ESP32 Internet of Things System on Module Overview

This is a guide to the help you understand the how hardware is connected in the SoM, and how to implement these systems in your own design. Software examples are linked to help you get started.

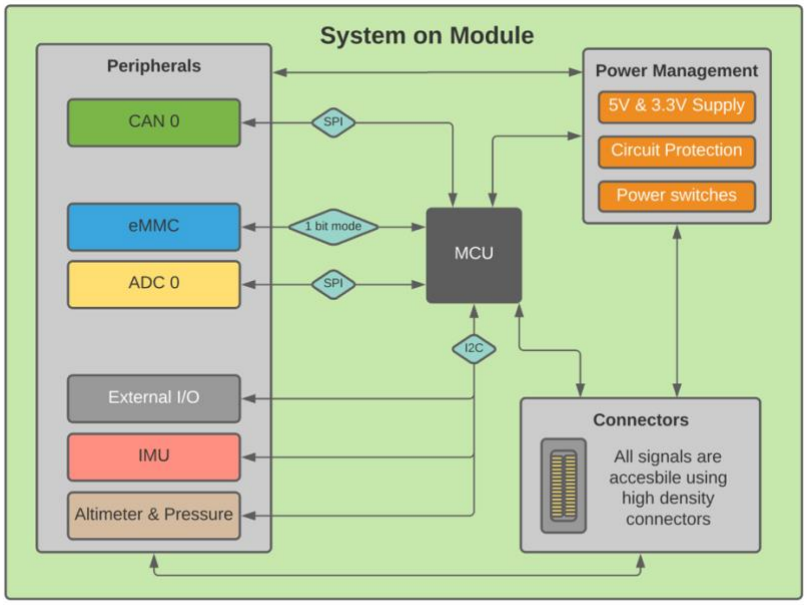

Figure 1: ESP32 SoM Overview

The *ESP32 System on Module* (SoM) is a Printed Circuit Board (PCB) that contains a variety of peripherals & power management circuitry that is commonly used in IoT & Industrial microcontroller applications.

The project was created to help decrease design time by abstracting circuit design, and by utilizing parts that are known to work with the MCU of choice. It also helps avoid wasting expensive components on a prototype.

To accompany this hardware, C++ libraries, compatible with the Arduino framework were written or ported for each peripheral on board, as well as for debugging, error logging, & process time tracking. These libraries can be found on GitHub.

All the sections in this document have a connection guide, a software example with links to websites that can help you learn and understand each system better than I could possibly explain it.

The specific pins and connections are in the examples too, and not shown here. This is only meant to be an overview of each system.

1. Microcontroller (μC)

The microcontroller of choice is the **ESP32** from *Espressif Systems*. An overview of the System on Chip is given below:

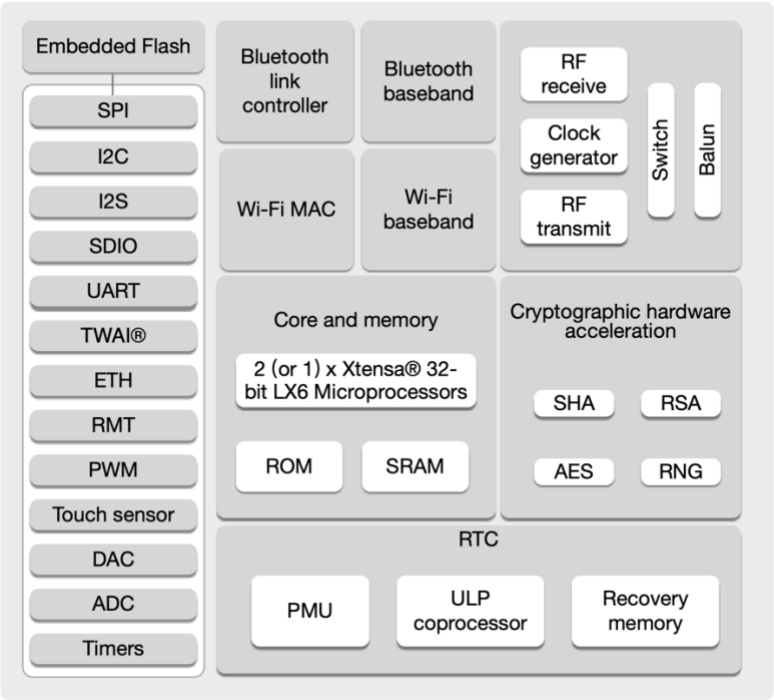

Figure 2: ESP32 System on Chip

- **Cost**
  - $5 chip (16MB Flash)
- **Performance**
  - 600 DMIPS
  - A wide variety of peripherals (80 MHz SPI, eMMC driver, 6 DMA Channels, Timers, etc.)
- **Part Availability**
  - Product was designed at the dawn of the Silicon shortage in 2021, and last we checked these were stocked at a variety of distributors.
- **Software support:**
  - Extensive software support is provided
  - Code can be reused across multiple chips
- **Software examples**
  - There's a great *number of libraries available in the Arduino community for digital signal processing, & a wide variety of peripherals*. ***In fact, this is how all the peripherals were chosen for this system.***

This microcontroller (MCU) has a great balance of computing power, lots of RAM, and a port of FreeRTOS for its dual core architecture.

The specific module used for REV1.0 of the SoM is the ESP32-WROOM-32E. This module has all the supporting circuitry to boot the chip (clocks, RF matching, etc.). By using a module rather than the SoC on the design, we can costs & unnecessary *headaches* by trying to get FFC approved. Read more about this here. Booting the MCU is also as simple as giving it a power source, and pulling the enable pin logic high, and *voila,* the chip will boot.

Flashing firmware to the MCU

Software development for the MCU is usually done in a host machine. Regardless of the development platform used, your code will be compiled into an executable binary for the Xtensa cores inside the chip.

This binary gets uploaded to the program memory via USB connection. The ESP does not have native USB connectivity, so a UART to USB bridge is used to accomplish this. For the SoM a USB2.0 compliant chip was used, so baud rates of up 5Mb/s can be achieved (limited by the ESP)

When you upload your code, the serial programmer will toggle the *DTR & RTS* pins of the UART bridge. These pins are connected to GPIO0 & the ESP enable pin respectively. By toggling these pins in a specific pattern, the MCU will be put into download mode, and the binary will be uploaded to the flash memory of the device.

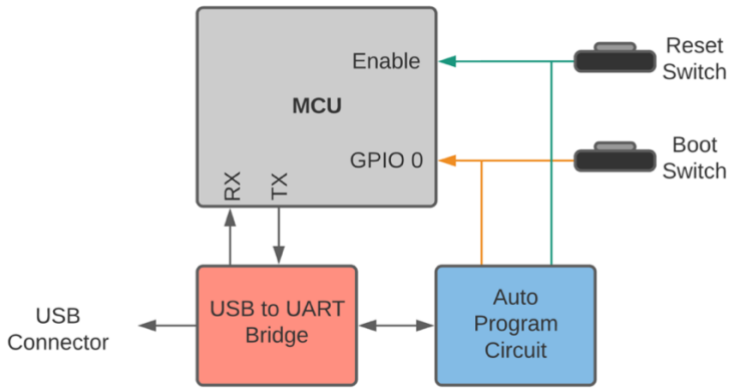

Figure 3: MCU Auto Program Circuit

Similarly, there are two buttons on the top layer of the board labeled **EN** (resets the MCU) **& Boot** (forces the MCU into program mode). These switches can be taken out of BOM for production and are only on board for debugging purposes.

General Purpose I/O

All the pins from the MCU are routed out into the board-to-board connectors. It's recommended you only use certain pins as intended (such as bus interfaces like I2C & SPI), since the **peripherals on the module are connected through these buses**.

The biggest downside of the MCU of choice is the amount of GPIO available after all the on-board peripherals are connected. Only 5 pins from the MCU are available for free use. These pins have internal pull-down or pull-up resistor options, and they can all be used as inputs. However, only 3 of these can be used as outputs.

To decrease this limitation, an **I/O expansion** (IC) has been added to the module. This I/O is connected through a dedicated I2C bus, and the minimum input **interrupt latency** is around **500µS**. These pins can also be used as **outputs**, with a latency of **800µs**.

