Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE97089 - PSoC® 4 ADC to Memory Buffer DMA Transfer

Objective

This code example demonstrates how to set up a PSoC® 4 direct memory access (DMA) channel to transfer data from an ADC peripheral to the memory buffer.

Overview

This code example sets up the PSoC 4 DMA controller to transfer data from the ADC result register to an array in SRAM. A transfer is initiated each time the ADC asserts an end of conversion (EOC) signal. At the end of each DMA transfer, an interrupt is generated. The CPU implements a simple moving average filter on the data in the array and sends the results through a UART.

Requirements

Tool: PSoC Creator™ 3.2

Associated Parts: PSoC 4200M and PSoC 4100M

Related Hardware: CY8CKIT-044

Design

Every DMA channel in PSoC 4 has two descriptor structures. The descriptor comprises information regarding the source and destination address, the modes of transfer, and other specifics related to a transfer. You can choose to use one or both the descriptors in the channel. Refer the DMA Component datasheet for more details on the use of descriptors.

In this code example, a single DMA channel is configured with a single descriptor, which has the source address as the ADC result register and the destination as a buffer in memory. The trigger (tr_in) for the DMA is the ADC EOC. The transfer mode is set as “Single data element per trigger,” which leads to the ADC result being transferred to the memory buffer every time the EOC is asserted. Figure 1 illustrates the concept of this code example, and Figure 2 shows the PSoC Creator implementation.

Figure 1. ADC to Memory Buffer DMA Transfer

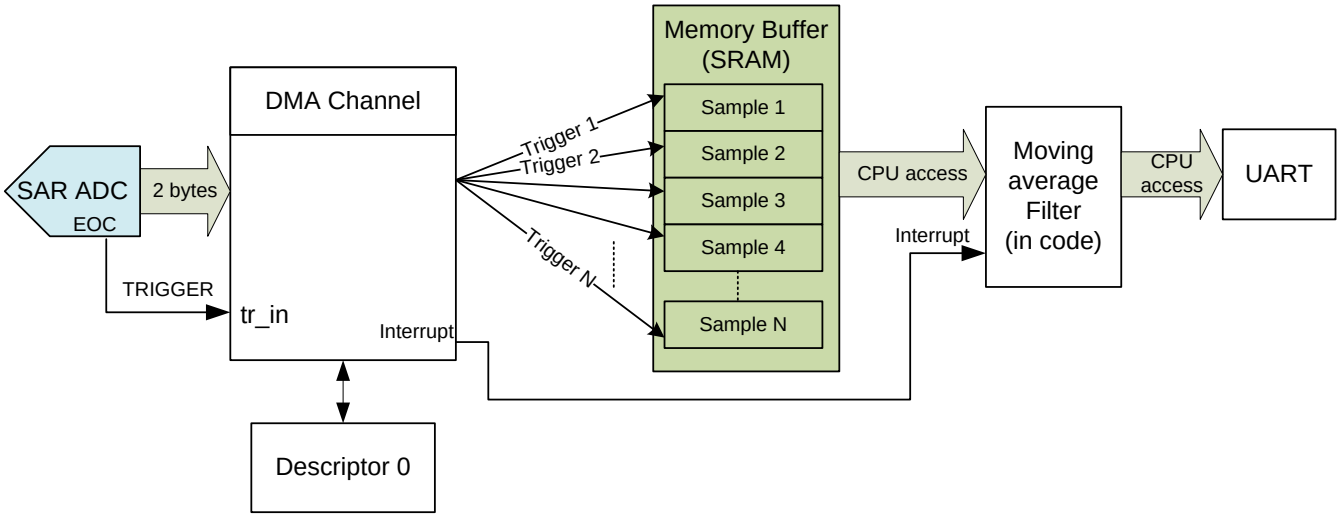
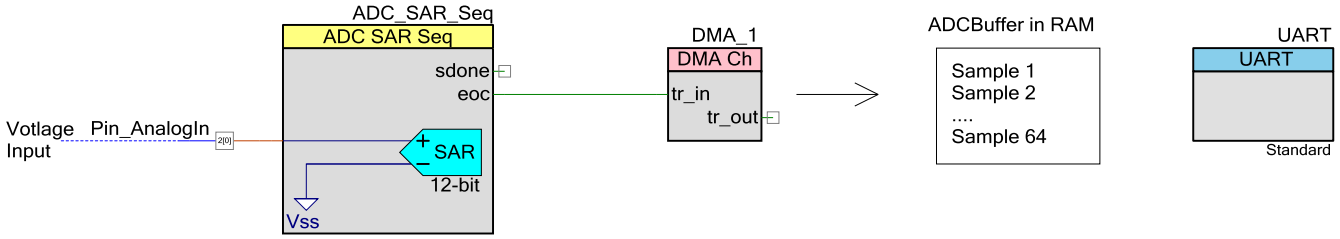


