8-Bit Delta Sigma ADC Datasheet DELSIG8V 3.2

Copyright © 2002-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45						
1st Order Modulator	1	0	1	116	6	1
2nd Order Modulator	1	0	2	139	9	1

See AN2239, ADC Selection Guide for other converters.

For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

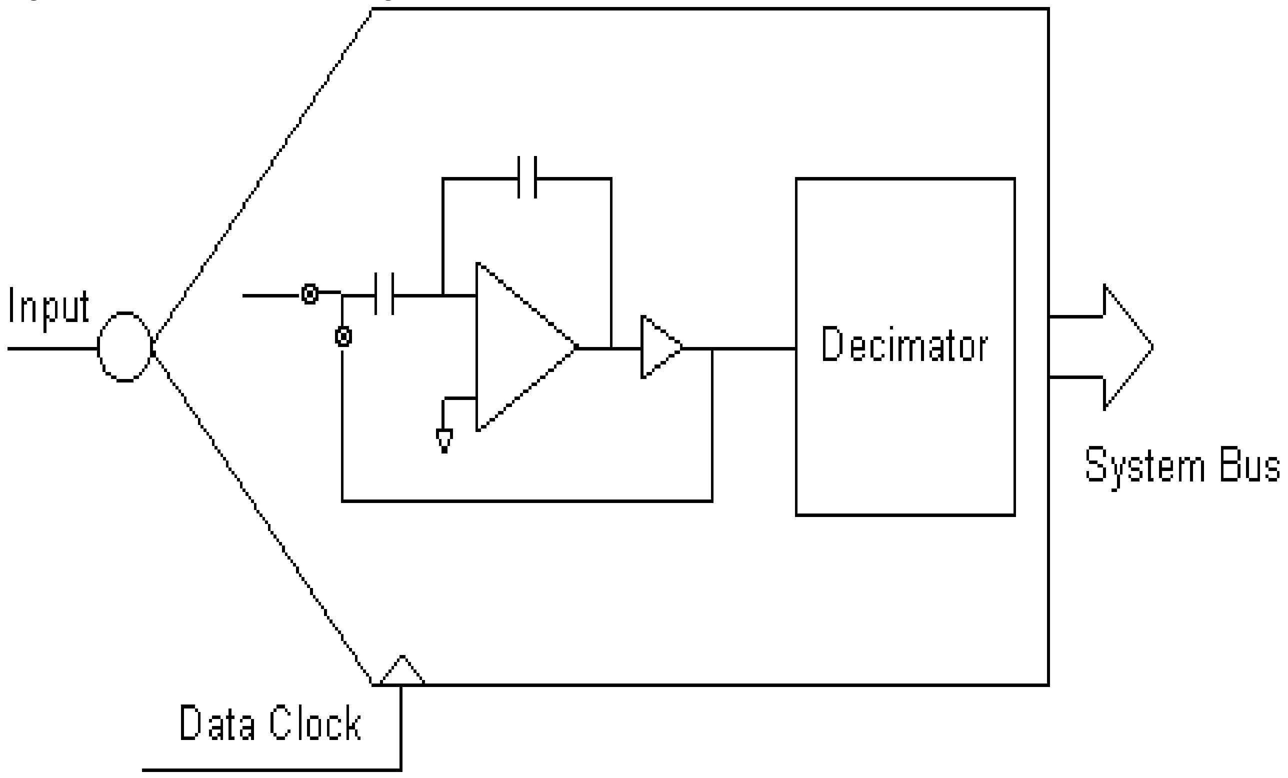
- 8-bit resolution
- Data format available in 2's complement
- Sample rate up to 32 ksps
- 64X over sampling with sinc^2 filter reduces antialias requirements
- Input range defined by internal and external reference options
- Internal or external clock
- Second order modulator available for the CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45families of PSoC devices

Note If this user module is used with the 29K family, it consumes an extra 6 mA. As an alternate, use the Delsig User Module instead.

The DELSIG8 User Module provides an 8-bit 2's complement conversion of an 2.6 volt full scale input signal centered around a user selected AGND, when the reference selection in the global parameter window is set to ± Bandgap. It supports sample rates from 1.8 ksps to 31 ksps. The sample rate is determined by the data clock input and is selectable by the user. Data generated by the DELSIG8 is available in the interrupt routine where the data is collected or through polling functions furnished by the DELSIG8 API.

The DELSIG8 is a pipelined integrating converter, requiring 127 integration cycles to generate a single output sample. If this converter is to have a multiplexed input, two samples must pass before the third and following samples are valid. Note that you need to review the Parameters section before module placement.

