CE218552 – PSoC 6: DMA transfer between UART and a Memory Buffer: Using ModusToolbox

Objective

This code example demonstrates a UART-to-memory buffer data transfer using DMA, with no CPU usage, on the PSoC® 6 MCU, using ModusToolbox IDE.

Requirements

Tool: ModusToolbox™ IDE 1.1

Programming Language: C

Associated Parts: All PSoC 6 MCU parts

Related Hardware: PSoC 6 WiFi-BT Prototyping Kit

Overview

This example demonstrates how a PSoC® 6 DMA channel transfers data received from the UART to a buffer in memory. When the buffer is filled, a second DMA channel drains the buffer to the UART, to be echoed back.

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure the kit is configured correctly.

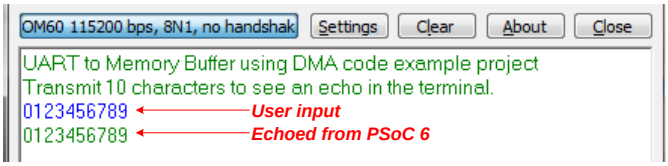
Software Setup

This example uses a terminal emulator. Install one if you don't have one.

Operation

1. Connect the kit to your PC using the provided USB cable.
2. Open your terminal software and select the KitProg COM port, with a baud rate setting of 115200 bps. Set the other serial port parameters to 8N1.
3. Import the code example into a new workspace. If you aren't familiar with this process, see KBA225201.
4. Program the PSoC 6 MCU device. In the project explorer, select the **mainapp** project. In the Quick Panel, scroll to the **Launches** section and click the **Program (KitProg3)** configuration.
5. A welcome text appears as shown in Figure 1.

Figure 1. Terminal Output



6. Now, you can enter bytes through the terminal input, which are received by the PSoC 6 UART and stored in the buffer. When the number of characters sent is equal to 10, PSoC 6 echoes back all the 10 characters.

Debugging

You can debug the example to step through the code. Use a **Program+Debug** configuration. If you are unfamiliar with how to start a debug session with ModusToolbox IDE, see KBA224621.