Figure 2. PSoC Creator Implementation



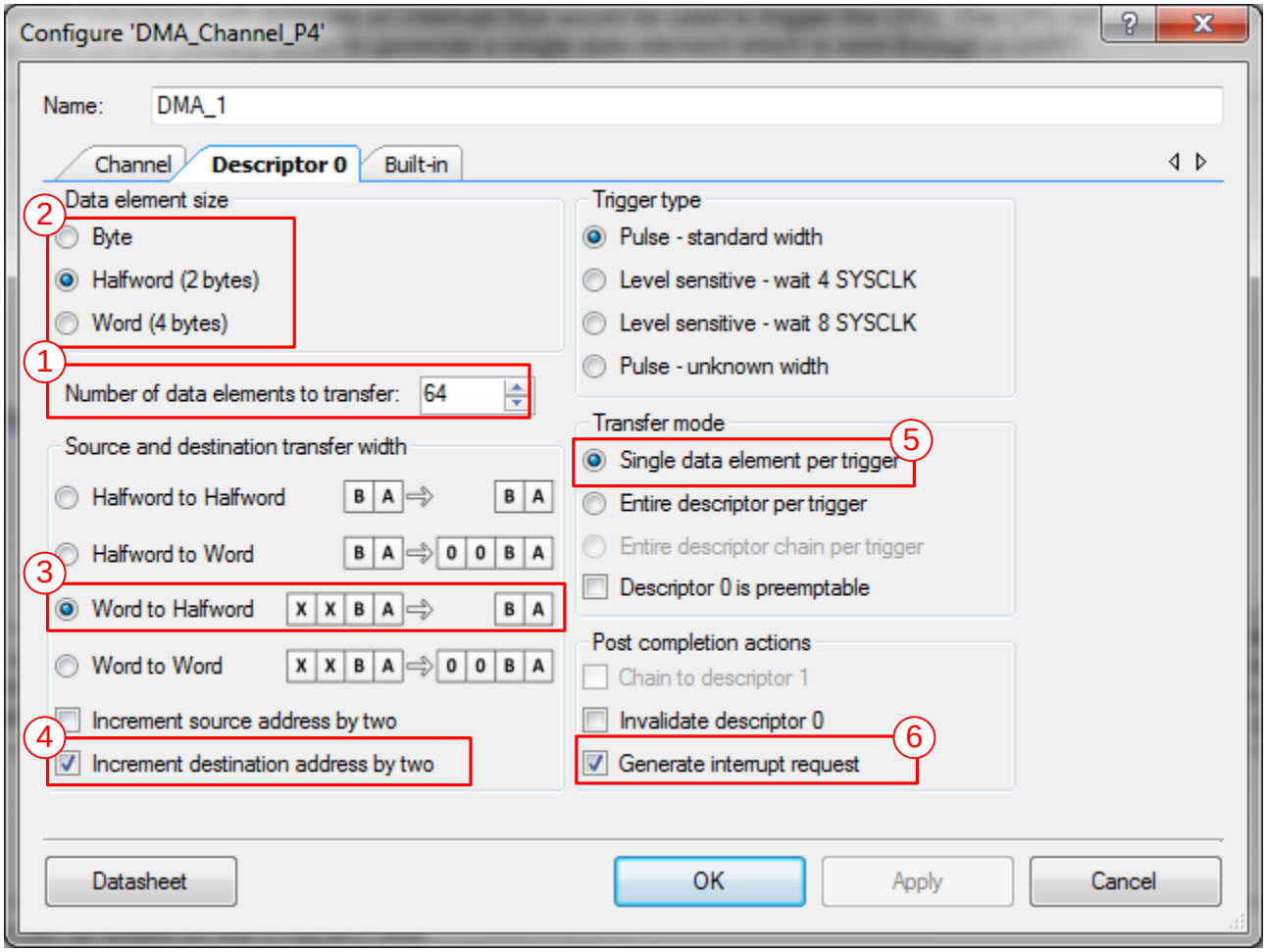
The SAR ADC is configured for a sample rate of 6400 sps. The DMA descriptor is configured to transfer a total of 64 ADC results to the memory buffer. The DMA descriptor is completed every 10 ms and is configured to generate an interrupt on completion of the descriptor, informing the CPU that the DMA is complete and the memory buffer is full.

