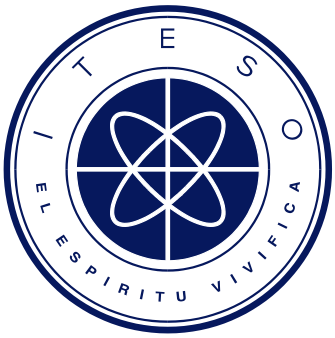
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

ESPECIALIDAD EN SISTEMAS EMBEBIDOS



PLATFORM UPGRADE OF AN IRIDIUM-BASED SATELLITE COMMUNICATION SYSTEM FOR ENVIRONMENTAL TELEMETRY

TOG Trabajo de obtención de grado para obtener el grado de:

Especialista en sistemas embebidos

Presenta(n)
Nombre del(os) autor(es): Gerardo Ramses Trejo Soria
Nombre de los co-directores: Luis Rizo Dominguez

San Pedro Tlaquepaque, Jalisco. agosto de 2026.

Platform Upgrade of an Iridium-Based Satellite Communication System for Environmental Telemetry

Gerardo Ramses Trejo Soria
Departamento de Electrónica, Sistemas e Informática
Instituto Tecnológico y de Estudios Superiores de Occidente
Tlaquepaque, Jalisco, México
gerardo.trejo@iteso.mx

Luis Rizo Dominguez
Departamento de Electrónica, Sistemas e Informática
Instituto Tecnológico y de Estudios Superiores de Occidente
Tlaquepaque, Jalisco, México
lrizo@iteso.mx

***Abstract*— Remote environmental telemetry payloads, such as the experimental module for the iterative design for satellite subsystems (EMIDSS-8), a collaborative suborbital research mission supported by NASA and Mexican academic institutions, face severe battery capacity constraints while requiring reliable data acquisition and satellite transmission. To address these limitations, this work details the design and implementation of a low-power, layered embedded software architecture targeting the NXP S32K312 microcontroller. The system employs a dual-mode power strategy that combines a software-based cooperative scheduler with a deep hardware standby sleep state, utilizing standby SRAM to maintain clock synchronization and state persistence across wake-up reset boundaries. The implemented architecture successfully acquired temperature, pressure, and humidity data via an I²C interface and managed local data logging, while executing periodic telemetry transmissions over the Iridium satellite constellation. This architecture provides a highly robust and energy-efficient firmware framework directly applicable to extended-duration suborbital missions and remote satellite nodes.**

***Keywords*— Satellite transmission, embedded systems, low power, scheduler, Iridium network.**

INTRODUCTION

Nanosatellites have emerged as cost-effective platforms for environmental data acquisition of the Earth's stratosphere, enabling the collection of variables such as temperature, humidity, and pressure. However, achieving reliable data transmission remains a persistent challenge due to stringent power, size, and operational constraints, which are largely dictated by the limited computational capabilities of onboard processors [1]. To navigate these limitations, embedded systems play a critical role in facilitating efficient processing, communication, and power management within these resource-constrained environments. Ultimately, overcoming these constraints requires a careful tradeoff between enhancing processing capabilities and minimizing power consumption.

The Experimental Module for the Iterative Design for Satellite Subsystems (EMIDSS) is a research initiative developed by the Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO), in collaboration with the *Universidad Nacional Autonoma de Mexico* (UNAM) and the *Instituto Politecnico Nacional* (IPN), with support of the National Aeronautics and Space Administration (NASA). EMIDSS has undergone seven prior iterations (EMIDSS 1-7). In EMDISS-5 an embedded system architecture was designed following AUTOSAR standards using onboard sensors and memory to collect data powered by batteries [2], however, data storage occurred locally and became accessible only after the balloon returned to Earth. The data required physical recovery before it could be read and processed. EMIDSS-7, integrated an Iridium 9603N satellite modem with an S32K144W, utilizing a FlexIO and an I²C instance, as well as an algorithm to improve the robustness and reliability of temperature, humidity and pressure sensor data [3]. The EMIDSS-8 mission migrates from the NXP S32K144W microcontroller to the more advanced S32K312 evaluation board [4], enabling enhanced processing capabilities and improved peripheral support. In addition, a dedicated I²C peripheral interface is implemented to stablish an instance between the Iridium modem and the CPU.