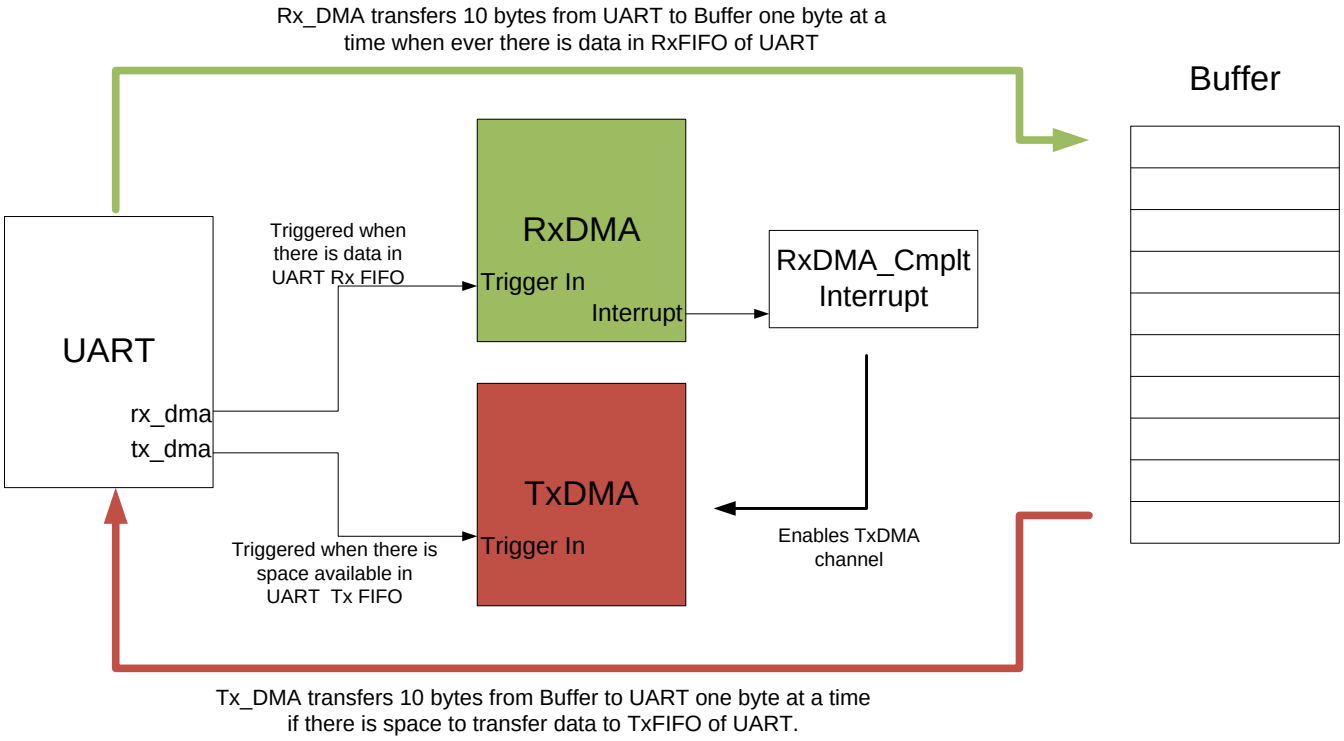
Design and Implementation

The design implements a DMA interface between a UART and a buffer in RAM. There are two DMA channels, as Figure 2 shows. RxDMA handles the transfer of data from the UART's Rx FIFO to the buffer. TxDMA handles the transfer of data back from the buffer to the UART TxFIFO. The buffer size is 10 bytes.

The UART is configured to trigger the RxDMA channel when there is at least one byte of data present in the UART's RxFIFO. The RxDMA moves one byte of data from the UART to the buffer each time it is triggered. RxDMA's transfer descriptor is configured to transfer 10 bytes, which is the size of the buffer. After transferring 10 bytes, RxDMA generates an interrupt (RxDMA_Cmplt), which enables the TxDMA channel.

The TxDMA channel's transfer descriptor is configured to transfer 10 bytes from the buffer to the UART's TxFIFO. Each byte transfer of the TxDMA is triggered by the UART's TxFIFO status. The UART triggers TxDMA when there is at least one byte of space available in the TxFIFO. This mechanism ensures that TxDMA transfer to the UART does not result in overflow of TxFIFO.

Figure 2. Block diagram of the Code example



Resources and Settings

Table 1 lists some of the ModusToolbox resources used in the example, and how they are used in the design. The *design.modus* file contains all the configuration settings. For example, for pin usage and configuration, open the **Pins** tab of the design file.

Table 1. ModusToolbox Resources

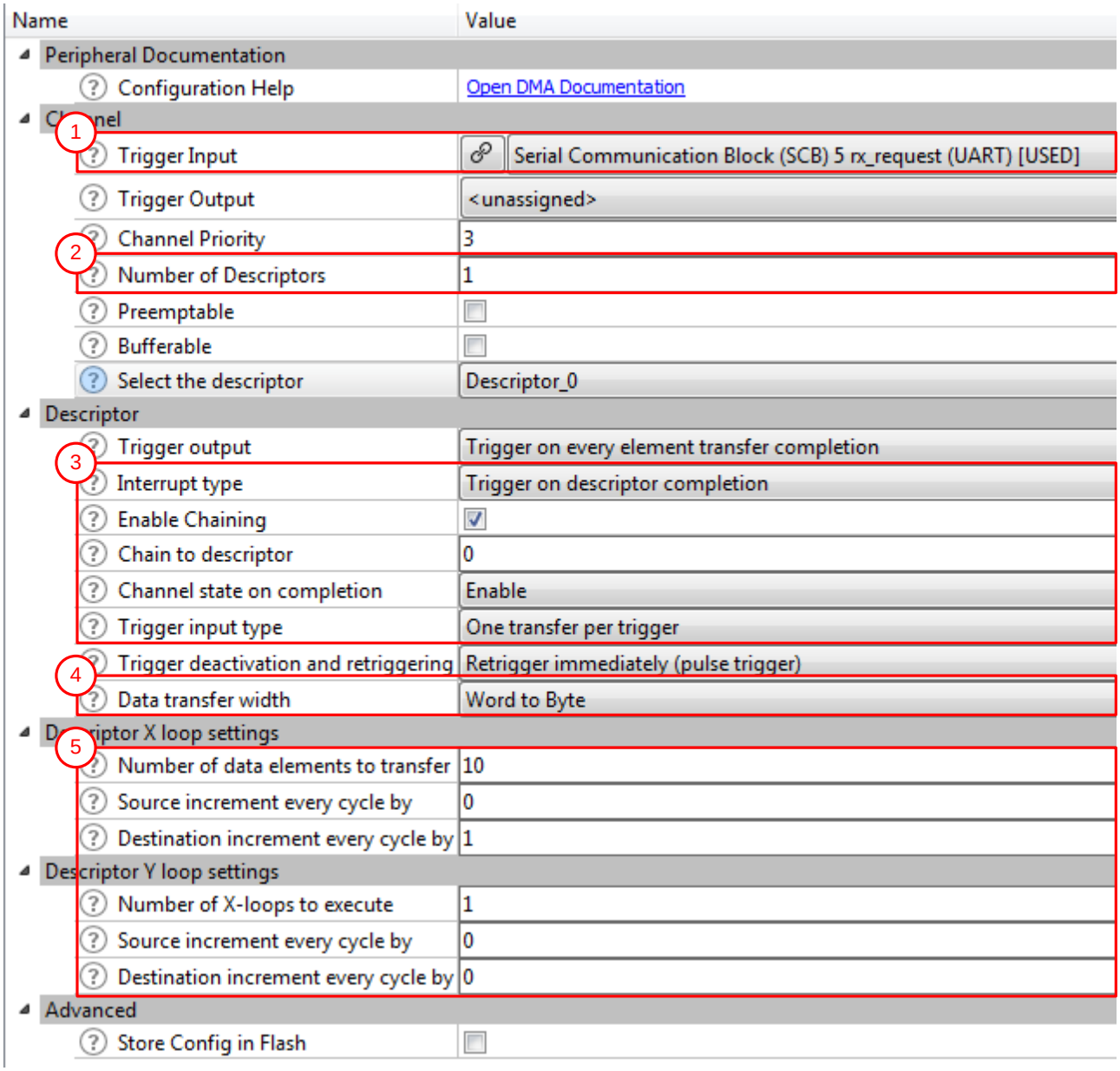
Resource	Alias	Purpose
SCB 5 (UART)	UART	Implements a UART interface
DMA DataWire0: Channel 26	TxDMA	Transfers data from the buffer to the UART for transmission