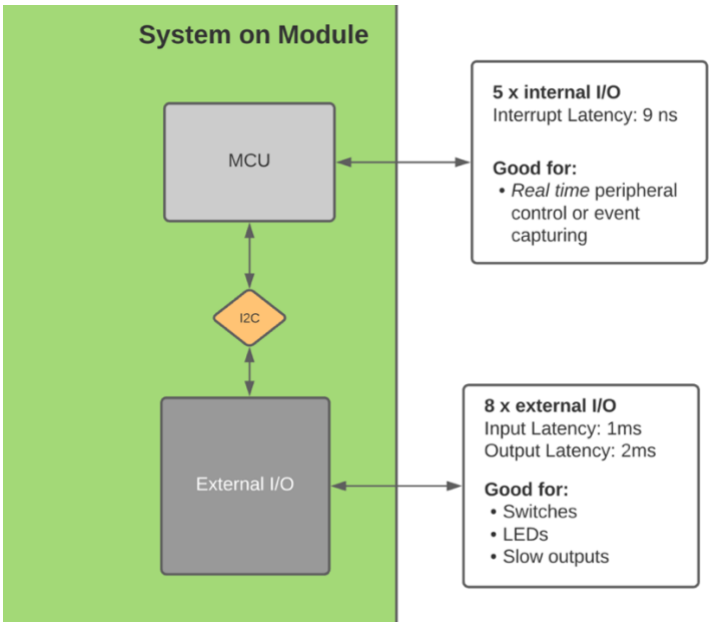

Figure 4: Available User I/O

Although some latency is introduced in the system if you use these pins, the I/O expansion offers features such as dynamic switch debouncing, & PWM control, that could reduce CPU overhead if you plan to use these in your application.

Software Support

- SystemOnChip example: Enables easy access to any of the systems inside the SoC (SPI, I2C, UART, Timers, etc.)
- IO Expansion example: Port of Sparkfun's SX1509 driver. All methods can be accessed in the same manner as original library.

2. Power Management

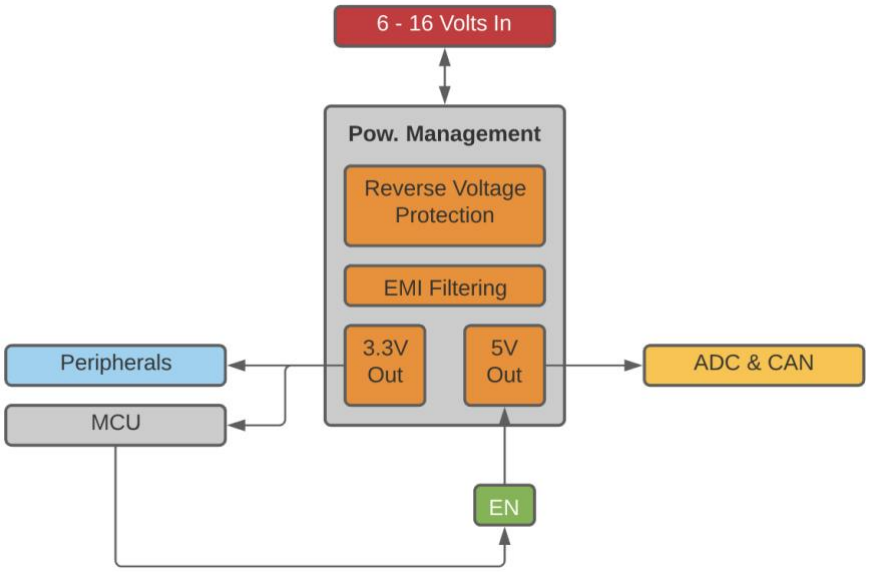

Figure 5: Power System Overview

There are three major components of the power system:

2.1 Circuit Protection & EMI filtering

The main voltage input has reverse polarity protection, as well as an EMI filter based on a reference design by *Texas Instruments*. If this EMI filter does not meet the requirements of your application, you can learn how to design one here.

2.2 Power Supplies

The board was designed to have an input voltage of at least 6V. This is because the ADC and the CAN transceiver require 5V to operate. All other hardware on board is powered with the second buck converter at 3.3V.

To maintain power efficiency, buck-converters are used. The 5V supply is OFF by default and can be enabled through software. The **3.3V supply is always on** if the SoM is enabled. The buck-converter used has a very low power consumption when enabled (appx. 47uA).

