AN61102

PSoC 3 and PSoC 5LP - ADC Data Buffering Using DMA

Author: Anu M D, Anup Mohan

Associated Project: Yes

Associated Part Family: All PSoC® 3 and PSoC 5LP parts

Software Version: PSoC Creator™ 4.2

Related Application Notes: AN52705 and AN84810

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the Cypress video training library here.

AN61102 describes how to configure the direct memory access (DMA) to buffer the analog-to-digital converter (ADC) data. It discusses how to overcome some of the limitations of the DMA when buffering the ADC data.

Contents

1 Introduction

The DMA controller in PSoC® 3 and PSoC 5LP is used to handle data transfer without CPU intervention. This is useful in applications that require ADC data buffering and allows the CPU to do simultaneous tasks.

This application note explains the basics of 8-bit, 16-bit, and 20-bit Delta Sigma ADC data buffering using DMA with example projects. The 20-bit example project accompanying this application note demonstrates problems with data buffering using DMA. These problems occur when the peripheral's spoke width is less than the actual data width. The project describes how to tackle this using multiple DMA channels. The application note also includes an example project on 12-bit SAR ADC data buffering for PSoC 5LP device.

This document assumes that you know how to create designs for PSoC 3 and PSoC 5LP using PSoC Creator. If you are new to these products, you can find introductions in AN54181 – Getting Started with PSoC 3 and AN77759 – Getting Started with PSoC 5LP. If you are new to PSoC Creator, see the PSoC Creator home page. You should also be familiar with the topics discussed in the basic DMA application note, AN52705 – Getting Started with DMA. For an advanced DMA application, see AN84810 – PSoC® 3 and PSoC 5LP Advanced DMA Topics.

2 Basic Concepts

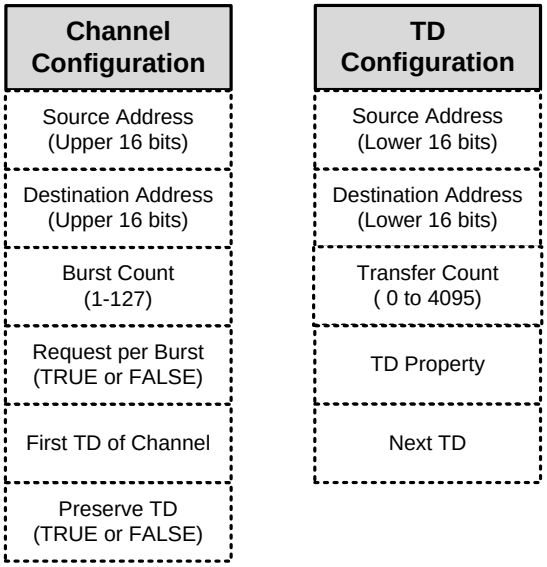
The DMA is used to move data from a source to destination without CPU intervention. The DMA controller (DMAC) controls these data transfers. Some basic concepts of the ADC and the DMA used in this document are as follows:

- **PHUB**: The Peripheral HUB (PHUB) is the central hub that has data buses that connects various on-chip peripherals and memory. The DMAC resides within the PHUB.
- **Spoke**: Spokes are data buses that branch out from the PHUB to various peripherals. Each spoke is connected to one or more peripherals. The spoke data bus width can be either 16 or 32 bits. See *PSoC® 3, PSoC® 5 Technical Reference Manual* for more details. The data width of the spoke attached to Delta sigma ADC in PSoC 3 / PSoC 5LP is 16-bit.
- **Channel**: Channel resides in the DMA controller. Channels use the PHUB to transfer data. The channels fetch the transaction descriptors, access the PHUB spoke for the source and the destination, and transfer data.
- **Transaction Descriptor (TD)**: The TD stores all information required for the data transfer. The information stored in the TD includes the source and the destination addresses, the number of bytes to transfer and other properties of the transfer.
- **drq**: This refers to the data request terminal of the DMA. This terminal becomes visible when you enable the hardware request for the DMA channel.
- The DMA channel is triggered by the signal given to the hardware request terminal. The ADC generates an End of Conversion (EoC) signal at the end of each conversion and this can be used as the DMA channel trigger to buffer the ADC data.
- **nrq**: This output terminal of the DMA component can be used to monitor whether the DMA transfer is complete. The TD should be configured (using APIs) to generate appropriate termout signal on the nrq line when the transfer is complete. If the TD is configured to generate termout, a pulse signal (two bus clock cycles wide) appears on the nrq line once the TD transfer is complete. The following sections describe some additional concepts.

3 DMA Configuration

The various parameters of a DMA transfer are configured using the channel configuration and the TD configuration registers as shown in Figure 1 described in the subsequent sections.

Figure 1. DMA Configuration



4 Channel Configuration

The source addresses and the destination addresses in PSoC 3 and PSoC 5LP are 32 bits wide. The upper 16-bits are configured in channel configuration registers and the lower 16-bits are configured in TD configuration registers.

- **Upper Source Address (16-bits)**
  Upper 16-bits of 32-bit source address configured in channel configuration registers.
- **Upper Destination Address (16-bits)**
  Upper 16-bits of 32-bit destination address configured in channel configuration registers
- **Lower Source Address (16-bit)**
  Lower 16-bits of 32-bit source address configured in TD configuration registers
- **Lower Destination Address (16-bit)**
  Lower 16-bits of 32-bit destination address configured in TD configuration registers
- **Burst Count (1 to 127)**
  Number of bytes the DMA channel must move from the source to the destination before it releases the spoke. The DMAC acquires the spoke for each burst data movement, moves (copies) the specified number of bytes from the source to the destination (configured in burst count parameter of channel configuration registers) and then releases the spoke. It re-acquires the spoke during the next burst transfer.
- **Request Per Burst(0 or 1)**
  When multiple bursts are required to finish the DMA data transfer, this parameter determines the nature of the bursts.
  0: All subsequent bursts after the first burst are automatically done without a separate request. (Only the first burst transfer must have a DMA request.)
  1: All subsequent bursts after the first burst must have individual requests.
- **Initial TD**
  The channel collects information from the first TD pointer and subsequent TD pointers and keeps it in the TD itself, similar to a linked list. The pointer to the first TD is stored in channel configuration memory and subsequent TD pointers are stored in TD configuration memory, similar to a linked list.
- **Preserve TD(0 or 1)**
  Defines whether to use TD configuration registers or separate the PHUB working registers to store intermediate TD states.
  0: Store the intermediate states on top of the original TD chain (TD configuration registers).
  1: Store the intermediate states separately in a working register to keep the original TD configuration.
  Typically, TD configurations are preserved so that TD can be repeated.

4.1 TD Configuration

- **Transfer Count(0 to 4095)**
  The total number of bytes to be moved from the source to the destination.
  For example, if you want to move 100 bytes of the data from a 16-bit peripheral to a memory buffer, the burst count is set to 2 and transfer count is set to 100.
- **Next TD**
  Points to the next TD, similar to a linked list
- **TD Property (Configurable from the list below)**

Increment Source Address

Increases source address after each burst transfer.

Increment Destination Address

Increments destination address after each burst transfer.

Swap Enable

The PSoC 3 Keil Compiler uses big endian format to store 16-bit and 32-bit variables. However, PSoC 3 peripheral registers uses little endian format. A byte swap on 2-byte or 4-byte words must occur to move the data between array and peripheral registers. For this reason, the DMA must be configured to swap bytes while it moves the data between the peripheral registers and the memory in PSoC 3.
If this TD property is set, DMA swaps the data bytes while it moves the data from the source to the destination.

Swap Size: Used with the **Swap Enable** setting.

0: Swap size is 2 bytes. Every 2 bytes are endian swapped during the DMA transfer.
1: Swap size is 4 bytes. Every 4 bytes are endian swapped during the DMA transfer.

Auto Execute Next TD

0: The next TD in the chain will be executed only after the next DMA request.
1: The next TD in the chain is automatically executed soon after the current TD transfer is finished.

DMA Completion Event

A DMA “done signal” is generated after the data transfer is finished. This is typically used to create an interrupt after the transfer is finished.

4.2 Delta-Sigma ADC Output

The Delta-Sigma ADC has programmable resolutions from 8-bits to 20-bits. The Delta-Sigma ADC output is available in 32-bit format consisting of four 8-bit registers: OUTSAMP, OUTSAMPM, OUTSAMPH, and OUTSAMPS registers. The OUTSAMPS register gives sign extension of the data if OUTSAMPH is read as a 16-bit register.

In the default ADC configuration, the output is aligned to the least significant bit (LSB). Therefore, for an ‘n’ bit resolution, the ADC result is always available in the least ‘n’ bits starting from OUTSAMP.

Figure 2. 8-Bit ADC Result

OUTSAMPS (Sign Ext -0x4e13)	**OUTSAMPH (0x4e12)**	**OUTSAMPM (0x4e11)**	**OUTSAMP (Addr: 0x4e10)**

Figure 3. 16-Bit ADC Result

OUTSAMPS (Sign Ext -0x4e13)	**OUTSAMPH (0x4e12)**	**OUTSAMPM (0x4e11)**	**OUTSAMP (Addr: 0x4e10)**

Figure 4. 20-Bit ADC Result

OUTSAMPS (Sign Ext -0x4e13)	**OUTSAMPH (0x4e12)**	**OUTSAMPM (0x4e11)**	**OUTSAMP (Addr: 0x4e10)**

For SAR ADCs, the 12-bit output data is available in the registers ‘SAR[0,1]. WRK0’ and ‘SAR[0,1]. WRK1’.

