Application Note

Developing a Custom Multi-Channel SPI for Connecting Four ADC Channels to an MCU/MPU

Thomas Mauer and Alexander Walton

ABSTRACT

This application note presents a comprehensive guide on interfacing four synchronous ADCs with a 4-channel serial SPI communication to an MCU/MPU without requiring additional logic. The ADCs are synchronized using a common clock and start-of-conversion signal, along with a shared SDI data line. The captured ADC data from the SDO lines of all four ADCs is simultaneously collected by the microprocessor using the integrated PRU-ICSS peripheral. The MCU efficiently acquires the 4-channel ADC data, which is stored in configurable dual circular buffers, and subsequently performs Fast Fourier Transform (FFT) computations on the acquired data. The resultant outcomes are then transmitted via a Single Pair Ethernet (SPE) connection, utilizing JSON and MQTT protocols for data exchange and communication.

Table of Contents

Trademarks

Sitara™ is a trademark of Ti.
All trademarks are the property of their respective owners.

1 Introduction

Sensors are linked via an analog front-end (AFE) to facilitate signal conditioning and bias voltage adjustments for sensor data. In the case of IEPE sensors, which produce low voltages, amplification is necessary prior to Analog-to-Digital Conversion (ADC) for capturing sensor samples. These samples are subsequently made accessible to a microprocessor for further signal processing.

A significant technical challenge arises when aiming to synchronize and collect data from four parallel ADCs over a serial interface. This is due to the requirement of capturing data from four individual lines while utilizing the serial capabilities of the microcontroller's peripherals, such as the Serial Peripheral Interface (SPI). Addressing this challenge necessitates a microcontroller that supports four SPI data input lines, or involves additional technical complexities like external hardware incorporation or ADC daisy chaining. However, these solutions can hinder SPI communication speed for reading ADC samples or result in increased Bill of Materials (BOM) costs.

This application note explains the microprocessor's role in interfacing with these four ADC data streams and outlines how to parallelly connect the SPI data lines of all four ADCs to the microprocessor. This configuration enables rapid retrieval of ADC samples. The SPI interface is implemented utilizing Sitara's Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS), which empowers the incorporation of FPGA-like functionality within a real-time core on the MCU chip. Subsequently, the ADC sample buffers/bins are processed on the MCU using Fast Fourier Transformation (FFT), and the results are transmitted via JSON/MQTT protocols over a single pair Ethernet connection to an SPE gateway. This gateway facilitates the visualization and analysis of vibration sensor samples.

The described use case facilitates predictive maintenance by allowing the identification of motor, generator, and gearbox degradation during operation. This timely recognition permits the scheduling of maintenance before system failure occurs. As a result, maintenance costs are minimized, and the risk of unexpected machinery breakdowns is mitigated.

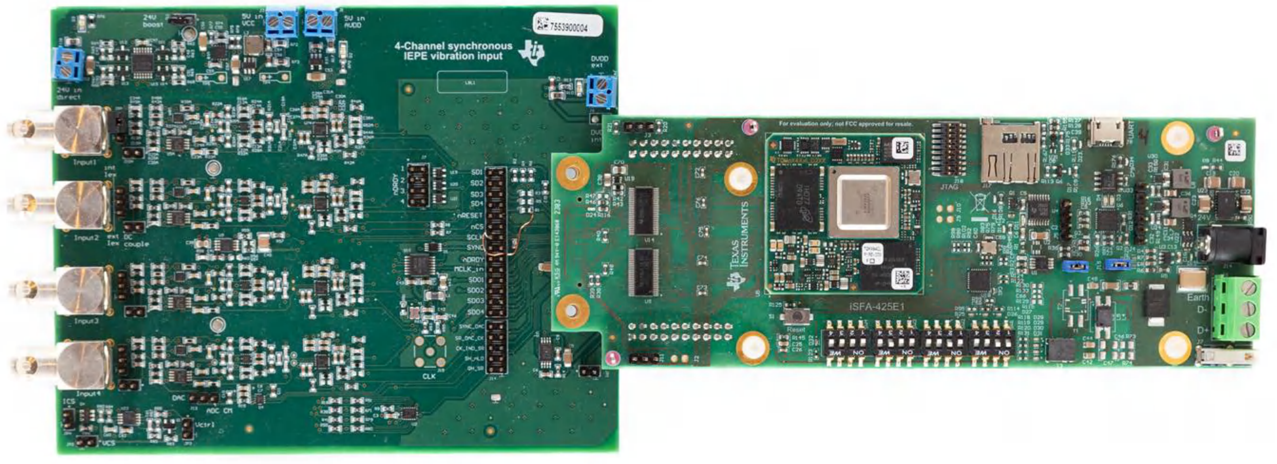


Figure 1-1. TIDA-010249 and TIDA-010261 EVMs Utilizing the Multi-Channel SPI

Figure 1-1 illustrates the configuration of the TIDA-010249 and TIDA-010261 Evaluation Modules (EVMs) employing the multi-channel SPI interface. The TIDA-010249 EVM comprises the four-channel ADCs, while the TIDA-010261 EVM features the AM243x microprocessor equipped with ARM Cortex R5F and PRU-ICSSG. The interconnection of these components forms the foundation for achieving synchronized ADC data acquisition and high-speed communication with the microprocessor for subsequent signal processing.

2 Implementation

System introduction

The Texas Instruments AM243x microcontroller device contains two instances of the PRU-ICSS (ICSSG0) peripheral, alongside 2 to 4 ARM R5F cores depending on the part name version.

