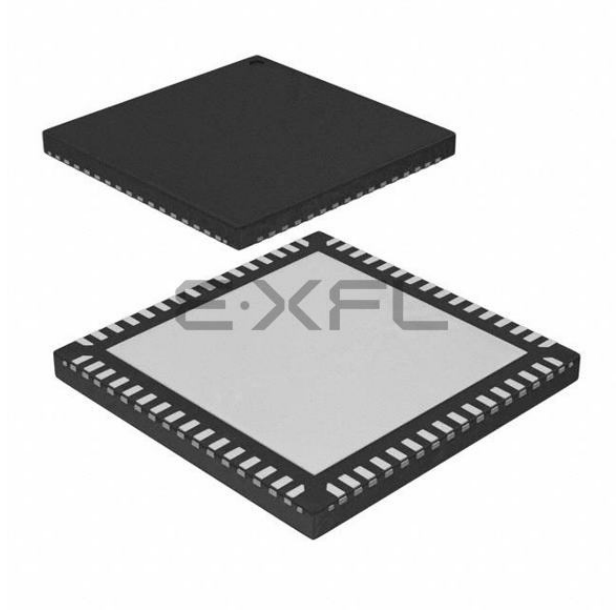


Welcome to E-XFL.COM

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

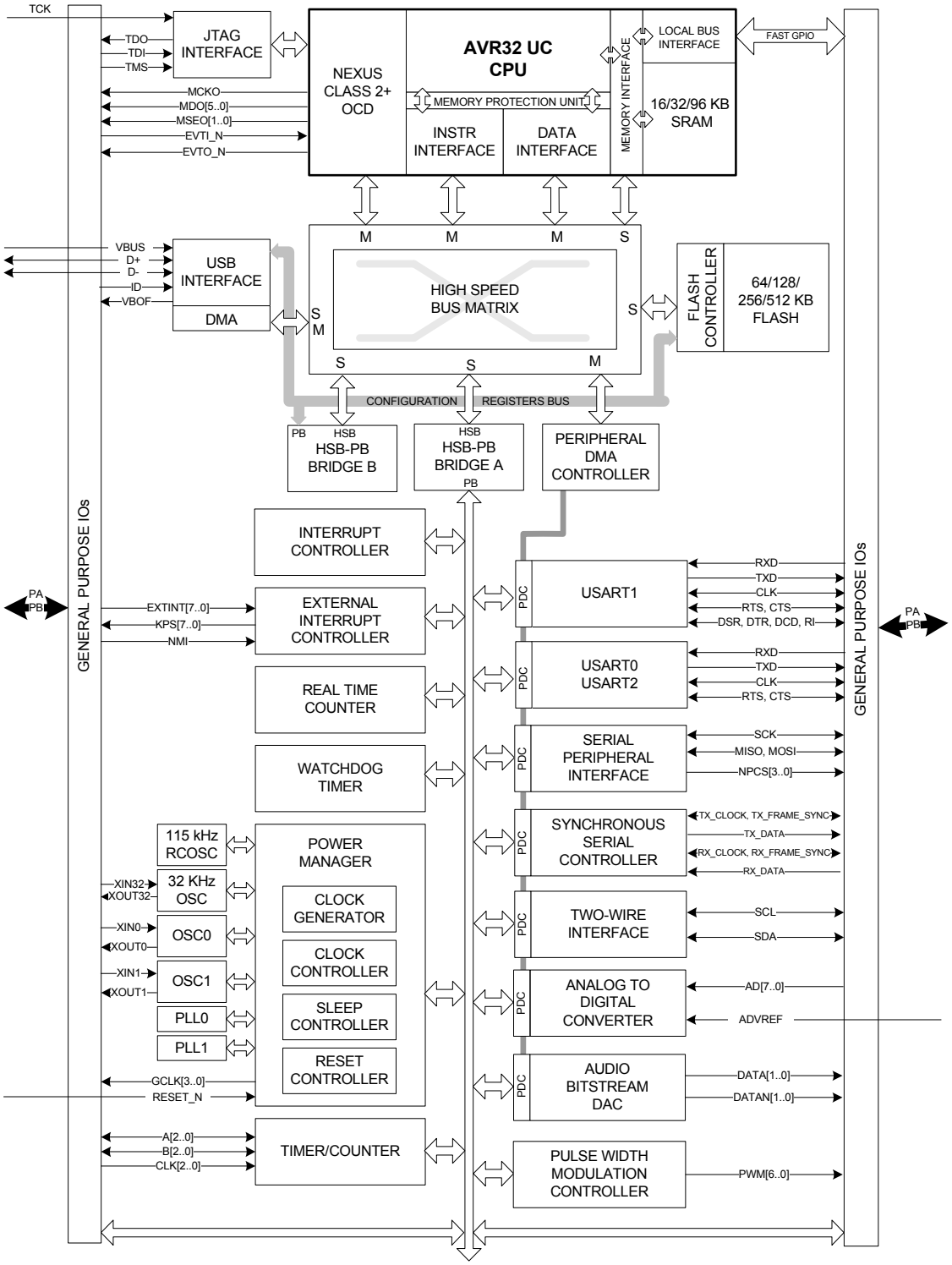
Applications of "Embedded - Microcontrollers"