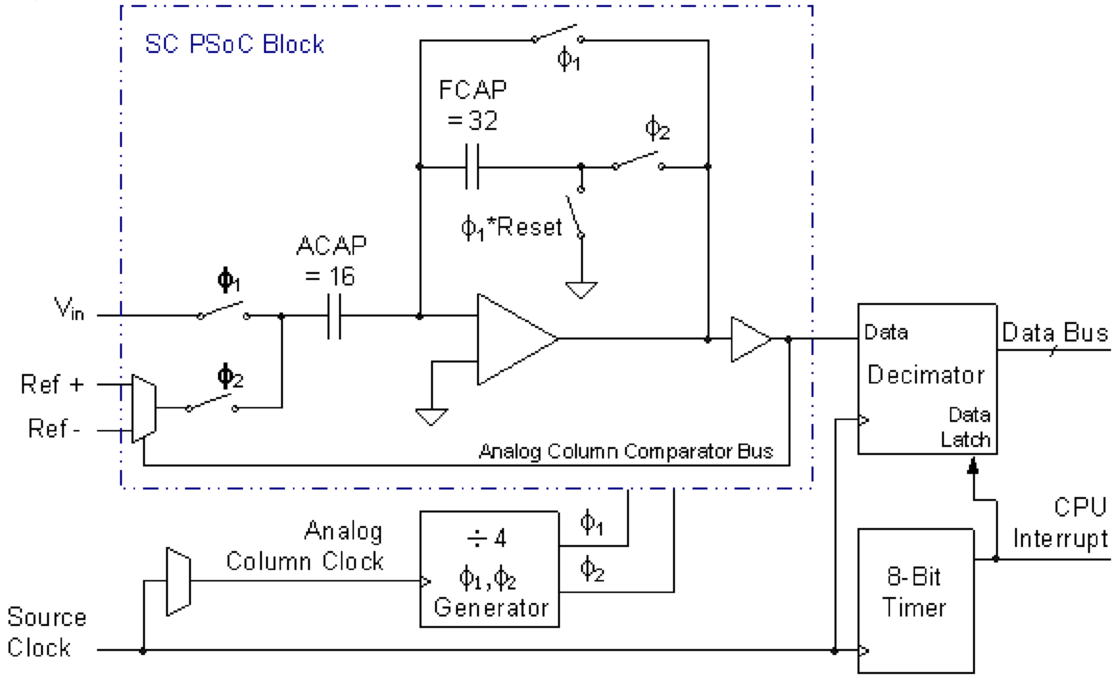
Figure 1. DELSIG8 Block Diagram



Functional Description

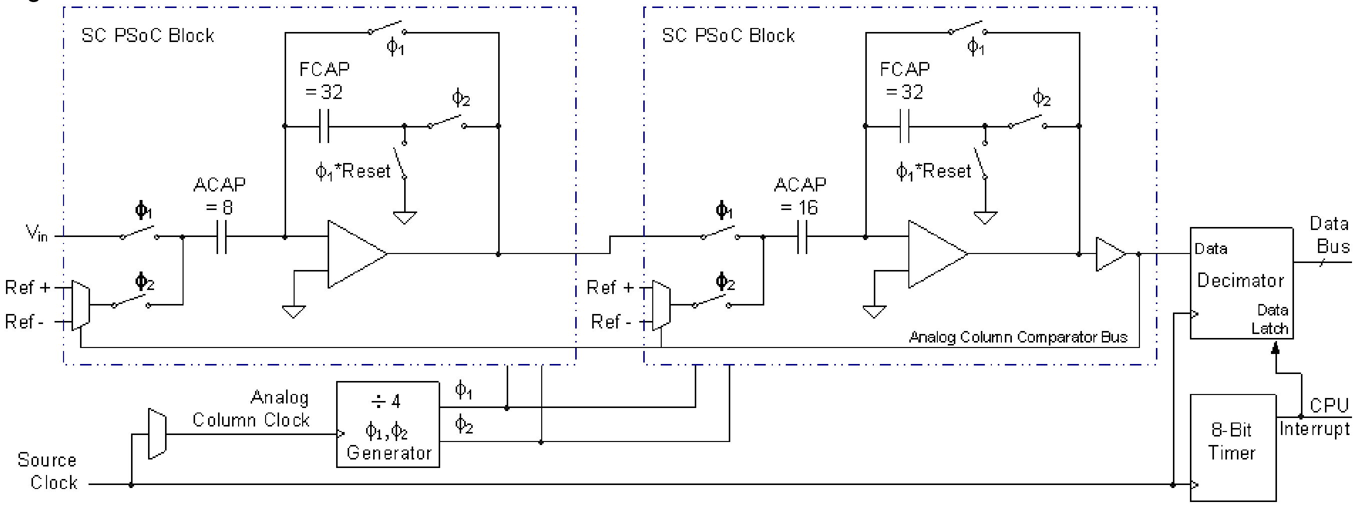
The DELSIG8 provides a first order modulator formed from a single analog switched capacitor PSoC block, one digital PSoC block, and the decimator, as shown in this figure:

Figure 2. Schematic of the DELSIG8 with First Order Modulator



A second order modulator can be constructed in the CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45families of PSoC devices using a second switched capacitor PSoC block. This improves performance by shifting more of the quantization noise out of band, for improved SNR. The second order circuit shown in the following schematic diagram uses an analog column comparator bus to modulate the reference selection.

