AN-1427
APPLICATION NOTE

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

SensorStrobe, Ultra Low Power, Time Synchronized, Sensor Data Sampling in the ADuCM3027/ADuCM3029

INTRODUCTION

Precise sampling of sensors synchronized to an accurate time base is a requirement in a variety of wireless sensor network applications, such as structural health monitoring, wearable devices, and environment sensing. The sampling of sensor data is dictated by the microcontroller unit (MCU). In the traditional approach, the software on the MCU generates a general-purpose input/output (GPIO) pulse, triggering the sensors at specific intervals to collect sensor data.

The traditional approach has two issues. The approach involves considerable software overhead, which increases current consumption. The triggering of a pulse depends on the software of the MCU and, thus, can drift as time progresses.

This application note describes SensorStrobe®, a mechanism from Analog Devices, Inc., that recognizes low power, consistent, and synchronized data acquisition from sensors.

The SensorStrobe addresses the issues of the traditional software approach due to the following reasons:

- The SensorStrobe works in hibernate mode, resulting in a >10× reduction in the current consumption.
- No software intervention is required after setup.
- The pulse triggering mechanism is independent of the software execution, generating continuous triggers (and no drift) even during software execution.

This application note uses an example setup consisting of the ADuCM3027/ADuCM3029 MCUs connected to the ADXL363 accelerometer to prove the >10× reduction in current consumption, while acquiring the sample data using the SensorStrobe mechanism. This reduction is evident when using the SensorStrobe mechanism compared to a non SensorStrobe software approach. Note that throughout this application note, multifunction pins, such as XINT0_WAKE2/GPIO13, are referred to either by the entire pin name or by a single function of the pin, for example, XINT0_WAKE2, when only that function is relevant.

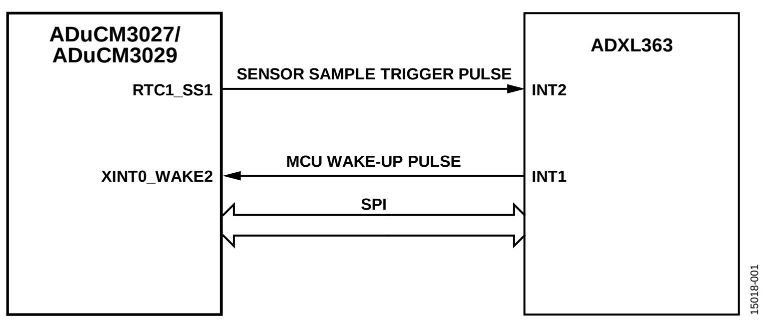

Figure 1. ADuCM3027/ADuCM3029 and ADXL363 Connection Diagram

TABLE OF CONTENTS

REVISION HISTORY

6/2018—Rev. 0 to Rev. A

Changes to System Description Section, Figure 3, and Table 1.. 5
Deleted Figure 8; Renumbered Sequentially 11
Changes to Power Measurement Section 12

3/2017—Revision 0: Initial Version

SENSORSTROBE OVERVIEW

SensorStrobe is a mechanism for sampling sensors in an efficient, low power, intrinsically synchronized manner. The ADuCM3027/ADuCM3029 support this mechanism. SensorStrobe can be used in the active, Flexi™ and hibernate power modes of the ADuCM3027/ADuCM3029.

The SensorStrobe mechanism allows the ADuCM3027/ ADuCM3029 to be in hibernate mode (750 nA), while the sensors collect data at periodic intervals.

The SensorStrobe mechanism is combined with the external trigger feature of the ADXL363 to collect the sensor data at the lowest possible power consumption.

SensorStrobe is an alarm function of the real-time clock (RTC) in the ADuCM3027/ADuCM3029. In this mechanism, the ADuCM3027/ADuCM3029 provide an external trigger for the ADXL363 accelerometer. The trigger is on the SPI1_CS1/ SYS_CLKOUT/GPIO43/RTC1_SS1 (RTC SensorStrobe) pin and is a one-cycle, high pulse of the low frequency clock source (32 kHz) driven out through a single GPIO on the ADuCM3027/ ADuCM3029. This pulse is periodic, ensuring no time variability in the sensor sampling with a high degree of configurability for the periodicity.

FEATURES OF THE ADXL363

The ADXL363 is an ultra low power, three sensor device combining a 3-axis microelectromechanical systems (MEMS) accelerometer, a temperature sensor, and an analog-to-digital converter (ADC) input for synchronized conversions of external signals.