Details	
Product Status	Active
Core Processor	AVR
Core Size	32-Bit Single-Core
Speed	60MHz
Connectivity	I²C, IrDA, SPI, SSC, UART/USART, USB
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	44
Program Memory Size	512KB (512K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	96K x 8
Voltage - Supply (Vcc/Vdd)	1.65V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	64-VFQFN Exposed Pad
Supplier Device Package	64-QFN (9x9)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at32uc3b0512-z2ur

2. Overview

2.1 Blockdiagram

Figure 2-1. Block diagram



3. Configuration Summary

The table below lists all AT32UC3B memory and package configurations:

Table 3-1. Configuration Summary

Feature	AT32UC3B0512	AT32UC3B0256/128/64	AT32UC3B1512	AT32UC3B1256/128/64
Flash	512 KB	256/128/64 KB	512 KB	256/128/64 KB
SRAM	96KB	32/32/16KB	96KB	32/16/16KB
GPIO	44		28	
External Interrupts	8		6	
TWI	1			
USART	3			
Peripheral DMA Channels	7			
SPI	1			
Full Speed USB	Mini-Host + Device		Device	
SSC	1		0	
Audio Bitstream DAC	1	0	1	0
Timer/Counter Channels	3			
PWM Channels	7			
Watchdog Timer	1			
Real-Time Clock Timer	1			
Power Manager	1			
Oscillators	PLL 80-240 MHz (PLL0/PLL1) Crystal Oscillators 0.4-20 MHz (OSC0) Crystal Oscillator 32 KHz (OSC32K) RC Oscillator 115 kHz (RCSYS)			
	Crystal Oscillators 0.4-20 MHz (OSC1)			
10-bit ADC number of channels	8		6	
JTAG	1			
Max Frequency	60 MHz			
Package	TQFP64, QFN64		TQFP48, QFN48	

Table 4-4. Oscillator pinout

QFP48 pin	QFP64 pin	Pad	Oscillator pin
30	39	PA18	XIN0
	41	PA28	XIN1
22	30	PA11	XIN32
31	40	PA19	XOUT0
	42	PA29	XOUT1
23	31	PA12	XOUT32

4.3 High Drive Current GPIO

Ones of GPIOs can be used to drive twice current than other GPIO capability (see Electrical Characteristics section).

Table 4-5. High Drive Current GPIO

GPIO Name
PA20
PA21
PA22
PA23

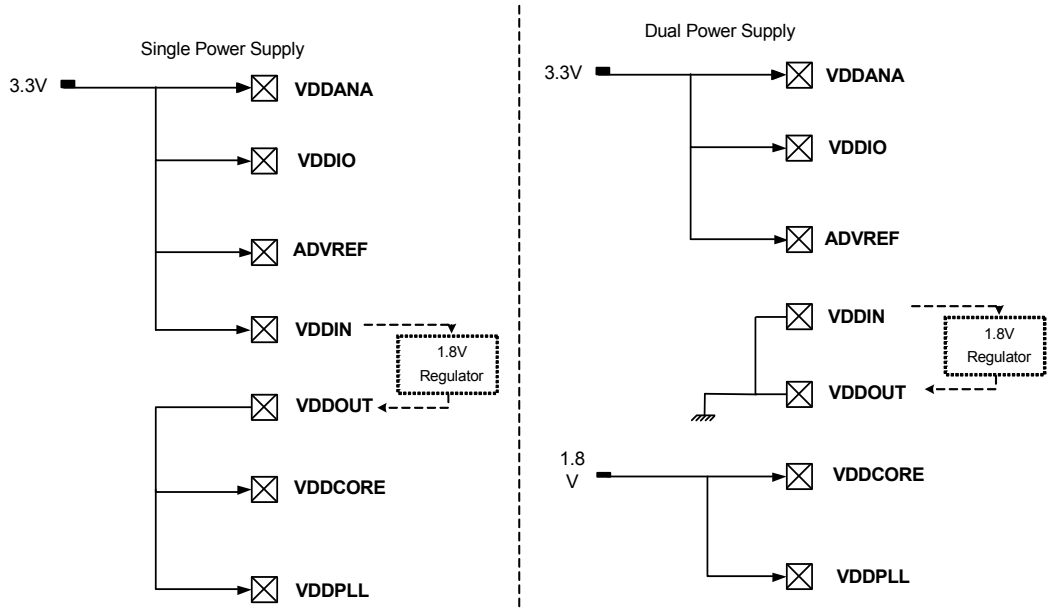
5. Signals Description

The following table gives details on the signal name classified by peripheral.