DMA1 DataWire0: Channel 27	RxDMA	Transfers data received at the UART to the buffer

RxDMA Configuration

RxDMA transfers data from the UART to the buffer. The basic configuration for the RxDMA Component is shown in Figure 3

Figure 3: RxDMA Configuration



1. The **trigger input** to the RxDMA is routed from the SCB5's rx_request.
2. The DMA channel is configured to a simple UART register-to-buffer transfer; only one descriptor is needed. The **number of descriptors** is set to 1.
3. RxDMA implements an interrupt on completion of the descriptor. Therefore, the **Interrupt type** is set as "Trigger on descriptor completion". Once the descriptor is complete, the next receive event on the UART should continue the DMA transfers to the buffer. This is achieved by setting the **Chain to descriptor** as "Descriptor_1" which makes the DMA

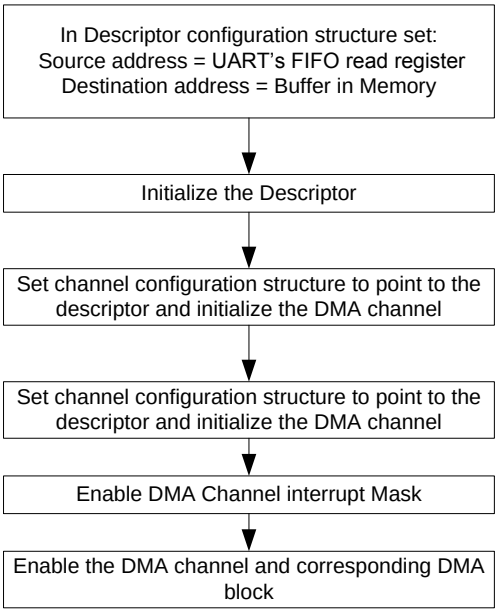
execute the same descriptor in a loop. In addition, the **Channel state on completion** is set to “Enable” so that the descriptor is running in a loop with no interruption. The input trigger options set how the descriptor responds to the trigger inputs. In this code example, the **trigger input type** is set to one transfer per trigger because a single byte transfer from the UART to the buffer is needed on every trigger generated by the UART

4. The data is transferred from a UART hardware register to a memory location. The data being transferred from the memory is a byte wide. Access to the UART register is 32-bit. **Data transfer width** is set to “Word to byte”.
5. The DMA can set up two nested loops of transfer; x loop is the inner loop of transfer. This descriptor transfers 10 bytes from the UART to buffer. Therefore, the **number of data elements to transfer** in **X loop settings** is set as 10. Because the data source is always the UART Rx FIFO register, **source increment every cycle by 0**. Because the destination is a buffer array, **destination increment every cycle by 1**. Thus, the data from the UART register is moved to sequential locations in the buffer.

