dest_ptr++ = *adc0_ptr++;
             *dest_ptr++ = *adc1_ptr++;

             #if (INTERLEAVED_ADC_COUNT  >  2)
                 *dest_ptr++ = *adc2_ptr++;
             #endif

             #if (INTERLEAVED_ADC_COUNT  >  3)
                 *dest_ptr++ = *adc3_ptr++;
             #endif

             #if (INTERLEAVED_ADC_COUNT  >  4)
                 *dest_ptr++ = *adc4_ptr++;
             #endif

             #if (INTERLEAVED_ADC_COUNT  >  5)
                 *dest_ptr++ = *adc5_ptr++;
             #endif
        }
        while(dest_ptr < end_ptr); // Continue Re-arranging ADC data until all Buffer A data for active ADCs is complete

        buffer_a_index++;     // Increment buffer_a_index by 1 at the end of re-arrangement of Buffer A data.
    }

    //***************************************************************************
    // DMA Ping-Pong Buffer "B" full processing
    // Each ADC samples (1/INTERLEAVED_ADC_COUNT)of the analog input signal. For
    // CPU processing of the interleaved ADC data it must be read from each of the
    // active interleaved ADC's dedicated ping-pong buffer "B" and rearranged in
    // sequential order in the CPU ADC result system SRAM buffer.
    //***************************************************************************
    if(ADC_var & DMA_BUFB_FULL)       //Last sequential ADC RAM DMA ping-pong Buffer B full
    {
        adc0_ptr = (unsigned short*)&adc_dma_raw_buffer + ADC_INTER_BUFFER_OFFSET; // Init ADC0 Src ptr to ADC0_BUFFER_B in SRAM
        adc1_ptr = adc0_ptr +ADC_SAMPLE_OFFSET;                    // Init ADC1 Src ptr to ADC1_BUFFER_B in SRAM

        #if (INTERLEAVED_ADC_COUNT  >  2)
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
            adc2_ptr = adc1_ptr +ADC_SAMPLE_OFFSET;    // Init ADC2 Src ptr to ADC2_BUFFER_B
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  3)
            adc3_ptr = adc2_ptr +ADC_SAMPLE_OFFSET;    // Init ADC3 Src ptr to ADC3_BUFFER_B
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  4)
            adc4_ptr = adc3_ptr +ADC_SAMPLE_OFFSET;    // Init ADC4 Src ptr to ADC4_BUFFER_B
        #endif

        #if (INTERLEAVED_ADC_COUNT  >  5)
            adc5_ptr = adc4_ptr +ADC_SAMPLE_OFFSET;    // Init ADC5 Src ptr to ADC5_BUFFER_B
        #endif

       //*****************************************************************************
       // Initialize DMA destination pointer to base address where re-arranged
       // ADC data from Buffer B is stored in SRAM
       //*****************************************************************************
        dest_ptr = (unsigned short*)&adc_data_rearranged_buffer[buffer_index][0] + HALF_BUFFER_LENGTH;

       //*****************************************************************************
       // Initialize DMA End address value for re-arranged ADC Data from Buffer B.
       //*****************************************************************************
        end_ptr  = (unsigned short*)&adc_data_rearranged_buffer[buffer_index][0]+TOTAL_BUFFER_LENGTH; // End addr for re-arranged
ADC Data from Buffer B.

        do
        {
          *dest_ptr++ = *adc0_ptr++;
          *dest_ptr++ = *adc1_ptr++;

          #if (INTERLEAVED_ADC_COUNT  >  2)
              *dest_ptr++ = *adc2_ptr++;
          #endif

          #if (INTERLEAVED_ADC_COUNT  >  3)
              *dest_ptr++ = *adc3_ptr++;
          #endif

          #if (INTERLEAVED_ADC_COUNT  >  4)
              *dest_ptr++ = *adc4_ptr++;
          #endif

          #if (INTERLEAVED_ADC_COUNT  >  5)
              *dest_ptr++ = *adc5_ptr++;
          #endif

        }
        while(dest_ptr < end_ptr);    // Continue Re-arranging ADC data until all Buffer A data for active ADCs is complete

        buffer_b_index++;          // Increment buffer_a_index by 1 at the end of re-arrangement of Buffer A data.
    }

    //*****************************************************************************
    // adc_data_rearranged_buffer is a 2 dimensional buffer. If both Buffer A and
```

PIC32MKxxMCFxx 20msps Interleaved ADC Sample Code Example

```
    // Buffer B has been transferred, increment the most significant buffer index
    // for next set of Buffer A and Buffer B data
    //***************************************************************************
    if(buffer_a_index == buffer_b_index) buffer_index++;

    //***************************************************************************
    // adc_data_rearranged_buffer is also circular in nature, after reaching the
    // end of the buffer, the index is re-initialized to start of the buffer.
    //***************************************************************************
    if (buffer_index >= BUFFER_RPT_COUNT) buffer_index = 0;