Important note: *Although the board is designed to work off a LiPO battery, the SoM will still consume some current even if all hardware is turned off & the processor is in sleep mode. This current however, is not more than 1mA. Considering how much current the device consumes when it's ON (appx. 200mA if all systems are on & Wi-Fi/BLE is transmitting), it's a negligible amount and designing for any lower power than this when switched OFF, is simply* ***not worth the effort****.*

2.3 Power Switches

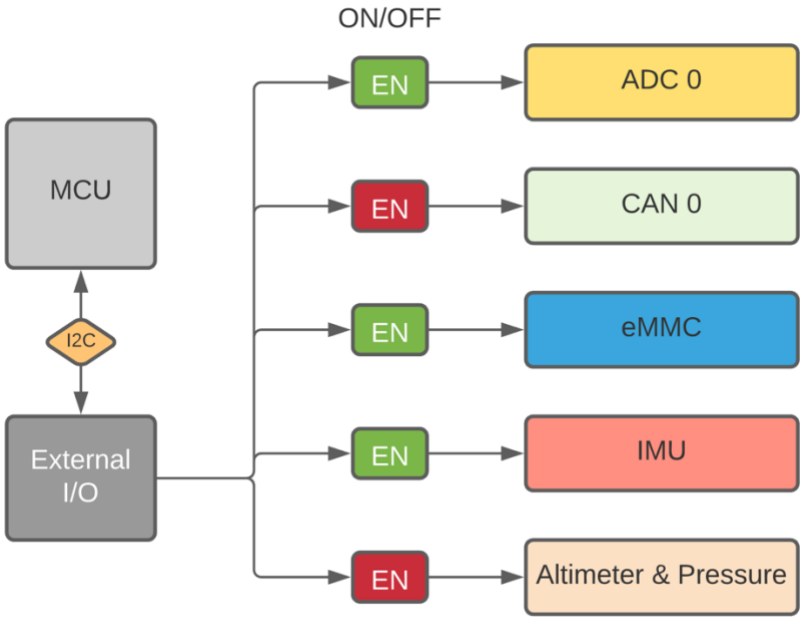

Figure 6: Power Switches

Every peripheral, **excluding the I/O expansion**, has a simple power-switch that enable the power supply to the peripheral. These EN pins are ON by default and can be disabled through software. These switches are implemented using low on-resistance P-Channel MOSFETS, like this one.

Software Support

- Not needed. Toggle using normal GPIO function calls.

3. Non-Volatile Storage

The SoM has a micro-SD card connector on the bottom layer. The ESP has a MMC driver, which can be implemented to drive a multimedia card in 1-bit mode & 4-bit mode.

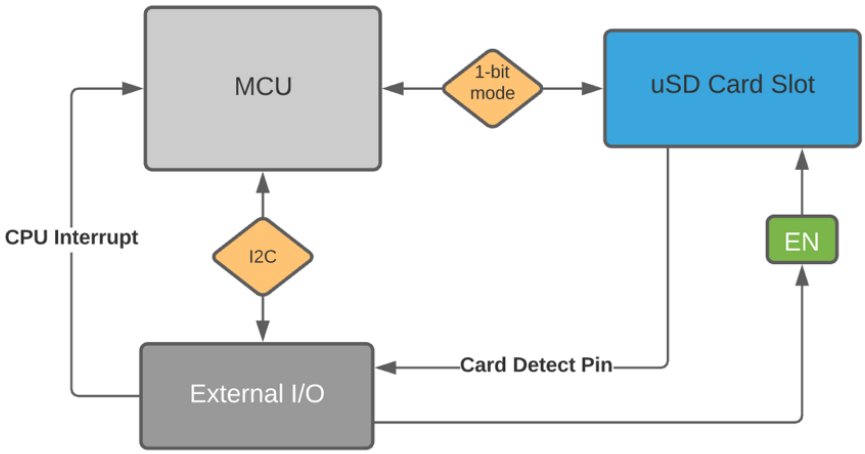

Figure 7: eMMC System Overview

In the SoM, the card is connected using 1-bit mode, as this was perfect balance between I/O used & read/write speeds. Additionally, the card-detect pin is connected to the GPIO expansion, which helps with switch debounce, and interrupts the CPU only 1 time.