Figure 3. Schematic of the DELSIG8 with Second Order Modulator



The range of the DELSIG8 is set at ±V_{Ref}, where V_{Ref} is set by the user in the Global Resources window of PSoC Designer. For fixed scale, V_{Ref} is set to ±$V_{Bandgap}$ or, for the CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45families of PSoC devices, ±1.6 $V_{Bandgap}$. For adjustable scale, V_{Ref} is set to ±Port 2[6]. For supply ratio metric scale, V_{Ref} is set to ±V_{DD}/2.

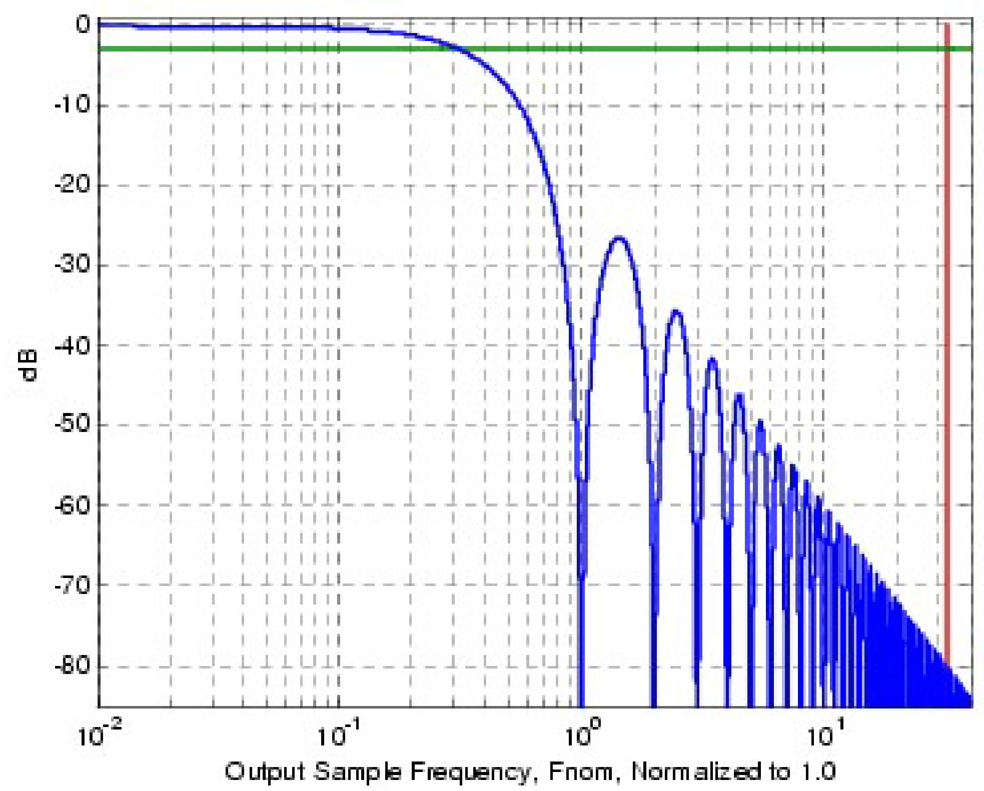
The analog block is configured as an integrator. The reference control is configured so that the reference voltage is either added or subtracted, depending on the output polarity, from the input and placed in the integrator. This reference control attempts to pull the integrator output back towards zero. The single-bit comparator output is fed to the circuitry used to implement a decimator sinc^2 filter. The response of this filter is given by the following z-domain relation:

Equation 1

$$H(z) = \left[\frac{1 - z^{-n}}{1 - z^{-1}}\right]^2, \text{ where n is the decimation level.}$$

The frequency domain magnitude plot below normalizes the frequency so the 8-bit sample rate, F_{nom}=1.0. The -3 dB point occurs just above 0.318?×?F_{nom} and zeros of the function occur at each integer multiple of F_S. Since the DELSIG8 actually samples 64 times faster than the nominal output rate, the Nyquist limit is 32 time higher, 5 octaves above F_{nom}, which significantly reduces the requirements for an antialias filter.

