FRONTGRADE

APPLICATION NOTE

UT32M0R500

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

12/2/2022
Version #: 1.0.0

Table 1: Cross Reference of Applicable Products

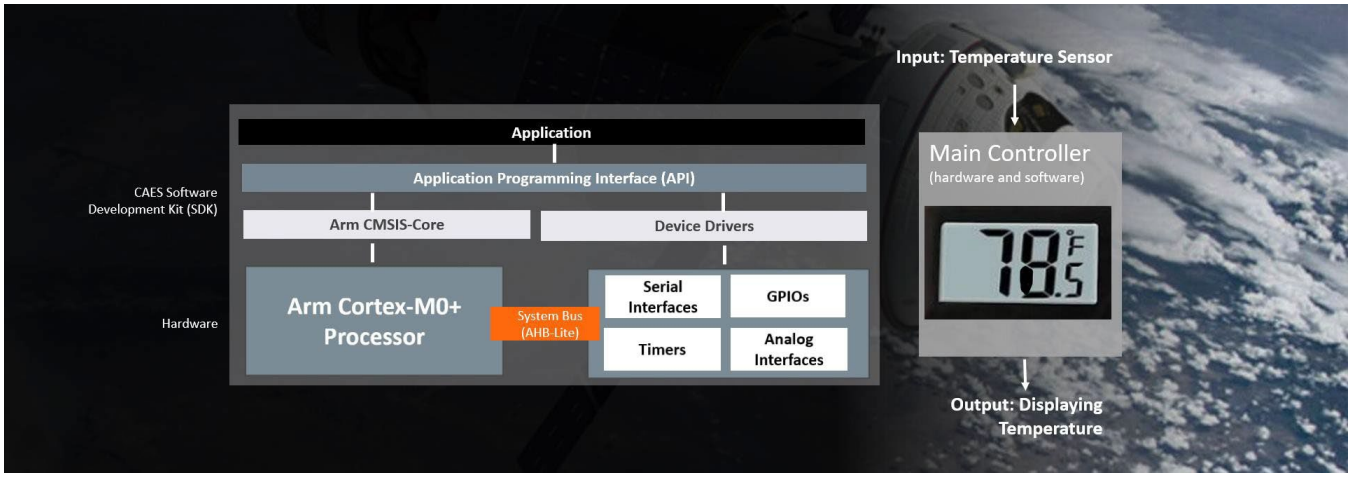
Product Name	Manufacturer Part Number	Smd #	Device Type	Internal Pic Number
Arm Cortex M0+	UT32M0R500	5962-17212		QS30

Overview

This app note gives an introduction to embedded systems, then goes over the processor architecture, the different microcontroller peripherals, software development kit or **SDK**, and finally an application.

The picture shows an embedded system with an input, an output and a main controller. The input reads a signal from a temperature sensor, the output writes a value to the LCD displaying the temperature in degrees Fahrenheit, and the main controller performs the specific task with hardware and software.

The picture on the left expands the UT32M0R500 microcontroller into hardware and software. The hardware shows the processor and different peripherals. The software starts with the ARM CMSIS core, which are drivers for the processor core and to the right of it are drivers for the different peripherals; on top of the drivers, the picture shows the application programming interface or API, which has function calls for the different device drivers; and finally, the program application puts everything together.



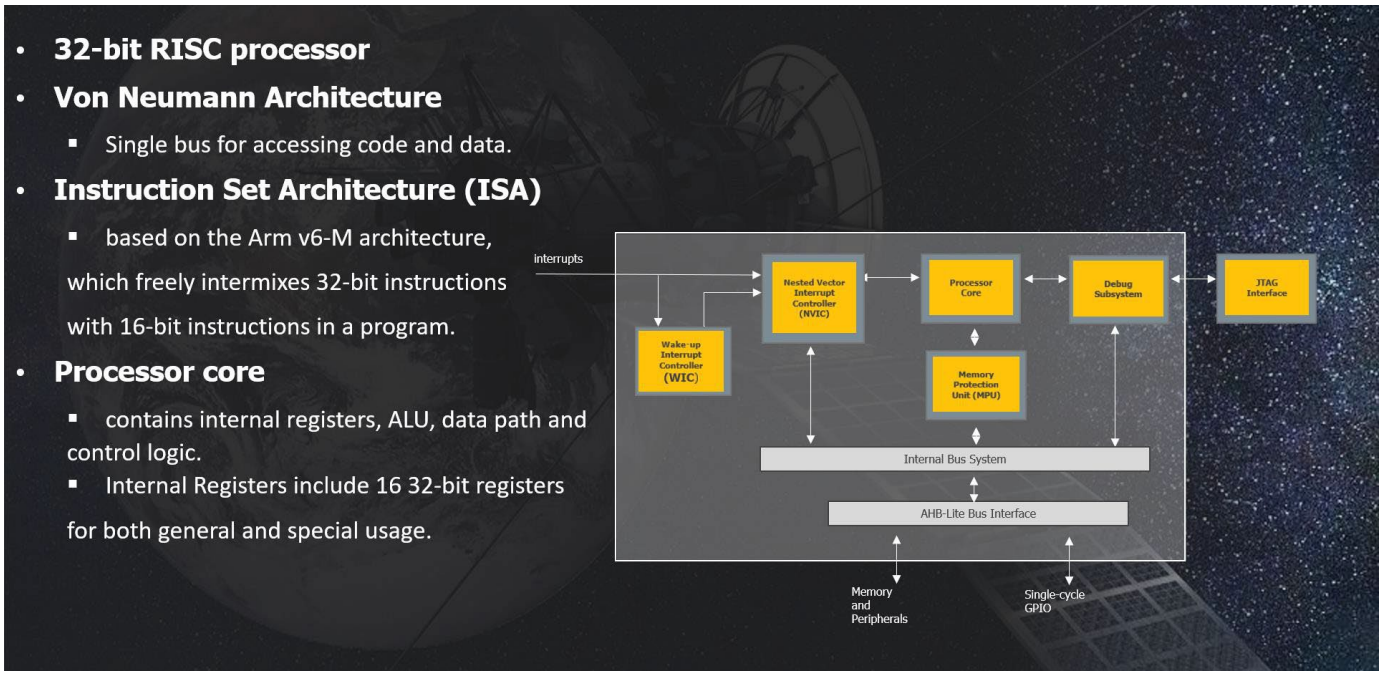
Cortex M0+ Processor Architecture

Overview

The ARM Cortex-M0+ is a 32-bit RISC processor, with Von Neumann architecture, which means single bus for accessing code and data.

The instruction set architecture or ISA is based on the ARMv6-M architecture, which freely intermixes 32 and 16-bit instructions in a program.

The processor core contains internal registers, ALU, data path and control logic. The internal registers include 16 32-bit registers for both general and special purposes.



The processor is a two-stage pipeline: execute and fetch and can fetch up to two 16-bit instructions in one transfer.

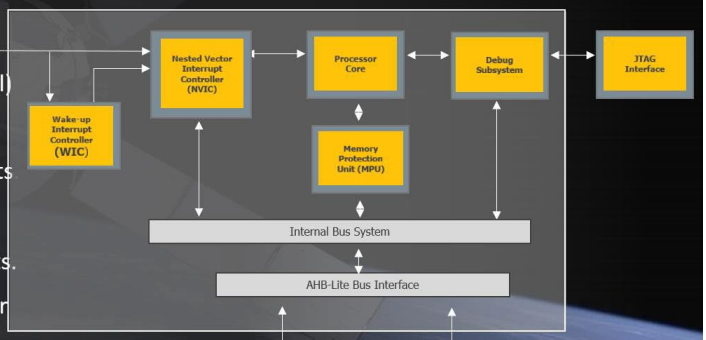
The nested vector interrupt or **NVIC** has one non-maskable interrupt or **NMI** and up to 32 physical interrupts with four priority levels, the lower the number, the higher the priority level. The NVIC automatically handles nested interrupts. The processor supports both level and edge-triggered interrupts.

The wakeup interrupt controller or **WIC** enters sleep mode by shutting most of the components. When an IRQ event is detected, the WIC informs the power management unit to power up the system. It uses WFI (wait for interrupt), WFE (wait for event) instructions and sleep on exit.

- **Two-Stage Pipeline**
  - Fetch and Execute
  - Up to two 16-bit instructions can be fetched in

  One transfer.
- **Nested Vector Interrupt (NVIC)**
  - One NMI and up to 32 physical interrupts
  - 4 interrupt priority levels (lower # higher priority level)
  - automatically handles nested interrupts
  - The processor supports both level and pulse interrupts
- **Wakeup Interrupt Controller (WIC)**
  - Enter sleep mode by shutting most of the components.
  - When IRQ event detected, the WIC informs the power management unit to power up the system.
  - WFI (wait for interrupt) and WFE (wait for event)

  instructions and sleep on exit.



The bus interconnect is a 32-bit AMBA-3 AHB-lite interface for integrating memory and peripherals.

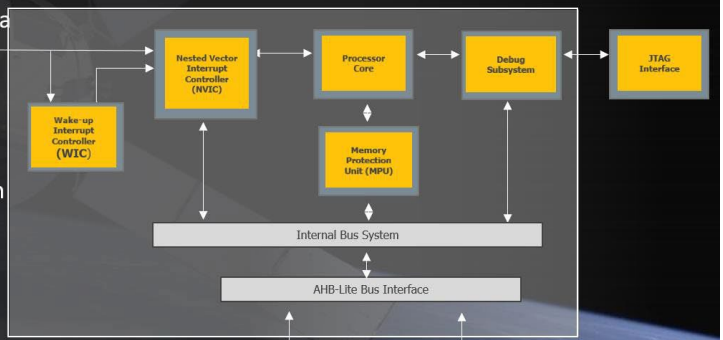
The debug subsystem handles debug control, program breakpoints, and data watchpoints. It has a JTAG port which uses the Keil ARM ULINK2 Debug Adapter for programming and debugging applications.

The processor has a memory protection unit or **MPU** with 8 regions, subregions and a background region.

Finally, the processor has the micro trace buffer or **MTB** for tracing instructions.

- **Bus Interconnect**
  - 32-bit AMBA-3 AHB-lite system interface

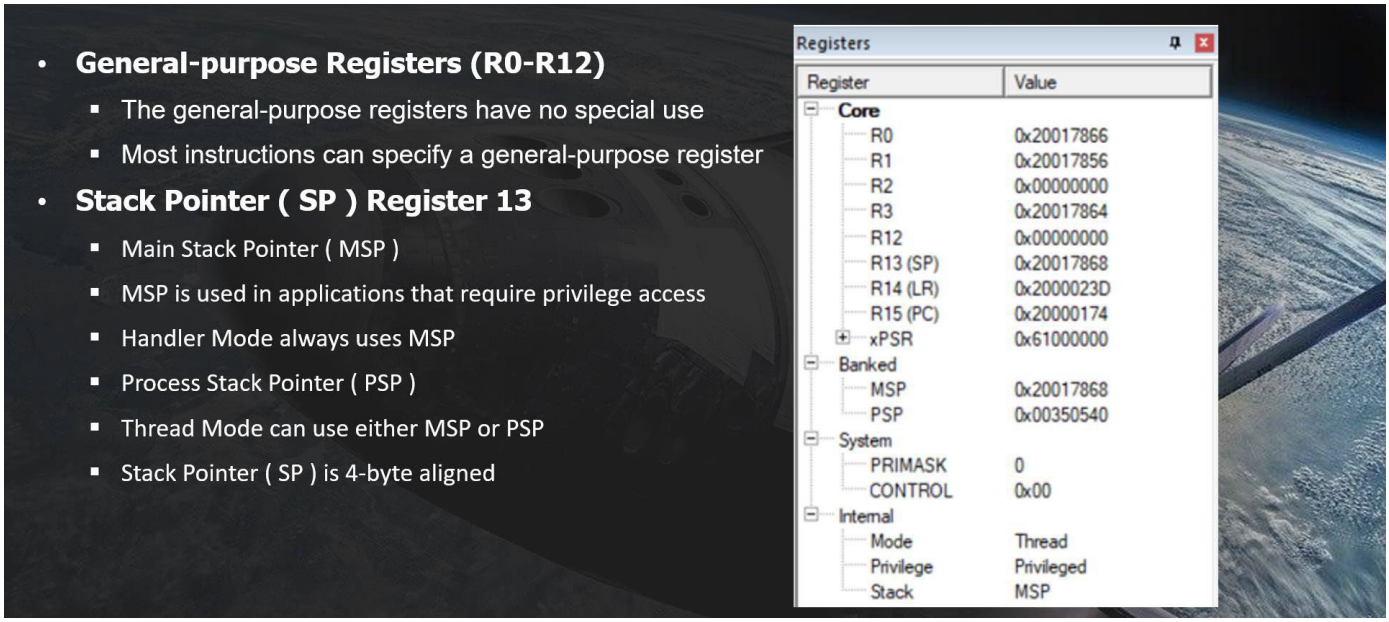
  for integrating memory and peripherals
- **Debug Subsystem**
  - handles debug control, program breakpoints, and data watchpoints
  - JTAG port uses the Keil ARM ULINK2 Debug Adapter
- **Memory Protection Unit (MPU)**
  - 8 region MPU with subregions and background region
- **Micro Trace Buffer (MTB)**
  - Micro trace buffer for tracing instructions.



Registers

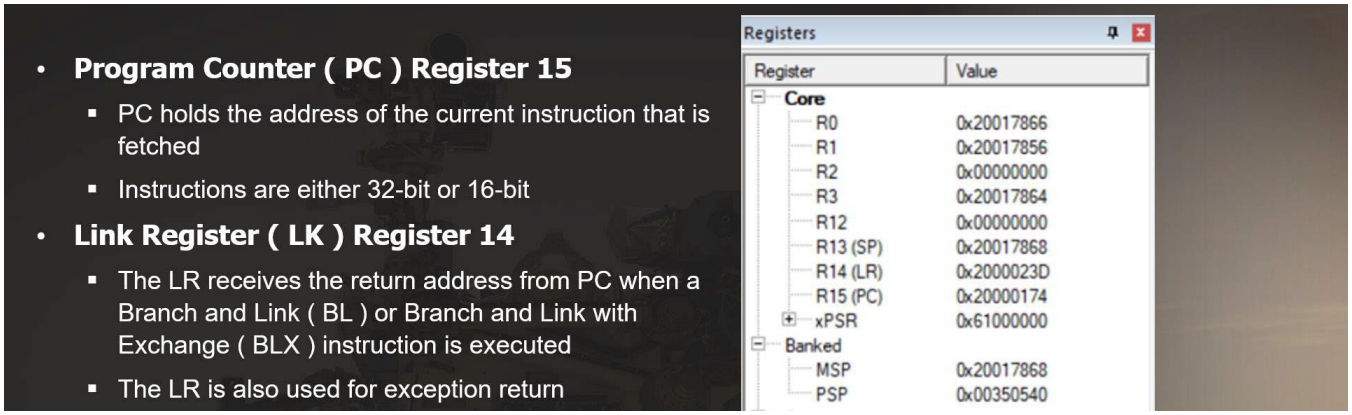
The internal registers are the workhorse of the processor. The processor has 13 general purpose registers **(R0-R12)** plus special registers. Most instructions can specify a general-purpose register.

Register 13 is the stack pointer or **SP.** The SP is 4-byte align. Main stack pointer or **MSP** is for applications that require privilege access while process stack pointer or **PSP** is not.

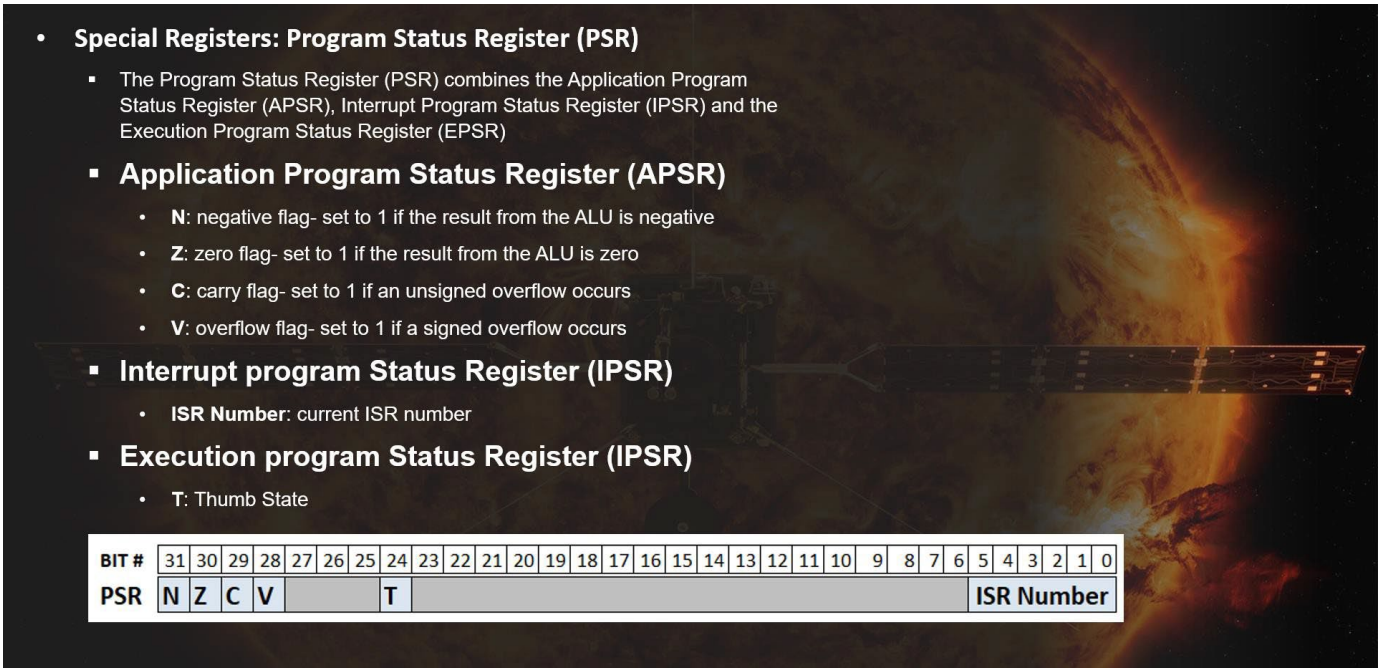


Register 15 is the program counter or **PC.** PC holds the address of the current instruction and instructions can be either 32 or 16-bit.