This paper is organized as follows: Section II introduces the system and software architecture design. Section III details the migration and software development processes. Section IV explains the implementation of the cooperative non-blocking scheduler. Section V examines the low-power modes of operation, while Section VI analyzes the experimental results. Finally, Section VII presents the conclusions and future work.

SYSTEM AND SOFTWARE ARCHITECTURE DESIGN

This section covers the system and software architecture re-design required for the technological migration to the NXP S32K312 platform [5]. This transition introduces critical hardware upgrades over the legacy S32K144W [6], including a core architectural shift from a 32-bit Arm Cortex-M4F (operating up to 80 MHz in normal RUN mode with a normalized performance of 1.25 DMIPS/MHz) [5] [6], to a high-performance Cortex-M7 core (running up to 240 MHz with 2.4 DMIPS/MHz). This frequency scalability introduces a more rigorous clocking topology managed by the Clock Generation Module (MC_CGM). Finally, communication capability is changed by upgrading to two native LPI²C, instances and four LPSPI channels, eliminating the need for CPU-intensive software

protocol emulation during telemetry logging and satellite transactions. The details of these changes are detailed below

System Architecture Design

To facilitate future scalability, the EMIDSS-8 platform adopts the NXP S32K312 microcontroller and its advanced peripheral suite. The specifics of this hardware and bare-metal software architecture are described below:

I²C1 driver: This new platform has enough peripheral drives available to use. Thus, the *I²C*1is now dedicated to full duplex communication between the platform core and the Iridium 9603N modem.

I²C0 driver: This driver is dedicated to request data acquired from the MS8607 sensor.

The SPI0 driver: The SPI0 will be connected to a memory that will store the data acquired during the mission

ARM cortex-M7 and Clock Frequency.

Clock Frequency: The S32K312 MCU provides increased frequency capabilities, enabling higher operating speeds. However, this introduces a more complex power consumption profile, necessitating the implementation of dedicated control logic to minimize energy use for the EMIDSS-8 mission.

The system architecture diagram is shown in "*Fig. 1*"

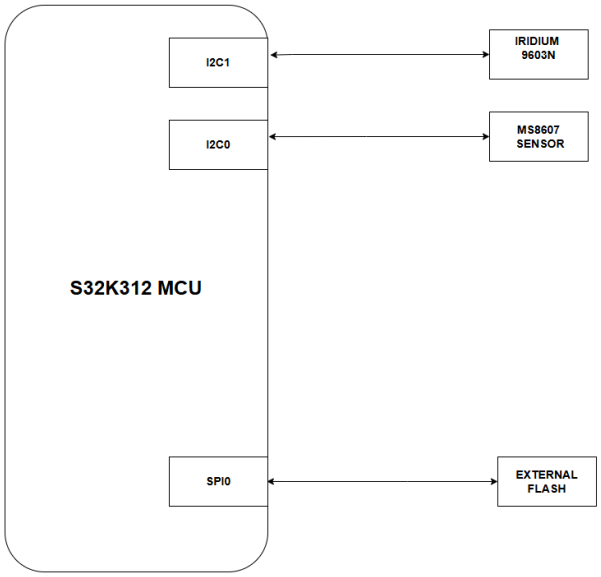


Fig. 1. EMIDSS-8 System Architecture Design.

SOFTWARE ARCHITECTURE

The EMIDSS-8 mission introduces specific platform and driver modifications, such as the integration of modern AUTOSAR-compliant Real-Time Drivers (RTD). These updates create an opportunity to re-design the software architecture and align it with mission requirements. While maintaining compliance with AUTOSAR standards, the new architecture incorporates a scheduler framework and a low-power mode of operation. The details of these changes are presented below.

Drivers: The S32K312 platform includes Real Time Drivers (RTD), compliant with modern AUTOSAR specifications. To manage the driver configuration such as pin routing and general configuration (as shown in "*Fig*. 2") and generate the necessary APIs for the application interface hardware. To simplify this configuration process, the NXP S32 Design Studio provides a dedicated initialization tool.

