ADuCM3027/ADuCM3029 User Guide

UG-1091

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

How to Set Up and Use the ADuCM3027/ADuCM3029

INTRODUCTION

This user guide provides detailed information of the ADuCM3027/ADuCM3029 microcontroller functionality and features. Each section describes a different feature.

GENERAL DESCRIPTION

The ADuCM3027/ADuCM3029 processor is an ultra low power, integrated, mixed signal, microcontroller system used for processing, control, and connectivity. The MCU subsystem is based on the ARM® Cortex™-M3 processor, a collection of digital peripherals, cache embedded SRAM and flash memory, and an analog subsystem, which provides clocking, reset, and power management capabilities along with the analog-to-digital converter (ADC).

The ADuCM3027/ADuCM3029 processor provides a collection of power modes and features, such as dynamic and software controlled clock gating and power gating, to support extremely low dynamic and hibernate power management.

Full specifications on the ADuCM3027/ADuCM3029 are available in the product data sheet and the ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference Manual.

FEATURES

System features that are common across the ADuCM3027/ADuCM3029 devices include the following:

- Up to 26 MHz ARM Cortex®-M3 processor.
- Up to 256 kB of embedded flash memory with error correction code (ECC).
- Optional 4 kB cache for lower active power.
- 64 kB system SRAM with parity.
- Power management unit (PMU).
- Multilayer advanced microcontroller bus architecture (AMBA) bus matrix.
- Central direct memory access (DMA) controller.
- Beeper interface.
- Serial port (SPORT), three serial peripheral interfaces (SPIs), inter integrated circuit (I^{2}C), and universal asynchronous receiver/transmitter (UART) peripheral interfaces.
- Cryptographic hardware support with advanced encryption standard (AES) and secure hash algorithm (SHA) −256.
- Two real-time clocks (RTCs).
- Three general-purpose timers and one watchdog timer.
- Programmable general-purpose input/output (GPIO) pins.
- Hardware cyclical redundancy check (CRC) calculator with programmable generator polynomial.
- Power-on-reset (POR) and power supply monitor (PSM).
- 12-bit successive approximation register (SAR) ADC.

ADuCM3027/ADuCM3029 FUNCTIONAL BLOCK DIAGRAM

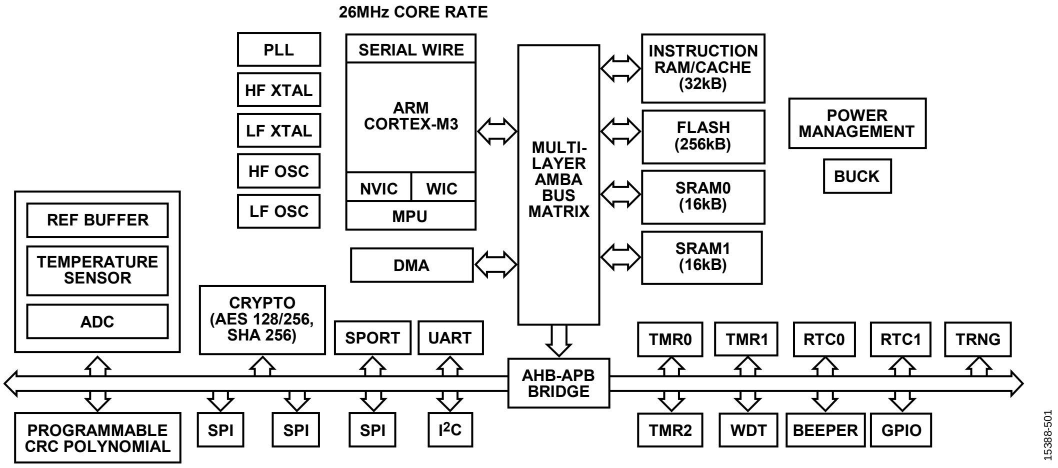


Figure 1.

See the last page for an important warning and disclaimers.

TABLE OF CONTENTS

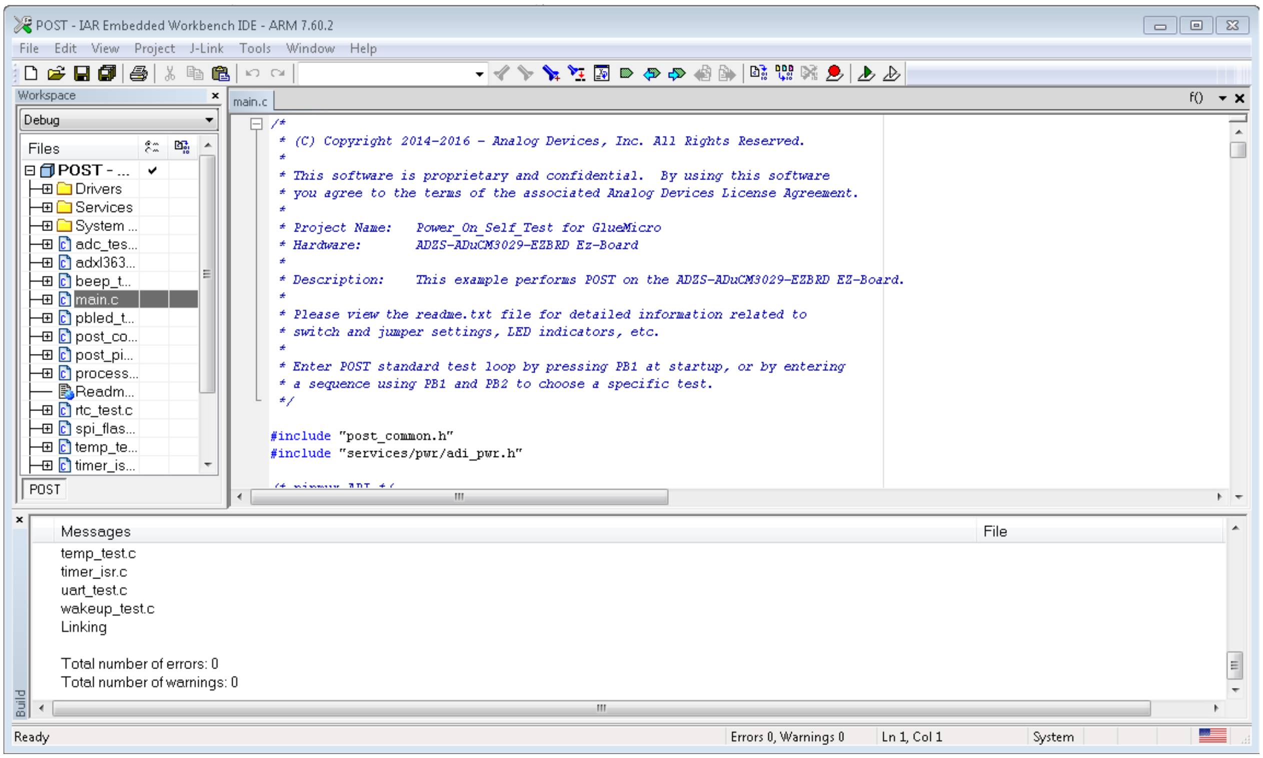
REVISION HISTORY

3/2017—Revision 0: Initial Version

GETTING STARTED

This section introduces the tools and support packages required to develop an application for the ADuCM3029/ADuCM3027 microcontrollers. This section describes how to download, install, and configure the programs used to program the ADuCM3027/ADuCM3029.

This section works as a tutorial by describing different steps in developing an application by using the IAR workbench as an integrated development environment (IDE). This section also describes how to download and run sample codes provided with the board support package (BSP) drivers.



```
/*
 * (C) Copyright 2014-2016 - Analog Devices, Inc. All Rights Reserved.
 *
 * This software is proprietary and confidential.  By using this software
 * you agree to the terms of the associated Analog Devices License Agreement.
 *
 * Project Name:   Power_On_Self_Test for GlueMicro
 * Hardware:       ADZS-ADuCM3029-EZBRD Ez-Board
 *
 * Description:    This example performs POST on the ADZS-ADuCM3029-EZBRD EZ-Board.
 *
 * Please view the readme.txt file for detailed information related to
 * switch and jumper settings, LED indicators, etc.
 *
 * Enter POST standard test loop by pressing PB1 at startup, or by entering
 * a sequence using PB1 and PB2 to choose a specific test.
 */

#include "post_common.h"
#include "services/pwr/adi_pwr.h"
```

```
Messages                                                    File
temp_test.c
timer_isr.c
uart_test.c
wakeup_test.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

Figure 2. IAR Embedded Workbench

SOFTWARE INSTALLATION

The software tools required to develop applications with the ADuCM3027/ADuCM3029 are available for download at http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-aducm3029-ezkit.html.

Table 1. Required Software Tools

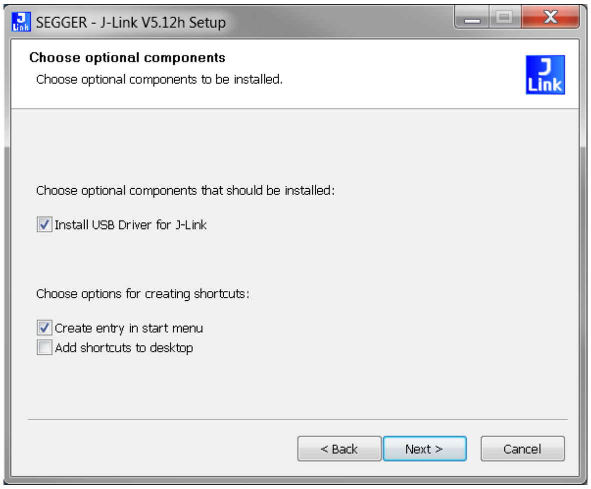
Tool	Functions
IAR Embedded Workbench	Used for compiling, debugging, and code development
Segger J-Link Software	J-Link software and documentation pack includes USB drivers for the emulator, J-Link Commander, and so on
ADuCM3027/ ADuCM3029 BSP Drivers	Includes ADuCM3027/ADuCM3029 peripheral drivers and libraries, IAR configuration files, and example programs

Installing the Segger J-Link Driver

The Segger J-Link USB driver must be installed before using a serial wire interface, such as the interface of the IAR embedded workbench, to download and debug code.

Use the following procedure to install the J-Link USB driver:

1. Download the latest Segger J-Link software and documentation pack.
2. Open the executable software installer on the download directory.
3. Follow the on-screen instructions to complete the installation. Ensure that the **Install USB Driver for J-Link** option is checked, as shown in Figure 3.


SEGGER - J-Link V5.12h Setup

Choose optional components

Choose optional components to be installed.

Choose optional components that should be installed:

☑ Install USB Driver for J-Link

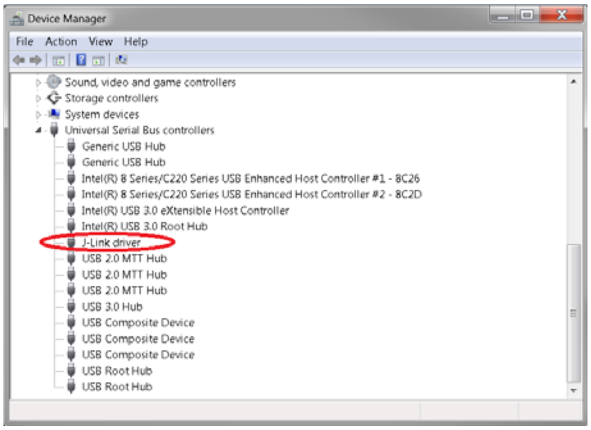
Choose options for creating shortcuts:

☑ Create entry in start menu

☐ Add shortcuts to desktop

Figure 3. Segger J-Link Driver Installation Options

4. Plug in the emulator board and open the device manager.
5. Check that the emulator board appears in the Windows® device manager in the USB controllers lists (see Figure 4).


Device Manager

- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers
  - Generic USB Hub
  - Generic USB Hub
  - Intel(R) 8 Series/C220 Series USB Enhanced Host Controller #1 - 8C26
  - Intel(R) 8 Series/C220 Series USB Enhanced Host Controller #2 - 8C2D
  - Intel(R) USB 3.0 eXtensible Host Controller
  - Intel(R) USB 3.0 Root Hub
  - J-Link driver
  - USB 2.0 MTT Hub
  - USB 2.0 MTT Hub
  - USB 2.0 MTT Hub
  - USB 3.0 Hub
  - USB Composite Device
  - USB Composite Device
  - USB Composite Device
  - USB Root Hub
  - USB Root Hub

Figure 4. Device Manager

After following the software installation procedures, the USB driver for the J-Link is installed and verified.

IAR Tools Installation

The IAR embedded workbench and the included IAR C/C++ compiler generates the fastest performing, most compact code in the industry for ARM-based applications. Therefore, Analog Devices, Inc., created the BSP drivers for the ADuCM3027/ ADuCM3029 for the IAR workbench.

The KickStart edition is a free starter kit and evaluation version of IAR. This edition has limitations, both in code size (32 kB) and in the service and support provided.

The IAR software is available for download on the IAR website. Download a free trial to download the software.

For a detailed procedure on installing the IAR and adding license details if required, refer to the *Installation and Licensing Guide for IAR Embedded Workbench*®, available from the IAR website.

Note that the BSP examples are for IAR Version 7.40.2. Project compatibility issues may occur when using different versions.

ADuCM3027/ADuCM3029 Board Support Package (BSP)

The ADuCM3027/ADuCM3029 BSP provides the configuration, support files, and components required to ease the development of the ADuCM3027/ADuCM3029.

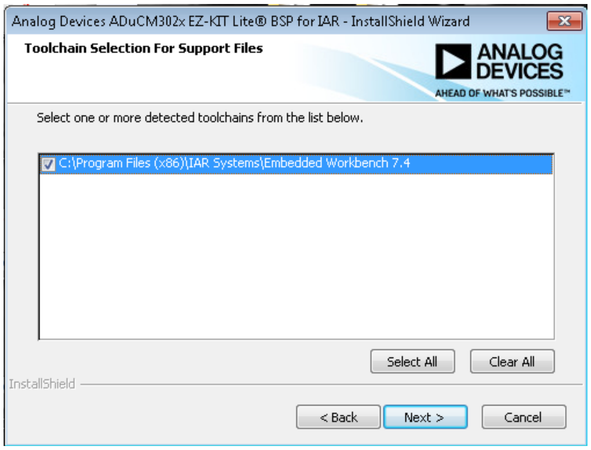
The contents of the BSP are as follows:

- Source files for the device drivers and services for use on the ADuCM3027/ADuCM3029 processor.
- Examples for devices drivers and services.
- Tool chain support. These components are installed in the IAR embedded workspace to configure the tool chain to recognize the ADuCM3027/ADuCM3029.
- Documentation.

Note that the IAR embedded workbench must be installed before installing the BSP.

Use the following procedure to install the BSP:

1. Download the ADuCM3027/ADuCM3029 BSP installer at http://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-aducm3029-ezkit.html#eb-relatedsoftware.
2. Execute the installer application.
3. Follow the instructions to complete the installation.
4. Select the IAR workbench version where the toolchain support is to be installed (see Figure 5).



Analog Devices ADuCM302x EZ-KIT Lite® BSP for IAR - InstallShield Wizard

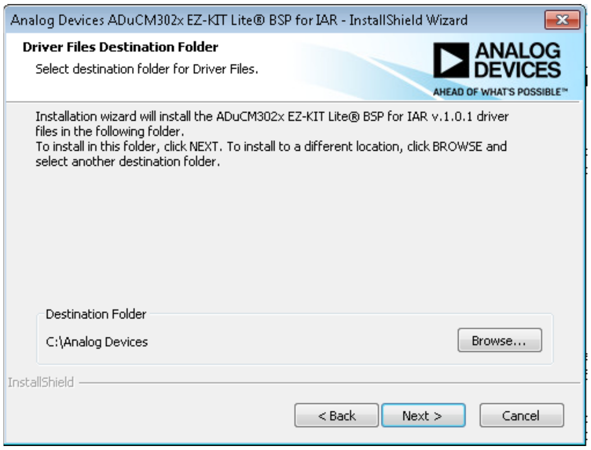
Toolchain Selection For Support Files

Select one or more detected toolchains from the list below.

☑ C:\Program Files (x86)\IAR Systems\Embedded Workbench 7.4

Figure 5. Toolchain Selection Dialogue Box

5. Select the location for the driver and source files of the BSP (see Figure 6). The recommended location is outside the **Program Files** folder due to the access restrictions of Windows.



Analog Devices ADuCM302x EZ-KIT Lite® BSP for IAR - InstallShield Wizard

Driver Files Destination Folder

Select destination folder for Driver Files.

Installation wizard will install the ADuCM302x EZ-KIT Lite® BSP for IAR v.1.0.1 driver files in the following folder.
To install in this folder, click NEXT. To install to a different location, click BROWSE and select another destination folder.

Destination Folder

C:\Analog Devices

Figure 6. BSP Destination Folder Selection Dialogue Box

6. Complete the installation process. The user can view the installation folder of the BSP to check the files and documentation for further information and guidance.

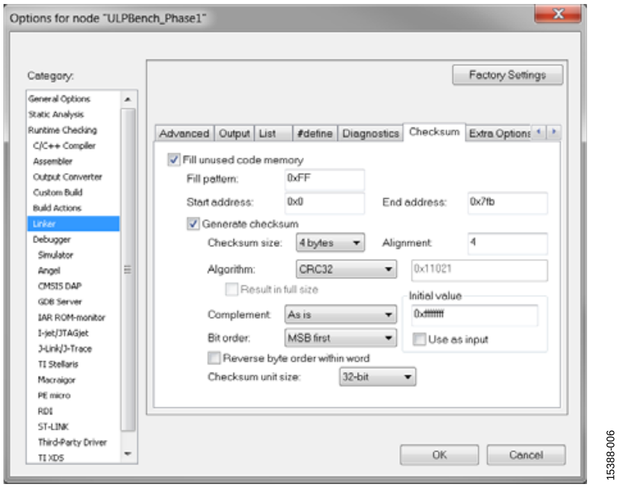
IAR CONFIGURATION

This section describes the IAR configuration procedure for proper operation of the ADuCM3027/ADuCM3029. Only the sections that must be modified from the default values are described.

1. Ensure that, in the general options, the Analog Devices ADuCM3029/ADuCM3027 device option is selected as the target.
2. Under **C/C++ Compiler** > **Optimizations**, select different optimization options for speed, code size, balance, and so on, depending on the application needs. Sometimes, the compiler identifies the writes of a register as eligible to be optimized, which may cause unexpected behavior. In such situations, it is recommended to protect configuration functions from being optimized by using following code:

```
#pragma optimize=none
```

3. Under **C/C++ Compiler** > **Preprocessor,** include the path of the included directories, depending on the code to be run.



☑ Fill unused code memory

Fill pattern: 0xFF

Start address: 0x0 End address: 0x7fb

☑ Generate checksum

Checksum size: 4 bytes Alignment: 4

Algorithm: CRC32 0x11021

□ Result in full size

Initial value: 0xffffffff

□ Use as input

Complement: As is

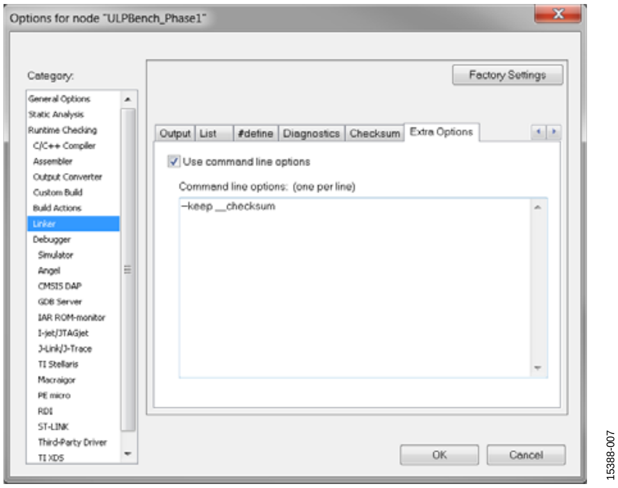
Bit order: MSB first

□ Reverse byte order within word

Checksum unit size: 32-bit

*Figure 7. **Linker** > **Checksum** Configuration Tab*

4. A 32-bit CRC checksum stored in the signature field allows the user code to request an integrity check of the user space. Therefore, it is necessary to configure the checksum as shown in Figure 7 and Figure 8, both configurations in the **Linker** menu.



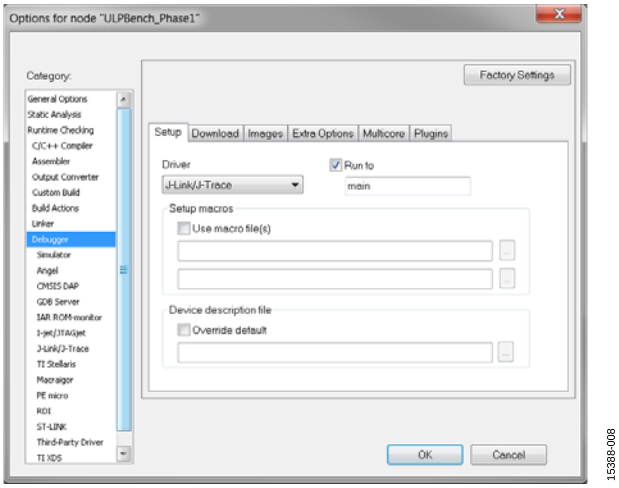
☑ Use command line options

Command line options: (one per line)