    IFS3bits.AD1FCBTIF = 0;  // Clear the ADC DMA Interrupt

} //End DMA ISR
```

5. PIC32MZxxxx 12.5msps Four ADC Interleaved Code Example

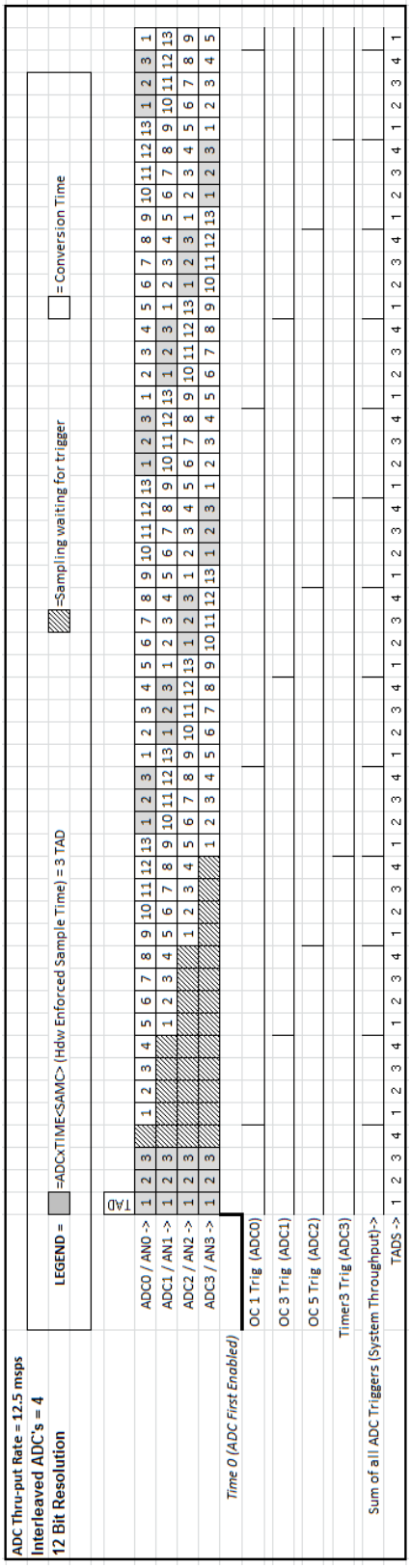


Figure 5-1. PIC32MZ 12bit 12.5 msps Code Example Using Four Interleaved ADCs

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
// ******************************************************************************************
// PIC32MZ Family 12.5msps Interleaved ADC sample code example
//
// SUMMARY:
//    TARGET CPU: PIC32MZxxEFxx >= 100 pin
//    OSC INPUT TYPE = 24 MHz Clock oscillator (EC Mode)
//    SYSCLK=200 MHz
//    PBCLK = 100 MHz
//    ADC TAD = SYSCLK / 4 = 50 MHz = 20ns
//    ADC SAMC=3 TAD
//    Interleaved ADC SAMPLE RATE = 12.5msps(max)
//    INTERLEAVED ADCS
//      ADC 0: 12bit mode, ADCTRG1<TRGSRC0> Output Compare 1
//      ADC 1: 12bit mode, ADCTRG1<TRGSRC1> Output Compare 3
//      ADC 2: 12bit mode, ADCTRG1<TRGSRC2> Output Compare 5
//      ADC 3: 12bit mode, ADCTRG1<TRGSRC3> Timer3
//
//    interleaved ADC0/1/2/3 with AN0/AN1/AN2/AN3 inputs respectively
//    (12.5 msps combined throughput rate)
//
// USER NOTE:
// Although the number of interleaved ADC's the user wishes to use is selectable, it is required
// "FOR THIS CODE ONLY" that those ADCx modules be sequential starting from AN0 to the last sequential
// ANx to be used for interleaving. In different code than below, any combination of interleaved ADC are allowed
//
//   NOTE: (FYI only)
//     A user could configure the PIC32MZ to have multiple sets of
//     interleaved ADC's to measure multiple different analog input streams like:
//           ----------------------------  EXAMPLE  1  CONFIGURATION  ----------------------------
//       1) AN0-AN1 pair with triggers OC5 and TMR3 @ 6.25msps(max)
//       2) AN2-AN3 pair with triggers OC1 and TMR5 @ 6.25msps(max)
//           ----------------------------  EXAMPLE  2  CONFIGURATION  ----------------------------
//       3) ADC0-ADC2 (3) ADCs with triggers OC1, OC3 and TMR5 @ 8.333333 msps(max) combined
//       4) ADC3-ADC4 (2) ADCs with triggers OC5 and TMR3 @ 6.25msps(max)
//           ----------------------------  EXAMPLE  3  CONFIGURATION  ----------------------------
//       5) ADC0-ADC3 (4) ADCs with triggers OC1, OC3, OC5 and TMR3 @ 12.5msps(max) combined
// ******************************************************************************************
//
// ******************************************************************************************
// USER NOTE:
// Although the number of interleaved ADC's the user wishes to use is selectable in this code
// example, it is required "FOR THIS CODE ONLY" that those ADCx modules be sequential starting
// from AN0 to the last sequential ANx to be used for interleaving.
// In a user's own original code however they can define any combination
// of ADC's they prefer.
//
// ******************************************************************************
//       F I L E   I N C L U D E S
// ******************************************************************************
#include <xc.h>
#include <stdbool.h>
#include <sys/attribs.h>

// ******************************************************************************
//       C O N F I G U R A T I O N   W O R D S
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
// ******************************************************************************
//
//------------------------------------------------------------
//                        DEVCFG0
//------------------------------------------------------------
#pragma config DEBUG =      OFF
#pragma config JTAGEN =     OFF
#pragma config ICESEL =     ICS_PGx2
#pragma config TRCEN =      OFF
#pragma config BOOTISA =    MIPS32
#pragma config FECCCON =    OFF_UNLOCKED
#pragma config FSLEEP =     OFF
#pragma config DBGPER =     PG_ALL
#pragma config SMCLR =      MCLR_NORM
#pragma config SOSCGAIN =   GAIN_2X
#pragma config SOSCBOOST =  ON
#pragma config POSCGAIN =   GAIN_2X
#pragma config POSCBOOST =  ON
#pragma config EJTAGBEN =   NORMAL
#pragma config CP =         OFF

//------------------------------------------------------------
//                        DEVCFG1
//------------------------------------------------------------
#pragma config FNOSC =      SPLL
#pragma config DMTINTV =    WIN_127_128
#pragma config FSOSCEN =    OFF
#pragma config IESO =       OFF
#pragma config POSCMOD =    EC    //24 Mhz External clock osc
#pragma config OSCIOFNC =   OFF
#pragma config FCKSM =      CSECME
#pragma config WDTPS =      PS1048576
#pragma config WDTSPGM =    STOP
#pragma config FWDTEN =     OFF
#pragma config WINDIS =     NORMAL
#pragma config FWDTWINSZ =  WINSZ_25
#pragma config DMTCNT =     DMT31
#pragma config FDMTEN =     OFF

//------------------------------------------------------------
//                        DEVCFG2
// EC Ext OSC in =24Mhz
// SYSCLK=200Mhz, PBxCLK=100Mhz
//------------------------------------------------------------
#pragma config FPLLIDIV =   DIV_3
#pragma config FPLLRNG =    RANGE_5_10_MHZ
#pragma config FPLLICLK =   PLL_POSC
#pragma config FPLLMULT =   MUL_50
#pragma config FPLLODIV =   DIV_2
#pragma config UPLLFSEL =   FREQ_24MHZ

//------------------------------------------------------------
//                        DEVCFG3
//------------------------------------------------------------
#pragma config USERID =     0x0ADC
#pragma config FMIIEN =     OFF
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
#pragma config FETHIO =     OFF
#pragma config PGL1WAY =    OFF
#pragma config PMDL1WAY =   OFF
#pragma config IOL1WAY =    OFF
#pragma config FUSBIDIO =   OFF

//------------------------------------------------------------
//                          BF1SEQ0
//------------------------------------------------------------
#pragma config TSEQ =       0x0000
#pragma config CSEQ =       0xFFFF

// ************************************************************************************************
//                                  U S E R     D E F I N E S
// Note: User must configure as required
// ************************************************************************************************
#define TAD_TRIGGER_SOURCE_SPACING 4 // TAD trigger spacing value from ADC Table 3-5 or 3-7 accordingly  w/12bit & 4 interleaved
ADCs
#define INTERLEAVED_ADC_COUNT      4 // Number of interleaved ADC (Cannot be <2 or >4). Changing this value will
                                     // automatically scale the number of interleaved ADC and triggers utilized in this code.
#define ADC_DMA_BUFF_SIZE      2048  //ADC DMA buffer size

//
// ************************************************************************************************
// EQUATION #1: (Assumptions: INTERLEAVED_ADC_COUNT=4)
//    (Fastest Triggering possible)
//    TAD_TRIGGER_SOURCE_SPACING(min)  =
//        = ((SAMC(min) + (((#bits Resolution+1) + ADJ) * TAD)) / INTERLEAVED_ADC_COUNT)
//        = (3+12+1) + ADJ) * TAD)) / 4 ADCs)
//        = (16 + ADJ) * TAD)) / 4 ADCs
//        = (16 + 0) * TAD)) / 4 ADCs
//        = 4.0 TAD (minimum trigger spacing possible)
//
// NOTE: "ADJ" term = A user selected whole number adjustment factor between 0-4
//                    that must yield a Minimum TAD Trigger interval that equates to an exact
//                    multiple of a ½ TAD. This represents the minimum TAD_TRIGGER_SOURCE_SPACING
//                    and therefore the fastest throughput possible. Additional TAD_TRIGGER_SOURCE_SPACING
//                    and hence ADC Throughput Rates are possible in ½ TAD trigger spacing increments.
//                    (See examples 1-3 below)
//
//  NOTE: Once the minimum, TAD trigger spacing possible is determined, the user can configure for any
//        additional sampling frequency required if needed in 0.5 TAD trigger spacing increments.
//
//        Example 1: (INTERLEAVED_ADC_COUNT=4)
//          TAD_TRIGGER_SOURCE_SPACING(min) = 4
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 50Mhz / 4.0(min)
//                                          = 12.5msps
//
//        Example 2: (INTERLEAVED_ADC_COUNT=4)
//          TAD_TRIGGER_SOURCE_SPACING = 4.5
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 50Mhz / 4.5
//                                          = 11.111111msps
//
//        Example 3: (INTERLEAVED_ADC_COUNT=4)
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
//          TAD_TRIGGER_SOURCE_SPACING = 5.0
//          interleaved ADC Throughput rate = TAD clock freq / TAD_TRIGGER_SOURCE_SPACING
//                                          = 50Mhz / 5.0
//                                          = 10msps
//
// **************************************************************************************************
//
// **************************************************************************************************
//                              P R O G R A M    D E F I N E S
//                                      (User Do Not Change)
// Note:
//       TAD_TRIGGER_SOURCE_SPACING is multiplied by (2) because
//       OCx and TMR3 trigger source time base is 2x faster than an ADC TAD clock.
//       2 TMR3 clocks = 1 ADC TAD Clock
// ***************************************************************************************************
// ---------------------------------------------------------------------------------------------------------------------
// NOTE: Remember that the peripheral trigger clock frequency is 2x TAD clock frequency.
//       This is why the 2x. The -1 is because the Si peripheral triggers are always
//       on (match+1), so to compensate for correct timing subtract 1.
//

#define ADC_TRIG        (TAD_TRIGGER_SOURCE_SPACING * 2)     //
#define OC1_ADC_TRIG    (ADC_TRIG - 1)                     //
#define OC3_ADC_TRIG    ((2 * ADC_TRIG) - 1)               //
#define OC5_ADC_TRIG    ((3 * ADC_TRIG) - 1)                //
#define T3_ADC_TRIG     ((INTERLEAVED_ADC_COUNT * ADC_TRIG) - 1) //

#define ADC0_IRQ    59    //PIC32MZxxEFxx ADC0 Data_0 IRQ number
#define ADC1_IRQ    60    //PIC32MZxxEFxx ADC1 Data_1 IRQ number
#define ADC2_IRQ    61    //PIC32MZxxEFxx ADC2 Data_2 IRQ number
#define ADC3_IRQ    62    //PIC32MZxxEFxx ADC3 Data_3 IRQ number

#define VirtAddr_TO_PhysAddr(v) ((unsigned long)(v) & 0x1FFFFFFF)

#if ((INTERLEAVED_ADC_COUNT  <  2) | (INTERLEAVED_ADC_COUNT  >  4))`
#error  INTERLEAVED_ADC_COUNT error, it is either less than 2 or greater than 4
#endif

// **************************************************************************************************
//        F U N C T I O N     P R O T O T Y P E S
// **************************************************************************************************
void init_ADC(void);
void init_DMA(void);
void init_TMR3(void);

// **************************************************************************************************
//        G L O B A L   V A R I A B L E S
// **************************************************************************************************
uint32_t __attribute__((coherent)) adc_bufA[INTERLEAVED_ADC_COUNT * ADC_DMA_BUFF_SIZE];
uint32_t __attribute__((coherent)) adc_bufB[INTERLEAVED_ADC_COUNT * ADC_DMA_BUFF_SIZE];
uint32_t adc_buf_index = 0;
bool adc_dma_buf_full_flg = 0;

// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $                 M A I N    R O U T I N E                        $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
int main(void)
{
    // ******************************************************************************
    // CPU Performance Optimization:
    // ******************************************************************************
    register unsigned long tmp_cache;                 //KSEG0 cache enable
    asm("mfc0 %0,$16,0" :  "=r"(tmp_cache));
    tmp_cache = (tmp_cache & ~7) | 3;
    asm("mtc0 %0,$16,0" :: "r" (tmp_cache));

    PRECONbits.PFMWS=2;           //Flash wait states = 2 CPU clock cycles @ 200Mhz
    PRECONbits.PREFEN = 2;        //Enable predictive prefetch for CPU instructions and CPU data
    PRISSbits.PRI7SS = 7;    //DMA Interrupt with priority level of 7 uses Shadow Set 7
    PRISSbits.PRI6SS = 6;    //Interrupt with priority level of 6 uses Shadow Set 6
    PRISSbits.PRI5SS = 5;    //Interrupt with priority level of 5 uses Shadow Set 5
    PRISSbits.PRI4SS = 4;    //Interrupt with priority level of 4 uses Shadow Set 4
    PRISSbits.PRI3SS = 3;    //Interrupt with priority level of 3 uses Shadow Set 3
    PRISSbits.PRI2SS = 2;    //Interrupt with priority level of 2 uses Shadow Set 2

    INTCONbits.MVEC = 1;    //Enable multi-vector interrupts
    __builtin_mtc0(12,0,(__builtin_mfc0(12,0) | 0x0001)); // Global Interrupt Enable

    // ******************************************************************************
    // * Disable all ANx pins that are enabled by default on reset
    // ******************************************************************************
    ANSELACLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELBCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELCCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELDCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELECLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELFCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELGCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELHCLR = 0xFFFF;        //Disable all analog ANx input pins
    ANSELJCLR = 0xFFFF;        //Disable all analog ANx input pins

    // ******************************************************************************
    // Initialize and enable ADC(s) followed by ADC trigger sources 2nd
    // ******************************************************************************
    init_ADC();               //Initialize and enable interleaved ADCs first before PWM ADC triggers are enabled

    while(1)
    {
        if(adc_dma_buf_full_flg)    //If current DMA buffer is full
        {
            adc_dma_buf_full_flg = 0;    //Clr DMA buff full flag

            //*****************************************************************************
            // In main(), "adc_buf_index" represents the current active buffer being
            // filled as the companion buffer is already full.
            //*****************************************************************************
            if(adc_buf_index == 0)    //if "adc_bufA" active then "adc_bufB" full
            {
                        // User must process here previously filled ADC DMA Buffer B ptr starting at "adc_bufB"
            }