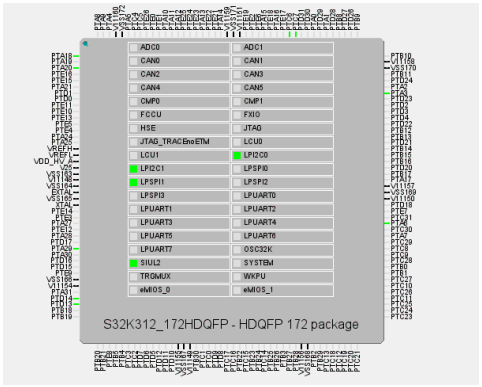


Fig. 2 Example of NXP's configuration tool

Software Architecture: The software architecture was designed following AUTOSAR standards to implement a layered development approach. The upper layer, known as the Application Software (ASW), contains the main application logic; this includes, for example, the tasks that are called every 1s, 1m, and 7m. Meanwhile, the bottom layer, known as the Basic Software (BSW), houses the drivers and APIs that expose hardware read and write functionalities to the ASW. "*Fig. 3*" shows the software configuration tree of EMIDSS-8.

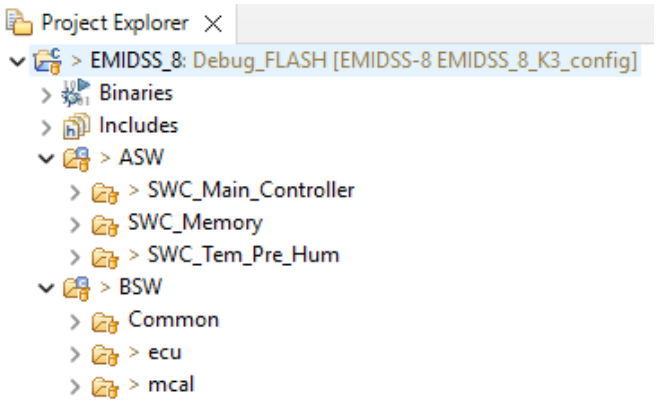


Fig. 3. Tree of Software Architecture AUTOSAR compliance.

Scheduler: A scheduler was introduced to manage the execution of data acquisition and transmission tasks. Operating on a one-millisecond time base (or time "slice"), the scheduler triggers the data acquisition task every minute and the data transmission task every seven minutes. Additionally, it features fault-handling capabilities that allow it to suspend or restart any task that fails.

LOW POWER MODES OF OPERATION

Power management in the S32K312 microcontroller family is built around a highly flexible scheme that balances processing performance with energy efficiency, allowing

seamless transitions between three primary system power operating modes [4] described below:

RUN Mode: The core operates at its maximum frequency. To limit active dynamic current consumption, dynamic clock gating is applied via the MC_CGM to disable clocks for unused peripherals, such as the LPI²C or LPSPI, when transmission is inactive

VERY LOW SPEED Mode: Implemented via software configuration, this mode divides the native 48 MHz Fast Internal RC (FIRC) oscillator by 16 using CGM divider settings to generate a 3 MHz system clock, while disabling all Phase-Locked Loops (PLLs). This state is utilized for essential CPU tasks that do not require high processing speeds, effectively reducing active current from approximately 50 mA to 5 mA.

STANDBY Mode: Serving as the deepest power-saving state, the CPU core, peripheral platform, flash memory, and PLLs are entirely power-gated. In this mode, system current consumption drops to the microampere range (approximately 15 to 35 µA).

MIGRATION AND DEVELOPMENT OF SOFTWARE ARCHITECTURE

Powered by a single 32-bit Arm Cortex-M7 core running at 120 MHz, engineered to operate reliably in harsh electrical environments, the platform offers an optimal balance of processing performance, excellent energy efficiency [5][6]. The MCU provides a broad range of low-power connectivity options; specifically, it integrates two dedicated I2C (LPI²C,) modules and four SPI (LPSPI) communication ports [5][6]. This section describes the process of migration from the previous implementation of the EMIDSS to this core platform using NXP's RTD configuration tool, and the software development to introduce a scheduler controller and low power operation.

System Configuration

The primary methodological shift involved transitioning from the direct register-level manipulation used in the Cortex-M4-based EMIDSS-7 to a standardized Hardware Abstraction Layer (HAL) provided by NXP's Real-Time Drivers (RTD) for the Cortex-M7-based S32K312. This new architecture was implemented utilizing the configuration tool provided within the S32 Design Studio IDE.