The ADXL363 has a 512 sample, first in, first out (FIFO) buffer to store sensor data. This large FIFO saves power at the system level. The MCU can be in hibernate mode while the ADXL363 autonomously records data in the FIFO buffer.

The ADXL363 is configured in the external trigger mode. The ADuCM3027/ADuCM3029 generate these trigger pulses on the SPI1_CS1/SYS_CLKOUT/GPIO43/RTC1_SS1 pin. On each trigger, the ADXL363 collects and stores the data to the FIFO (up to 512 samples of two bytes each) buffer.

The ADXL363 is programmed to interrupt and wake up the MCU when the FIFO buffer reaches a watermark of 480 samples of two bytes each. Using the watermark feature leaves room in the FIFO buffer for more samples to be taken while the MCU wakes and begins draining the FIFO buffer.

The ADXL363 supports register read and write access over the serial peripheral interface (SPI). The access can be a single byte or multiple bytes. The FIFO buffer implementation is for consecutive samples to be read continuously via a multibyte read of unlimited length. Thus, one FIFO buffer read instruction can drain the entire contents of the FIFO buffer.

In other accelerometers, each read instruction retrieves only a single sample. In addition, the ADXL363 FIFO buffer can be drained using the ADuCM3027/ADuCM3029 direct memory access (DMA) controller.

The ADuCM3027/ADuCM3029 efficiently communicate with the ADXL363 using the read command mode in the SPI, which reduces the overall system power by lowering SPI protocol overhead.

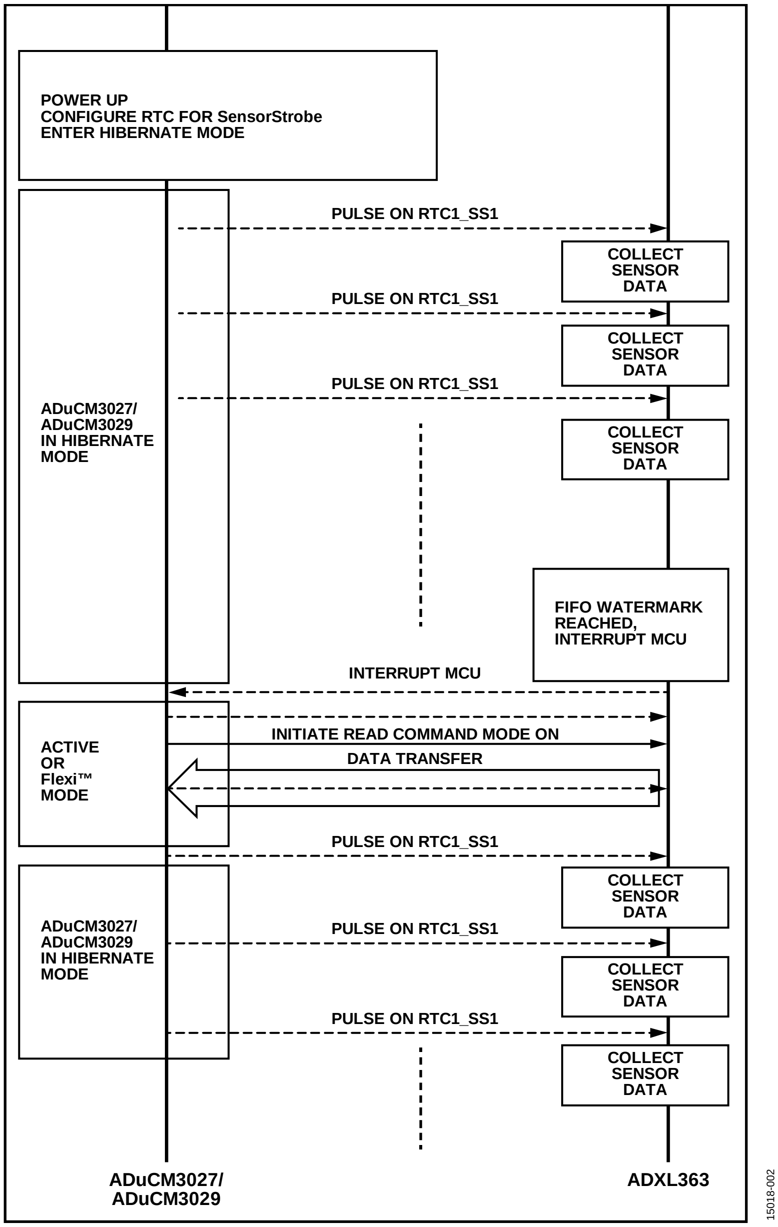

Figure 2. Data Sequence Diagram

SYSTEM DESCRIPTION

An example system was built to demonstrate the benefits of using the SensorStrobe. This system consists of an EV-COG-AD3029LZ connected with a multimeter. These system components connect in series to measure the system current consumption.