When the interrupt is generated, the CPU implements an averaging on the 64 elements in the buffer and prints out the result through the UART. The UART is configured for a baud rate of 115.2 kbps.

Figure 3 shows the DMA configuration for the project:

1. The **Number of data elements to transfer** is set to “64.” This configures the descriptor to transfer 64 elements before generating a completion interrupt.
2. The **Data element size** specifies the width of the data being transferred. This field also determines the address increment size when the source increment or destination increment features are enabled. In this example, since the data transferred is a 12-bit ADC result and the destination is an int16 array, the data element size selected is “Halfword (2 bytes).”
3. The ADC result register (source) is a 32-bit register, and the memory location (destination) is a 16-bit array. Hence, the **Source and destination transfer width** is configured as “Word to Halfword.”
4. The source is the ADC result register and does not require incrementing after each transfer. The destination is a buffer and needs to be incremented after each element transfer, so “Increment destination address by two” is enabled. The destination address is incremented until 64 elements are transferred, after which it is reset to the starting address.
5. The **Transfer mode** is set as “Single data element per trigger.” This means that one transfer from the ADC result register to the buffer is triggered every time the trigger (ADC EOC) is asserted. To complete the full DMA descriptor, 64 separate ADC EOC triggers are required.
6. In the **Post completion actions** field, “Generate interrupt request” is enabled, which generates an interrupt at the completion of the descriptor. In this example, an interrupt will be generated after the completion of 64 transfers. This interrupt can be used to trigger the CPU action to average the results and send them over UART.

Figure 3. DMA Configuration



The code utilizes an int16 array (ADCBuffer) to act as the destination buffer for the DMA transfer. The DMA is initialized with the source and destination addresses.

```
DMA_1_Start((void *) ADC_SAR_Seq_SAR_CHAN0_RESULT_PTR, ADCBuffer);
```

Apart from that, you must set up the DMA interrupt. The DMA has a single interrupt for all eight channels. This interrupt service routine is implemented in CY_ISR(CyDmaInterrupt) in *CyDMA.c*. The interrupt automatically identifies the DMA channel that was the source of the interrupt and branches to the callback that is assigned to the DMA channel. So, to implement the DMA interrupt for this project, you need to register a function as a callback for the DMA channel using the DMA_1_SetInterruptCallback ().

```
DMA_1_SetInterruptCallback(DMA_Done_interrupt);
```

The DMA_Done_interrupt() sets a flag that is polled in the main loop to initiate the averaging and UART output operation.

The main loop polls only for DMA descriptor completion, and when it encounters the completion, it executes the averaging of the buffer data and prints it out through the UART.

```
if(DMADoneFlag == DMA_COMPLETE)
{
    //Reset Flag to specify that a descriptor is in progress
    DMADoneFlag = DMA_IN_PROGRESS;
    //This loop gets the sum of all elements in the buffer
    for(i=0; i<ADC_BUFFER_SIZE; i++)
    {
        Result = Result + ADCBuffer[i];
    }
   Result = Result >> 6;//Divides result by 64 to get average
   sprintf(tmpStr, "%d mV\r\n", ADC_SAR_Seq_CountsTo_mVolts(0,Result));
   UART_UartPutString(tmpStr);         //Print result on the UART interface
   Result=0;
}
```

Design Considerations

In this example, the code is executed only when the DMA interrupt is generated (once every 10 ms). The CPU is free at all other times. The DMA can be functional in the PSoC 4 Sleep mode, so you can optionally put the device to sleep after the data has been transmitted through the UART. This can help to reduce system power consumption. The device will be automatically woken up to Active mode when the DMA interrupt occurs.