   The Y loop is not used in this code example. Therefore, the number of data elements to transfer is set as 1. Both source and destination increments are set to 0.

The DMA requires some initialization code, which is in the main.c file. Figure 4 shows the flow chart for the DMA initialization code for RxDMA channel.

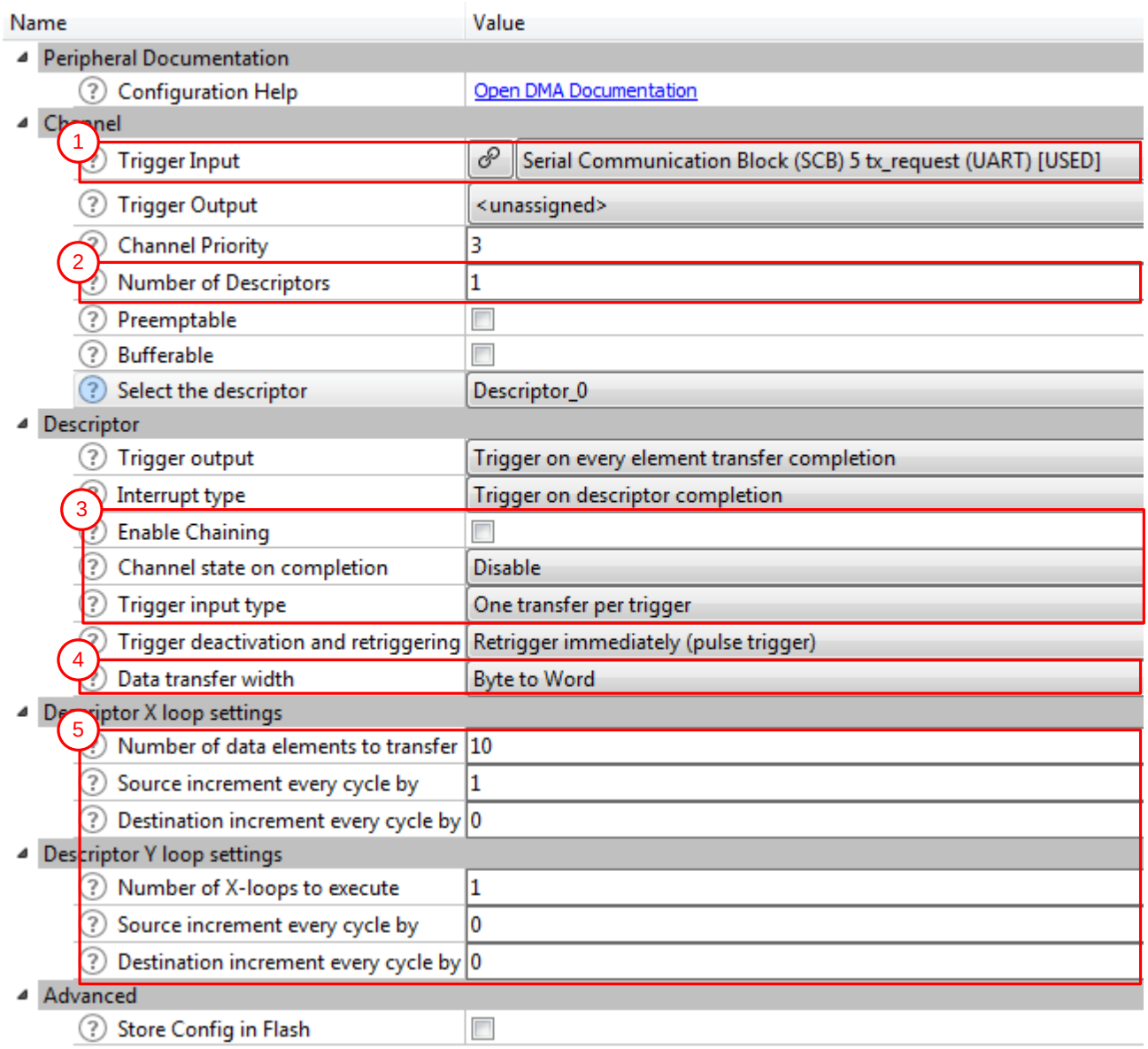
Figure 4. Flowchart for DMA Initialization Code



TxDMA Configuration

TxDMA transfers data from the buffer to UART. The basic configuration for the TxDMA Component is shown in Figure 5.