Figure 5. 12-Bit SAR ADC Result

SAR[0,1].WRK1		**SAR[0,1].WRK0**

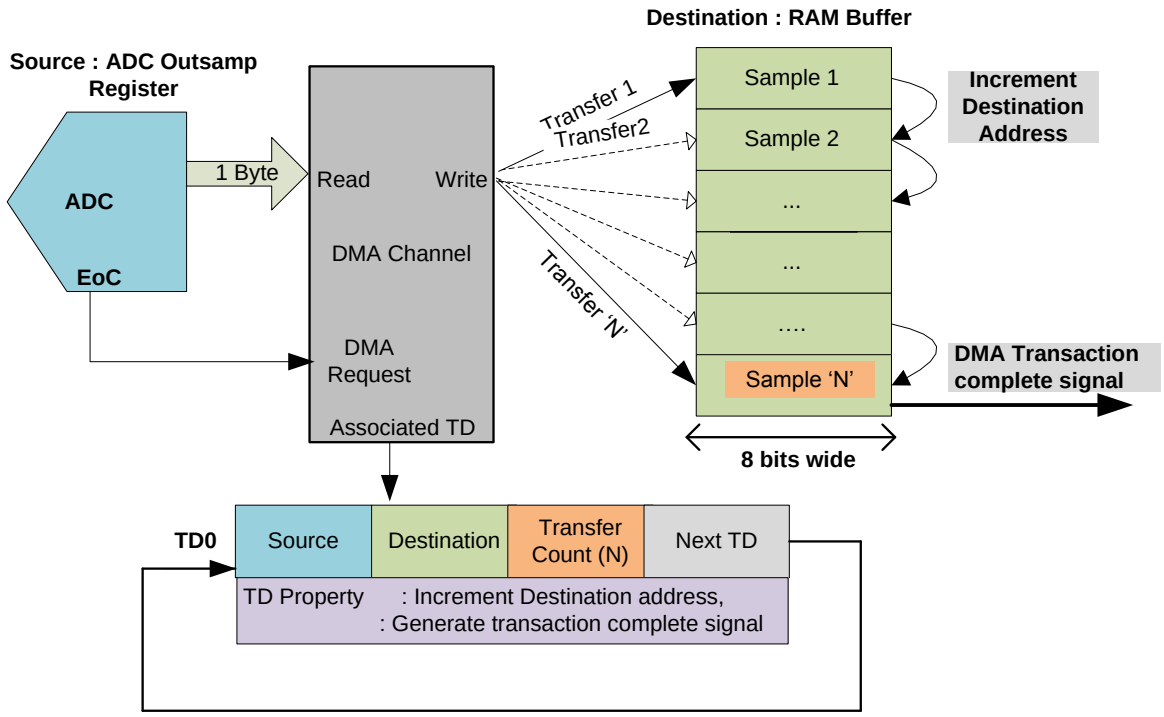
4.3 Delta Sigma ADC Coherency Key

The ADC result registers are protected on reads so that the underlying hardware does not update it when partially read by CPU or DMA. Since the ADC lies on a 16-bit spoke it will only support 8-bit or 16-bit read operation of these registers. When CPU/DMA is reading a 32-bit ADC result (4 bytes) as a multiple byte (16 bit) read operation, it is possible for the ADC to overwrite the result register with a new sample while the CPU or DMA is reading the current sample. To avoid this problem, the ADC module allows the user to specify the coherency byte using DEC_COHER[SAMP_KEY<1:0>] bits. If any byte of ADC result register is read by CPU or DMA, it will lock the result register from being overwritten until the coherency byte is read. Depending on the configuration of the block, not all bytes of the result registers may be needed. The coherency methodology allows for any size output field and handles it properly.

5 8-Bit ADC Data Buffering Using DMA

For 8-bit ADC data buffering, the contents of OUTSAMP register should be moved to memory buffer on each EoC. The DMA is triggered using EoC signal from ADC. Figure 6 shows the block diagram illustrating 8-bit transfer.

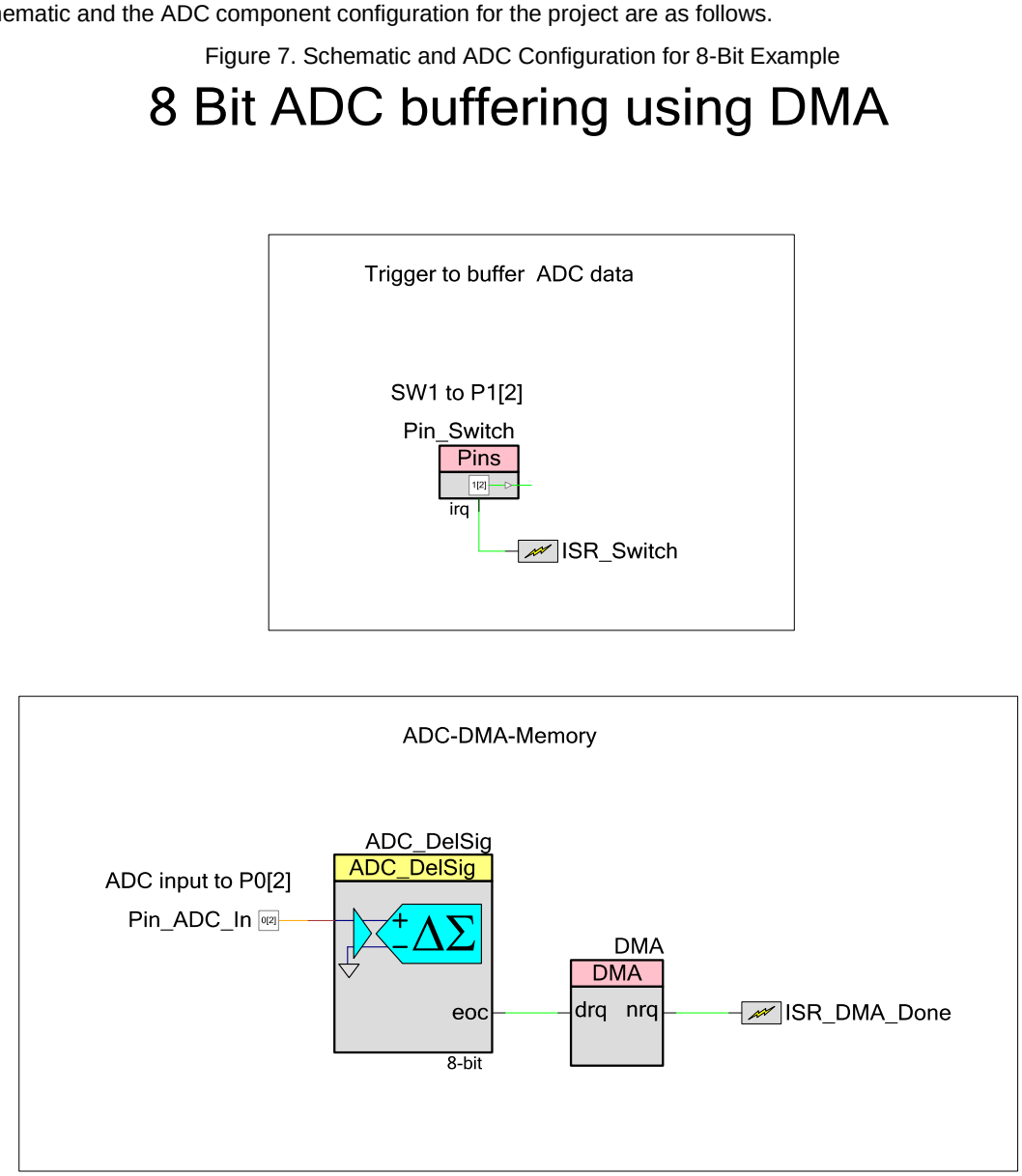
Figure 6. Block Diagram Demonstrating 8-bit Transfer

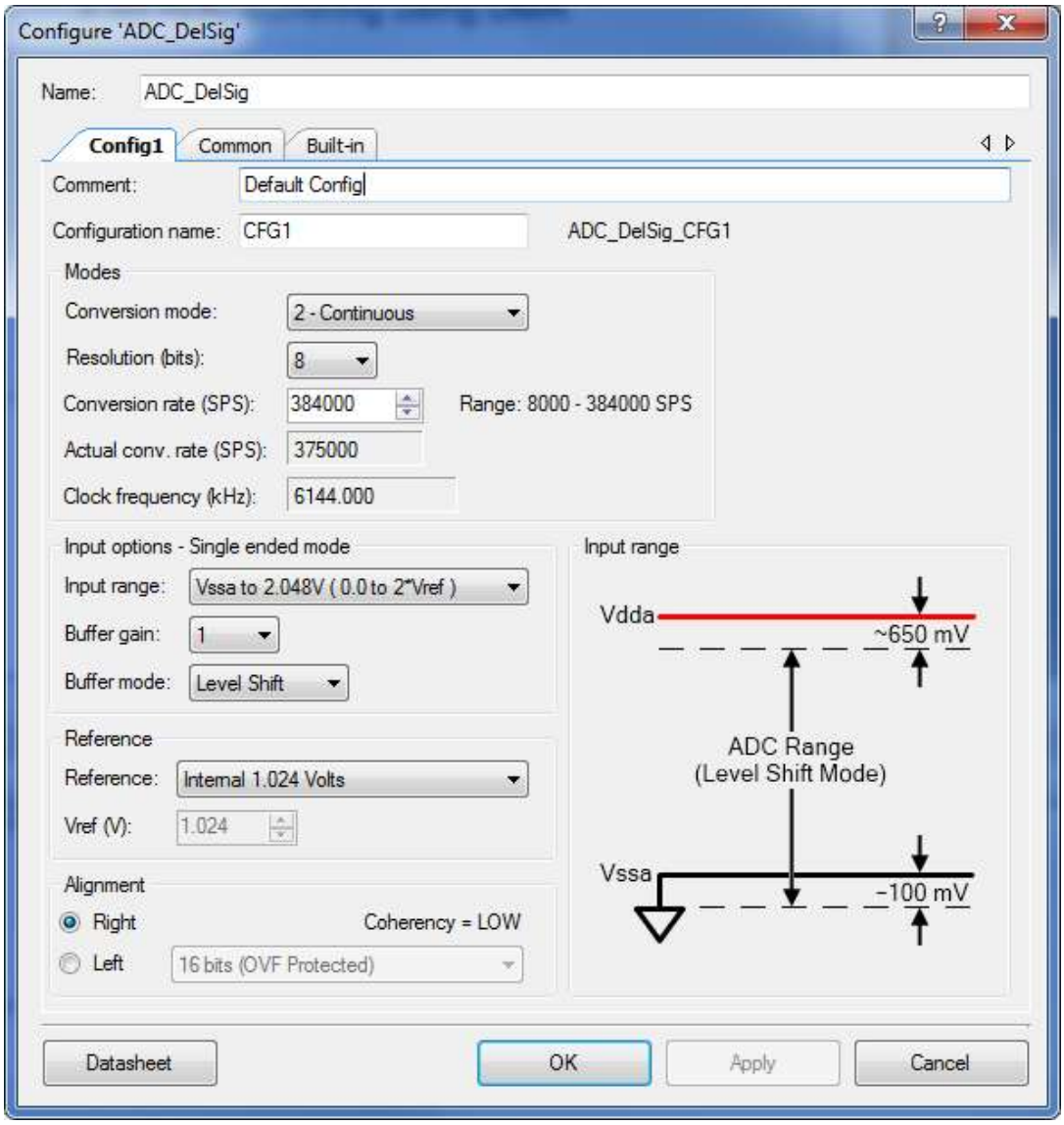


5.1 Example Project

The schematic and the ADC component configuration for the project are as follows.