Figure 4. Sinc^2 Decimation Filter Magnitude Response, with -3?dB point and Nyquist Frequency



This filter is implemented with a combination of hardware and software and is used for both the first and second order modulator topologies. The denominator of the filter is a double integrator that is implemented in hardware. It operates at the data rate. The numerator is a double differentiator. It is calculated at the decimation rate and is implemented in software. To make the integrator function as a delta-sigma ADC, the following digital resources are used:

- A timer to allow the proper number of integration cycles.
- A decimator to process the single-bit output stream from the analog block.

Normally, building an 8-bit delta sigma ACD converter requires a 6-bit timer. It would have to be clocked at one-fourth the frequency used to clock the integrator. For this implementation the timer is clocked with the same clock used by the integrator block. This causes the timer to be incremented by four for each integration cycle, thus requiring an 8-bit timer. *When placing this module, it is imperative that it is configured with the same source clock for both the analog and digital blocks. Failure to do so causes it to operate incorrectly.*

The timer is set up to generate an interrupt every 256 counts. At the same time the interrupt is generated, the data in the decimator is latched to output registers. The A/D value is calculated in the interrupt routine.

The conversion equation governing the 8-bit Delta Sigma ADC is shown in the following equation. This equation shows that the input range is limited to V_{ref}:

Equation 2

$$V_{in} = \frac{n-128}{128}V_{ref}$$

Example 1

For a V_{ref} of 1.3V and we can easily calculate the input voltage based on the value read from the incremental ADC at the time the data is ready. The equation which can be used is:

Equation 3

$$V_{in} = \frac{200-128}{128}1.3 = 0.73V$$

The result of the calculation is referenced to AGND. For an ADC data value of 200 the Voltage measured can be calculated to be 0.73V using this equation:

Equation 4

$$V_{in} = \frac{n-128}{128}1.3$$

The value calculated is an ideal value and may differ based on system noise and chips offsets.

To determine the code to be expected given a specific input voltage the equation can be rearranged to give:

Equation 5

$$n = \frac{128 \cdot V_{in}}{V_{ref}} + 128$$

Example 2

For a V_{ref} of 1.3V we can easily calculate the expected ADC code based on the input Voltage. The equation which can be used is:

Equation 6

$$n = \frac{128 \cdot V_{in}}{1.3} + 128$$

For an input voltage of -1V below AGND the code from the ADC can be expected to be 29.53 based on this calculation:

Equation 7

$$n = \frac{128 \cdot (-1)}{1.3} + 128 = 29.53$$

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified here, TA = 25°C, Vdd = 5.0V, Power HIGH, OpAmp bias LOW, output referenced to 2.5V external Analog Ground on P2[4] with 1.25 external Vref on P2[6].

Table 1. 5.0V 2nd Order Modulator DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45Families of Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	
Resolution	---	8	Bits	
Sample Rate	---	.125 to 31.25	ksps	
SNR	46	---	dB	
DC Accuracy				
DNL	0.1	---	LSB	Column Clock 2 MHz
INL	0.5	---	LSB	
Offset Error	10	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.1	--	% FSR	
Operating Current				
Low Power	180	---	uA	
Med Power	840	---	uA	
High Power	3450	---	uA	
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

Table 2. 5.0V 1st Order Modulator DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45Families of Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd		Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	
Resolution	---	8	Bits	
Sample Rate	---	.125 to 31.25	ksps	
SNR	44	---	dB	
DC Accuracy				
DNL	0.6	---	LSB	Column Clock 2 MHz
INL	0.7	---	LSB	
Offset Error	5	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.1	--	% FSR	
Operating Current				
Low Power	50	---	uA	
Med Power	500	---	uA	
High Power	1900	---	uA	
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified here, TA = 25°C, Vdd = 3.3V, Power HIGH, OpAmp bias LOW, output referenced to 1.64V external Analog Ground on P2[4] with 1.25 external Vref on P2[6].

Table 3. 3.3V 2nd Order Modulator DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45Families of Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	

Parameter	Typical	Limit	Units	Conditions and Notes
Resolution	---	8	Bits	
Sample Rate	---	.125 to 31.25	ksps	
SNR	46	---	dB	
DC Accuracy				
DNL	0.1	---	LSB	Column Clock 2 MHz
INL	0.5	---	LSB	
Offset Error	10	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.3	--	% FSR	
Operating Current				
Low Power	130	---	uA	
Med Power	840	---	uA	
High Power	3370	---	uA	
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

Table 4. 3.3V 1st Order Modulator DC and AC Electrical Characteristics, CY8C29/27/24/22xxx, CY8C23x33, CY8CLED08/16, CY8C28x45Families of Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd	V	Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	
Resolution	---	8	Bits	
Sample Rate	---	.125 to 31.25	ksps	
SNR	44	---	dB	
DC Accuracy				
DNL	0.6	---	LSB	Column Clock 2 MHz
INL	0.8	---	LSB	
Offset Error	6	---	mV	
Gain Error				