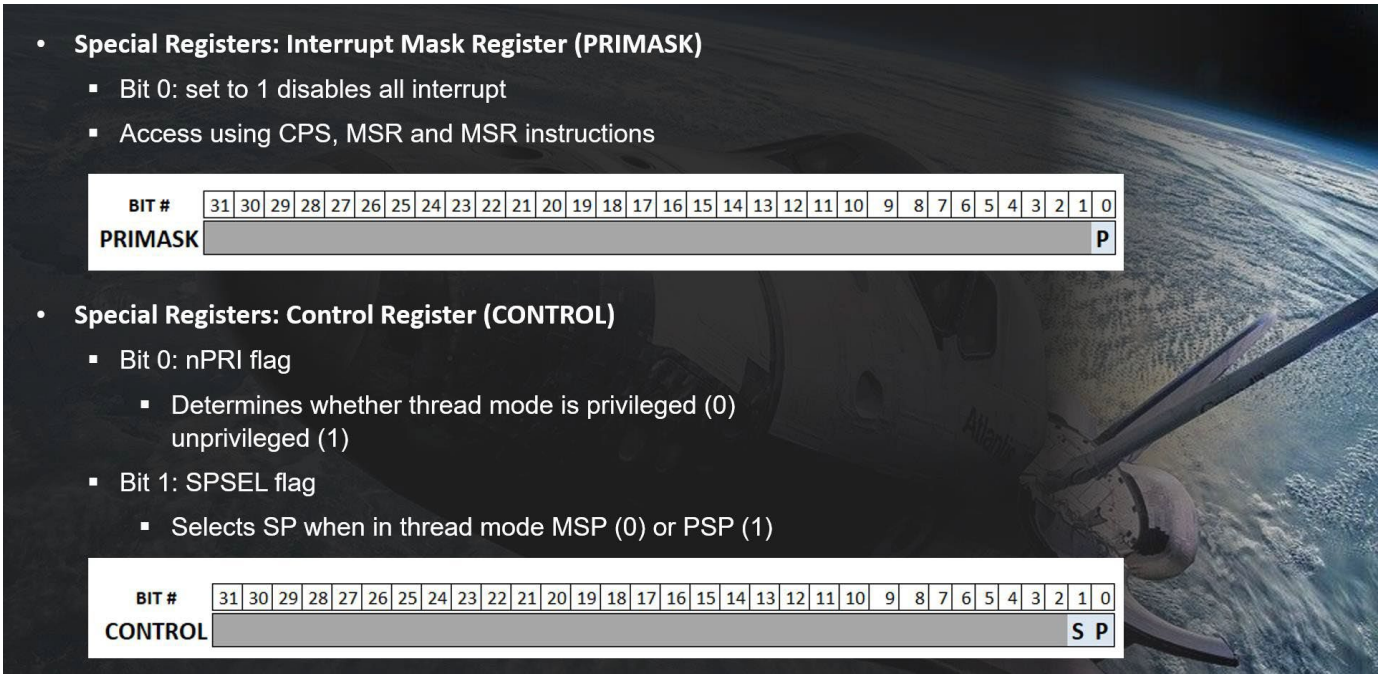
Register 14 is the link register or **LR.** The LR receives the return address from the **PC** when a Branch and Link **(BL)** or Branch and Link with Exchange **(BLX)** instruction is executed.



Program status register or **PSR** combines the application, interrupt and execution registers into one. The application register has the flags **n, z, c** and **v;** the interrupt register has the **ISR** number; and the execution register is always in thumb state for the ARM Cortex M0+ processor.

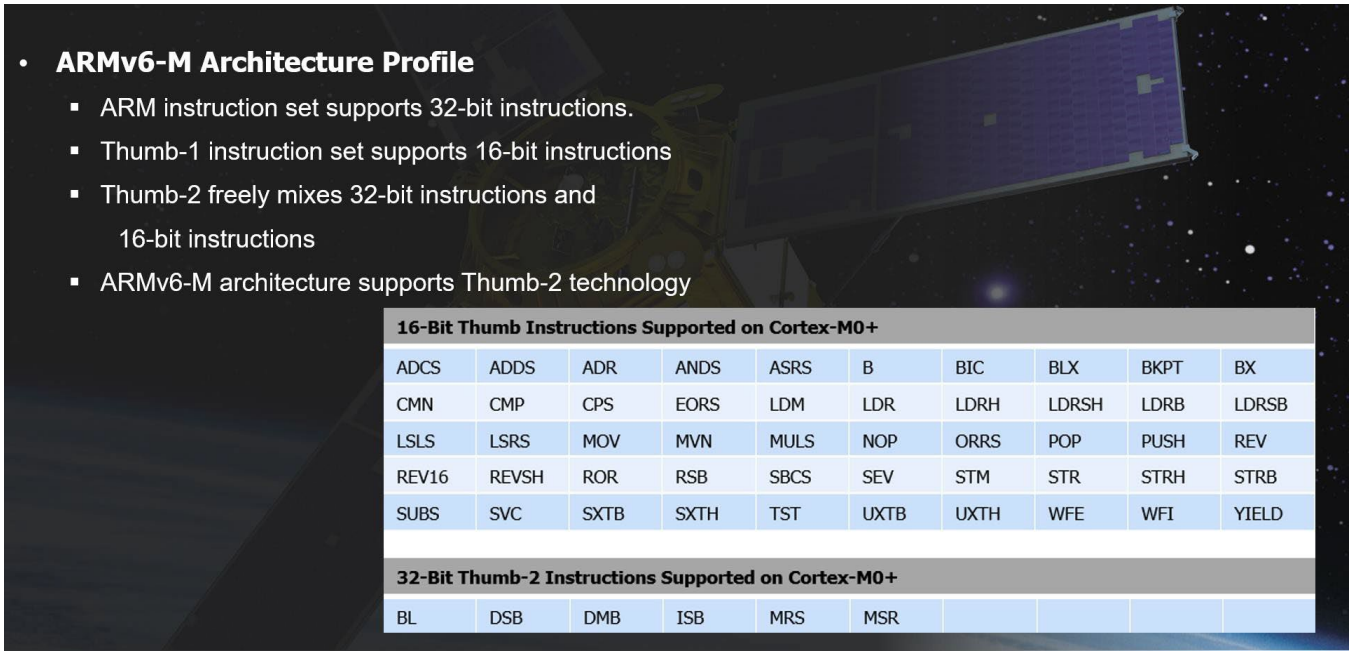


The **PRIMASK** register either enables or disables all interrupts. The **CONTROL** register selects the SP and determines whether the SP is privilege or not.



Armv6-M Architecture

The **instruction set architecture** or **ISA** is based on the ARMv6-M architecture profile. The picture shows that most instructions are 16-bit instructions while only a few instructions are 32-bit instructions. Thumb-2 freely mixes 32 and 16-bit instructions, and the ARMv6-M supports Thumb-2 technology.

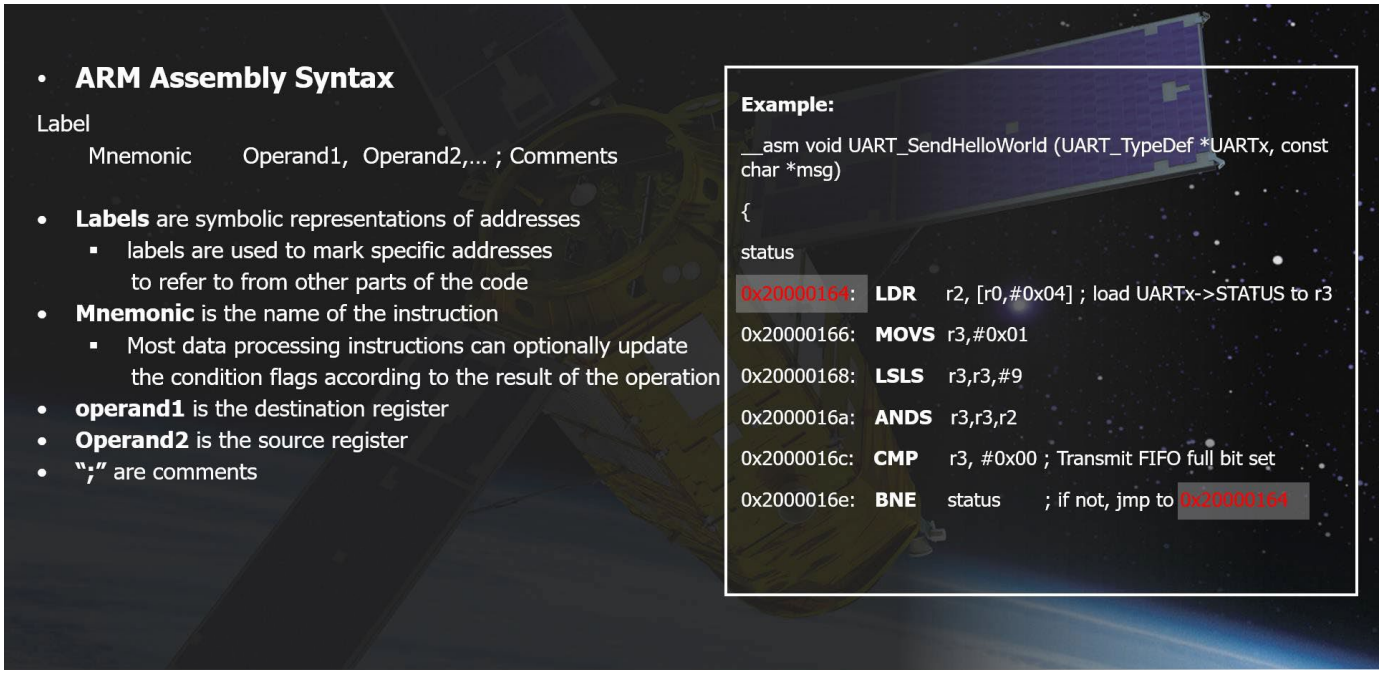


16-Bit Thumb Instructions Supported on Cortex-M0+									
ADCS	ADDS	ADR	ANDS	ASRS	B	BIC	BLX	BKPT	BX
CMN	CMP	CPS	EORS	LDM	LDR	LDRH	LDRSH	LDRB	LDRSB
LSLS	LSRS	MOV	MVN	MULS	NOP	ORRS	POP	PUSH	REV
REV16	REVSH	ROR	RSB	SBCS	SEV	STM	STR	STRH	STRB
SUBS	SVC	SXTB	SXTH	TST	UXTB	UXTH	WFE	WFI	YIELD

32-Bit Thumb-2 Instructions Supported on Cortex-M0+									
BL	DSB	DMB	ISB	MRS	MSR				

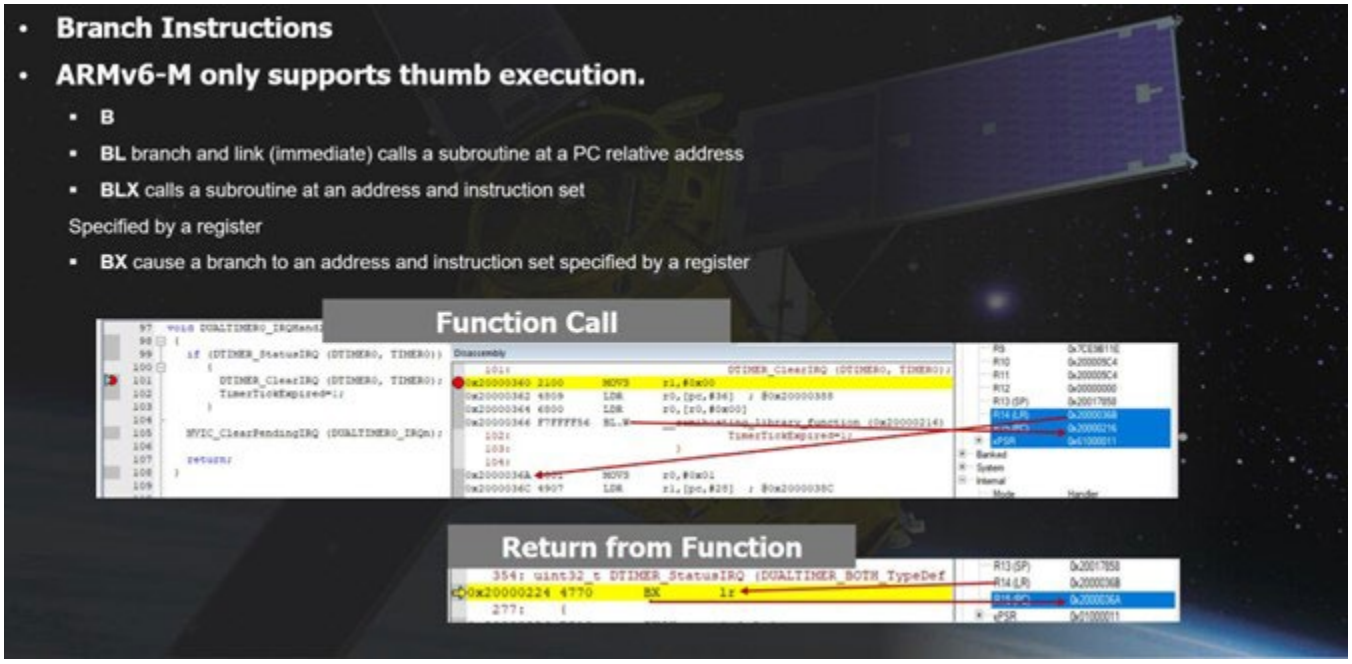
Arm Assembly Syntax

Continuing with the **ISA,** the picture shows the ARM assembly syntax. **Labels** are on the left, followed by **mnemonic, operands** and finally a **";"** is for comments. **Labels** are a symbolic representation of an address (the picture shows the "loop" label representing address **0x20000164**). **Mnemonic** is the name of the instruction (the picture shows the instructions highlighted in bold). Finally, **operands** are registers.



Arm Branch Instructions

Branch instructions BL, BLX and BX are subroutine instructions. When a subroutine is called, the assembly program uses the BL instruction. The BL instruction loads the LR with the address following the BL instruction and loads the PC with the address of first instruction of the subroutine. Finally, when returning from the subroutine, the “BX LR” loads the PC with the address loaded in the LR, and the main program resumes execution after the function call.

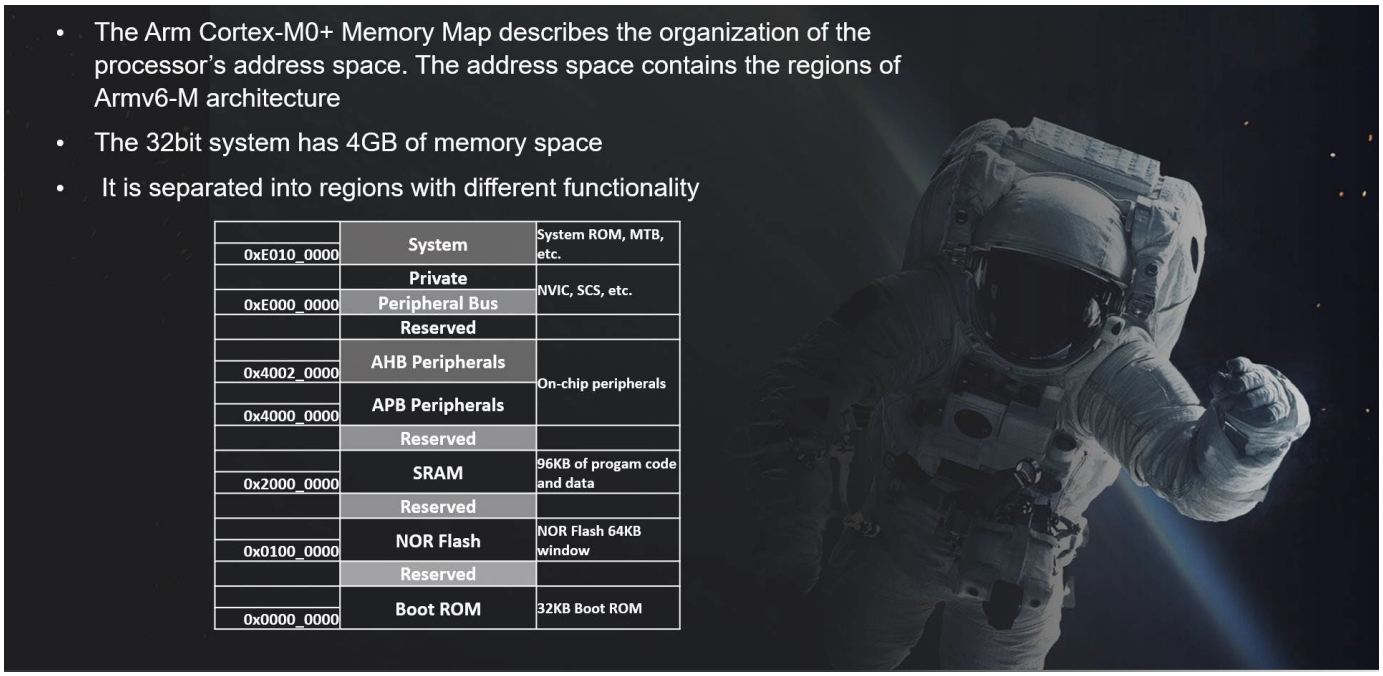


Memory Map

The ARM Cortex-M0+ **memory map** has **4 GB** of address space. The processor is separated into fixed regions. Hex 0x00_000_000 to hex 0x20_000_000 minus one is the code region. This region has both boot ROM and NOR Flash.