            else            //if "adc_bufB" active then "adc_bufA" full
            {
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
                            // User must process here previously filled ADC DMA Buffer A ptr starting at "adc_bufA"
            }
        }
        //*********************************
        //* USERS OTHER MAIN CODE GOES HERE
        //*********************************
    }
}


// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $        F U N C T I O N   S U B - R O U T I N E S                 $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$


// ***************************************************************************
// * init_ADC(void)
// *  I N T E R L E A V E D   A D C   I N I T I L I Z A T I O N   R O U T I N E
// *   Used analog inputs if:
// *      INTERLEAVED_ADC_COUNT  = 2   (Analog inputs, AN0 and AN1 only)
// *      INTERLEAVED_ADC_COUNT  = 3   (Analog inputs, AN0, AN1 and AN2 only)
// *      INTERLEAVED_ADC_COUNT  = 4   (Analog inputs, AN0, AN1, AN2 and AN3 only)
// ***************************************************************************
// ***************************************************************************
//  These steps are already done in this code example, this is just a reminder
//   to the user for any new original code development.
//
// CAUTION:(Required)
// 1) The user SHOULD always set the minimum sampling time of each Class_1 ADC
//      equal to 3TAD as defined by ADCxTIME<SAMC>.
// 2) The user should insure that the selected trigger source time base clock rate is
//      exactly 2x faster than the ADC clock TAD rate. This will provide the highest
//      timing resolution along with the most sampling rate options as defined in
//      Table 3-5 or 3-7 accordingly in this app note.
// 3) The user selected trigger sources have to be able to be mapped to a single
//      synchronized time base.
//      For example:
//        a) TMR3 & OCx (x=1,3,5 PIC32MZxxxx)
//        b) TMR3 & OCx (x=1-4 PIC32MKxxGPxx)
//        c) PTMRx & TRIGx ( x=1-12 PIC32MKxxMCxx PWM)
//        d) PTMRx & STRIGx ( x=1-12 PIC32MKxxMCxx PWM)
// 4) In a same group of interleaved ADC's a user "MUST NOT" use two separate timer trigger sources.
//     Example: ADC0, ADC1 and ADC2 are being used in an interleaved group.
//       Trigger sources can be any combination of valid ADC OCx edge triggered sources synchronized to a SINGLE TMRx.
//       It would not be allowed to use in the same group of interleaved ADCs OCx, TMR3 & TMR5 for example.
//       Separate TMRx in different interleaved ADC groups is OK.
// 5  If using the DMA for the ADC the user "MUST" use SYSCLK as the ADC
//     source clock, (i.e. ADCCON3<ADCSEL>).
// 6) User must configure ADC for 512 TAD warm-up time
//     (ADCANCONbits.WKUPCLKCNT = 0x9) "BEFORE" enabling ADC.
// 7) Users SW  must wait for ADC BANGAP and ADC warm-up time "AFTER"
//      enabling ADC and before activating the ADC trigger sources.
// 8) ADC must always be initialized 1st "BEFORE" activating the ADC trigger sources
//
// NOTE: Failure to follow these recommendations will result in inaccurate ADC
//             data acquisition and poor performance.
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
//*****************************************************************************
void init_ADC(void)
{
    ADC0CFG = DEVADC0;        //Load ADC0 Calibration values
    ADC1CFG = DEVADC1;        //Load ADC1 Calibration values
    ADC2CFG = DEVADC2;        //Load ADC2 Calibration values
    ADC3CFG = DEVADC3;        //Load ADC3 Calibration values
    ADC4CFG = DEVADC4;        //Load ADC4 Calibration values
    ADC7CFG = DEVADC7;        //Load ADC7 Calibration values

    ADCANCONbits.WKUPCLKCNT = 0x9; // ADC Warm up delay = (512 * TADx)
    ADCCON1bits.AICPMPEN = 0;       //Disable ADC charge pump
    CFGCONbits.IOANCPEN = 0;       //Disable ADC I/O charge pump

    // ***************************************************************************
    // * If using the DMA the user MUST use SYSCLK as the ADC source clock
    // ***************************************************************************
    ADCCON1bits.FSSCLKEN = 1;       //Fast synchronous SYSCLK to ADC control clock is enabled
    ADCCON3bits.ADCSEL = 1;        // Select ADC input clock source = SYSCLK 200Mhz
    ADCCON3bits.CONCLKDIV = 1;     // Analog-to-Digital Control Clock (TQ) Divider = SYSCLK/2

    // ***************************************************************************
    // * ADC0 Module setup:
    // *   TAD = SYSCLK/4, 12bit, SAMC=3TAD, OC1 Trigger
    // ***************************************************************************
    ANSELBSET = 0x1;         //Enable analog AN0 input
    ADC0TIMEbits.ADCDIV = 1;    // ADCx Clock Divisor, Divide by 2, 50Mhz
    ADC0TIMEbits.SAMC = 1;    // 3 TAD (Hardware enforced sample time)
    ADC0TIMEbits.SELRES = 3;    // ADC0 12bit resolution mode
    ADCTRG1bits.TRGSRC0 = 8;    // Set AN0 to trigger from OC1
    ADCANCONbits.ANEN0 = 1;    // Enable ADC0 analog bias/logic
    ADCCON3bits.DIGEN0 = 1;    // Enable ADC0 digital logic
    ADCGIRQEN1bits.AGIEN0 = 1;        //Enb ADC0 AN0 interrupts for DMA

    // ***************************************************************************
    // * ADC1 Module setup:
    // *   TAD = SYSCLK/4, 12bit, SAMC=3TAD
    // *   if(INTERLEAVED_ADC_COUNT == 2) TMR3 trigger else OC3 Trigger
    // ***************************************************************************
    ANSELBSET = 0x2;         //Enable analog AN1 input
    ADC1TIMEbits.ADCDIV = 1;    // ADCx Clock Divisor, Divide by 2, 50Mhz
    ADC1TIMEbits.SAMC = 1;    // 3 TAD
    ADC1TIMEbits.SELRES = 3;    // ADC1 12bit resolution mode

    if(INTERLEAVED_ADC_COUNT == 2) ADCTRG1bits.TRGSRC1 = 6; // Set ADC1 to trigger from TMR3
    else ADCTRG1bits.TRGSRC1 = 9;    // Set ADC1 to trigger from OC3

    ADCANCONbits.ANEN1 = 1;    // Enable ADC1 analog bias/logic
    ADCCON3bits.DIGEN1 = 1;    // Enable ADC1 digital logic
    ADCGIRQEN1bits.AGIEN1 = 1;    //Enb ADC1 AN1 interrupts for DMA

    // ***************************************************************************
    // * ADC2 Module setup:
    // *    TAD = SYSCLK/4, 12bit, SAMC=3TAD, OC5 Trigger
    // *   if(INTERLEAVED_ADC_COUNT == 3) TMR3 trigger else OC5 Trigger
    // ***************************************************************************
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
    if(INTERLEAVED_ADC_COUNT  >  2)
    {
        ANSELBSET = 0x4;        //Enable analog AN2 input
        ADC2TIMEbits.ADCDIV = 1;    // ADCx Clock Divisor, Divide by 2, 50Mhz
        ADC2TIMEbits.SAMC = 1;        // 3 TAD
        ADC2TIMEbits.SELRES = 3;    // ADC2 12bit resolution mode

        if(INTERLEAVED_ADC_COUNT == 3) ADCTRG1bits.TRGSRC2 = 6; // Set ADC2 to trigger from TMR3
        else ADCTRG1bits.TRGSRC2 = 10; // Set ADC2 to trigger from OC5

        ADCANCONbits.ANEN2 = 1;    // Enable ADC2 analog bias/logic
        ADCCON3bits.DIGEN2 = 1;    // Enable ADC2 digital logic
        ADCGIRQEN1bits.AGIEN2 = 1;    //Enb ADC2 AN2 interrupts for DMA
    }

    // ******************************************************************************
    // * ADC3 Module setup:
    // *   if(INTERLEAVED_ADC_COUNT == 4)
    // *   TAD = SYSCLK/4, 12bit, SAMC=3TAD, TMR3 Trigger
    // ******************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  3)
    {
        ANSELBSET = 0x8;        //Enable analog AN3 input
        ADC3TIMEbits.ADCDIV = 1;    // ADCx Clock Divisor, Divide by 2, 50Mhz
        ADC3TIMEbits.SAMC = 1;        // 3 TAD
        ADC3TIMEbits.SELRES = 3;    // ADC3 12bit resolution mode
        ADCTRG1bits.TRGSRC3 = 6;    // Set ADC3 to trigger from TMR3
        ADCANCONbits.ANEN3 = 1;    // Enable ADC3 analog bias/logic
        ADCCON3bits.DIGEN3 = 1;    // Enable ADC3 digital logic
        ADCGIRQEN1bits.AGIEN3 = 1;    //Enb ADC3 AN3 interrupts for DMA
    }

    ADCCON1bits.ON = 1;               // Turn the ADC on

    // ******************************************************************************
    // * Waiting for ADC warm-up time and ADC bandgap reference to stabilize
    // ******************************************************************************
    while(!ADCCON2bits.BGVRRDY);    // Wait until the reference voltage is ready
    while(!ADCANCONbits.WKRDY0);     // Wait until ADC0 is ready
    while(!ADCANCONbits.WKRDY1);     // Wait until ADC1 is ready
    if(INTERLEAVED_ADC_COUNT  >  2) while(!ADCANCONbits.WKRDY2); // Wait until ADC2 is ready
    if(INTERLEAVED_ADC_COUNT  >  3) while(!ADCANCONbits.WKRDY3); // Wait until ADC3 is ready

    ADCDATA0;   // Read ADC data to make sure that data ready bits are clear.
    ADCDATA1;   //
    ADCDATA2;   //
    ADCDATA3;   //

    init_DMA();        //Initialize DMA
    init_TMR3();    //Initialize and enable ADC interleaved triggers
}


// ************************************************************
// *    D M A   I N I T I A L I Z A T I O N   R O U T I N E   *
// ************************************************************
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
void init_DMA(void)
{
    // *********************************************************
    // * Set the DMA trigger source.  Each interrupt on the indicated
    // * ADC_DATA_VECTOR begins a DMA transfer.
    // *********************************************************
    if(INTERLEAVED_ADC_COUNT == 2)DCH0ECONbits.CHSIRQ = ADC1_IRQ; //DMA_TRIGGER_ADC0_DATA0;
    if(INTERLEAVED_ADC_COUNT > 2)DCH0ECONbits.CHSIRQ = ADC2_IRQ; //DMA_TRIGGER_ADC2_DATA2;
    if(INTERLEAVED_ADC_COUNT > 3)DCH0ECONbits.CHSIRQ = ADC3_IRQ; //DMA_TRIGGER_ADC3_DATA3;

    // *********************************************************
    // * Enable DMA Channel Start IRQ.
    // *********************************************************
    DCH0ECONbits.SIRQEN = 1;    //DMA Chan_0 Start IRQ Enable

    // *********************************************************
    // * Set DMA Source Starting Address
    // *********************************************************
    DCH0SSA = (uint32_t) VirtAddr_TO_PhysAddr((const void *)&ADCDATA0);

    // *********************************************************
    // * Set DMA Source Size in bytes
    // *********************************************************
    if(INTERLEAVED_ADC_COUNT == 2) DCH0SSIZ = 8;  // ADC0 + ADC1, 4+4bytes = 8bytes
    if(INTERLEAVED_ADC_COUNT > 2) DCH0SSIZ = 12;  //ADC0 + ADC1 + ADC2 = 12bytes
    if(INTERLEAVED_ADC_COUNT > 3) DCH0SSIZ = 16;  //ADC0 + ADC1 + ADC2 + ADC3 = 16bytes

    // *********************************************************
    // * Set DMA Destination Starting Address
    // *********************************************************
    DCH0DSA = (uint32_t) VirtAddr_TO_PhysAddr((const void *)&adc_bufA[0]);

    // *********************************************************
    // * Set DMA Destination Size
    // *********************************************************
    DCH0DSIZ = (uint16_t) INTERLEAVED_ADC_COUNT*(ADC_DMA_BUFF_SIZE)*sizeof(adc_bufA[0]);

    // *********************************************************
    // * Set Cell Size
    // *********************************************************
    DCH0CSIZ = (uint16_t) sizeof(adc_bufA[0]);

    DCH0INTbits.CHDDIF = 0;    //Clr Chan Dest Done Int Flg
    DCH0INTbits.CHDDIE = 1;    //Enable DMA Destination Interrupt

    // *********************************************************
    // * Enable DMA channel 0
    // *********************************************************
    DCH0CONbits.CHEN = 1;    //DMA Chan 0 enable

    // *********************************************************
    // * Enable DMA and DMA interrupts
    // *********************************************************
    IFS4bits.DMA0IF = 0;    //Clr DMA Chan0 int flg
    IPC33bits.DMA0IP = 7;    //Set DMA Chan0 int priority 7
    IPC33bits.DMA0IS = 3;    //Set DMA Chan0 sub priority 3
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
    IEC4bits.DMA0IE = 1;    //Enable DMA Chan0 interrupt

    DMACONbits.ON = 1;    //Enb DMA
}


// *****************************************************************************
// * I N T E R L E A V E D   A D C   T R I G G E R   I N I T I A L I Z A T I O N
// *****************************************************************************
void init_TMR3(void)
{
    // *************************************************************************
    // * AN0-ANx Interleaved Triggers: (x=1-3)
    // * Same analog input connected to all interleaved ANx inputs w/staggered triggers:
    // *************************************************************************
    PR3 = T3_ADC_TRIG;        //ADC3 TRM3 Trigger throughput rate

    // *************************************************************************
    // * ADC0 Trigger setup:
    // *   TMR3 = OC1 time base, continuous pulse
    // *************************************************************************
    CFGCONbits.OCACLK = 0;    //
    OC1CONbits.OCTSEL = 1;    //Timer"y" is clk source for OC1 module
    OC1CONbits.OCM = 5;     //Init OC1 low; gen continuous output pulses
    OC1R = OC1_ADC_TRIG;    //ADC0 OC1 Trigger throughput rate
    OC1RS = OC1_ADC_TRIG+2;
    OC1CONSET = 0x8000;        // Enable OC1

    // *************************************************************************
    // * ADCx Trigger setup:
    // *   TMR3 = OC3 time base, continuous pulse
    // *************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  2)
    {
        OC3CONbits.OCTSEL = 1;    //Timer"y" is clk source for OC3 module
        OC3CONbits.OCM = 5;     //Init OC3 low; gen continuous output pulses
        OC3R = OC3_ADC_TRIG;    //ADC1 OC3 Trigger throughput rate
        OC3RS  = OC3_ADC_TRIG+2;
        OC3CONSET = 0x8000;    // Enable OC3
    }

    // *************************************************************************
    // * ADCy Trigger setup:
    // *   TMR3 = OC5 time base, continuous pulse
    // *************************************************************************
    if(INTERLEAVED_ADC_COUNT  >  3)
    {
        OC5CONbits.OCTSEL = 1;    //Timer"y" is clk source for OC5 module
        OC5CONbits.OCM = 5;     //Init OC5 low; gen continuous output pulses
        OC5R = OC5_ADC_TRIG;    //ADC2 OC5 Trigger throughput rate
        OC5RS  = OC5_ADC_TRIG+2;
        OC5CONSET = 0x8000;    // Enable OC5
    }

    T3CONSET = 0x8000;        //Start TMR3
```

