Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

PRODUCT NAME	MANUFACTURER PART NUMBER	SMD #	DEVICE TYPE	INTERNAL PIC NUMBER
Arm Cortex M0+	UT32M0R500	5962-17212		QS30

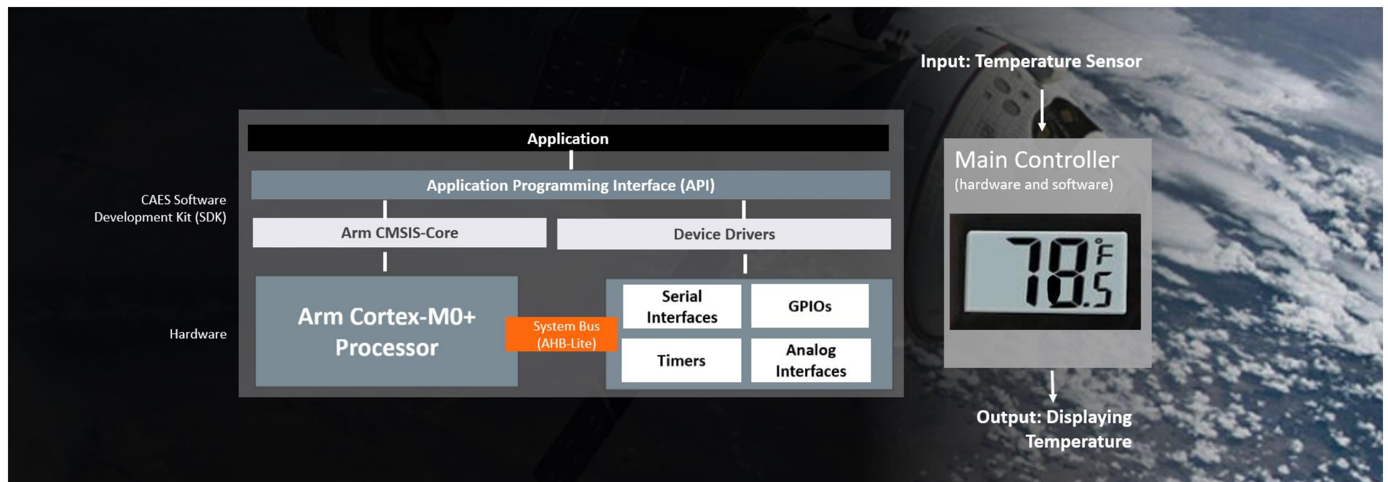
Table 1: Cross Reference of Applicable Products

1.0 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.



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

2.0 Cortex M0+ Processor Architecture

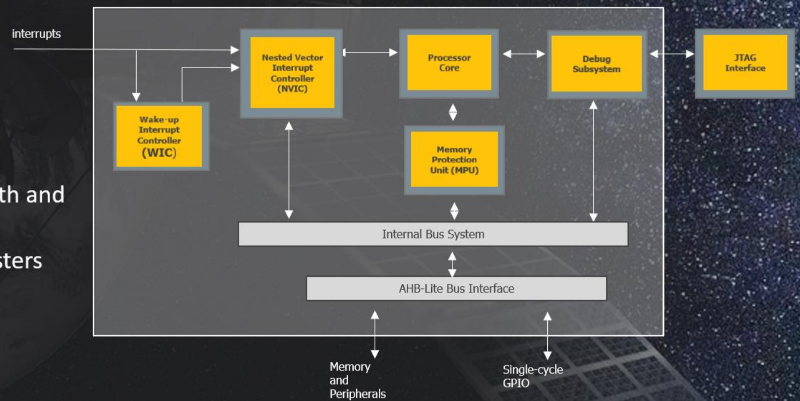
Overview

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.

- **32-bit RISC processor**
- **Von Neumann Architecture**
  - Single bus for accessing code and data.
- **Instruction Set Architecture (ISA)**
  - based on the Arm v6-M architecture, which freely intermixes 32-bit instructions with 16-bit instructions in a program.
- **Processor core**
  - contains internal registers, ALU, data path and control logic.
  - Internal Registers include 16 32-bit registers for both general and special usage.



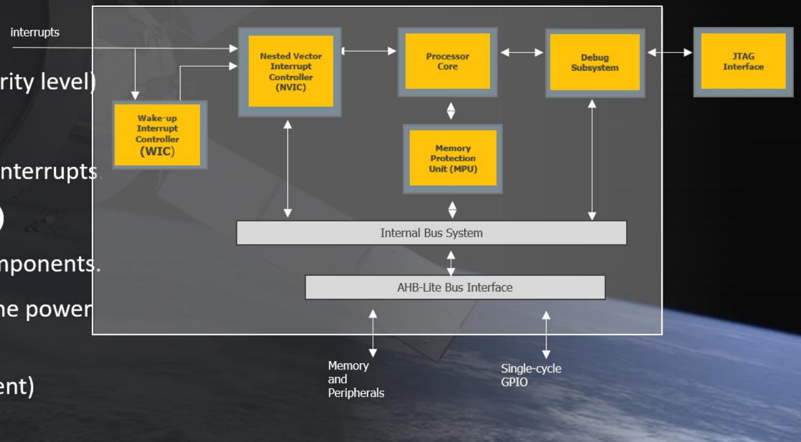
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

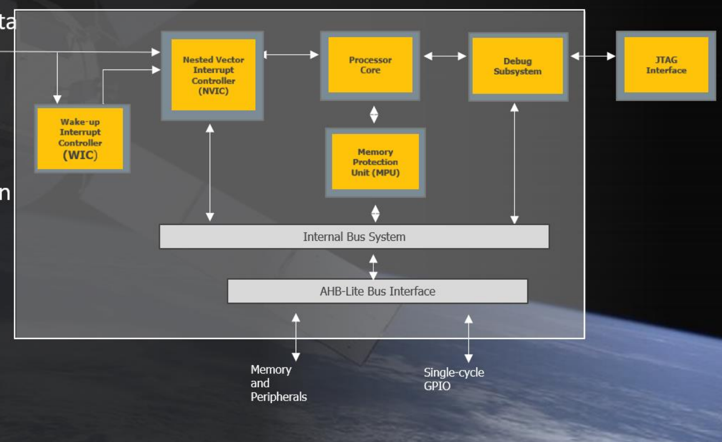
The bus interconnect is a 32-bit AMBA-3 AHB-lite interface for integrating memory and peripherals.

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.



APPLICATION NOTE

Version 1.0.0

12/2/2022

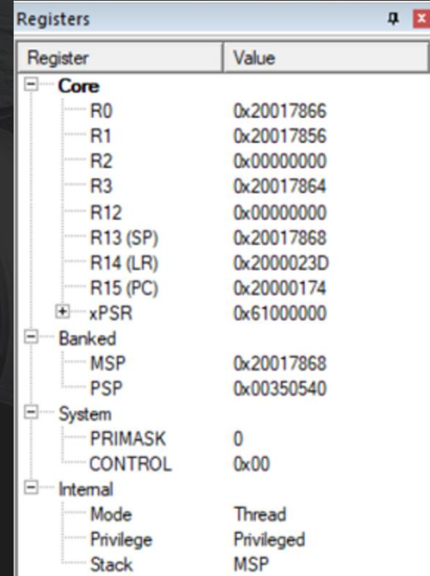
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **General-purpose Registers (R0-R12)**
  - The general-purpose registers have no special use
  - Most instructions can specify a general-purpose register
- **Stack Pointer (SP) Register 13**
  - Main Stack Pointer (MSP)
  - MSP is used in applications that require privilege access
  - Handler Mode always uses MSP
  - Process Stack Pointer (PSP)
  - Thread Mode can use either MSP or PSP
  - Stack Pointer (SP) is 4-byte aligned