Figure 7. Schematic and ADC Configuration for 8-Bit Example





The hardware request of the DMA channel component is set to rising edge. The hardware request terminal of the DMA channel is connected to the ADC EoC signal so that the DMA channel is requested whenever the ADC result is available. The DMA is enabled on a switch press. Once enabled the DMA channel moves the 16-bit ADC data to memory on every EoC event. Once the DMA buffers the required number of ADC samples, a DMA Transaction Complete signal is generated on the nrq terminal. This signal activates an ISR that disables the DMA channel.

Table 1. Channel Configuration

Parameter	Project Setting
Upper Source Address	`HI16(CYDEV_PERIPH_BASE)`
Upper Destination Address	`HI16(CYDEV_SRAM_BASE)`
Burst Count	1 Byte
Request Per Burst	True (1)
Initial TD	`DMA_TD[0]`
Preserve TD	Yes(1)

The upper 16 bits of the source address is set to `HI16(CYDEV_PERIPH_BASE)`.'CYDEV_PERIPH_BASE' is defined in the header file *cydevice.h* that is generated by PSoC Creator. This gives the 32-bit base address for all PSoC 3 and PSoC 5LP peripherals including ADC. The HI16 macro gives the upper 16 bits of this 32-bit address.

The upper 16 bits of RAM variables are given by the macro `HI16(CYDEV_SRAM_BASE)`, where `CYDEV_SRAM_BASE` is the SRAM base address defined in *`cydevice.h`*.

In this example, the 8-bit ADC result must be moved from the ADC to memory on each DMA request. For this reason, the burst count is set to 1 and the request per burst is set to true.

The 'Preserve TD' parameter of the channel is set to '1' to preserve the original source and the destination address. This is done so that after N samples are buffered (TD transfer is complete), the source address, the destination address, and the transfer count are automatically reloaded with the initial values and the TD can be repeated again.

Table 2. TD Configuration

Parameter	Project Setting
Lower Source Address	`LO16 (ADC_DEC_OUTSAMP_PTR)`
Lower Destination Address	`LO16 (ADC_sample)`
Transfer Count	N×1 (No. of samples × Bytes per sample)
TD property	Increment Destination Address Generate DMA done event
Next TD	None/repeat to same TD

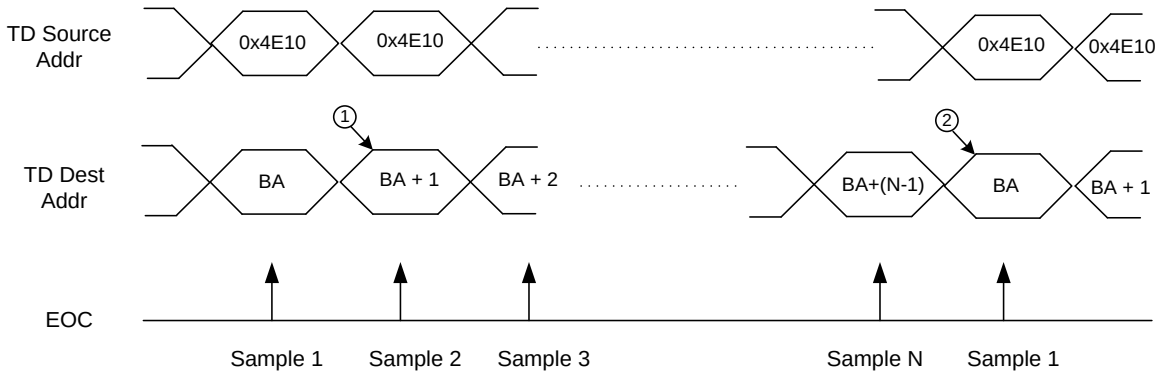
The lower 16 bits of the source and the destination address are identified by the LO16 macro. The destination is the 16-bit RAM array "ADC_sample".

The transfer count identifies the total number of bytes to be moved from the source to the destination to finish the transaction. This is set to 'Number of samples × Bytes per Sample' (N).

The TD property (TD_INC_DST_PTR) is set to increment the destination address after each burst transfer. The TD is also defined to generate a transaction complete signal (DMA__TD_TERMOUT_EN) after the specified number of bytes is moved from the ADC to buffer.

The timing diagram for the 8-bit transfer is as shown in Figure 8.

Figure 8. Timing Diagram for 8-Bit Transfer



BA : Base Address of Memory buffer
N : Number of samples

① Destination Address incremented after each transaction

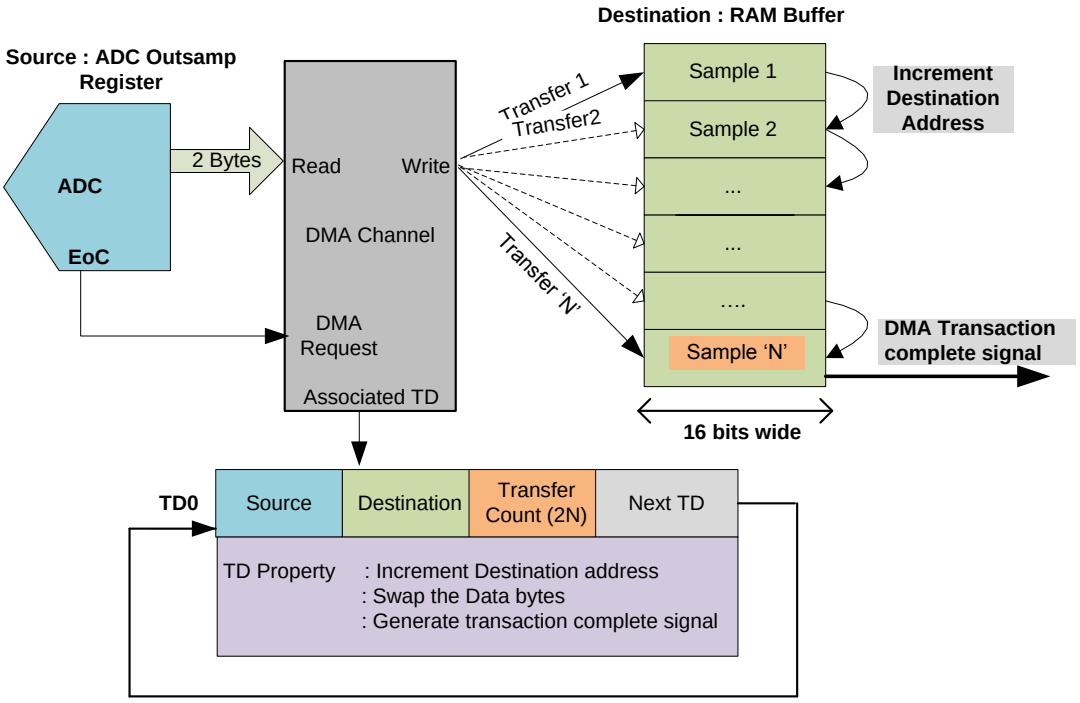
② Source and destination addresses are automatically reset to the base address after N samples are buffered

See Example Projects: Operation and Test Procedure.

6 16-Bit ADC Data Buffering Using DMA

In a 16-bit ADC configuration, the contents of OUTSAMP and OUTSAMPM registers must be buffered using the DMA. Hence, the burst count of the DMA channel is set to '2' and the TD transfer count is set to '2 × total number of samples to be buffered'. Figure 9 shows the block diagram illustrating 16-bit transfer.