Parameter	Typical	Limit	Units	Conditions and Notes
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.3	--	% FSR	
Operating Current				
Low Power	50	---	uA	
Med Power	500	---	uA	
High Power	1900	---	uA	
Data Clock	---	0.032 to 8.0	MHz	Input to digital blocks and analog column clock

Electrical Characteristics Notes

1. Includes I/O pin.
2. Reference Gain Error measured by comparing the external reference to $V_{RefHigh}$ and V_{RefLow} routed through the test mux and back out to a pin.
3. Typical values represent parametric norm at +25°C.
4. Input voltages above the maximum generates a maximum positive reading. Input voltages below the minimum generate a maximum negative reading.
5. User module only, not including I/O pin.
6. The input capacitance or impedance is only applicable when input to analog block is directly to a pin.
7. C = input capacitance, clk = data clock (Analog Column Clock).
8. DataClock = SampleRate x 256.
9. SNR = Ratio of power of full scale single tone divided by total noise integrated to Fsample/2.

Placement

The first order modulator design requires two PSoC blocks, one digital and one analog. No inherent limitations govern placement of the analog block; the only considerations are input and clock availability. The digital block, however, must be able to feed the hardware decimator. In the CY8C27xxx family the qualified digital blocks are DBB01, DBB02, DDB05 and DCB06. In the CY8C29/24/22xxx and all other supported device families any of the digital blocks can be used. As noted later, both blocks must use the same source clock.

Placement for second order modulator design differs from the first order design in that there is a second switched capacitor PSoC block. Both analog blocks must lie in the same column so they can share the column comparator bus. The digital block is subject to the same restrictions for both first and second order modulators.

Although there are a number of placements possible for the analog and digital blocks, the DELSIG8 also uses the PSoC device's only hardware decimator. Only one DELSIG8 instance may be placed for a given configuration. With dynamic reconfiguration it is possible to load more than one configuration as long as the blocks do not overlap. Though both instances appear to work, only the output of the one most recently loaded would be correct.

Parameters and Resources

Once a DELSIG8 instance is placed, four parameters must be configured for proper operation: the Input, Clock Phase, Data Clock, and the Polling selection.

Clock Phase

The selection of the Clock Phase is used to synchronize the output of one analog PSoC block to the input of another. The switched capacitor analog PSoC blocks use a two-phase clock ($\Phi 1$, $\Phi 2$) to acquire and transfer signals. Normally, the input to the DELSIG8 is sampled on $\Phi 1$. A problem arises in that many of the user modules autozero their output during $\Phi 1$ and only provide a valid output during $\Phi 2$. If such a module's output is fed to the DELSIG8's input, the DELSIG8 acquires an autozeroed output instead of a valid signal. The Clock Phase selection allows the phases to be swapped, so that the input signal is acquired during $\Phi 2$.

Input

This parameter determines the signal source for the input to the ADC.

TMR Clock

The TMR Clock determines the sample rate. This clock goes to both PSoC blocks of the first order modulator design and to all three PSoC blocks of the second order design.

Note It is imperative that the same clock is selected for both the digital block and the analog column clock or this user module does not function correctly.

Note This parameter is called the "data clock" throughout this datasheet and other PSoC user module datasheets.

The timer is set to provide an interrupt every 256 counts of the TMR clock. The sample rate is defined as:

Equation 8

$$SampleRate = \frac{DataClock}{256}$$

Example 1

If a sample rate of 20 ksps is desired, the TMR clock is:

Equation 9

$$DataClock = SampleRate \times 256 = 20ksps \times 256 = 5.12MHz$$

The CPU overhead is also dependent on the TMR clock. The interrupt routine that calculates the ADC value uses 190 CPU cycles, so its CPU overhead is:

Equation 10

$$CPUOverhead = \frac{190 \times Samplerate}{CPUClock} = \frac{0.743 \times DataClock}{CPUClock}$$

Example 2

If the sample rate is 20 ksps and the CPU clock is 24 MHz, the CPU overhead is:

Equation 11

$$CPUOverhead = \frac{190 \times 20ksps}{24MHz} = 15.8\%$$

Polling

The conversion time for the DELSIG8 can be as little as 30 μs, so fast retrieval of the data may be important. The data produced by the sinc^2 filter, is computed under control of the digital block's Interrupt Service Routine (ISR). Two options are available for accessing this data controlled by the Polling parameter which may be set to Enable or Disable.