The read/write speeds for the different connection modes available are shown below.

Connection Mode	Write-speed	Read speed	I/O needed
SPI	268,040 bytes/s	372,495 bytes/s	4
1-bit	577,409 bytes/s	1,205,259 bytes/s	3
4-bit	624,524 bytes/s	1,466,539 bytes/s	6

The read/write speeds are also dependent on the SD card of choice. So, finding a fast SD card is also important is the goal is to read/write at high rates.

Software support

- EMMC example: This library supports methods for saving to .csv & .json files. It also handles card removal & initialization on card detected.

4. Internal Motion Unit & Environmental Sensor

The internal motion unit on the SoM is the MPU9250 from *InvenSense, featuring the following:*

- High-Precision 9-axis Gyroscope (XY 0.05 degrees Accuracy)
- Accelerometer with programmable ranges from +/2G up to +/16G
- 3-axis monolithic hall-effect magnetic sensor

This allows the SoM to log pitch, roll, yaw, acceleration data & gyroscope data, using sensor fusion

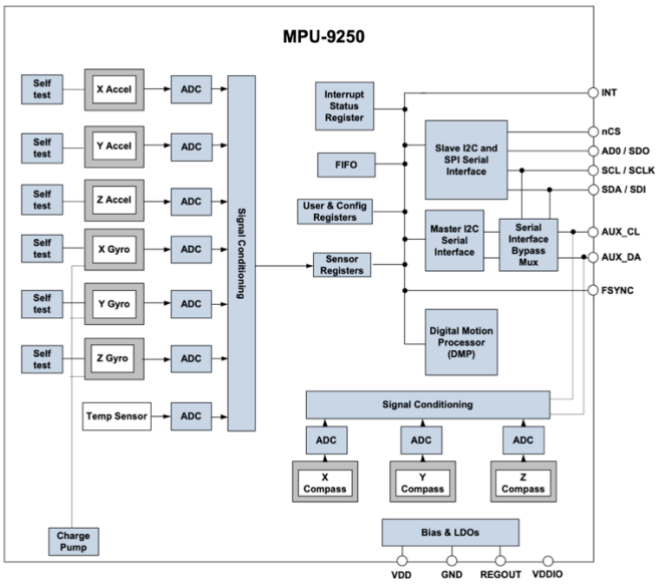

Figure 8: Internal Motion Unit MPU9250

Along with this IMU, the SoM has the environmental sensor BME688 from *Bosch*. This sensor contains ***temperature, humidity, barometric pressure and VOC gas sensing*** capabilities. It also shares the same I2C bus than the IMU and can also be enabled/disabled through software.

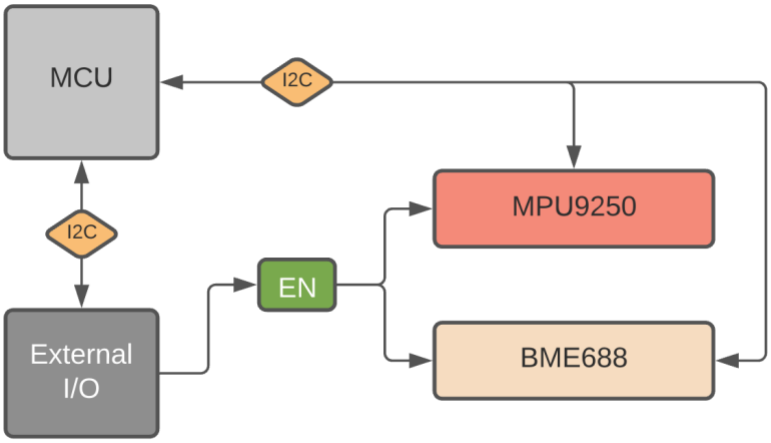

Figure 9: IMU System Overview

Software support

- Internal Motion Unit example

5. Analog to Digital Converter

The Analog to Digital Converter in the ESP32 has very low-resolution, and it is rather difficult to implement any sort of temperature compensation curve, given the non-linear behavior of the ADC.

Regardless, most MCUs simply don't offer high-quality ADCs due to the silicon substrates digital circuits are manufactured on.