Figure 9. Block Diagram Demonstrating 16-bit Transfer

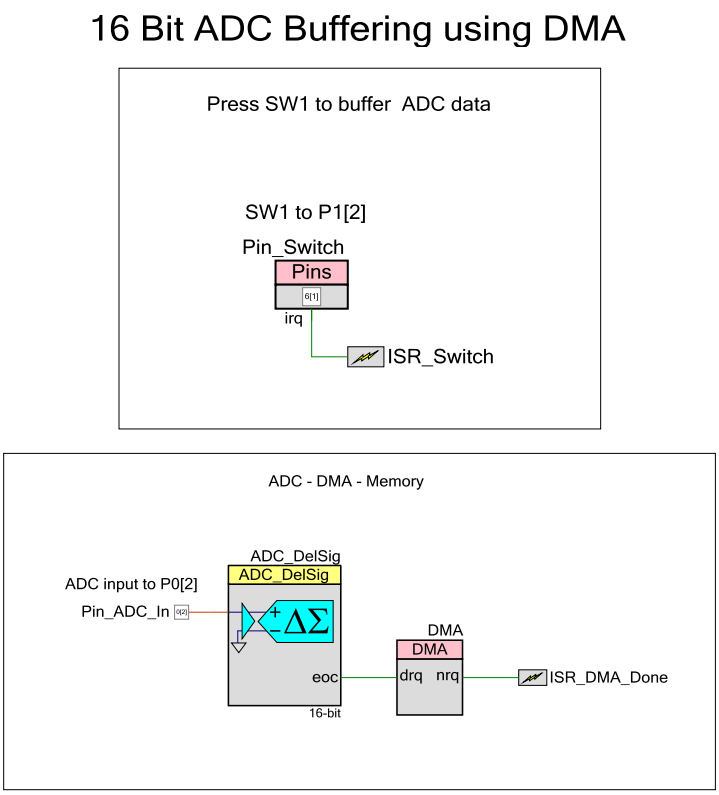


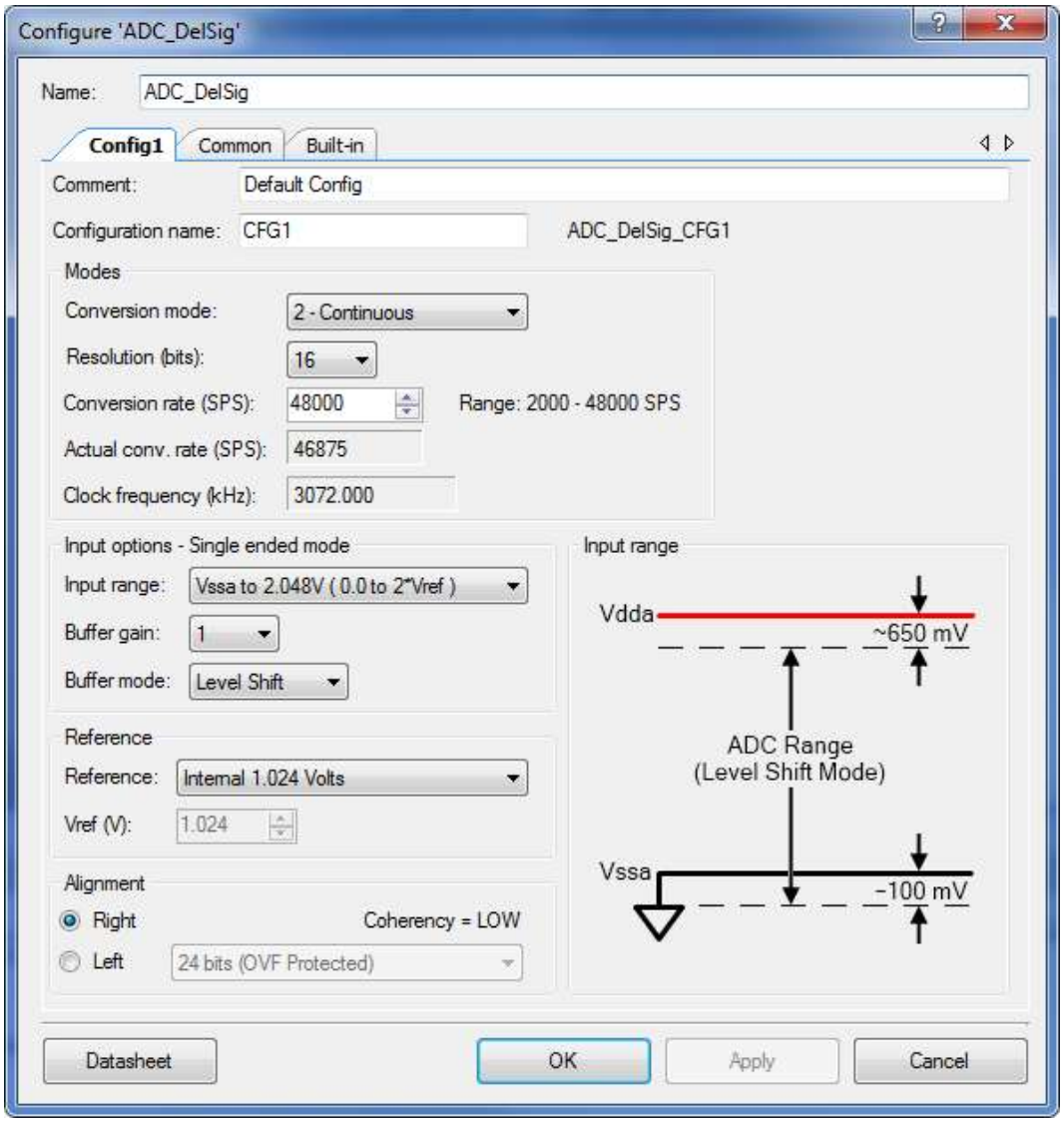
As the previous figure shows, the DMA must move the ADC result (2 bytes) from the source ADC to the destination RAM buffer each time it receives a request. The RAM buffer pointer must be increased after each data movement to point to the next sample location. After the specific number of ADC samples is collected, the DMA must send a signal that the transaction is finished.

6.1 Example Project

The schematic and the ADC component configuration for the project are as follows.

Figure 10. Schematic and ADC Configuration for 16-Bit Example





The hardware request of the DMA channel component is set to rising edge. The hardware request terminal of the DMA channel is connected to the ADC EoC signal so that the DMA channel is requested whenever the ADC result is available. The DMA is enabled on a switch press. Once enabled the DMA channel moves the 16-bit ADC data to memory on every EoC event. Once the DMA buffers the required number of the ADC samples, a DMA Transaction Complete signal is generated on the nrq terminal. This signal activates an ISR that disables the DMA channel.

6.2 Channel Configuration

Table 3. Channel Configuration

Parameter	Project Setting
Upper Source Address	`HI16(CYDEV_PERIPH_BASE)`
Upper Destination Address	`HI16(CYDEV_SRAM_BASE)`
Burst Count	2 Bytes
Request Per Burst	True (1)
Initial TD	`DMA_TD[0]`
Preserve TD	Yes(1)

The upper 16 bits of the source address is set to `HI16(CYDEV_PERIPH_BASE)`.

'`CYDEV_PERIPH_BASE`' is defined in the header file *`cydevice.h`* that is created generated by PSoC Creator. This gives the 32-bit base address for all PSoC 3 and PSoC 5LP peripherals including the ADC. The HI16 macro gives the upper 16-bits of this 32-bit address.

The upper 16-bits of RAM variables are given by the macro `HI16(CYDEV_SRAM_BASE)`, where `CYDEV_SRAM_BASE` is the SRAM base address defined in *`cydevice.h`*.

In this example, a 2-byte ADC result must be moved from ADC to memory on each DMA request. For this reason, the burst count is set to 2 and the request per burst is set to true.

The 'Preserve TD' parameter of the channel is set to '1' to preserve the original source and the destination address. This is done so that after N samples are buffered (TD transfer is complete), the source address, the destination address, and the transfer count are automatically reloaded with the initial values and the TD can be repeated again.

6.3 TD Configuration

Table 4. TD Configuration

Parameter	Project Setting
Lower Source Address	`LO16 (ADC_DEC_OUTSAMP_PTR)`
Lower Destination Address	`LO16 (ADC_sample)`
Transfer Count	N×2 (No. of samples × Bytes per sample)
TD property	Increment Destination Address Generate DMA done event Swap Enable required for PSoC3
Next TD	None/repeat to same TD

The lower 16 bits of the source and the destination address are identified by the LO16 macro. The destination is the 16-bit RAM array "`ADC_sample`".

The transfer count identifies the total number of bytes to be moved from the source to the destination to finish the transaction. This is set to 'Number of samples × Bytes per Sample' (2N).

The TD property (`TD_INC_DST_PTR`) is set to increment the destination address and the RAM buffer pointer after each burst transfer. The TD is also defined to generate a transaction complete signal (`DMA__TD_TERMOUT_EN`) after the specified number of bytes is moved from the ADC to buffer.

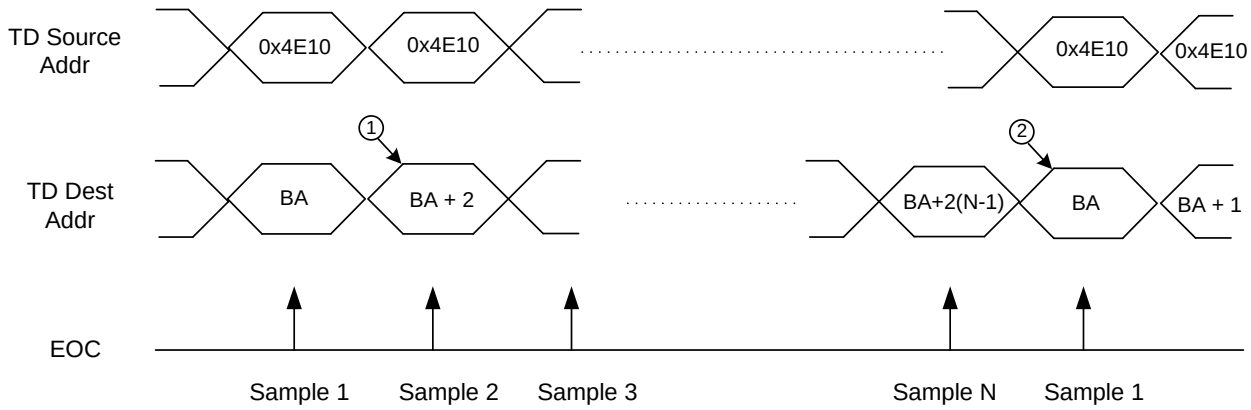
For 16-bit and 20-bit ADC data buffering, the data bytes should be swapped while moving data from the ADC to memory in PSoC 3. This is explained in the following section.

6.4 Endian Format

With PSoC 3, the Keil 8051 compiler uses big endian format for 16- and 32-bit variables. The PSoC 5LP device uses little endian format for multibyte values. All PSoC 3 and PSoC 5LP peripheral registers including the ADC, store data in little endian format. Therefore, the data byte should be swapped while moving multibyte data from the ADC to memory in PSoC 3. DMA transaction descriptors can be programmed to have bytes swapped while transferring data. The swap size should be set to 2 bytes for 16-bit transfers or 4 bytes for 32-bit transfers. `TD_SWAP_EN` configuration of the TD is used to swap the bytes while moving data using DMA; the default swap size is 2 bytes.

The timing diagram for a 16-bit transfer is as follows.

Figure 11. Timing Diagram for 16-Bit Transfer



BA : Base Address of Memory buffer
N : Number of samples

① Destination address incremented after each transaction

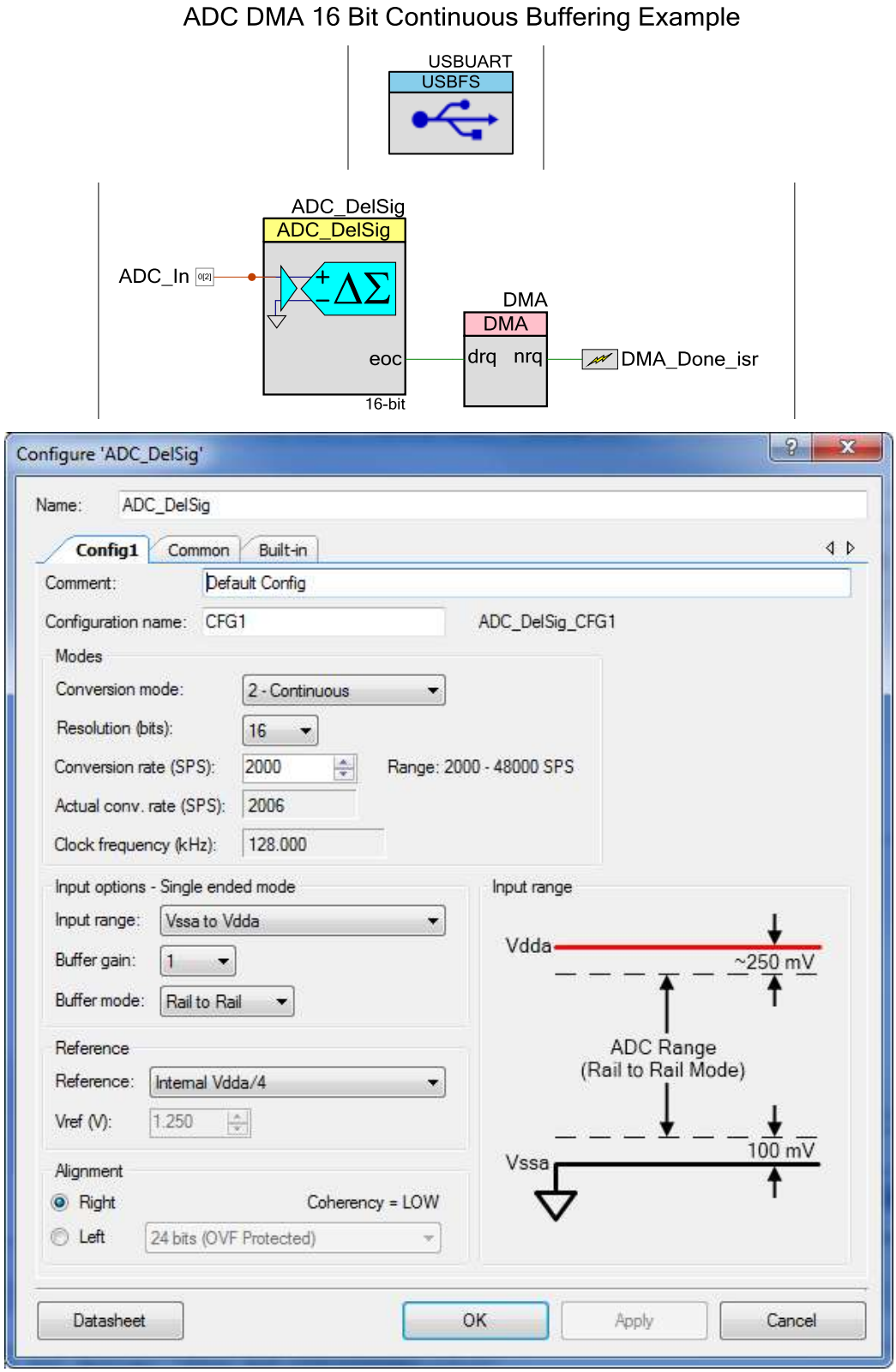
② Source and destination addresses are automatically reset to the base address after N samples are buffered