Table 5-1. Signal Description List

Signal Name	Function	Type	Active Level	Comments
Power				
VDDPLL	PLL Power Supply	Power Input		1.65V to 1.95 V
VDDCORE	Core Power Supply	Power Input		1.65V to 1.95 V
VDDIO	I/O Power Supply	Power Input		3.0V to 3.6V
VDDANA	Analog Power Supply	Power Input		3.0V to 3.6V
VDDIN	Voltage Regulator Input Supply	Power Input		3.0V to 3.6V

Figure 5-1. Power Supply



5.6.2 Voltage Regulator

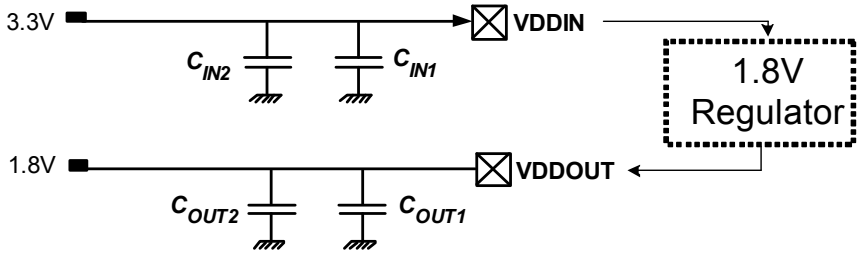
5.6.2.1 Single Power Supply

The AT32UC3B embeds a voltage regulator that converts from 3.3V to 1.8V. The regulator takes its input voltage from VDDIN, and supplies the output voltage on VDDOUT that should be externally connected to the 1.8V domains.

Adequate input supply decoupling is mandatory for VDDIN in order to improve startup stability and reduce source voltage drop. Two input decoupling capacitors must be placed close to the chip.

Adequate output supply decoupling is mandatory for VDDOUT to reduce ripple and avoid oscillations. The best way to achieve this is to use two capacitors in parallel between VDDOUT and GND as close to the chip as possible

Figure 5-2. Supply Decoupling



6. Processor and Architecture

Rev: 1.0.0.0

This chapter gives an overview of the AVR32UC CPU. AVR32UC is an implementation of the AVR32 architecture. A summary of the programming model, instruction set, and MPU is presented. For further details, see the *AVR32 Architecture Manual* and the *AVR32UC Technical Reference Manual*.

6.1 Features

- **32-bit load/store AVR32A RISC architecture**
  - **15 general-purpose 32-bit registers**
  - **32-bit Stack Pointer, Program Counter and Link Register reside in register file**
  - **Fully orthogonal instruction set**
  - **Privileged and unprivileged modes enabling efficient and secure Operating Systems**
  - **Innovative instruction set together with variable instruction length ensuring industry leading code density**
  - **DSP extention with saturating arithmetic, and a wide variety of multiply instructions**
- **3-stage pipeline allows one instruction per clock cycle for most instructions**
  - **Byte, halfword, word and double word memory access**
  - **Multiple interrupt priority levels**
- **MPU allows for operating systems with memory protection**

6.2 AVR32 Architecture

AVR32 is a high-performance 32-bit RISC microprocessor architecture, designed for cost-sensitive embedded applications, with particular emphasis on low power consumption and high code density. In addition, the instruction set architecture has been tuned to allow a variety of microarchitectures, enabling the AVR32 to be implemented as low-, mid-, or high-performance processors. AVR32 extends the AVR family into the world of 32- and 64-bit applications.

Through a quantitative approach, a large set of industry recognized benchmarks has been compiled and analyzed to achieve the best code density in its class. In addition to lowering the memory requirements, a compact code size also contributes to the core's low power characteristics. The processor supports byte and halfword data types without penalty in code size and performance.

Memory load and store operations are provided for byte, halfword, word, and double word data with automatic sign- or zero extension of halfword and byte data. The C-compiler is closely linked to the architecture and is able to exploit code optimization features, both for size and speed.

In order to reduce code size to a minimum, some instructions have multiple addressing modes. As an example, instructions with immediates often have a compact format with a smaller immediate, and an extended format with a larger immediate. In this way, the compiler is able to use the format giving the smallest code size.