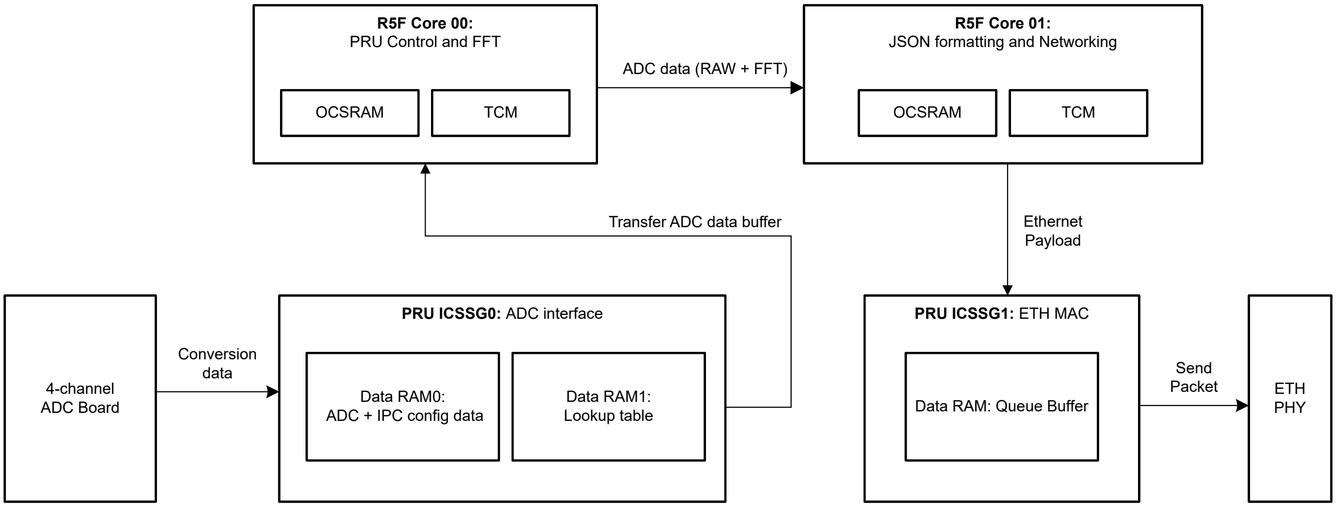


Figure 2-1. System Block Diagram of the Sensor Processing Board

Figure 2-1 offers insight into the system block diagram. One instance of PRU-ICSS interfaces with the 4-channel IEPE ADC board, serving as the analog front-end for IEPE sensors. The upcoming sections delve into the particulars of this 4-channel SPI interface, showing the implementation towards the ADC boards. The second PRU-ICSS instance functions as an Ethernet MAC, establishing connectivity with the single-pair Ethernet PHY. This connection facilitates the transmission of ADC samples and FFT data to the SPE gateway board.

To effectively distribute the workload, two ARM R5F cores come into play. R5F core 00 undertakes tasks such as ADC interface initialization (including configuration), reception of the four ADC sample buffers, and the execution of FFT calculations on these buffers. Results from the FFT computations and the unprocessed ADC data are made accessible to the second R5F core, labeled as R5F core 01. The latter core is responsible for tasks like JSON formatting and MQTT networking, ultimately transmitting the Ethernet frame through the MAC network interface hosted on the second PRU-ICSS instance (ICSSG1).

This application note focuses primarily on the PRU-ICSSG0 ADC interface in conjunction with the 4-channel ADC board.

PRU-ICSS Multi-Channel SPI to ADC Interface

Figure 2-2 provides a simplified representation of the hardware signal connections between the four ADCs and PRU-ICSS over the custom 4-channel SPI interface.