Preserve TD parameter is set to 1 to automatically re-initialize the TD after the transfer is complete

6.5 16-Bit ADC Continuous Data Buffering Using DMA Example

The continuous buffering example shows how to transfer the data buffered using DMA continuously to a PC using the USBUART of the PSoC. This project is created by making minor modifications to the above example. The data is transferred in blocks of 64 bytes to the PC. Refer to the USBFS Component Datasheet for more information on USBUART functionality.

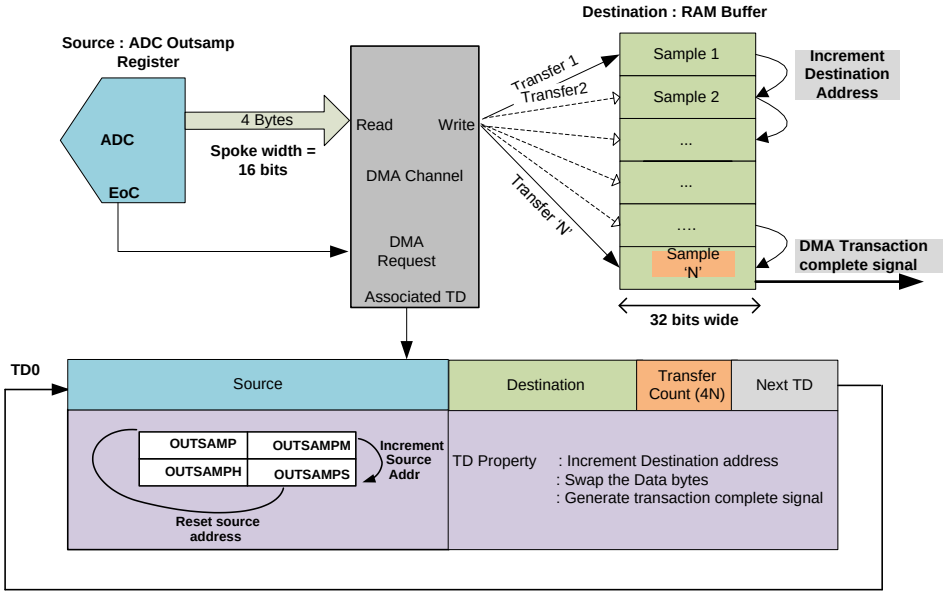
Figure 12. Schematic and ADC Configuration for 16-Bit Continuous Data Buffering Example



7 20-Bit ADC Data Buffering Using DMA

The block diagram illustrating 20-bit transfer is as follows.

Figure 13. Block Diagram Demonstrating 20-bit Transfer



In this project, DMA reads the 32-bit ADC result register on every EoC event and stores the “N” samples in a buffer.

The ADC module is mapped onto a 16-bit spoke and hence the CPU or DMA can only read a maximum of 16-bit data at a time. To read 32-bit ADC result register, the DMA must perform two read operation with address increment and the address must be reset after second read operation to read the ADC result register again in the next EoC event.

The number of bytes to be read on every end of conversion is 4 bytes and hence the burst size should be set to 4 bytes.

Resetting the source register after reading the 4-byte ADC result requires followings:

1. The transfer count must be set to 4 bytes. This way the DMA channel is forced to move to next TD after reading 4 bytes.
2. The TD must be looped to itself and the DMA channel must be configured to preserve the TD to retain the original source, the destination address and the transfer count to read the 4-byte ADC result register on the following EoC event.

The DMA destination address must be incremented to store the ADC samples in consecutive locations for every conversion. Note that the destination register also is reset after reading the 4-byte ADC result register and therefore, this DMA configuration reads the 4-byte ADC result register and writes to same memory locations on every EoC event.

If we want to buffer “N” samples of ADC in memory buffer, then it is impossible to directly buffer the samples due to this limitation.

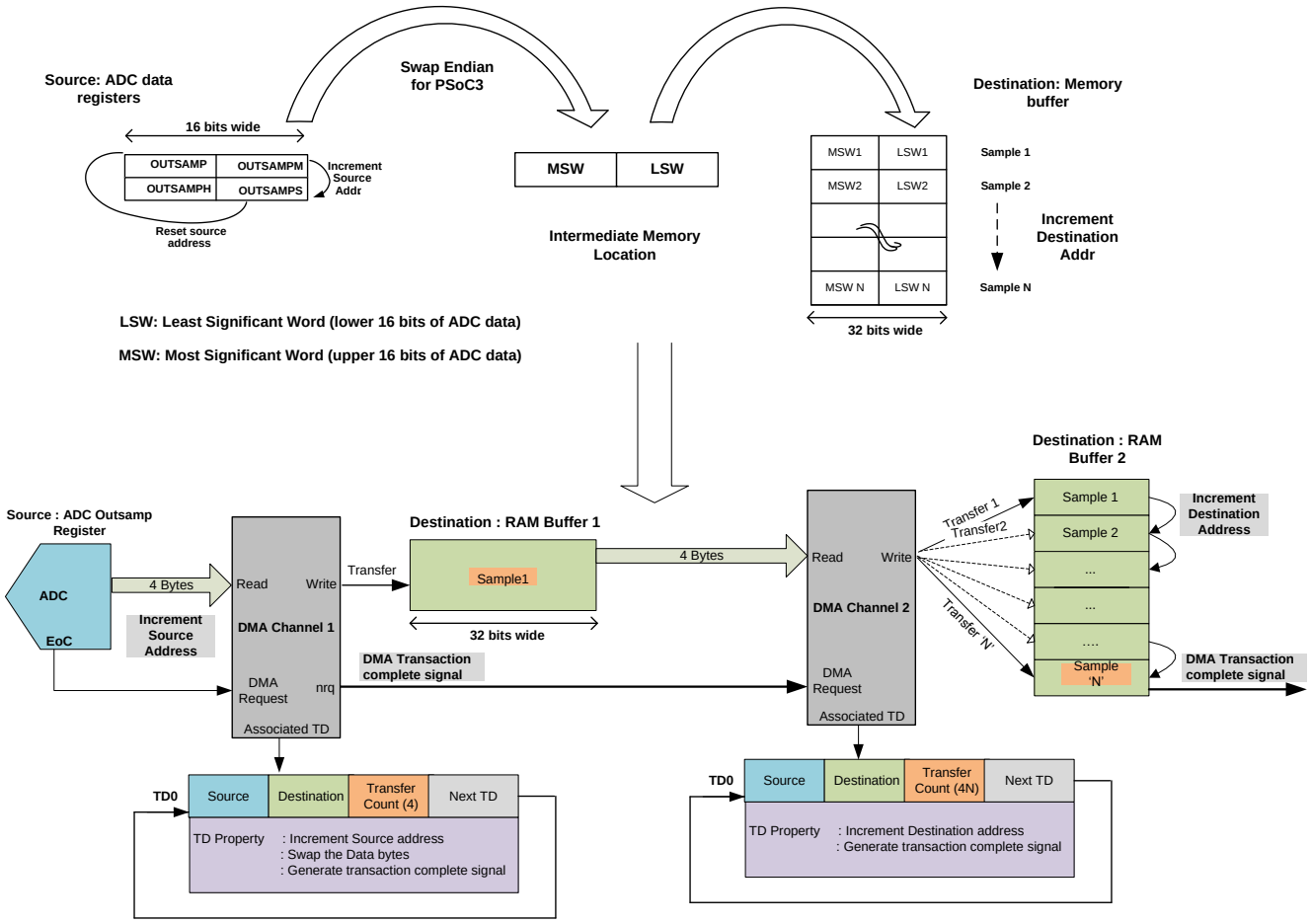
The simplest way to tackle this problem is to move the ADC data to an intermediate memory location using the one DMA channel and move data from the intermediate location to the destination buffer using the second DMA channel. This is possible because the internal memory is mapped into 32-bit spoke and does not suffer the limitation of the ADC register being mapped to 16-bit spoke.

7.1 Using Intermediate Memory Location

In this method, the data is moved to an intermediate memory location using one DMA channel and from this intermediate memory location to the destination memory buffer using the second DMA channel as shown in Figure 15 and Figure 16.

The memory is mapped to a 32-bit spoke in PSoC 3 and PSoC 5LP. Therefore, the DMA can transfer 32-bit data from memory to memory in a burst which eliminates the need for incrementing the address for a 32-bit memory to memory transfer. However, in this case, the destination address needs to be incremented after each burst transfer to point to the next sample location in the destination buffer.