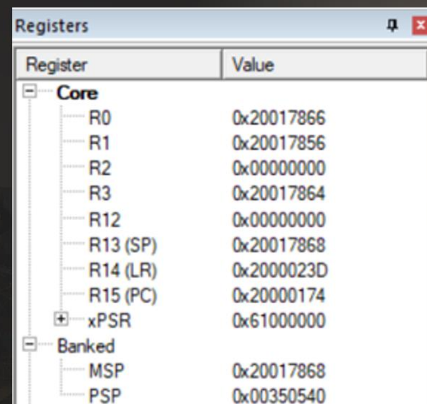
Registers

Register	Value
Core	
R0	0x20017866
R1	0x20017856
R2	0x00000000
R3	0x20017864
R12	0x00000000
R13 (SP)	0x20017868
R14 (LR)	0x2000023D
R15 (PC)	0x20000174
xPSR	0x61000000
Banked	
MSP	0x20017868
PSP	0x00350540
System	
PRIMASK	0
CONTROL	0x00
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

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 Counter (PC) Register 15**
  - PC holds the address of the current instruction that is fetched
  - Instructions are either 32-bit or 16-bit
- **Link Register (LK) Register 14**
  - The LR receives the return address from PC when a Branch and Link (BL) or Branch and Link with Exchange (BLX) instruction is executed
  - The LR is also used for exception return



Registers

Register	Value
Core	
R0	0x20017866
R1	0x20017856
R2	0x00000000
R3	0x20017864
R12	0x00000000
R13 (SP)	0x20017868
R14 (LR)	0x2000023D
R15 (PC)	0x20000174
xPSR	0x61000000
Banked	
MSP	0x20017868
PSP	0x00350540

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **Special Registers: Program Status Register (PSR)**
  - The Program Status Register (PSR) combines the Application Program Status Register (APSR), Interrupt Program Status Register (IPSR) and the Execution Program Status Register (EPSR)
  - **Application Program Status Register (APSR)**
    - **N**: negative flag- set to 1 if the result from the ALU is negative
    - **Z**: zero flag- set to 1 if the result from the ALU is zero
    - **C**: carry flag- set to 1 if an unsigned overflow occurs
    - **V**: overflow flag- set to 1 if a signed overflow occurs
  - **Interrupt program Status Register (IPSR)**
    - **ISR Number**: current ISR number
  - **Execution program Status Register (IPSR)**
    - **T**: Thumb State

BIT #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSR	N	Z	C	V				T																			ISR Number					

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

- **Special Registers: Interrupt Mask Register (PRIMASK)**
  - Bit 0: set to 1 disables all interrupt
  - Access using CPS, MSR and MSR instructions

BIT #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PRIMASK																																P

- **Special Registers: Control Register (CONTROL)**
  - Bit 0: nPRI flag
    - Determines whether thread mode is privileged (0) unprivileged (1)
  - Bit 1: SPSEL flag
    - Selects SP when in thread mode MSP (0) or PSP (1)

BIT #	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONTROL																															S	P

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **ARMv6-M Architecture Profile**
  - ARM instruction set supports 32-bit instructions.
  - Thumb-1 instruction set supports 16-bit instructions
  - Thumb-2 freely mixes 32-bit instructions and 16-bit instructions
  - ARMv6-M architecture 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				

CAES PIONEERING ADVANCED ELECTRONICS

4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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 Assembly Syntax**

Label
Mnemonic Operand1, Operand2,… ; Comments

- **Labels** are symbolic representations of addresses
  - labels are used to mark specific addresses to refer to from other parts of the code
- **Mnemonic** is the name of the instruction
  - Most data processing instructions can optionally update the condition flags according to the result of the operation
- **operand1** is the destination register
- **Operand2** is the source register
- **";"** are comments

Example:

```
__asm void UART_SendHelloWorld (UART_TypeDef *UARTx, const char *msg)
{
status
0x20000164: LDR   r2, [r0,#0x04] ; load UARTx->STATUS to r3
0x20000166: MOVS  r3,#0x01
0x20000168: LSLS  r3,r3,#9
0x2000016a: ANDS  r3,r3,r2
0x2000016c: CMP   r3, #0x00 ; Transmit FIFO full bit set
0x2000016e: BNE   status     ; if not, jmp to 0x20000164
```

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **Brach Instructions**
- **ARMv6-M only supports thumb execution.**
  - **B**
  - **BL** branch and link (immediate) calls a subroutine at a PC relative address
  - **BLX** calls a subroutine at an address and instruction set

  Specified by a register
  - **BX** cause a branch to an address and instruction set specified by a register

Function Call

Return from Function

CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- The Arm Cortex-M0+ Memory Map describes the organization of the processor's address space. The address space contains the regions of Armv6-M architecture
- The 32bit system has 4GB of memory space
- It is separated into regions with different functionality

Address	Region	Description
0xE010_0000	System	System ROM, MTB, etc.
0xE000_0000	Private Peripheral Bus	NVIC, SCS, etc.
	Reserved	
0x4002_0000	AHB Peripherals	On-chip peripherals
0x4000_0000	APB Peripherals	
	Reserved	
0x2000_0000	SRAM	96KB of progam code and data
	Reserved	
0x0100_0000	NOR Flash	NOR Flash 64KB window
	Reserved	
0x0000_0000	Boot ROM	32KB Boot ROM

CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

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.

- The bootloader is stored in on-chip ROM
- It is programmed into the chip at manufacture time
- The bootloader copies an executable image from NOR Flash into SRAM and jumps to the image.

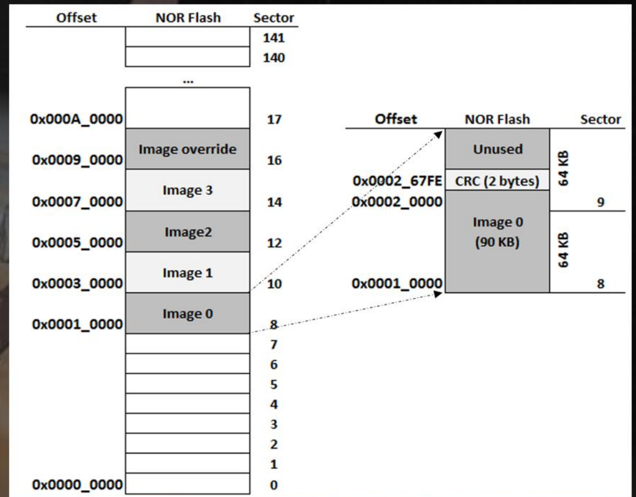
Boot mode selection pins		Boot Mode	Description
BOOTCFG1	BOOTCFG0		
0	0	0	Load image from internal Flash memory into SRAM and execute
0	1	1	Reserved
1	0	2	Load/Update image over UART0 into flash (reset required)
1	1	3	Load/Update image over CAN0 into flash (reset required)

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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).

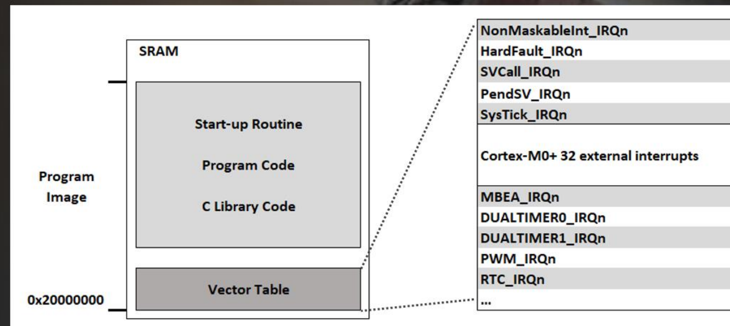


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- the total RAM memory is 96KB for both code and data.
- The program image contains:
  - Vector Table
  - Start-up routing
  - Application code and data
  - C library functions



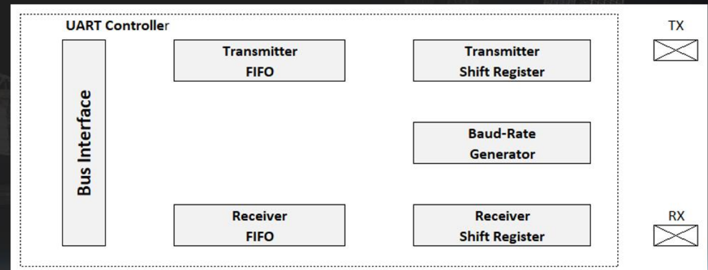
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