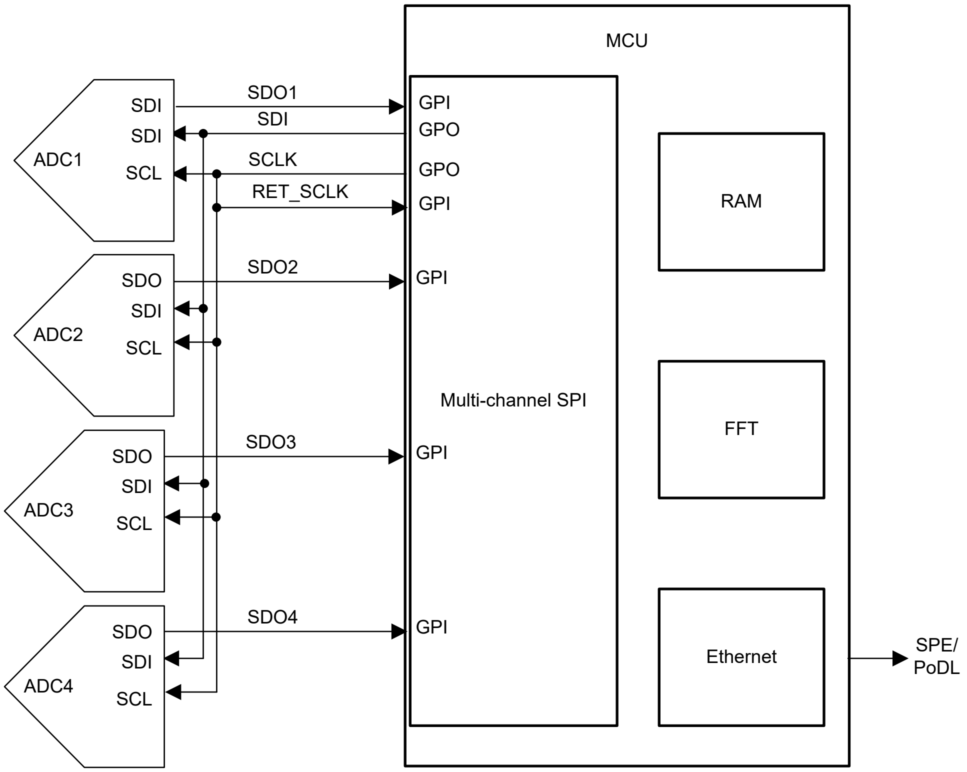


Figure 2-2. Four ADCs to Multi-Channel SPI using PRU-ICSS

The implementation of the multi-channel SPI function is facilitated by PRU-ICSS firmware utilizing dedicated General Purpose Input (GPI) and General Purpose Output (GPO) signals. These GPI and GPO signals are mapped to PRU registers and offer minimal latency, enabling real-time control of external peripherals through firmware. In this context, the GPI and GPO signals are leveraged to emulate an SPI interface.

The system employs a unified Serial Data Out (SDO) line from PRU-ICSS to all ADCs. This design choice permits simultaneous configuration of all ADCs with identical settings, essential for synchronized sampling. The PRU accesses the SDO line through the R30 register.

SPI lines 1 to 4 from each ADC are channeled into dedicated GPI lines on the PRU-ICSS, and these lines are read by the PRU firmware via the R31 register. The subsequent sections delve into the intricate process of converting the serial data stream from each SDI line into parallel 24-bit ADC data.

The Serial Clock (SCLK) signal is generated by the PRU-ICSS. Two options exist for generating SCLK:

- The PRU can manipulate the GPO via firmware to set and clear the line. This entails an active involvement of the PRU to manage the line's state.
- The Integrated Event Processor (IEP) timer independently generates the clock, decoupling the clock from PRU firmware activity.

This system implementation employs the IEP timer for generating SCLK.

The Return_SCLK signal serves as the clock signal for data sampling in the PRU firmware. Return_SCLK is derived from SCLK and transmitted over the high-speed connector. This choice proves advantageous when utilizing high-speed clocks above 10 MHz. Isolation devices introduce signal delays for all signals traversing the isolation barrier. Employing Return_SCLK for sampling permits the PRU-ICSS to capture the exact clock edge used by the ADC to output sampling data on the SDI lines, thereby achieving accurate data capture.

Figure 2-3 visually depicts the transfer of SCLK, Return_SCLK, and ADC data samples over the SDO lines. SDO1 to SDO4 and Return SCLK are input to the PRU-ICSS via GPIs. SCLK clock signal is generated via GPO in PRU-ICSS. Return SCLK is utilized by PRU firmware to capture SDO data bits.