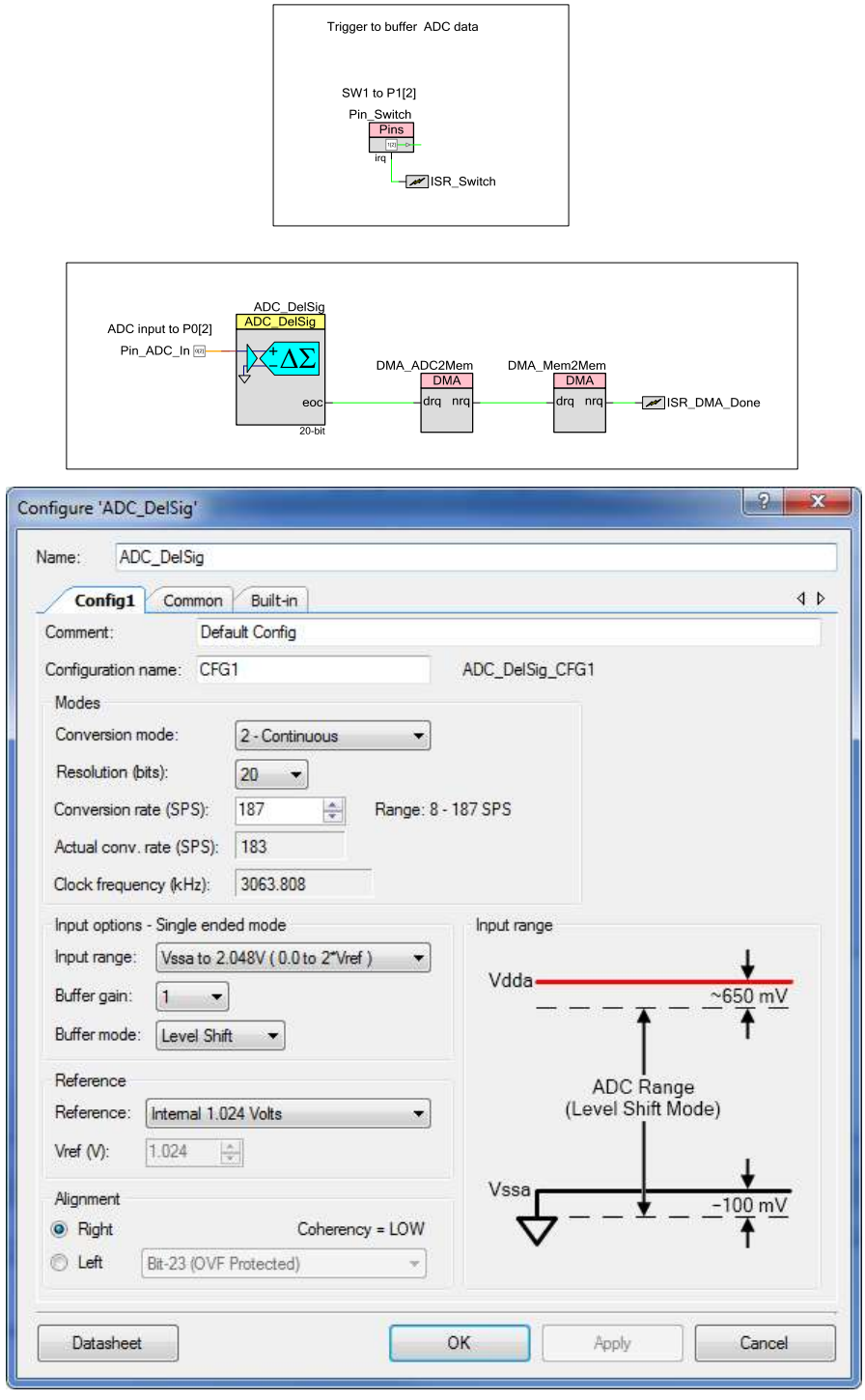
Figure 14. 20-Bit Data Buffering Using Intermediate Memory Location



7.2 Example Project

The schematic and the ADC component configuration for the project are as follows.

Figure 15. ADC_DMA_Memory_20bit Schematic and ADC Configuration



DMA collects a specified number of ADC samples (N) on a switch press. The DMA channels are enabled on each switch press and disabled after it collects the specified number of ADC samples. A DMA Transaction Complete signal on the nrq terminal on the first DMA channel triggers the second DMA channel and the nrq signal on the second DMA channel activates an ISR, which disables the DMA channels.

The DMA component instance has the name DMA_ADC2Mem. The hardware request of this DMA channel component is set to rising edge. The hardware request terminal of the DMA channel is connected to the ADC EoC signal so that the DMA channel is requested whenever the ADC result is available. The second DMA component has the instance name DMA_Mem2Mem. The hardware request of this DMA channel component is set to rising edge and the hardware request terminal is connected to the nrq terminal for the DMA_ADC2Mem DMA component.

7.3 Channel Configuration

Table 5. Channel Configuration

Channel: DMA_ADC2Mem	
Parameter	**Project Setting**
Upper Source Address	`HI16(CYDEV_PERIPH_BASE)`
Upper Destination Address	`HI16(CYDEV_SRAM_BASE)`
Burst Count	4 Bytes
Request Per Burst	True (1)
Initial TD	`DMA_ADC2Mem_TD[0]`
Preserve TD	Yes(1)

Channel: DMA_Mem2Mem	
Parameter	**Project Setting**
Upper Source Address	`HI16(CYDEV_SRAM_BASE)`
Upper Destination Address	`HI16(CYDEV_SRAM_BASE)`
Burst Count	4 Bytes
Request Per Burst	True (1)
Initial TD	`DMA_Mem2Mem_TD[0]`
Preserve TD	Yes(1)

The upper 16 bits of the source address of DMA_ADC2Mem channel is set to `HI16(CYDEV_PERIPH_BASE)`. '`CYDEV_PERIPH_BASE`' is defined in the header file *`cydevice.h`* that is created generated by PSoC Creator. This gives the 32-bit base address for all PSoC 3 and PSoC 5LP peripherals including the ADC. The HI16 macro gives the upper 16 bits of this 32-bit address.

The upper 16 bits of RAM variables are given by the macro `HI16(CYDEV_SRAM_BASE)`, where CYDEV_SRAM_BASE is the SRAM base address defined in *`cydevice.h`*.

In this example for both the DMA channels, a 4-byte data must be moved to memory on each DMA request. For this reason, the burst count is set to 4 and the request per burst is set to true.

The 'Preserve TD' parameter of the channel is set to '1' to preserve the original source and the destination address. This is done so that after N samples are buffered (TD transfer is complete), the source address, the destination address, and the transfer count are automatically reloaded with the initial values and the TD can be repeated again.

7.4 TD Configuration

Table 6. TD Configuration

DMA_ADC2Mem	
Parameter	**Project Setting**
Lower Source Address	`LO16 (ADC_DEC_OUTSAMP_PTR)`
Lower Destination Address	`LO16 (adc_temp)`
Transfer Count	N×4 (No. of samples × Bytes per sample)
TD property	Increment Source Address Generate DMA done event Swap Enable required for PSoC3 Set Swap size to 4 bytes
Next TD	None/repeat to same TD

DMA_Mem2Mem	
Parameter	**Project Setting**
Lower Source Address	`LO16 (ADC_DEC_OUTSAMP_PTR)`
Lower Destination Address	`LO16 (ADC_sample)`
Transfer Count	N×4 (No. of samples × Bytes per sample)
TD property	Increment Destination Address Generate DMA done event
Next TD	None/repeat to same TD

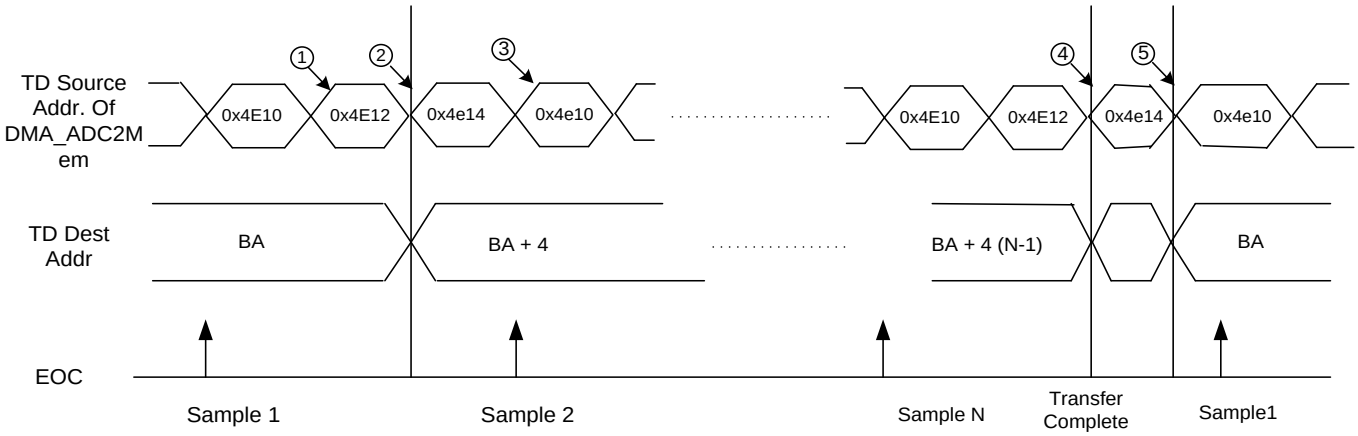
The lower 16 bits of the source and the destination address are identified by the LO16 macro. The destination for the DMA_ADC2Mem is the 32-bit RAM variable `adc_temp` and the destination for DMA_Mem2Mem is the 32-bit RAM array `ADC_sample`.

The transfer count identifies the total number of bytes to be moved from source to destination to finish the transaction. This is set to 'Number of samples × Bytes per Sample' (4N).

When the 20-bit data is moved from ADC to memory in PSoC 3, the bytes must be swapped. This is because PSoC 3 peripheral registers use little endian format and the Keil compiler uses big endian format. For more information, see the 'Endian Format' section. The `TD_SWAP_EN` and `TD_SWAP_SIZE4` configurations make the DMA able to swap 4 bytes while it moves data from peripheral to memory.

Also in this case please make sure that the intermediate variable to transfer the ADC data is on an even address boundary since DMA cannot transfer 32 bit data in one burst unless the source and destination address are aligned at the boundaries. Use compiler directives/keywords as given in the attached sample projects to make sure that the temporary location is 32-bit boundary aligned.

Figure 16. Timing Diagram for 32-Bit Transfer



Note 1. The source (ADC) is on a 16-bit spoke and hence the source address is incremented by two after fetching the first two bytes least significant to fetch the most significant word of ADC data.

Note 2. Because the TD is configured to increment source and destination addresses, the source and destination addresses continue to increment after each transaction. The destination is on a 32-bit spoke; therefore, the destination address is incremented by 4 after each burst of 4 bytes.

Note 3. On each EoC trigger, the source address register of DMA_ADC2Mem is reset back to OUTSAMP using DMA_UpdateTDAddr. The DMA_ADC2Mem then moves 32 bits from ADC to memory.

Note 4. Because the 'Preserve TD' parameter of DMA_ADC2Mem is set to zero, source address, destination address, and transfer count are NOT reinitialized by DMAC when the transfer is complete.