Another feature of the instruction set is that frequently used instructions, like add, have a compact format with two operands as well as an extended format with three operands. The larger format increases performance, allowing an addition and a data move in the same instruction in a single cycle. Load and store instructions have several different formats in order to reduce code size and speed up execution.

All interrupt levels are by default disabled when debug state is entered, but they can individually be switched on by the monitor routine by clearing the respective mask bit in the status register.

Debug state can be entered as described in the *AVR32UC Technical Reference Manual*.

Debug state is exited by the *retd* instruction.

6.4.4 System Registers

The system registers are placed outside of the virtual memory space, and are only accessible using the privileged *mfsr* and *mtsr* instructions. The table below lists the system registers specified in the AVR32 architecture, some of which are unused in AVR32UC. The programmer is responsible for maintaining correct sequencing of any instructions following a *mtsr* instruction. For detail on the system registers, refer to the *AVR32UC Technical Reference Manual*.

Table 6-3. System Registers

Reg #	Address	Name	Function
0	0	SR	Status Register
1	4	EVBA	Exception Vector Base Address
2	8	ACBA	Application Call Base Address
3	12	CPUCR	CPU Control Register
4	16	ECR	Exception Cause Register
5	20	RSR_SUP	Unused in AVR32UC
6	24	RSR_INT0	Unused in AVR32UC
7	28	RSR_INT1	Unused in AVR32UC
8	32	RSR_INT2	Unused in AVR32UC
9	36	RSR_INT3	Unused in AVR32UC
10	40	RSR_EX	Unused in AVR32UC
11	44	RSR_NMI	Unused in AVR32UC
12	48	RSR_DBG	Return Status Register for Debug mode
13	52	RAR_SUP	Unused in AVR32UC
14	56	RAR_INT0	Unused in AVR32UC
15	60	RAR_INT1	Unused in AVR32UC
16	64	RAR_INT2	Unused in AVR32UC
17	68	RAR_INT3	Unused in AVR32UC
18	72	RAR_EX	Unused in AVR32UC
19	76	RAR_NMI	Unused in AVR32UC
20	80	RAR_DBG	Return Address Register for Debug mode
21	84	JECR	Unused in AVR32UC
22	88	JOSP	Unused in AVR32UC
23	92	JAVA_LV0	Unused in AVR32UC
24	96	JAVA_LV1	Unused in AVR32UC
25	100	JAVA_LV2	Unused in AVR32UC

7.3 Peripheral Address Map

Table 7-2. Peripheral Address Mapping

Address		Peripheral Name
0xFFFE0000	USB	USB 2.0 Interface - USB
0xFFFE1000	HMATRIX	HSB Matrix - HMATRIX
0xFFFE1400	HFLASHC	Flash Controller - HFLASHC
0xFFFF0000	PDCA	Peripheral DMA Controller - PDCA
0xFFFF0800	INTC	Interrupt controller - INTC
0xFFFF0C00	PM	Power Manager - PM
0xFFFF0D00	RTC	Real Time Counter - RTC
0xFFFF0D30	WDT	Watchdog Timer - WDT
0xFFFF0D80	EIM	External Interrupt Controller - EIM
0xFFFF1000	GPIO	General Purpose Input/Output Controller - GPIO
0xFFFF1400	USART0	Universal Synchronous/Asynchronous Receiver/Transmitter - USART0
0xFFFF1800	USART1	Universal Synchronous/Asynchronous Receiver/Transmitter - USART1
0xFFFF1C00	USART2	Universal Synchronous/Asynchronous Receiver/Transmitter - USART2
0xFFFF2400	SPI0	Serial Peripheral Interface - SPI0
0xFFFF2C00	TWI	Two-wire Interface - TWI
0xFFFF3000	PWM	Pulse Width Modulation Controller - PWM
0xFFFF3400	SSC	Synchronous Serial Controller - SSC
0xFFFF3800	TC	Timer/Counter - TC

Table 9-7. BOD Timing

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
T_{BOD}	Minimum time with VDDCORE < VBOD to detect power failure	Falling VDDCORE from 1.8V to 1.1V		300	800	ns