3.0 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().**

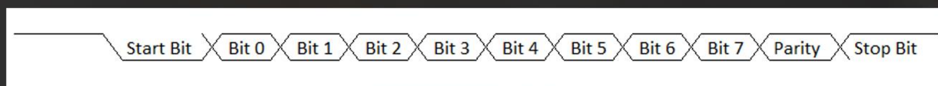
- First UART was designed by Gordon Bell of DEC in 1960s
- Separate transmit and receive wires
- UARTs implements asynchronous serial communication from parallel data (sender does not have to send a clock signal)
- Used for serial communication with buffering FIFO queues to speed communications
- The UT32M0R500 uses the UART for the serial console, but it can be used for other applications



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

UART Message Format

1. Start bit, data starts by driving the tx line low for one clock cycle.
2. Data, in the next 8 clock cycles, the transmitter sends 8 bits sequentially (parity is for transfer reliability, but it is optional).
3. Stop bit, transmission ends by pulling the tx line high.

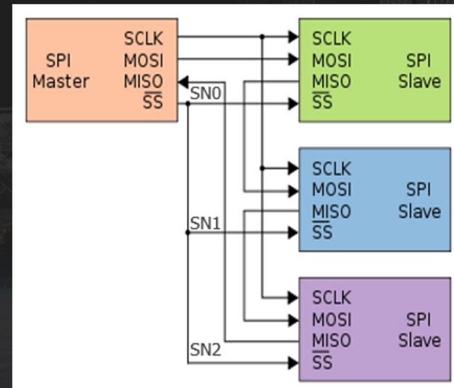


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

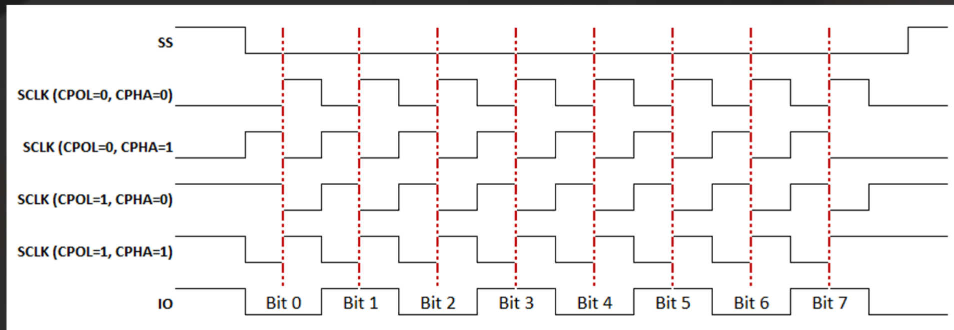
- Protocol introduced by Motorola in late 1980s
- Synchronous serial communication
  - Master and slave operate from the same clock
- Full duplex serial communication
  - Data is transmitted and received by using separate lines
- The UT32M0R500 SPI supports Master Mode only
- 4 line protocol
  - **SCLK** - 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



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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

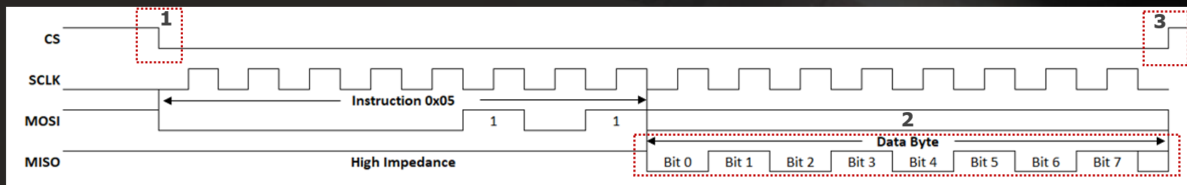


Clock Polarity and Phase		SPI Mode	Data Sampling
CPOL	CPHA		
0	0	0	Leading rising edge of the clock
0	1	1	Trailing falling edge of the clock
1	0	2	Leading falling edge of the clock
1	1	3	Trailing rising edge of the clock

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.

SPI Message Format

1. Master selects the corresponding slave by pulling SS line low
2. The slave transmits data on the MISO line one bit per clock
3. The master receives the data bit by bit and ends the transaction by setting the SS line high



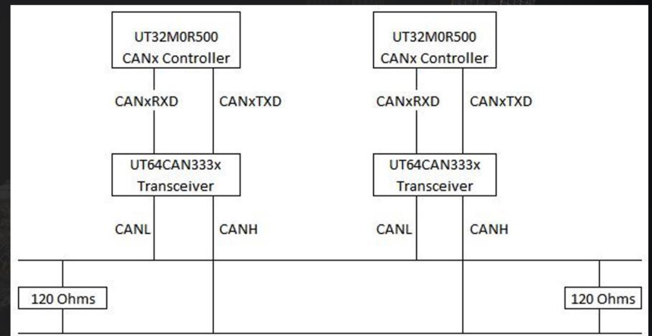
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- Developed by Bosch Corporation for automobiles
- The CAN protocol is part of the ISO 11989 standard
- Max speed: 1 Mbit/s
- Master/Slave half-duplex communication
- Nodes have unique address bits
- 11 or 29 bit addresses to identify devices (nodes)
- The CAN system consists of the bus with CANH and CANL wires terminated with 120 Ohm resistors, the UT64CAN333x transceiver (recommended), and the UT32M0R500 CAN controller



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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

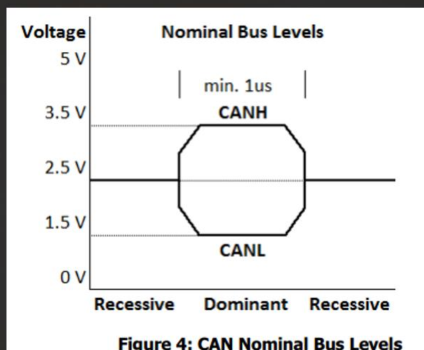


Figure 4: CAN Nominal Bus Levels

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.

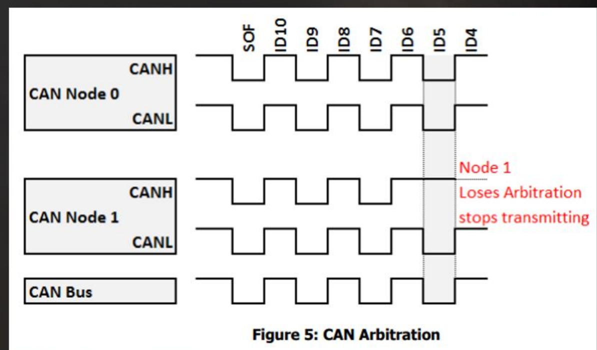
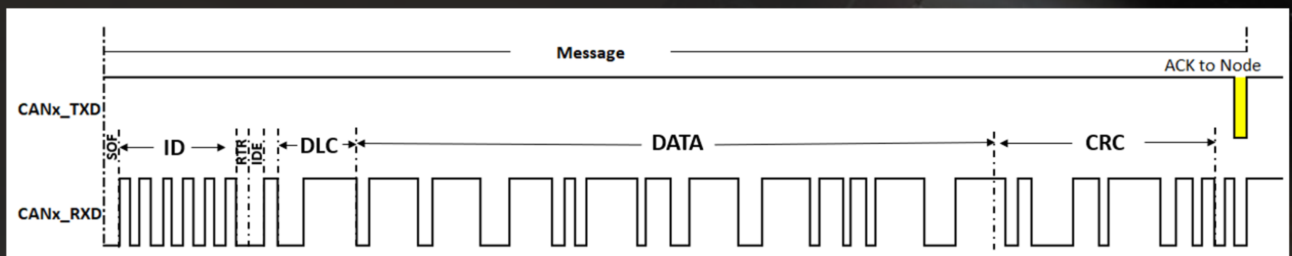