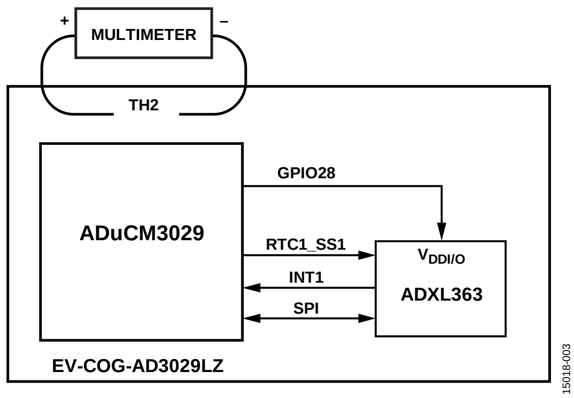

Figure 3. System Connection for Current Measurement

INTERFACE BETWEEN THE MCU AND THE ADXL363

The ADXL363 is configured for measurement mode and to interrupt the MCU when the FIFO watermark is reached. The ADXL363 configuration is further described in the Software Overview section.

The SensorStrobe mechanism on the ADuCM3027/ADuCM3029 is enabled, and the ADuCM3027/ADuCM3029 are placed in hibernate mode. The trigger pulse is generated at 128 Hz.

On every pulse, the ADXL363 takes a sample and stores the sample in the FIFO buffer. When the FIFO upper watermark is reached, the ADXL363 interrupts the ADuCM3027/ADuCM3029 via the XINT0_WAKE2/GPIO13 pin.

The ADuCM3027/ADuCM3029 use the read mode feature to drain the entire FIFO in a single command, minimizing the SPI protocol overhead. The DMA controller can drain the FIFO buffer, further reducing the active time and system current consumption of the MCU.

SensorStrobe enables the ADuCM3027/ADuCM3029 to generate trigger pulses on the SPI1_CS1/SYS_CLKOUT/GPIO43/RTC_SS1 pin, even in hibernate mode. The configuration of the pulse generation is in the RTC1 registers and GPIO pin multiplexing.

In Flexi mode, DMA can transfer SPI data, further reducing the power consumption of the system.

Table 1. Interface Connection Between the ADuCM3027/ADuCM3029 MCUs and the ADXL363 on EV-COG-AD3029LZ

MCU	ADXL363	Description
SPI1_MOSI/GPIO23	MOSI	SPI data (from the MCU to the ADXL363.
SPI1_MISO/GPIO24	MISO	SPI data (from the ADXL363 to the MCU).
SPI1_CLK/GPIO22	SCLK	SPI clock.
SPI0_CS1/SYS_CLKIN/SPI1_CS3/GPIO26	$\overline{CS}$	SPI chip select.
SPI1_CS1/SYS_CLKOUT/GPIO43/RTC1_SS1	INT2	RTC_SS1.
XINT0_WAKE2/GPIO13	INT1	The ADXL363 uses the INT1 pin to wake up the MCU from hibernate mode.
GPIO28	$V_{DDI/O}$	Power up the ADXL363 using GPIO.

DATA TRANSFER SEQUENCE

The sensor data collection by the MCU occurs in two phases. Figure 4 and Figure 5 show the activity of the signals during these phases.

First, the SPI1_CS1/SYS_CLKOUT/GPIO43/RTC1_SS1 pin acts as an external trigger for the ADXL363 to collect samples in the FIFO buffer. Second, the ADXL363 FIFO buffer reads the contents over the SPI.