9.4.3 Reset Sequence

Table 9-8. Electrical Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
V_{DDRR}	VDDCORE rise rate to ensure power-on-reset		2.5			V/ms
V_{DDFR}	VDDCORE fall rate to ensure power-on-reset		0.01		400	V/ms
V_{POR+}	Rising threshold voltage: voltage up to which device is kept under reset by POR on rising VDDCORE	Rising VDDCORE: $V_{RESTART}$ -> V_{POR+}	1.4	1.55	1.65	V
V_{POR-}	Falling threshold voltage: voltage when POR resets device on falling VDDCORE	Falling VDDCORE: 1.8V -> V_{POR+}	1.2	1.3	1.4	V
$V_{RESTART}$	On falling VDDCORE, voltage must go down to this value before supply can rise again to ensure reset signal is released at V_{POR+}	Falling VDDCORE: 1.8V -> $V_{RESTART}$	-0.1		0.5	V
T_{POR}	Minimum time with VDDCORE < V_{POR-}	Falling VDDCORE: 1.8V -> 1.1V		15		µs
T_{RST}	Time for reset signal to be propagated to system			200	400	µs
T_{SSU1}	Time for Cold System Startup: Time for CPU to fetch its first instruction (RCosc not calibrated)		480		960	µs
T_{SSU2}	Time for Hot System Startup: Time for CPU to fetch its first instruction (RCosc calibrated)			420		µs

Figure 9-1. MCU Cold Start-Up RESET_N tied to VDDIN

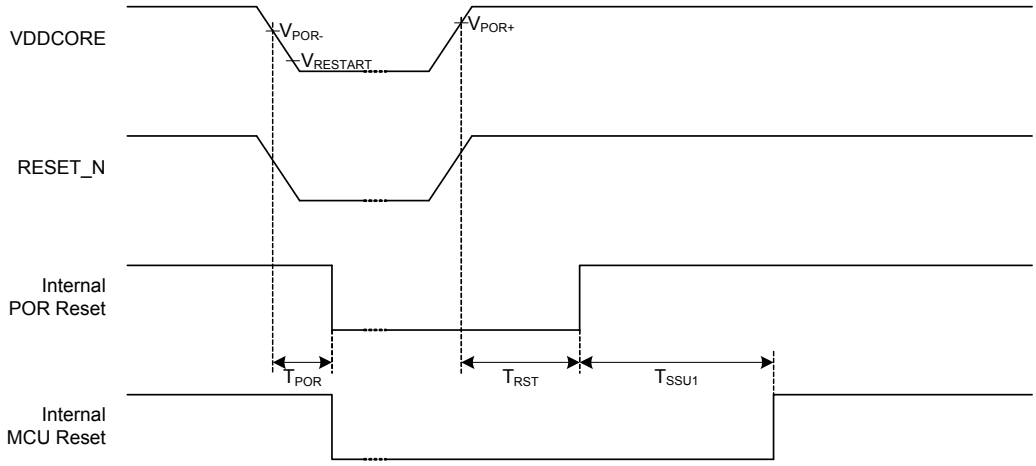


Figure 9-2. MCU Cold Start-Up RESET_N Externally Driven

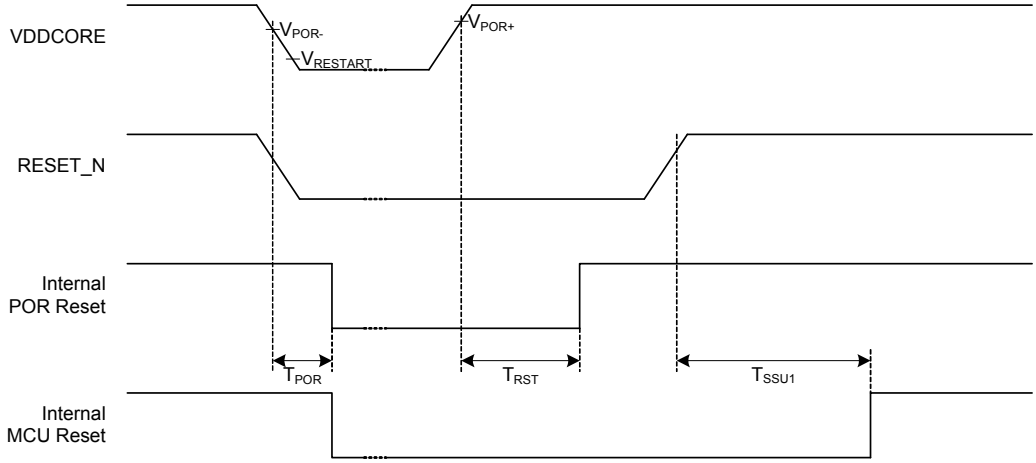
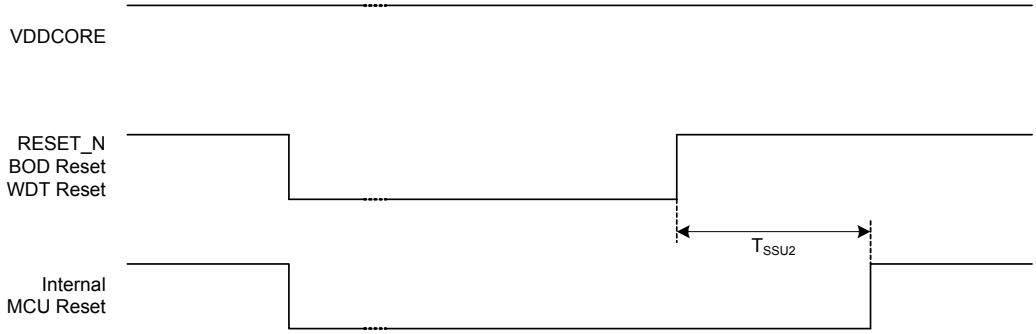