Figure 5: CAN Arbitration

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

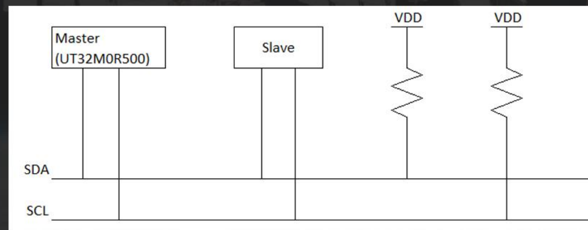


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- Protocol introduced by Philips Semiconductor
- Standard mode: 100 kbit/s
- Full speed: 400 kbit/s
- Fast mode: 1 Mbit/s
- Master/Slave half-duplex communication
- Slaves have unique address bits
- 112 devices addressable with 7 bit addresses
- 1008 devices with 10 bit addresses
- Two data lines –serial clock (SCL) and serial data (SDA)

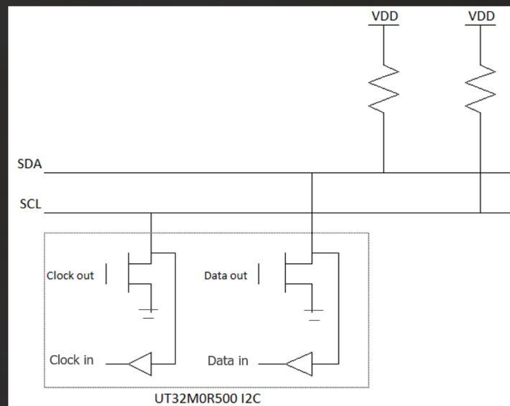


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

I2C Bus Connections

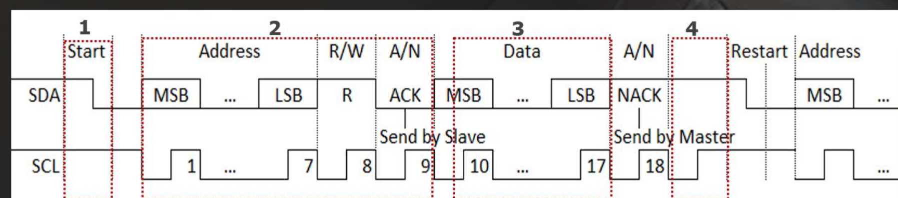
Bus is typically controlled by master device with slaves responding when addressed.
Resistors pull up lines to VDD
Open-drain transistors pull lines down to ground
Master generates SCL clock signal
- Can range from 100 KHz to 1 MHz



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.

I2C Message Format

Bus Message-oriented data transfer with four parts:
1. Start condition
2. Master generates SCL clock signal
   - Address
   - Command (read or write)
   - Acknowledgement by receiver
3. Data fields
   - Data byte
   - Acknowledgement by receiver
4. Stop condition



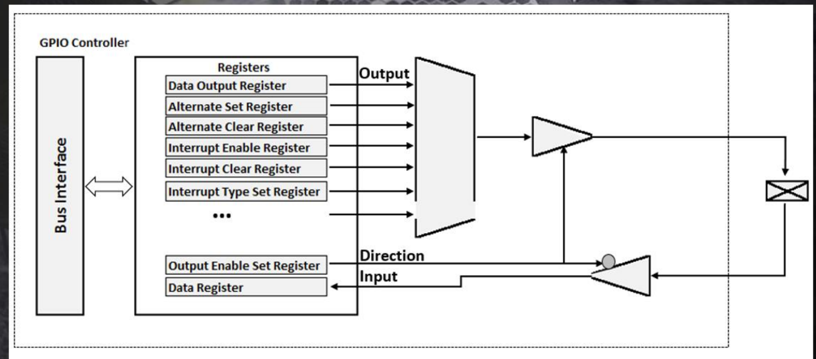
CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

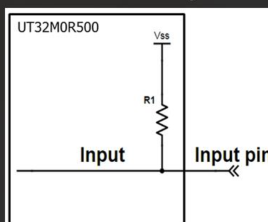
- 48x bi-directional GPIOs
- 18 dedicated GPIO pins: GPIO0-15, GPIO30-31
  - Dedicated GPIO are initialized as inputs
- GPIO with alternate functions are initialized to use the alternate function
- The GPIOs are configured in three banks of 16 pins each
  - Bank 0[15:0] = GPIO[15:0]
  - Bank 1[15:0] = GPIO[31:16]
  - Bank 2[15:0] = GPIO[47:32]

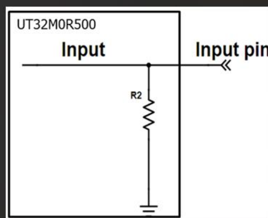


Inputs are in either pull-up or pull-down mode to ensure the input pins are either 1 or 0 when an external circuit doesn't drive the pin. **Outputs** are in either push-pull mode or open-drain mode. In push-pull mode, the output is either 1 or 0 and in open-drain, the output is pull low to ground or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

Input Pull-up Pull-down Mode

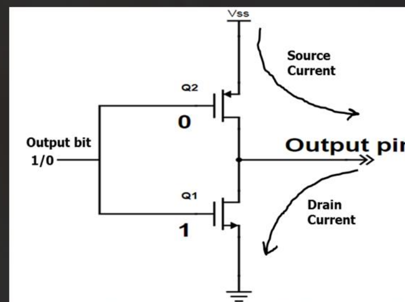
pull-up, pull-down ensures the input pin is either 1 or 0 when an external circuit doesn't drive the pin





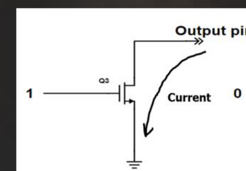
Output Push-Pull Mode

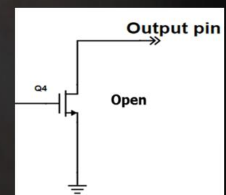
With a push-pull GPIO, a transistor connects to VCC or GND to drive a signal high or low. When the output goes low, the signal is actively "pulled" to ground, and when the output goes high it is actively "pushed" to VCC



Output Open-drain Mode

When the output goes low, the signal is actively "pulled" to ground, and when the output goes high, it is left floating





CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

APPLICATION NOTE

Version 1.0.0

12/2/2022

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

Timers

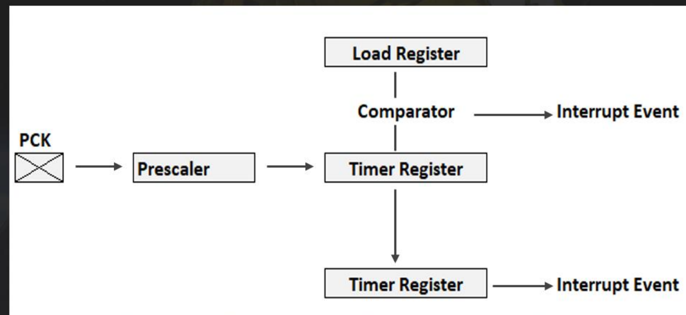
A **Timer** is a counter with a tick as the basic time unit and generates an interrupt when it reaches a predefined value. It has four main components: pre-scaler, compare register, timer register and capture register. A pre-scaler divides the system clock by a predefined value either 1, 16, 256 and 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.

Pre-scaler
Clock source as input
Divides input frequency by 1, 16, 256
Outputs a tick to the other components

Timer Register
Increases or decreases at a fix tick interval

Compare Register
Preloaded with desire value to compare against timer register
An interrupt is generated when values match

Compare Register
Loads the value from timer register at certain events
An interrupt is generated at the occurrence of certain events.

