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.

CE97091- PSoC® 4: Time-Stamped ADC Data Transfer Using DMA

Objective

This code example uses a DMA channel with two descriptors to implement a time-stamped ADC data transfer. It uses the Watch Dog Timer (WDT) and SAR ADC.

Overview

This code example implements a time-stamped ADC using the DMA controller in PSoC 4. The timing is maintained with the WDT, which is clocked by the 32-kHz watch crystal oscillator (WCO). Two channels of the ADC are sampled and the results are transferred along with the timer value from the WDT. The transfers from the ADC and the WDT are accomplished using a single DMA channel.

Requirements

Tool: PSoC Creator™ 3.3

Associated Parts: PSoC 4200M and PSoC 4100M

Related Hardware: CY8CKIT-044

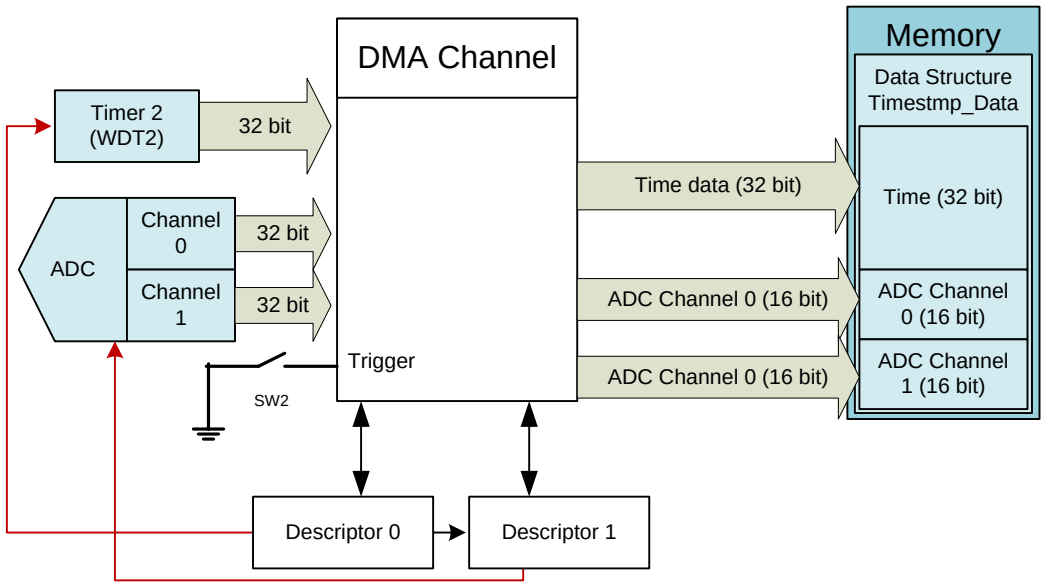
Design

Every DMA channel in PSoC 4 has two descriptor structures associated with it. 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 to the DMA Component datasheet for more details on the use of descriptors.

In this code example, one DMA channel is configured with two descriptors: Descriptor 0 and Descriptor 1. The time-stamped data is a data structure with the two ADC channel results along with the corresponding time of the sampled data. The time stamp data is stored as a data structure (timestmp_Data) in memory. Descriptor 0 is configured to transfer the time value from the WDT to the data structure, timestmp_Data in memory. After completion, Descriptor 0 automatically triggers Descriptor 1, which transfers the ADC result for the two channels to consecutive locations in the data structure. The data structure is updated every time there is a switch press event on the mechanical switch SW2.

Figure 1 illustrates this concept.