Note 5. The TD source address, destination address, and transfer count are re-initialized using APIs after the transfer is complete.

7.5 ADC Coherency

The ADC coherency key should be set to the last sample byte that is read by DMA. By default, the ADC component sets OUTSAMP as the coherency key byte. However, for 32-bit buffering using DMA, LSW (OUTSAMP and OUTSAMPM) is read first, followed by MSW (OUTSAMPH and OUTSAMPS). Therefore, the coherency key must be changed to OUTSAMPH, after the ADC component initialization.

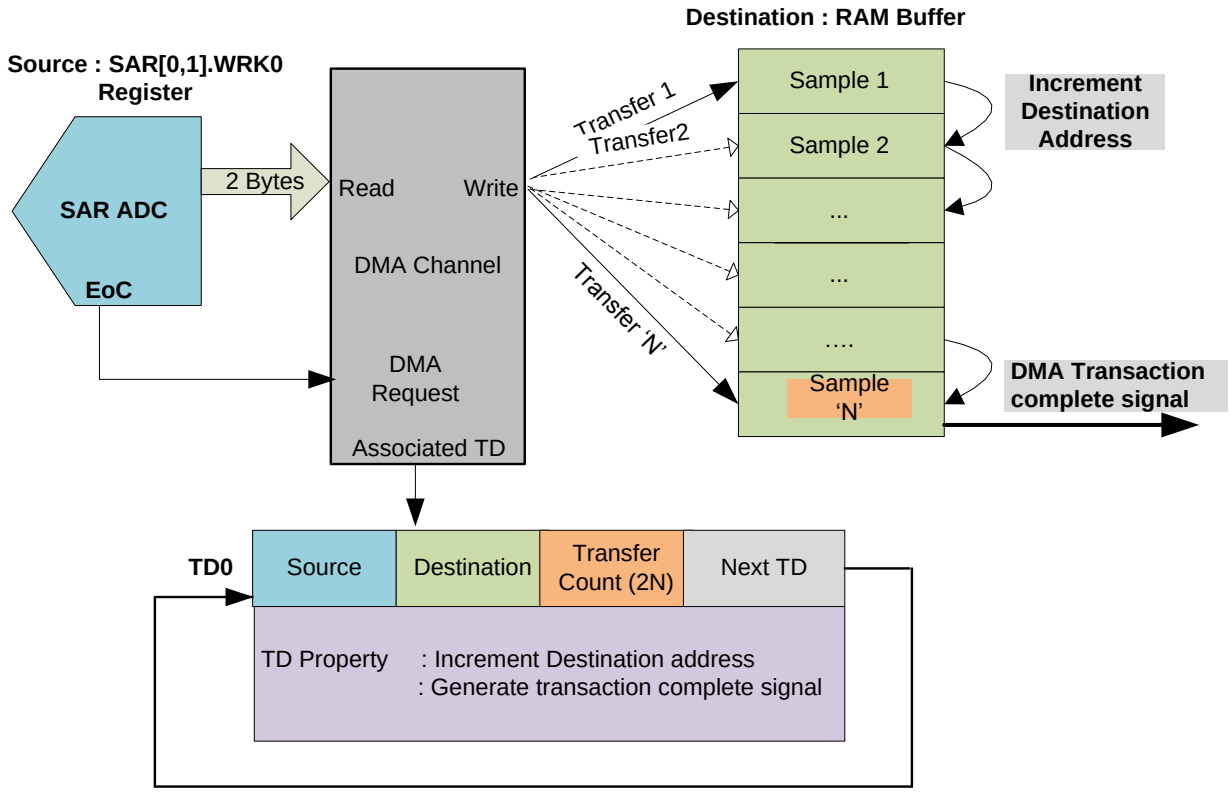
```
/*Change the ADC coherent key to high byte*/
ADC_DelSig_DEC_COHER_REG |= ADC_DelSig_DEC_SAMP_KEY_HIGH;
```

This is required because there is no option to automatically decrement TD source address to configure the DMA to read MSW before LSW.

8 12–Bit SAR ADC Data Buffering Using DMA

In addition to Delta Sigma ADC, the PSoC 5LP device has two SAR ADCs. The maximum resolution of the SAR ADC is 12 bits and the maximum conversion speed attainable is 700 ksps. The DMA configuration required to buffer the 12-bit SAR ADC data are similar to that mentioned in the 16-Bit ADC Data Buffering Using DMA section. The only difference is that instead of OUTSAMP register, the source address of DMA would be set to the SAR ADC data register namely SAR [0, 1].WRK0. The following figure shows a 12-bit SAR ADC to memory signal chain that uses DMA.

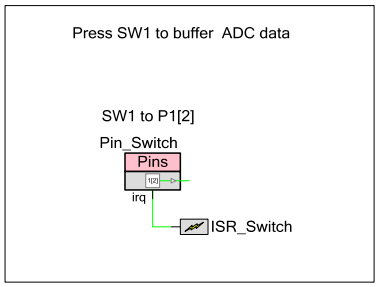
Figure 17. Block Diagram Demonstrating 12-bit Transfer

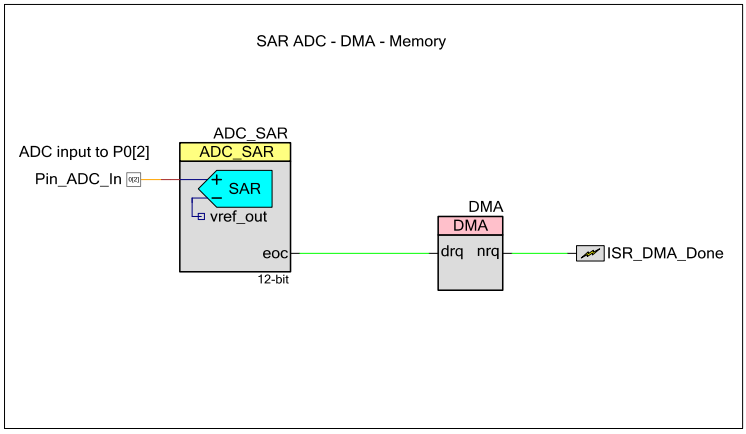


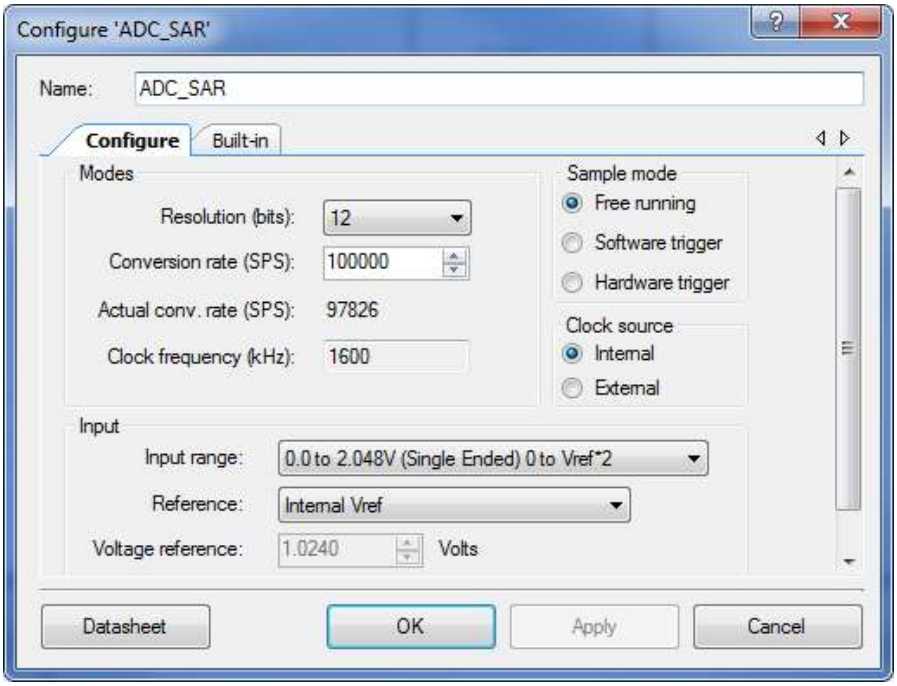
An example project named SAR_ADC_DMA_Memory is attached with this application note, which demonstrates 12-bit SAR ADC data buffering using the DMA. More details on the working and test procedure of the project can be found at the section Example Project: Operation and Test Procedure. The schematic and the SAR ADC component configuration for the project are as follows.

Figure 18. SAR_ADC_DMA_Memory Schematic and SAR ADC Configuration

12 Bit SAR ADC Buffering using DMA







DMA collects a specified number of ADC samples (N) on a switch press. The DMA channel is enabled on each switch press and disabled after it collects the specified number of ADC samples. A DMA Transaction Complete signal on the nrq terminal activates an ISR, which disables the DMA channel. The hardware request of the DMA channel component is set to rising edge. The hardware request terminal of the DMA channel is connected to the ADC EoC signal so that the DMA channel is requested whenever ADC result is available.

8.1 Channel Configuration

Table 7. Channel Configuration

Parameter	Project Setting
Upper Source Address	`HI16(CYDEV_PERIPH_BASE)`
Upper Destination Address	`HI16(CYDEV_SRAM_BASE)`
Burst Count	2 Bytes
Request Per Burst	True (1)
Initial TD	`DMA_TD[0]`
Preserve TD	Yes(1)

The upper 16-bits of the source address are set to `HI16(CYDEV_PERIPH_BASE)`. 'CYDEV_PERIPH_BASE' is defined in the header file *cydevice.h* that is created generated by PSoC Creator. This gives the 32-bit base address for all PSoC 3 and PSoC 5LP peripherals including ADC. The HI16 macro gives the upper 16-bits of this 32-bit address.

The upper 16-bits of RAM variables are given by the macro `HI16(CYDEV_SRAM_BASE)`, where `CYDEV_SRAM_BASE` is the SRAM base address defined in *cydevice.h*.

In this example, a 2-byte ADC result must be moved from the ADC to memory on each DMA request. For this reason, the burst count is set to 2 and the request per burst is set to true.

The 'Preserve TD' parameter of the channel is set to '1' to preserve the original source and the destination address. This is done so that after N samples are buffered (TD transfer is complete), the source address, the destination address, and the transfer count are automatically reloaded with the initial values and the TD can be repeated again.