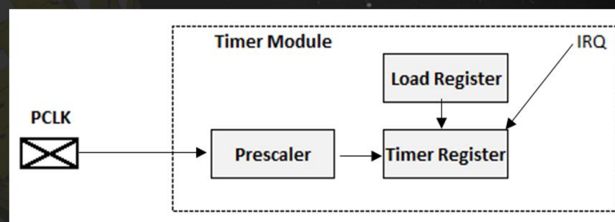


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- 2 independent programmable timer modules
- Each timer can be either 16 or 32-bit.
- Each timer supports a clock pre-scaler value of 1, 16, or 256
- Each timer module can be independently enabled or disabled
- Each module supports three different modes of operation:
    - free-running, periodic timer and one-shot timer.

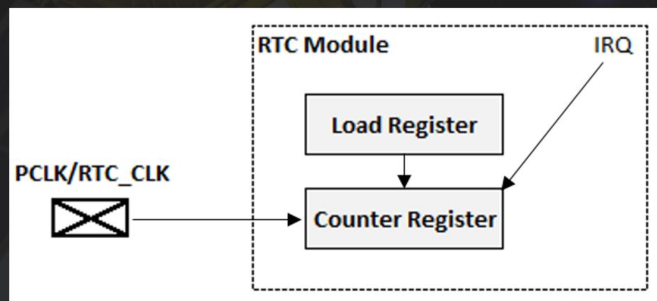


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- 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
- counter wraps at matched period specified load register value



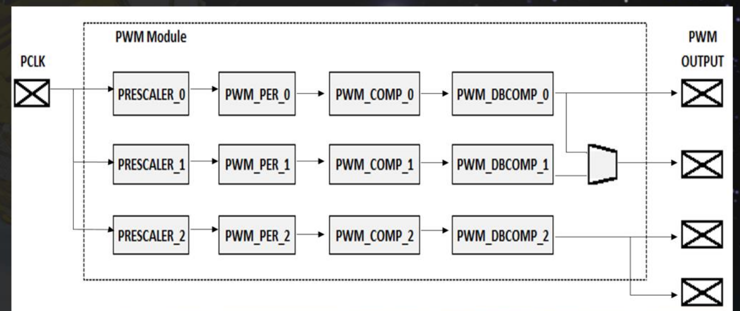
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.



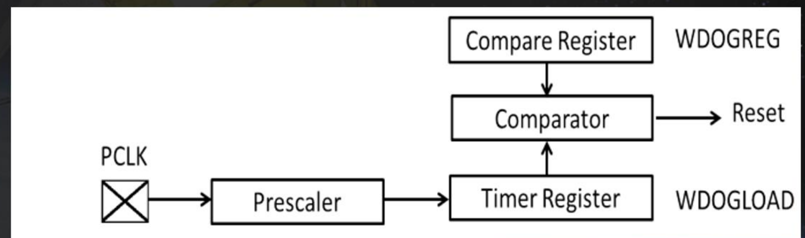
CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- The watchdog begins counting down from the value loaded in the Load register when WDOGCLKEN = 1 and Interrupt Enable = 1
- 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 WDOGREG will be asserted



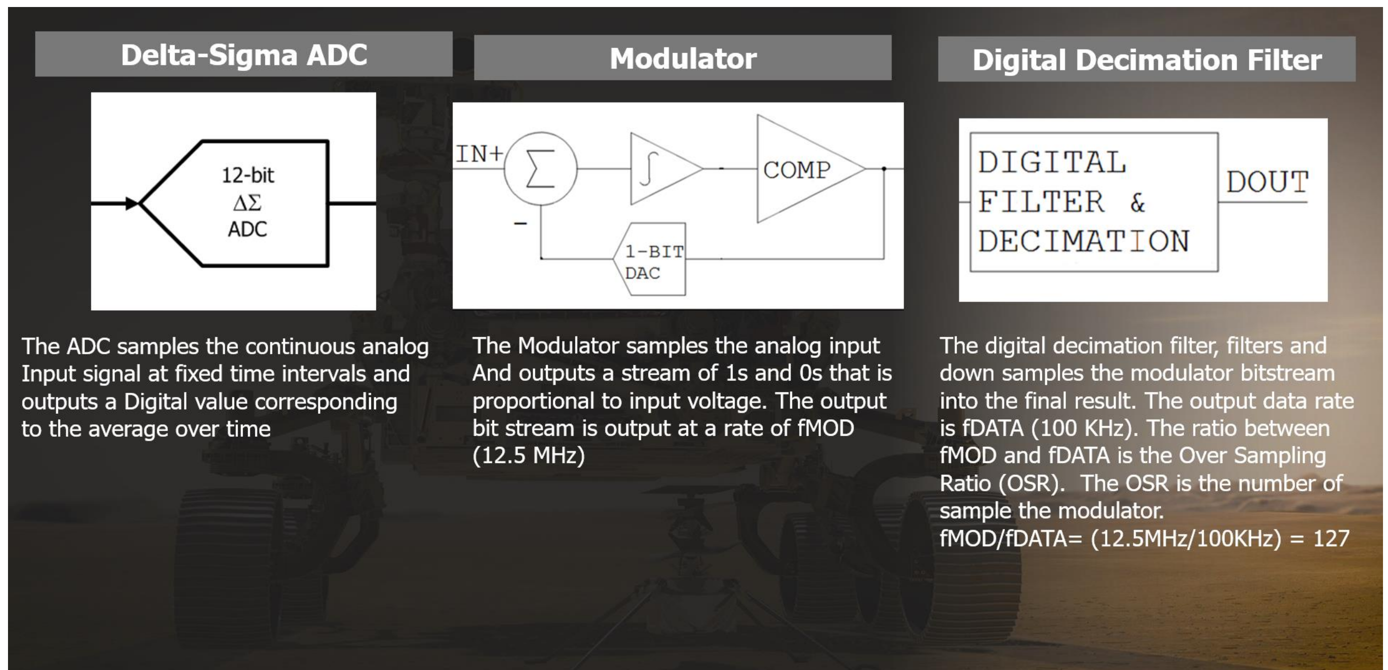
CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.



CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

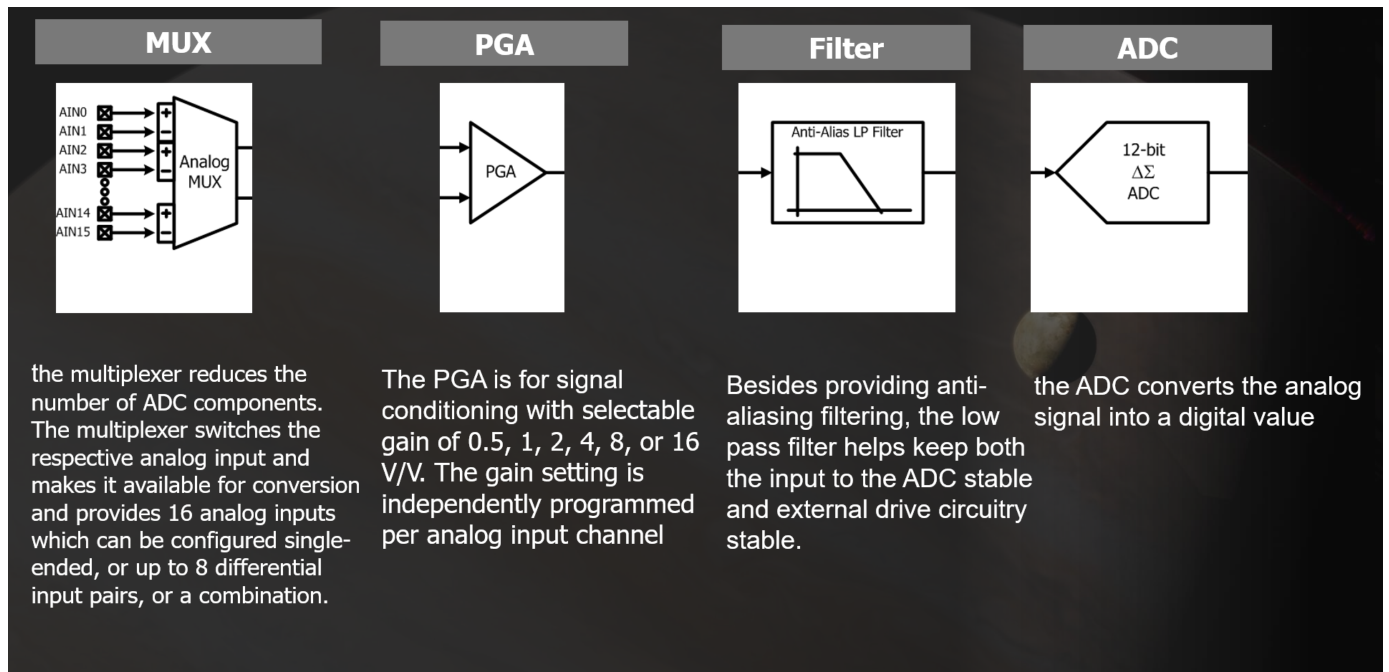
ADC: Analog Interface

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.