Figure 1: Time-Stamped ADC Data Transfer

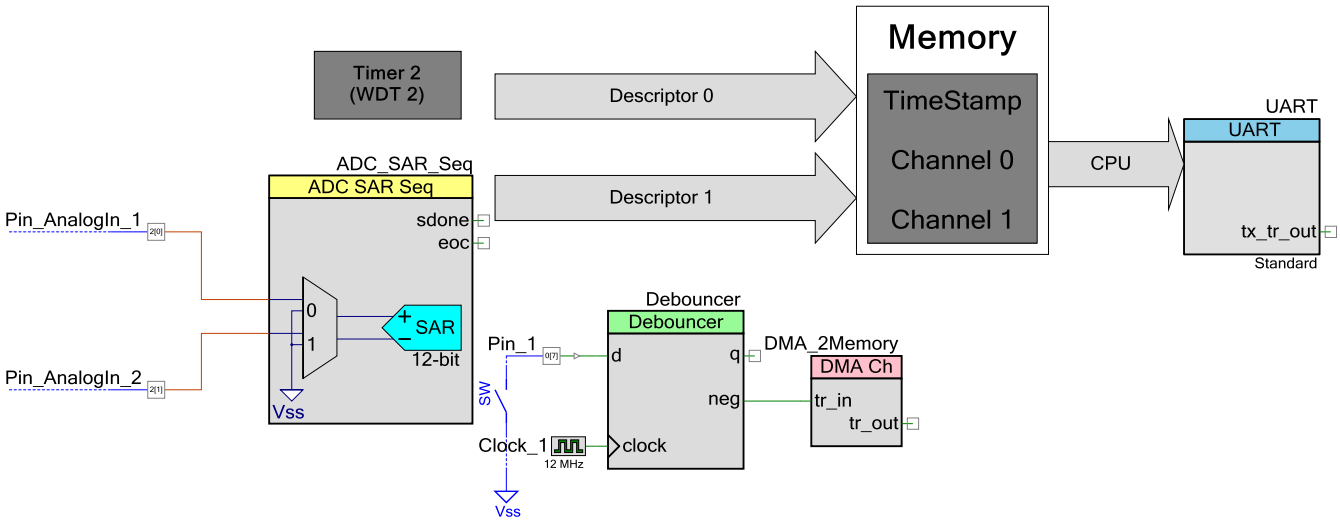


The mechanical switch SW2 is connected to trigger the DMA channel. A debouncer Component is used to avoid the switch from providing multiple trigger signals. The DMA channel is configured to transfer the time information from the WDT register using Descriptor 0 and then the two ADC channel results using Descriptor 1.

This code example also integrates a UART Component to communicate the time-stamped data over a COM interface. This is done to observe the transferred data. The project uses the CPU to fetch the data, process it and convert it into a string that is readable on a terminal tool, and transmit it over UART.