```
--keep __checksum
```

*Figure 8. **Linker** > **Extra Options** Configuration Tab*

5. Select **J-Link/J-Trace** as the debugger under the **Driver** dropdown menu. Verify that both the **Verify download** and **Use flash loader(s)** boxes are checked on the **Debugger** > **Download** menu as shown in Figure 9 and Figure 10.



Driver: J-Link/J-Trace

☑ Run to: main

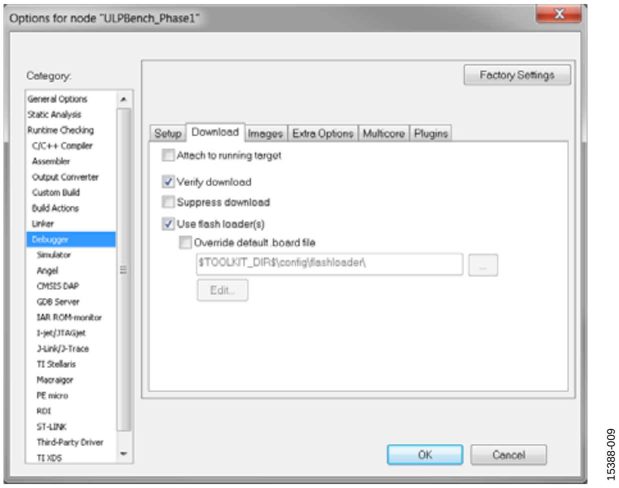
Setup macros

□ Use macro file(s)

Device description file

□ Override default

*Figure 9. **Debugger** > **Setup** Tab*



□ Attach to running target

☑ Verify download

□ Suppress download

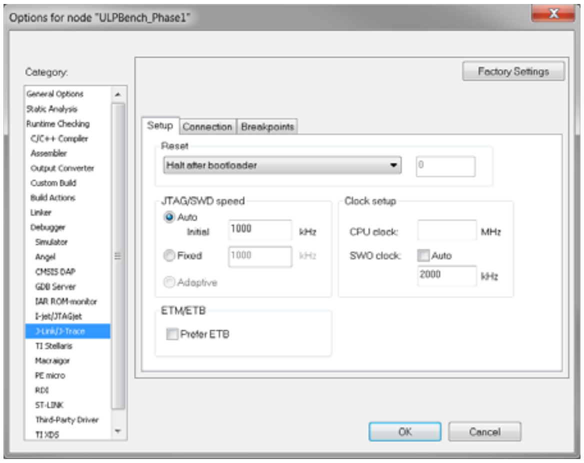
☑ Use flash loader(s)

□ Override default .board file

$TOOLKIT_DIR$\config\flashloader\

*Figure 10. **Debugger** > **Download** Tab*

6. Figure 11 and Figure 12 show the J-Link/J-trace configuration. Be sure to use the **Halt after bootloader** target reset strategy under the **Reset** dropdown menu, otherwise a kernel corruption occurs and the device locks down. To recover from this corruption, the kernel must be reflashed.


Options for node "ULPBench_Phase1"

Category: J-Link/J-Trace

Setup tab

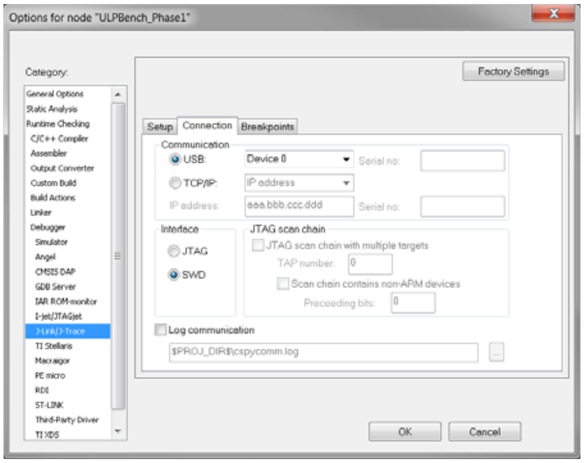
Reset: Halt after bootloader; 0

JTAG/SWD speed: ● Auto, Initial 1000 kHz; ○ Fixed 1000 kHz; ○ Adaptive

Clock setup: CPU clock: MHz; SWO clock: □ Auto, 2000 kHz

ETM/ETB: □ Prefer ETB

Figure 11. ***J-Link/J-Trace*** > ***Setup*** Configuration Tab


Options for node "ULPBench_Phase1"

Category: J-Link/J-Trace

Connection tab

Communication: ● USB: Device 0; Serial no:; ○ TCP/IP: IP address; IP address: aaa.bbb.ccc.ddd; Serial no:

Interface: ○ JTAG; ● SWD

JTAG scan chain: □ JTAG scan chain with multiple targets; TAP number: 0; □ Scan chain contains non-ARM devices; Preceeding bits: 0

□ Log communication: $PROJ_DIR$\cspycomm.log

Figure 12. ***J-Link/J-Trace*** > ***Connection*** Configuration Tab

BUILDING DEMO PROJECTS

IAR Provided Code

Several example projects are available in the **examples** folder, located in the ADuCM3027/ADuCM3029 BSP directory as follows: **BSP Install Folder** > **\ADuCM302x_EZ_Kit_Lite** > **examples**. To open an example project, from the **File** menu, choose **Open** > **Workspace…**, and navigate to the workspace file.

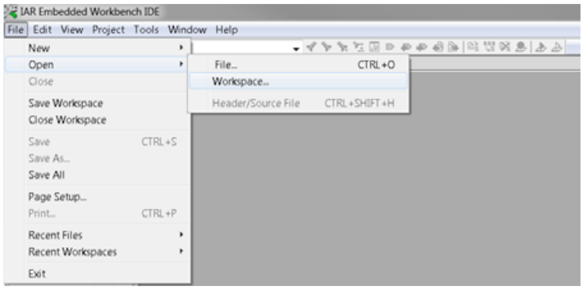
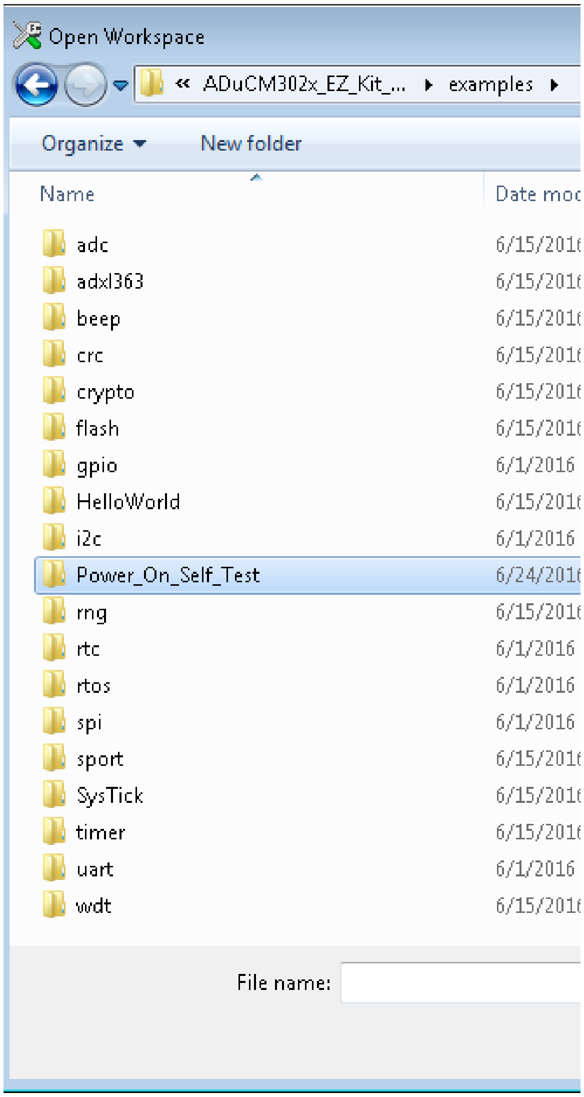


Figure 13. Opening the IAR Workspace

Several relevant projects are available within the **examples** folder of the BSP, as shown in Figure 14.


Open Workspace

« ADuCM302x_EZ_Kit_... ▸ examples ▸

Name	Date moc
adc	6/15/201
adxl363	6/15/201
beep	6/15/201
crc	6/15/201
crypto	6/15/201
flash	6/15/201
gpio	6/1/2016
HelloWorld	6/15/201
i2c	6/1/2016
Power_On_Self_Test	6/24/201
rng	6/15/201
rtc	6/1/2016
rtos	6/1/2016
spi	6/1/2016
sport	6/15/201
SysTick	6/15/201
timer	6/15/201
uart	6/1/2016
wdt	6/15/201

File name:

Figure 14. Available Projects in the BSP Folder

Each example includes a comprehensive, low level, peripheral library that can be used to interface to the peripherals of the ADuCM3027/ADuCM3029. Comprehensive documentation for both the libraries and the examples are included.

Changing Projects

To change projects, right click on a different project in the workspace and click **Set as Active** from the menu that appears (see Figure 15).

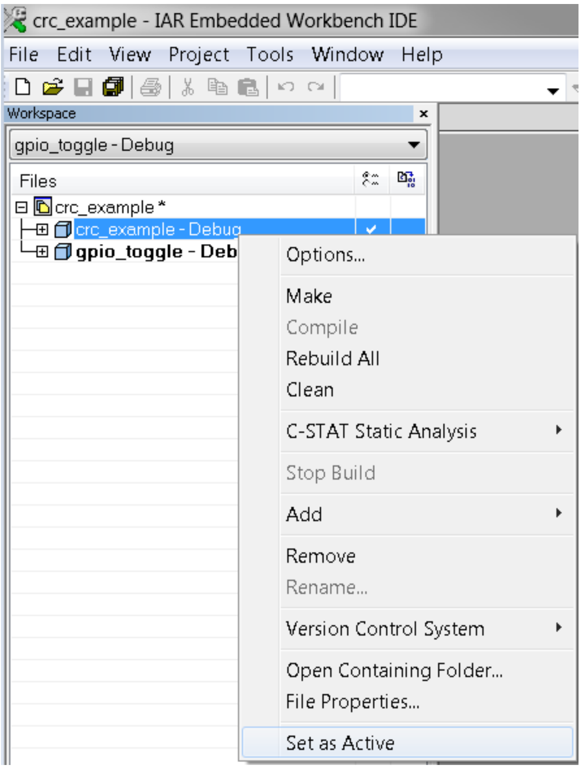


Figure 15. Changing Projects

Building the Application

To build or compile the application code, the user can either click **Project** > **Make** (or press F7) as shown in Figure 16, or click the make button, shown in Figure 17.

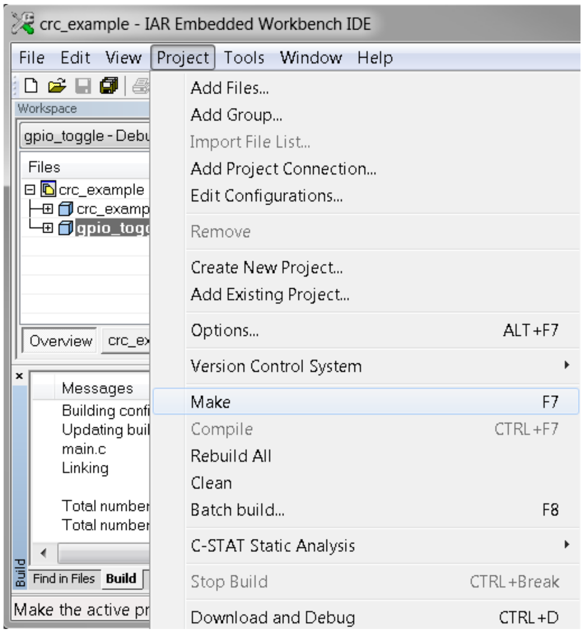


*Figure 16. Building an Application Using **Project > Make***

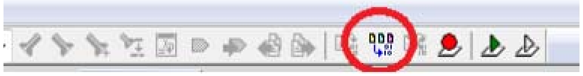


Figure 17. Building an Application Using the IAR Make Toolbar Button

To rebuild the full application code, click **Project** > **Rebuild All**. This action cleans, recompiles, and links all the project files.

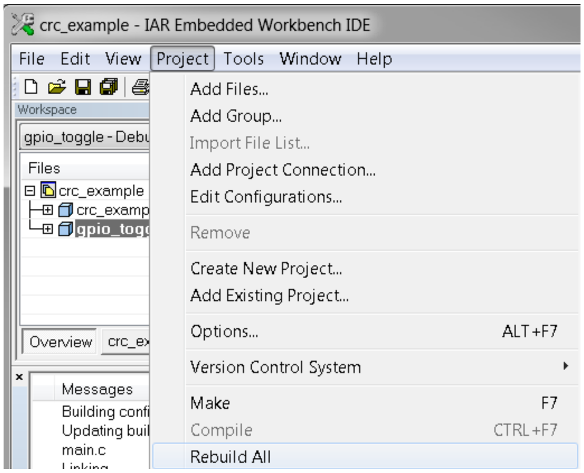
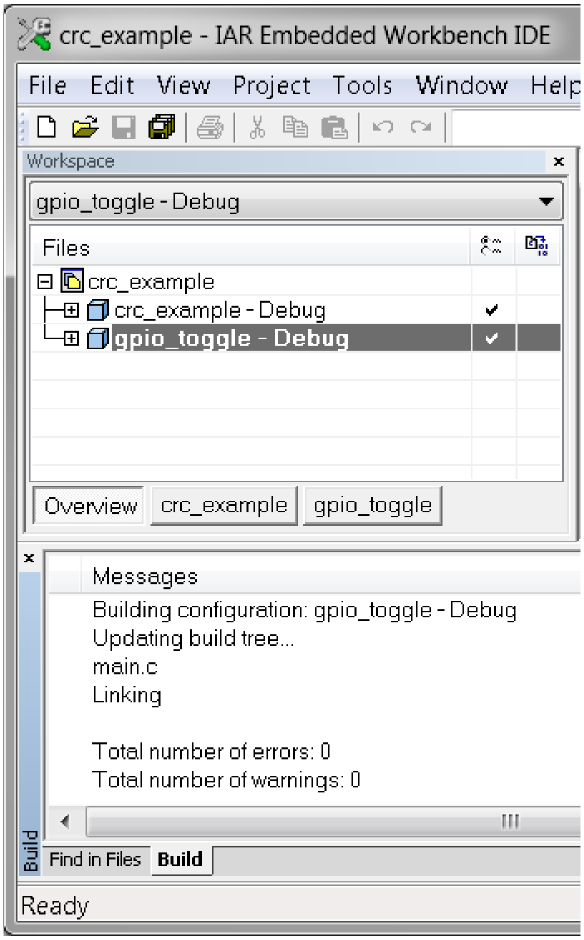


*Figure 18. Building an Application Using **Rebuild All***

In all cases, it is possible to compile code without errors, as shown in Figure 19.



```
Messages
Building configuration: gpio_toggle - Debug
Updating build tree...
main.c
Linking

Total number of errors: 0
Total number of warnings: 0
```

Figure 19. Recompiling Projects

Downloading the Application Code

To download the code, click **Project** > **Download** > **Download active application**.

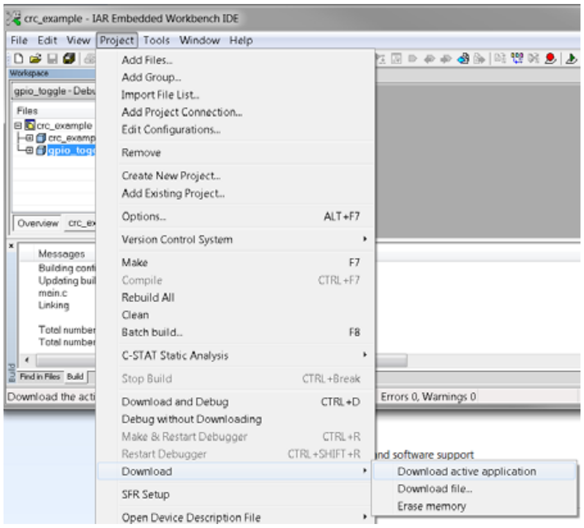


Figure 20. Downloading the Application

After power cycling the device, the code runs on the ADuCM3027/ADuCM3029.

Debugging the Application Code

Use the following procedure to download and debug a project:

1. Click debug. Debugging of the code is executed at the beginning of the main function. The following debugging features are available: single step, step over, and breakpoint.

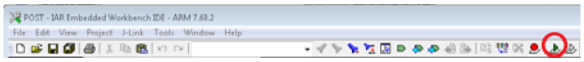

POST - IAR Embedded Workbench IDE - ARM 7.60.2
File Edit View Project J-Link Tools Window Help
15388-020

Figure 21. IAR Debug Toolbar Button

2. Begin debugging by either clicking the go icon in the toolbar as shown in Figure 22, or by navigating to **Debug** > **Go** (or by or pressing F5) as shown in Figure 24. The code then executes on the ADuCM3027/ADuCM3029.

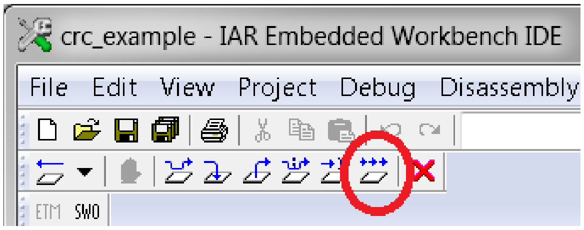

crc_example - IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly
ETM SWO
15388-021

Figure 22. IAR Go Toolbar Button

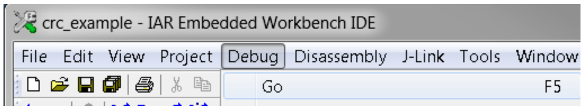

crc_example - IAR Embedded Workbench IDE
File Edit View Project Debug Disassembly J-Link Tools Window
Go F5
15388-022

*Figure 23. **Debug** > **Go***

Running an Example Project

The ADuCM3027/ADuCM3029 BSP has many sample projects to test and evaluate the microcontroller. In this user guide, the ADuCM3029 EZ-Kit evaluation board and IAR workbench are used to run a sample project from the BSP package.

The LED_polled_button example project is used in this section. This example uses the GPIO service to configure the push buttons (the GPIO is configured as an input) and the LEDs (the GPIO is configured as an output) on the ADuCM3029/ADuCM3027 EZ-Kit board.

To execute the LED_polled_button project,

1. Navigate to **Project** > **Add Existing Project…**.

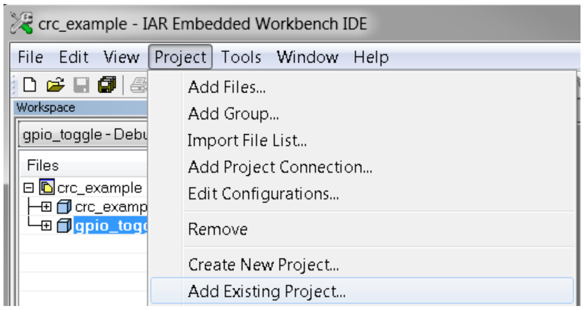


*Figure 24. **Add Existing Project…** Menu Option*

2. In the file selection dialogue box, go to the BSP folder, then go to the **examples** folder. In the **examples** folder, go to **gpio\LED_button_polled\ADuCM3029\iar**. Click the **LED_button_polled.ewp** file and then click the **Open** button (see Figure 25).

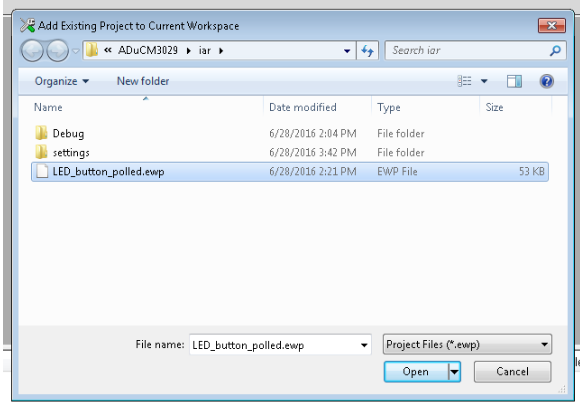

Add Existing Project to Current Workspace
« ADuCM3029 ▸ iar ▸
Search iar
Organize ▾ New folder
Name Date modified Type Size
Debug 6/28/2016 2:04 PM File folder
settings 6/28/2016 3:42 PM File folder
LED_button_polled.ewp 6/28/2016 2:21 PM EWP File 53 KB
File name: LED_button_polled.ewp
Project Files (*.ewp)
Open
Cancel
15388-024

Figure 25. Project Selection Dialogue Box

3. Build the project by clicking the **Project** dropdown menu and navigating to **Make**. There are no error or warning messages on the build console.

4. To debug the application, click the **Project** dropdown menu and navigate to **Download and Debug** or press Ctrl + D (see Figure 26). After downloading the application, the IDE switches to debug mode and displays other windows, such as the **Disassembly** window, the debug toolbar, and the debug logger. Click **Debug** > **Go** to run the application.

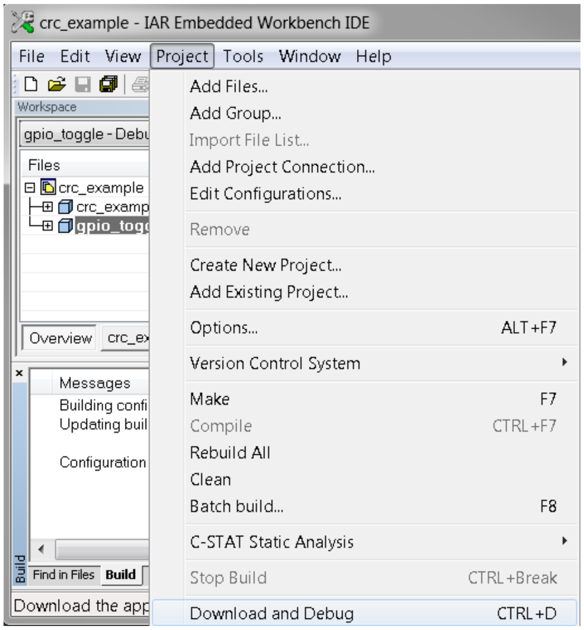


Figure 26. ***Download and Debug***

POWER OPTIMIZATION FOR THE ADuCM3027/ADuCM3029 PROCESSORS

Choosing a low power MCU is a difficult task because it involves reviewing data sheets to analyze electrical specifications. It is often difficult to relate these specifications to applicable, system level use cases.

Evaluating various power modes while considering peripheral operations emulating real use case scenarios is an essential step in choosing the right MCU for a power sensitive application. Key aspects to evaluate when choosing an MCU for low power applications include the following:

- Availability of low power modes and the impact of these modes on the ability to retain the contents of SRAM.
- Power consumption with the RTC running while the rest of the system is in low power mode.
- Wake-up times from low power mode(s).
- Supply voltage range from an application standpoint. The designer can adjust and select the system supply voltage, depending on the component requirements.
- Power consumption in active mode.
- Core activity—example algorithm processing.
- Peripheral activity—DMA operations.
- Simultaneous core and peripheral activity.
- Flexibility in choosing core and peripheral clock frequencies that meet system requirements while keeping the power consumption low.
- Hardware DMA blocks that enable the CPU to be in low power mode during peripheral activity.

The ADuCM3027/ADuCM3029 processors are an Ultra low power, integrated, mixed signal, MCU system for processing, control, and connectivity. The MCU system is based on an ARM® Cortex®-M3 processor, offering up to 33 MIPS of peak performance at 26 MHz, combined with a collection of digital peripherals, embedded SRAM and flash memories, and an analog subsystem that provides clocking, reset, and power management capability in addition to an ADC subsystem.

The ADuCM3027/ADuCM3029 processors are two of the few low power MCUs on the market that offer a cache controller. Programs that repeatedly access the same data or instructions make effective use of cache memory, thereby reducing the overall power consumption.

The power consumption of an MCU largely depends on two factors: the operating voltage and the frequency at which the system operates. The ADuCM3027/ADuCM3029 processors incorporate several power modes that are useful in building battery-powered or self powered (energy harvesting) applications.

This section discusses the power modes of the ADuCM3027/ADuCM3029 processors in detail and provides example power measurements for several scenarios, with the intent of helping developers choose the power modes that best fit low power application requirements.

ADuCM3029/ADuCM3027 PROCESSOR POWER MANAGEMENT

The ADuCM3029/ADuCM3027 processors incorporate a highly customizable power management and clocking system that offers application developers the flexibility to balance power and performance. The power management blocks consist of integrated regulators, a clock gating scheme, and switches applicable to numerous application scenarios.

The power management system features include the following:

- An integrated, 1.2 V LDO and an optional capacitive buck regulator.
- Integrated power switches for low standby current in hibernate mode.
- Power gating to reduce leakage in sleep modes.
- A power supply monitor with a selectable voltage range

ADuCM3029/ADuCM3027 PROCESSOR POWER MODES

The power management system provides the following low power modes:

- Active mode with customized clock gating features.
- Flexi™ mode with smart peripherals.
- Hibernate mode with optional SRAM retention capability.
- Shutdown mode without SRAM retention.

Each mode provides a low power benefit with potential functionality trade-offs. Table 2 summarizes the status of the system blocks in each low power mode.

Table 2. Power Mode System Block States

Functional Block	ARM Cortex-M3 Core	Buck	PERIPHERAL -DMA	HF-XTAL	HFOSC	LFXTAL	PLL	LFOSC	RTC0	RTC1	ADC	SRAM	FLASH
Active Mode	On	User[1]	User[1]	User[1]	User[1]	User[1]	User[1]	On	User[1]	User[1]	User[1]	On	On
Flexi Mode	Off	User[1]	User[1]	User[1]	User[1]	User[1]	User[1]	On	User[1]	User[1]	User[1]	On	On
Hibernate Mode	Off	Off	Off	Off	Off	User	Off	On	User[1]	User[1]	Off	On[2]	Off
Shutdown Mode	Off	Off	Off	Off	Off	User	Off	Off	User[1]	Off	Off	Off	Off

[1] In the user application code, this functional block can be configured to be on or off.
[2] The retainable SRAM size is configurable.

Active Mode

In active mode (also called full on mode), the ARM Cortex-M3 is active and executes instructions from flash memory and/or SRAM. All peripherals can be enabled or disabled at the discretion of the user, and active mode power can be enhanced by optimized clock management.

Several power saving options are available in active mode:

- Using the buck converter.
- Enabling the cache.
- Dynamic clock scaling.
- Clock gating.

Buck Converter

The optional integrated buck converter feature saves power in active mode. The buck converter powers the linear regulator, which powers the digital core domain. The buck converter enters bypass mode after the battery voltage (VBAT) falls below ~2.3 V. After entering bypass mode, the buck converter output follows the input. Figure 27 shows the external circuitry recommended for buck converter enabled designs.

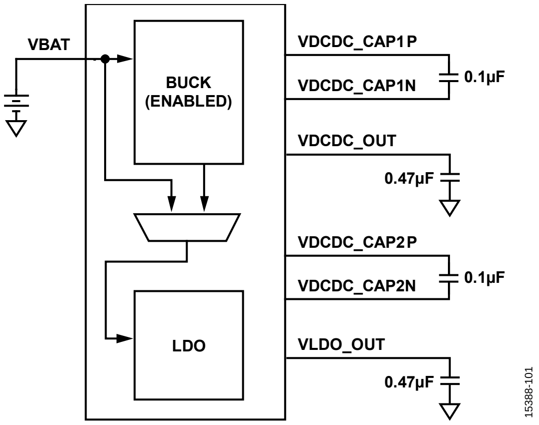


Figure 27. External Circuitry for Buck Converter Enabled Designs

For designs in which the optional buck converter is not used, the VDCDC_CAP1P, VDCDC_CAP1N, VDCDC_OUT, VDCDC_CAP2P, and VDCDC_CAP2N pins must be left unconnected.

The buck converter is solely for processor usage. An external load cannot be connected to the buck converter output.

Enable the buck converter by setting the CTL1.HPBUCKEN bit per the following code:

```
*pREG_PMG0_CTL1 |= (1<<ITP_PMG_CTL1_HPBUCKEN);
```

Figure 28 compares the power consumption of the ADuCM3027/ADuCM3029 processors when computing prime numbers with the following conditions:

- VBAT = 3.0 V
- HCLK = PCLK = 26 MHz
- Cache memory disabled

The buck converter impacts current consumption positively at higher VBAT values. Specifically, there is roughly a 50% decrease in the active current when VBAT ≥ 3 V.

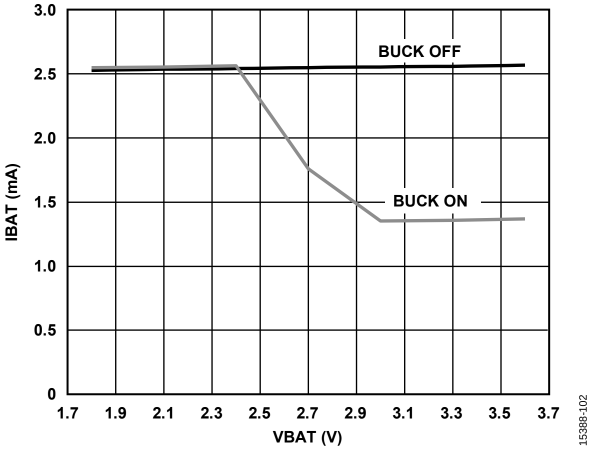


Figure 28. Impact of the Buck Converter on Active Mode Power Consumption Enabling Cache Memory

Cache memory reduces the average time to access data from flash memory. For scenarios where the CPU is required to run an algorithm, or when the same data must be accessed repeatedly, cacheable memory can reduce the power consumption because execution is from the internal instruction SRAM. When the cache controller is enabled, 4 kB of instruction SRAM is reserved as cache memory.

Cache memory is disabled at startup by default. Use the following procedure to enable the cache memory:

1. Read the cache enable status bit (Bit 0 in the CACHESTAT register) to ensure that cache memory is disabled. Poll this bit until it clears.
2. Write the user key to the CACHEKEY register. For example, write

```
*pREG_FLCC0_CACHE_KEY = 0xF123F456;
```

3. Set the instruction cache enable bit (ICEN in the CACHESETUP register) as follows:

```
*pREG_FLCC0_CACHE_SETUP |= (1 <<
BITP_FLCC_CACHE_SETUP_ICEN);
```

Figure 29 compares the power consumption of the ADuCM3027/ADuCM3029 processors when computing prime numbers with the following conditions:

- VBAT = 3.0 V
- HCLK = PCLK = 26 MHz
- Buck converter disabled

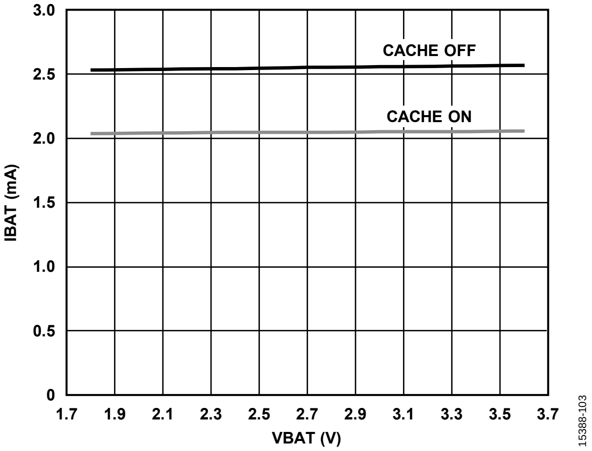


Figure 29. Impact of Cache Memory on Active Mode Power Consumption

Enabling the cache memory reduces the average active current consumption by ~18%.

Dynamic Clock Scaling

Dynamic clock/frequency scaling is a proven method to reduce power consumption. The ADuCM3027/ADuCM3029 processors have a flexible clock architecture that allows dynamic modification of the CPU and peripheral clock frequencies. A combination of clock dividers and a phase-locked loop (PLL) provides flexibility in deriving an optimum system clock frequency that guarantees system performance while keeping the power consumption low, as compared to a fixed clock scheme. Programmable clock dividers are available to generate the clocks in the system, and the divisors can be configured on-the-fly.

Figure 30 plots the power consumption of the ADuCM3027 ADuCM3029 processors when computing prime numbers with the following conditions:

- VBAT = 3.0 V
- HCLK = PCLK (the source of the root clock is HFOSC)
- Buck converter disabled
- Cache disabled

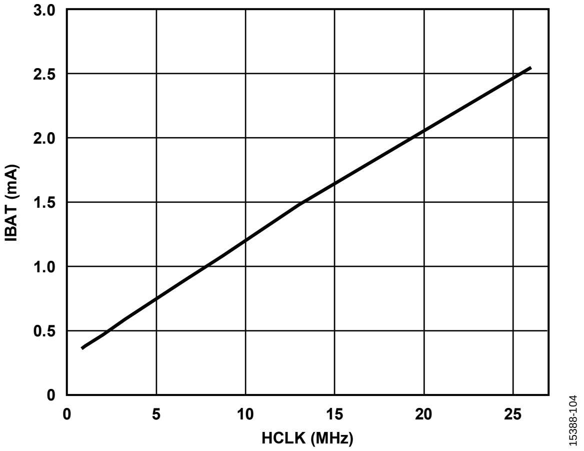


Figure 30. Impact of Core Clock Frequency on Active Mode Power Consumption

As can be expected, power dissipation decreases as core clock frequency decreases.

Clock Gating

The system is heavily clock gated and uses automatic clock gating techniques. Most peripherals are automatically clock gated when the peripheral is disabled, such that the clock is running only when the peripheral is enabled. The exceptions are I^{2}C, GPIO, and the general-purpose timer (GPTMR). These blocks must be manually clock gated using the CLKCON5 register. Gate the peripheral clock completely by setting the CLKCON5.PERCLKOFF bit.

Any access to the clock gated peripherals overrides the clock gate settings in the CLKCON5 register.

For application scenarios where the core is processing data and no peripheral activity is desired, the peripheral clock (PCLK) can be turned off to save power. Figure 31 shows the power consumption of the ADuCM3027/ADuCM3029 processors when computing prime numbers with the following conditions:

- VBAT = 3.0 V
- Buck converter disabled
- Cache memory disabled
- HCLK = PCLK = 26 MHz

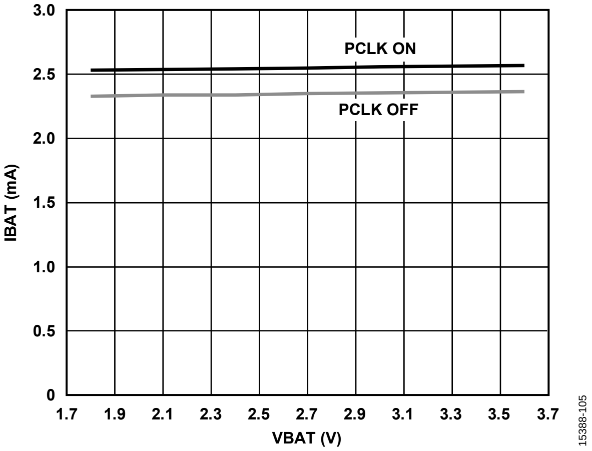


Figure 31. Impact of Peripheral Clock Gating on Active Mode Power Consumption

As shown in Figure 31, a ~0.2 mA reduction in the active current is observed when the peripheral clock is gated.

In active mode, the four techniques described in this section can be combined to achieve maximum power savings.

Flexi Mode

Flexi mode is a flexible sleep mode useful in scenarios where the core must wait for a peripheral data transfer to complete before it can start processing. In Flexi mode, the core is clock gated and the remainder of the system is active. Flexi mode can be used to substantially reduce active power when a very low speed activity is expected to complete (for example, reading a certain number of bytes from a sensor) before the processor must be woken up to process the data.

Consider a scenario where the CPU configures an SPI DMA and must wait for the DMA to complete. Figure 32 shows the power consumption of the ADuCM3027/ADuCM3029 processors transferring data over the SPI using DMA accesses with the following conditions:

- VBAT = 3.0 V
- Buck converter disabled
- Cache disabled
- PCLK = 6.5 MHz
- SPI_DIV = 49

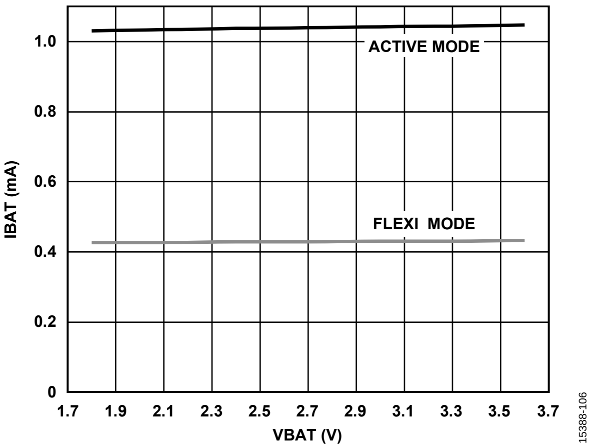


Figure 32. Impact of Flexi Mode on Power Consumption

As shown in Figure 32, there is nearly a 66% savings in power when Flexi mode is used while the DMA is ongoing, rather than keeping the core in active mode.

There are a number of wake-up sources that can be used to exit Flexi mode (for example, DMA interrupts, external interrupts, timer interrupts, and so on), and it typically takes only one CPU clock cycle to exit.

The buck converter can also be enabled in Flexi mode to save additional power. Figure 33 shows the power consumption of the ADuCM3027/ADuCM3029 processor across VBAT in Flexi mode with the buck converter on while transferring data over the SPI using DMA accesses with the following conditions:

- VBAT = 3.0 V
- Cache memory disabled
- PCLK = 6.5 MHz
- SPI_DIV = 49

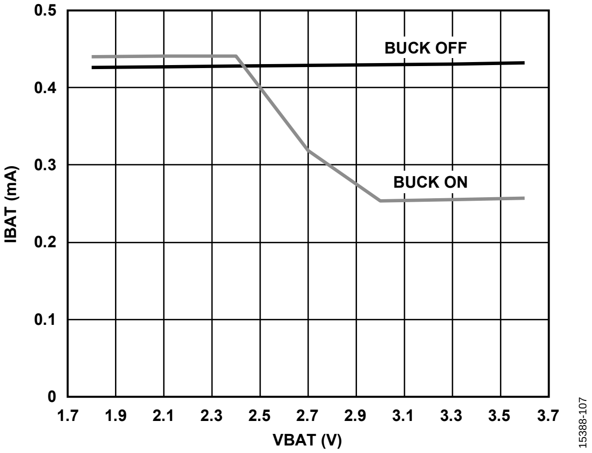


Figure 33. Impact of the Buck Converter on Flexi Mode Power Consumption

As seen in Figure 33, a similar power improvement pattern to the impact of the buck converter in active mode is shown in Figure 28. Specifically, when VBAT ≥ 3 V, a 50% improvement in power is observed.

Hibernate Mode

In hibernate mode, the ARM Cortex-M3 core and all digital peripherals are off with configurable SRAM retention, port pin retention, a limited number of wake-up interrupts, and, optionally, an active RTC. All GPIO pin states are retained in hibernate mode. The ADuCM3027/ADuCM3029 processors also incorporate the SensorStrobe™ mechanism in the RTC block, which enables ultra low power sensor data measurement.

Before entering hibernate mode, most of the enabled peripherals must be programmed to undergo a specific sequence to properly enter or exit hibernate mode, and several system memory map registers (MMRs) and peripheral registers are retained while in hibernate mode. For more details, refer to the relevant peripheral information in the ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference.

Configurable Retainable SRAM

The ADuCM3027/ADuCM3029 processors support SRAM block sizes of 8 kB (default), 16 kB, 24 kB, and 32 kB to be retained while in hibernate mode. The more SRAM that must be retained, the higher the power consumption is while in hibernate mode, as shown in Figure 34.

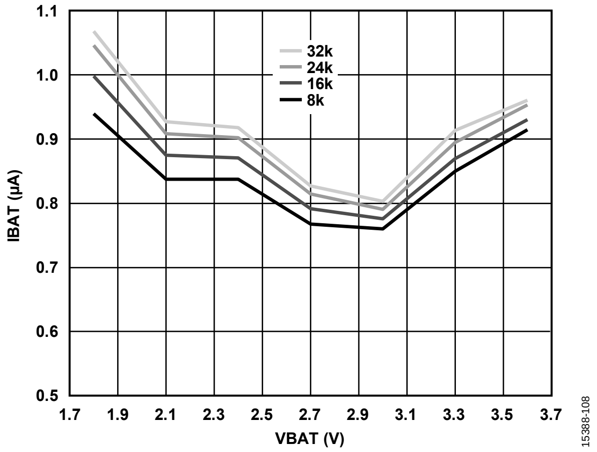


Figure 34. Current on the VBAT Supply Pin (I_{BAT}) for Various Retained SRAM Values

The SRAM retention size can be configured by setting the appropriate bits in the SRAMRET register. For example, to enable 32 kB of SRAM to be retained while in hibernate mode, use the following code in the SRAMRET register:

```
*pREG_PMG_PWRKEY = 0x4859;
*pREG_PMG_SRAMRET |=
((1 << BITP_PMG_SRAMRET_SRAM_RET1_EN)|
(1 << BITP_PMG_SRAMRET_SRAM_RET2_EN));
```

If parity is enabled, initialization of nonretained SRAM regions may be required upon waking from hibernate mode.

Wake-Up Sources

The following events are capable of waking the devices up from hibernate mode:

- External Interrupt 0 to External Interrupt 3
- RTC0 and RTC1 interrupt
- Battery voltage range interrupt
- UART Rx pin activity

Of the two real-time clocks, RTC1 is the recommended wake-up source from hibernate mode because, in an application where both hibernate mode and shutdown mode must be used, only RTC0 can be used for exiting shutdown mode.

The wake-up time from hibernate mode from any of these events is ~14 µs when executing from flash, and ~10 µs when executing from SRAM.

RTC Clock Sources

The ADuCM3027/ADuCM3029 processors offer two clock choices for the RTC1 block:

- A low power internal RC oscillator (LFOSC)
- An external crystal oscillator (LFXTAL)

Choosing to implement either LFOSC or LFXTAL is a trade-off between accuracy and power consumption. LFXTAL is more accurate (depending on the crystal manufacturer) compared to the LFOSC, but LFOSC dissipates less power, as shown in Figure 35.

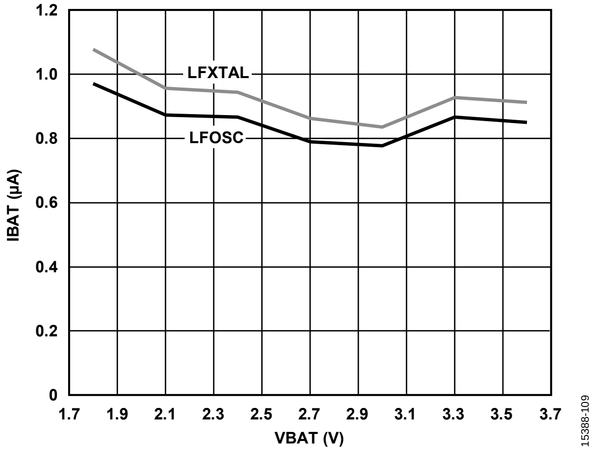


Figure 35. Hibernate Current with RTC1 as the Wake-Up Source (LFOSC vs. LFXTAL)

SensorStrobe

The SensorStrobe mechanism allows the ADuCM3027/ADuCM3029 processor to be used as a programmable clock generator in all power modes, including hibernate mode. In this way, the external sensors can have their timing domains mastered by the ADuCM3027/ADuCM3029 processors because the SensorStrobe output is a programmable divider from FLEX_RTC, which can operate up to a resolution of 30.7 µs. The sensors and microcontroller are synchronous, which removes the need for additional resampling of data to time align the microcontroller and the sensors.

Shutdown Mode

Shutdown mode is the deepest sleep mode, in which digital and analog circuits are powered down. The state of the digital core and the SRAM memory content is not retained; however, the state of the pads is preserved, as is the wake-up interrupt configuration.

The configuration of the pads is preserved and locked after waking up from shutdown mode. The user must unlock the state of the pads by writing 0x58FA to the PGM_TST_CLR_LATCH_GPIOS register, preferably inside the ISR routine:

```
*pREG_PMG0_TST_CLR_LATCH_GPIOS = 0x58FA;
```

Additionally, the user must configure the appropriate wake-up source, choosing from the following options:

- External Interrupt 0 to External Interrupt 2
- External reset
- Battery falling below 1.6 V
- RTC0 timer

The RTC0 block can (optionally) be enabled in this mode, which allows the processor to be periodically woken up by the RTC0 interrupt.

The clock source for RTC0 must be the LFXTAL because the LFOSC is disabled in shutdown mode.

Because the RTC0 block must be powered to serve as a wake-up source, it adds to the power dissipation while in shutdown mode, as shown in Figure 36.

When the device wakes up from shutdown mode, the POR sequence is followed, and code execution starts from the beginning.

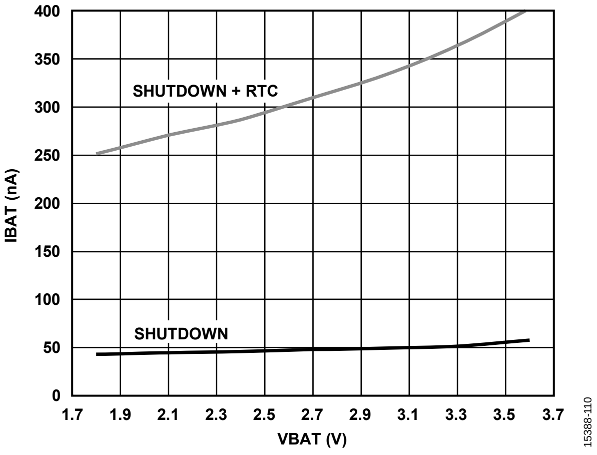


Figure 36. Shutdown Mode Current (External Sources vs. RTC0)

USING THE ADuCM3029/ADuCM3027 PROCESSOR BOOT KERNEL

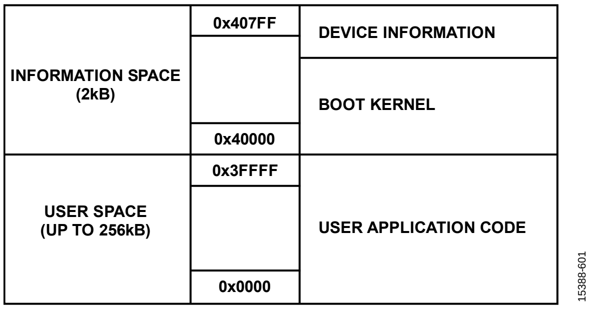


Figure 37. Flash Information Memory Space

The ADuCM3029/ADuCM3027 processors feature integrated flash memory that contains the user application code (user space) and a dedicated 2 kB bank of memory, information space arranged as shown in Figure 37.

Some devices feature 128 kB of user space. Refer to the ADuCM3027/ADuCM3029 data sheet for details.

As shown in Figure 37, the information space block is further broken down into the boot kernel, residing in the upper 2 kB of flash memory, and the device information. The boot kernel is responsible for implementing a secure environment, where user application code can optionally be read and/or write protected, and executes the application from flash memory upon reset. The boot kernel also provides a mechanism to upgrade the firmware through a UART downloader.

This section describes the information space region of the on-chip flash memory, as well as both the boot process and how to use the UART downloader to perform field upgrades to the processor firmware.

BOOT KERNEL OVERVIEW

The boot kernel switches to the user application after performing certain checks (including the CRC integrity of the user application), or the boot kernel enters UART downloader mode to upgrade the user application in flash memory, depending on the SYS_BMODE0 boot mode pin state at reset.

The boot kernel supports in field updates to the user application through the UART port. For security reasons, the boot kernel itself does not provide the flash programming feature; however, it allows the firmware update code, which has flash driver code for updating the user flash, to be downloaded to the device over the UART port. This code is referred to as a second stage loader (SSL) and is run from SRAM. The SSL must be authenticated before it can be provided run access. The security scheme implemented is discussed in the following sections, describing the critical part of the kernel to provide the secure environment in which the user code can be read and/or write protected, allowing intellectual property security.

The serial download capability allows developers to reprogram the device while it is soldered directly onto the target system, avoiding the need for an external device programmer and removing the need to swap the device out of the system. The serial download feature also enables system upgrades to be performed in the field, provided the hardware infrastructure involving the SYS_BMODE0 pin and the UART port are implemented on the target board.

Configuring Security Options

The boot kernel provides the flexibility to configure the security options of the device by allowing the user to program certain keys and parameters in predefined locations in Page 0 of the user flash memory. The kernel provides the user code security and integrity, which depends on the number of user-defined parameters in the first page of the user flash memory. Table 3 summarizes the list of keys and parameters, as well as their locations in the user flash memory.

Table 3. List of Keys and Parameters

Address Range	Size	Description
0x0000_0180 to 0x0000_018F	128 bits (16 bytes)	Read protection key hash
0x0000_0190 to 0x0000_0193	32 bits (4 bytes)	CRC of read protection key hash
0x0000_0194 to 0x0000_0197	32 bits (4 bytes)	Length of user boot loader or entire user code (used for CRC verification before boot)
0x0000_0198 to 0x0000_019B	32-bit word	In-circuit write protection if set to NOWR
0x0000_019C to 0x0000_019F	32-bit word	CPU write protection of individual flash blocks

Read Protection Key Hash

Program the 128-bit read protection key hash at Address 0x00000180 in the first page of the user flash memory. The value of this hash depends on the kind of security desired in the system because this security defines the read accessibility to the device. The key hash defines the state of the serial wire debugger (SWD), as well as the access permission of the SSL downloaded for upgrades via the UART:

- The reset state of the flash memory of all logic high memory cells (along with a valid key hash CRC) indicates that the user does not desire read protection. In this case, the SWD interface is automatically enabled during booting.
- Any nonreset value results in the SWD being locked; therefore, there is no SWD access to the device.
- The key hash is the 128-bit, truncated secure hash algorithm (SHA-256) of the user key (which is 128 bits in length), which can be sent along with the SSL during the UART download phase. If the user key is valid and the hash of the received key matches the key hash stored, the SSL runs with all permissions. If the user key fails the key hash check, the SSL only has write permission to the user flash.

Key Hash CRC

The key hash has a 32-bit CRC checksum stored at Address 0x00000190. The key hash is valid only if the associated 4-byte checksum is valid. The key hash has a separate key hash to protect it against flash tempering attacks. For all practical purposes, the user must ensure that a valid CRC for the key hash is stored along with the key hash itself.

In-Circuit Write Protect Key

The 32-bit in-circuit write protect key at Address 0x00000198 of the user flash memory prevents in-circuit programming of the device.To disable in-circuit reprogramming, program the hexadecimal value of the NoWr ASCII string (without the terminating null character) to this address. In this case, SWD access to the device is locked, and the only way to update the device code is via the UART downloader.

Use in-circuit write protection along with read protection (providing both read and write protection). In-circuit write protection alone does not have any significance.

Write Protection

Pages can be locked to prevent code from accidentally erasing and reprogramming critical flash memory blocks (such as the user code boot loader). There is a hardware register in the flash controller that disables the programming of pages grouped into blocks. This register is not automatically loaded via the hardware; rather, it is written to via the kernel. The kernel reads the write protection word from the user flash address, Address 0x0000019C, and writes it to the write protection register in the flash controller. The user can write the appropriate word to this location, depending on the pages that are intended to be protected against accidental writes. The pages are protected in groups of four, with each bit in the 32-bit word corresponding to four continuous flash pages. Refer to the ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference Manual for details.

User Code Length

There is a 32-bit value stored at a flash memory address, Address 0x00000194, that defines the CRC protected user code length. The value programmed in this field defines the page number of the user flash memory up to which the CRC protection is desired by the user. The value of N means that CRC protection is desired from Page 0 to Page N of the flash memory, protecting a total of N + 1 pages.

Valid values for the field in Address 0x00000194 are 0 to 127 for the 256 kB ADuCM3029 processor, and 0 to 63 for the 128 kB ADuCM3027 processor. Any value outside this range is treated as invalid and results in a CRC check failure.

User Code CRC

The user code CRC is stored at the end of Page N. The 32-bit CRC (MSB first) with a polynomial of 0x4C11DB7 is expected by the kernel. If the page number is N, the CRC is expected to reside at flash memory address (N × 0x800) + 0x7FC. For example, if N = 5, a total of six pages are CRC protected, and the CRC is stored at Address 0x2FFC. There is an option to disable the CRC check by programming 0xFFFFFFFF to the expected CRC location. After the kernel sees this value in the CRC location, it skips the CRC check.

Boot Code Flow

This section describes how the kernel operates, based on the user programmable parameters described previously. Figure 38 shows a flowchart of the boot code.

After reset, the boot kernel inspects all the parameters stored in Page 0 of the user flash memory. The user has not requested the read protection if the user read protection key hash is not programmed (meaning that it is set to all FFs) and the key hash CRC is valid. As such, the SWD is enabled; however, flash access may be protected, depending on the state of the user code CRC.

If the CRC is valid, access to user flash memory is unrestricted.

If the user disables the CRC by programming 0xFFFFFFFF to the CRC location, access to the user flash memory is also unrestricted.

If the CRC is invalid, the user flash memory is protected with no read or write access allowed. Only the flash mass erase command is allowed. In this case, user code execution is not allowed.

Take care to program a valid CRC (or 0xFFFFFFFF) in the defined CRC location; otherwise, the user flash memory is read protected by the kernel. In this case, flash-based applications fail to load unless the user flash memory is mass erased.

If the user read protection key hash is programmed with a nonreset value, meaning that the read protection is enabled, or if the key hash CRC is invalid, the SWD is disabled by the kernel and SWD access to the device is not possible. Perform this programming only after product development is complete and SWD access is not intended in the field. However, when read protection is enabled, the SWD is opened up only if CRC protection is enabled and the CRC is corrupted, which allows device recovery when the CRC is accidentally corrupted. In this case, the SWD is opened up, but the user flash memory is protected with no read, write, or page erase accessibility (to maintain the user code confidentiality while allowing device recovery). However, mass erase is still possible, which results in the user flash memory being open again (with read and write access).

The user flash memory (user space) is completely blank when shipped; therefore, none of the security keys and parameters are programmed. Therefore, most of the parameters, such as the key hash CRC and the user code length, have invalid values, meaning the parameters are all set to 0xFF. In this case, the kernel performs a check of the user flash memory to identify if it is blank or completely unprogrammed. If the user flash is blank, the kernel skips all the checks and opens up the SWD. In addition to opening up the SWD, which allows users to connect to the device through SWD for their development, the boot kernel also enters the UART downloader mode and awaits reception of the SSL.

After the device is programmed via the SWD, the user flash memory is no longer blank, and the kernel relies on the state of the SYS_BMODE0 pin to decide if the user code must be executed (after performing all the previously explained checks), or if the boot kernel must enter the UART downloader mode.

If the SYS_BMODE0 pin is asserted (low), the kernel enters the UART downloader and waits for the SSL to be downloaded.

If the SYS_BMODE0 pin is deasserted (high), the kernel jumps to the user reset vector in the user flash memory after performing all the security checks.

The only case where the kernel enters the UART download mode without sampling the SYS_BMODE0 pin is when the user flash memory is blank.

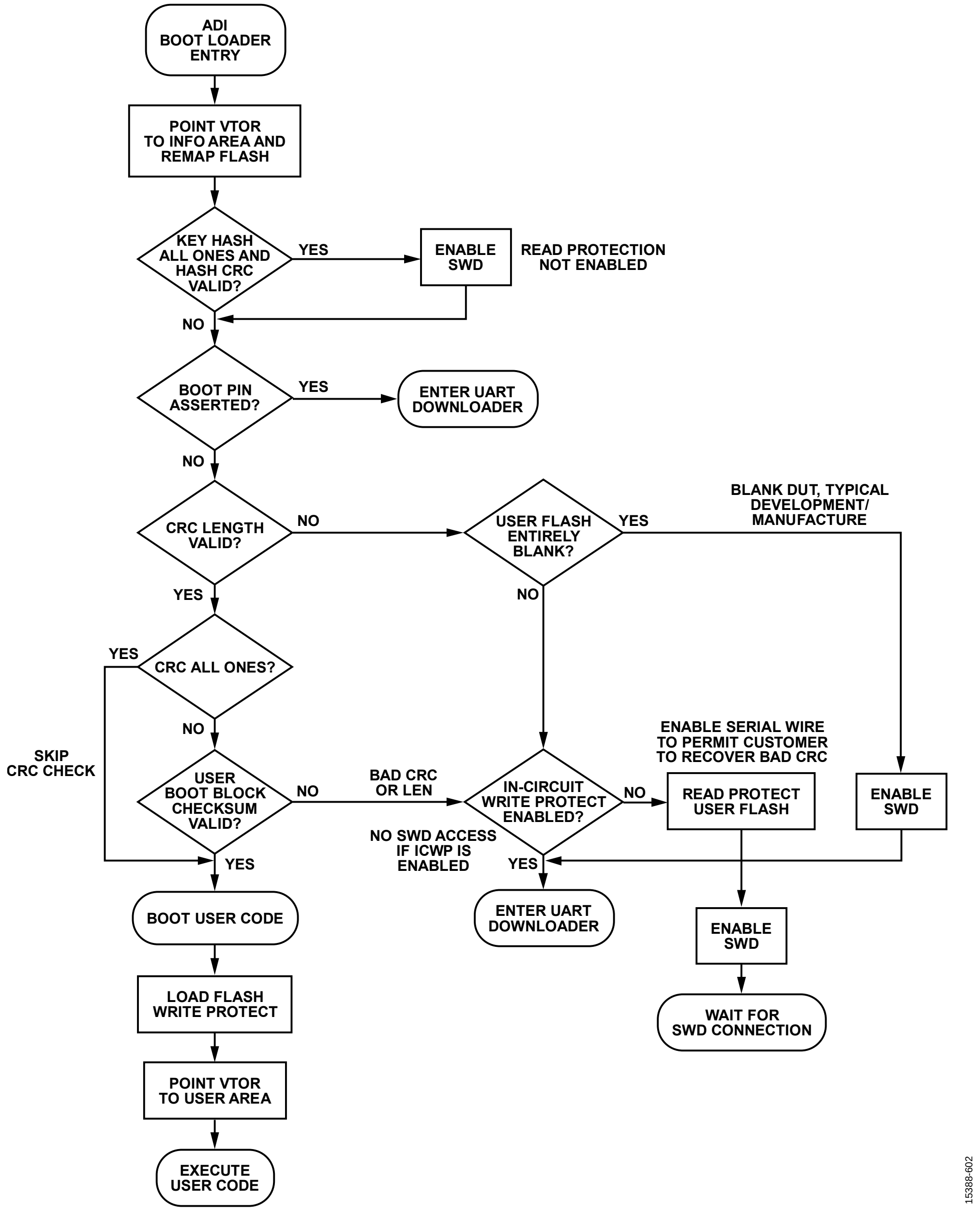


Figure 38. Boot Kernel Flowchart

SSL code that is downloaded over the UART must be mapped to the SRAM. In UART downloader mode, the SSL is loaded to the SRAM and has flash programming capabilities. The kernel authenticates the SSL and allows execution only if authentication is successful. This code is responsible for downloading and upgrading the actual firmware (for example, the user application) in the user flash memory. The kernel does not support direct updates to the user flash memory; therefore, the SSL is required to perform such actions.

The kernel follows a specific protocol to download the SSL to the processor, which must be adhered to by the transmitting host. If the SSL follows the same packet protocol as the kernel, the host interface is simplified (for example, communication with the kernel and the SSL is uniform). The details of the protocol are discussed in the following sections.

UART DOWNLOADER

The ADuCM3027/ADuCM3029 processors enter UART downloader mode if the SYS_BMODE0 pin (GPIO17) is pulled low. If this condition is detected by the device at power-on or hard reset, the device enters serial download mode. In this mode, an on-chip loader routine in the kernel is initiated, which configures the UART port of the device and, via a specific serial download protocol, communicates with a host to manage the firmware upgrade process. Figure 39 shows the UART downloader flow.

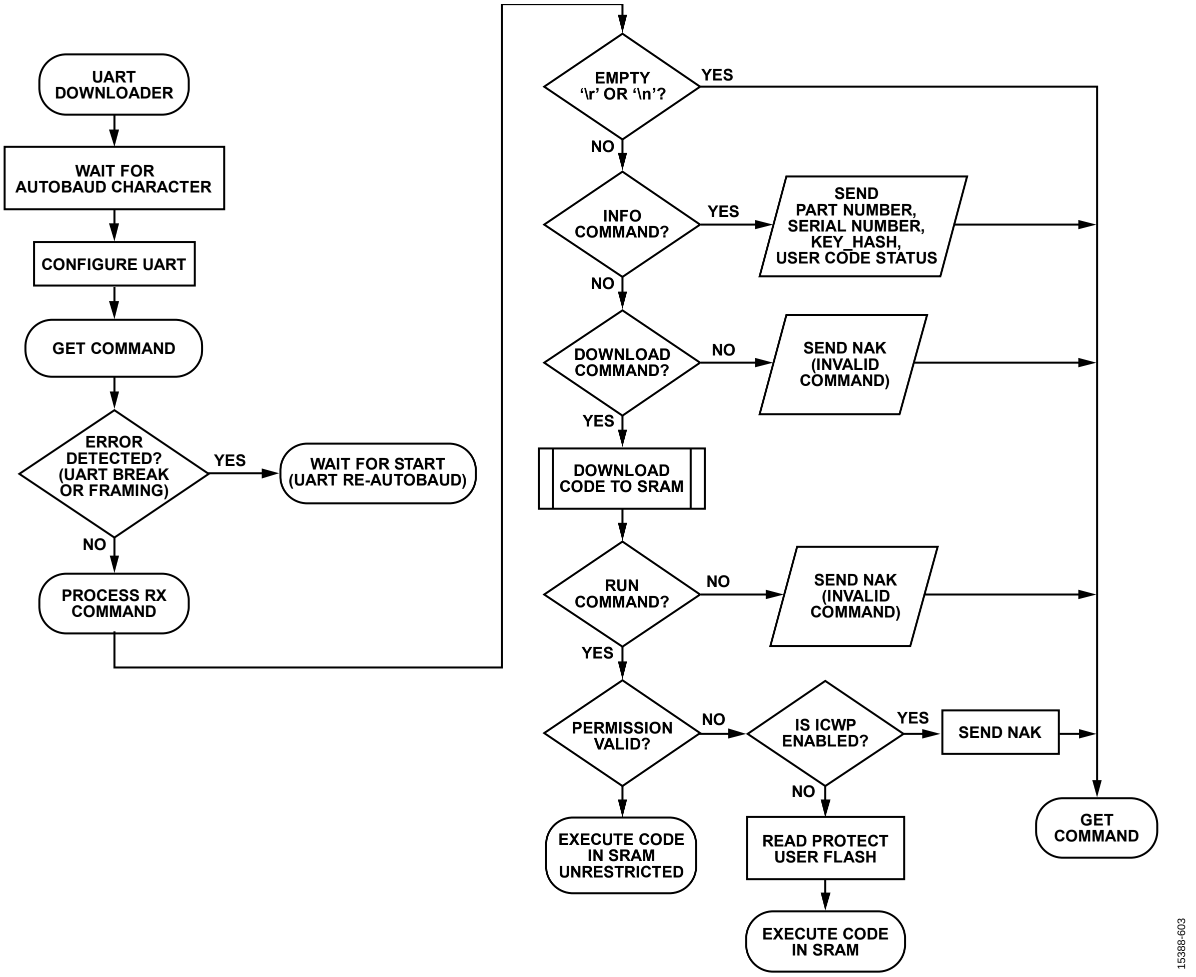


Figure 39. UART Downloader Flowchart

Protocol

After the serial downloader is triggered by asserting the SYS_BMODE0 pin, the kernel waits for the host to send a carriage return character (ASCII 0x0D, as shown in Figure 40) to initiate the UART autobaud process.

The kernel makes use of the UART autobaud feature to detect the baud rate of the host and to subsequently configure the UART port to transmit or receive, at thebaud rate of the host, with eight data bits and no parity. Due to the 6.5 MHz reset peripheral clock (PCLK), the UART can be configured by the kernel to support baud rates up to 230,400 bps. Baud rates greater then this value contain more errors and may result in an unreliable data transfer. However, after loading the SSL, higher baud rates are possible if the SSL increases the PCLK (up to 26 MHz) and performs a second autobaud detection via the UART.

After receiving the autobaud character, the kernel calculates the required clock divisor values and configures the UART, at which point the kernel sends the device information as part of a 57-byte ID data packet, as shown in Table 4, to acknowledge that the autobaud detection process is successful.

Packet Structure

In addition to indicating to the host that the processor is now ready to communicate, the autobaud acknowledgement also contains information about the device, the state of the user flash memory, and security restrictions. After the autobaud acknowledgement, the data transfer itself can begin, as governed by the communications data transport packet format shown in Table 5.

Packet Start ID Field, ID0 and ID1

The first transfer field is the two-byte packet start ID field (ID0 and ID1), comprised of two start characters (0x07 for ID0 and 0x0E for ID1). These bytes are constant and are used by the loader to detect the beginning of a valid data packet.

Number of Data Bytes Field

The next transfer field is the total number of data bytes field, which includes the 1-byte command (CMD), the 4-byte address (value), and the remaining payload (data). The minimum number of data bytes is five, which corresponds to a command and address only. The maximum number of data bytes is 255, supporting a command, an address, and up to 250 bytes of data.

Command Function Field (CMD), Data Byte 1

The command function field describes the function of the data packet. Three commands are supported by the kernel, represented in ASCII format:

- W (0x57)—write command
- R (0x52)—run command
- I (0x45)—information command

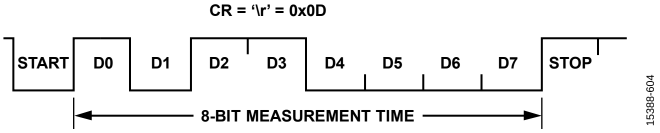


Figure 40. Autobaud Character

Table 4. Autobaud Response

Bytes	Contents
1 to 15	Product identifier: ADuCM302xand six spaces, where x = 7 or 9
16 to 18	Hardware and firmware version numbers
19	User code blank; x means the code to execute, and a dash (-) means the user code is blank
20	User code checksum; P means that the checksum passed, and F means that the checksum failed
21	Write protection enabled; W means that write protection is disabled, and a dash (-) means that the write protection is enabled
22	Read protection enabled; R means that read protection is disabled, and a dash (-) means that read protection is enabled
23	Space
24 to 55	128-bit serial number, as a 32-digit uppercase hexadecimal number (for example, 0123456789ABCDEF0123456789ABCDEF)
56	Line feed
57	Carriage return

Table 5. UART Packet Structure

ID0	ID1	Number of Data Bytes	CMD	Value	Data	Checksum
0x07	0x0E	5 to 255	W, R, or I	h, u, m, l	xx	CS

Write Command

The write command packet shown in Table 6 includes the number of data bytes (5 + n, where n is the payload size in bytes), the write command (W), the 32-bit start address to write to, and the n data bytes in the payload.

When a write command packet is received by the kernel, the payload bytes are placed sequentially in the SRAM as they arrive, beginning at the start address. The kernel sends a no acknowledge command if the checksum is incorrect or if the received address is out of range. If the host receives a no acknowledge from the loader, abort and restart the download process.

Run Command

After the host transmits all the data packets to the kernel, it can send a final packet instructing the kernel to start executing code. This final packet is achieved by sending the run command packet, which is comprised of the run command (R) and the 32-bit address to begin running from, as shown in Table 7.

When the kernel receives a run command packet, it jumps to the address supplied in the packet only after the permission checks pass.

Information Command

The host can send the information command packet shown in Table 8 at any time; this packet is comprised of the command (I) and a 32-bit address. Though the value field is required for the packet to be properly received by the kernel, the content of the value field is irrelevant.

When the kernel receives the information command packet, it responds with the 57-byte ID packet (see Table 9).

Table 6. Write Command Packet

ID0	ID1	Number of Data Bytes	CMD	Value	Data	Checksum
0x07	0x0E	5 + n	W (0x57)	Start address	n bytes	CS

Table 7. Run Command Packet

ID0	ID1	Number of Data Bytes	CMD	Value	Checksum
0x07	0x0E	5	R (0x52)	Start address	CS

Table 8. Information Command Packet

ID0	ID1	Number of Data Bytes	CMD	Value	Checksum
0x07	0x0E	5	I (0x52)	0xXXXXXXXX	CS

Value Field (Data Byte 2 to Data Byte 5)

The value field contains a 32-bit address that includes h, u, m, and locations. The MSB is in the h location (Data Byte 2) and the LSB is in the l location (Data Byte 5). As described previously,

- In a write command packet, the value field indicates the start address in memory to which the data payload is written.
- In a run command packet, the value field indicates the address in SRAM where the SSL code begins.
- In an information command packet, the value field has no meaning.

Data Field (Data Byte 6 Data Byte 255)

User code is downloaded one byte at a time, and the data field can contain a maximum of 250 bytes. The data is normally stripped out of the Intel® HEX extended 16-byte record format, reassembled by the host, and then sent in packet form using a series of write command packets to the ADuCM3027/ ADuCM3029 processors.

Checksum Field (CS)

The data packet checksum is written to the checksum field. This twos complement checksum is calculated from the summation of the hexadecimal values that span the number of bytes field to the end of the data field. Thus, the 8-bit LSB of the sum of all the bytes in the packet from the number of data bytes field, up to and including the checksum field, is 0.

Acknowledge of Command

The loader routine issues a no acknowledge command (0x07) as a negative response, or an acknowledge (0x06) as a positive response to each data packet received.

The loader transmits a no acknowledge if it meets any of following conditions:

- The loader receives an incorrect checksum.
- A UART framing or break error occurs (this error may not reach the host if the UART link is invalid).
- The SRAM code verification fails.

If any one of these conditions is met, it is required to reset the target and restart the firmware upgrade process. If none of these conditions are met, an acknowledge command is transmitted.

READ PROTECTION KEY AND HASHING

The read protection key can be used to allow access to the device during failure analysis. If the device is read protected and failure analysis of the current flash memory content is necessary, enable the SWD interface by sending the key corresponding to the hash stored in the user flash memory. It is recommended that the key be unique to the device and be based on the unique identifier of the device (for example, serial number stored in the information space).

A hash is stored in the user flash memory after the interrupt vectors. This is the hash of a secret customer key. It is strongly recommended that this key be unique to the device for security reasons, and that the unlock key is valid for that one specific device. To maintain a unique key per device, there must be a device identifier to associate which key belongs to a particular device. For simple key management, it is advised to make the key a hash of a master secret and the device identifier. For example,

Read Protection Key = *Hash* (*Master Secret* || *Unique Device Identifier*);

Key Hash = *Hash* (*Read Protection Key*)

When the kernel is in UART loader mode, it can accept the read protection key. Then, the boot loader performs a hash of the read protection key and compares it to the stored key hash. Upon a successful match, the bootloader permits the downloaded SSL code in the SRAM to be executed with all the permissions enabled. If the key hash check fails, then the kernel checks the ICWP key in the user flash memory. If ICWP is turned off by the user by programming any value to Address 0x00000198 other than the hexadecimal equivalent of the NoWr ASCII string, then the SSL is allowed to run after protecting the flash against read and write accesses. In this case, the SSL must first issue a mass erase of the user flash memory before attempting to perform any access to the user flash memory space. If ICWP is also enabled by the user, then the SSL is not granted permission to run unless the key hash authentication passes.

The 128-bit read protection key is passed as a part of the SRAM code. This key must be stored in big endian format in the SRAM as a data payload starting at Address 0x20000180, and must be oriented is a specific fashion in the memory for the kernel to parse it correctly. Specifically, if the read protection key is represented as ABCDEFGHIJKLMNOP, where each letter represents one byte (with A being the first byte and P being the last byte), the required arrangement of the bytes in memory is shown in Table 9.

Table 9. Read Protection in SRAM

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x20000180	D	C	B	A
0x20000184	H	G	F	E
0x20000188	L	K	J	I
0x2000018C	P	O	N	M

For example, if the read protection key is 0x00010203040506070-8090A0B0C0D0E0F, then Table 10 shows how the memory must be written.

Table 10. Example Read Protection Key in SRAM

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x20000180	0x03	0x02	0x01	0x00
0x20000184	0x07	0x06	0x05	0x04
0x20000188	0x0B	0x0A	0x09	0x08
0x2000018C	0x0F	0x0E	0x0D	0x0C

The kernel computes the SHA-256 hash of this key, truncates it to a 128-bit hash, and then compares it to the hash stored in Page 0 of the user flash memory at Address 0x00000180. The user must store the 128-bit truncated hash of the key to the flash memory using a similar pattern. The SHA-256 hash for the example key shown in Table 10 is 0xBE45CB2605BF36BEBDE68-4841A28F0FD43C69850A3DCE5FEDBA69928EE3A8991, which means the 128-bit truncated hash that must be stored properly to the user flash memory space is 0x43C69850A3DCE-5FEDBA69928EE3A8991, arranged as shown in Table 11.

Table 11. Example Read Protection Key Hash in Flash Memory

Address	Byte 0	Byte 1	Byte 2	Byte 3
0x00000180	0x50	0x98	0xC6	0x00
0x00000184	0xFE	0xE5	0xDC	0xA3
0x00000188	0x28	0x99	0xA6	0xDB
0x0000018C	0x91	0x89	0x3A	0xEE

The CRC32 of the key hash is calculated with a polynomial of 0x04C11DB7 and a seed value of 0xFFFFFFFF, and it is stored in LSB format in the flash memory space at Address 0x00000190.

MEMORY CONFIGURATION

Table 12 summarizes the different keys and parameters stored in Page 0 of the user flash memory, the associated addresses, and the values programmed to Page 0 when creating a project with the default startup file.

Table 12. Page 0 Memory Configuration

	Address Range		Size		
Content	Start Address	End Address	(Bytes)	Section Name	Default Content
Vector Table	0x0000_0000	0x0000_017F	384	.intvec	Vector table
Read Protection Key Hash	0x0000_0180	0x0000_018F	16	ReadProtection KeyHash	0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF 0xFFFFFFFF
CRC of Read Protection Key	0x0000_0190	0x0000_0193	4	CRC_ReadProtection KeyHash	0xA79C3203
Number of Pages the CRC Computes	0x0000_0194	0x0000_0197	4	NumCRCPages	0
Checksum	0x0000_07FC	0x0000_07FF	4	Checksum	Checksum of 0 to 0x7FB (if enabled in tools by the user)
Page 0 User Memory	0x0000_01A0	0x0000_07FC	1628	Page0_region	User application

HANDLING CRC IN THE IAR WORKBENCH

Calculate the CRC from part of the application image to be loaded into the first several pages of the flash memory. Store the page number of the last page involved in the CRC calculation at Address 0x194 as a 32-bit integer. For example, if only Page 0 is involved in the CRC calculation, store the value of 0x00 at Address 0x194. If the CRC is calculated for the first three pages, the value must be 0x02.

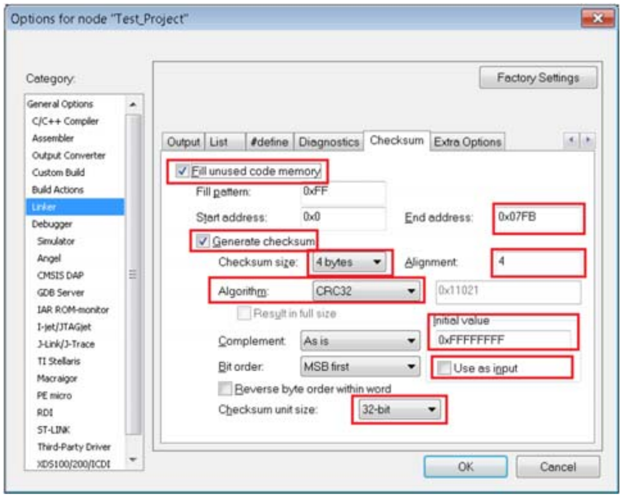
When the CRC is calculated, the last four bytes of the last page included in the CRC calculation are excluded. These four bytes are used for storing the CRC value itself. For example, if the last page is Page 0, the CRC is calculated from Address 0x000 up to and including Address 0x7FB. The tool stores the calculated CRC value at Address 0x7FC as a 32-bit integer.

The standard CRC calculation is CRC32 with a polynomial of 0x04C11DB7, stored in MSB first format, with an initial value of 0xFFFFFFFF. The unit size is 32 bits, which means the tool must read 32 bits at one time from the image when calculating the CRC.

Checksum Tab

There is a **Checksum** tab under the **Linker** category in the IAR tools, which can be used to generate the CRC of the user application code. To store the correct CRC, the following settings must be used (see Figure 41):

- Check the **Fill unused code memory** box.
- Set the **End address:** field to 0x7FB (this value changes depending on the page number).
- Check the **Generate checksum** box.
- Select the **4 bytes** option from the **Checksum size:** pull-down menu
- Set the **Alignment:** field to 4 (which indicates 4 bytes).
- Select the **CRC32** option from the **Algorithm:** pull-down menu.
- Set the **Initial value** field to 0xFFFFFFFF and ensure that the **Use as input** box is not checked.
- Select the **32-bit** option from the **Checksum unit size:** pull-down menu.



☑ Fill unused code memory
Fill pattern: 0xFF
Start address: 0x0
End address: 0x07FB
☑ Generate checksum
Checksum size: 4 bytes
Alignment: 4
Algorithm: CRC32
0x11021
□ Result in full size
Initial value
0xFFFFFFFF
Complement: As is
Bit order: MSB first
□ Use as input
□ Reverse byte order within word
Checksum unit size: 32-bit

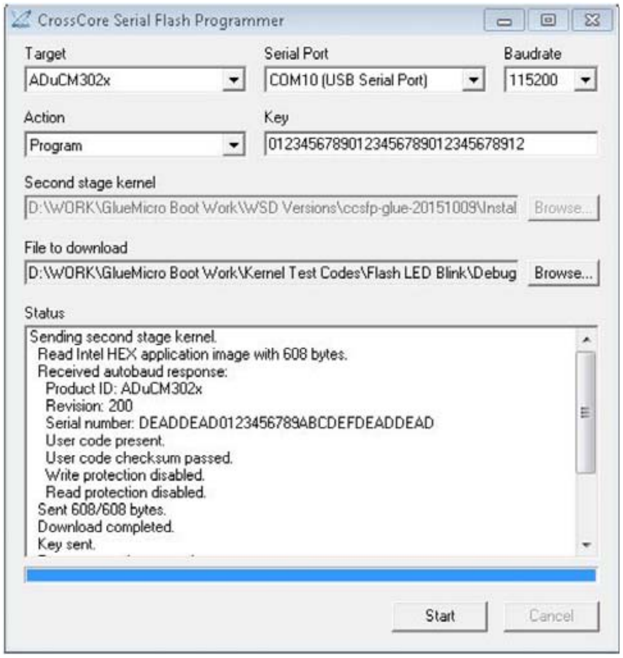
Figure 41. Checksum Settings

CrossCore Serial Flash Programmer

The CrossCore® Serial Flash Programmer (CCSFP) is a PC-based host utility, provided by Analog Devices, that can be used to upgrade the user code over the UART port. CCSFP provides a graphical user interface (GUI) to provide the following options for the UART upgrade:

- Target processor
- UART PC port number
- Baud rate
- SSL hexadecimal file to be used for the upgrade
- User application hexadecimal file to be upgraded
- Key to authenticate the SSL

Figure 42 shows the GUI for the CCSFP. The user must provide the SSL in the **Second stage kernel** field, which is first downloaded into the SRAM of the processor and then is executed before the user application in the **File to download** field is sent to flash, based on the authentication. The 128-bit key for the authentication can be entered in the **Key** field.



Target: ADuCM302x
Serial Port: COM10 (USB Serial Port)
Baudrate: 115200
Action: Program
Key: 01234567890123456789012345678912
Second stage kernel: D:\WORK\GlueMicro Boot Work\WSD Versions\ccsfp-glue-20151009\Instal
File to download: D:\WORK\GlueMicro Boot Work\Kernel Test Codes\Flash LED Blink\Debug

Status

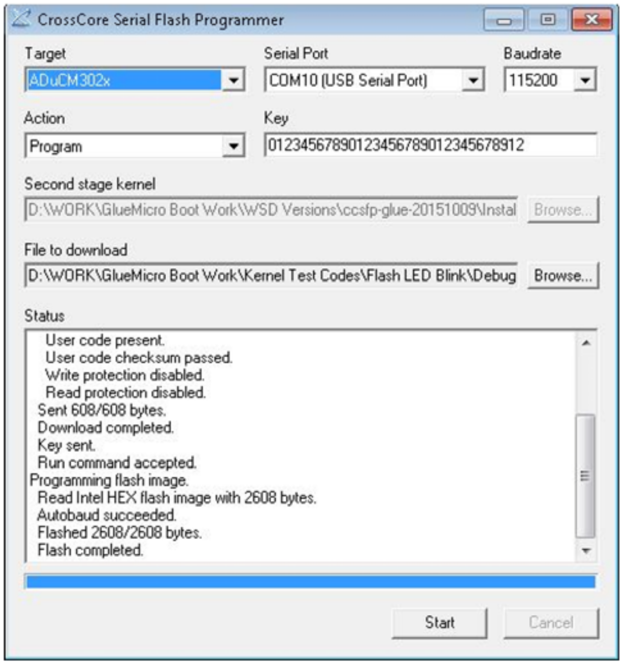
```
Sending second stage kernel.
  Read Intel HEX application image with 608 bytes.
  Received autobaud response:
    Product ID: ADuCM302x
    Revision: 200
    Serial number: DEADDEAD0123456789ABCDEFDEADDEAD
    User code present.
    User code checksum passed.
    Write protection disabled.
    Read protection disabled.
  Sent 608/608 bytes.
  Download completed.
  Key sent.
```

15388-610

Figure 42. CrossCore Serial Flash Programmer GUI

The **Status** window shows the state of the UART download process, device related information, and the status of the commands as returned by the kernel. As shown in Figure 42, the **Status** window shows the device information sent by the kernel, showing the product ID, serial number, and user code status. After the SSL is downloaded, as indicated by the **Download completed** message displayed in the **Status** window, the SSL is then authenticated by the kernel and the actual user application is sent.

Figure 43 shows the SSL executing on the device, receiving the user application, and writing it to the user flash memory space.



Target: ADuCM302x
Serial Port: COM10 (USB Serial Port)
Baudrate: 115200
Action: Program
Key: 01234567890123456789012345678912
Second stage kernel: D:\WORK\GlueMicro Boot Work\WSD Versions\ccsfp-glue-20151009\Instal
File to download: D:\WORK\GlueMicro Boot Work\Kernel Test Codes\Flash LED Blink\Debug

Status