Clock configuration: NXP's S32 Design Studio includes a clock configuration utility that eliminates the need for manual CPU clock setup. This ensures the synchronized initialization of the system oscillator and peripheral clocks according to the project requirements. For this implementation, the main system clock was statically configured to reach the maximum hardware runtime frequency of 120 MHz. The clock configuration tree is shown in "*Fig. 4*"

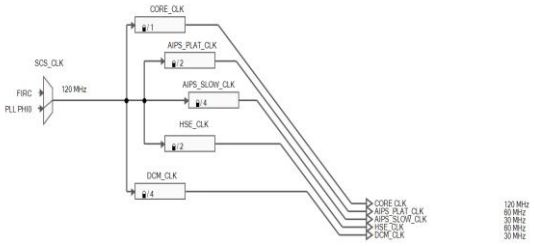


Fig. 4 Clocks diagrams.

Peripherals: As with the system clocks, the peripherals were configured using the S32 Design Studio configuration tool. Within the routing interface, the Peripheral column dictates the driver configuration, while the signal column specifies the pin multiplexing function. The routed pin/signal corresponds to the physical pin on the board, and the direction setting defines the hardware pin as either an input or output. Additionally, internal pull-up resistors were explicitly enabled for the I2C communication lines. The "*Fig. 5*" fully illustrates the complete peripheral configuration utilized for the project.

#	Peripheral	Signal	Arrow	Routed pin/signal
32	SIUL2	gpio, 29	<->	[32] PTA29
141	LPI2C1	lpi2c1_sda	<->	[141] PTC6
140	LPI2C1	lpi2c1_scl	<->	[140] PTC7
1	LPSPI1	lpspi1_sout	->	[1] PTA18
3	LPSPI1	lpspi1_sin	<-	[3] PTA20
123	LPSPI1	lpspi1_sck	<-	[123] PTA3
102	LPSPI1	lpspi1_pcs1	<->	[102] PTA6
41	LPI2C0	lpi2c0_sda	<->	[41] PTD13
40	LPI2C0	lpi2c0_scl	<->	[40] PTD14

Fig. 5 Peripherals pin configuration.

Software Driver Abstraction (RTD)

Real-Time Drivers (RTD) provided by NXP introduce modern, AUTOSAR-compliant driver architecture. Notably, the LP (Low Power) prefix indicates a complete redesign of the peripheral IP blocks to optimize power consumption. The transition and configuration from the legacy project to the new RTD framework are detailed below.

Dual I²C configuration: To interface with both the Iridium modem and the MS8607 sensor, two I²C peripherals were configured in master mode. These peripherals utilize an independent clock domain distinct from the main system clock. Specifically, I²C1 was configured with a prescaler divider of 16 to operate at a standard 100 kHz baud rate. Conversely, I²C0 was configured with a clock divider of 4, achieving a 400 kHz baud rate to support Fast-mode operation.

Synchronous SPI driver: An SPI driver utilizing a synchronous transmission model was implemented to support the command-response protocol of the NOR Flash memory. The implementation leverages a dedicated hardware abstraction layer (HAL) routine to autonomously manage Peripheral Chip Select (PCS) assertion and de-assertion, ensuring stability during high-speed data logging operations.

Sensor Driver Modernization: To integrate with the RTD-based MCAL, the sensor driver was modernized by eliminating legacy S32K1 SDK dependencies. Furthermore,

the sensor's timing requirements were synchronized with the newly implemented delay system, ensuring reliable temperature, pressure, and humidity data acquisition through Instance 0 of the LPI²C peripheral.

Timing and Task Orchestration

To ensure precise task execution and orchestration, a deterministic timing module was implemented by replacing the legacy clock initialization sequence with an AUTOSAR-compliant hardware timer abstraction. However, since this abstraction initializes the timer without natively enabling the core tick interrupt on the target architecture, a custom system tick configuration was required. By bypassing standard abstraction layers to write directly to the ARM Cortex-M7 core registers, this hardware-level workaround forces a strict 1ms interrupt, successfully establishing the deterministic execution required for the asynchronous delay module.

COOPERATIVE NON-BLOCKING SCHEDULER

To maintain system responsiveness and precise timing control during these lengthy transactions, of Iridium tasks and sensing acquisition, a cooperative non-blocking scheduler was implemented. By utilizing a time-triggered task-slicing approach instead of a fully preemptive RTOS kernel, the system achieves zero context-switching overhead, which is critical for conserving the processing bandwidth of the Cortex-M7 core under stratospheric flight constraints [7].