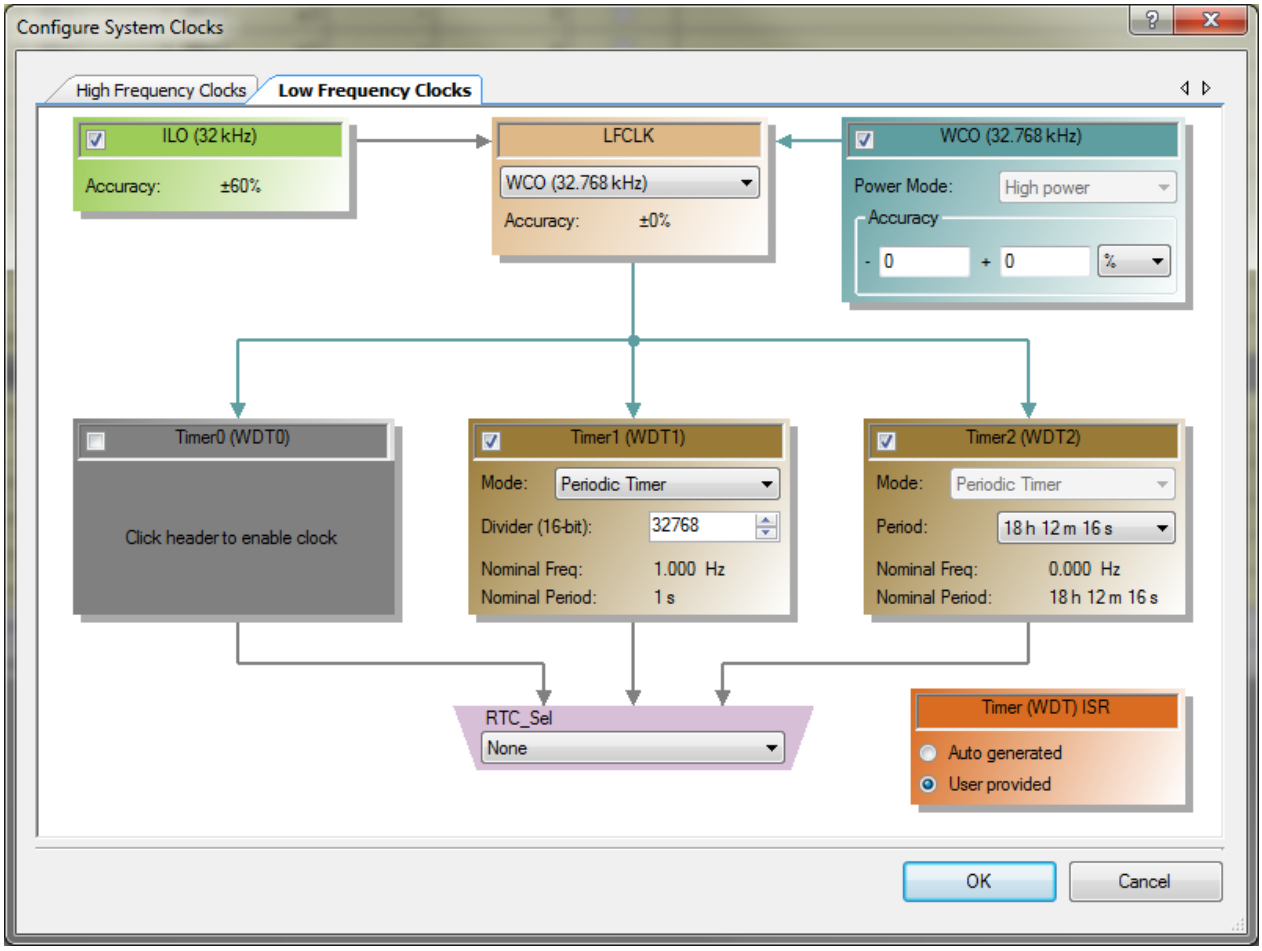
Figure 2 shows the PSoC Creator implementation of the time-stamped ADC using a DMA channel. The ADC in this design is configured to sample two channels at a sample rate of 3014 sps and resolution of 12 bits.

Figure 2: Time-Stamped ADC Data Transfer to Memory- PSoC Creator Implementation



The time stamp is implemented using the Timer 1 and Timer 2 blocks of the WDT. Timer 1 is used to divide the 32-kHz clock to get a 1-second timing. Timer 2 is clocked by Timer 1 to implement a timer counting in seconds. The Timer 2 value is used for time stamping. This configuration is done in the Design Wide resources/clocks configuration as shown in Figure 3.

Figure 3: WDT Configuration



Cascading of the Timer 1 output to Timer 2 is done in the firmware using the following code:

```
/* Configuring the WDT to cascade Timer1(WDT1) and Timer2 (WDT2) */
CySysWdtSetCascade(CY_SYS_WDT_CASCADE_12);
```

For more details on the WDT and Timers, see the *System Reference Guide*.