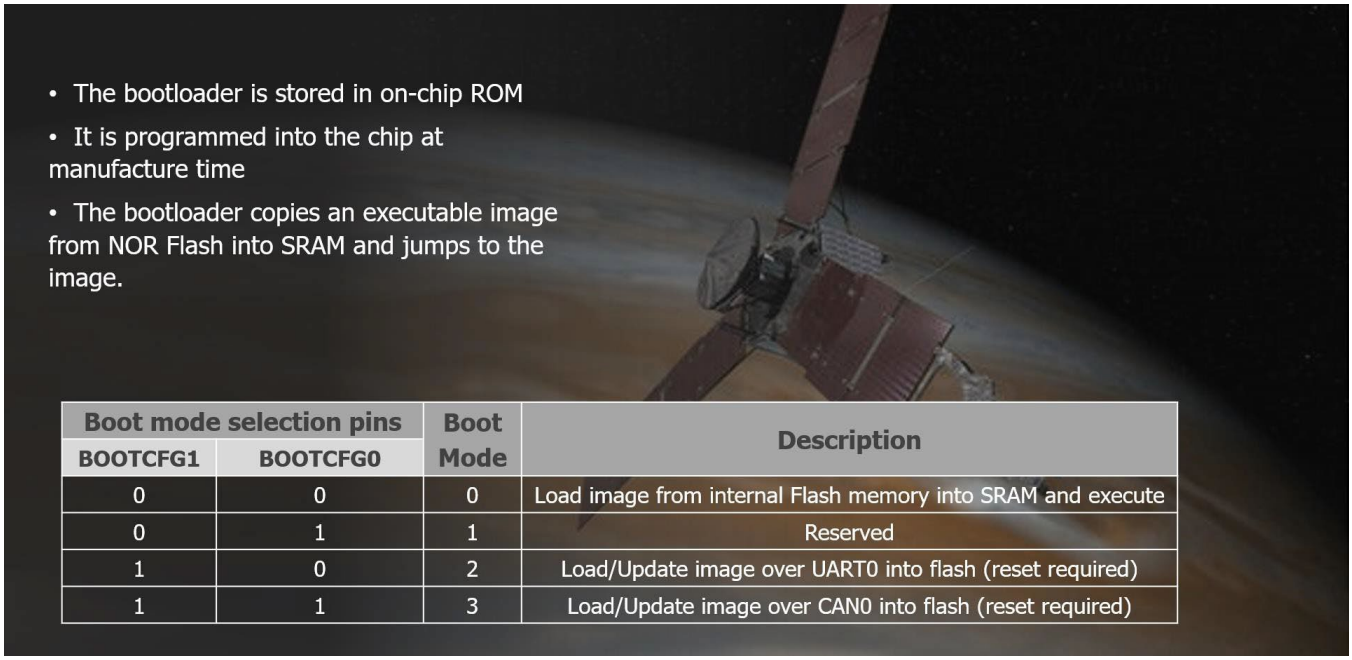
From hex 0x20_000_000 to hex 0x40_000_000 minus one is the SRAM region. Starting from hex 0x40_000_000 is the peripheral region.

And finally, starting from hex 0xE0_000_000 is the processor internal components region.



Code Region: Bootloader

The bootloader is stored in on-chip ROM at manufacture time. It has 4 boot modes: in mode 0, the bootloader loads an image from Flash to SRAM and jumps to the image to start program execution. Mode 2 and 3 are for loading/updating an image over UART0 and CAN0 respectively. For these modes, after loading/updating the image, the user needs to set mode 0 and reset the device for changes to take effect.

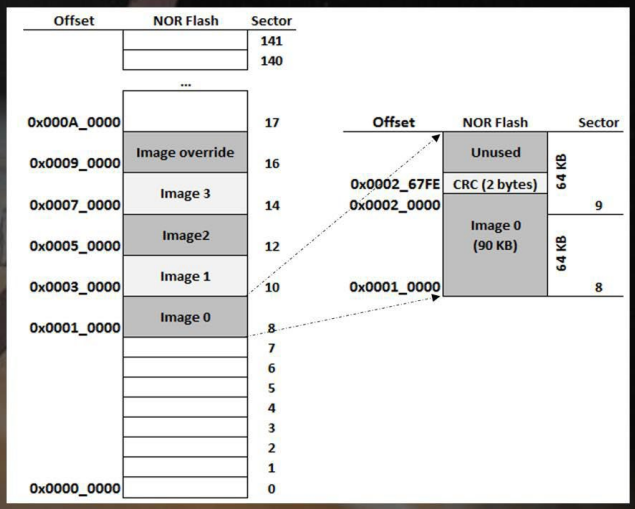


Boot mode selection pins		Boot Mode	Description
BOOTCFG1	BOOTCFG0		
0	0	0	Load image from internal Flash memory into SRAM and execute
0	1	1	Reserved
1	0	2	Load/Update image over UART0 into flash (reset required)
1	1	3	Load/Update image over CAN0 into flash (reset required)

Flash Program Image(s)

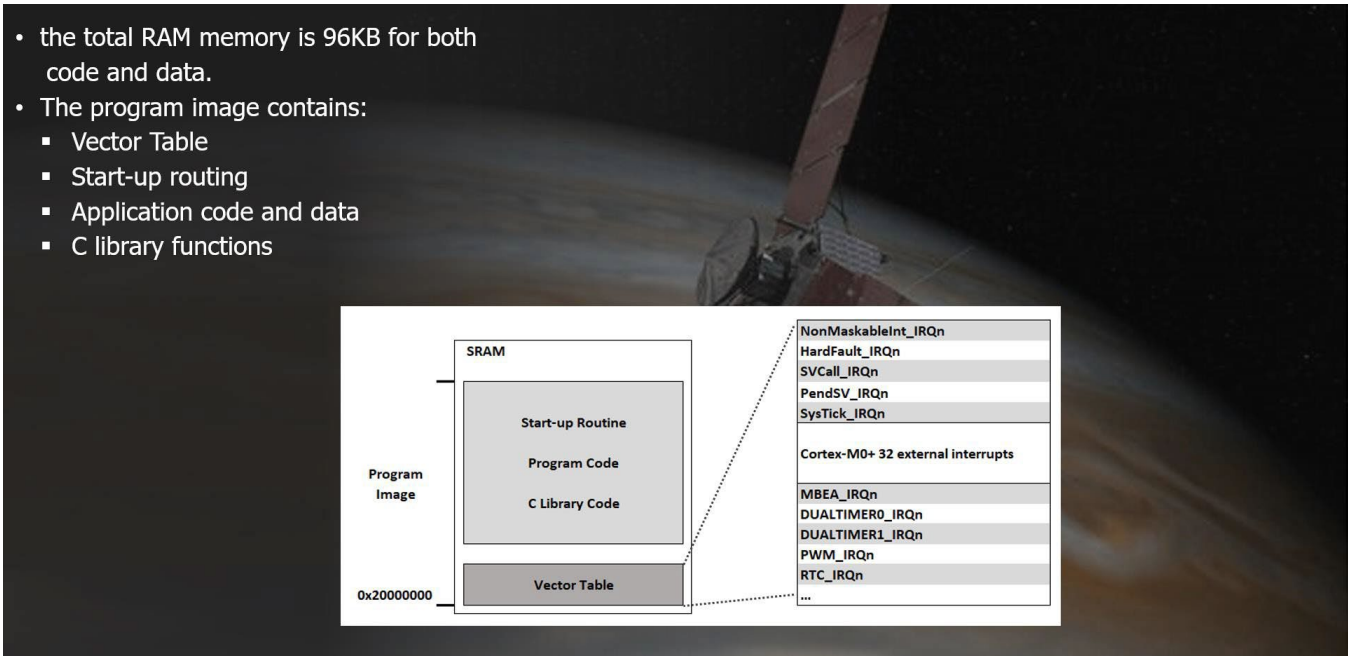
From before, using mode 2 and 3, the bootloader can load/update up to 4 images to Flash. Each program image is 90 KB with a 2-byte CRC.

- The NOR Flash is an 8MB of on-chip flash memory.
- Up to 4 program images can be programmed into it.
- Each image can be loaded/updated via UART0 or CAN0 interfaces.
- Each program image has two sectors of 64KB each.
- The maximum program image size is 90KB or 0x167FE (0x16800 - 2 bytes for CRC).



SRAM Program Image

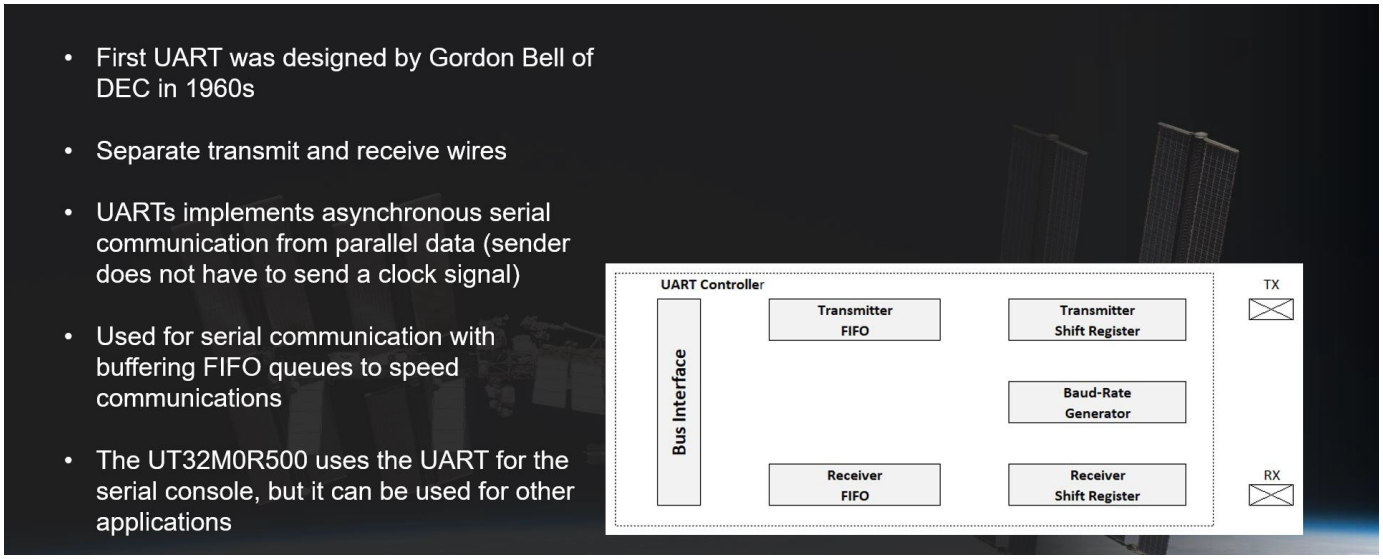
SRAM has the program image with a total of 96 KB for code and data. The program image contains the vector table, start-up routine, application code, data and C library functions. After reset, the processor reads the MSP value, which is the address of the beginning of the stack, then it reads the reset vector, which jumps to the beginning of the program and execution starts from there line by line.



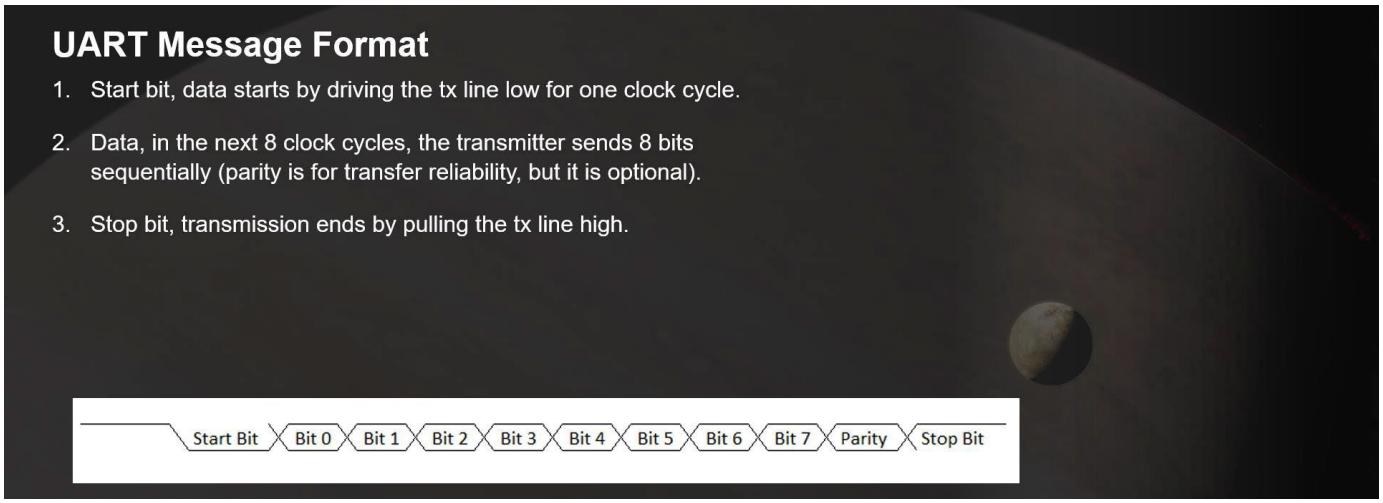
UT32M0R500 Peripherals

UART

Peripherals include serial interfaces, GPIO's, timers and analog interfaces. Starting with serial interfaces, the first **UART** was designed by Gordon Bell of DEC in 1960s. it has separate transmit and receive wires. The UART implements asynchronous serial communication (no clock needed, but devices must have the same baud rate) and uses FIFO queues to speed up communication. The UART is often used for early software debugging by printing values to a **Terminal** using **printf().**



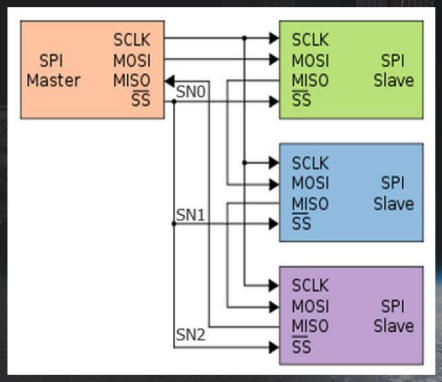
The message starts with first, the Start bit by driving the tx line low for one clock cycle. Second, data is transmitted in the next 8 clock cycles bit by bit with optionally sending the parity bit in the 9 clock cycle. Finally, the stop bit by pulling the tx line high to end the transmission



SPI

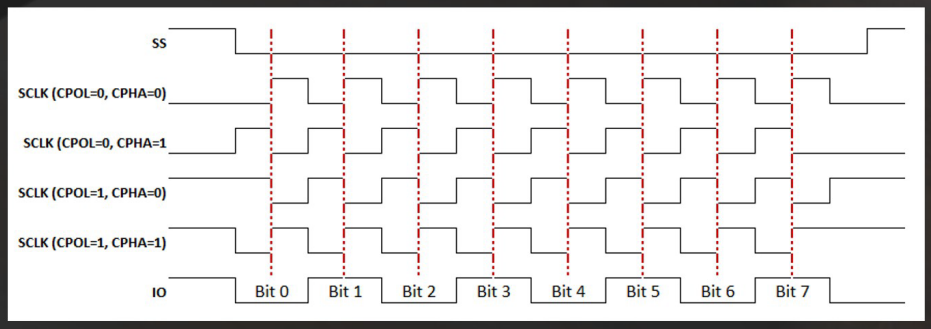
SPI Protocol was introduced by Motorola in late 1980s. the SPI is full duplex Synchronous serial communication. Master and slave operating from the same clock with User-selectable data rate at even-integer division of system clock. The UT32M0R500 SPI supports Master Mode only with user-selectable data widths of 4-16 bits.

- Protocol introduced by Motorola in late 1980s
- Synchronous serial communication
  - Master and slave operate from the same clock
- Full duplex serial communication
  - Data is transmitted and received by using separate lines
- The UT32M0R500 SPI supports Master Mode only
- 4 line protocol
  - **SCLK** - Clock generated by the master
  - **MOSI** - Master Out, Slave In – transmits data from master
  - **MISO** - Master In, Slave Out – transmits data form slave
  - **SS** - Slave Select – line asserted to select a particular slave
- User-selectable SPI data widths of 4 – 16 bits
- Slaves do not need a unique address
- User-selectable data rate at even-integer division of system clock, min of 2



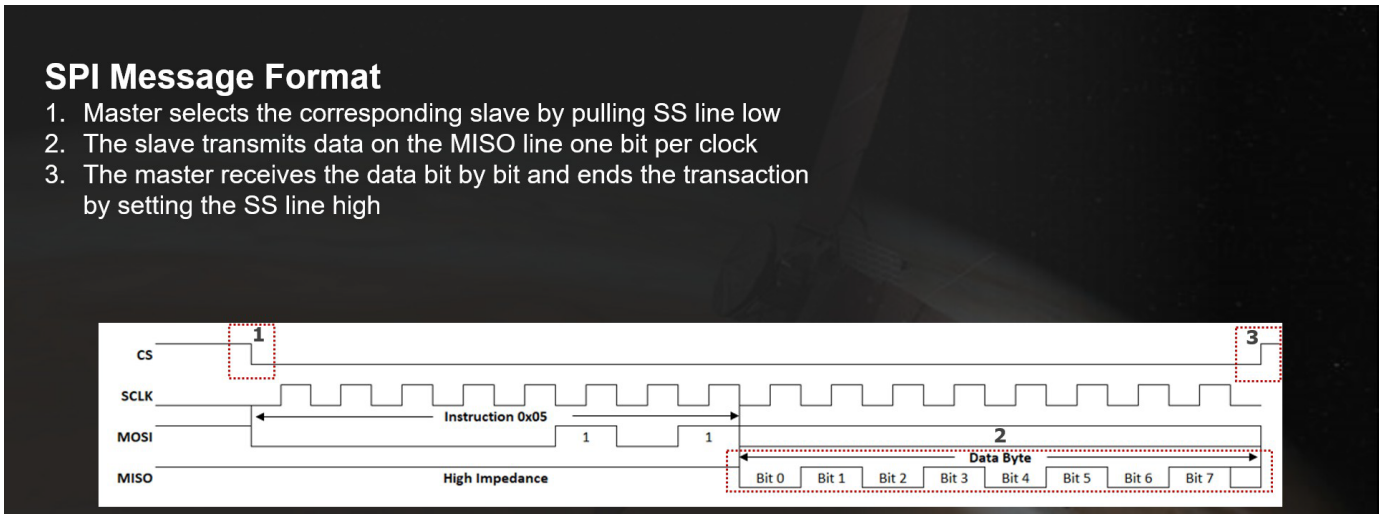
The SPI has 4 different clock modes: in Mode 0, the clock starts low and data is sampled on the leading rising edge of the clock; in mode 1, clock starts low and data is sampled on the trailing falling edge of the clock; in mode 2, clock starts high and data is sampled on the leading falling edge of the clock; and in mode 3, clock starts high and data is sampled on the trailing rising edge of the clock.