4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

APPLICATION NOTE

Version 1.0.0

12/2/2022

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

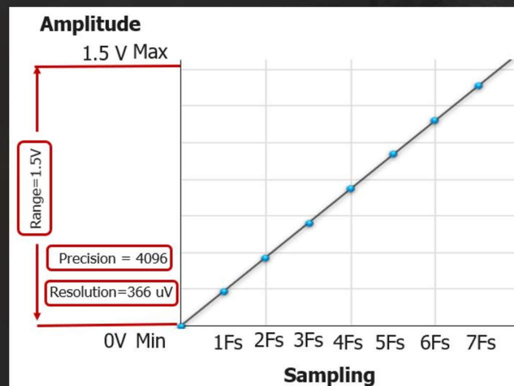
In terms of **sampling**, the ADC defines the sampling frequency, number of samples and frequency resolution. Sampling frequency is defined as the minimum frequency that signals can be sampled without violating the Nyquist theorem, which states that if we sample at 2Khz, the maximum signal we can represent 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.

Amplitude

- Range
  - Min to max or 1.5 V
- Precision
  - total number of discrete values or 4096
- Resolution
  - Smallest step size the ADC can measure or 366 uV

- **Sampling Frequency**

- Sampling Frequency, Fs
  - Signal of interest (1/2) Fs
- Number of samples N
- Frequency resolution Fs/N



APPLICATION NOTE

Version 1.0.0

12/2/2022

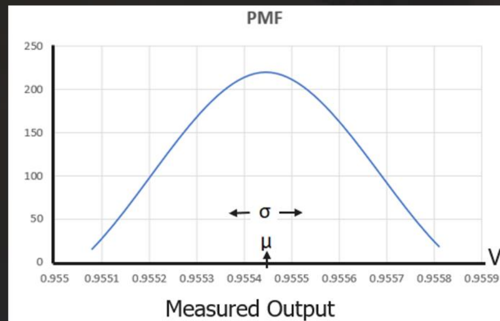
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

Noise

- The probability mass function (PMF) gives the number of times each measurement was created
- signal μ=mean (average)
- noise σ=stdev (standard deviation)
- Signal to noise μ /σ (S/N)
- Precision or effective number of bits = log2(μ/ σ)

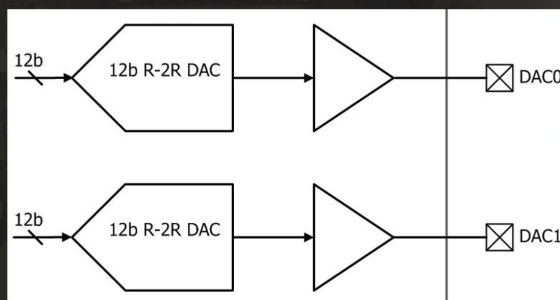


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **DAC Overview**
  - Two 12-bit Voltage-Mode R-2R DACs
  - Dual independent DAC outputs
  - Synchronous or Independent update
  - Power-down mode Soft-Reset supported by enable bit



4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

Analog Comparator

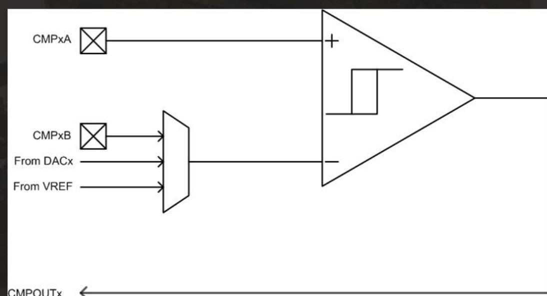
The UT32M0R500 has two High-speed **Analog Comparators** with Hysteresis on Inputs and Rail-to-Rail Input Common-Mode Range; Four selectable inputs to the negative input of each comparator.

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.

- **Analog Comparator Overview**

- Two High-speed Analog Comparators
- Hysteresis on Inputs
- Rail-to-Rail Input Common-Mode Range
- Four selectable inputs to the negative input of each comparator
- Functions as support for monitoring of analog signals
- Low Power Shutdown Mode

- **Sensing**
- the analog comparator compares two analog voltages and outputs a digital ouput.
- 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.



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

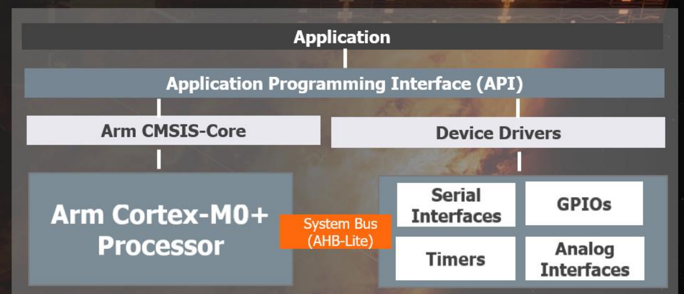
4.0 CAES Software Development Kit (SDK)

CAES 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.

- CAES Software Development Kit (SDK)
  - ARM Cortex Microcontroller Software Interface Standard (CMSIS)
  - Software Drivers
  - Application Peripheral Interface (API)
- UT32M0R500 Evaluation Board
- ARM Keil Microcontroller Development Kit (MDK)
  - The MDK is an Integrated Development Environment (IDE)
  - The IDE contains Compiler, Editor, Debugger and other Tools
- Program Application

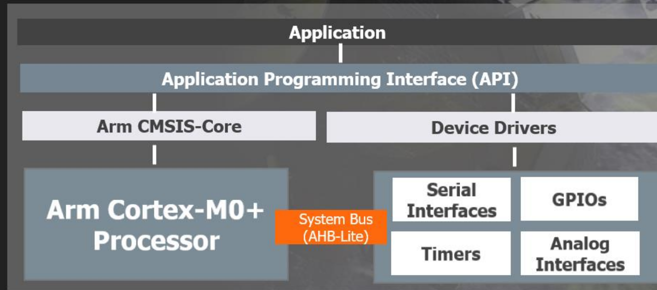


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

Cortex Microcontroller Software Interface Standard (CMSIS)

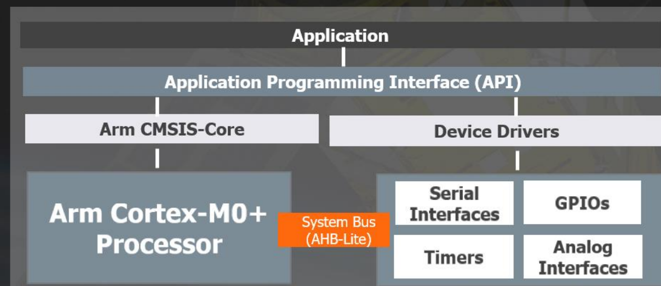
Cortex Microcontroller Software Interface Standard (CMSIS) is an ARM software interface standard. It supports the Cortex-M0+ processor and provides a standardized software interface to the processor features. **Functions** include**:** Access to NVIC, System Control Block (SCB) and System Tick Timer;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

- **CMSIS 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
  - 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.