When Polling is set to Disable, retrieving the data is accomplished by inserting the code into the assembly language interrupt routine DELSIG8_ADConversion_ISR, located in the assembly file *delsig8INT.asm*. The point to insert code is clearly marked. The section Sample Code, ahead, demonstrates this approach in several examples.

When Polling is set to Enable, two additional bytes of RAM are allocated, one to store a copy of the data and one to describe its availability. These variables and associated polling functions are described in the next section.

Interrupt Generation Control

There following parameter is only available if the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting “ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting “OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns the DELSIG8_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable and constant symbol. In the following descriptions the instance name has been shortened to DELSIG8 for simplicity.

Global Variable DELSIG8_bfStatus

Description:

This variable is available only if the value of the Polling parameter is set to Enable. It is set to a nonzero value when the global variable DELSIG8_cResult contains valid data. This variable may be accessed directly or indirectly through the API functions DELSIG8_ClearFlag, DELSIG8_fIsDataAvailable, and DELSIG8_GetDataClearFlag.

C Prototype:

```
BOOL DELSIG8_bfStatus;
```

Global Variable DELSIG8_cResult

Description:

This variable is available only if the value of the Polling parameter is set to Enable. Each sample of data converted by the DELSIG8 is placed in this variable and the value of the related boolean variable, DELSIG_bfStatus, is set to a nonzero value. The contents of this variable may be accessed directly or indirectly through the API functions DELSIG8_GetData and DELSIG8_GetDataClearFlag. Also see the API function DELSIG8_fIsDataAvailable.

C Prototype:

```
CHAR DELSIG8_cResult;
```

DELSIG8_Start

Description:

Performs all required initialization for this user module and sets the power level for the switched capacitor PSoC block. To actually start the conversion process, call the API function DELSIG8_StartAD after calling DELSIG8_Start.

C Prototype:

```
void  DELSIG8_Start (BYTE bPowerSetting)
```

Assembly:

```
mov   A, bPowerSetting
lcall  DELSIG8_Start
```

Parameters:

bPowerSetting: One byte that specifies the power level. Following reset and configuration, the analog PSoC block assigned to DELSIG8 is powered down. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value
DELSIG8_OFF	0
DELSIG8_LOWPOWER	1
DELSIG8_MEDPOWER	2
DELSIG8_HIGHPOWER	3

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

DELSIG8_SetPower

Description:

Sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  DELSIG8_SetPower (BYTE bPowerSetting)
```

Assembly:

```
mov   A, bPowerSetting
lcall  DELSIG8_SetPower
```

Parameters:

bPowerSetting: Same as the bPowerSetting parameter used for the Start entry point.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

DELSIG8_Stop

Description:

Sets the power level on the switched capacitor PSoC block to OFF.

C Prototype:

```
void  DELSIG8_Stop (void)
```

Assembly:

```
lcall  DELSIG8_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

DELSIG8_StartAD

Description:

Starts the conversion (sampling) process by enabling the timer, the integrator and the decimator.

C Prototype:

```
void  DELSIG8_StartAD (void)
```

Assembly:

```
lcall  DELSIG8_StartAD
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

DELSIG8_StopAD

Description:

Halts the conversion (sampling) process by disabling the timer, resetting the integrator and disabling the decimator.

C Prototype:

```
void  DELSIG8_StopAD (void)
```

Assembly:

```
lcall  DELSIG8_StopAD
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

DELSIG8_fIsDataAvailable

Description:

Checks the availability of sampled data.

C Prototype:

```
BYTE  DELSIG8_fIsDataAvailable(void)
```

Assembly:

```
lcall  DELSIG8_fIsDataAvailable
cmp   A, 0
jz    .DataNotAvailable
```

Parameters:

None

Return Value:

Returns a nonzero value if data has been converted and is ready to read.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

DELSIG8_cGetData

Description:

Returns converted data. DELSIG8_fIsDataAvailable() should be called to verify that the data sample is ready. There is a possibility that the returned data is corrupted if the call to this function is done exactly at the end of an integration period. It is therefore highly recommended that the data retrieval be done at a higher frequency than the sampling rate, or if that cannot be guaranteed that interrupts be turned off before calling this function.

C Prototype:

```
CHAR  DELSIG8_cGetData(void)
```

Assembly:

```
lcall  DELSIG8_cGetData           ; Data will be in A upon return
```

Parameters:

None

Return Value:

Returns the converted data sample in 8-bit 2’s complement format.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

DELSIG8_ClearFlag

Description:

Resets the data available flag.