PIC32MZ Family 12.5msps Interleaved ADC Sample Code Example

```
}  //End TMR3 Subroutine


// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// $     I N T E R R U P T   S E R V I C E    R O U T I N E S        $
// $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$

// ***********************************************************************************************
//        D M A   I S R (Interrupt Service Routine)
// ***********************************************************************************************
void __attribute__((interrupt(ipl7auto), vector(_DMA0_VECTOR), aligned(16), nomips16)) isr ()
{
    IFS4bits.DMA0IF = 0;    //Clr DMA Chan 0 int flg
    DCH0INTbits.CHDDIF = 0;    //Clr DMA dest done flg

    adc_dma_buf_full_flg = 1;    //Current DMA buffer is full

    if(adc_buf_index == 0)    //if "adc_bufA" full, change DMA dest to "adc_bufB"
    {
        DCH0DSA = (uint32_t) VirtAddr_TO_PhysAddr((const void *)&adc_bufB[0]);
        adc_buf_index = 1;
    }
    else            //if "adc_bufB" full, change DMA dest to "adc_bufA"
    {
        DCH0DSA = (uint32_t) VirtAddr_TO_PhysAddr((const void *)&adc_bufA[0]);
        adc_buf_index = 0;
    }
    DCH0CONbits.CHEN = 1;    //DMA Chan 0 enable
}
```