Figure 5: TxDMA Descriptor Configuration



1. The **trigger input** to the TxDMA, is routed from the SCB5's tx_request.
2. The DMA implements a simple buffer to UART register transfer, which can be achieved by using a single descriptor. The **number of descriptors** is set to 1.
3. On completion of the descriptor, TxDMA should go to a disabled state with no active descriptor. This is achieved by disabling **Enable chaining**. Also, the **Channel state on completion** is set to "Disable" so that the descriptor is disabled at the end of the transfer. The TxDMA will be re-enabled in the RxDMA_Cmplt interrupt.

   The **trigger input type** is set to one transfer per trigger because a single byte transfer is needed from the Buffer to UART on every trigger generated by the UART.
4. The data is transferred from a memory location to the UART hardware register. Access to hardware registers like the UART register is 32-bit while the memory access can be 8-, 16-, or 32-bit. **Data transfer width** is set to "Byte to Word".

5. The X loop transfer setting sets up the x loop for the descriptor. The DMA can set up two nested loops of transfer; x loop is the inner loop of transfer. Refer to the DMA Component datasheet for details. This descriptor transfers 10 bytes from the buffer to UART. Therefore, the **number of data elements to transfer** is set as 10. Because the data source is the buffer array, **source increment every cycle by 1**. Because the destination is a UART register, **destination increment every cycle by 0**. Thus, the data from the buffer array is moved sequentially to the UART.

   The Y loop is not used in this code example. Therefore, the number of data elements to transfer is set as 1. Both source and destination increments are set to 0.

Similar to RxDMA, there is some code needed to initialize TxDMA. The initialization code for TxDMA is very similar to the one for RxDMA, shown in Figure 4. The initialization code first configures the descriptor to source as the buffer and the destination as the UART's TX_FIFO_WR register. After this, the initialization code simply initializes the descriptor and the channel. For TxDMA, there is no Interrupt enabled because a DMA channel interrupt is not used from TxDMA.

Reusing This Example

This example is configured for the PSoC 6 WiFi-BT Prototyping Kit. To port the design to a different PSoC 6 MCU device, right-click an application project and choose **Change Device**. If changing to a different kit, you will need to reassign the DMA channels.

Table 2. DMA Mapping across PSoC 6 MCU Kits

Kit name	Device Used	RxDMA	TxDMA
CY8CKIT-062-WiFi-BT	CY8C6247BZI-D54	DMA DataWire0: Channel 1	DMA DataWire0: Channel 0
CY8CKIT-062-BLE	CY8C6347BZI-BLD53	DMA DataWire0: Channel 1	DMA DataWire0: Channel 0
CY8CPROTO-062-4343W	CY8C624ABZI-D44	DMA DataWire0: Channel 27	DMA DataWire0: Channel 26

Related Documents

Application Notes	
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN215656 – PSoC 6 MCU: Dual-CPU System Design	Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design
Code Examples	
Visit the Cypress GitHub site for a comprehensive collection of code examples using ModusToolbox IDE	
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
PSoC 6 MCU: PSoC 62 Datasheet	PSoC 62 Datasheet
PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual (TRM)	PSoC 6 MCU with BLE Architecture Technical Reference Manual
PSoC 6 MCU: PSoC 63 with BLE Register Technical Reference Manual	PSoC 6 MCU with BLE Register Technical Reference Manual
Development Kits	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	
CY8CKIT-062-WiFi-BT PSoC 6 WiFi-BT Pioneer Kit	
CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit	
Tool Documentation	
ModusToolbox IDE	The Cypress IDE for IoT designers
WICED SDK with PSoC 6 Support	Installed with ModusToolbox IDE

Cypress Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right device, and quickly and effectively integrate the device into your design.

For the PSoC 6 MCU devices, see KBA223067 in the Cypress community for a comprehensive list of PSoC 6 MCU resources.

Document History

Document Title: CE218552 – PSoC 6: DMA transfer between UART and a Memory Buffer: Using ModusToolbox

Document Number: 002-25481

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6376543	QVS	02/19/2019	New code example- ModusToolbox 1.1

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 | Code Examples | Projects | Videos | Blogs | Training| Components

Technical Support

cypress.com/support

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