To solve this problem, an external Successive Approximation ADC was chosen. Specifically, the **AD7689** was chosen because it offers a high-speed SPI interface, allowing to sample the ADC up to 150Ksps. If you want to read more about this, read this article

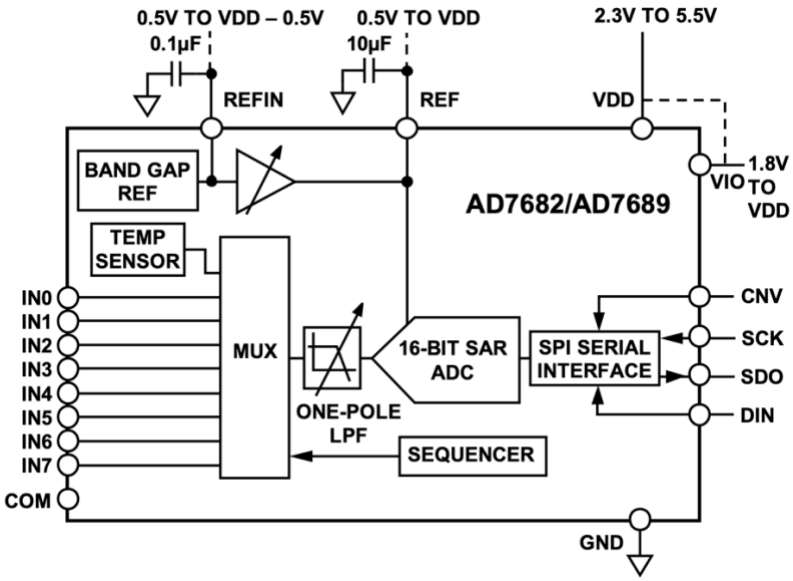

Figure 10: ADC AD7689 from Analog Devices

Some of the features of this chip are

- 8 – Inputs, 16-bit resolution
- Internal temperature sensor used for drift compensation.
- Option to use external, more-stable voltage reference for improved accuracy
- Fast communication interface allows high sampling rates with no missing codes

The inputs of the ADC are 0V to 4.096V. To protect these inputs, clamping diodes are placed at all the inputs, as well as a small RC filter recommended by the manufacturer to help reduce noise in single ended readings.

The ADC is powered by an LDO that drops the input voltage from 5V to 4.75V. This helps reduce any noise introduced in the supply by the buck-converter. It also helps enable-disable the ADC through software.

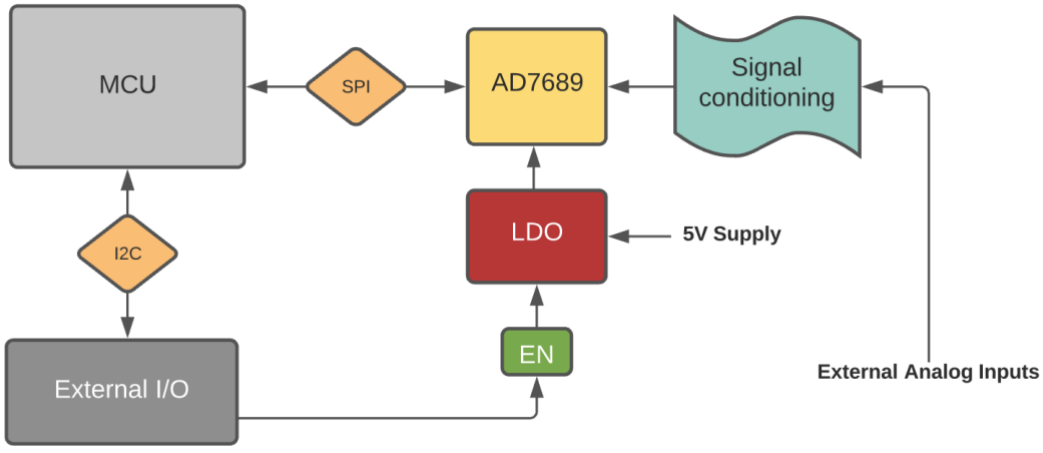

Figure 11: ADC System Overview

Since most applications require voltage measurements in higher voltage ranges than the maximum input voltage of the ADC, external signal conditioning is required to use this ADC. This can usually be done using inexpensive Op Amps & resistor dividers.