6. ADC DMA Requirements

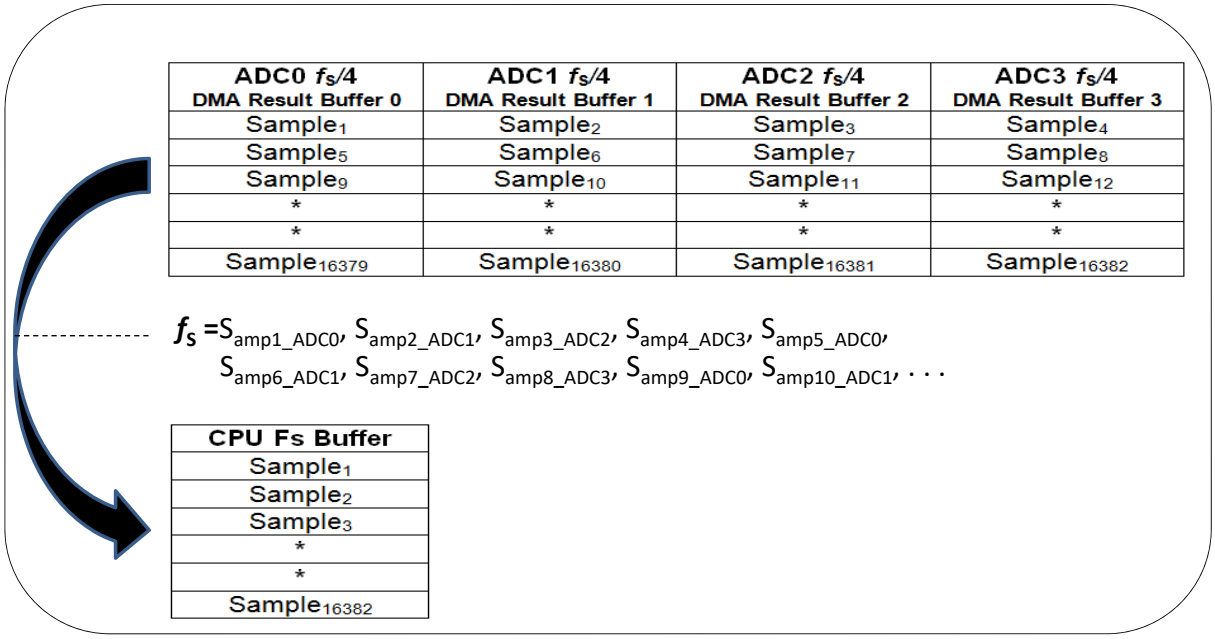
There are architectural differences between the PIC32MKxxxx and PIC32MZxxEF/DAxx families, and that will affect the method of retrieving and processing the ADC DMA buffer information. Therefore the code must be prepared to deal with these differences which are described as follows.