```
    User code present.
    User code checksum passed.
    Write protection disabled.
    Read protection disabled.
  Sent 608/608 bytes.
  Download completed.
  Key sent.
  Run command accepted.
Programming flash image.
  Read Intel HEX flash image with 2608 bytes.
  Autobaud succeeded.
  Flashed 2608/2608 bytes.
  Flash completed.
```

15388-611

Figure 43. User Application Code Being Written by the SSL to User Flash Memory via the CCSFP

CACHE MEMORY IN THE ADuCM3027/ADuCM3029

The memory in the ADuCM3027/ADuCM3029 has up to 256 kB of embedded flash memory within the ECC, a 32 kB data static random access memory (SRAM) with parity, and 32 kB user configurable instruction and data SRAM with parity. Four kilobytes of SRAM can be used as cache memory to reduce active power consumption by reducing access to the flash memory.

The cache in the ADuCM3027/ADuCM3029 consists of a lower power cache controller for the instruction code (ICODE) and data code (DCODE) accesses, a 4 kB instruction cache with 2-way associativity, and a line size of 256 bits. The instruction cache has a least recently used replacement policy. The cache writes to flash, and the core can issue writes to the flash only through the advanced peripheral bus (APB) interface of the flash controller. If the code is placed in the flash, enabling the cache helps the speed of execution. For more information, see the Effects of Cache on the Speed of Execution section. For details on current consumption, see the Current Consumption Comparison section.

This section highlights the use of an on-board cache controller to use a portion of the SRAM as instruction and data cache for user code that otherwise executes from the flash memory.

BLOCK DIAGRAM

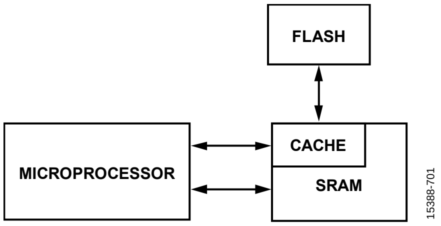


Figure 44. Block Diagram

The ADuCM3027/ADuCM3029 cache architecture consists of a digital cache controller, a cache memory implemented as part of the system SRAM, a digital flash controller, and a flash memory. The cache architecture decreases the average latency of instruction and data accesses by utilizing the faster SRAM memory, and decreases the frequency of accesses to the relatively higher power flash memory.

When code is executed from the flash memory with the cache enabled, frequently used instructions are automatically cached in a dedicated region of the SRAM. In most applications, no further user effort is required, though locking and control features are provided.

FLASH CONTROLLER

The flash controller is coupled with a cache controller module, which provides two advanced microcontroller bus architecture high performance bus (AMBA AHB) ports; one port for reading data (DCODE), the other for reading instructions (ICODE). The flash controller supports simultaneous ICODE and DCODE read accesses. DCODE has priority on contention.

The flash controller implements a prefetch mechanism to optimize ICODE read performance. This mechanism provides optimal performance when reading consecutive addresses on the ICODE interface. Simultaneous reads are possible if the ICODE read returns buffered data from prefetch.

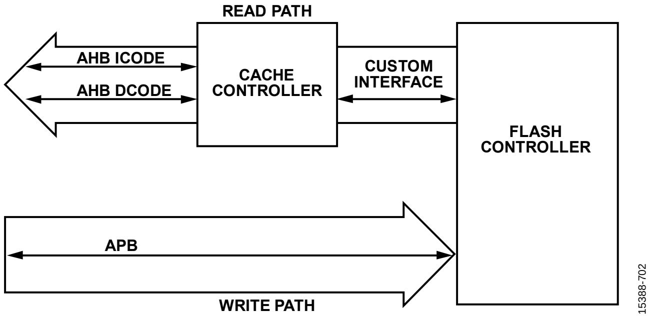


Figure 45. Flash and Cache Controllers

EFFECTS OF CACHE

Effects of Cache on the Speed of Execution

The flash memory and the SRAM memory have distinct power and performance profiles.

The cache architecture copies a portion of user code into the SRAM during execution where instruction and data read latency is lower. For every instruction or data read that is satisfied by the cache memory, the overall system performance is improved.

When using the cache along with the flash, the increase in speed depends on the type of code. Generally observing real code, for a loop code that fits into the cache completely, the speed of access is 15% to 20% faster than using only flash. For a linear code that cannot fit into the cache, the speed of access can increase by 10% to 15%.

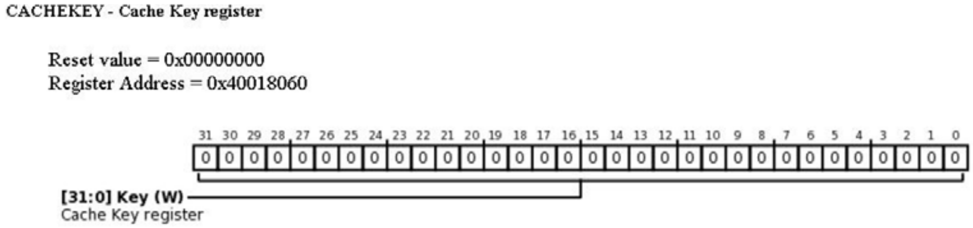
Using the cache generally increases the speed of execution; however, the extent of increase depends on the type of code used. If the code has loops that fit completely into the cache, the speed of execution increases significantly, because the majority of instruction accesses is served from the faster SRAM memory. If the code is generally linear and/or jumps between segments too large to fit into the cache, the speed of execution is not significantly improved, because the majority of instruction accesses is served from the slower flash memory.

Each cache miss results in a cache line fill consisting of four 64-bit reads from the flash memory.

Follow these instructions to enable or disable the cache:

1. The instruction cache (ICACHE) is disabled by default. To enable the ICACHE or toggle, the 0xF123F456 key must to be written into the CACHEKEY register.
2. The details of the CACHESETUP register are shown this section.
3. To enable the ICACHE, set the ICacheEnable_mmr bit in the CACHESETUP register. Clear this bit to disable the ICACHE.
4. It is possible to use the ICacheLock_mmr bits to lock the cache so that no more misses are replaced in the cache. This feature is helpful with loops that are larger than the cache size, where power is saved by placing appropriate locks.
5. To lock the contents of the ICACHE, set the ICacheLock_mmr bit in the CACHESETUP register. Unlock the cache by clearing the ICacheLock_mmr bit in the CACHESETUP register.

Cache Key Register

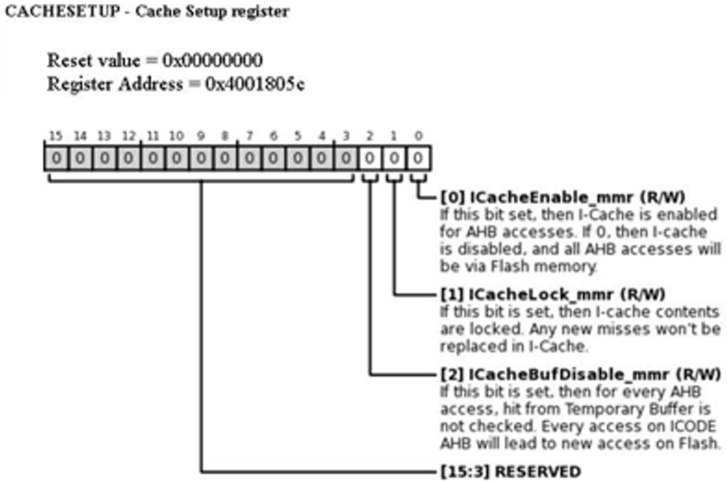


CACHEKEY - Cache Key register

Reset value = 0x00000000
Register Address = 0x40018060

[31:0] Key (W)
Cache Key register

Cache Setup Register



CACHESETUP - Cache Setup register

Reset value = 0x00000000
Register Address = 0x4001805c

[0] ICacheEnable_mmr (R/W)
If this bit set, then I-Cache is enabled for AHB accesses. If 0, then I-cache is disabled, and all AHB accesses will be via Flash memory.

[1] ICacheLock_mmr (R/W)
If this bit is set, then I-cache contents are locked. Any new misses won't be replaced in I-Cache.

[2] ICacheBufDisable_mmr (R/W)
If this bit is set, then for every AHB access, hit from Temporary Buffer is not checked. Every access on ICODE AHB will lead to new access on Flash.

[15:3] RESERVED

Effects of Cache on Current

When it comes to current consumption, SRAM accesses consume less current than flash accesses. Therefore, when the cache is enabled during the execution of code from the flash memory, the current consumption is generally between that of code executing directly from the flash or directly from the SRAM, unless the code is such that every cache access is a miss. In this case, the current is higher than executing from flash alone: cache line fills result in approximately 2× more flash reads than occur if executing directly from the flash. This read rate is 4× more, if not for the prefetch buffer in the flash also performing a read, which must miss too, if the cache misses.

$$Current_{SRAM} \leq Current_{CACHE}$$

When using the cache, the current consumption is proportional to the cache miss rate. This result is because of a scalar current reduction for each cache hit, because the data or code access is served from the lower current SRAM, rather than the higher current flash memory. Therefore, code consisting of many small loops is more greatly affected than linear code or code consisting of segments too large to fit into the cache memory.

The cache usage can also be inferred from the current consumption. If the current consumption using cache and flash is nearer to the current consumption when using the SRAM, the cache hit rate must be high. If the current consumption using cache and flash is nearer to the current consumption when using only the flash, the cache hit rate must be low.

CURRENT CONSUMPTION COMPARISON

For a loop code (prime number code in this example), it is seen that the current consumption using the flash and cache (980 µA) is very close to the current consumption in the SRAM (950 µA). This result is because the code consists of many loops each small enough to fit into the cache. Therefore, the accesses to the flash are relatively rare and the code mostly executes from the relatively low power cache. Accesses are minimal, therefore a minimal increase in the current consumption in flash and cache compared to the current consumption using the SRAM is seen.

For a linear code (ULPBench code in this example) it is seen that the current consumption using the flash and cache deviates away from the current consumption seen using only the SRAM. This result is because the ULPBench code is mostly linear and does not fit well into the cache, and therefore there is still a similar number of accesses into the flash memory as there are when executing directly from the flash. The cache misses are too high; therefore, the current consumption deviates away to a greater extent.

Table 13. Current Consumption Comparison

Type of Code	SRAM (µA)	Flash (µA)	Flash and Cache (µA)	Cache Misses (~12 sec of Execution)
Loop code (prime number)	950	1280	980	24
Linear code (ULPBench)	911	1493	1083	~800000

DUAL RTC FEATURE IN THE ADuCM3027/ADuCM3029

In many applications, an RTC is used to time stamp sensor data. The RTC must run even when the MCU is in a deep sleep mode. A low power RTC is crucial to achieving a long battery life.

Notable features of ADuCM3027/ADuCM3029 RTCs include the following:

- A dual RTC (RTC0 and RTC1). Both RTCs can be used as wake-up timers.
- SensorStrobe and input capture features.

This section provides guidelines for choosing between RTC0 and RTC1, depending on the power modes and functionality required.

COMPARISON OF THE RTC FEATURES

The ADuCM3027/ADuCM3029 has two RTCs, RTC0 and RTC1 (also named FLEX_RTC). Table 14 shows differences between RTC0 and RTC1.

POWER CONSIDERATIONS

Table 15 shows current consumption when using RTC0 and RTC1 in different use cases. Four scenarios are considered as follows:

- Scenario 1. The device switches between the active and hibernate power modes, and the application requires high time accuracy. Either RTC0 or RTC1 can be used in this scenario, but using RTC1 is recommended from a power point of view because RTC1 uses less power.
- Scenario 2. The device switches between the active and hibernate power modes, and the application does not require high time accuracy. Either RTC0 or RTC1 can be used in this scenario, but using RTC1 is recommended from a power point of view because RTC1 uses less power.
- Scenario 3. The device switches between the active and shutdown power modes. Only RTC0 can be used in this scenario.
- Scenario 4. The device switches between the active, shutdown, and hibernate power modes. Only RTC0 can be used, in this scenario because RTC1 is not active in shutdown mode.

A basic program comprised of an RTC alert to wake up the ADuCM3027/ADuCM3029 microcontroller from low power mode and to toggle an LED was used for measuring current in the sleep power mode (hibernate or shutdown mode, depending on the scenario).

CONCLUSION

Use RTC0 in applications that use shutdown mode and require an RTC.

RTC1 is a feature rich RTC that enables ultra low power consumption in applications that only use the hibernate mode, in addition to active or Flexi modes. Typical applications for which RTC1 is suited are applications where the ADuCM3027/ADuCM3029 microcontroller sends output pulses to external sensors via a general-purpose input/output. Note that the SensorStrobe mechanism is only available in RTC1.

Table 14. Summary of the Differences Between RTC0 and RTC1

Feature	RTC0	RTC1
Resolution of the Time Base (Prescaling)	RTC0 counts time at 1 Hz in units of seconds only	RTC1 can prescale the clock by any power of 2 from 1 to 15, counting time in units of any of these 15 possible prescale settings
Wake-Up Timer	The wake-up time is specified in units of seconds	The wake-up time can be specified in units of any power of 2 multiple of 30.7 µs up to 1 second
Number of Alarms	1 alarm only, which uses an absolute, nonrepeating alarm time	2 alarms: one absolute alarm time and one periodic alarm, repeating every 60 prescaled time units
Power Domain	Powered off VBAT domain and is always on; RTC0 can function in all power modes	Powered off 1.2 V (VREG) domain; RTC1 can function in all power modes, except shutdown mode
SensorStrobe and Input Capture Features	Not supported	Supports four input capture channels and one SensorStrobe channel; refer to AN-1427 for further information on these features
Source Clock	Low frequency crystal (LFXTAL)	Depending on the low frequency multiplexer (LFMUX) configuration, the RTC is clocked by LFXTAL or the low frequency oscillator (LFOSC)

Table 15. Comparison of Current Consumption in Different Use Case Scenarios

Scenario Number	Use Case[1]	Recommended RTC	Sleep Mode Current (nA)
1	Active to hibernate	RTC1 (LFXTAL)	830
2	Active to hibernate	RTC1 (LFOSC)	750
3	Active to shutdown	RTC0 (LFXTAL)	330
4	Active to hibernate	RTC0 (LFXTAL)	830

[1] The device switches between the modes listed in this column.

BENEFITS OF THE ADuCM3027/ADuCM3029 DC-TO-DC CONVERTER

This section discusses the advantages and disadvantages of charge pump converters vs. inductor converters, the latter of which are frequently used. This section demonstrates why this architecture is used on the ADuCM3027/ADuCM3029 microcontroller, accounting for advantages in many aspects such as price, area, simplicity, and ease of use.

Direct current-to-direct current (dc-to-dc) converters are key blocks in designs where it is required to manage different voltage domains, such as in the ADuCM3027/ADuCM3029 microcontrollers.

Methods of dc-to-dc conversion are briefly explained in this section to provide users with context. A charge pump converter is chosen for use in the ADuCM3027/ADuCM3029 because of its advantages when compared to other configurations.

The purpose of this section is to help users understand why the capacitive dc-to-dc converter is a better alternative to inductive conversion solutions in the ultra low power applications for which the ADuCM3027/ADuCM3029 is intended.

This section provides details and examples to prove the qualities and benefits of this charge pump converter solution. Figure 46 shows the buck enabled design present in the ADuCM3027/ ADuCM3029 microcontrollers. The ADuCM3027/ADuCM3029 uses a charge pump converter, which is not used in the majority of microcontrollers with similar characteristics available on the market; these other microcontrollers tend to use traditional inductor converter architectures.

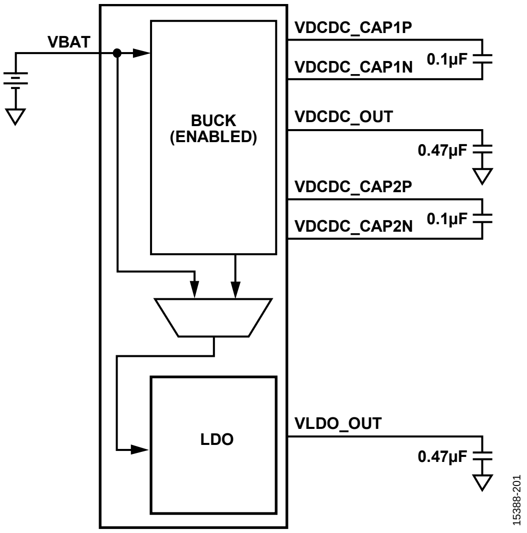


Figure 46. ADuCM3027/ADuCM3029 Buck Enabled Design

DC-TO-DC BASICS

The ADuCM3027/ADuCM3029 processors are intended for ultra low power applications. Power efficiency is one of the key considerations in such applications; therefore, using a dc-to-dc converter is crucial in designs where power must be used as efficiently as possible.

There are different ways to perform dc-to-dc voltage conversions; such conversions involve stepping up or stepping down the dc voltage used to power the device.

DC-to-DC Conversion Methods

The most extensively used methods for regulating the different power domains of a system are switching conversion and linear regulation. Select the method that best meets the requirements of the design or application.

Linear Regulators

Linear regulators consist of a network of resistive dividers that dissipate excess voltage. Linear regulators are widely used due to the ease of use and implementation, as well as the low cost.

In ultra low power applications, linear regulators are less efficient when compared to switching converters. In a linear regulator, the output current is approximately the same as the input current, and its operating principle is to dissipate any leftover voltage. Switching converters perform the same action more efficiently.

The ADP165/ADP166 devices are very low quiescent current, low dropout (LDO), linear regulators. The ground current represents the difference between input and output currents. Figure 47 represents the ADP165/ADP166 ground current vs. the load current (I_{LOAD}), showing the small difference in currents in a linear regulator.

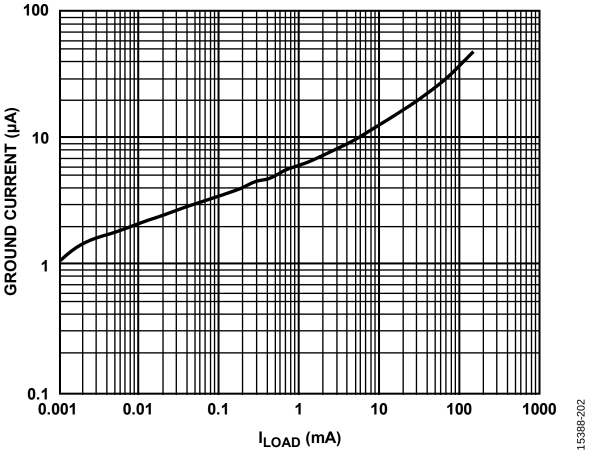


Figure 47. ADP165/ADP166 Ground Current vs. Load Current (I_{LOAD})

To analyze the power efficiency of the solution, consider a typical application based on linear regulators. For an input voltage of 3 V, an output voltage of 1 V, and an output current of 1 µA, the input current is approximately 1 µA. This scenario results in 33% efficiency (see Equation 3).

$$Efficiency = \frac{Energy\ Output}{Energy\ Input} \times 100\% \qquad (1)$$

$$Efficiency = \frac{I_{OUT} \times V_{OUT} \times t}{I_{IN} \times V_{IN} \times t} \times 100\% \qquad (2)$$

$$Linear\ Regulator\ Efficiency = \frac{1 \times 1 \times t}{1 \times 3 \times t} \times 100\% = 33\% \qquad (3)$$

Using a switching converter instead of a linear regulator, the input current is 1/3 µA, leading to 100 % efficiency in an ideal performance, as shown in Equation 4.

$$Efficiency = \frac{1 \times 1 \times t}{1/3 \times 3 \times t} \times 100\% = 100\% \qquad (4)$$

In general, switching converters are more efficient than linear regulators. Moreover, losses in efficiency produce an increase in temperature that is much higher in linear regulators because their dissipation must be larger to achieve the same conversion. Additionally, linear regulators require more investment in management to reduce temperature.

Traditionally, Analog Devices used linear regulators in designs that precede the ADuCM3027/ADuCM3029 microcontrollers because of their simplicity and low cost. Currently, it is common to locate linear regulators at the output of charge pump converters to stabilize their rippled output.

Switching Converters

Switching converters use switches and components with low losses, such as inductors or capacitors, to regulate voltage. Typically, these components are charged and discharged by switching transistors. This section discusses two types of switching converters: charge pump converters and inductor converters.

Inductor converters are among the most commonly used converters in microcontroller designs to achieve ultra low power with high efficiency. This efficiency and the wide gain range make this architecture desirable.

The charge pump, or switched capacitor converter, is an alternative to inductive converters. The charge pump process is carried out by connecting and disconnecting switches to charge and discharge capacitors. This process is achieved without inductors, which saves space and costs.

The ADP2503/ADP2504 are high efficiency inductor converters that can operate at input voltages greater than, less than, or equal to the regulated output voltage. The ADM660/ADM8660 are charge pump voltage converters that can achieve efficiency greater than 90% with low output currents (up to 50 mA). Figure 48 and Figure 49 show the efficiency for a given input voltage and output currents for the ADP2503/ADP2504 devices and ADM660/ADM8660 devices, respectively.

As observed in these two graphs, charge pump converters are less efficient than inductor converters due to the output shape for output load currents. In contrast, charge pump converters are an appropriate solution to low load current applications.

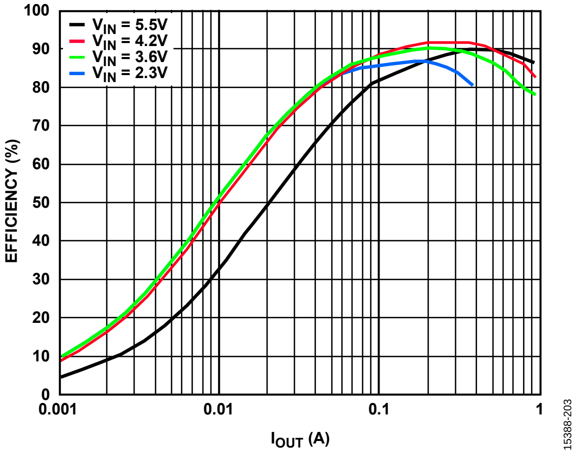


Figure 48. ADP2503/ADP2504 Efficiency vs. Output Current (I_{OUT})

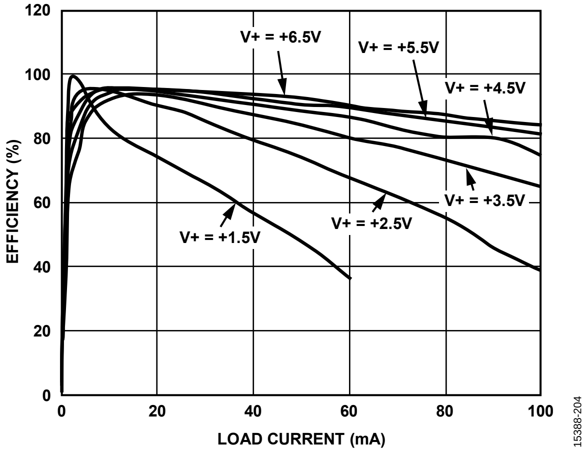


Figure 49. ADM660/ADM8660 Efficiency vs. Load Current

The ADuCM3027/ADuCM3029 microcontrollers have a linear regulator at the charge pump converter output to adjust and stabilize the supply of the digital core and memories. Furthermore, the devices have the ability to bypass the charge pump converter to only use the linear regulator to reduce and adjust the voltage. This feature allows the user to decide between using a traditional solution or to improve efficiency and increase power savings using the charge pump block at the expense of two extra 0.1 µF capacitors.

In general, inductor converters do not require a linear regulator at their output; this is inconvenient in charge pump converter designs. However, some microcontrollers available on the market that use an inductor converter solution include a linear regulator in the inductor converter. Despite this disadvantage, charge pump converters offer a breadth of advantages to be considered (as follows):

- Area
- Thickness
- Price
- Design simplicity
- Ease of use
- Electromagnetic interference (EMI)

The following section discusses each advantage of charge pump converters when compared to inductor converters.

CAPACITORS vs. INDUCTOR CONVERTERS

Area

Charge pump converters do not require inductors to accomplish dc-to-dc voltage conversion, whereas inductor converters require inductors, capacitors, and other components, such as resistors, to fulfill this task. This fact allows the design of smaller printed circuit boards (PCBs), saving area and cost.

This section compares the components required for using the ADuCM3027/ADuCM3029 charge pump converter against inductor converters used in other similar solutions available on the ultra low power microcontroller market. The components required in these architectures are included in their respective data sheets.

The two bill of materials shown in Table 16 and Table 17 demonstrate that charge pump converter covers less area than the inductive converters. The dimensions of capacitors (length × width × thickness) are 0.6 mm × 0.3 mm × 0.3 mm; the dimensions for inductors are 1 mm × 0.5 mm × 0.55 mm. Both capacitors and inductors are in 0603 packages.

Table 16. Area of Charge Pump Converter Components in the ADuCM3027/ADuCM3029

Component	Value (µF)	Quantity	Area (mm²)
Capacitor	0.1	2	0.36
Capacitor	0.47	1	0.18
Total			0.52

Table 17. Area of Inductor Converters Components

Component	Value	Quantity	Area (mm²)
Capacitor	1 µF	2	0.36
Inductor	2.2 µH	2	1
Total			1.36

As expected, the area in a charge pump converter is smaller because this type of converter employs fewer and smaller components than inductive solutions. Inductor converters use more than the double the area of charge pump converters.

In inductor converters solutions, the thickness is determined by the inductors, because inductors are thicker than capacitors—0.55 mm vs. 0.3 mm.

Inductors are higher than capacitors along the three dimensions; therefore, inductors determine and increase the area of the design.

Regarding this approach, it is possible to look through microscopes to see this evident difference in dimensions. Figure 50 and Figure 51 compare the dimensions of a 2.2 µH inductor and a 0.1 µF capacitor.

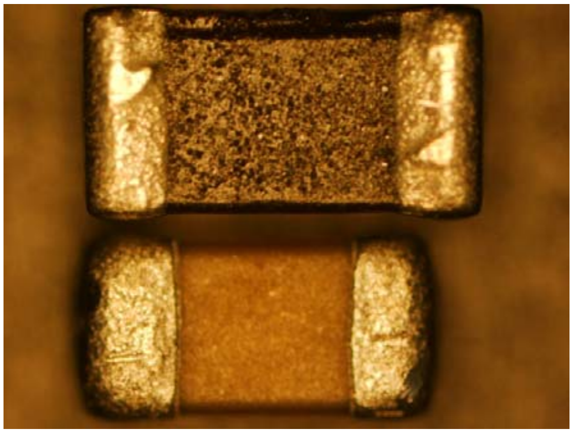


Figure 50. Length Comparison of a 0.1 µF Capacitor (Bottom) vs. a 2.2 µH Inductor (Top)

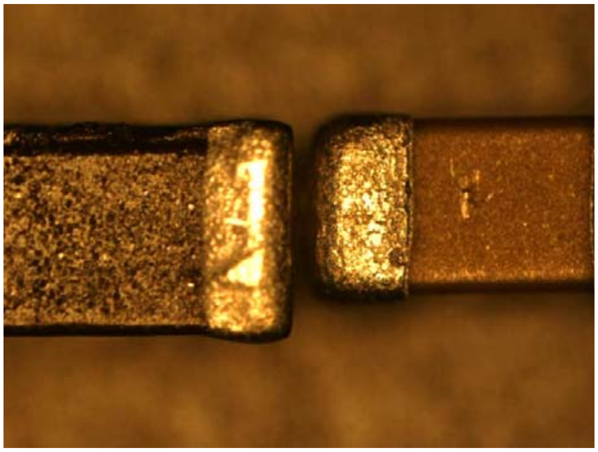


Figure 51. Width Comparison of a 0.1 µF Capacitor (Right) vs. a 2.2 µH Inductor (Left)

Price

Issues to consider when working with inductor converters include the large number of components and the cost of these components. The capacitance of capacitors is larger and the price of inductors is much greater than the price of capacitors.

Table 18 and Table 19 show the price of the ADuCM3027/ ADuCM3029 solution and an inductor based solution, respectively.

Table 18. Price of Charge Pump Converter Components for the ADuCM3027/ADuCM3029

Component	Value (µF)	Quantity	Price ($)
Capacitor	0.1	2	0.1046
Capacitor	0.47	1	0.459
Total			0.5636

Table 19. Price of Inductor Converters Components

Component	Value	Quantity	Price ($)
Capacitor	1 µF	2	1.068
Inductor	2.2 µH	2	0.538
Total			1.606

When considering both lists of materials, note that the difference in the price of each component is around $1. This amount, though seemingly small, produces a notable impact when multiplying the cost across many products; the difference in price is incremented because, in inductor designs, more components are present and the engaged area is wider. Allowing smaller PCB designs contributes to a reduction in cost.

Note that prices are based on the cheapest and smallest components available; this lack of inductor quality may lead to increased power dissipation and a degradation in efficiency. Therefore, the use of cheap inductors prevents the designer from taking advantage of inductor-based performance. In contrast, avoiding this constraint implies another increase in price for inductor converters.

There are clear advantages in area and price when comparing charge pump converters and conventional inductor-based solutions.

Efficiency

As with the opportunity to reduce area, there also exists an opportunity to improve integration employing embedded components. This is a suitable scenario for charge pump converters to enhance efficiency, rather than inductors.

It is thought that charge pump converters are less efficient than inductor converters, which can be true when input voltages and loads change.

In charge pump converters, load changes are not a problem in ultra low power applications where low loads are managed. Optimal efficiency is achieved with low load currents. Charge pump converters perform proper efficiencies with low loads, which is easily achieved by applying integration in an ultra low power application. The lower the required load, the better integration and efficiency are in charge pump converters.

By setting the appropriate configuration, charge pump converters are able to change their gain according to the input/output voltage ratio (V_{IN}/V_{OUT}). This process improves efficiency to achieve the same performance available with inductor converters.

Inductor based solutions use pulse-width modulation (PWM) to adjust the duty cycle to achieve suitable gain. Through this regulation, high efficiency is obtained, which decreases when the load lowers. Noise effects also appear during PWM, which results in increased cost for more expensive inductors.

If integration is required in inductor converter designs, embedded inductors require high frequency switching to work. High frequency switching results in power dissipation and efficiency losses, which is an undesirable outcome.

Electromagnetic Interference (EMI)

In charge pump converters, electromagnetic emissions are not relevant. Such radiation is not a cause for concern, unlike inductor magnetic radiation.

EMI becomes inconvenient when using inductors, even more so if they are switched inductors with behavior similar to an emitting antenna. Unpredictable interferences can occur in other parts of the design or the board. Furthermore, it becomes a sensitive problem if radio frequency tasks are being performed.

Inductor converters replace PWM with pulse frequency modulation (PFM) when low loads are required to improve efficiency. If PWM is performed, switching noise and output voltage ripple are easily improved by a simple filter at the output voltage of the converter. However, the PFM method has a variable frequency band and may produce the resonance frequency of the filter. In addition, this wide frequency spectrum also results in high EMI.

CONCLUSIONS

Inductor converters are not suitable solutions in many senses when considering ultra low power applications. Inductor converters lose efficiency as load decreases, their area is larger, which can lead to expensive components and greater costs, inductor radiation poses a problem, EMI is more likely to occur at low loads, and so on.

Table 20 summarizes the advantages and disadvantages of three types of dc-to-dc converters. Evaluate the specific type of converter that best suits the application in question.

In conclusion, charge pump converters are the best solution in ultra low power applications. While other solutions worsen with low loads, charge pump converters are even better than in other situations.

Table 20. Comparison of Different Types of DC-to-DC Converters

Type of Converter	Advantages	Disadvantages
LDO	Simple Low cost No inductor No EMI	Less efficient than charge pumps and inductives
Charge Pump	Simple Low cost No inductor Cheaper than inductive Low loads Small area More efficient than LDO Low EMI	Less efficient than inductive at high loads EMI (less than inductive)
Inductive	Most efficient (not in low loads)	EMI Area Cost Poor efficiency at low loads Complex design

UART SOFTWARE FLOW CONTROL

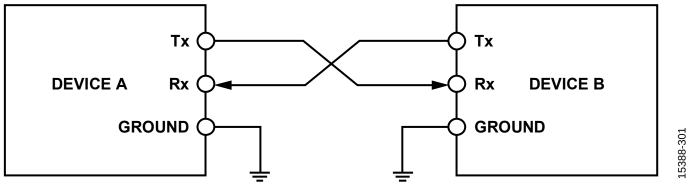


Figure 52. Software Flow Control Block Diagram

Flow control is the process of managing the rate of data transmission between two nodes to prevent a fast transmitter from overwhelming a slow receiver. Flow control provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from the transmitting node.

UART flow control is a method for slow and fast devices to communicate with each other over the UART without the risk of losing data. Consider the case where two units are communicating over the UART. A transmitter, Tx, is sending a long stream of bytes to a receiver, Rx. Rx is a slower device than Tx and at some point Rx cannot keep up with the speed of the data being transmitted. Therefore, Rx must either process some of the data or empty buffers before it can continue to receive data. Rx must instruct Tx to stop transmitting until Rx is ready to accept data. This method of waiting to transmit is known as flow control.

Flow control requires extra signaling to inform the transmitter to stop (pause) or start (resume) the transmission. The traditional hardware flow control in UART requires two extra signals: request to send (RTS) and clear to send (CTS). The logic level on these signals defines whether the transmitter continues to send data or must stop sending data. With software flow control, special characters are sent over the normal data lines to start or stop the transmission, thus using fewer signals.

This section describes the UART software flow control mechanism using the ADuCM3027/ADuCM3029.

UART FLOW CONTROL

Hardware Flow Control

The hardware flow control mechanism uses out of band signaling to control the flow of data. In addition to the data signals, two extra signals—RTS and CTS—are required. These flow control signals are cross coupled between the two devices, with RTS on one device being connected to CTS on the remote device, and vice versa.

Each device uses the RTS to signal if it is ready to accept new data, and reads the CTS signal to check if it is allowed to send data to the other device. As long as a device is ready to accept more data, the RTS signal is kept asserted. The device deasserts the RTS signal when its receive buffer is full.

The other device is required to respect the flow control signal and pause the transmission until the RTS signal is asserted again.

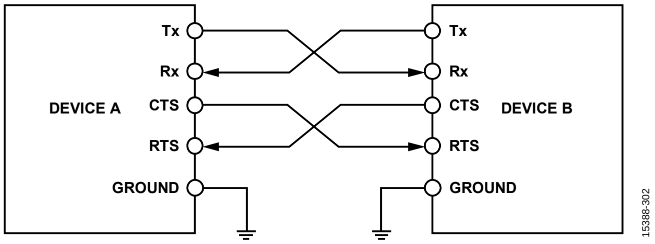


Figure 53. Hardware Flow Control

The flow control is bidirectional, meaning that both devices can request a halt in transmission. If one of the devices never requests a stop in transmission (for example, if the device is fast enough to always receive data), the CTS signal on the other device can be tied to the asserted logic level. Thus, the RTS pin on the fast device can become free to perform other functions.

Software Flow Control Using the XON and XOFF Signals

Software flow control does not require extra out of band signals. Only three signals are required: Rx, Tx, and ground. Software flow control is achieved by using special control flow characters. The control flow characters are sent over the normal Tx and Rx lines. These characters are typically ASCII codes, specifically XON (0x11) and XOFF (0x13), for resuming and halting the transfer, respectively.

If Device A sends XOFF to Device B, Device B halts transmission to Device A until Device B receives an XON character from Device A. If the data contains the XON and/or XOFF character, insert an escape character before the XON and/or XOFF character. The escape character used in this case is \ with ASCII Value 92 (0x5C). When this escape character is encountered, the character following it is considered to be a data character, not a flow control signal. If the data itself contains an escape character, ensure that another escape character precedes the present escape character that is present in the data.

Sequence Diagram

Consider data communication between two devices—the ADuCM3027/ADuCM3029 MCU and a peer—where the MCU is transmitting and the peer is receiving. If the peer is slower than the MCU, the data transmission overwhelms the peer. At this stage, the peer sends an XOFF character to pause the transmission until the peer is able to process the data again. The MCU waits to receive a XON character from the peer. When the peer is ready to receive the data, it sends an XON character, instructing the MCU to resume transmission. In this way, using software flow control ensures that no data is lost. Figure 54 shows the sequence diagram of this described communication.

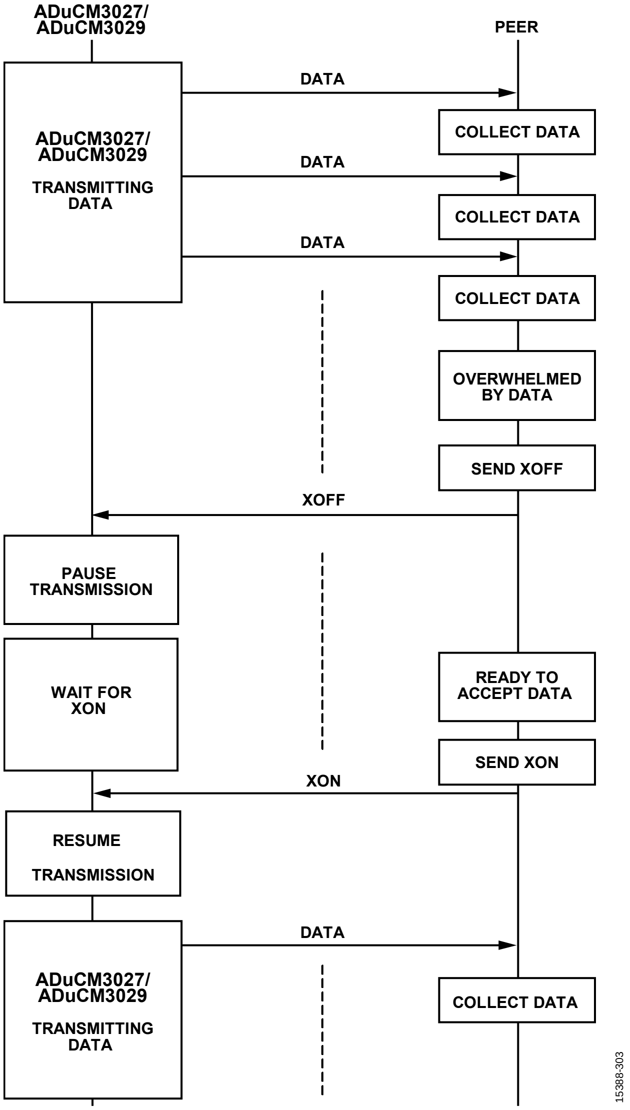


Figure 54. Software Flow Control Sequence Diagram

SYSTEM DESCRIPTION

Demonstration of UART software flow control using the ADuCM3027/ADuCM3029 is performed using the EZ-Kit evaluation kit. A PC with a terminal program running (such as HyperTerminal) is connected to the EZ-Kit UART port.

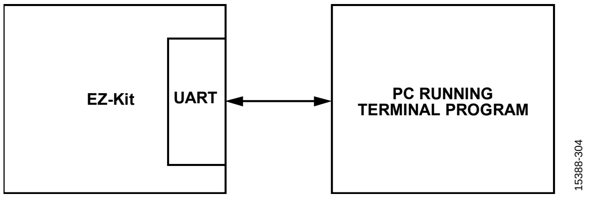


Figure 55. Connection Diagram

Handling Flow Control Signals from a Peer Device

The ADuCM3027/ADuCM3029 BSP contains drivers for all the peripherals, including UART. The software flow control mechanism is implemented in addition to the available UART driver functions.

The adi_uart_Write_fc function sends the XOFF and XON characters, and the UART interrupt service routine processes the XON and XOFF signals received from the PC.

adi_uart_Write_fc Function

When a write is issued using the adi_uart_Write_fc function, the global RECV_XON flag is checked to be aware whether the peer is ready to accept data. If the RECV_XON flag is false, it means that an XOFF signal is received, the peer cannot accept data, and, therefore, a failure is returned. If the RECV_XON flag is true, the peer is ready to accept data. The data is transmitted and a success is returned.

Example Code for Flow Control

The following code is used to transmit data using flow control:

```
ADI_UART_RESULT adi_uart_Write_fc(
ADI_UART_HANDLE const hDevice, void *const pBuffer, uint32_t nBufSize)
{
    /* Return code */
    ADI_UART_RESULT eResult;
    /* If there is no XOFF received, safe to transmit data */
    if(RECV_XON == true)
        eResult = adi_uart_Write (hDevice, pBuffer, nBufSize);
    /* If XOFF is received, return fail */
    else
        eResult = ADI_UART_FAILED;
    return eResult;
}
```

Figure 56 shows the design of adi_uart_Write_fc function. When a write is issued, the RECV_XON flag is checked and, if the flag is true, the write is processed. If the flag is not true, it returns a failure. The RECV_XON == TRUE block in Figure 56 indicates the checking of the RECV_XON flag.

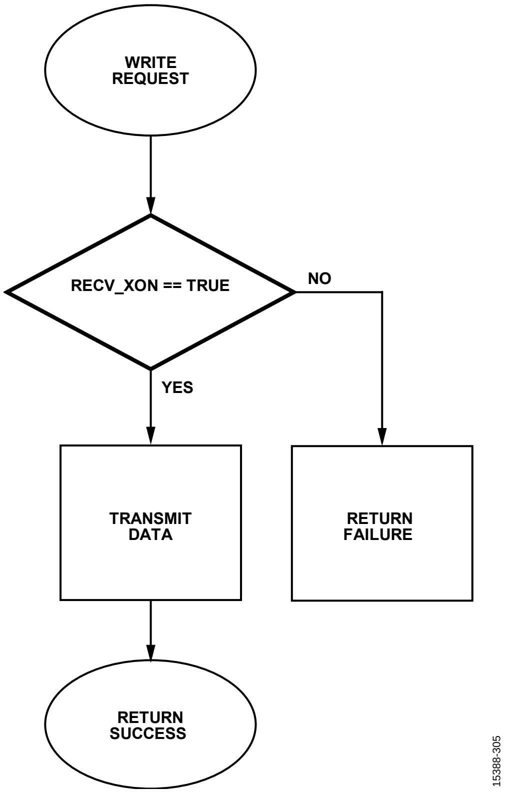


Figure 56. Flowchart of the adi_uart_Write_fc Function

Processing Control Signals from the Peer Through an Interrupt Service Routine (ISR)

The data received through the UART is monitored to check if it is a control signal or data. If the data received is an escape character, an escape flag (bEscFlag) is asserted so that the data following it is to be considered data and not as a control signal. If an XOFF or XON signal is received, it is checked to confirm if the escape flag is set. If the escape flag is not set, a global flag (RECV_XON) is updated.

When receiving an XOFF signal without the escape flag set, the RECV_XON flag is deasserted, meaning that it received an XOFF signal and data transmission must not happen. In the same way, the RECV_XON flag is asserted when receiving an XON signal without the escape flag set.

Data Processing Code Example

The following code processes the data that is received.