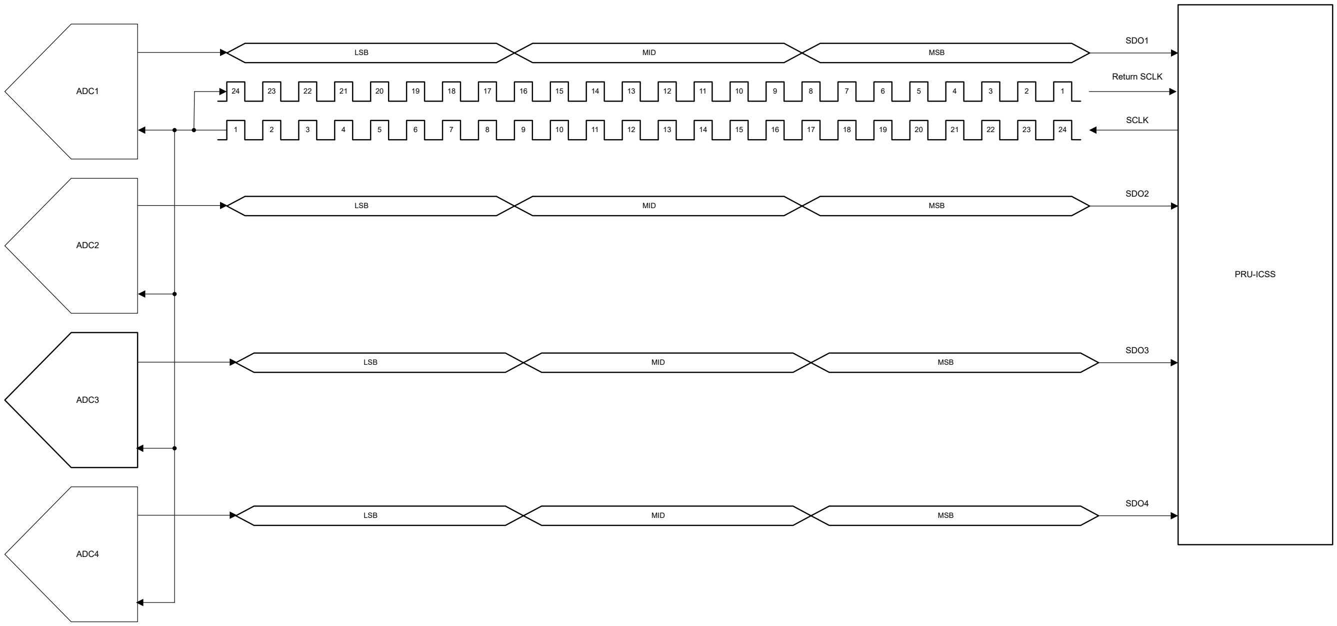


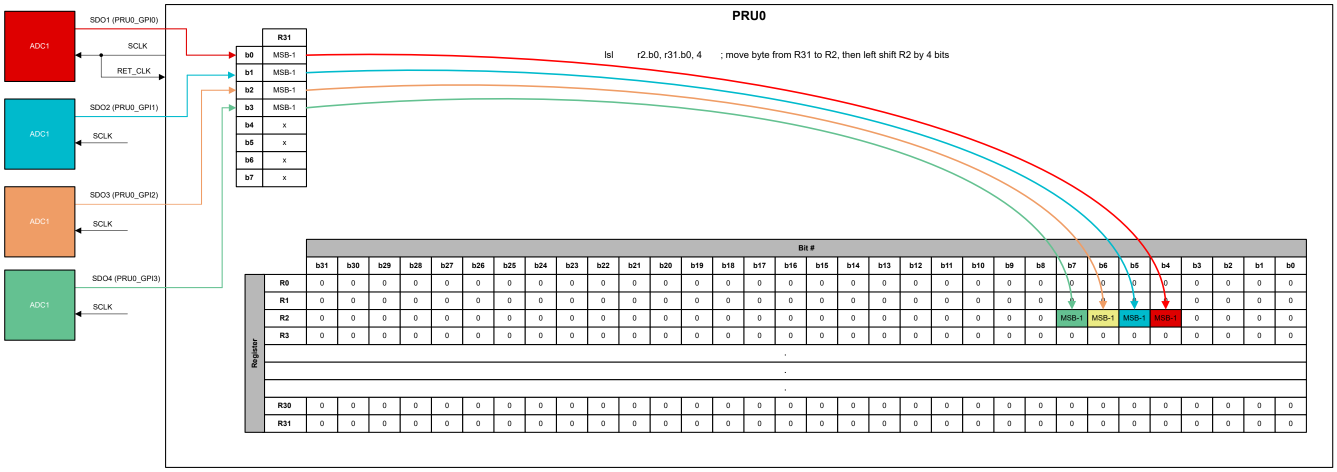
Figure 2-3. ADC Samples Over SDO Data Lines with SCLK and Return SCLK

Each ADC furnishes 24-bit data samples, comprising Least Significant Bit (LSB), Middle Bits (MID), and Most Significant Bit (MSB). These data samples are placed on the SDO lines using the SCLK clock signal. Thus, a continuous 24-clock cycle SCLK signal is necessary, prompting the ADCs to present all data samples on the SDO line.

The PRU firmware undertakes the subsequent tasks:

- Initiates SCLK clock output by enabling IEP SYNC output.
- Utilizes Return_SCLK to sample data bits across SDO1 to SDO4.
- Reorganizes the bit stream from each SDO line to construct a 24-bit ADC sample for each channel.
- Stores these 24-bit samples in a shared RAM data buffer.
- Upon filling the designated buffer, dispatches an event via the INTC to the ARM processor, triggering subsequent actions within the ARM application (such as FFT calculations).

Accessing the SDO lines directly involves utilizing bits b0 to b3 of register R31. PRU firmware samples data by copying and left-shifting the lower 8 data bits to register R2 (byte 0), as demonstrated in Figure 2-4. After sampling, the initial 1-bit of data for each channel is located in R2 from bit 4 to bit 7.

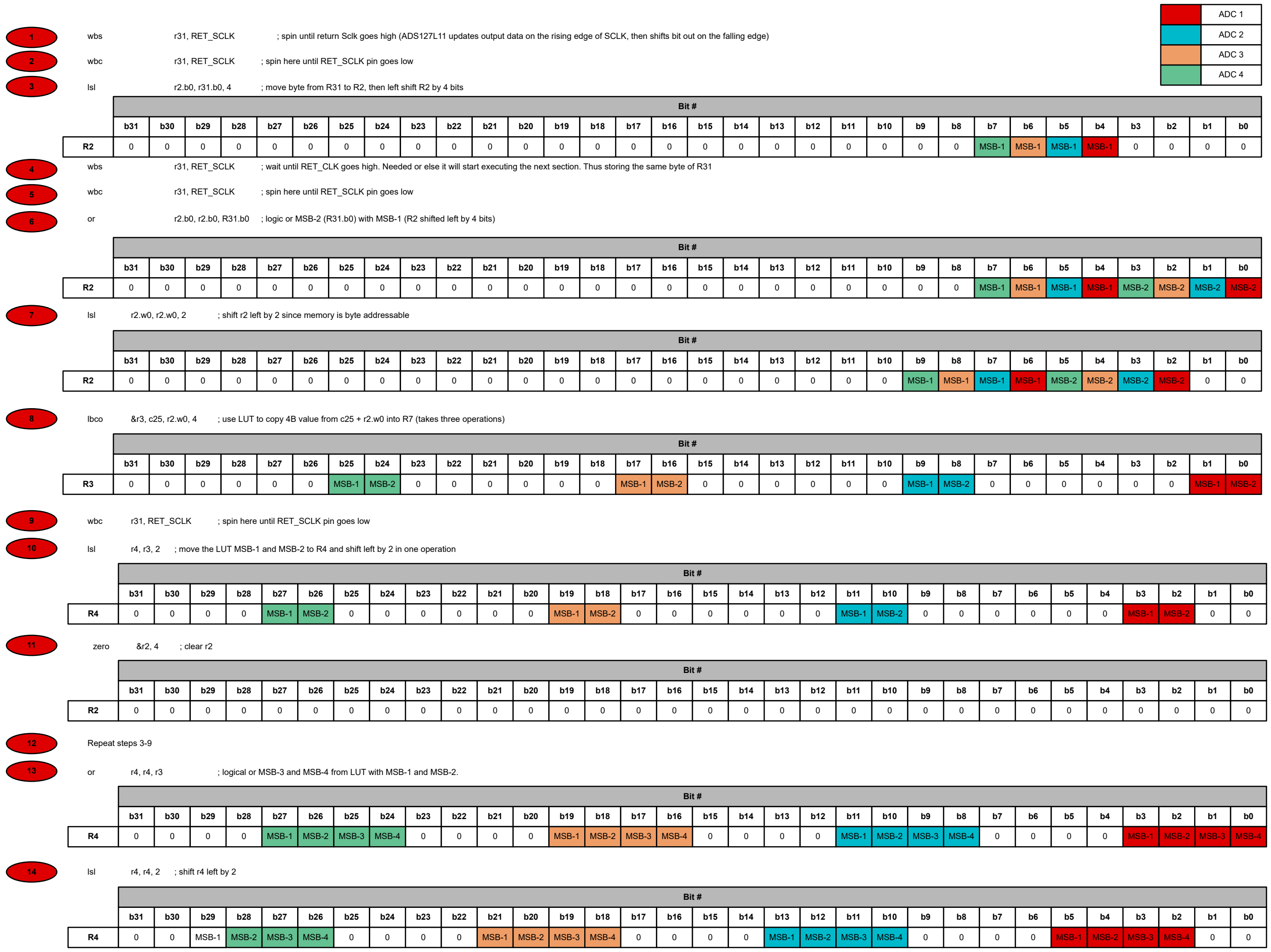


```
lsl      r2.b0, r31.b0, 4      ; move byte from R31 to R2, then left shift R2 by 4 bits
```