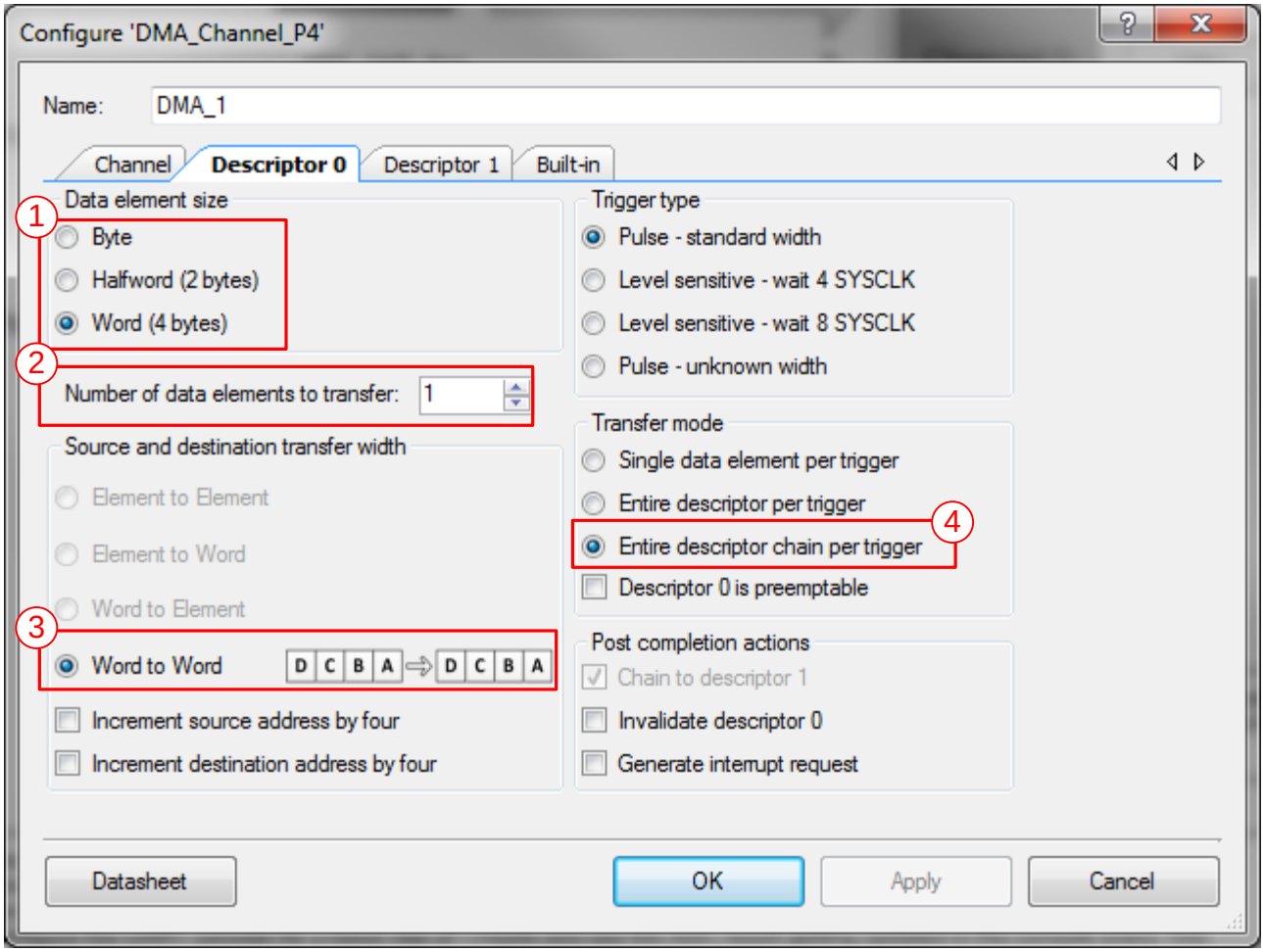
The DMA channel is configured to have two descriptors. The destination of all transfers is a data structure named timestmp_Data.

```
struct data_structure
{
    int32 time;
    int16 Channel[2];
} timestmp_Data;
```

Descriptor 0 Configuration

Descriptor 0 is configured as shown in Figure 4. It transfers data from the Timer 2 (WDT2) counter register to the data structure (timestmp_Data.time) in memory.

Figure 4: Descriptor 0 Configuration



The settings for Descriptor 0 are as follows:

1. The source of the time data is the Timer 2 counter register. The data being transferred is the seconds count from Timer 2, which is a 32-bit value. The destination of the transfer is also a 32-bit element (timestmp_Data.time) in the data structure. Therefore, the **Data element size** for the transfer is set as Word (four bytes).
2. Only a single 32-bit data is transferred as part of this descriptor; therefore, the **Number of data elements to transfer** is set to "1".
3. The data width of PSoC 4 peripheral registers is four bytes (word); therefore, typically, the Source or Destination transfer width should be set to "Word" when DMA is using a peripheral as its source or destination. The source and destination transfer width for the DMA Component must match the addressable width of the source and destination, regardless of the amount of data that needs to be moved. The memory can be accessed as 8-bit, 16-bit, or 32-bit.

   For Descriptor 0, the source is the timer register (peripheral register), which needs to be accessed as 32-bit. The destination is a 32-bit variable in memory. Because the source and destination for this transfer are 32 bits wide, the **Source and destination transfer width** is set as "Word to Word".