```
switch (readVal)
{
  /* If an escape is received */
  case FCESCAPE:
    /* If escape already received,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
    else
      bEscFlag = true;
    break;
  /* If an XON is received */
  case XON:
    /* If escape received before,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
      /* Valid control signal,
         update send flag */
    else
      RECV_XON = true;
    break;
  /* If an XOFF is received */
  case XOFF:
    /* If escape received before,
       consider it as data */
    if(bEscFlag == true)
      bEscFlag = false;
    /* Valid control signal,
       update send flag */
    else
      RECV_XON = false;
    break;
  default:
    break;
}
```

Figure 57 shows the design of the algorithm that handles and processes the flow control signals from the peer. The data received is first checked for an escape character; if it is an escape character, the data following it is considered to be data and not a control signal. The received data is then checked for XON and XOFF control signals and the global RECV_XON flag is updated accordingly. The gray blocks in Figure 57 indicate the updating of the RECV_XON flag.

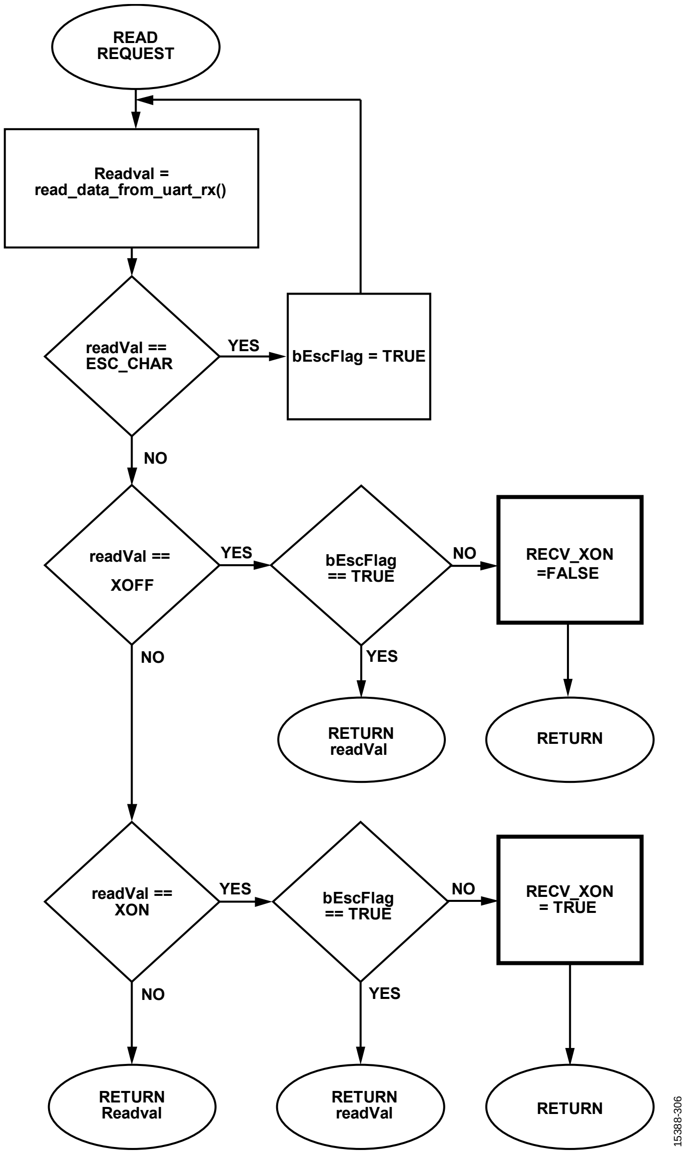


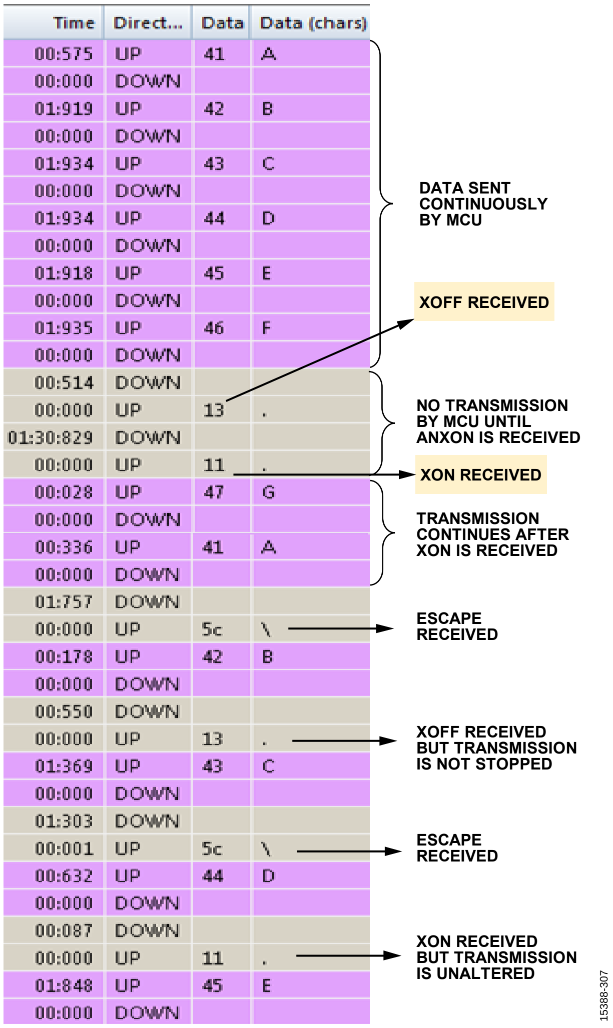
Figure 57. Flowchart of the Control Signals in the ISR Being Processed

DATA CAPTURE

In the setup for data capture, the MCU is connected to a PC and is communicating with a terminal program running on the PC. A UART sniffer, such as the serial port monitor, monitors the data communication occurring at a Baud rate of 9600. The data capture is performed using the sniffer, as discussed in this section.

Handling Flow Control Characters in the ADuCM3027/ ADuCM3029

Figure 58 shows an example of handling the flow control signals from the peer.



Time	Direct...	Data	Data (chars)
00:575	UP	41	A
00:000	DOWN		
01:919	UP	42	B
00:000	DOWN		
01:934	UP	43	C
00:000	DOWN		
01:934	UP	44	D
00:000	DOWN		
01:918	UP	45	E
00:000	DOWN		
01:935	UP	46	F
00:000	DOWN		
00:514	DOWN		
00:000	UP	13	.
01:30:829	DOWN		
00:000	UP	11	.
00:028	UP	47	G
00:000	DOWN		
00:336	UP	41	A
00:000	DOWN		
01:757	DOWN		
00:000	UP	5c	\
00:178	UP	42	B
00:000	DOWN		
00:550	DOWN		
00:000	UP	13	.
01:369	UP	43	C
00:000	DOWN		
01:303	DOWN		
00:001	UP	5c	\
00:632	UP	44	D
00:000	DOWN		
00:087	DOWN		
00:000	UP	11	.
01:848	UP	45	E
00:000	DOWN		

DATA SENT CONTINUOUSLY BY MCU

XOFF RECEIVED

NO TRANSMISSION BY MCU UNTIL ANXON IS RECEIVED

XON RECEIVED

TRANSMISSION CONTINUES AFTER XON IS RECEIVED

ESCAPE RECEIVED

XOFF RECEIVED BUT TRANSMISSION IS NOT STOPPED

ESCAPE RECEIVED

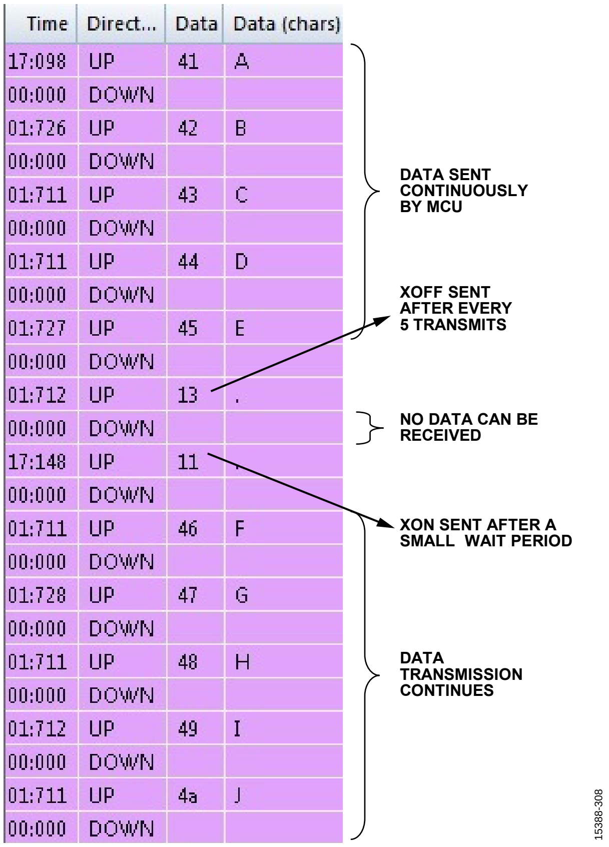
XON RECEIVED BUT TRANSMISSION IS UNALTERED

Figure 58. Data Capture Using a Sniffer Program

Controlling the Received Data Flow

A simple procedure is implemented when controlling the received data flow to send the control signals from the ADuCM3027/ADuCM3029 MCU. In this case, the MCU is slower compared to the peer. The mechanism to send the control signals from the MCU is application specific and the user can write their own algorithm for sending the control signals.

As shown in Figure 59, an XOFF signal is sent after every five transmissions sent from the MCU. An XON signal is sent after a short interval of time. This implementation is a simple and is described only for demonstration purposes. The user can develop their own mechanism to handle the data and to send XON and XOFF signals.



Time	Direct...	Data	Data (chars)
17:098	UP	41	A
00:000	DOWN		
01:726	UP	42	B
00:000	DOWN		
01:711	UP	43	C
00:000	DOWN		
01:711	UP	44	D
00:000	DOWN		
01:727	UP	45	E
00:000	DOWN		
01:712	UP	13	.
00:000	DOWN		
17:148	UP	11	.
00:000	DOWN		
01:711	UP	46	F
00:000	DOWN		
01:728	UP	47	G
00:000	DOWN		
01:711	UP	48	H
00:000	DOWN		
01:712	UP	49	I
00:000	DOWN		
01:711	UP	4a	J
00:000	DOWN		

DATA SENT CONTINUOUSLY BY MCU

XOFF SENT AFTER EVERY 5 TRANSMITS

NO DATA CAN BE RECEIVED

XON SENT AFTER A SMALL WAIT PERIOD

DATA TRANSMISSION CONTINUES

Figure 59. ADuCM3027/ADuCM3029 Transmitting Control Signals

SPI FLOW CONTROL METHODS

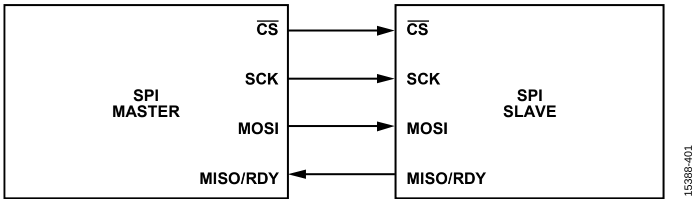


Figure 60. SPI Signals

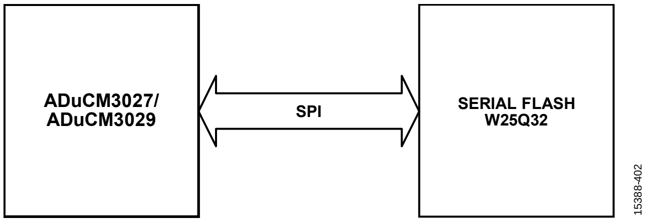


Figure 61. Application Block Diagram

The SPI is an industry-standard, synchronous serial link that allows full duplex operation to other SPI-compatible devices.

The ADuCM3027/ADuCM3029 SPI has enhanced modes of operation that provide the user the flexibility of half duplex operation and flow control options. The SPI data transfers use DMA transactions, allowing the ADuCM3027/ADuCM3029 core to be in sleep mode. Along with multibyte transfers in half duplex mode, this reduction in power consumption offers power savings that are essential for battery-powered designs, such as in wireless sensor networks.

Some notable features of the ADuCM3027/ADuCM3029 SPI are as follows:

- Continuous transfer mode.
- Read command mode for half duplex operation.
- Flow control.
- $\overline{\text{CS}}$ software override.
- Support for 3-pin SPI master or slave mode.
- LSB first transfer option.
- Interrupt mode. An interrupt is available after 1, 2, 3, 4, 5, 6, 7, or 8 bytes.

This section provides an understanding of the read command mode and flow control methods. These methods help lower the system power consumption when used with SPI slaves such as sensors, serial flash devices, ADCs, and RF transceivers.

SPI READ COMMAND MODE

Standard SPI masters communicate with slaves using the serial clock (SCK), master out, slave in (MOSI), master in, slave out (MISO), and chip select ($\overline{\text{CS}}$) lines. The SCK, MOSI, and MISO signals can be shared by slaves, whereas each slave has a unique $\overline{\text{CS}}$ line. During an SPI transfer, data is simultaneously transmitted and received. The serial clock line synchronizes shifting and sampling of the information on the two serial data lines.

SPI transfers are typically full duplex. The transfers are controlled by the master. To receive data from the slave, the master must provide the clock, which is typically initiated when the data must be sent on the MOSI line.

Most SPI slaves mandate a protocol that must be used by the mater for successful communication. The protocol can be as simple as a command, followed by an address (optional), and data (optional).

For example, a write command is unidirectional and typically involves the master transmitting the command, address (optional), and the data to be written to the address in the slave.

A read command requires the master to transmit the command and address (optional), and then reads the data associated with the address from the slave. If the data is multibyte, then the software on the master must write dummy data on the MOSI, which keeps the clock alive, to successfully read all the data bytes.

However, some SPI slaves require that, after the transmission of the read command byte on the MOSI, the data be read on MISO in a single $\overline{\text{CS}}$ transaction. An example of this requirement is shown in Figure 62.

The ADuCM3027/ADuCM3029 provides the read command mode to support such half duplex operations. The read command mode helps reduce the burden on the software and thereby the core execution cycles. In this mode, the user must specify the number of bytes to be transmitted and the number of bytes to be received in a transaction. It is also possible to specify if the data on the MISO must be ignored when the transmission on MOSI is in progress.

Using the read command mode allows the user to transmit a single byte and receive a set of data bytes from the slave, which is useful when the slave is a sensor or ADC providing a set of measured and processed data.

An application scenario is described in the System Description section, wherein the ADuCM3027/ADuCM3029 is the SPI master and a serial flash, W25Q32, is the SPI slave, as shown in Figure 61. Read command mode is helpful when pages of data must be read from the flash memory.

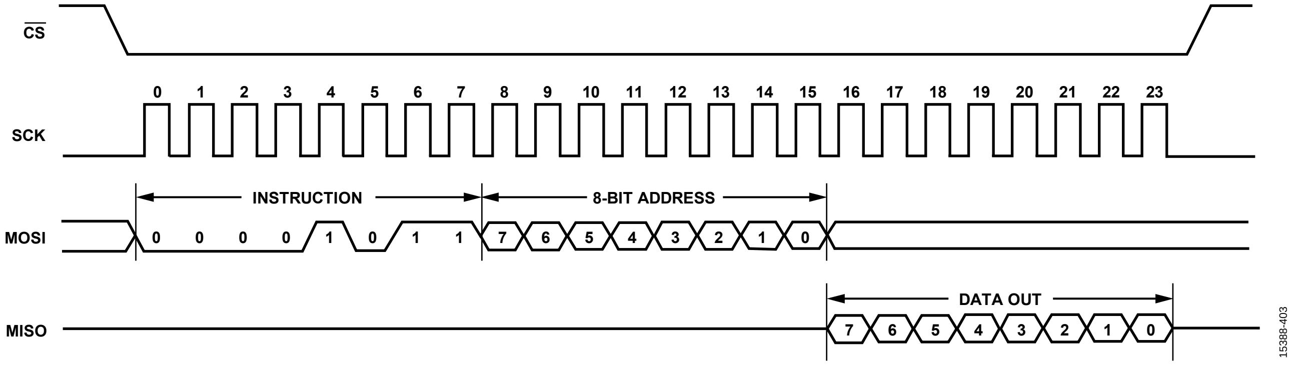


Figure 62. Read Command Mode

System Description

To showcase the read command mode, the following setup is used:

- Firmware—power-on self test application from the ADuCM3027/ADuCM3029 BSP for IAR.
- Hardware—ADuCM3027/ADuCM3029 EZ-Kit board.

An oscilloscope is connected to the SPI lines to capture the signals. The oscilloscope plots, Figure 63 to Figure 67, show the SPI transfer between the ADuCM3027/ADuCM3029 as the SPI master, and a serial flash W25Q32 as the SPI slave.

It is up to the user application to decide the transactions in which to use the read command mode.

Without Read Command Mode

In the reference application, the erase process of a 4 kB sector of the flash memory does not use read command mode. The absence of this mode can be observed from the transfer of the erase command and the address in individual chip select frames in Figure 63 and Figure 64.

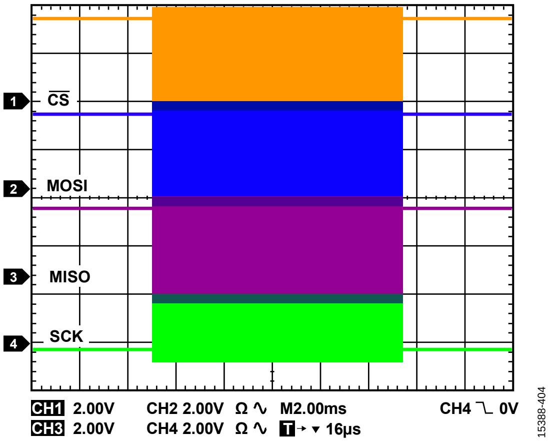


Figure 63. Sector Erase

Figure 64 shows a single $\overline{\text{CS}}$ frame capture in the entire erase sequence. The $\overline{\text{CS}}$ line is toggled for the transfer of every command byte.

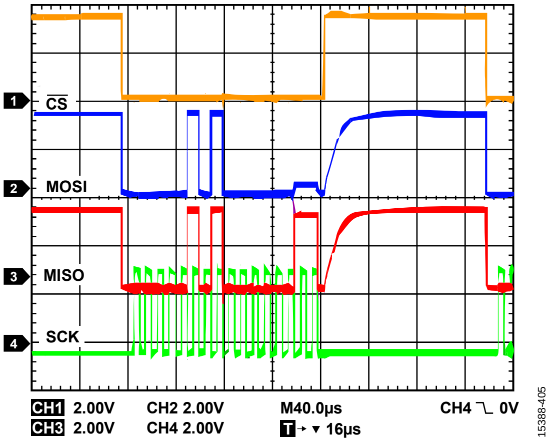


Figure 64. Sector Erase—Single $\overline{\text{CS}}$ Frame Capture

With Read Command Mode

Figure 65 shows one page read from the external flash. The size of one page of the flash is 256 bytes. This read sequence uses the read command mode and the entire read of the page happens in one $\overline{\text{CS}}$ transaction.

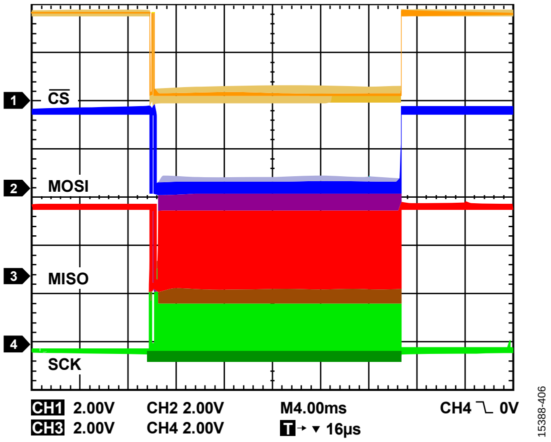


Figure 65. Page Read Sequence

Figure 66 shows the start of the page read sequence where the ADuCM3027/ADuCM3029 transfers the command and address bytes. This transfer is followed by the page data from the serial flash memory.

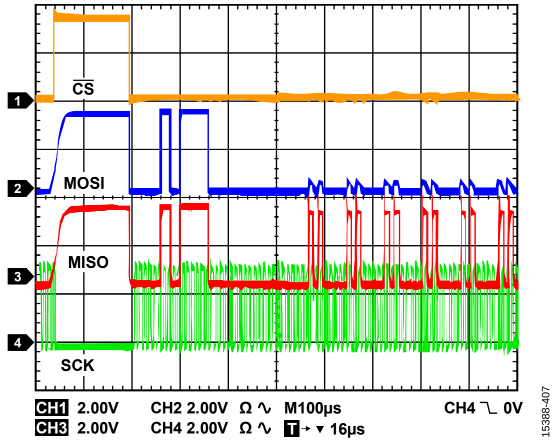


Figure 66. Page Read Start Sequence (Command and Address Bytes)

Figure 67 shows a single-byte read, which is part of the page read sequence.

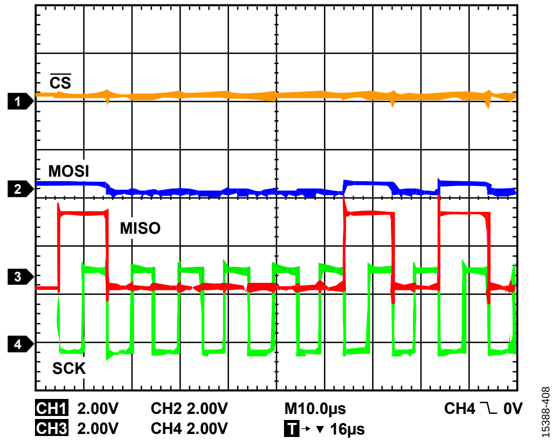


Figure 67. Page Read—Single Data Byte

FLOW CONTROL MODES

Flow control is necessary to synchronize the data flow between a master and slave. The ADuCM3027 provides flow control as a differentiating feature in the SPI. Along with read command mode, flow control can be used to receive multiple data bytes.

With flow control, the data transfer between the SPI master and slave is controlled based on the application requirements in terms of periodic data or demand-based data read.

The SPI master in the ADuCM3027/ADuCM3029 supports the following modes of flow control.

- Pin-based flow control, controlled by the SPI slave.
- Timer-based flow control, controlled by the SPI master.

The flow control modes are described in the following sections in more detail. The mode field in the SPI flow control register (SPI_FLOW_CTL) configures the flow control mode to any one of the three modes.

Note that flow control mechanisms can be used only when the ADuCM3027/ADuCM3029 is configured as an SPI master.

Pin-Based Flow Control

Using a Separate RDY Pin

Some SPI slaves have a dedicated RDY pin that is connected to the SPI_RDY pin of the SPI master, which in this case is the ADuCM3027/ADuCM3029. The SPI_RDY pin is a dedicated pin (as an alternate functionality to a GPIO) for every SPI instance.

For example, the CAT64LC40 serial flash uses a dedicated RDY pin to signal the availability of data to the SPI master.

The RDY pin of the ADuCM3027/ADuCM3029 can be wired to an interrupt pin of the SPI slave in case the slave does not support a dedicated RDY pin. The slave uses the RDY pin to indicate that the acquisition and data processing is complete. The master does not provide SPI clock until it sees an active level on this pin.

The user can configure the number of bytes to be read when the RDY pin is asserted. Perform this is configuration by setting the RDBURSTSZ field in the SPI flow control register (SPI_FLOW_CTL). After receiving this burst of bytes on MISO, the SPI master continues to wait for the next RDY pin assertion to receive the next set of bytes. This process is repeated until all bytes as set in the SPI count register (SPI_CNT) are received.

Note that using read command mode, a maximum of 16 bytes can be transmitted. This transmission is configured using the TXBYTES field of the SPI read control register (SPI_RD_CTL). The number of bytes received in one burst when using flow control is set in the RDBURSTSZ field of the SPI flow control register (SPI_FLOW_CTL) however, the total number of bytes to be received does not have an imposed maximum limit.

Using the MISO Pin

Some SPI slaves do not have a dedicated RDY pin but have a provision to reuse the MISO pin to inform the SPI master that the data is ready to be sent on MISO.