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

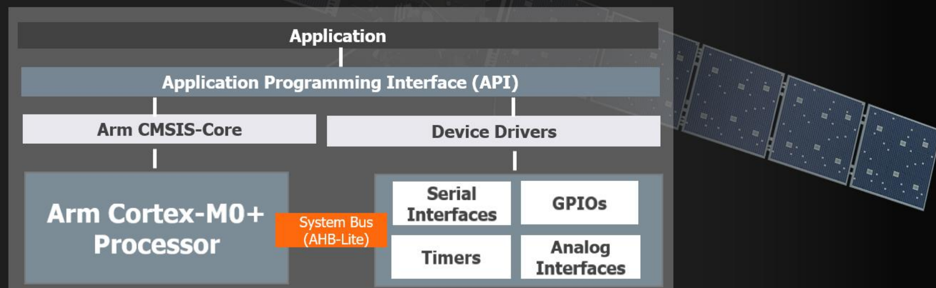


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- **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.



Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- The UT32M0R500 header file UT32M0R500.h contains:
  - Interrupt numbers (IRQn) for all exceptions and interrupts
  - Configuration of the processor and peripherals
  - data structures and the address mapping for all peripherals
  - define the peripherals as a memory pointer to data structures to access their registers

```
typedef enum IRQn
  {
  /******  Cortex-M0+ Processor Exceptions Numbers ***************************************************/

  NonMaskableInt_IRQn  = -14,   /*!<  2 Non Maskable Interrupt -- WATCHDOG      */
  HardFault_IRQn       = -13,   /*!<  3 Cortex-M0+ Hard Fault Interrupt         */
  SVCall_IRQn          =  -5,   /*!< 11 Cortex-M0+ SV Call Interrupt            */
  PendSV_IRQn          =  -2,   /*!< 14 Cortex-M0+ Pend SV Interrupt            */
  SysTick_IRQn         =  -1,   /*!< 15 Cortex-M0+ System Tick Interrupt        */

  /******  UT32M0R500 Specific Interrupt Numbers ******************************************************/

                                /*!< Cortex-M0+ supports only 32 interrupts     */

  MBEA_IRQn            =   0,   /*!< IRQ  0: MBEA */
  DUALTIMER0_IRQn      =   1,   /*!< IRQ  1: DualTimer0 */


/* APB peripherals */

#define RTC_BASE                (PERIPH_BASE + 0x0000ul)     // 0x40000000 + 0x0000 = 0x40000000
#define DUALTIMER0_BASE         (PERIPH_BASE + 0x1000ul)     // 0x40000000 + 0x1000 = 0x40001000
#define DUALTIMER1_BASE         (PERIPH_BASE + 0x2000ul)     // :
#define PWM_BASE                (PERIPH_BASE + 0x3000ul)     // :

typedef struct
  {
  __IO  uint8_t  CONTROL;               /*!< Offset: 0x00: Control Register                 (R/W) */
   ...
  } BASI_CAN;
  typedef struct
  {
  __IO  uint32_t DATA;       /*!< Offset: 0x00, 8 LSbits:  Data Register             (R/W) */
  ...
  } UART_TypeDef;

  DUALTIMER_BOTH_TypeDef *DTIMER0 = (DUALTIMER_BOTH_TypeDef *) DUALTIMER0_BASE;
  GPIO_TypeDef *GPIO2 = (GPIO_TypeDef *) GPIO2_BASE;
```

Version 1.0.0

12/2/2022

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

APPLICATION NOTE

Version 1.0.0

12/2/2022

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.

The API provides easy-to-use functions by combining the functions of both CMSIS and peripheral drivers.

API Function Examples	Description
`void DTIMER_Init (DUALTIMER_BOTH_TypeDef *DTIMERx, DTIMER_InitTypeDef *Config)`	`/*initializes the DTIMER for default operation*/`
`void DTIMER_StructInit (DTIMER_InitTypeDef *DTIMER_InitStruct)`	`/*initializes the DTIMER_InitStruct to known values*/`
`void DTIMER_Cmd (DUALTIMER_BOTH_TypeDef *DTIMERx, DTIMER_NUM Num, DTIMER_ENABLE Enable)`	`/*enables/disables the DTIMERx peripheral*/`
`void DTIMER_EnableIRQ (DUALTIMER_BOTH_TypeDef *DTIMERx, DTIMER_NUM Num)`	`/*enables/disables the DTIMERx IRQ*/`
`__STATIC_INLINE void __NVIC_EnableIRQ(IRQn_Type IRQn)`	`/*Enables a device specific interrupt in the NVIC*/`

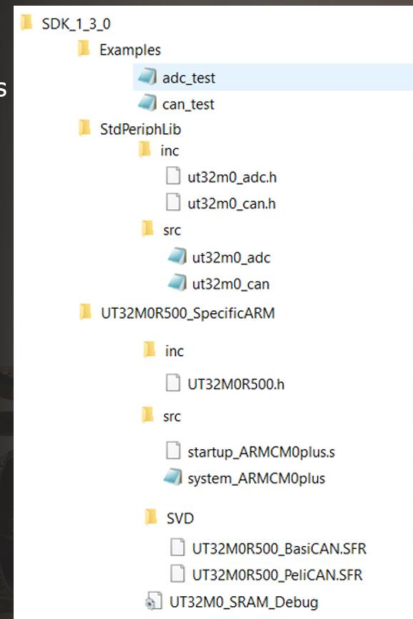
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

SDK Library Structure

The **CAES 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.

- **The CAES SDK Directory Structure**
  - \SDK_x_y_z\Examples
    - example code for configuring and testing all the different peripherals
  - \SDK_x_y_z\StdPeriphLib\inc
    - header files for defining the device driver APIs
  - \SDK_x_y_z\StdPeriphLib\src
    - software programs for all device drivers
  - \SDK_x_y_z\UT32M0R500_SpecificARM
    - ARM specific files, i.e., the ARM Cortex M0+ startup file

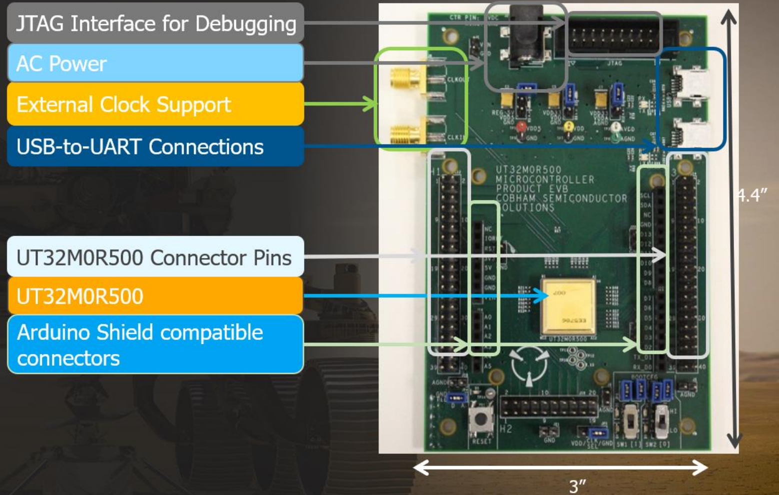


Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- UT32M0R500 EVB
  - Quickest way to get started with the UT32M0R500 microcontroller
  - Optimized for rapid prototyping
    - supports Arduino Uno connectivity
  - The UT32M0R500-EVB development board is available for purchase



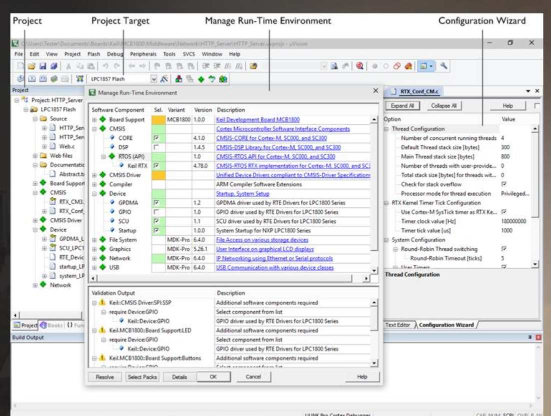
CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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.