4. The switch press will trigger a burst that will transfer the time stamp and ADC data one after the other. This DMA channel is set up to operate when a single trigger initiates all transfers. The **Transfer mode** is set to “Entire descriptor chain per trigger”. When this setting is selected, the trigger signal will initiate and complete both the descriptors in the DMA channel.

The source and destination of this descriptor are set as a part of the DMA start routine in the *main.c* code.

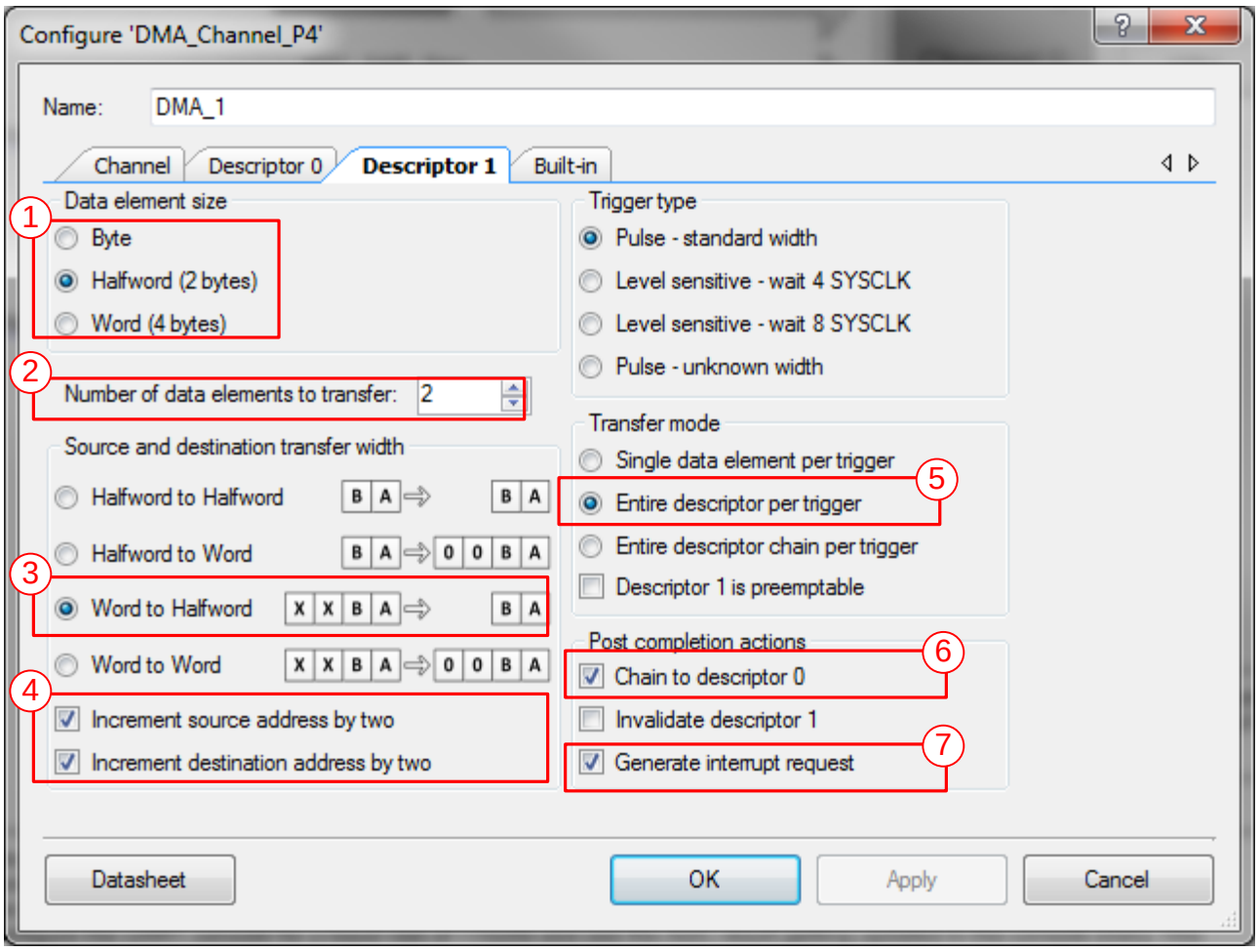
```
DMA_1_Start((void*)CY_SYS_WDT_CTRHIGH_PTR,(void*)&timestmp_Data.time);
```

You can obtain the source register name from either the LFCLK datasheet under “System Reference Guides” or the *cy_lfclk.h* register definitions. The destination is set as timestmp_Data.time.