KEY LEARNING: When using the DMA for servicing the ADC the user must use the SYSCLK as the ADC clock source to bypass the ADC clock synchronization logic, which will delay the relationship between the external trigger clock sources and the ADC clock thereby corrupting the samples by skewing and phase shifting the expected data sample rate.

6.1 PIC32MK Family DMA Architecture

The PIC32MKxxxx family has dedicated DMA bus masters. As each ADC has its own dedicated DMA, the system general purpose DMA module cannot access any of the ADCs. The dedicated DMA bus maters, which will be an advantage in standard Non-Interleaved mode, present a challenge for Interleaved mode. Each ADC has independent dedicated DMAs and fixed independent buffers. In Interleave mode, each ADC samples a fraction of the analog input signal, (i.e., 1 / #interleaved ADC), into separate physical buffers, the CPU must therefore reassemble the samples from each ADC DMA buffer into a second SRAM buffer into sequential order for subsequent CPU processing. At the maximum 20 msps sample rate and 120 Mhz CPU clock, this will leave approximately 10-15 us for application processing before the buffers must be serviced again. Reading and reordering (6) buffers into a final result buffer takes almost 38 us of CPU bandwidth. The following example describes the PIC32MKxxxx using four interleaved ADCs and how the data is stored by the DMA for each.

Figure 6-1. Example of PIC32MK Four Interleaved ADC Result DMA Buffers Each Representing fS/4



For subsequent PIC32MKxxxx, only processing of the ADC data by the users application, hence it is necessary that the CPU un-wrap the fragmented data samples in the DMA buffers into a sequential coherent order in a separate data buffer as depicted in the previous figure.

$f_S = S_{amp1,ADC0}, S_{amp2,ADC1}, S_{amp3, ADC2}, S_{amp4,ADC3}, S_{amp5,ADC0}, S_{amp6,ADC1}, S_{amp7, ADC2}, S_{amp8,ADC3}, S_{amp9,ADC0}, S_{amp10,ADC1}, \ldots$

6.2 PIC32MZ Family DMA Architecture

Unlike the PIC32MK, the PIC32MZ family does not have dedicated DMA masters for each ADC. The general purpose system DMA is used instead which in the case of the interleaved ADCs is a distinct advantage. Since the ADCx result registers are sequential, a single DMA channel can be used to service all the interleaved ADCs, provided the interleaved ADC modules are sequential. This also means that the ADC DMA destination buffer data is automatically in sequential order eliminating the CPU requirement to reorder the data as in the PIC32MK for application processing.

7. Interleaved ADC Performance Data Analysis

7.1 PIC32MZ 10msps Interleaved Test Results Example

To measure the effectiveness of the interleaved ADC technique, four interleaved ADCs were used each sampling at a frequency *f*s/4, (i.e., 2.5 msps/ea), to capture a signal at *f*s, (i.e., 10 msps).

7.1.1 10 msps ADC Interleaved Test Conditions

- $F_{CAPTURE}$ = 13.000 Khz Input Sine Wave @ 1.5v
- Four Class_1 ADC each sampling at FS/4 = 2.5 msps, (i.e. 10 msps cumulative from all four ADCs)
- Analog input signal fed in parallel to primary analog inputs on ADC0/1/2/3 modules

7.1.2 10 msps FFT Analysis Results Summary

Table 7-1. Four Interleaved ADC Modules FFT Results Summary

	Phase Difference between Interleaved ADC module samples	Sampling Error	Noise	
			Peak	Average
ADC0	0	< 0.2%	-62db	-90db
ADC1				
ADC2				
ADC3				

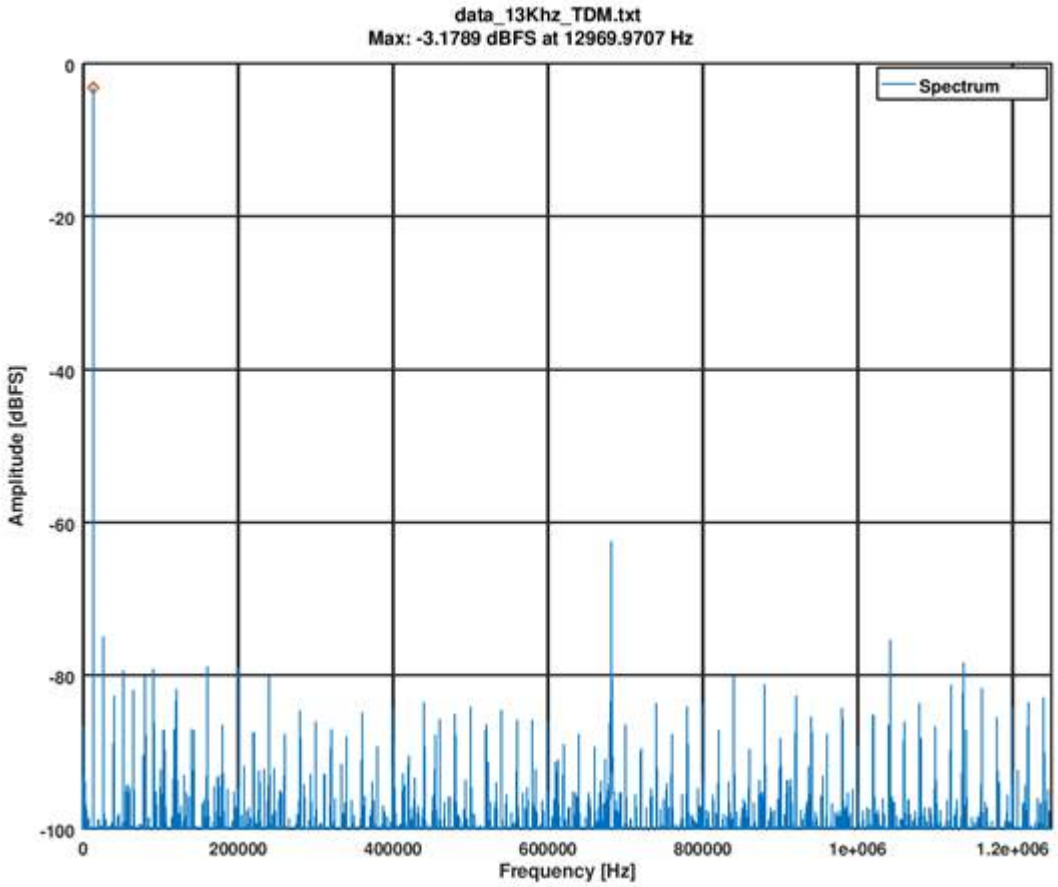
Note: Data was collected on the bench at 25°C on a PIC32MZ Embedded Connectivity with FPU (EF) Starter Kit, part number# (DM320007) which is less than ideal for high-speed ADC signal processing.

In the following five figures each of the four individual interleaved ADC FFT plots are presented followed by the FFT plot of the rollup of the data from all four ADC that represent 10 msps.