Figure 9-3. MCU Hot Start-Up



In dual supply configuration, the power up sequence must be carefully managed to ensure a safe startup of the device in all conditions.

The power up sequence must ensure that the internal logic is safely powered when the internal reset (Power On Reset) is released and that the internal Flash logic is safely powered when the CPU fetch the first instructions.

9.7 Oscillator Characteristics

The following characteristics are applicable to the operating temperature range: T_A = -40°C to 85°C and worst case of power supply, unless otherwise specified.

9.7.1 Slow Clock RC Oscillator

Table 9-16. RC Oscillator Frequency

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
F_{RC}	RC Oscillator Frequency	Calibration point: T_A = 85°C		115.2	116	KHz
		T_A = 25°C		112		KHz
		T_A = -40°C	105	108		KHz

9.7.2 32 KHz Oscillator

Table 9-17. 32 KHz Oscillator Characteristics

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Unit
$1/(t_{CP32KHz})$	Oscillator Frequency	External clock on XIN32			30	MHz
		Crystal		32 768		Hz
C_L	Equivalent Load Capacitance		6		12.5	pF
ESR	Crystal Equivalent Series Resistance				100	KΩ
t_{ST}	Startup Time	C_L = 6pF(1) C_L = 12.5pF(1)			600 1200	ms
t_{CH}	XIN32 Clock High Half-period		0.4 t_{CP}		0.6 t_{CP}	
t_{CL}	XIN32 Clock Low Half-period		0.4 t_{CP}		0.6 t_{CP}	
C_{IN}	XIN32 Input Capacitance				5	pF
I_{OSC}	Current Consumption	Active mode			1.8	µA
		Standby mode			0.1	µA

Note: 1. C_L is the equivalent load capacitance.

9.11 SPI Characteristics

Figure 9-7. SPI Master mode with (CPOL = NCPHA = 0) or (CPOL= NCPHA= 1)

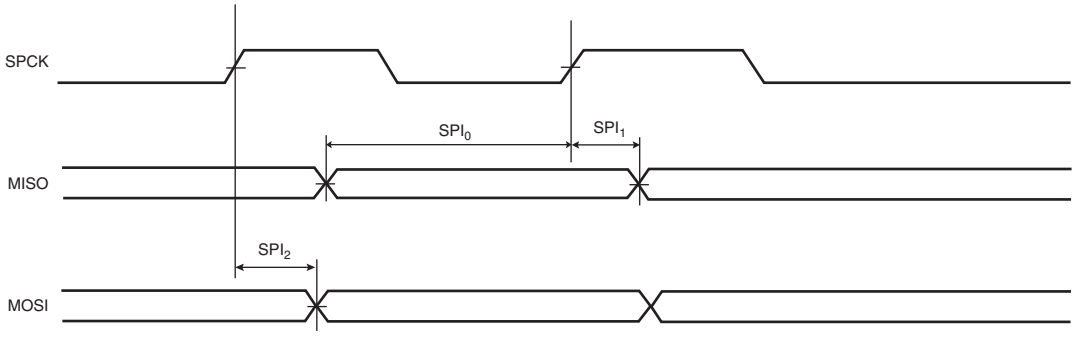


Figure 9-8. SPI Master mode with (CPOL=0 and NCPHA=1) or (CPOL=1 and NCPHA=0)