	R31
b0	MSB-1
b1	MSB-1
b2	MSB-1
b3	MSB-1
b4	x
b5	x
b6	x
b7	x

		Bit #																															
		b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Register	R0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	R1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSB-1	MSB-1	MSB-1	MSB-1	0	0	0	0
	R3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		.																															
		.																															
		.																															
	R30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	R31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 2-4. PRU GPO Mapping and Sampling



🟥	ADC 1
🟦	ADC 2
🟧	ADC 3
🟩	ADC 4

```
1   wbs    r31, RET_SCLK       ; spin until return Sclk goes high (ADS127L11 updates output data on the rising edge of SCLK, then shifts bit out on the falling edge)
2   wbc    r31, RET_SCLK       ; spin here until RET_SCLK pin goes low
3   lsl    r2.b0, r31.b0, 4    ; move byte from R31 to R2, then left shift R2 by 4 bits
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSB-1	MSB-1	MSB-1	MSB-1	0	0	0	0

```
4   wbs    r31, RET_SCLK       ; wait until RET_CLK goes high. Needed or else it will start executing the next section. Thus storing the same byte of R31
5   wbc    r31, RET_SCLK       ; spin here until RET_SCLK pin goes low
6   or     r2.b0, r2.b0, R31.b0   ; logic or MSB-2 (R31.b0) with MSB-1 (R2 shifted left by 4 bits)
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSB-1	MSB-1	MSB-1	MSB-1	MSB-2	MSB-2	MSB-2	MSB-2

```
7   lsl    r2.w0, r2.w0, 2     ; shift r2 left by 2 since memory is byte addressable
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSB-1	MSB-1	MSB-1	MSB-1	MSB-2	MSB-2	MSB-2	MSB-2	0	0

```
8   lbco   &r3, c25, r2.w0, 4  ; use LUT to copy 4B value from c25 + r2.w0 into R7 (takes three operations)
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R3	0	0	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2

```
9   wbc    r31, RET_SCLK       ; spin here until RET_SCLK pin goes low
10  lsl    r4, r3, 2   ; move the LUT MSB-1 and MSB-2 to R4 and shift left by 2 in one operation
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2	0	0	0	0	0	0	MSB-1	MSB-2	0	0

```
11  zero   &r2, 4     ; clear r2
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
12  Repeat steps 3-9
13  or     r4, r4, r3          ; logical or MSB-3 and MSB-4 from LUT with MSB-1 and MSB-2.
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4