Delay Hook Mechanism

To maintain deterministic timing control, the 1ms interrupt serves as the fundamental time base. The scheduler calculates the total elapsed time by accumulating these hardware ticks, as described with the "Equation 1"

$$Elapsed\ Time\ = Current\ Ticks - Start\ Ticks$$

Equation 1:

Cooperative Scheduler Design

The scheduler is executed on ever tick polling cycle in the main loop of the code and during blocking wait loops, the "Fig. 7" is a flow diagram of the scheduler design

Non-Blocking Delay

During extended peripheral operations, such as the transmission of an I²C data frame, traditional execution relies on synchronous blocking delays. To prevent system stalls, a non-blocking execution model was implemented. This non-blocking allows the cooperative scheduler to remain active, continuously evaluating and executing other pending tasks during these waiting periods. "Fig. 6" is a flow diagram of the non-blocking wait logic.

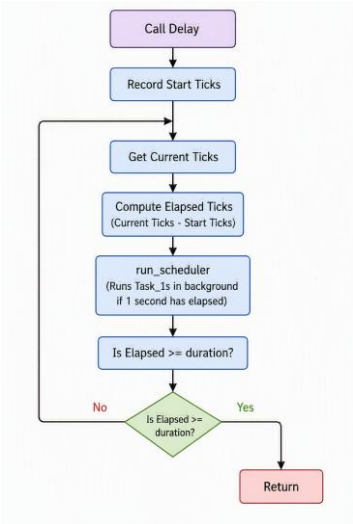


Fig. 6 non-blocking delay flow diagram.

Re-entrancy and Recursive Prevention

Invoking time delays within a task context by recursively executing the scheduler can lead to stack overflow conditions or the unintended re-triggering of active tasks. To resolve this vulnerability, a state flag mechanism was implemented. This logic enables the scheduler to explicitly track the execution state of each task, preventing the duplication or preemption of active operations. "Fig. 7" also includes the dependency in the design of these flags.

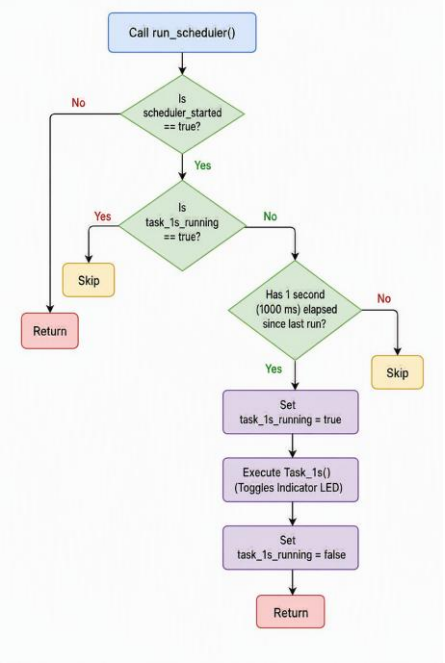


Fig. 7 Scheduler Design diagram.

LOW POWER MODES IMPLEMENTATION

The NXP S32K312 microcontroller architecture provides hardware power domains to achieve ultra-low leakage current during inactive states; this section describes in detail these hardware domains and the modes of operation utilized.

Power Architecture and Domains

The power architecture and its associated domains determine which peripherals and clocks remain active during each power mode of operation. Each domain is described in detail below:

The system's High-Voltage (HV): Specifically "VDD_HV_A" and "VDD_HV_B", are responsible for supplying the I/O pins, analog peripherals, and the internal Power Management Controller (PMC). Because the PMC relies on this supply, any significant voltage drop or loss of power on the VDD_HV_A rail immediately triggers a Low Voltage Reset (LVR), resulting in a destructive reset of the microcontroller.

The Core Power Domain (VDD_LV): Is a 1.1 V low-voltage rail that powers the ARM Cortex-M7 core, main SRAM arrays, flash memory controllers, and the underlying peripheral platform. To significantly minimize leakage current, this entire domain is power gated during STANDBY mode

The Standby Power Domain: Is a minimal, dedicated rail that remains energized when the system enters STANDBY mode. To facilitate system recovery and maintain timekeeping, this domain sustains critical low-power infrastructure, specifically supplying the Wake-up Unit (WKPU), the Real-Time Clock (RTC), a dedicated 32 KB block of Standby SRAM, and the 32 kHz Slow Internal RC (SIRC) oscillator.