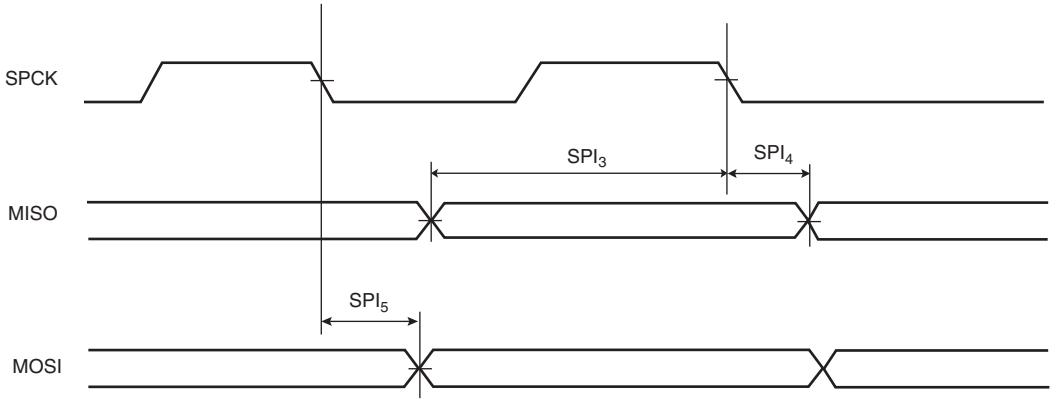
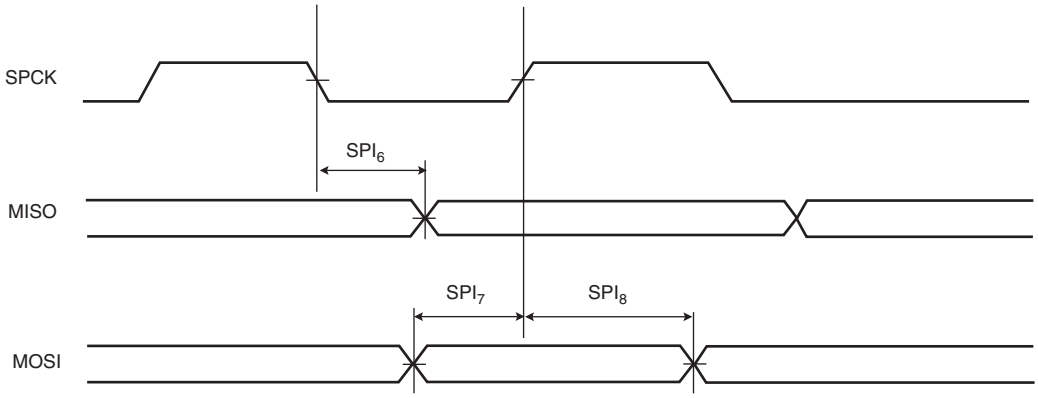


Figure 9-9. SPI Slave mode with (CPOL=0 and NCPHA=1) or (CPOL=1 and NCPHA=0)



10. Mechanical Characteristics

10.1 Thermal Considerations

10.1.1 Thermal Data

Table 10-1 summarizes the thermal resistance data depending on the package.

Table 10-1. Thermal Resistance Data

Symbol	Parameter	Condition	Package	Typ	Unit
θ_{JA}	Junction-to-ambient thermal resistance	Still Air	TQFP64	49.6	°C/W
θ_{JC}	Junction-to-case thermal resistance		TQFP64	13.5	
θ_{JA}	Junction-to-ambient thermal resistance	Still Air	TQFP48	51.1	°C/W
θ_{JC}	Junction-to-case thermal resistance		TQFP48	13.7	

10.1.2 Junction Temperature

The average chip-junction temperature, T_J, in °C can be obtained from the following:

1. $T_J = T_A + (P_D \times \theta_{JA})$
2. $T_J = T_A + (P_D \times (\theta_{HEATSINK} + \theta_{JC}))$

where:

- θ_{JA} = package thermal resistance, Junction-to-ambient (°C/W), provided in Table 10-1 on page 55.
- θ_{JC} = package thermal resistance, Junction-to-case thermal resistance (°C/W), provided in Table 10-1 on page 55.
- $\theta_{HEAT\ SINK}$ = cooling device thermal resistance (°C/W), provided in the device datasheet.
- P_D = device power consumption (W) estimated from data provided in the section "Power Consumption" on page 42.
- T_A = ambient temperature (°C).

From the first equation, the user can derive the estimated lifetime of the chip and decide if a cooling device is necessary or not. If a cooling device is to be fitted on the chip, the second equation should be used to compute the resulting average chip-junction temperature T_J in °C.

12. Errata

12.1 AT32UC3B0512, AT32UC3B1512

12.1.1 Rev D

- *PWM*

1. **PWM channel interrupt enabling triggers an interrupt**
   When enabling a PWM channel that is configured with center aligned period (CALG=1), an interrupt is signalled.
   Fix/Workaround
   When using center aligned mode, enable the channel and read the status before channel interrupt is enabled.

2. **PWN counter restarts at 0x0001**
   The PWM counter restarts at 0x0001 and not 0x0000 as specified. Because of this the first PWM period has one more clock cycle.
   Fix/Workaround
   - The first period is 0x0000, 0x0001, ..., period.
   - Consecutive periods are 0x0001, 0x0002, ..., period.