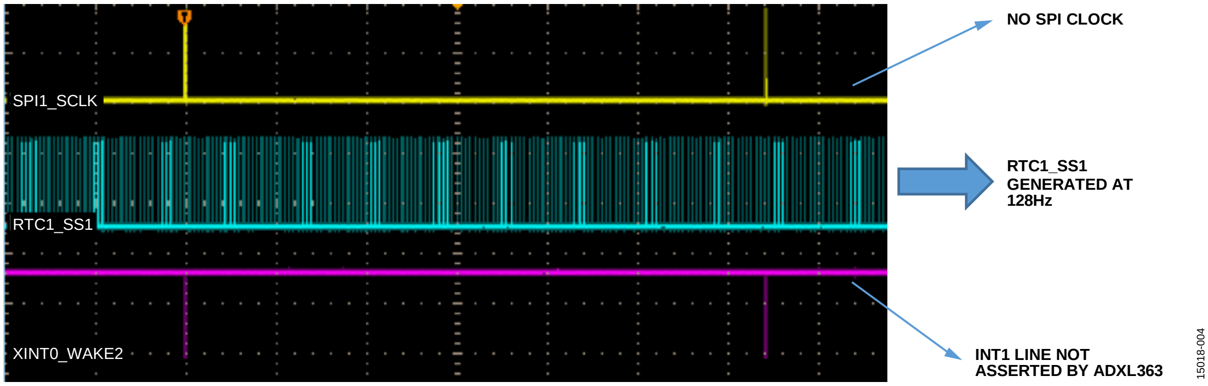

Figure 4. Phase 1—Data Acquisition Phase: RTC_SS1 Triggers to ADXL363

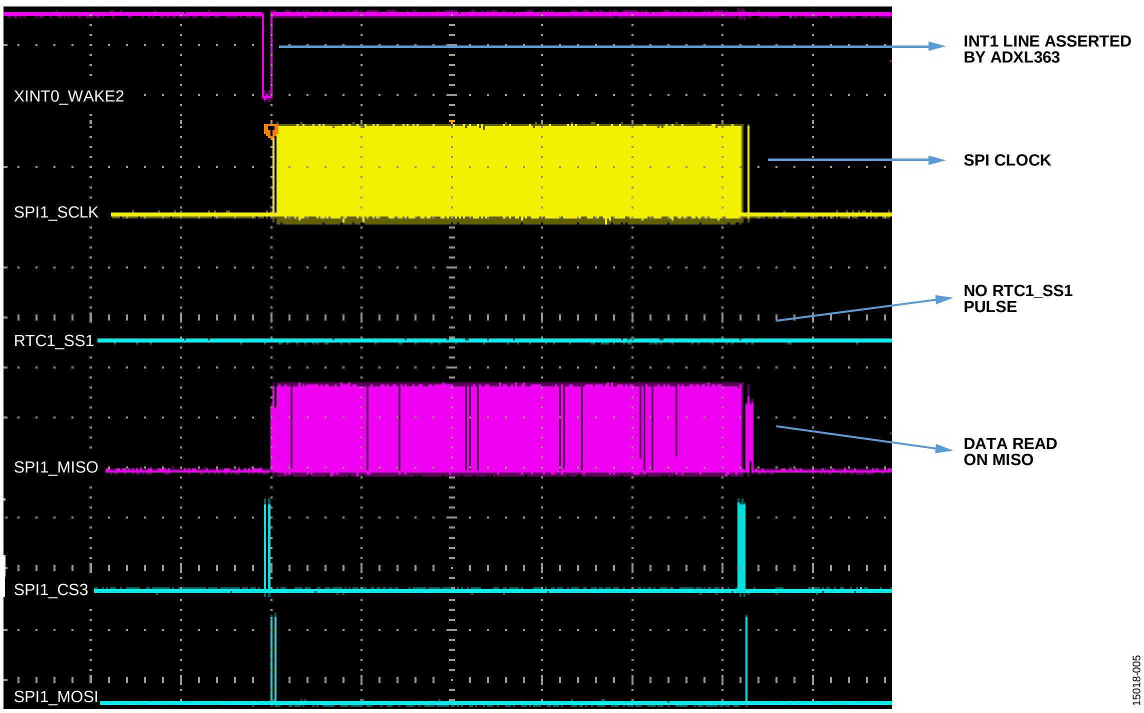

Figure 5. Phase 2—Data Transfer to the MCU: Reading the ADXL363 FIFO over the SPI

SOFTWARE OVERVIEW

This section describes the software flow in the example system.