SPI Clock Modes



Clock Polarity and Phase		SPI Mode	Data Sampling
CPOL	CPHA		
0	0	0	Leading rising edge of the clock
0	1	1	Trailing falling edge of the clock
1	0	2	Leading falling edge of the clock
1	1	3	Trailing rising edge of the clock

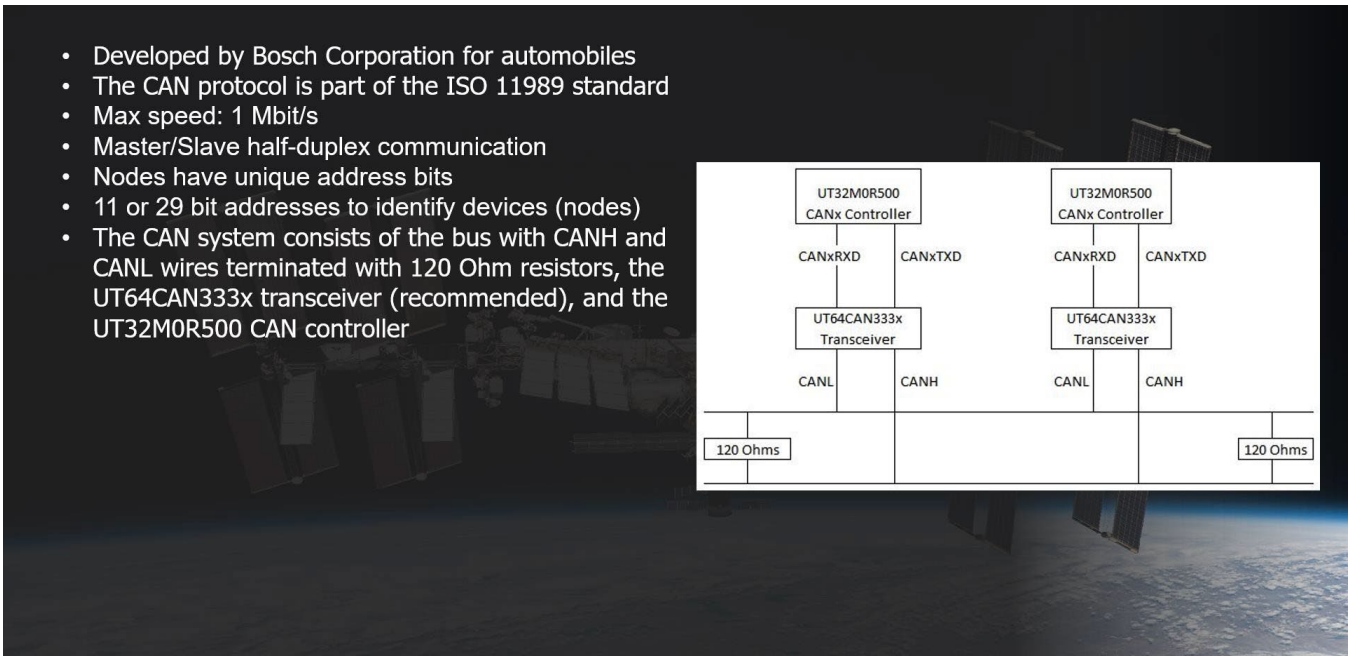
The message starts with first, driving SS line low to the corresponding slave. Second, slave transmits data on the MISO line one bit per clock. Finally, the master receives the data bit by bit and ends the transaction by pulling the SS line high.



CAN

CAN was initially developed by Bosh Corporation for automobiles, but since then, it has been used in industrial automation and control applications. The protocol is part of the ISO 11989 standard. CAN has a max speed of 1 Mbit/s with Master/Slave half-duplex communication, and nodes have unique address bits with 7 or 11 bit addresses to identify devices (nodes).

CAN operates at data rates of up to 1 Mbits/s. Master controls the bus and addresses a particular slave for communication. Transceiver uses 120 Ohms. Resistors pull up lines to VCC while Open-collector pulls lines down to GND.



The CAN bus contains two wires in differential mode for one logic bit; the wires are CANH and CANL. The state of the transmitter is either dominant or recessive. When two or more nodes are competing, their output follows a wired-AND mechanism with the dominant state overriding the recessive one.

CAN Nominal Bus Levels

The CAN bus contains two wires in differential mode for one logic bit; the wires are CANH and CANL. The state of the transmitter is either dominant or recessive

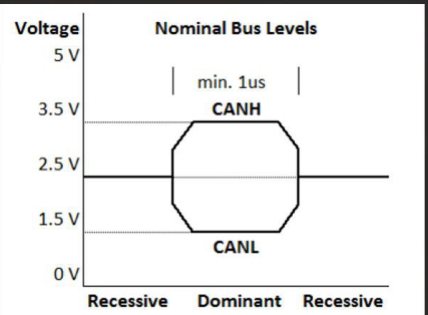


Figure 4: CAN Nominal Bus Levels

CAN Arbitration

When two or more nodes are competing, their output follows a wired-AND mechanism with the dominant state overriding the recessive one, see Figure 5 for CAN arbitration.

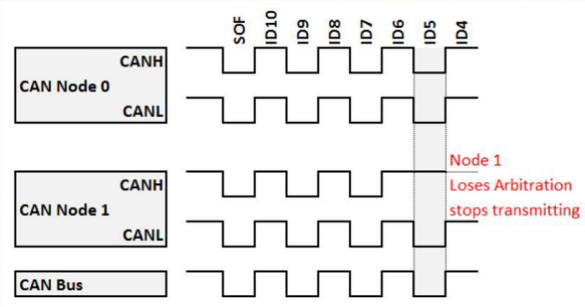
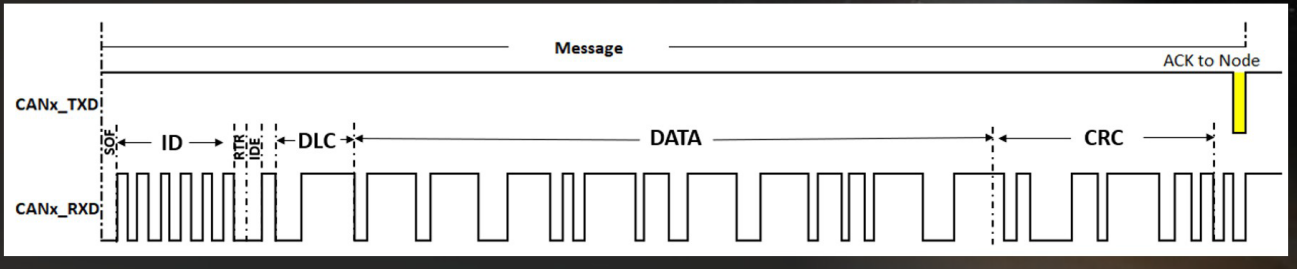


Figure 5: CAN Arbitration

The message starts with first, sending the start of frame (SOF) low. Second, the node sends the arbitration field which consist of an 11-bit identifier which also determines the priority of the message when nodes contend for the bus and data frame message type. Third, the node sends IDE low for basic mode. Fourth, DLC specifies the number of bytes. Then, the actual data up to 8 bytes followed by a 15-bit CRC for error detection. Finally, the controller sends an ACK when correctly receiving the message.

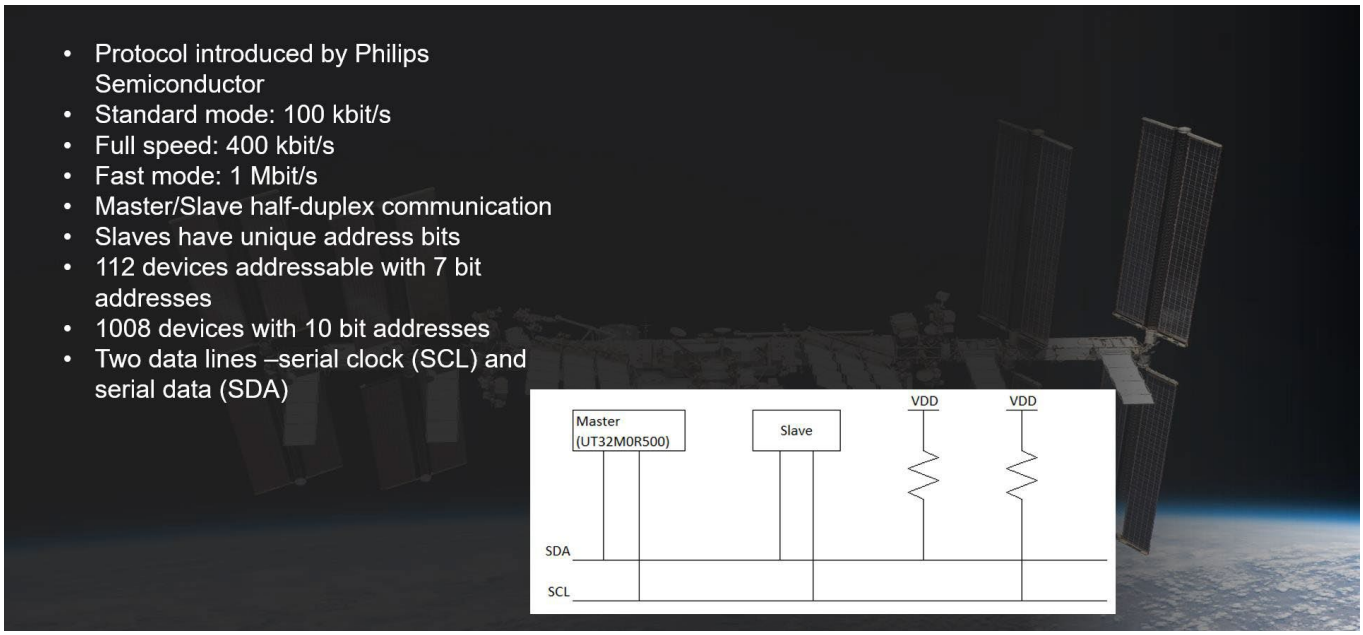
CAN Message Format

1. the start of frame (SOF)
2. Then the ID which also represents the priority
3. Remote transmission request (RTR) low for data frame
4. Data length code (DLC) specifies number of data bytes
5. Data bytes
6. Cyclic redundancy check (CRC) checksum for error detection
7. Finally, acknowledge

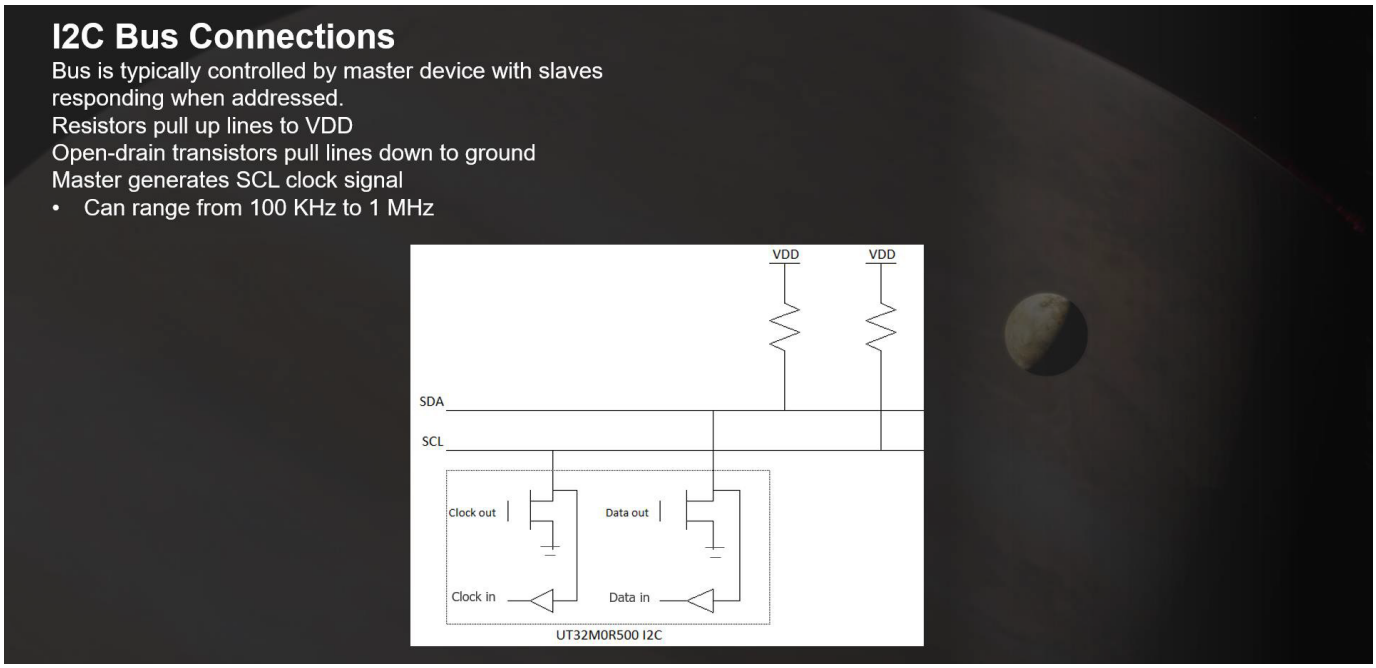


I2C

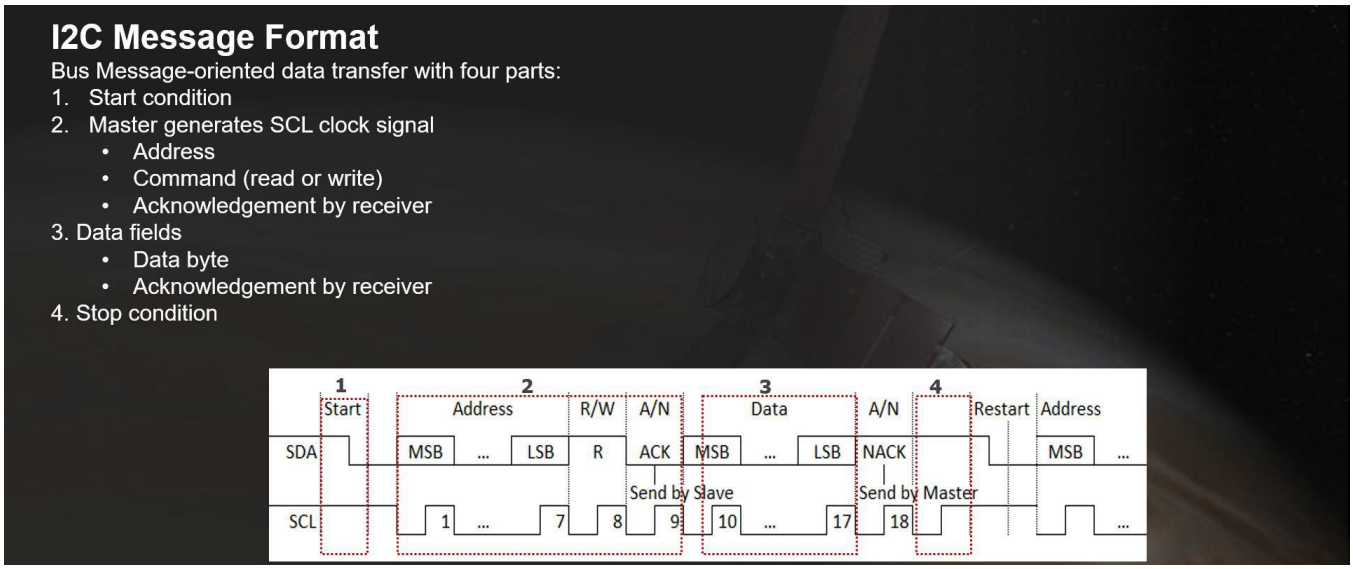
I2C Protocol was introduced by Philips Semiconductor. the UT32M0R500 I2C has standard mode, full speed and fast mode with 100, 400 Kbits/s and 1 Mbit/s respectively. Communication is Half duplex with two data lines SCL and SDA. The Slave has unique address bits for identification.



I2C operates at data rates of up to 1 Mbits/s with the master generating the clock signal. The master controls the bus and addresses a particular slave for communication. Resistors pull up lines to VDD while Open-drain pulls lines down to GND.

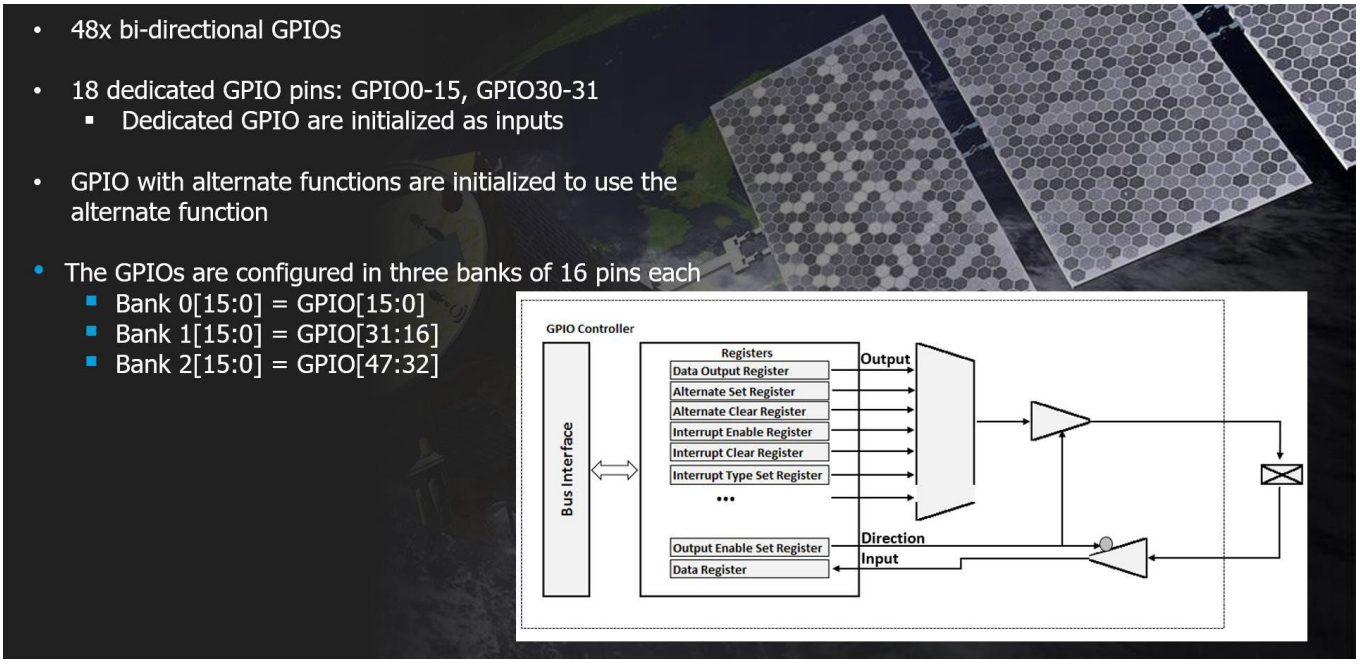


The message starts with first, a start condition, which is generated by pulling SDA low. Second, the next 7 bits are for addressing a particular device. The 8th bit indicates a read or write mode by the master. In write mode, the slave will receive data from the master; In read mode, the slave will send data to the master. Every byte is finished with a 9th acknowledge bit. Finally, the master ends the transaction by pulling SCL and SDA line high.