- ARM Keil Microcontroller Development Kit (MDK)
  - The 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
    - The ULINK2 connects to the UT32M0R500 microcontroller via JTAG and allows to program and debug applications



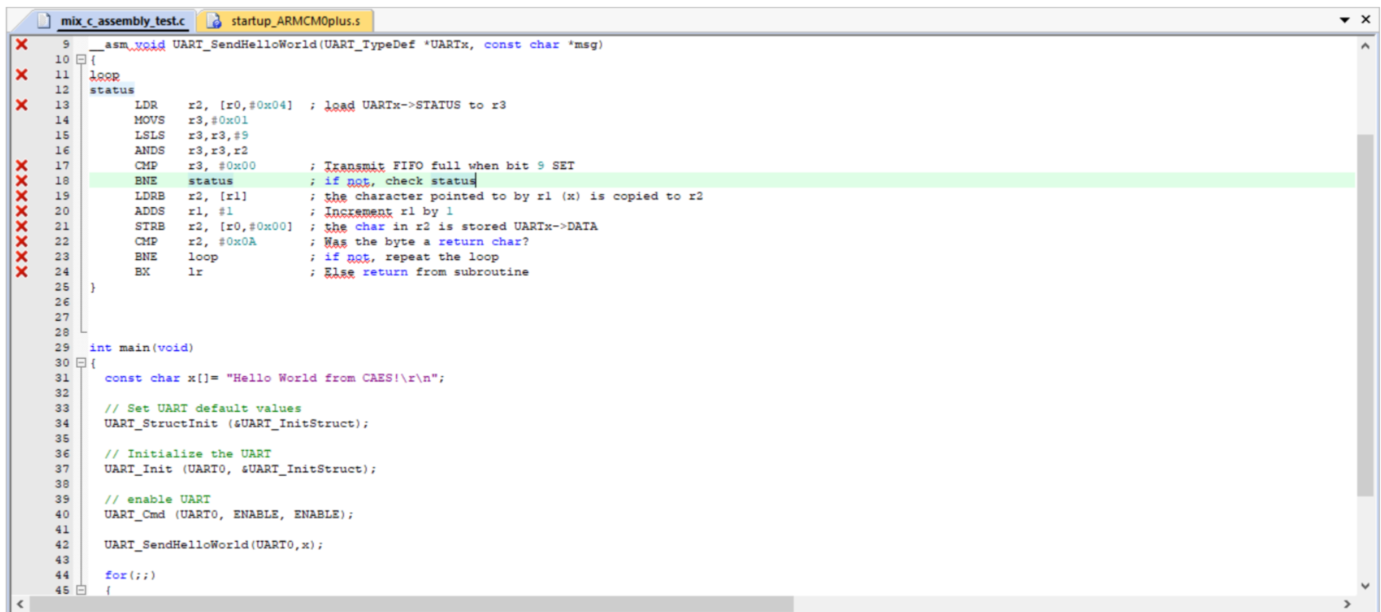
Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

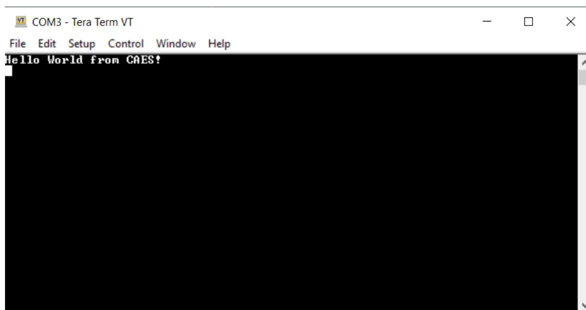
Application

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 **CAES** website.

The main program is written in C, but an assembly subroutine performs the operation of sending the "hello world from CAES!" 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 CAES!" 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.





CAES PIONEERING ADVANCED ELECTRONICS
4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

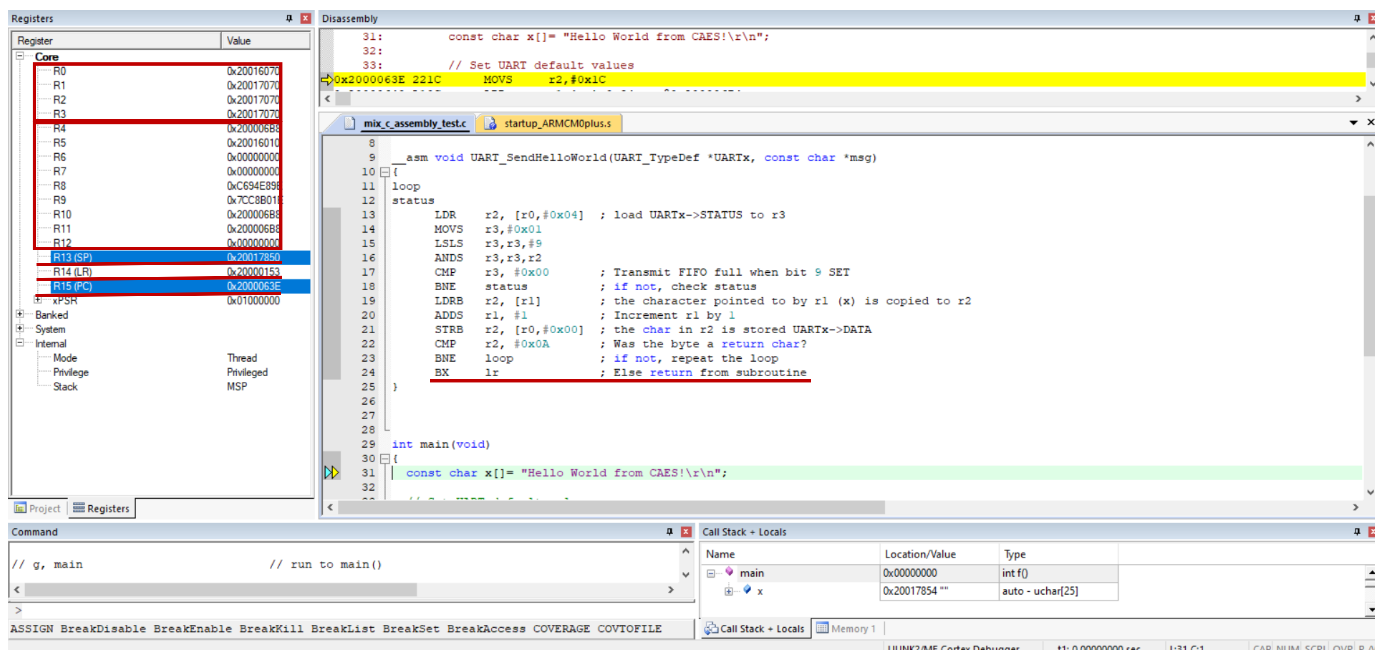
The window on the left shows the core registers.

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**.



APPLICATION NOTE

Version 1.0.0

12/2/2022

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

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

4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com

APPLICATION NOTE

Version 1.0.0

12/2/2022

Embedded Systems Fundamentals with the UT32M0R500 Microcontroller

REVISION HISTORY

Date	Rev. #	Author	Change Description
12/2/2022	1.0.0	JA	Initial Release.

The following United States (U.S.) Department of Commerce statement shall be applicable if these commodities, technology, or software are exported from the U.S.: These commodities, technology, or software were exported from the United States in accordance with the Export Administration Regulations. Diversion contrary to U.S. law is prohibited.

CAES Colorado Springs Inc. d/b/a Cobham Advanced Electronic Solutions (CAES) reserves the right to make changes to any products and services described herein at any time without notice. Consult an authorized sales representative to verify that the information in this data sheet is current before using this product. The company does not assume any responsibility or liability arising out of the application or use of any product or service described herein, except as expressly agreed to in writing; nor does the purchase, lease, or use of a product or service convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual rights of the company or of third parties.

CAES PIONEERING ADVANCED ELECTRONICS

4350 Centennial Blvd. | Colorado Springs CO 80907 USA | 1-800-645-8862 | caes.com