The ADuCM3027/ADuCM3029 SPI master waits for an active level transition on the MISO line and, when this is detected, reads RDBURSTSZ + 1 number of bytes and then goes back to a wait state until another active level is detected on MISO.

The polarity of the MISO/RDY pin can be configured using the POL field of the SPI flow control register (SPI_FLOW_CTL).

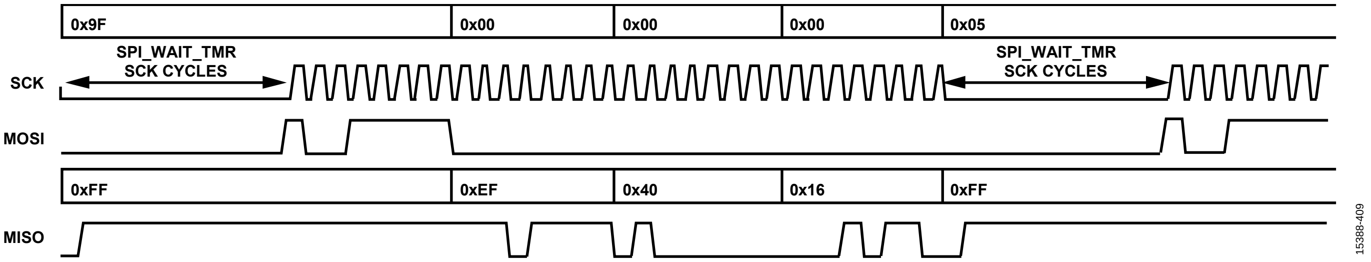


Figure 68. Software Flow Control with Timer

Timer-Based Flow Control

For slaves that do not have a dedicated pin to inform the availability of data to the master, the microcontroller uses a 16-bit timer to introduce wait states while reading data. When the timer triggers, the master reads a burst of bytes (RDBURSTSZ + 1) and then restarts the timer. The timer is clocked at the SPI clock rate (SCK) and the number of SCK cycles to wait before the timer is triggered can be set using the SPI_WAIT_TMR register. An example of this operation is shown in Figure 68.

When this scheme is used to stall and drive SCK for flow control, take care to ensure that the last SCK edge is a sampling edge. After the stall period is over, an SCK driving edge then causes the next data transfer.

System Description

This section uses the hardware flow control mode to demonstrate how the flow control feature can contribute to power savings in a system.

The system used to demonstrate this process consists of the ADuCM3027/ADuCM3029 MCU and a sensor (such as an accelerometer) connected over the SPI.

To design a power efficient system, it is essential to put the core in sleep mode whenever there is no processing required. In such a system, after a sensor reading is available, the core is woken up to receive and process the data from the sensor.

The flow control and read command modes in the ADuCM3027/ADuCM3029 enhances the efficiency of this process by offloading the MCU further. The system is put into Flexi mode, which keeps the core asleep and the SPI peripheral and the DMA active.

The sensor measures the data and uses the RDY pin to strobe the SPI peripheral of the data availability. There is a dedicated SPI_RDY pin (alternate functionality of a GPIO) for every SPI instance in the ADuCM3027/ADuCM3029 MCU.

Without waking up the MCU, the SPI then reads the data set using the read command mode. The sensor must be capable of multibyte data transfer to use this scheme effectively. In case of an accelerometer sensor, the x-, y-, and z-axis readings are sent as six bytes over the SPI.

DMA can be used to transfer the data into an allocated memory space without CPU intervention.

The application can collect the data instantaneously after every measurement, or can collect buffered data from the slave after a configured number of bytes are collected by the sensor.

After the user defined set of bytes are collected, the SPI peripheral or the DMA can wake up the MCU to process the sensor data.

Figure 69 shows the application flow diagram for an SPI data read from an accelerometer every time an activity is detected. In sensors such as the ADXL345, the data ready interrupt can be used to read the x-, y-, and z-axis readings in one SPI transaction. In other sensors, a FIFO configuration can be performed to store a number of samples in the sensor until the master reads the FIFO.

CONCLUSIONS

The different features of the ADuCM3027/ADuCM3029 SPI, such as read command mode and flow control, make the devices ideal for use in battery-powered systems where the SPI peripheral offloads the MCU and can be independently used for data collection.

This device suitability a significant advantage in wireless sensor networks where the battery life of the sensor is critical in system design. This also serves as a building block for designing smart sensors with on-board data acquisition, as well as sensor data analytics.

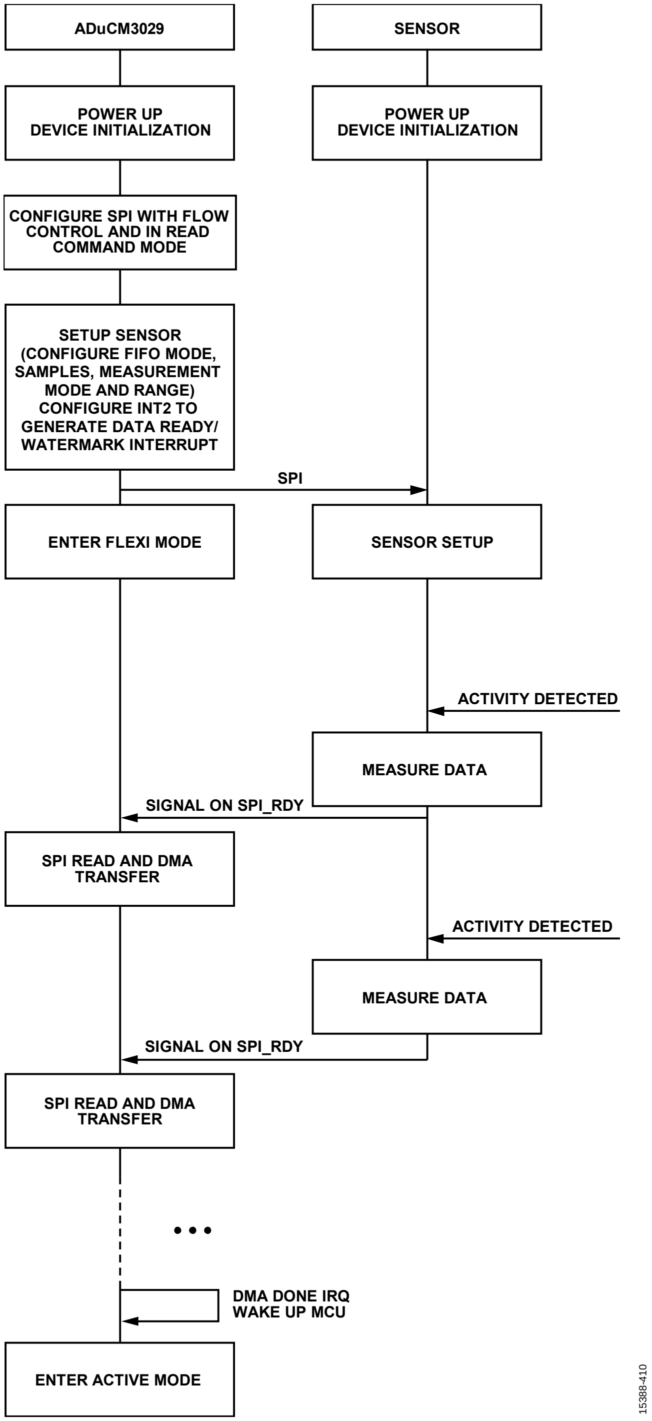

ADuCM3029
SENSOR
POWER UP DEVICE INITIALIZATION
POWER UP DEVICE INITIALIZATION
CONFIGURE SPI WITH FLOW CONTROL AND IN READ COMMAND MODE
SETUP SENSOR (CONFIGURE FIFO MODE, SAMPLES, MEASUREMENT MODE AND RANGE) CONFIGURE INT2 TO GENERATE DATA READY/ WATERMARK INTERRUPT
SPI
ENTER FLEXI MODE
SENSOR SETUP
ACTIVITY DETECTED
MEASURE DATA
SIGNAL ON SPI_RDY
SPI READ AND DMA TRANSFER
ACTIVITY DETECTED
MEASURE DATA
SIGNAL ON SPI_RDY
SPI READ AND DMA TRANSFER
• • •
DMA DONE IRQ WAKE UP MCU
ENTER ACTIVE MODE
15388-410

Figure 69. Application Flowchart

SLEEP ON EXIT

The ARM® Cortex™-M processors are ideal for low power applications due to their balance between energy and efficiency. These processors have a feature known as sleep on exit that allows saving clock cycles and energy.

The MCU subsystem of the ADuCM3027/ADuCM3029 processor is based on the ARM® Cortex™-M3 processor. The sleep on exit feature saves power when the microcontroller is sleeping and in interrupt handlers.

When sleep on exit is enabled, the processor enters directly to sleep when the ISR is finished. Interrupts are nested in case there is more than one interrupt. After the execution of these interrupts, the processor returns automatically to sleep mode.

BENEFITS

The sleep on exit feature presents some benefits in interrupt driven applications, where the system is sleeping and it only wakes up to run interrupts.

When sleep on exit is disabled, the workflow when an interrupt arrives involves more time spent executing instructions.

The steps to perform the interrupt with sleep on exit feature disabled are as follows:

1. Wake up the processor.
2. Push all the necessary information and the current state on the stack.
3. Pun the interrupt code.
4. Pop the information on the stack to restore the registers.
5. Return to sleep mode.

There are many instructions to run an interrupt. Therefore, on interrupt driven applications, the time spent on context switching is not optimal because the core pushes instructions in the stack and, subsequently, the core pops them again.

The process is simplified by enabling the sleep on exit feature. The processor immediately goes to sleep after finishing the interrupt. The device does not return to the normal thread and keeps the interrupt configuration, which avoids including push and pop tasks into the stack, saving the energy and clock cycles necessary to execute unnecessary instructions.

Figure 70 shows the flowchart when using the sleep on exit feature. The procedure for using this feature is as follows:

1. The program starts.
2. Sleep mode is invoked by a wait for interrupt (WFI) or a wait for events (WFE) instruction.
3. The system enters sleep mode.
4. The device is woken up by an interrupt or an event.
5. The system returns automatically to sleep mode when the interrupt is finished if the SLEEPONEXIT bit is set.

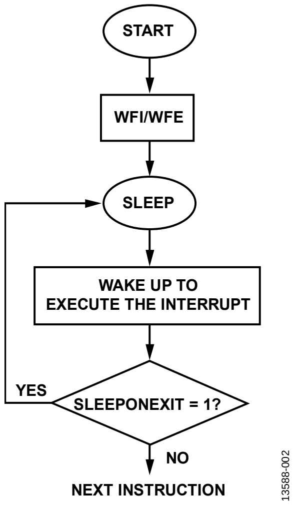


Figure 70. Sleep on Exit Flowchart

ENABLING THE SLEEP ON EXIT FEATURE

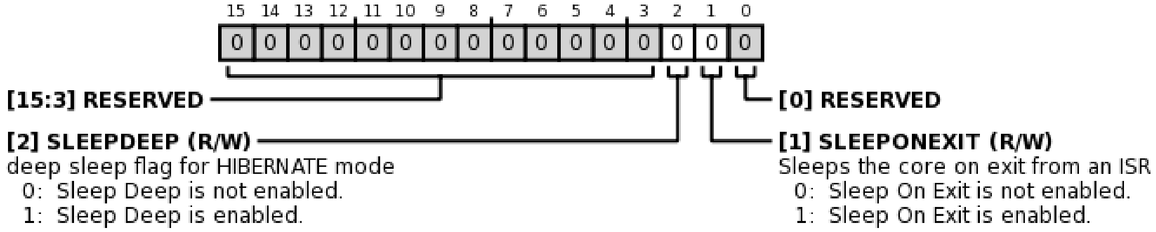
The ARM® Cortex™-M nested vector interrupt controller (NVIC) has a system control register with a bit field called SLEEPONEXIT. To enable the sleep on exit feature, it is only necessary to set the SLEEPONEXIT bit.

The address of the system control register is 0xE000ED10. The System Control Register in the ADuCM3027/ADuCM3029 section shows this register and its bit fields in the ADuCM3027/ADuCM3029 microcontroller.

SYSTEM CONTROL REGISTER IN THE ADuCM3027/ADuCM3029

INTCON0 - System Control Register

Reset value = 0x0000
Register Address = 0xe000ed10


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[15:3] RESERVED
[0] RESERVED
[2] SLEEPDEEP (R/W)
deep sleep flag for HIBERNATE mode
0: Sleep Deep is not enabled.
1: Sleep Deep is enabled.
[1] SLEEPONEXIT (R/W)
Sleeps the core on exit from an ISR
0: Sleep On Exit is not enabled.
1: Sleep On Exit is enabled.

Bits	Name	Description	Default	Access	Visibility
15:3		Reserved	0		
2	SLEEPDEEP	deep sleep flag for HIBERNATE mode 'b0 - Sleep Deep is not enabled 'b1 - Sleep Deep is enabled	'h0	R/W	Public
1	SLEEPONEXIT	Sleeps the core on exit from an ISR 'b0 - Sleep On Exit is not enabled 'b1 - Sleep On Exit is enabled	'h0	R/W	Public
0		Reserved	0		

ESD Caution

ESD (electrostatic discharge) sensitive device. Charged devices and circuit boards can discharge without detection. Although this product features patented or proprietary protection circuitry, damage may occur on devices subjected to high energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

Legal Terms and Conditions

By using the evaluation board discussed herein (together with any tools, components documentation or support materials, the "Evaluation Board"), you are agreeing to be bound by the terms and conditions set forth below ("Agreement") unless you have purchased the Evaluation Board, in which case the Analog Devices Standard Terms and Conditions of Sale shall govern. Do not use the Evaluation Board until you have read and agreed to the Agreement. Your use of the Evaluation Board shall signify your acceptance of the Agreement. This Agreement is made by and between you ("Customer") and Analog Devices, Inc. ("ADI"), with its principal place of business at One Technology Way, Norwood, MA 02062, USA. Subject to the terms and conditions of the Agreement, ADI hereby grants to Customer a free, limited, personal, temporary, non-exclusive, non-sublicensable, non-transferable license to use the Evaluation Board FOR EVALUATION PURPOSES ONLY. Customer understands and agrees that the Evaluation Board is provided for the sole and exclusive purpose referenced above, and agrees not to use the Evaluation Board for any other purpose. Furthermore, the license granted is expressly made subject to the following additional limitations: Customer shall not (i) rent, lease, display, sell, transfer, assign, sublicense, or distribute the Evaluation Board; and (ii) permit any Third Party to access the Evaluation Board. As used herein, the term "Third Party" includes any entity other than ADI, Customer, their employees, affiliates and in-house consultants. The Evaluation Board is NOT sold to Customer; all rights not expressly granted herein, including ownership of the Evaluation Board, are reserved by ADI. CONFIDENTIALITY. This Agreement and the Evaluation Board shall all be considered the confidential and proprietary information of ADI. Customer may not disclose or transfer any portion of the Evaluation Board to any other party for any reason. Upon discontinuation of use of the Evaluation Board or termination of this Agreement, Customer agrees to promptly return the Evaluation Board to ADI. ADDITIONAL RESTRICTIONS. Customer may not disassemble, decompile or reverse engineer chips on the Evaluation Board. Customer shall inform ADI of any occurred damages or any modifications or alterations it makes to the Evaluation Board, including but not limited to soldering or any other activity that affects the material content of the Evaluation Board. Modifications to the Evaluation Board must comply with applicable law, including but not limited to the RoHS Directive. TERMINATION. ADI may terminate this Agreement at any time upon giving written notice to Customer. Customer agrees to return to ADI the Evaluation Board at that time. LIMITATION OF LIABILITY. THE EVALUATION BOARD PROVIDED HEREUNDER IS PROVIDED "AS IS" AND ADI MAKES NO WARRANTIES OR REPRESENTATIONS OF ANY KIND WITH RESPECT TO IT. ADI SPECIFICALLY DISCLAIMS ANY REPRESENTATIONS, ENDORSEMENTS, GUARANTEES, OR WARRANTIES, EXPRESS OR IMPLIED, RELATED TO THE EVALUATION BOARD INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, TITLE, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. IN NO EVENT WILL ADI AND ITS LICENSORS BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES RESULTING FROM CUSTOMER'S POSSESSION OR USE OF THE EVALUATION BOARD, INCLUDING BUT NOT LIMITED TO LOST PROFITS, DELAY COSTS, LABOR COSTS OR LOSS OF GOODWILL. ADI'S TOTAL LIABILITY FROM ANY AND ALL CAUSES SHALL BE LIMITED TO THE AMOUNT OF ONE HUNDRED US DOLLARS ($100.00). EXPORT. Customer agrees that it will not directly or indirectly export the Evaluation Board to another country, and that it will comply with all applicable United States federal laws and regulations relating to exports. GOVERNING LAW. This Agreement shall be governed by and construed in accordance with the substantive laws of the Commonwealth of Massachusetts (excluding conflict of law rules). Any legal action regarding this Agreement will be heard in the state or federal courts having jurisdiction in Suffolk County, Massachusetts, and Customer hereby submits to the personal jurisdiction and venue of such courts. The United Nations Convention on Contracts for the International Sale of Goods shall not apply to this Agreement and is expressly disclaimed.

UG15388-0-3/17(0)

www.analog.com