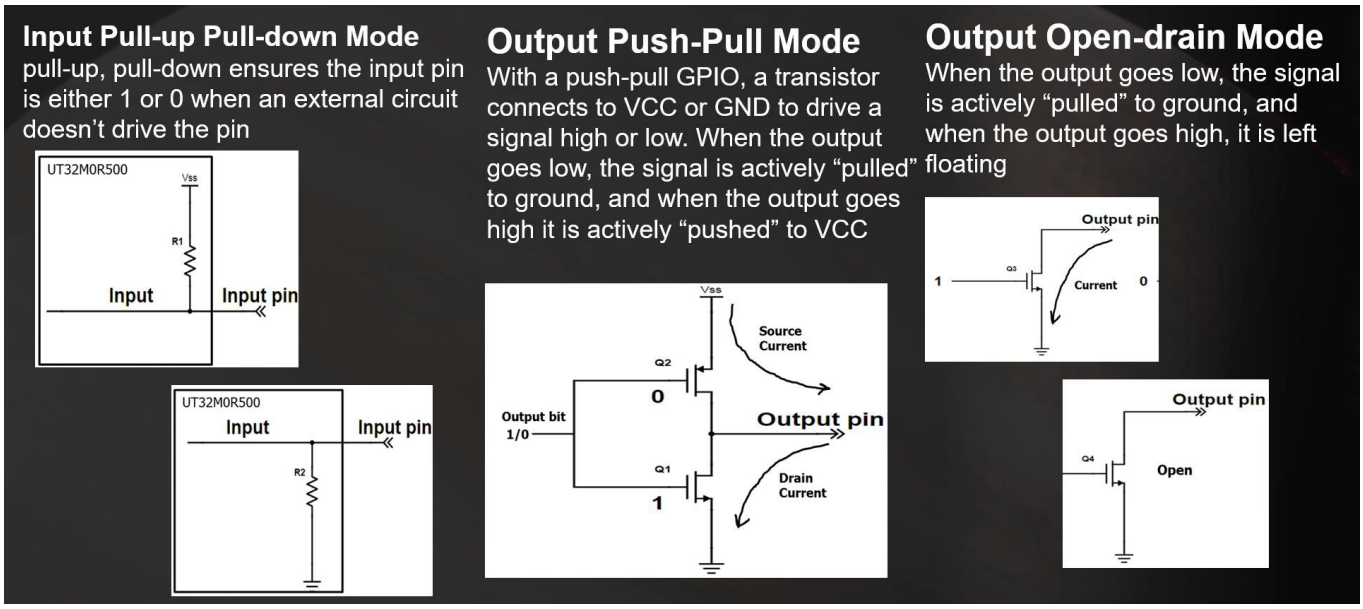


GPIO

The UT32M0R500 has 48 bi-directional GPIO's with 18 dedicated GPIO's initialized as inputs. GPIO's with alternate functions are initialized to use the alternate function. The GPIO's are configured in three banks of 16 pins each.

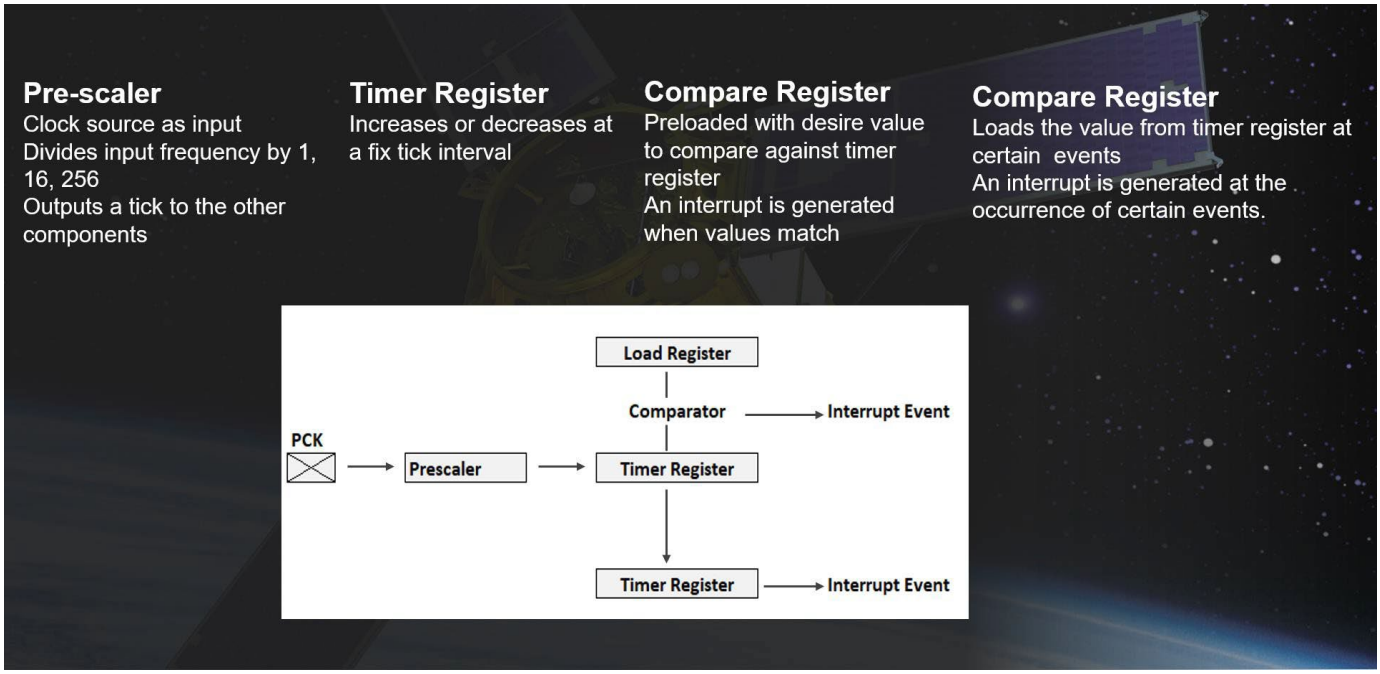


Inputs are in either pull-up or pull-down mode to ensure the input pins are either 1 or 0 when an external circuit doesn't drive the pin. **Outputs** are in either push-pull mode or open-drain mode. In push-pull mode, the output is either 1 or 0 and in open-drain, the output is pull low to ground or left floating (High-Z).



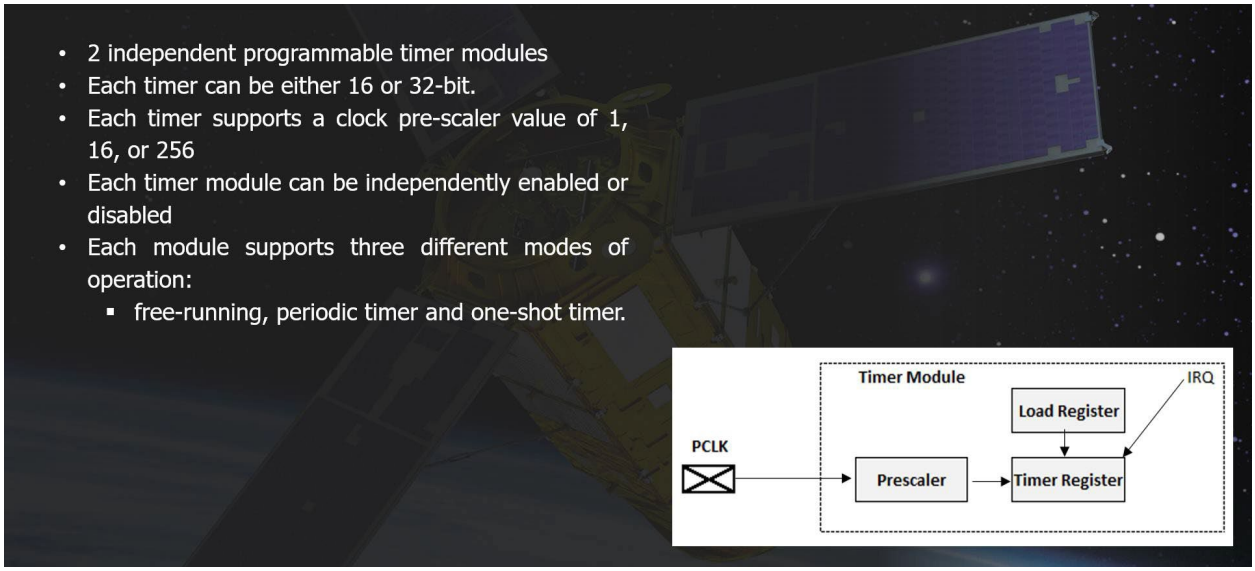
Timers

A **Timer** is a counter with a tick as the basic time unit and generates an interrupt when it reaches a predefined value. It has four main components: pre-scaler, compare register, timer register and capture register. A pre-scaler divides the system clock by a predefined value either 1, 16, 256 and outputs the timer tick. A timer register increases or decreases to a specified number of ticks. A compare register is preloaded with a desired value and if it matches the timer register value, an interrupt is generated. A capture register takes a snapshot of the timer register at certain moments in time.



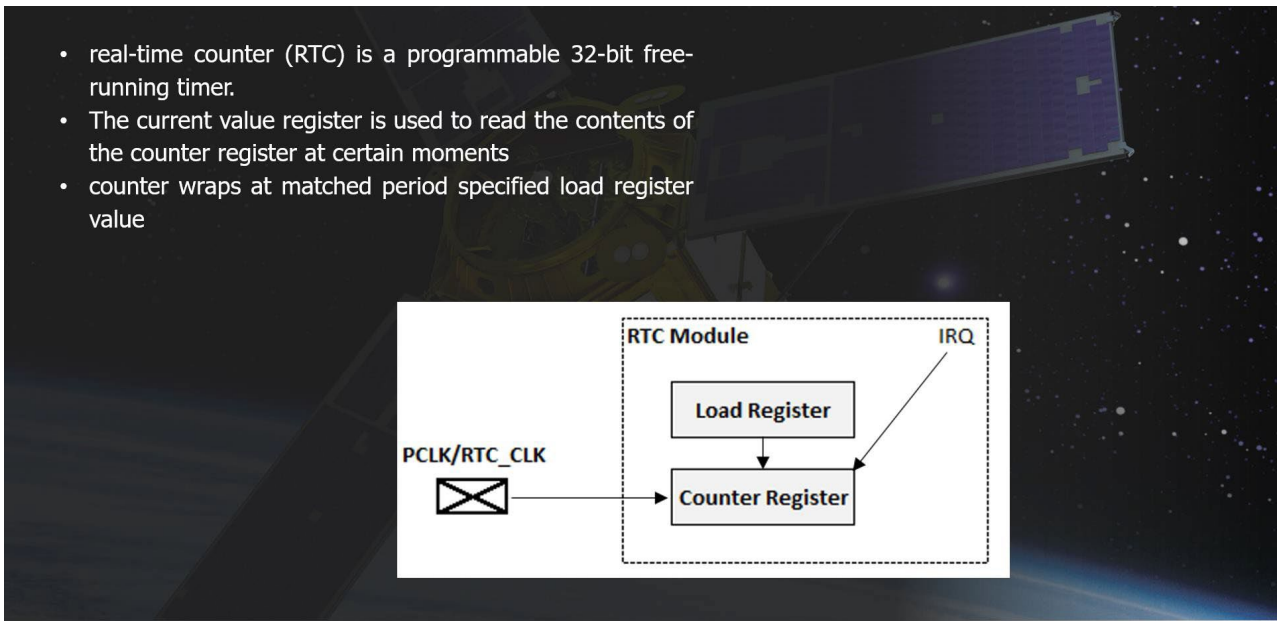
Timers: Dual Timers

The UT32M0R500 timers consist of dual timers, real-time counter (RTC), watchdog and PWM. For dual timers, each timer can be either 16 or 32-bit with a clock pre-scaler value of 1, 16, or 256 and supports three different modes of operation: free-running, periodic timer and one-shot timer.



Timers: Real Time Counter (RTC)

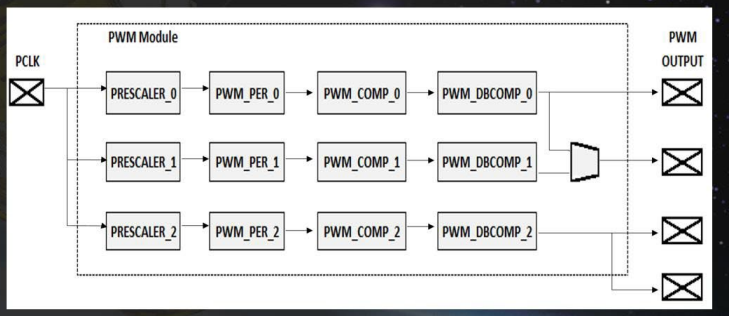
The real-time counter (RTC) is a programmable 32-bit free-running timer. The current value register is used to read the contents of the counter register at certain moments in time, and the counter wraps at matched period specified by the load register value.



Timers: PWM

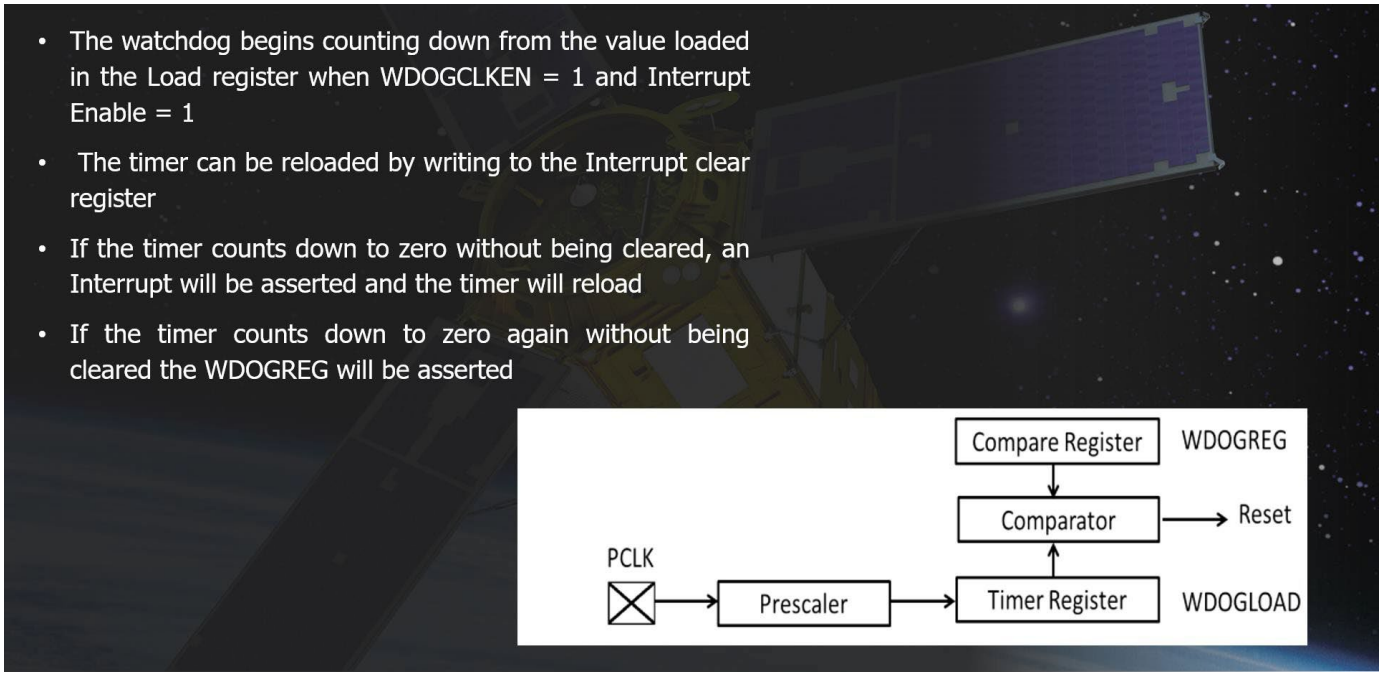
Each **PWM** controller can have a single output, or the three controllers can be combined to form two-paired outputs. Each PWM device is configured as a 16-bit channel, programmable dead-band scaler, programmable clock scaler, and all three devices have a single combined Interrupt.

- 1 pulse width modulation (PWM) module with three separate controllers.
- Each PWM controller can have a single output, or the three controllers can be combined to form two-paired outputs.
- Each PWM device is configured as a 16-bit channel.
- Each PWM device includes a programmable dead-band scaler with a range from 20ns to 81,920ns, programmable clock scaler for a max 332ms pulse, and all three devices have a single combined Interrupt.