A realistic throughput is around 5KHz per channel, after filtering & data averaging (every 10 samples). Although it could be pushed to higher sample rates (up to 250KS/s in 1 channel), it is very unlikely the processor can save or transmit this many samples.

Software Support

- AD7689 example: This library is a port of the original work found in this repository. The major changes were to the SPI transaction method.

6. CAN FD Controller

A dedicated CAN 2.0B and CAN FD controller **MCP2518FD** adds features like hardware filtering of given CAN messages, as well as selectable speeds & termination resistor.

The CAN protocol is widely used in the automotive and industrial applications, and this IC offers very useful features like a FIFO buffer, event and filter interrupts that off-load a lot of work from the CPU.

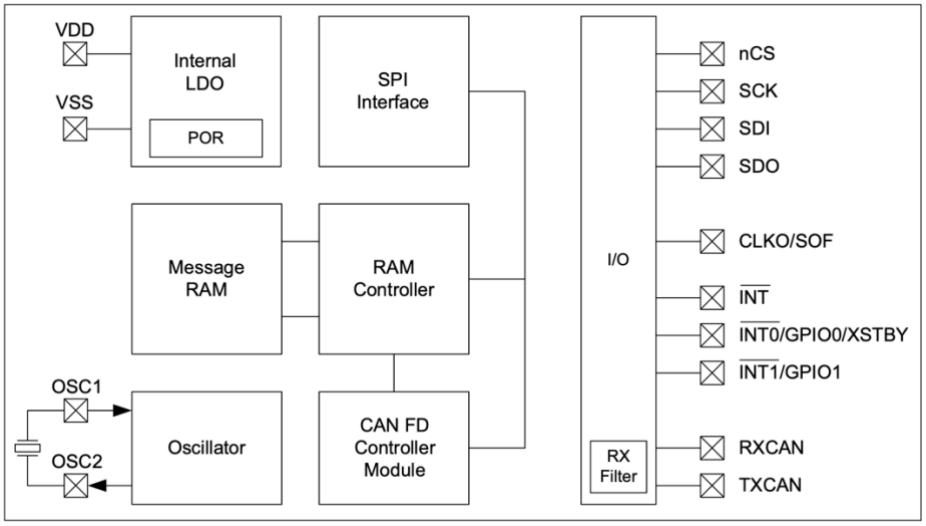

Figure 12: MCP2517FD CAN Controller

A couple of features can also be added to the CAN system.

6.1 Selectable termination: CAN uses a differential pair, which is ended with 120 Ohm resistor at the end nodes, and every other node in the middle does not need such a resistor. The SoM implements a design based off this reference design from Texas Instruments.

6.2 Silent mode: This is useful in the following cases:

- Disables transmitter in redundant systems
- Prevents a faulty CAN controller form disrupting all network communications

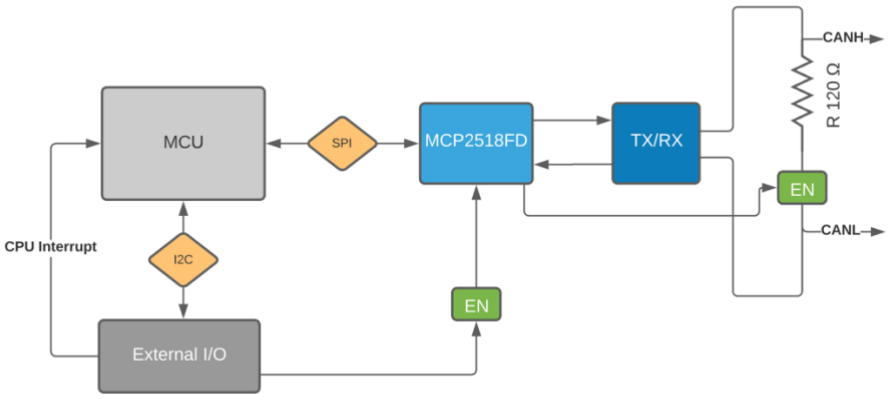

Figure 13: CAN System Overview

Software Support

- **ACAN2517FD Library:** This is a **phenomenal** library to control this chip. The author also took the time to write detailed documentation to help you write your own code and included plenty of examples. The port of this libraries in the *Utilities Libraries* is just added functionality for the selectable termination resistor.