C Prototype:

```
void  DELSIG8_ClearFlag(void)
```

Assembly:

```
lcall  DELSIG8_ClearFlag
```

Parameters:

None

Return Value:

None

Side Effect:

The global variable DELSIG8_bfStatus is set to zero.The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

DELSIG8_cGetDataClearFlag

Description:

Returns converted data and resets the data available flag. DELSIG8_fIsDataAvailable() should be called to verify that the data sample is ready. There is a possibility that the returned data is corrupted if the call to this function is done exactly at the end of an integration period. It is therefore highly recommended that the data retrieval be done at a higher frequency than the sampling rate, or if that cannot be guaranteed that interrupts be turned off before calling this function.

C Prototype:

```
CHAR  DELSIG8_cGetDataClearFlag(void)
```

Assembly:

```
lcall  DELSIG8_cGetDataClearFlag   ; Data will be in A upon return
```

Parameters:

None

Return Value:

Returns the converted data sample in 8-bit 2’s complement format.

Side Effects:

The global variable DELSIG8_bfStatus is set to zero. The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Sample Firmware Source Code

There are three examples in this section. A polling enabled example, a polling disabled example, and an example of how to alter the interrupt service routine (ISR).

Example 1: Polling Enabled, Direct Access

In this example, polling is used to determine when samples become available. Direct access to the global variables is used for maximum speed. A dummy routine is called to represent further handling of the sample obtained from the DELSIG8 ADC.

The following is an assembly language example.

```
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main
_main:
   M8C_EnableGInt                ; enable global interrupts
   mov   A,DELSIG8_HIGHPOWER      ; Establish power setting...
   call  DELSIG8_Start            ; and initialize
   call  DELSIG8_StartAD          ; Commence sampling process
mainloop:
   cmp   [DELSIG8_bfStatus], 0
   jz    mainloop                 ; spin lock until(data is Available)
   mov   [DELSIG8_bfStatus], 0    ; reset the data available flag
   mov   A, [DELSIG8_cResult]     ; grab the data now that its valid
   call  ProcessSample            ; pass the sample in A to the dummy fcn
   jmp   mainloop


ProcessSample:
   ...                             ; (do something useful with the data)
   ret
```

Here is equivalent code in C:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void ProcessSample( CHAR cSample )
{
    ; // (Do something useful with the data)
}

void main(void)
{
   extern BYTE DELSIG8_bfStatus;
   extern BYTE DELSIG8_cResult;
   M8C_EnableGInt;
 DELSIG8_Start( DELSIG8_HIGHPOWER );
 DELSIG8_StartAD();
 while (1) {
    if ( DELSIG8_bfStatus ) {
       DELSIG8_bfStatus = 0;
       ProcessSample( DELSIG8_cResult );
    }
 }
}
```

Example 2: Polling Enabled, Indirect Access

This example repeats the previous scenario but uses API functions rather than direct references to the global variables.

The following is an assembly language example.

```
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main
_main:
   M8C_EnableGInt                  ; enable global interrupts
   mov   A,DELSIG8_HIGHPOWER       ; Establish power setting...
   call  DELSIG8_Start             ; and initialize
   call  DELSIG8_StartAD           ; Commence sampling process
mainloop:
   call  DELSIG8_fIsDataAvailable  ; Retrieve the status byte
   cmp   A, 0
   jz    mainloop                  ; spin lock until(data is Available)
   call  DELSIG8_cGetDataClearFlag ; fastcall convention places data in A
   call  ProcessSample             ; pass the sample in A to the dummy fcn
   jmp   mainloop


ProcessSample:
   ...                              ; (do something useful with the data)
   ret
```

Again, the equivalent code in C:

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void ProcessSample( CHAR cSample )
{
    ; // (Do something useful with the data)
}

void main(void)
{
   M8C_EnableGInt;
   DELSIG8_Start( DELSIG8_HIGHPOWER );
   DELSIG8_StartAD();
   while (1) {
      if ( DELSIG8_fIsDataAvailable() ) {
         ProcessSample( DELSIG8_cGetDataClearFlag() );
      }
   }
}
```

Example 3: Modifying the Interrupt Service Routine

The DELSIG8 ISR can easily be modified. This example shows how a simple received signal strength algorithm could be implemented without a large increase in system interrupt latency. The hardware MAC (Multiply/accumulate unit) is used to implement an RMS computation integrating over 256 samples. If the sample rate is relatively high so that there are something like 2.5 to 5 samples per cycle, we can get a fairly good estimate of the signal strength, easily within 20% even for Signal-to-Noise ratios approaching 15 dB. This is because the period of observation, 256 samples, contains enough full cycles that the partial cycle at the end contributes a relatively small amount of error.