Timers: Watchdog

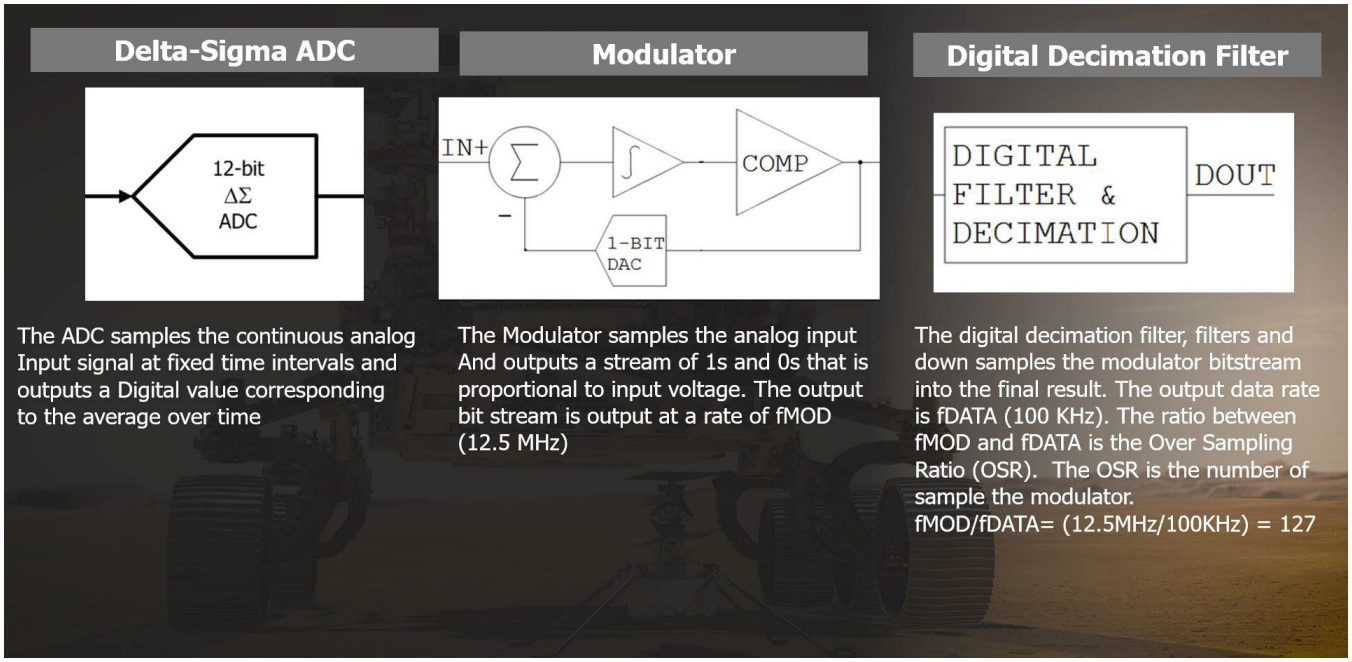
The **watchdog** begins counting down from the value loaded in the Load register. The timer can be reloaded by writing to the Interrupt clear register. If the timer counts down to zero without being cleared, an Interrupt will be asserted and the timer will reload. If the timer counts down to zero again without being cleared, the wdog output pin will be asserted.



ADC

The **Delta-sigma ADC** samples the continuous analog Input signal at fixed time intervals and outputs a Digital value corresponding to the average over time. The ADC has two main components: the modulator and the digital decimation filter. The **Modulator** samples the analog input and outputs a stream of 1s and 0s that is proportional to input voltage. The output bit stream is output at a rate of fMOD (12.5 MHz as the default value).

The **digital decimation filter,** filters and down samples the modulator bitstream into the final result. The output data rate is fDATA (100 KHz as the default value). The ratio between fMOD and fDATA is the Over Sampling Ratio (OSR). The OSR is the ratio between fMOD and fDATA which the default value is (12.5MHz/100KHz) = 127.



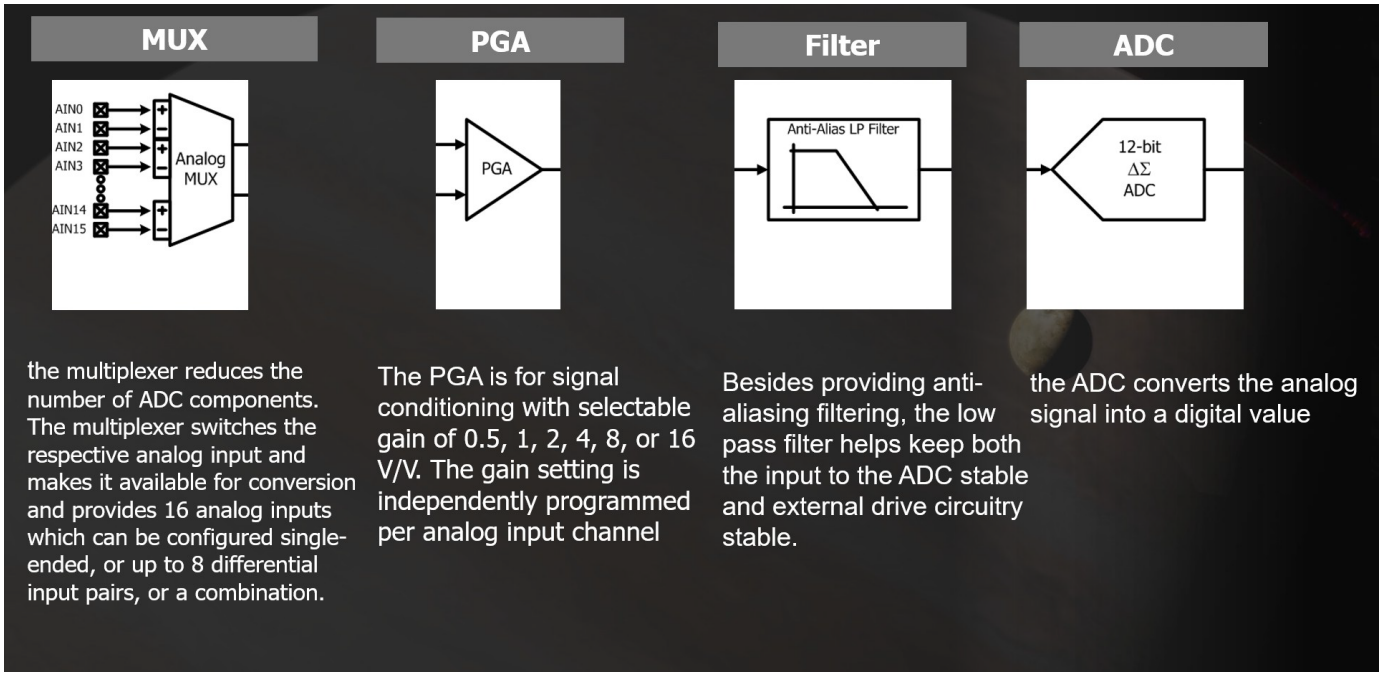
ADC: Analog Interface

The Delta-sigma ADC input signal chain consist of a multiplexer, programmable gain amplifier (PGA), anti-aliasing low pass filter, and ADC. The **multiplexer** reduces the number of ADC components, switches the respective analog input and makes it available for conversion. It provides 16 analog inputs which can be configured single-ended, or up to 8 differential input pairs, or a combination.

Next, The **PGA** is for signal conditioning with selectable gain of 0.5, 1, 2, 4, 8, or 16 V/V. The gain setting is independently programmed per analog input channel.

Besides providing **anti-aliasing filtering,** the low pass filter helps keep both the input to the ADC stable and external drive circuitry stable.

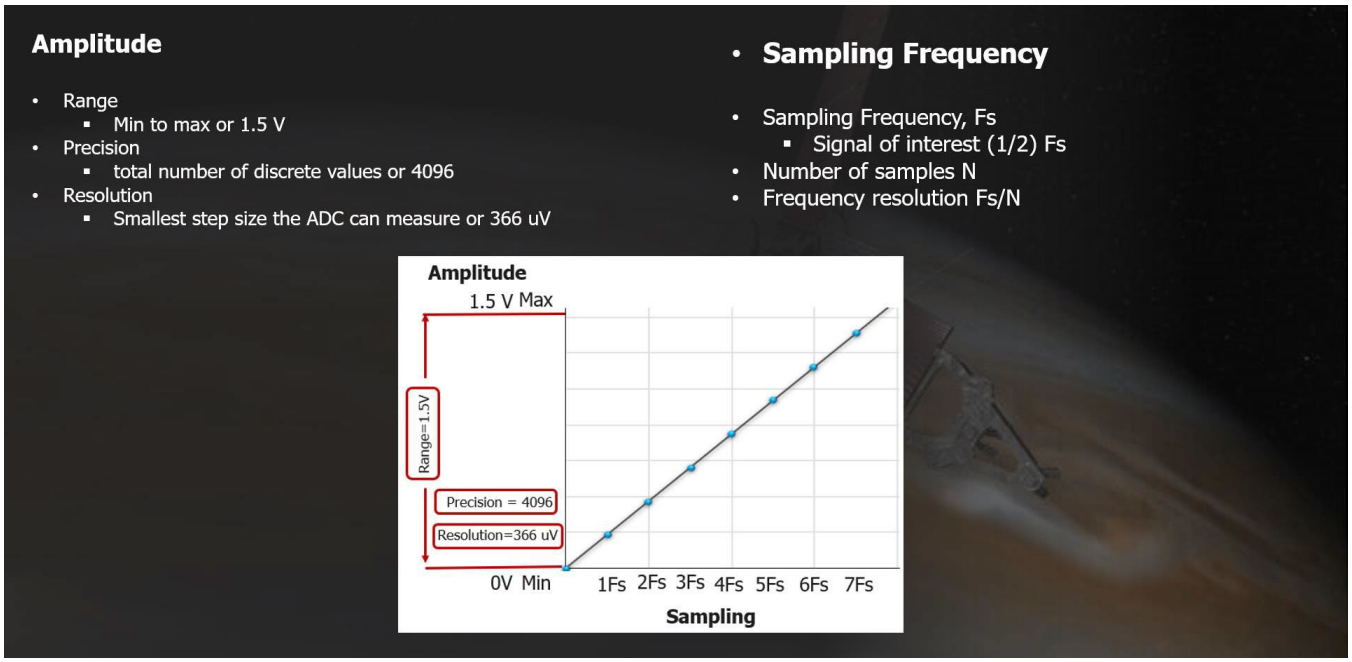
Finally, as stated before, the **ADC** converts the analog signal into a digital value.



ADC: Sampling

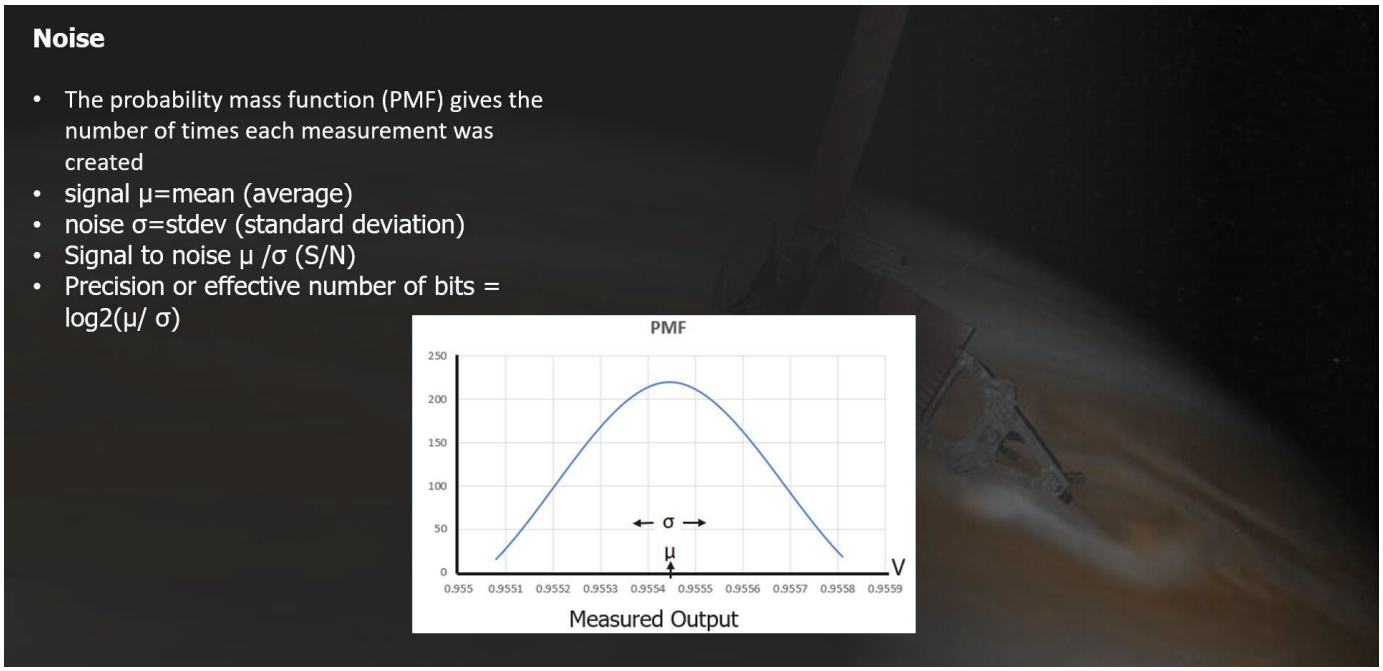
The ADC converts an analog signal that is continuous in time and amplitude to something discrete both in time and amplitude, so the discrete value will be an approximation to the analog signal. **Amplitude** is defined in terms of range, precision, and resolution. The Range is the Min to max or 1.5 V for ADC single-ended channels; Precision is the total number of discrete values or 4096 for the ADC; and resolution is the smallest step size the ADC can measure or 366 uV.

In terms of **sampling,** the ADC defines the sampling frequency, number of samples and frequency resolution. Sampling frequency is defined as the minimum frequency that signals can be sampled without violating the Nyquist theorem, which states that if we sample at 2Khz, the maximum signal we can represent is 1KHz. Frequency resolution is defined as the sampling frequency/number of samples. For instance, if we have a buffer size of 8, N=8, then the frequency resolution is 1KHz.



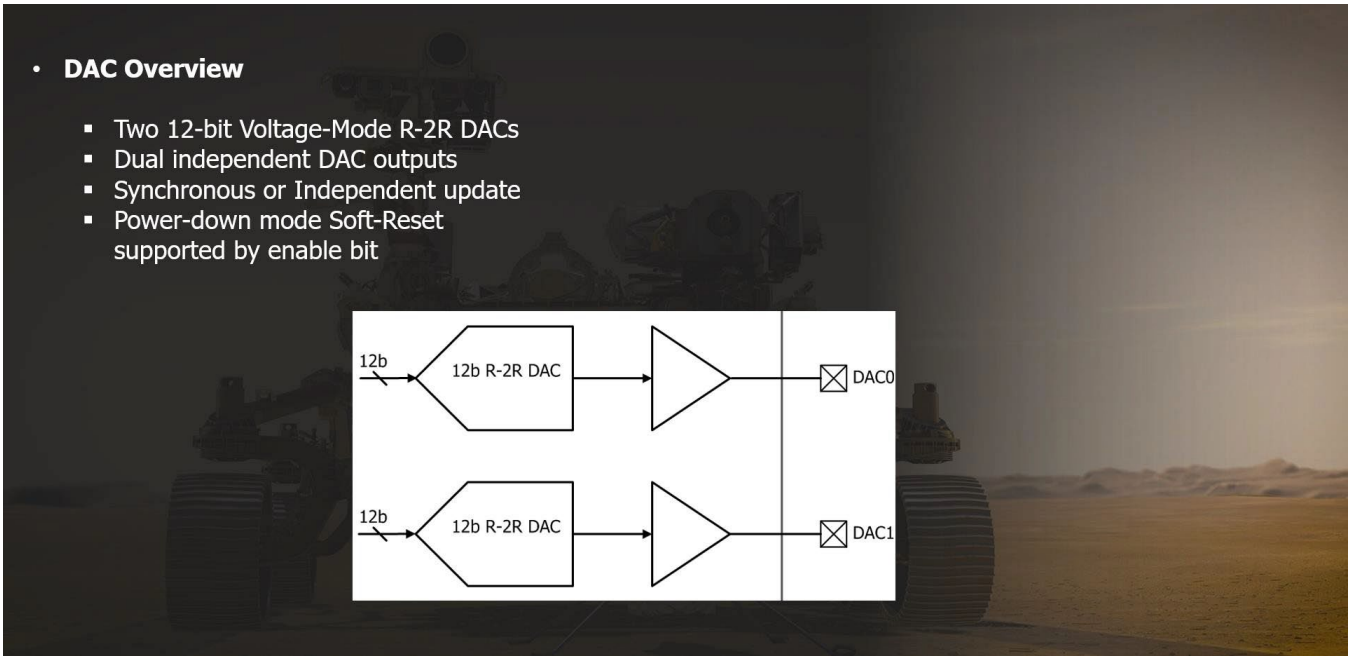
ADC: Noise

Noise is best explained by looking at a probability mass (PMF) function. The PMF gives the number of times each measurement was created. From the graph, the average or mean is the signal and the standard deviation is the noise. Comparing the average with the standard deviation, we get the signal-to-noise, which is the ratio of the amplitude of the signal relative to the noise. Now if we take the log base 2 of mu and sigma, we'll get the equivalent number of bits associated with the noise, and if the system is too noisy, in some cases, the resolution of a noisy conversion system can be improved by averaging.



DAC

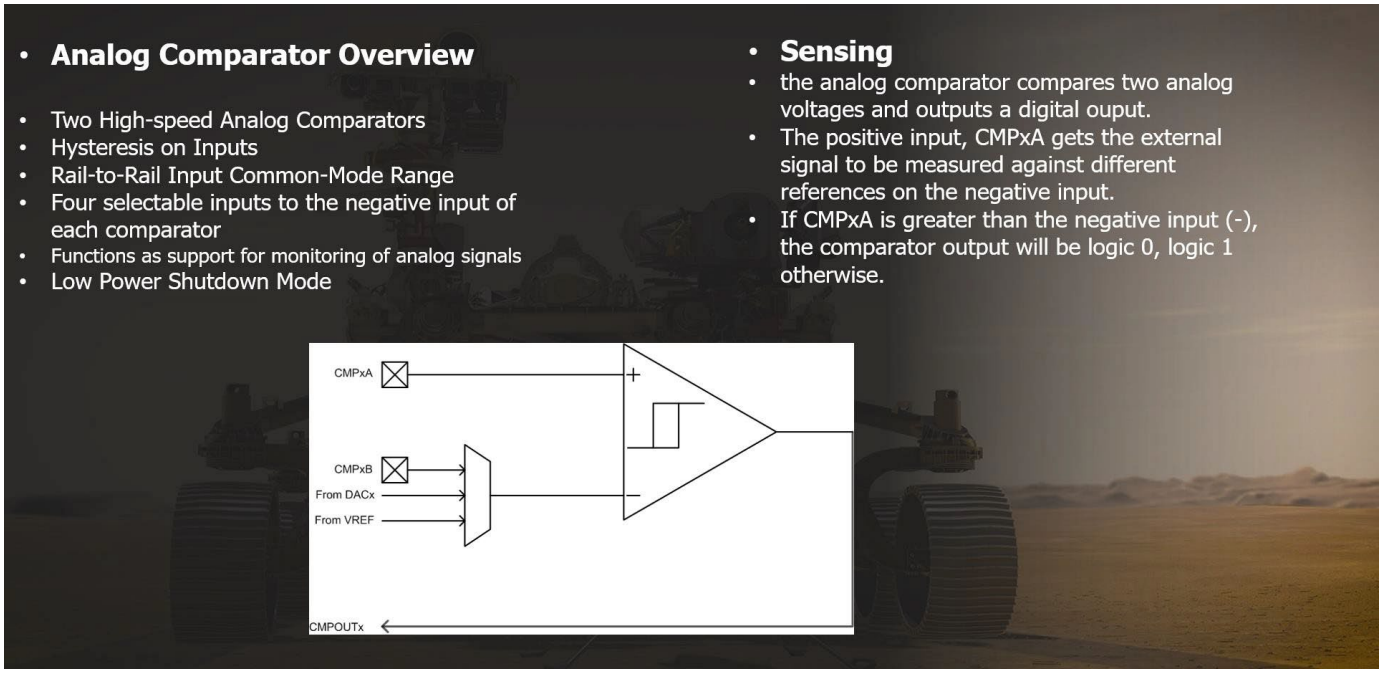
The **DAC** has similar parameters to the ADC. The DAC has two 12-bit Voltage-Mode R-2R DACs; dual independent DAC outputs; synchronous or Independent update; and power-down mode is supported.