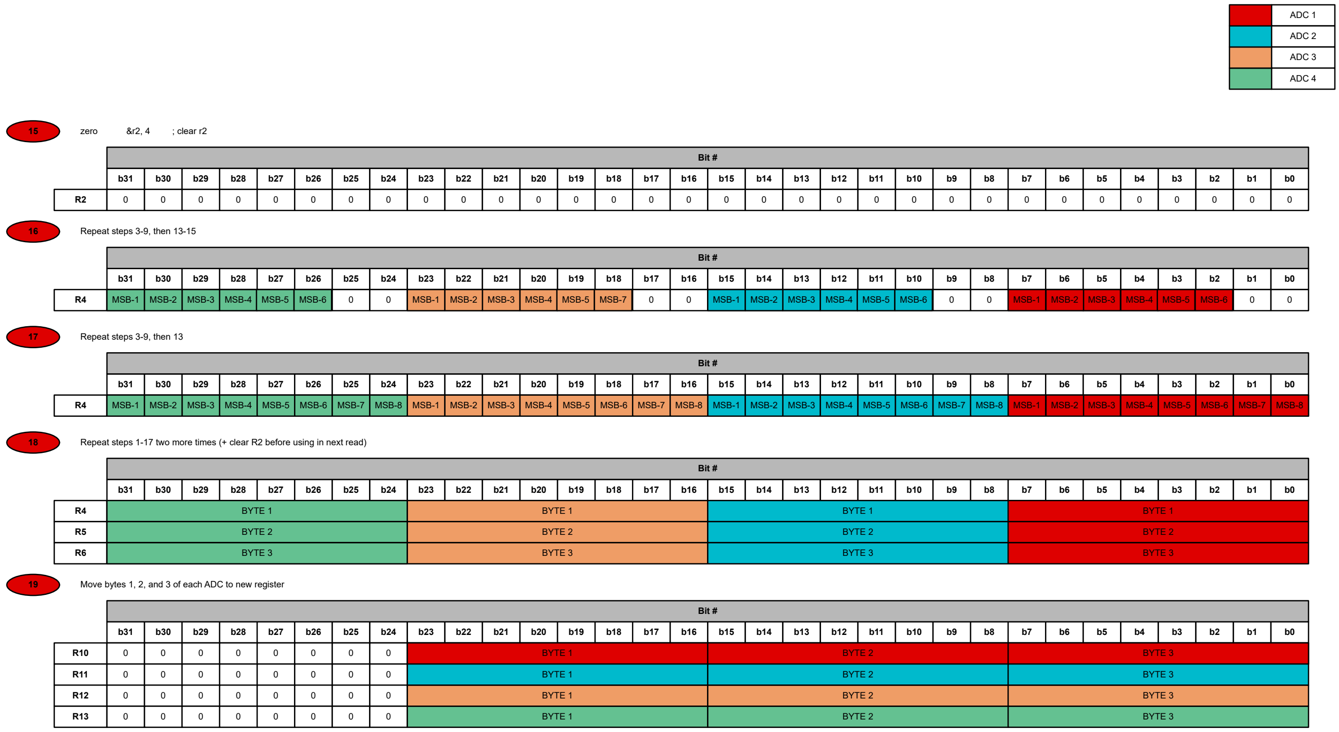
```
14  lsl    r4, r4, 2   ; shift r4 left by 2
```

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0	0	0	MSB-1	MSB-2	MSB-3	MSB-4	0	0

Figure 2-5. Transposing Conversion Steps of the Data Samples – Part 1

The PRU firmware executes the following task seen in Figure 2-5:

1. Waits for RET_SCLK to transition to 'high' before sampling MSB-1 in the subsequent step.
2. Awaits the RET_SCLK to go 'low' to perform data sampling.
3. Samples four SDO data bits by left-shifting 4 bytes and copying the outcome into register R2, byte 0. The color-coded cells depict the bit location of each ADC sample bit.
4. Awaits RET_SCLK to transition to 'high' again.
5. Waits for RET_SCLK to go 'low' before proceeding to sample MSB-2.
6. Samples the next four data bits by performing a logic OR operation between R31 byte 0 and R2 byte 0. The result of the OR operation is stored in R2 byte 0.
7. Prepares for lookup table access by left-shifting R2 by 2 bits.
8. Utilizes the 2-bit samples of each channel with the lookup table to rearrange them into consecutive 2 bits, using the sample bits as offsets for the lookup table. The memory access loads into register R3, showcasing the two consecutive bits for each channel.
9. Awaits RET_SCLK to transition to 'high'.
10. Left-shifts register R3 by two bits and stores the outcome in register R4.
11. Resets all values in register R2 to 0.
12. Repeats steps 3 to 9.
13. After repeating step 9, performs a logical OR operation between register R3 and R4, storing the result in R4. This step populates R4 with sorted MSB-1 to MSB-4 sample bits for each ADC channel.
14. Left-shifts register R4 by two bits and stores the result in R4.



🟥	ADC 1
🟦	ADC 2
🟧	ADC 3
🟩	ADC 4

15 zero &r2, 4 ; clear r2

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

16 Repeat steps 3-9, then 13-15

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	0	0	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-7	0	0	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	0	0	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	0	0

17 Repeat steps 3-9, then 13

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	MSB-7	MSB-8	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	MSB-7	MSB-8	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	MSB-7	MSB-8	MSB-1	MSB-2	MSB-3	MSB-4	MSB-5	MSB-6	MSB-7	MSB-8

18 Repeat steps 1-17 two more times (+ clear R2 before using in next read)

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R4	BYTE 1								BYTE 1								BYTE 1								BYTE 1							
R5	BYTE 2								BYTE 2								BYTE 2								BYTE 2							
R6	BYTE 3								BYTE 3								BYTE 3								BYTE 3							

19 Move bytes 1, 2, and 3 of each ADC to new register

	Bit #																															
	b31	b30	b29	b28	b27	b26	b25	b24	b23	b22	b21	b20	b19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
R10	0	0	0	0	0	0	0	0	BYTE 1								BYTE 2								BYTE 3							
R11	0	0	0	0	0	0	0	0	BYTE 1								BYTE 2								BYTE 3							
R12	0	0	0	0	0	0	0	0	BYTE 1								BYTE 2								BYTE 3							
R13	0	0	0	0	0	0	0	0	BYTE 1								BYTE 2								BYTE 3							

Figure 2-6. Transposing Conversion Steps of the Data Samples – Part 2

1. Clears register R2.
2. Repeats steps 3 to 9, then repeats steps 13 to 15.
3. Repeats steps 3 to 9, then performs step 13. At this stage, R4 contains 8 bytes of data samples, sorted for each ADC channel.
4. Repeats steps 1 to 17 two more times. Clears R3 before next use. At this point, PRU has accumulated and sorted 24 bits of samples. A complete ADC sample for every ADC resides in PRU registers R4 to R6.
5. Transfers data bytes into individual registers, enabling the PRU to store each 24-bit sample, converted to 32-bit registers, in the ICSS shared memory. Please refer to Figure 2-6 for a visual representation of this process.