8.2 TD Configuration

Table 8. TD Configuration

Parameter	Project Setting
Lower Source Address	`LO16 (ADC_SAR_SAR_WRK0_PTR)`
Lower Destination Address	`LO16 (ADC_sample)`
Transfer Count	N×2 (No. of samples × Bytes per sample)
TD property	Increment Destination Address Generate DMA done event
Next TD	None/repeat to same TD

The lower 16-bits of the source and the destination address are identified by the LO16 macro. The destination is the 16-bit RAM array `ADC_sample`. Since PSoC 5LP SRAM registers and the peripheral register store data in little endian format, byte swapping is not required.

9 Example Projects: Operation and Test Procedure

There are five example projects associated with this application note demonstrating 8-bit, 16-bit and 20-bit Delta Sigma ADC data buffering and 12-bit SAR ADC data buffering using the DMA. The projects are available in the AN61102.zip file and are arranged as follows:

- ADC_DMA_16Bit_Continuous (16 bit continuous buffering example)
- ADC_DMA_Memory_8bit (8 bit buffering example)
- ADC_DMA_Memory_16bit (16 bit buffering example)
- ADC_DMA_Memory_20bit (20 bit buffering example)
- SAR_ADC_DMA_Memory (12 bit SAR ADC buffering example)

The first example project *ADC_DMA_16Bit_Continuous* continuously buffers 16-bit ADC data using DMA. The DMA is configured to buffer the blocks of ADC data where each block has 64 ADC samples.

The last four example projects are similar except for the ADC configuration and the DMA configuration as explained in the previous sections. Therefore, the operation and test procedure for all the example projects are same.

9.1 Operation

In the first example project *ADC_DMA_16Bit_Continuous* the ADC starts conversion of the input signal and the 64 bytes of data are transferred to the internal memory. The ADC conversion is stopped after this step to allow transfer of data to the USBUART. The USB is configured for automatic DMA transfers. The ADC conversion is restarted once the data is transferred to terminal program.

In the last four example projects, the DMA is configured to buffer specified number of ADC samples on a switch press. The DMA channel is enabled on switch press and disabled after collecting the specified number of ADC samples. The DMA configuration details are available in *main.c*.

When the switch is pressed, ISR_Switch triggers and ‘switch_flag’ is set in the ISR to indicate switch press. The main loop monitors this flag and enables the DMA channels to buffer ADC data.

After buffering the specified number of ADC samples in memory, the nrq signal of the DMA channel connected to ISR_DMA_Done goes high. In *ISR_DMA_Done.c*, the ‘DMADone_flag’ is set to indicate that the DMA has completed the transfer and the main loop checks ‘DMADone_flag’ and disables the DMA channel if the flag is set.

9.2 Test Procedure

To verify the project,

For the first example project *ADC_DMA_16Bit_Continuous*

- Build and program the chip
- Provide the ADC data at pin P0[2]
- Connect USBUART com port to Bridge Control Panel (BCP).
- Set the variable name ***input*** as shown in Figure 19

Figure 19. Variable Setting in BCP

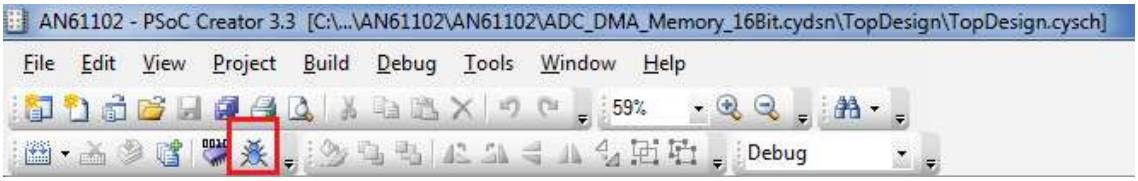
- Enter the command in ***Editor*** window as: ***rx8 @0input @1input***
- Repeat the command and verify the result in ***Chart*** window as shown in Figure 20.

Figure 20. ADC Data in Chart Window of BCP

For the rest of four example projects:

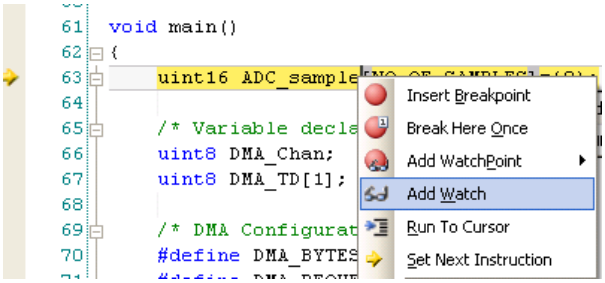
- Build and program the chip
- Press **F5** or click the **debug** icon to download the program and debug as shown in Figure 21.

Figure 21. Debug Icon in PSoC Creator



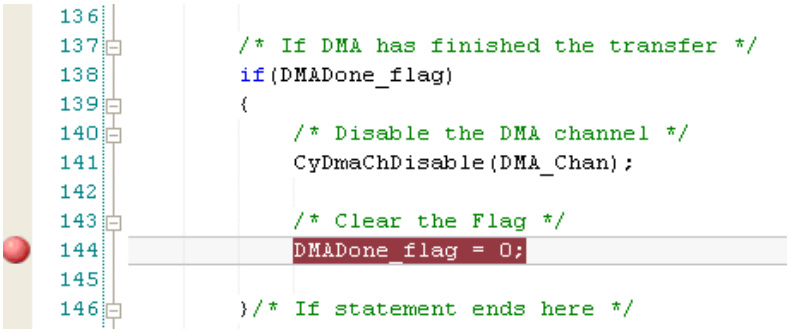
- Add ADC_sample as a watch variable as shown in Figure 22.

Figure 22. ADC_Sample Watch Variable



- Put a breakpoint inside `if(DMADone_flag)` as shown in the Figure 23.

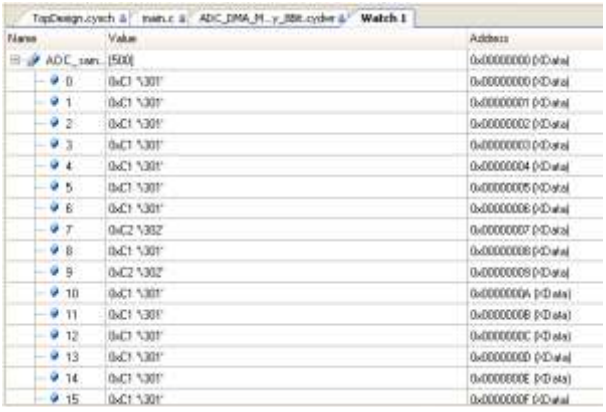
Figure 23. Breakpoint at DMADone_Flag



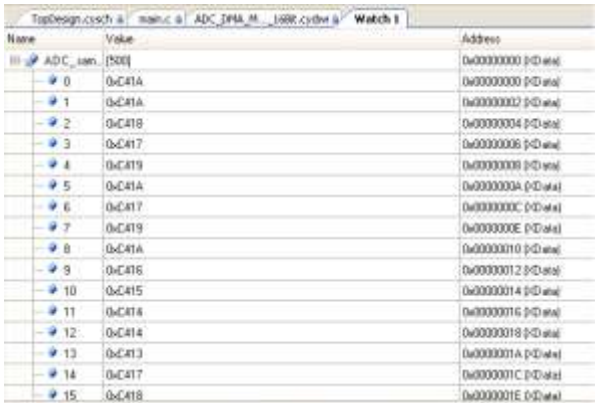
- Press **F5** to run the program. Press **SW1** connected to P1[2] to enable the DMA to start ADC sample buffering.
- The execution stops at the breakpoint after the DMA transfers the specified number of samples from ADC to memory and the result can be verified by monitoring the "`ADC_sample`" array in the watch window. A sample output for the four example projects are as shown in Figure 24.

Figure 24. Sample Output for Example Projects.

8-bit Example

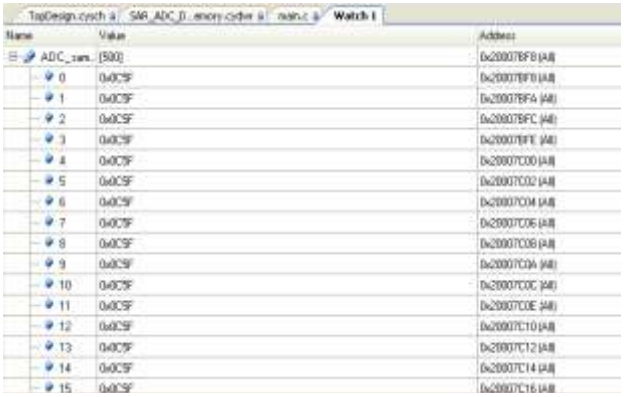


16-bit Example



20-bit Example

12-bit SAR ADC Example



10 Summary

The DMA in PSoC 3 and PSoC 5LP can buffer ADC data without any CPU intervention. Some of the limitations of DMA can be overcome using multiple DMA channels and TDs.

About the Author

Name:	Anu M D
Title:	Applications Engineer
Background:	Anu is an Applications Engineer in Cypress Semiconductor's Consumer and Computation Division focused on PSoC applications.

Document History

Document Title: AN61102 - PSoC 3 and PSoC 5LP - ADC Data Buffering Using DMA

Document Number: 001-61102

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	2963929	ANMD	06/30/2010	New application note
*A	3012497	ANMD	08/20/2010	Updated 20-Bit ADC Data Buffering Using DMA and projects to Beta 5.
*B	3156327	ANMD	01/27/2011	Updated for PSoC Creator 1.0.
*C	3294938	ANMD	06/28/2011	Updated 20-Bit ADC Data Buffering Using DMA and Basic Concept. Added Disable ADC interrupts. Updated project to SP2.
*D	3386960	ANUP	11/08/2011	Added figures 4, 6, 8, 14, 15, and 16. Added 12–Bit SAR ADC Data Buffering Using DMA section. Major rewrite of the application note and updated template.
*E	3446170	NIDH	11/29/2011	Updated for PSoC Creator 2.0
*F	3811896	RRSH	11/14/2012	Updated for PSoC 5LP
*G	3870753	ANMD	01/16/2013	Fixed broken link on page15.
*H	4429438	RNJT	07/03/2014	Fixed typographical errors, updated projects to PSoC Creator 3.0 SP1
*I	4515342	RNJT	09/26/2014	Updated the example projects to include a continuous data buffering example.
*J	5253989	ASRI	05/13/2016	Updated associated project for PSoC Creator 3.3 CP2 Added Figure 12, Figure 19, Figure 20. Updated the section Example Projects: Operation and Test Procedure for continuous data buffering example project Added reference for AN84810 - PSoC® 3 and PSoC 5LP Advanced DMA Topics Updated template
*K	5732858	AESATP12	05/16/2017	Updated logo and copyright.
*L	6131829	NIDH	04/11/2018	Updated project to PSoC Creator 4.2. Minor edits throughout the document.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709