Power State Selection

Although the microcontroller supports three distinct power states, only two operational modes were implemented to minimize architectural complexity. The configuration and utilization of each selected power state are detailed as follows.

RUN Mode: When waking up from sleep or in a first boot, the MCU needs to operate in full performance. It initializes necessary clocks, reads the sensors, stores the frames to memory and executes any needed satellite telemetry transmission at a full process speed. o limit active dynamic current consumption, dynamic clock gating is applied via the MC_CGM to disable clocks for unused peripherals, such as the LPI²C or LPSPI, when transmission is inactive.

STANDBY Mode: Once the tasks are completed (expected to take just a few milliseconds for sensor acquisition (every 1 minute), modem transmission (Every 7 minutes) and LED blinking (every 1 ms), the MCU immediately enters STANDBY mode, The CPU core, PLLs, Flash controllers, and standard RAM are powered off. Current consumption drops to the absolute minimum (~15mA - ~35µA). The MCU will remain in this state for 60 seconds, until RTC wakes it up and start the execution of the tasks. *Table 1* describes the clock core, system core and PLL state.

Table 1 Power modes comparison

Mode of Operation	CPU core State	System Clock	PLL State
RUN	Active	PLL (120 MHz)	Locked / On
STANDBY	Off (gated)	Off	Off

Low Power modes Logic implementation

Since standby mode completely power-gates the Cortex-M7 core, all volatile execution context, including CPU registers, the call stack, and general SRAM, are erased once the core enters standby. Consequently, exiting standby mode triggers a system reset sequence, forcing the microcontroller to fetch the base reset vector and re-execute the primary initialization routines. The EMIDSS-8 architecture handles this by checking the boot reason early in the main loop to differentiate between a cold power-on and a standby wake-up. "*Fig. 8*" presents a flow diagram for low power modes of operation logic.

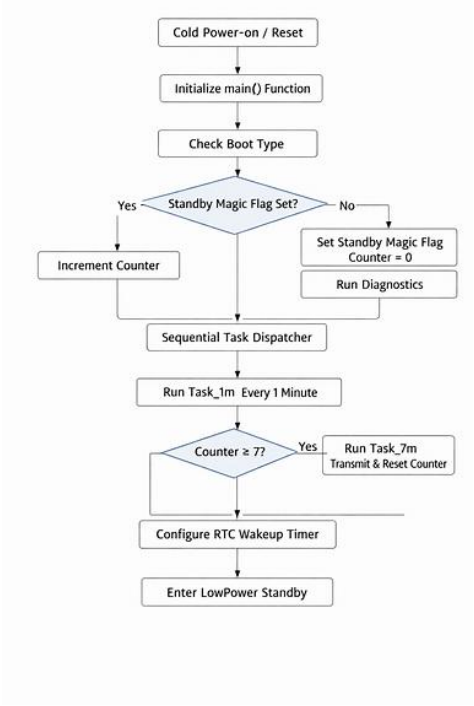


Fig. 8 Low power flow diagram.

Persistant state Storage

To prevent standard startup routines from zero-initializing critical variables during a system reset, this data is explicitly mapped to a dedicated non-volatile memory section (e.g. standby data) within the linker script [8]. Preserving this data in the Standby RAM block allows the MCU to successfully maintain its active minute counter, cold-boot status, validation flags, and system clock across power-down cycles. Furthermore, to prevent data corruption or buffer loss on the satellite modem when de-asserting its power lines during low-power transitions, the system executes an orderly shutdown protocol by issuing the AT*F command before power-gating, as recommended by satellite transceiver best practices [9].

```
__attribute__((section(".standby_data"))) uint32 wake_up_counter;
__attribute__((section(".standby_data"))) uint32 standby_magic_flag
__attribute__((section(".standby_data"))) uint8 transmission_counte
```

Fig. 9 Declaration of Persistent State Storage.

Experimental Results

Laboratory tests were conducted to system functionality with the Iridium modem, scheduler performance and power profiling.