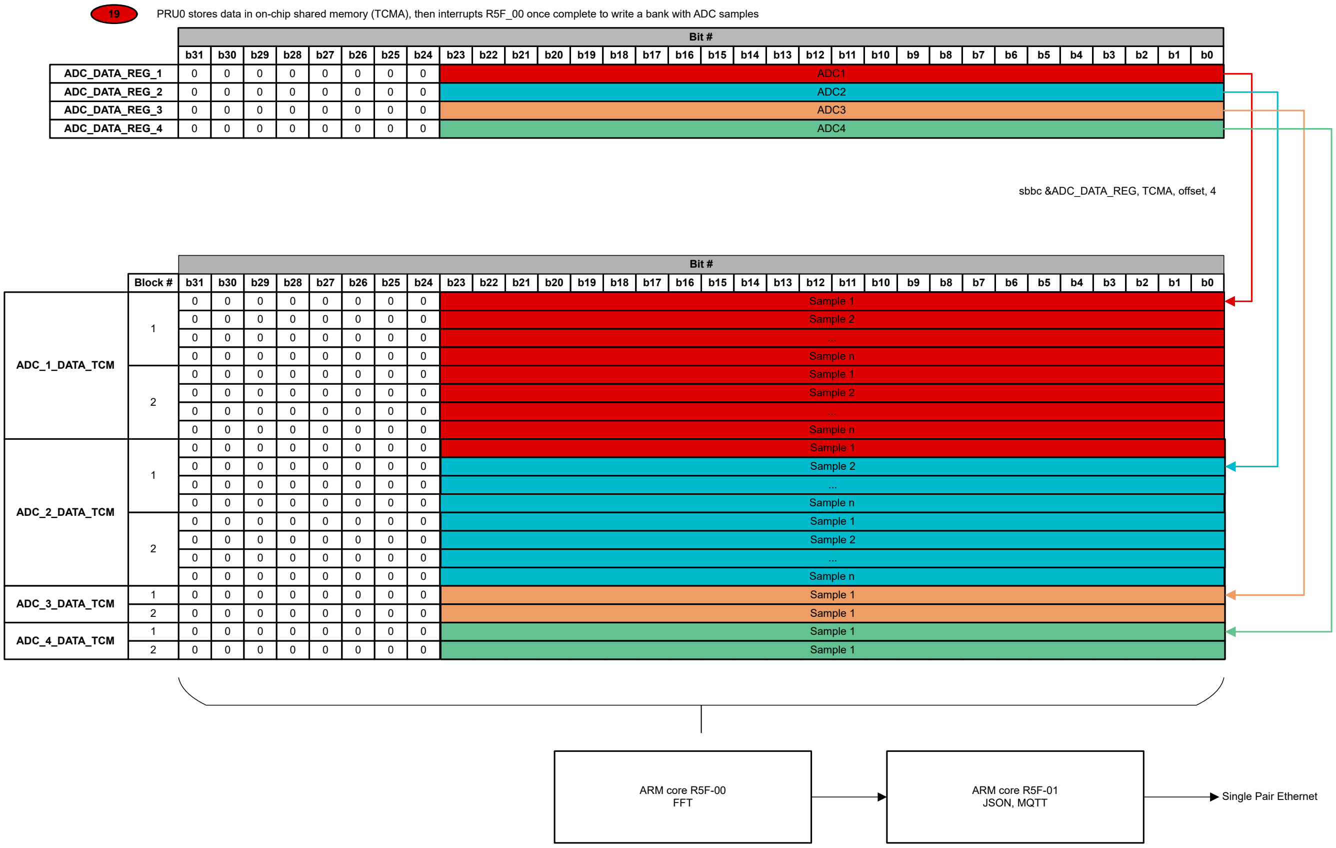

19
PRU0 stores data in on-chip shared memory (TCMA), then interrupts R5F_00 once complete to write a bank with ADC samples
Bit #
b31 b30 b29 b28 b27 b26 b25 b24 b23 b22 b21 b20 b19 b18 b17 b16 b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
ADC_DATA_REG_1
ADC_DATA_REG_2
ADC_DATA_REG_3
ADC_DATA_REG_4
ADC1
ADC2
ADC3
ADC4
sbbc &ADC_DATA_REG, TCMA, offset, 4
Block #
ADC_1_DATA_TCM
ADC_2_DATA_TCM
ADC_3_DATA_TCM
ADC_4_DATA_TCM
Sample 1
Sample 2
...
Sample n
ARM core R5F-00
FFT
ARM core R5F-01
JSON, MQTT
Single Pair Ethernet

Figure 2-7. Storing Transposed ADC Samples into Shared RAM

Following step 19, the PRU firmware executes four sbco instructions to store each 32-bit sample into the data buffer located in shared RAM. The PRU repeats this process, writing 32-bit data into the buffer multiple times until a complete sample buffer is filled. Please refer to Figure 2-7 for a visual representation of this process.