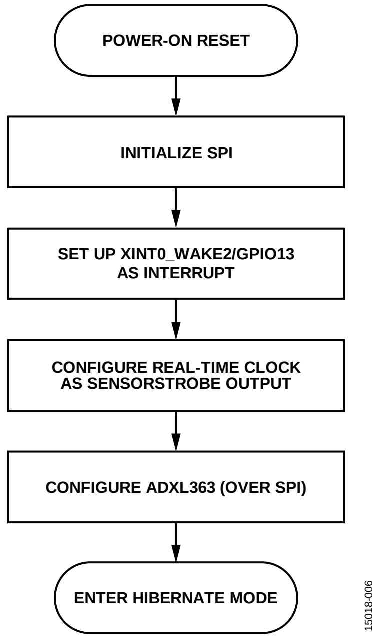

Figure 6. Initialization and Configuration Steps

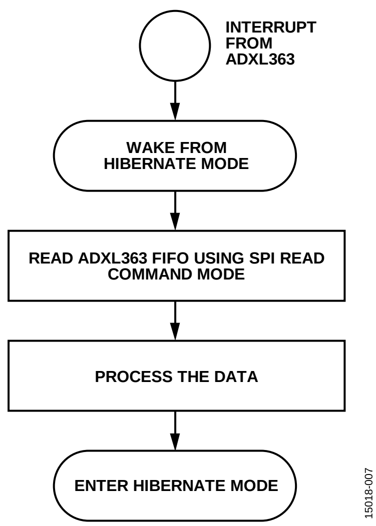

Figure 7. FIFO Read Over SPI (Using the Read Command): Reading Data from the ADXL363

SOURCE SNIPPETS

The reference source snippets for configuration and reading data are given in this section.

pREG_<module>_<name> is the board support package method of referring the register. The same register in the *ADuCM302x Ultra Low Power ARM Cortex-M3 MCU with Integrated Power Management Hardware Reference* is referred to as <module>_<name>.

RTC Configuration for SensorStrobe

```
#define PRD_VAL 255
void SensorStrobe_Cfg()
{
   // SensorStrobe Pin Mux
  *pREG_GPIO2_CFG |= (0x3 << BITP_GPIO_CFG_PIN11);
   // Reset the RTC counter
  while(*pREG_RTC1_SR1 & 0x0180); //wait until sync
    *pREG_RTC1_SR0 = 0xFF;
  *pREG_RTC1_CR0 = 0;
  while(!(*pREG_RTC1_SR0 & 0x0080 )); //wait for sync

  // SensorStrobe configuration
  // Initial trigger and auto reload value = 255 RTC counts
  // RTC runs at 32KHz ideally, 1 RTC count = 30.7us
  while(*pREG_RTC1_SR5 != 0);
  *pREG_RTC1_SS1ARL = PRD_VAL;

  // SensorStrobe to be triggered after 255 RTC counts
  *pREG_RTC1_CR4SS = (1 << 9); //Enable Autoreload
  *pREG_RTC1_SS1 = PRD_VAL; // Initial Compare Value

  // Enable SensorStrobe
  *pREG_RTC1_CR3SS = 1;
  while(*pREG_RTC1_SR4 != 0x77FF);

  // Initialize the counter value to zero
  while(*pREG_RTC1_SR1 & 0x0600);
  *pREG_RTC1_CNT0 = 0;
  *pREG_RTC1_CNT1 = 0;
  while(!(*pREG_RTC1_SR0 & 0x0400));

  // Enable (start) the counter
  while(*pREG_RTC1_SR1 & 0x0180); //wait until sync
  *pREG_RTC1_SR0 = 0xFF;
  *pREG_RTC1_CR0 = 1;
  while(!(*pREG_RTC1_SR0 & 0x0080)); //wait for sync
  return;
}
```

Configuration of the ADXL363

During the initialization phase, the ADXL363 is configured by the ADuCM3027/ADuCM3029 through the SPI. Every SPI transaction during the configuration consists of 2-byte transfers from the ADuCM3027/ADuCM3029. The first byte indicates the ADXL363 register address, and the second byte indicates the value to be written to the register.

The steps are shown in the following code snippet of the adxl_write_reg function. See the adxl_write_reg section for the code.

The adxl_configure function highlights the register writes required to configure the ADXL363, as used in the demonstration application. See the adxl_configure section for the code.