First, near the top of the ISR in the file *DELSIG8INT.asm*, locate the banners shown here. Add two variables, a flag to enable the operation of the RMS estimator, and a counter to keep track of the number of samples. The underscores make it possible for to access the variables directly from C.

```
   ;@PSoC_UserCode_INIT@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom declarations below this banner
   ;---------------------------------------------------
   export _bfRMS_Flag         ; make the variables visible outside the ISR
   export _cRMS_Count
                              ; Clear the MAC accum to zero and set...
   _cRMS_Count:       blk 1   ; this Count to 0 before enabling RMS below
   _bfRMS_Flag:       blk 1   ; 0 to disable RMS, anything else to enable
   ;---------------------------------------------------
   ; Insert your custom declarations above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

Near the bottom of the ISR is another banner that shows where to place code that uses a new sample. Here we place the code that performs the computation. In this case we'll assume that Polling is enabled and the routine doing the polling knows whether to expect samples or RMS estimates. Note the polling code already located within the banners and enabled by IF/ENDIF conditional compiling directives.

```
   ;@PSoC_UserCode_BODY@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ; Sample data is now in the A register.
   ; Be sure to preserve the X register if you modify it!

   cmp   [_bfRMS_Flag], 0         ; is the RMS calc enabled"
   jz    .RMS_SkipComputation     ; No, just deliver the raw data
   mov   reg[MUL_X], A            ; Yes, Set up the MAC unit...
   mov   reg[MAC_Y], A            ; and add the square of the sample
   dec   [_cRMS_Count]            ; Have we accumulated 256 squares"
   jnz   .RMS_EstimateNotReady    ; No, keep going
   mov   A, reg[ACC_DR3]          ; Yes, extract the estimate
   mov   reg[MAC_CL0], 0          ; and clear the MAC for next time
.RMS_SkipComputation:
IF (DELSIG8_POLL_ENABLE)
   mov  [DELSIG8_cResult],  A                       ; Save result in cResult
   mov  [DELSIG8_bfStatus], DELSIG8_DATA_READY_BIT ; Set valid data flag
ENDIF
.RMS_EstimateNotReady:
   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

This algorithm works because each 8 by 8-bit multiply produces a 16-bit result added to the MAC accumulator. After summing 256 such values, the result is in the next-most significant byte of the 32-bit accumulator. This code adds 13 cycles to the latency when the RMS computation is disabled, and 49 cycles when enabled. The latter figure represents a 26% increase.

Configuration Registers

Table 5. Registers used by the “ADC" Analog Switched Capacitor PSoC Block

Register	7	6	5	4	3	2	1	0
CR0	1	0	0	1	0	0	0	0
CR1	InputSource			0	0	0	0	0
CR2	0	1	AZ	0	0	0	0	0
CR3	1	1	1	FSW0	0	0	0	0

The ADC is a switched capacitor PSoC block. It is configured to make an analog modulator. To build the modulator, the block is configured to be an integrator with reference feedback that converts the input value into a digital pulse stream. The input multiplexer determines what signal is digitized.

InputSource field selects the input signal digitized by the converter. This parameter is set in the Device Editor.

The AZ and FSW0 are used by the TMR interrupt handler and various APIs to reset the integrator.

Table 6. Registers used by the TMR Digital PSoC Block

Register	7	6	5	4	3	2	1	0
Function	0	0	1	0	0	0	0	0
Input	0	0	0	1	Clock			
Output	0	0	0	0	0	0	0	0
DR0	Timer Down Count Value (Never Accessed by the API)							
DR1	1	1	1	1	1	1	1	1
DR2	Not Used							
CR0	0	0	1	0	0	0	0	Enable

The TMR is a digital PSoC block configured to have a timer with a period of 256 counts. At the interrupt, the decimator is read and the ADC value is calculated.

Clock selects the input clock from one of 16 sources. This parameter is set in the Device Editor. Note that the source chosen must also be used to control the analog clock for the column in with the ADC block resides.

Enable empowers the TMR when set. It is modified and controlled by the DELSIG8 API.

Table 7. Decimation Control Registers

Bit	7	6	5	4	3	2	1	0
DEC_CR	0	0	1	0	0	0	DCol	DCLKSEL
DEC_DH	High Byte Output of Decimator							
DEC_DL	Low Byte Output of Decimator							

The decimator is dedicated hardware used to implement a Sinc2 filter needed for a ΔΣADC. It consists of a control register and two data output registers. When the value in DR0 counts down to terminal count, an interrupt is called to decrement a higher value software counter and CNT reloads from DR1. The data is output through DR2.

DCol selects which column comparator is connected. DCLKSEL selects which digital block is used to control the decimator timing. Both parameters are set in Device Editor.

Version History

Version	Originator	Description
3.2	DHA	Added a design rule check (DRC) that produces a warning if: The source clock is different in digital and analog resources or the ADC Clock is higher than the CPU Clock.
3.2.b	MYKZ	Added design rules check for the situation when the ADC clock is faster than 8 MHz.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2002-2015 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.