Descriptor 1 Configuration

Figure 5 shows the Descriptor 1 configuration. Descriptor 1 is automatically triggered after the completion of Descriptor 0. Descriptor 1 transfers the ADC channel results from the ADC result registers to the data structure (timestmp_Data.channel).

Figure 5: Descriptor 1 Configuration



The settings for Descriptor 1 are as follows:

1. The source of the data is the ADC result register. Only 16 bits of the ADC result register are valid. The destination is a 16-bit variable in the data structure. Therefore, the **Data element size** parameter is set as “Half word (2 bytes)”.
2. Two ADC channel results are transferred as part of the descriptor; therefore, the **Number of data elements to transfer** is set as “2”.
3. The data width of PSoC 4 peripheral registers is four bytes (word), so typically the Source or Destination transfer width should be set to “Word” when DMA is using a peripheral as its source or destination. The source and destination transfer width for the DMA Component must match the addressable width of the source and destination, regardless of the amount of data that needs to be moved. The memory can be accessed as 8-bit, 16-bit, or 32-bit.

   The source of the transfer for this descriptor is the ADC result register (peripheral register), which needs to be accessed as a 32-bit register. The destination of the transfer is a variable in memory that can be accessed as 16-bit. Because the source of this transfer is 32-bit and the destination is 16-bit, the **Source and destination transfer width** is set as “Word to Half word”.
4. The source is two ADC channel results that are in consecutive memory locations. The destination of the transfer is two 16-bit variables in consecutive locations in memory. Therefore, both the source and destination addresses are set to “Increment”.

   Note: Because the **Source and destination transfer width** was set as “Word to Half word”, the source increments in steps of 32 bits, and the destination in steps of 16 bits.
5. This DMA channel is set up to operate when a single trigger initiates all transfers, as described in Descriptor 0 Configuration. The **Transfer mode** is set to “Entire descriptor chain per trigger”. When this setting is selected, the trigger signal will initiate and complete both the descriptors in the DMA channel.
6. Once Descriptor 1 is completed, the DMA channel transfer is considered complete for the current trigger. However, the subsequent triggers should start the transfers from Descriptor0. For this reason, the “**Chain to descriptor0”** option is enabled in Post completion actions. This selection makes sure that control is reset back to Descriptor 0.
7. The “**Generate interrupt request”** option generates an interrupt at the end of the descriptor transfer. This interrupt provides a flag for the CPU to initiate UART processing of the data in the data structure and transfer over UART.

The Descriptor 1 source address is the ADC’s result register and destination is the data structure (timestmp_Data.Channel). The ADC is sampling two channels, and the two channel results are in consecutive registers. The source and destination of transfer are set using the following code

```
DMA_1_SetSrcAddress(DESCRIPTOR_1, (void*)ADC_SAR_Seq_SAR_CHAN0_RESULT_PTR);
DMA_1_SetDstAddress(DESCRIPTOR_1, timestmp_Data.Channel);
```

Design Considerations

Note that no code is present after the initializations. The CPU is free after the initialization steps. The DMA can be functional in the PSoC 4 Sleep mode, so you can optionally put the device to sleep after the initializations are done. This can help to reduce system power consumption.

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. This may be noticed on either of the channel results when they are close to zero.

Hardware Setup

- Input setup: Connect a voltage input to the pins P2[0] and P2[1]. To do this, you can use a potentiometer connected between Vdd and GND with the wiper pin connecting to one of the input pins.

  The project also has a switch input to trigger the transfer which connects to pin P0[7]. SW2 is already connected on this pin and hence does not require additional connections.
- Output setup: On the CY8CKIT-044, the UART TX output is already connected to pin P7[1] which connects to the PSoC 5LP RX line. The PSoC 5LP on the board implements a UART-to-USB bridge, which enumerates as a COM port on the PC.