Analog Comparator

The UT32M0R500 has two High-speed Analog Comparators with Hysteresis on Inputs and Rail-to-Rail Input Common-Mode Range; Four selectable inputs to the negative input of each comparator.

For Sensing, the analog comparator compares two analog voltages and outputs a digital value. The positive input, CMPxA gets the external signal to be measured against different references on the negative input. If CMPxA is greater than the negative input (-), the comparator output will be logic 0, logic 1 otherwise.

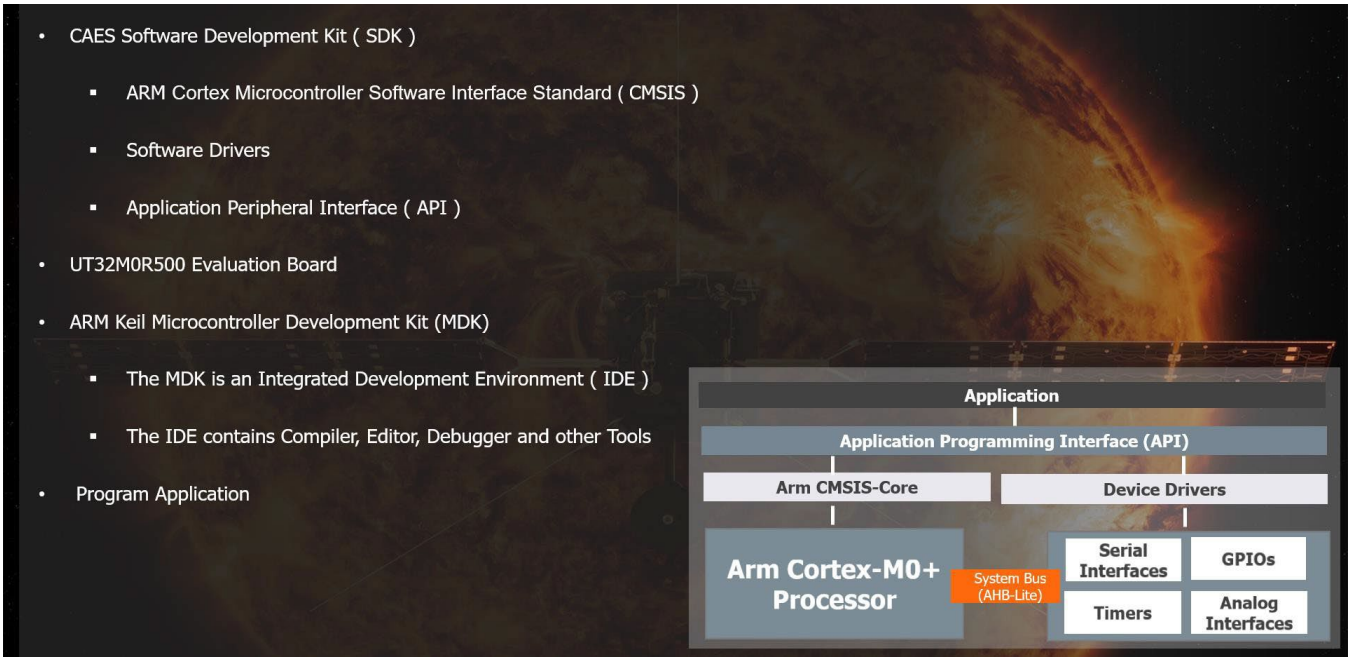


Frontgrade Software Development Kit (SDK)

Frontgrade Software Development Kit (SDK) includes:

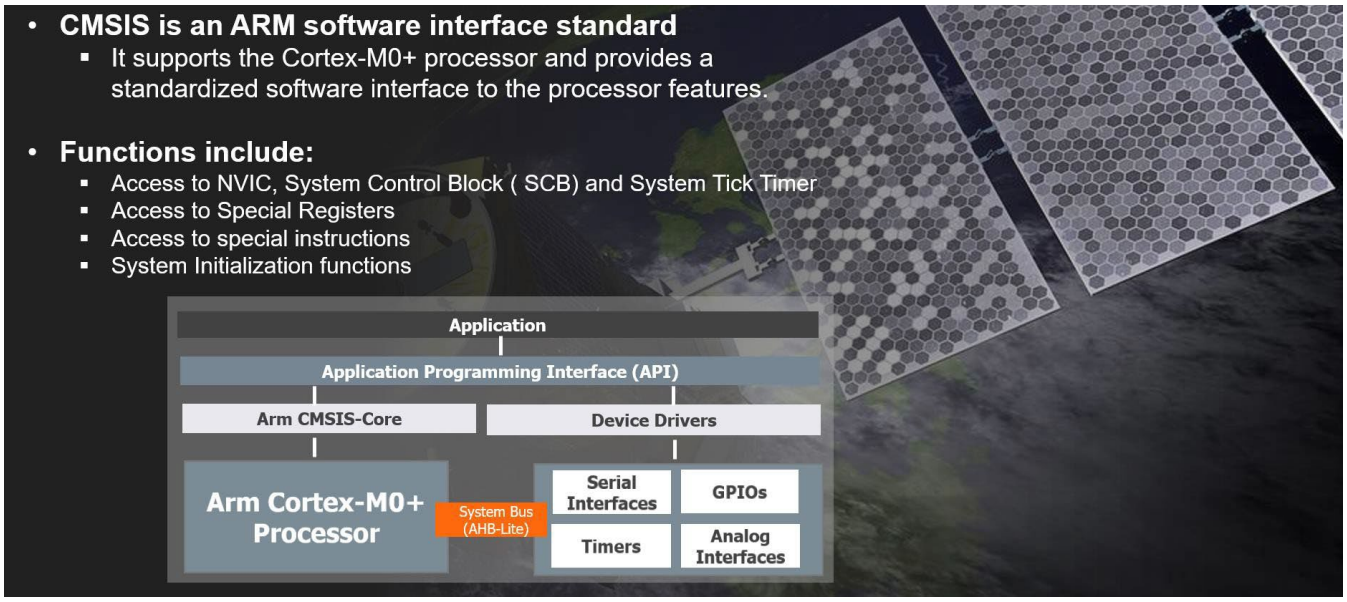
- ARM Cortex Microcontroller Software Interface Standard **(CMSIS)**.
- Software Drivers.
- and Application Peripheral Interface **(API)**.

Besides the SDK, the app note goes over UT32M0R500 Evaluation Board, then the ARM Keil Microcontroller Development Kit (MDK), which is an Integrated Development Environment **(IDE)** that contains Compiler, Editor, Debugger and other Tools, and finally, a program application.

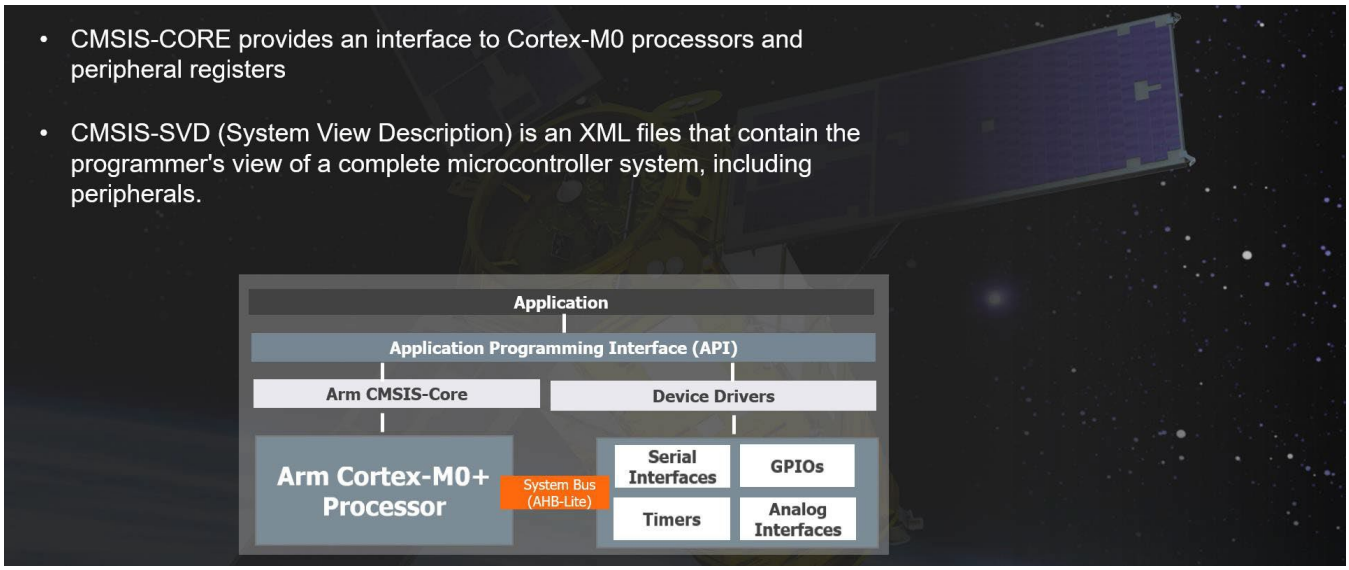


Cortex Microcontroller Software Interface Standard (CMSIS)

Cortex Microcontroller Software Interface Standard (CMSIS) is an ARM software interface standard. It supports the Cortex-M0+ processor and provides a standardized software interface to the processor features. **Functions** include: Access to NVIC, System Control Block (SCB) and System Tick Timer; Access to Special Registers; Access to special instructions; and System Initialization functions;

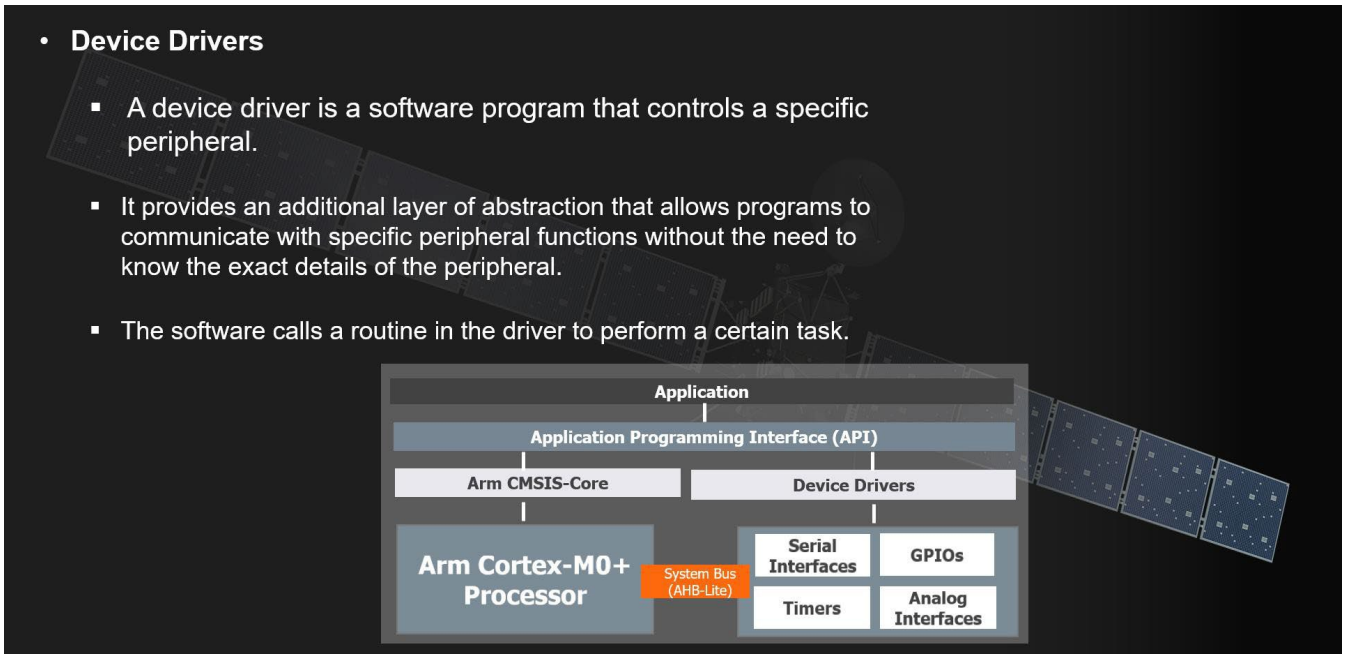


CMSIS-CORE provides an interface to Cortex-M0+ processors and peripheral registers. **CMSIS-SVD** (System View Description) is an XML file that contains the programmer's view of a complete microcontroller system, including peripherals.



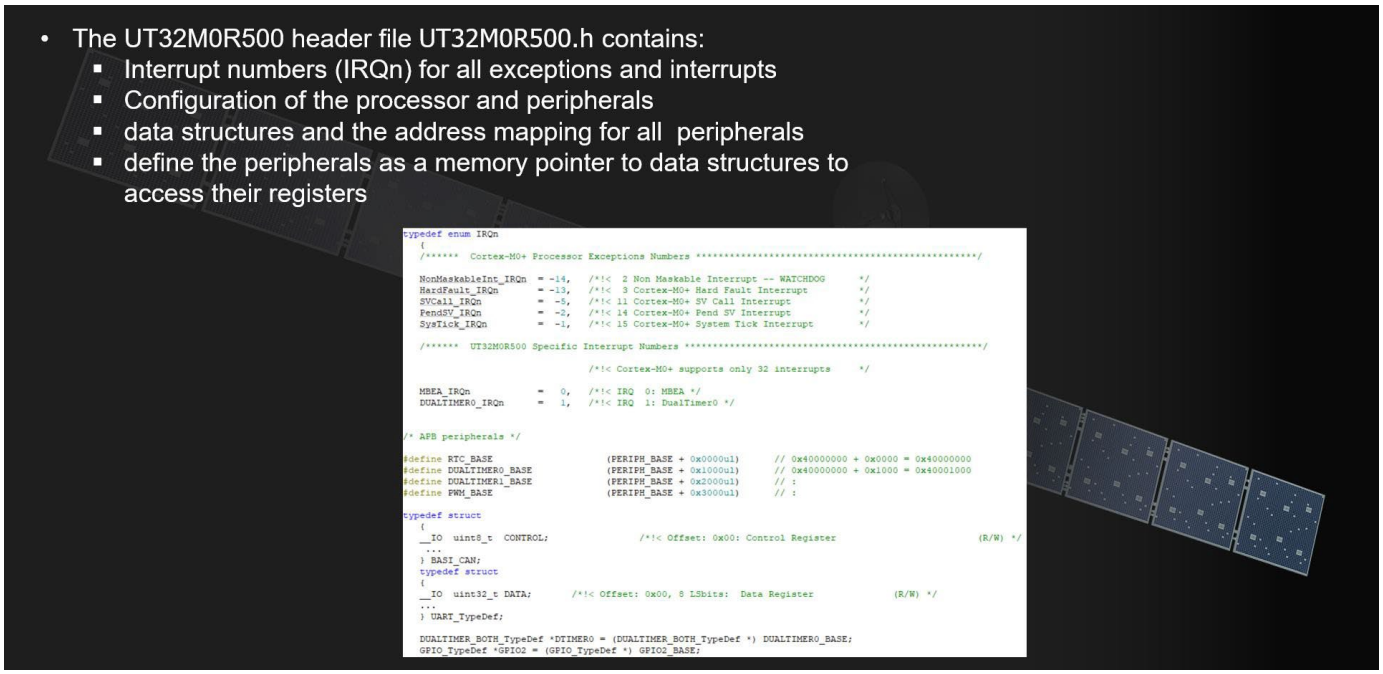
Device Drivers

A **device driver** is a software program that controls a specific peripheral. It provides an additional layer of abstraction that allows programs to communicate with specific peripheral functions without the need to know the exact details of the peripheral. The software calls a routine in the driver to perform a certain task.



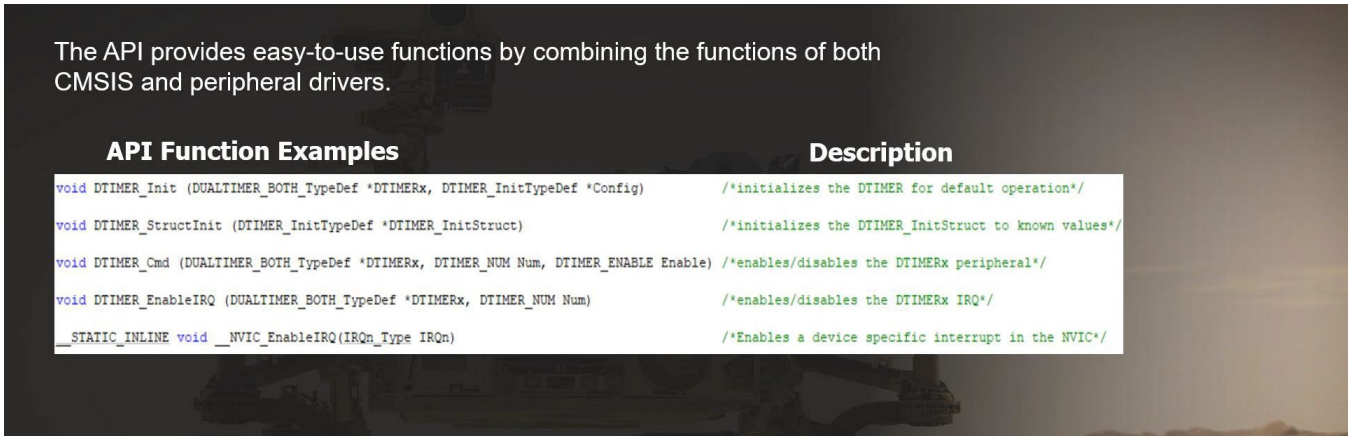
UT32M0R500 Header File

Device drivers define all the different peripherals in the UT32M0R500 header file using the same standard format. The UT32M0R500.h header file contains: Interrupt numbers (IRQn) for all exceptions and interrupts; configuration of the processor and peripherals, data structures and the address mapping for all peripherals; finally, the picture shows that the application can define peripherals as a memory pointer to data structures to access their registers.