System and Communication Validation

The S32K312 platform was successfully initialized with the main system clock operating stably at the target 120 MHz baseline. I²C peripherals were verified to operate reliably at a 100 kHz baud rate. Furthermore, the communication interfaces were fully validated, with the modem successfully connecting and transmitting data packets over Iridium satellite network. "*Fig. 10*" is a Logic Analyzer Capture of I²C Signal Transitions Between the MCU and the Iridium Modem.

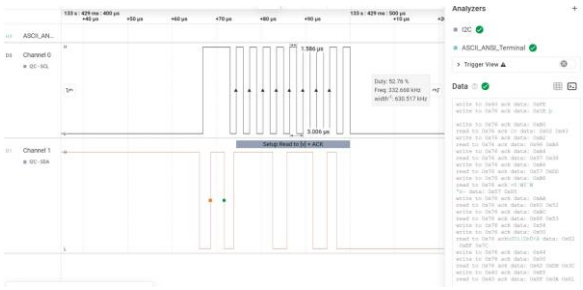


Fig 10 Communication with Iridium modem

Scheduler Performance

The custom designed non-blocking cooperative scheduler demonstrated reliable, deterministic behavior throughout the testing phase. Operating entirely without context switching, the scheduler accurately triggered the data acquisition task precisely every minute and the data transmission task every seven minutes. System execution and the non-blocking delay mechanisms were visually confirmed via a heartbeat LED that toggles during delay cycles.

Power Profiling

Power consumption was profiled across the system's operational states. In active RUN mode, the system draws an average current of approximately 140 mA. When the microcontroller enters STANDBY mode, the system current drops to 78 mA. It should be noted that this 78-mA standby floor is dictated by external hardware, specifically the Iridium modem, which remains fully powered to maintain its network connection rather than entering a low-power state. "*Fig. 11*" illustrates the power consumption of the EMIDSS-8 platform during the active state, drawing approximately 140 mA. Conversely, "*Fig. 12*" demonstrates the power consumption during the standby state, where the current draw drops to approximately 78 mA.

Fig 11 Power consumption during active periods

Fig 12 Power consumption during standby periods

Conclusions and Future Work

The migration to the S32K312 platform successfully modernized the EMIDSS-8 system architecture. By integrating a non-blocking cooperative scheduler and optimizing clock frequencies with Standby RAM retention, the design successfully balanced 120 MHz processing performance with minimal idle energy consumption. The validated hardware and reliable satellite communication links confirm the platform's robustness for advanced atmospheric data acquisition.

Future iterations will focus on further optimizing the overall power profile by implementing power-gating strategies for the external Iridium modem to reduce the 78mA standby floor. Additionally, rigorous thermal and vacuum testing are being considered to ensure qualification for extreme nanosatellite environments.

References

NXP Semiconductors, "S32K1 Arm Cortex-based Automotive MCUs," Brochure S32K1AUTOMCUBRA4, Rev. 8, 2021..

J. C. Aristizábal Aristizábal et al., "Design of an automotive embedded system for stratospheric environments," DESI - Trabajos de fin de especialidad en Sistemas Embebidos, Inst. Tecnol. Estud. Super. Occidente, Tlaquepaque, Mexico, 2024.

H. M. H. Vargas et al., "Design and Implementation of a Satellite Communication System using the Iridium Network for Environmental Data Transmission," Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO), Tlaquepaque, Mexico, Tech. Rep. EMIDSS-7-Version-5, 2025.

NXP Semiconductors, "S32K312MINI-EVB Evaluation Board for Automotive General Purpose," Product Information, 2024 [Online]. Available: https://www.nxp.com/part/S32K312MINI-EVB.

NXP Semiconductors, "S32K1 to S32K3 Migration Guidelines," Application Note AN13414, Rev. 0, Oct. 2021.

NXP Semiconductors, "S32K1xx Series Reference Manual," Document S32K1XXRM, Rev. 13, Apr. 2020.

M. Nahas and R. Bautista-Quintero, "Developing Scheduler Test Cases to Verify Scheduler Implementations in Time-Triggered Embedded Systems," International Journal of Embedded Systems and Applications (IJESA), vol. 6, no. 1/2, pp. 1-20, Jun. 2016

NXP Semiconductors, "S32K3 Memories Guide," Application Note AN13388, Rev. 1, Jul. 2022

Iridium Communications, "Iridium® Short Burst Data Service Best Practice Guide," Rev. 2.0, Nov. 2017

s