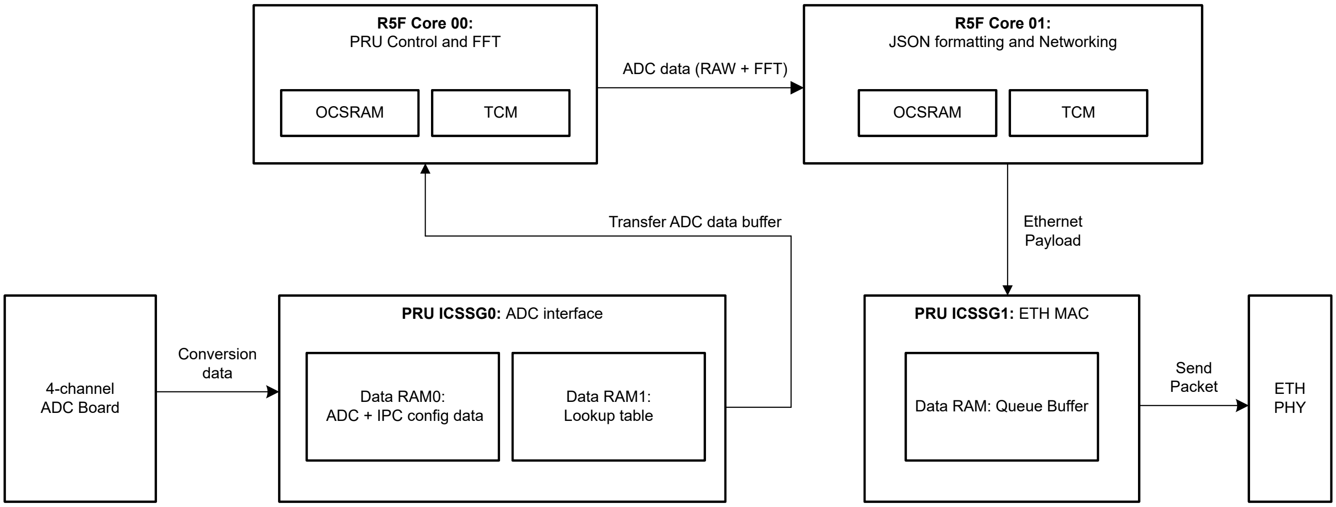


Figure 2-8. Complete Software Flow

Figure 2-8 illustrates the comprehensive software flow of the entire system. Following the completion of the sample buffer, the PRU triggers an Interrupt Controller (INTC) event towards the ARM. The ARM core R5F-00 receives this event as an interrupt, subsequently performing Fast Fourier Transform (FFT) calculations on the ADC sample buffer. Once the FFT calculation concludes, the samples are passed to the second R5F-01 core. This core manages the creation of JSON formatting and the transmission of data via the MQTT protocol over the Single Pair Ethernet (SPE) connection. The ICSSG1 manages Ethernet transmission through the SPE port.

This iterative process is continually repeated for each ADC buffer. FFT calculations and raw ADC samples are efficiently conveyed over the Single Pair Ethernet to the gateway board.

ARM application

The application on the ARM processor is split across two ARM Cortex R5F cores for efficient processing and communication.

R5F Core 00:

- R5F Core 00 is responsible for initializing the MPU (Microprocessing Unit), which includes subsystem configuration settings. These configuration details are specified in the SYSCONFIG file.
- RF5 Core 00 handles tasks like pinmux configuration, device setup, PRU ADC firmware loading and initiation, and the provisioning of configuration data to the ADC board.
- RF5 Core 00 receives events from PRU-ICSSG0 and performs Fast Fourier Transformation (FFT).

R5F Core 01:

- R5F Core 01 receives events from Core 00.
- This core handles the conversion of processed data (raw ADC and FFT results) into a format for transmission. Specifically, the core performs JSON (JavaScript Object Notation) formatting of the data.
- JSON formatting is followed by data transmission over the Single Pair Ethernet (SPE) using the MQTT (Message Queuing Telemetry Transport) protocol.
- Core 01 makes sure that the data is packaged and transmitted effectively over the network, making the data accessible for visualization and analysis.
- The MAC (Media Access Control) layer operates on ICSSG1, facilitating the Ethernet communication.

In summary, the ARM application is structured to leverage the capabilities of two Cortex R5F cores efficiently. Core 00 handles initialization, configuration, and interaction with the ADC subsystem, while Core 01 is dedicated to data processing and communication. This leads to the efficient transmission of processed data over the network.

3 Summary

The provided information gives a clear overview of the capabilities and intended usage of the Sitara™ MCU+ and MPU processors in this application note:

- **Integrated Flexible ADC Interface**: Sitara™ MCU+ and MPU processors offer a programmable multi-channel SPI ADC interface via PRU-ICSS, which are versatile and adaptable in interfacing with external ADCs.
- **4-Channel ADC Sampling**: The application note demonstrates the ability to capture data from four ADC channels concurrently. This synchronous sampling allows for synchronized acquisition of data from multiple sources, crucial in various sensor-based applications.
- **Edge Processing and FFT**: Following data capture, the processors are programmed to execute edge processing tasks, which can include data preprocessing or other initial calculations. Furthermore, the processors are capable of performing Fast Fourier Transform (FFT) computations on the acquired data. FFT is instrumental in transforming data from the time domain to the frequency domain, enabling advanced signal analysis and pattern recognition.
- **Data Transmission via MQTT**: The application showcases the capability to transmit the processed data to a gateway or cloud-based service using the MQTT (Message Queuing Telemetry Transport) protocol. This allows for efficient and reliable data communication, often employed in IoT (Internet of Things) and sensor data monitoring applications.

In summary, this example illustrates how the Sitara™ MCU+ and MPU processors can be employed for advanced data acquisition, processing, and transmission tasks, making them appropriate for a wide range of applications requiring real-time data analysis and connectivity with external systems or the cloud.

4 References

- Texas Instruments, TIDA-010261: 10BASE-T1L single-pair Ethernet sensor with power over data lines reference design
- Texas Instruments, TIDA-010249: Four-channel synchronous vibration sensor interface reference design
- Texas Instruments, 1-channel PRU ADC example form AM243x MCU+ SDK

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265