Application Programmer Interface (API)

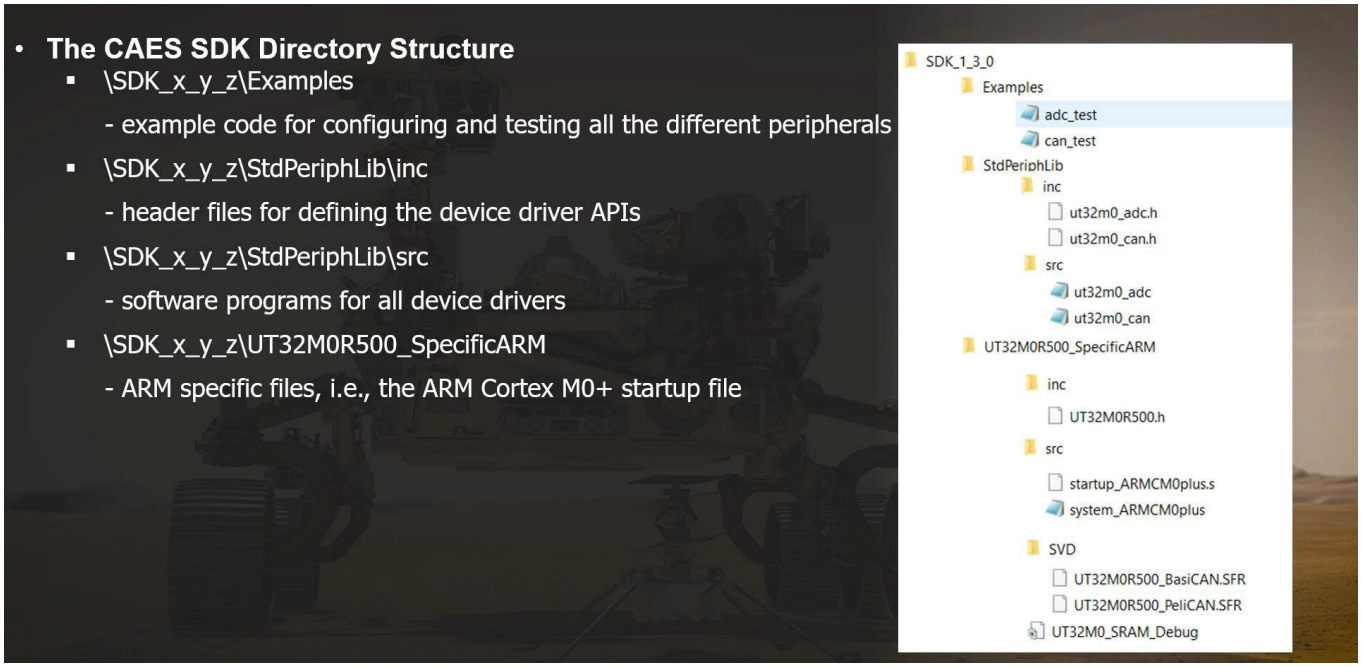
The application programming interface (API) provides easy-to-use functions by combining the functions of both CMSIS and peripheral drivers. The picture shows API function examples for the timer with description for each of the function calls.



SDK Library Structure

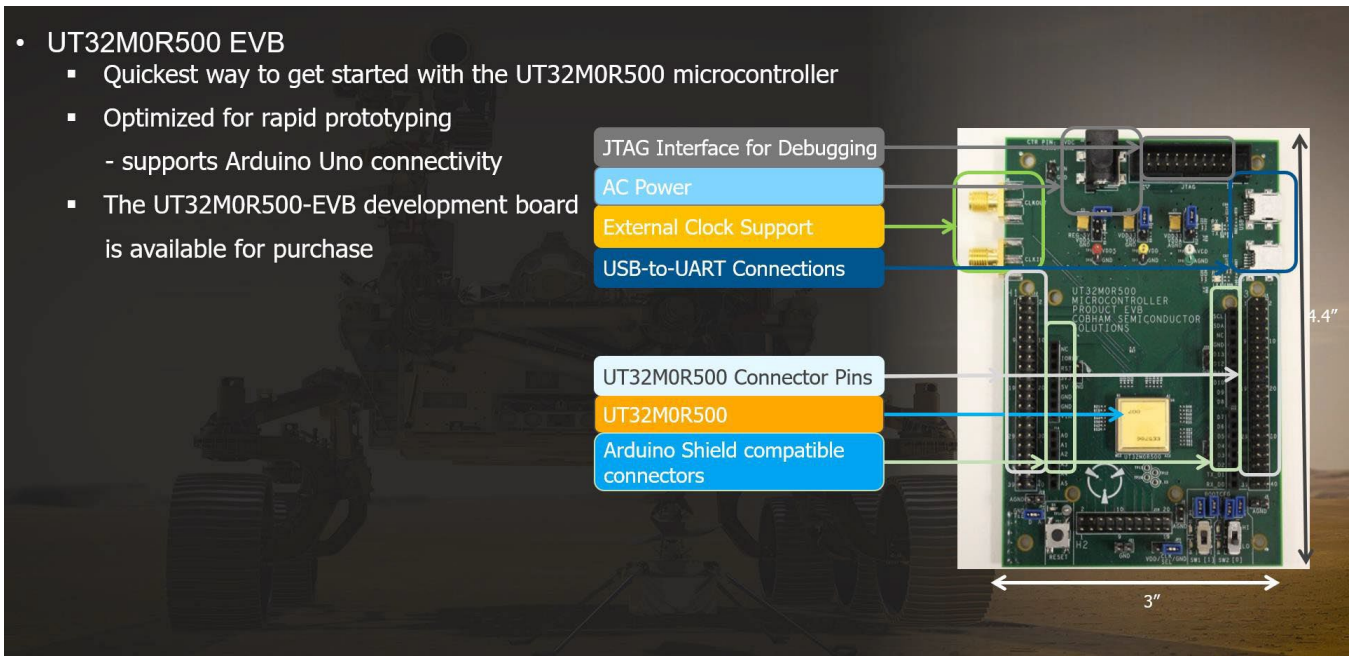
The Frontgrade SDK Library Directory Structure includes:

- Examples folder has example code for configuring and testing all the different peripherals.
- StdPeriphLib\inc folder has header files for defining the device driver API's.
- StdPeriphLib\src folder has software programs for all device drivers.
- UT32M0R500_SpecificARM directory has ARM specific files, i.e., the ARM Cortex M0+ startup file.



UT32M0R500 Evaluation Board

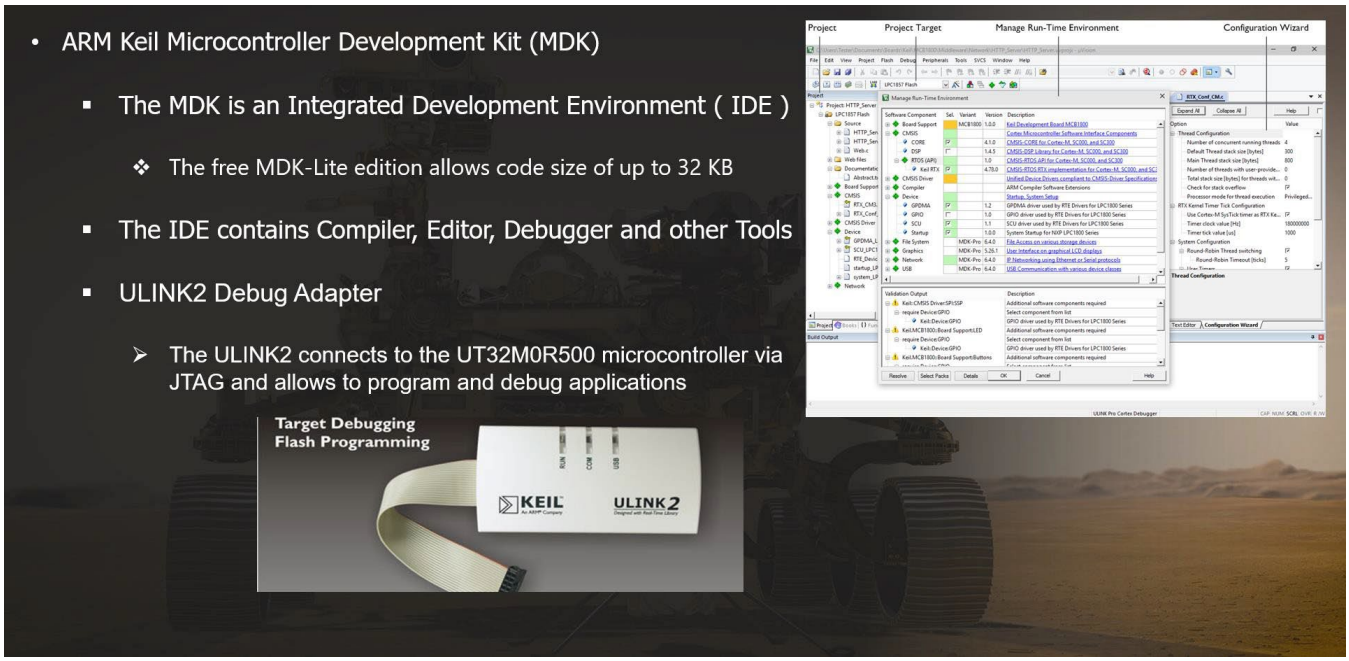
UT32M0R500 EVB allows for the quickest way to get started with the UT32M0R500 microcontroller. The EVB is optimized for rapid prototyping and supports Arduino Uno connectivity. The UT32M0R500-EVB development board is available for purchase.



Arm Keil Microcontroller Development Kit (MDK)

The ARM Keil Microcontroller Development Kit **(MDK)** is an Integrated Development Environment **(IDE).** The free MDK-Lite edition allows code size of up to 32 KB. The IDE contains Compiler, Editor, Debugger and other Tools.

ULINK2 Debug Adapter connects to the UT32M0R500 microcontroller via JTAG and allows to program and debug applications.

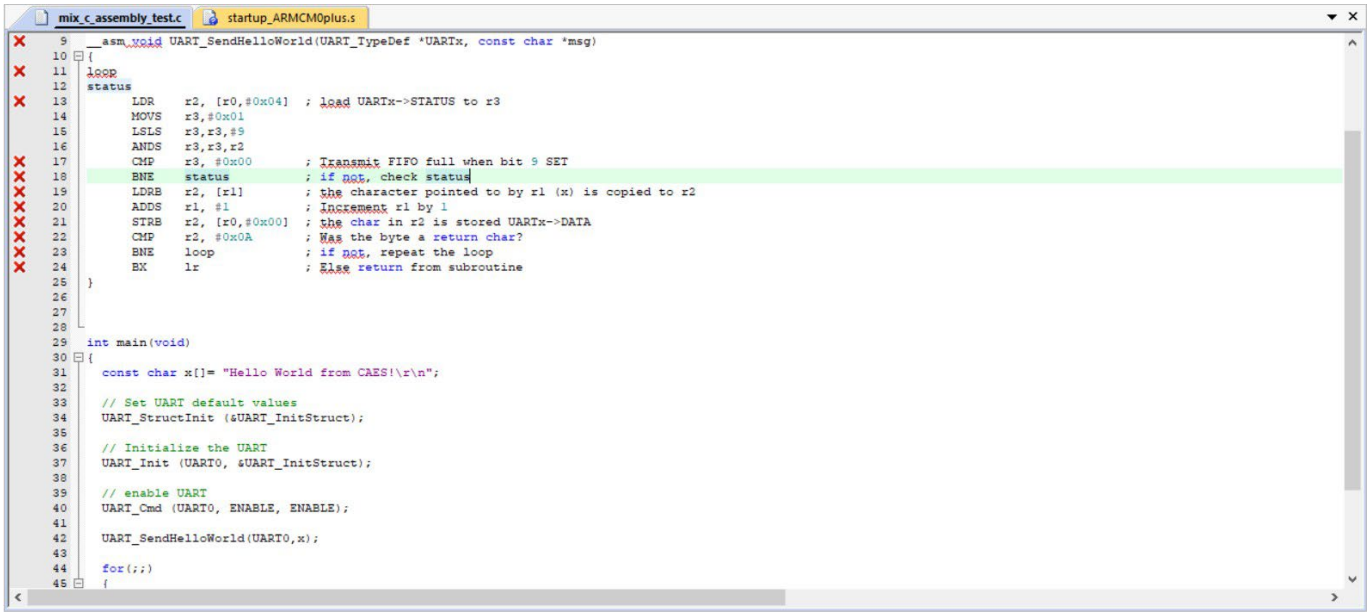


Application

The app note focuses on going through the application, and for details on how to create a project using **the Keil ARM** development tools, refer to the app note: **ApNote_UT32M0R500_Creating_Projects,** which can be downloaded from Frontgrade website.

The main program is written in C, but an assembly subroutine performs the operation of sending the “hello world from Frontgrade!” string to a **Terminal.** Most embedded systems are written in **C** with assembly language used only for critical-time tasks. This is because writing in **C** is much faster when compare to assembly language.

The main program contains one variable, which is a char array with the “hello world from Frontgrade!” message; the API function **UART_StructInit** initializes the UART structure to default values, **UART_Init** initializes the UART, and **UART_Cmd** enables the UART. Finally, the **UART_SendHelloWorld** subroutine sends one character at a time to the Terminal.



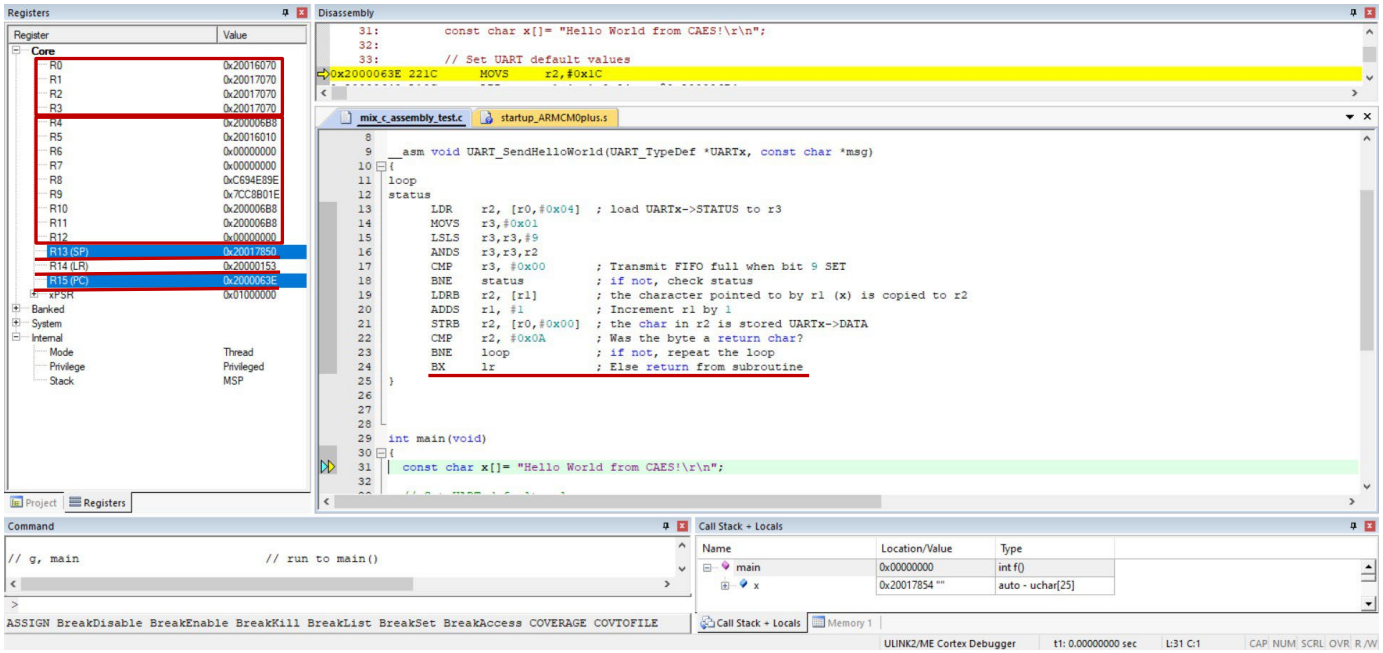
The window on the left shows the core registers.

When a function calls a subroutine, it places the return address in the link register **lr.** The arguments are passed in registers **r0** through **r3,** starting with **r0**. If there are more than 4 arguments, they are passed on the stack.

R0 through **r3** can be used for temporary storage if they are not used for arguments.

Registers **r4** through **r11** must be preserve by a subroutine. If any must be used, they must be saved first and restored before returning. This can be done by pushing to and popping them from the stack.

The **bx lr** instruction will reload the **pc** with the return address value from the **lr.** If the function returns a value, it will be pass through register **r0.**



Resources

UT32M0R500 Reference Manual: https://caes.com/sites/default/files/documents/Functional-Manual- UT32M0R500.pdf

UT32M0R500 Datasheet: https://caes.com/sites/default/files/documents/Datasheet-UT32M0R500.pdf

Cortex M0+ Technical Reference Manual: https://documentation-service.arm.com/static/60411750ee937942ba301773

Cortex M0+ Generic User Guide: https://documentation-service.arm.com/static/5f04abc8dbdee951c1cdc9f7

Cortex M0+ Processor Overview: https://developer.arm.com/Processors/Cortex-M0-Plus

UT32M0R500 App Notes: https://caes.com/product/ut32m0r500#downloads

Revision History

Date	Revision #	Author	Change Description	Page #
12/2/22	1.0.0	JA	Initial Release.	

Datasheet Definitions

	Definition
Advanced Datasheet	Frontgrade reserves the right to make changes to any products and services described herein at any time without notice. The product is still in the development stage and the **datasheet is subject to change**. Specifications can be **TBD** and the part package and pinout are **not final.**
Preliminary Datasheet	Frontgrade reserves the right to make changes to any products and services described herein at any time without notice. The product is in the characterization stage and prototypes are available.
Datasheet	Product is in production and any changes to the product and services described herein will follow a formal customer notification process for form, fit or function changes.