7.1.3 ADC0 Interleaved Column 1 FFT Results: (1 of 4)

- F_C actual = 13.000 Khz
- Sample Rate = [Hz]:2.5e6
- DC Offset: 0
- Peak: 12969.9707 Hz : -3.1789 db
- Sampling Error: 0.002

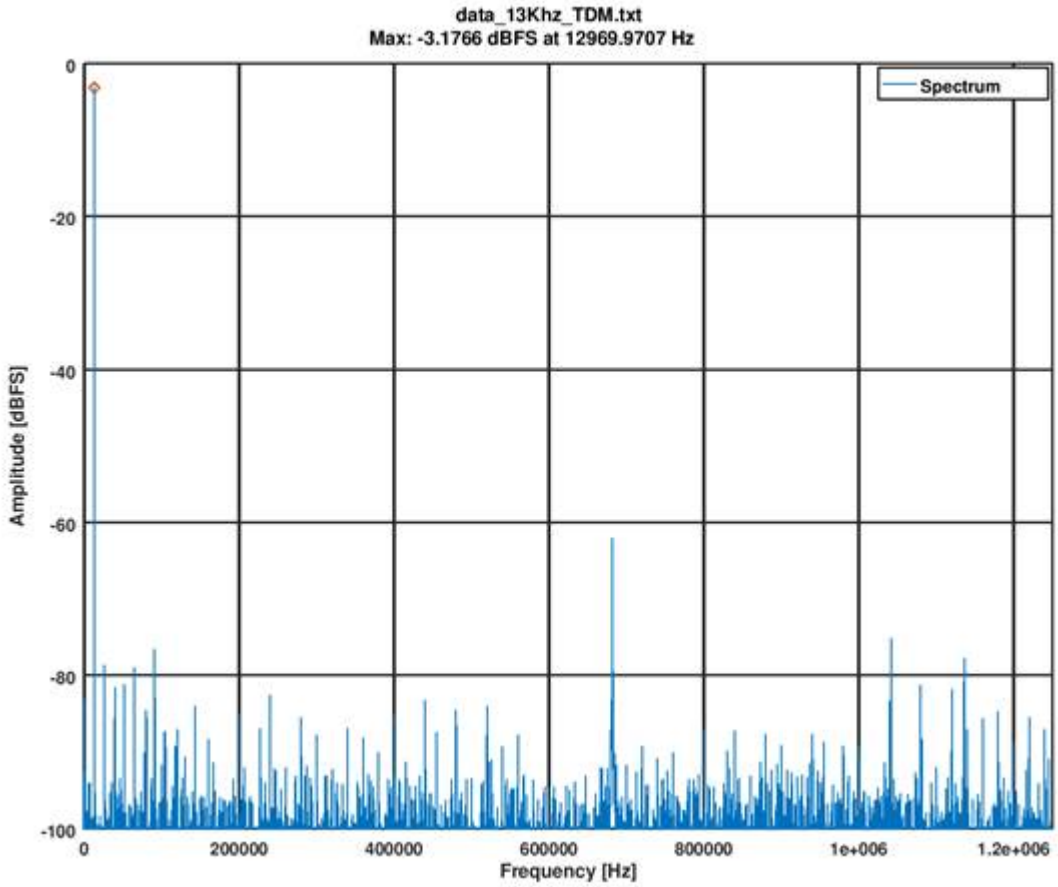
Figure 7-1. ADC0 Interleaved FFT Plot



7.1.4 ADC1 Interleaved Column 2 FFT Results: (2 of 4)

- FC actual = 13.000 Khz
- Sample Rate = [Hz]:2.5e6
- DC Offset: 0
- Peaks: 12969.9707 Hz : -3.1789 db
- Sampling Error: 0.002

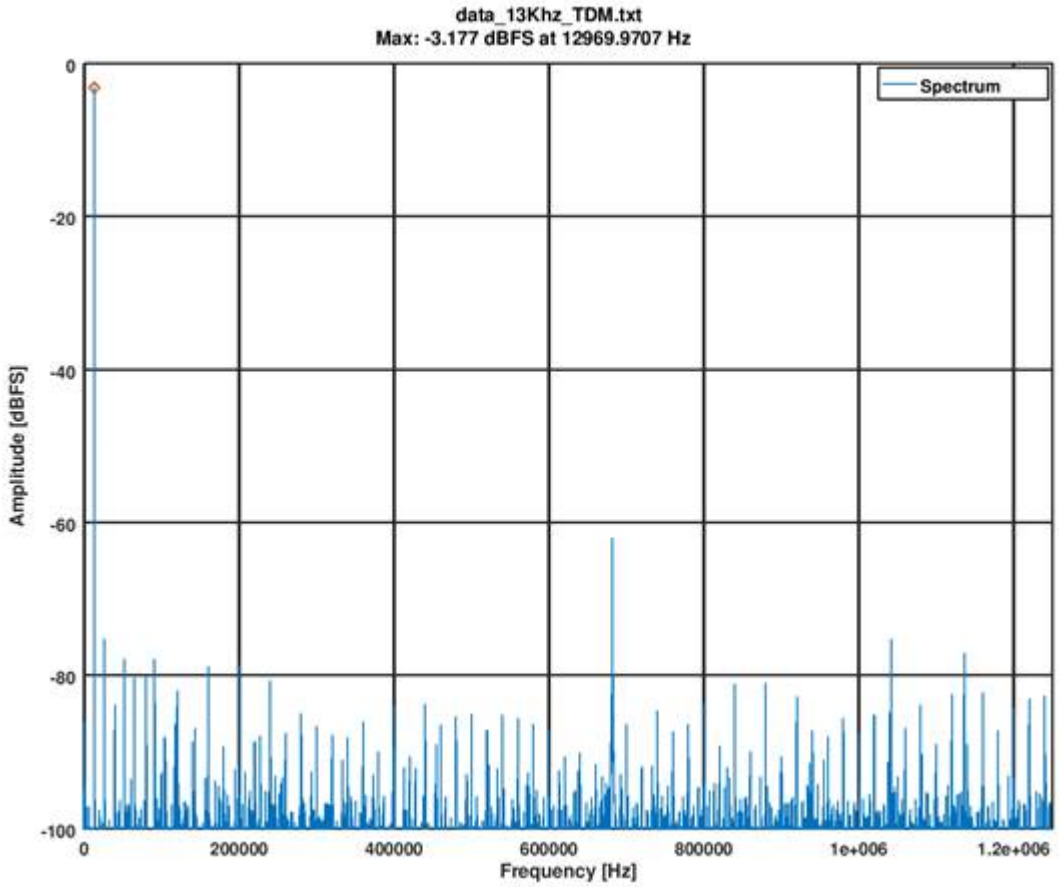
Figure 7-2. ADC1 Interleaved FFT Plot



7.1.5 ADC2 Interleaved Column 3 FFT Results: (3 of 4)

- F_C actual = 13.000 Khz
- Sample Rate = [Hz]:2.5e6
- DC Offset: 0
- Peaks: 12969.9707 Hz : -3.1789 db
- Sampling Error: 0.002