The averaging of the ADCBuffer can be implemented internal to the ADC itself. In this example, the averaging is done using the CPU to illustrate a scenario of the CPU taking over after a DMA transfer to process the transferred data.

The design implements a flag variable that is marked complete when a DMA interrupt is encountered. The flag variable is reverted to the DMA_PROGRESS state when the ADCBuffer is being processed by the CPU. In this example, the CPU speed is much higher than the rate at which the ADC would be filling up the buffer. This eliminates the possibility of the buffer getting overwritten by the ADC while the CPU is processing it.

For design scenarios in which the ADC sample rates are higher, there are two ways to maintain ADCBuffer integrity during the CPU processing phase, as follows. This code example does not implement both these methods for the sake of simplicity.

- ADC/DMA can be stopped on completion of a transfer and reinitiated once the CPU is done processing the ADCBuffer.
- Double buffering: One DMA descriptor transfers the ADC data to a memory buffer (ADCBuffer). The completion of this transfer can then trigger another DMA transfer that copies the entire content of ADCBuffer to another buffer in a single trigger. The second DMA transfer from memory to memory will be faster. The CPU accesses can happen from the second DMA buffer.

Note When the voltage on the ADC is close to zero, there is a chance of the ADC result going negative due to a negative offset.

Hardware Setup

- Input setup: Connect a voltage input between Vdd and GND, with the wiper pin connecting to P2[0].
- Output setup: The UART TX output is already connected to pin P7[1], which connects to the PSoC 5LP RX line on CY8CKIT-044. The PSoC 5LP on the board implements a UART-to-USB bridge, which enumerates as a COM port on the PC.

You need to set up a terminal program such as Termite to receive the UART data being transferred. You also need to find and connect to the COM port enumerated. The UART settings for the terminal are as follows:

- Baud rate: 115200 bps
- Data bits: 8 bits
- Stop bits: 1 bit
- Parity: None
- Flow control: None

Operation

Once the terminal program is started, the average values sent will start appearing. The change in potentiometer voltage will be reflected in the values being printed.

Note When the voltage on the ADC is close to zero, there is a chance of the ADC result going negative due to a negative offset.

Components

Table 1 lists the PSoC Creator Components used in this example, as well as the hardware resources used by each.

Table 1. List of PSoC Creator Components

Component	Hardware Resources
ADC_SAR_SEQ	SAR ADC
DMA	1 DMA channel
UART	SCB

Related Documents

Table 2 lists the relevant application notes, code examples, Component datasheets, and device and DVK documentation.

Table 2. Related Documents

Application Notes
AN79953 – Getting Started with PSoC 4
Code Examples
CE95275 – Sequencing SAR ADC and Die Temperature Sensor with PSoC 4
CE95272 – SAR ADC in Differential Mode using Pre-Amplifier with PSoC 4
CE95366 – UART Transmit and Receive using a Serial Communication Block (SCB) with PSoC 4
PSoC Creator Component Datasheets
PSoC 4 Sequencing Successive Approximation ADC (ADC_SAR_Seq)
Universal Asynchronous Receiver Transmitter (UART)
PSoC 4 Serial Communication Block (SCB)
Direct Memory Access (DMA)
Device Documentation
PSoC 4 Datasheets
PSoC 4 Technical Reference Manuals
Development Kit (DVK) Documentation
CY8CKIT-044 – PSoC 4 M-Series Pioneer Kit

For questions or suggestions on this code example please contact qvs@cypress.com.

Document History

Document Title: CE97089 - PSoC® 4 ADC to Memory Buffer DMA Transfer

Document Number: 001-97089

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4865821	QVS	08/13/2015	New code example.
*A	5739995	AESATP12	05/25/2017	Updated logo and copyright.

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

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

© Cypress Semiconductor Corporation, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.