Refer to the ADXL363 data sheet for information on the registers and settings.

adxl_write_reg

```
void adxl_write_reg (unsigned char reg, unsigned char data)
{
  adxl_access(1);   //Enable chipselect
  spi_byte_tx(0x0A);       //Enable write
  spi_byte_tx(reg);        //Write register
  spi_byte_tx(data);       //Write data
  adxl_access(0);   //Disable chipselect
}
```

adxl_configure

```
void adxl_configure()
{
  // Softreset
  adxl_write_reg (0x1F,0x52);
  // Activity configuration
  adxl_write_reg (0x27,0x35);
  // FIFO configuration
  adxl_write_reg (0x28,0x00);
  adxl_write_reg (0x28,0xFF);
  // FIFO samples configuration
  adxl_write_reg (0x29,0xDF);
  // FIFO_watermark int
  adxl_write_reg (0x2A,0x84);
  // Filter control
  adxl_write_reg (0x2C,0x08);
  // Power control
  adxl_write_reg (0x2D,0x06);
}
```

THE ADXL363 FIFO READ

When the ADXL363 reaches the FIFO watermark (960 bytes), an interrupt is triggered to wake the MCU, which drains the ADXL363 FIFO via the SPI.

```
void ReadFifo_adxl()
{
  int j=0;
  // SPI2 configuration
   *pREG_SPI2_CTL = 0x0803;
  // Number of bytes to read
  *pREG_SPI2_CNT = 960;
  *pREG_SPI2_IEN = 7;
  // Enable Read command mode
  *pREG_SPI2_RD_CTL = 1;
  // Enable ADXl Chipselect
  adxl_access(1);
  // Flush out Rx FIFO
  while(*pREG_SPI2_FIFO_STAT & 0xF00)
        *pREG_SPI2_RX;
  // 0x0D written to kickstart the FIFO read from ADXL363
  *pREG_SPI2_TX = 0x0D;
  // Dummy read
  *pREG_SPI2_RX;
  // Number of samples
  while(j < 120)
  {if(*pREG_SPI2_STAT & BITM_SPI_STAT_RXIRQ)
    {*pREG_SPI2_STAT |= BITM_SPI_STAT_RXIRQ;
      i=0;
      while(i<8)
       {*(data_ref + (8 * j) + i) =
                            *pREG_SPI2_RX;
        i++ ; }
      j++;}
   }
  adxl_access(0);
  // Reset SPI
  *pREG_SPI2_RD_CTL = 0;
  *pREG_SPI2_IEN = 0;
  *pREG_SPI2_CTL = 0x0843;
}
```

SYSTEM POWER ANALYSIS

The ADuCM3027/ADuCM3029 host processor is responsible for the following:

- Switching between power modes (for example, active and hibernate modes, as required).
- Configuring the RTC to generate sample trigger pulses to the ADXL363.
- Controlling SPI communication with the ADXL363.
- Storing the raw data from the ADXL363 (processing or transmitting resultant data is not implemented in the system described in this application note).

The ADXL363 sensor is responsible for the following:

- Sampling and storing raw sensor data to the FIFO buffer when triggered by the ADuCM3027/ADuCM3029.
- Responding to SPI communication from the host processor.
- Interrupting and waking up the host processor as the FIFO buffer fills.

POWER MEASUREMENT

The following steps describe how to monitor the current consumption of the system:

1. Load the application code into the MCU.
2. Connect the positive terminal of the multimeter to the left pin of TH2 on the EV-COG-AD3029LZ.
3. Connect the other end of the multimeter to the right pin of TH2 on the EV-COG-AD3029LZ.
4. Click the **RESET** button on the board.
5. Monitor the current consumption on the multimeter.

Table 2 shows the current consumption of both the ADuCM3027/ADuCM3029 and the ADXL363 in the different power modes used in this application note.

Figure 8 and Figure 9 show the difference in power consumption during the sensor data collection, with and without the SensorStrobe mechanism, respectively. All the measurements are performed on the ADuCM3027/ADuCM3029 EV-COG-AD3029LZ.

Power Measurement with SensorStrobe

Figure 8 shows the current profile of the demonstration system when using SensorStrobe methodology. SensorStrobe enables the host processor to remain in hibernate mode while generating trigger pulses driving the sensor to capture and store data samples. The average current measured is approximately 4.2 µA.