If you are using a different hardware (not CY8CKIT-044), make sure you connect the UART TX pin from the project to the right pin where the UART bridge connection is made.

You need to setup 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: 115,200 bps
- Data bits : 8 bits
- Stop bits : 1 bit
- Parity: None
- Flow control: None

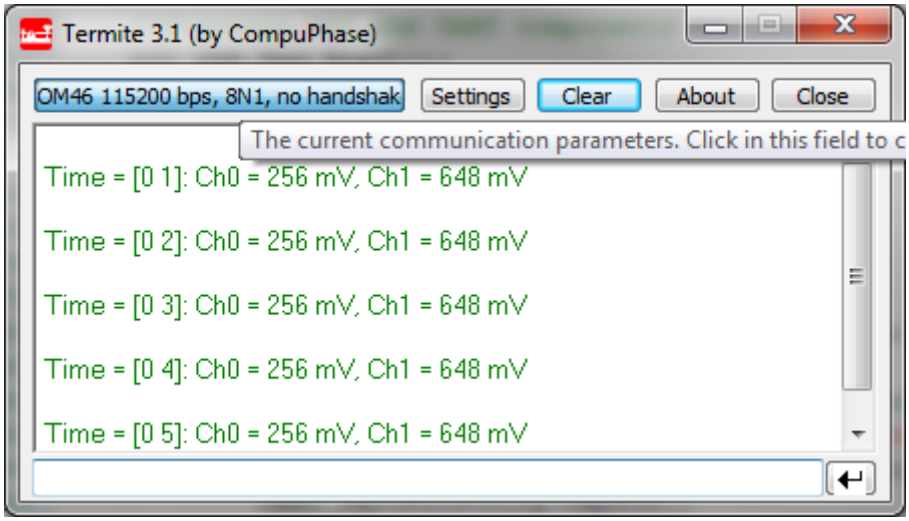
Operation

1. Make the input and output setup as described in the Hardware Setup section.
2. Start the terminal program and make sure it is connected to the right COM port for receiving data from the device.
3. Press the switch SW2 and verify that instantaneous time-stamped data is displayed on the terminal program.

   An example screen shot taken from Termite is shown in the Figure 6. The data is displayed in the following format:

   Time = [<time in seconds>]: Ch0= <Channel0 voltage in mV>, Ch1= <Channel1 voltage in mV>.

Figure 6: Data streamed through UART



4. An instance of time stamped data is transferred every time the SW2 switch is pressed and this gets displayed on the screen.

   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 or User Module	Hardware Resources
ADC_SAR_SEQ	SAR ADC
DMA	1 DMA channel
Debouncer	PLD (5 macrocells)
UART	SCB

Related Documents

Table 2 lists all relevant application notes, code examples, knowledge base articles, device datasheets, and Component datasheets.

Table 2. Related Documents

Application Notes	
AN79953: Getting Started with PSoC® 4	
Code Examples	
CE97088 - PSoC ® 4: ADC TO PWM DMA TRANSFER	
CE97089 – PSoC® 4: ADC to Memory Buffer DMA Transfer	
CE95275:Sequencing SAR ADC and Die temperature sensor with PSoC 4	
CE95272:SAR ADC in Differential Mode using Pre-Amplifier with PSoC 4	
PSoC Creator Component Datasheets	
ADC_SAR_Seq: PSoC 4 Sequencing Successive Approximation ADC	
DMA: PSoC 4 Direct Memory Access (DMA) Channel	
Debouncer : Debouncer	
UART: PSoC 4 Serial Communication Block (SCB)	
Device Documentation	
PSoC 4 Datasheets	PSoC 4 Technical Reference Manuals
Development Kit (DVK) Documentation	
PSoC® 4 M-Series Pioneer Kit	

Document History

Document Title: CE97091- PSoC® 4: Time-Stamped ADC Data Transfer Using DMA

Document Number: 001-97091

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5021986	QVS	01/06/2016	New code example

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

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 |PSoC 3 | PSoC 4 |PSoC 5LP

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corp. 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

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2016. 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.
This 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.