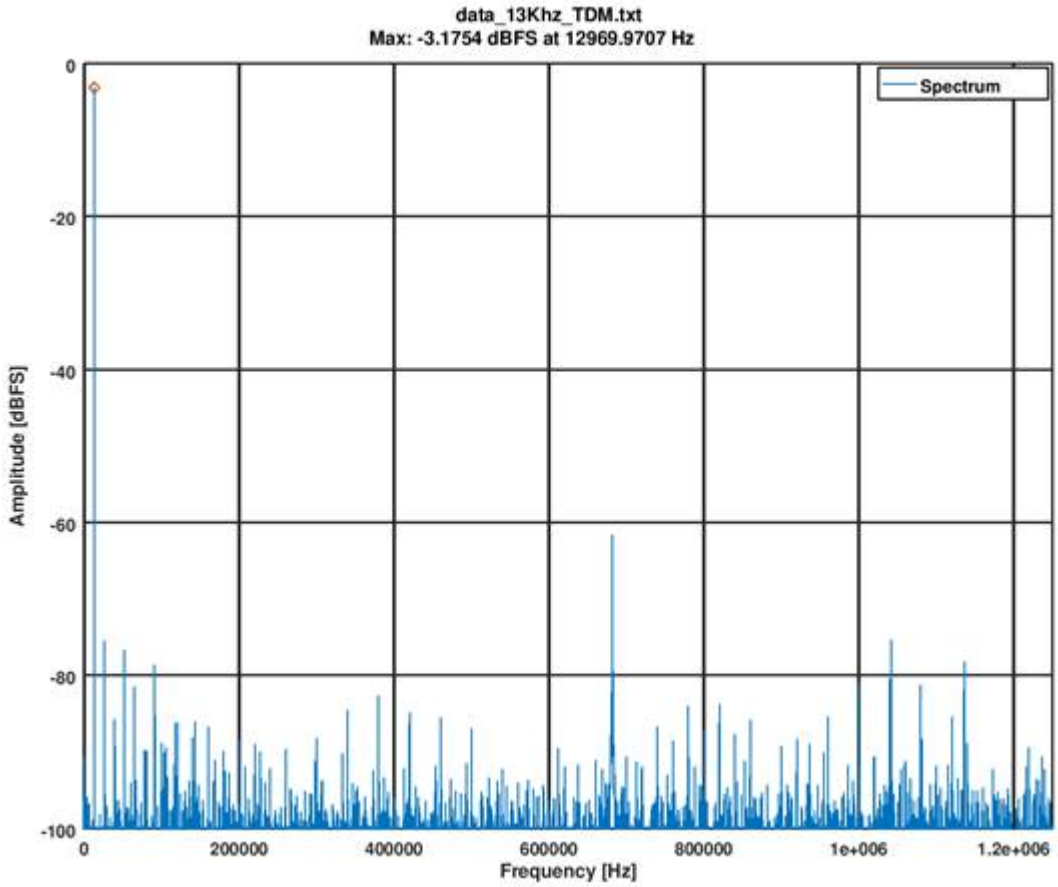
Figure 7-3. ADC2 Interleaved FFT Plot



7.1.6 ADC3 Interleaved Column 4 FTT Results: (4 of 4)

- F_C actual = 13.000 Khz
- Sample Rate = [Hz]:2.5e6
- DC Offset: 0
- Peaks: 12969.9707 Hz : -3.1789 db
- Sampling Error: 0.002

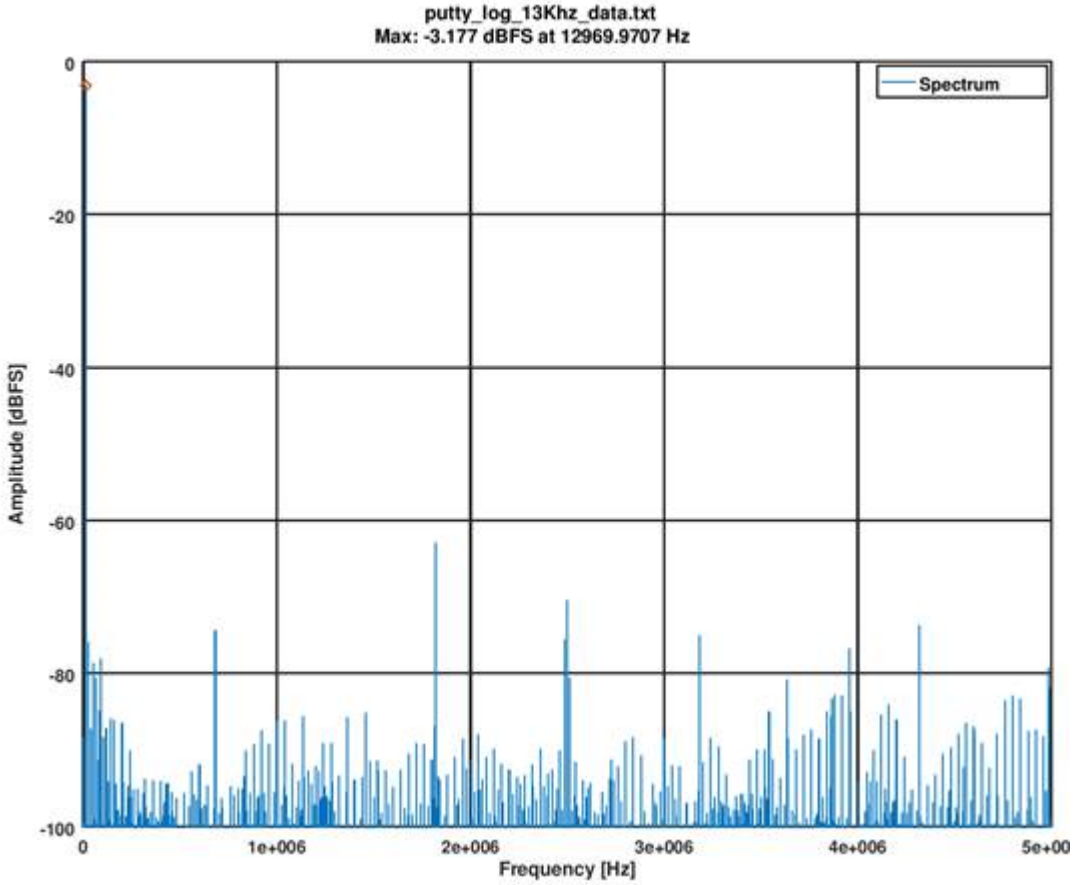
Figure 7-4. ADC3 Interleaved FFT Plot



7.1.7 Combined 10msps ADC0-3 Interleaved FFT Result

- F_C actual = 13.000 Khz
- Sample Rate = [Hz]:10e6 (ADC0 + ADC1 + ADC2 + ADC3)
- DC Offset: 0
- Peaks: 12969.9707 Hz : -3.1777 db
- Sampling Error: 0.002
- Peak spurious 18,18237.3047, Hz -62 db

Figure 7-5. Combined 10msps ADC0 - 3 Interleaved FFT Result



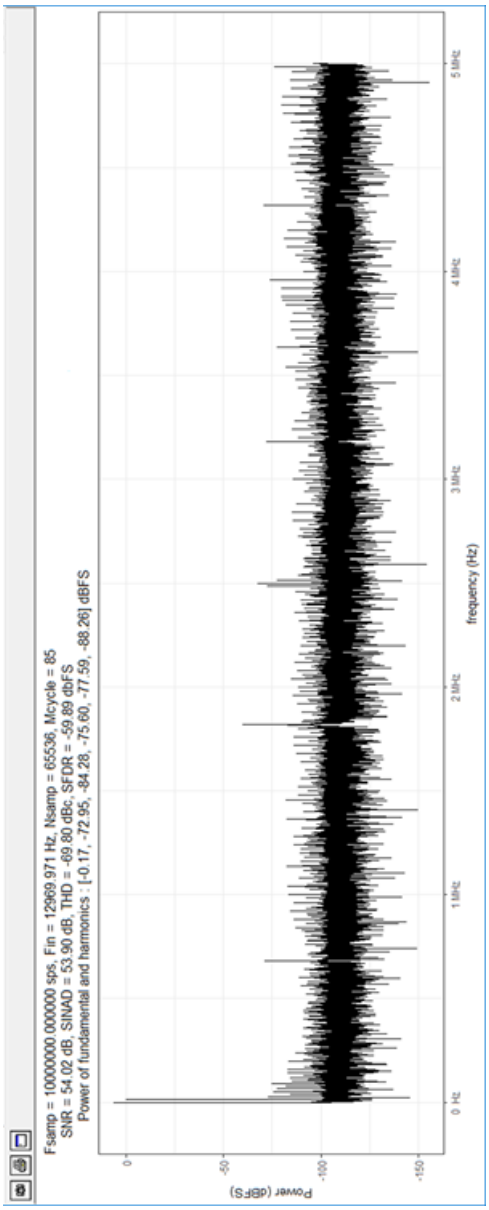


Figure 7-6. 10msps Interleaved ADC SNR, SINAD, THD, SFDR Plot

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-3452-8

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820