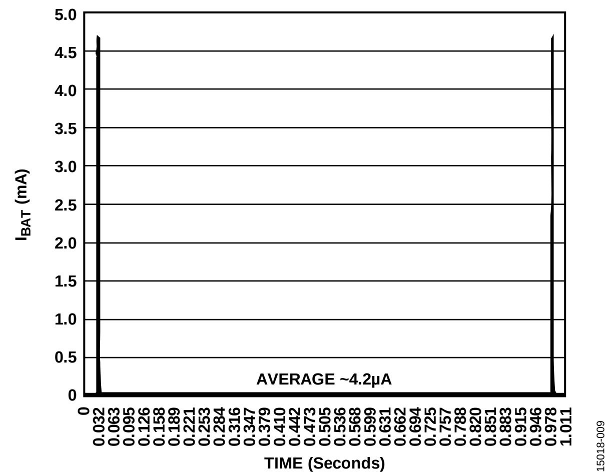

Figure 8. Power Measurement with SensorStrobe

Power Measurement Without SensorStrobe

Figure 9 shows the current profile of the sample acquisition system when not using SensorStrobe. In the absence of the SensorStrobe mechanism, the host processor generates the trigger pulses by waking up on an RTC interrupt and toggling a GPIO pin. The average current measured is approximately 75 µA.

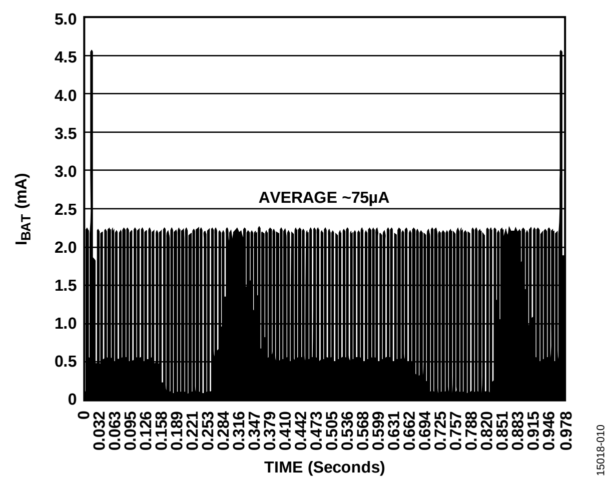

Figure 9. Power Measurement Without SensorStrobe

Table 2. Power Consumption of the ADuCM3027/ADuCM3029 and the ADXL363

Device	Mode	Current Consumption	Description
ADuCM3027/ ADuCM3029	Hibernate mode	750 nA	Power of core and peripherals gated with 8 kB random access memory (RAM) is retained and the low frequency crystal enabled.
	Flexi mode	300 µA	The core is clock gated, but the remainder of the system is active. DMA transfers can continue between peripherals and memory.
	Active mode	30 µA/MHz	Full system is active.
ADXL363	Normal mode	1.8 µA	Measurement mode.

CONCLUSION

With the SensorStrobe mechanism, the current consumption of a sensor sample acquisition system reduces drastically. In the demonstration system explained in this application note, the average current consumption reduces from 75 μA to 4.2 μA.

The SensorStrobe mechanism of the ADuCM3027/ADuCM3029 helps designers leverage the low power feature of the ADXL363, which results in a reduction of >10× in current consumption.

The SensorStrobe mechanism of the ADuCM3027/ADuCM3029 benefits an ultra low power system by reducing the overall current consumption, which extends the battery life of the system. Some of the applications that can leverage this benefit are mentioned in Structural Health Monitoring (SHM) section, the Healthcare Monitoring section, and the Environmental Sensing section.

STRUCTURAL HEALTH MONITORING (SHM)

Sensor nodes in SHM are often deployed in unpowered structures such as bridges, towers, or geographically remote locations. Periodic battery replacement is a hindrance and increases the lifetime maintenance cost. A sensor node with a longer battery life reduces maintenance costs considerably.

HEALTHCARE MONITORING

Healthcare monitoring applications involve monitoring the body position measurement, the location of the person, and the vital signs of the patient. These monitoring devices are usually wearable or implant devices. Reducing the system current consumption increases the battery health of these devices.

ENVIRONMENTAL SENSING

Various environmental sensing applications, such as air pollution monitoring, forest fire detection, and landslide detection, require sensor node deployment in remote areas. These remote areas are often not line powered. Extending the battery life of these nodes reduces maintenance cost.

AN15018-0-6/18(A)