3. **PWM update period to a 0 value does not work**
   It is impossible to update a period equal to 0 by the using the PWM update register (PWM_CUPD).
   Fix/Workaround
   Do not update the PWM_CUPD register with a value equal to 0.

4. **SPI**

5. **SPI Slave / PDCA transfer: no TX UNDERRUN flag**
   There is no TX UNDERRUN flag available, therefore in SPI slave mode, there is no way to be informed of a character lost in transmission.
   Fix/Workaround
   For PDCA transfer: none.

6. **SPI bad serial clock generation on 2nd chip_select when SCBR=1, CPOL=1, and NCPHA=0**
   When multiple chip selects (CS) are in use, if one of the baudrates equal 1 while one (CSRn.SCBR=1) of the others do not equal 1, and CSRn.CPOL=1 and CSRn.NCPHA=0, then an additional pulse will be generated on SCK.
   Fix/Workaround
   When multiple CS are in use, if one of the baudrates equals 1, the others must also equal 1 if CSRn.CPOL=1 and CSRn.NCPHA=0.

7. **TC**

8. **Channel chaining skips first pulse for upper channel**
   When chaining two channels using the Block Mode Register, the first pulse of the clock between the channels is skipped.
   Fix/Workaround
   Configure the lower channel with RA = 0x1 and RC = 0x2 to produce a dummy clock cycle for the upper channel. After the dummy cycle has been generated, indicated by the SR.CPCS bit, reconfigure the RA and RC registers for the lower channel with the real values.

- *Processor and Architecture*

1. **LDM instruction with PC in the register list and without ++ increments Rp**
   For LDM with PC in the register list: the instruction behaves as if the ++ field is always set, ie the pointer is always updated. This happens even if the ++ field is cleared. Specifically, the increment of the pointer is done in parallel with the testing of R12.
   Fix/Workaround
   None.

2. **RETE instruction does not clear SREG[L] from interrupts**
   The RETE instruction clears SREG[L] as expected from exceptions.
   Fix/Workaround
   When using the STCOND instruction, clear SREG[L] in the stacked value of SR before returning from interrupts with RETE.

3. **Privilege violation when using interrupts in application mode with protected system stack**
   If the system stack is protected by the MPU and an interrupt occurs in application mode, an MPU DTLB exception will occur.
   Fix/Workaround
   Make a DTLB Protection (Write) exception handler which permits the interrupt request to be handled in privileged mode.

4. **USART**

5. **ISO7816 info register US_NER cannot be read**
   The NER register always returns zero.
   Fix/Workaround
   None.

6. **ISO7816 Mode T1: RX impossible after any TX**
   RX impossible after any TX.
   Fix/Workaround
   SOFT_RESET on RX+ Config US_MR + Config_US_CR.

7. **The RTS output does not function correctly in hardware handshaking mode**
   The RTS signal is not generated properly when the USART receives data in hardware handshaking mode. When the Peripheral DMA receive buffer becomes full, the RTS output should go high, but it will stay low.
   Fix/Workaround
   Do not use the hardware handshaking mode of the USART. If it is necessary to drive the RTS output high when the Peripheral DMA receive buffer becomes full, use the normal mode of the USART. Configure the Peripheral DMA Controller to signal an interrupt when

- *Processor and Architecture*

1. **LDM instruction with PC in the register list and without ++ increments Rp**
   For LDM with PC in the register list: the instruction behaves as if the ++ field is always set, ie the pointer is always updated. This happens even if the ++ field is cleared. Specifically, the increment of the pointer is done in parallel with the testing of R12.
   Fix/Workaround
   None.

2. **RETE instruction does not clear SREG[L] from interrupts**
   The RETE instruction clears SREG[L] as expected from exceptions.
   Fix/Workaround
   When using the STCOND instruction, clear SREG[L] in the stacked value of SR before returning from interrupts with RETE.

3. **Privilege violation when using interrupts in application mode with protected system stack**
   If the system stack is protected by the MPU and an interrupt occurs in application mode, an MPU DTLB exception will occur.
   Fix/Workaround
   Make a DTLB Protection (Write) exception handler which permits the interrupt request to be handled in privileged mode.

4. **Flash**

5. **Reset vector is 80000020h rather than 80000000h**
   Reset vector is 80000020h rather than 80000000h.
   Fix/Workaround
   The flash program code must start at the address 80000020h. The flash memory range 80000000h-80000020h must be programmed with 00000000h.

- *USART*

1. **ISO7816 info register US_NER cannot be read**
   The NER register always returns zero.
   Fix/Workaround
   None.

2. **ISO7816 Mode T1: RX impossible after any TX**
   RX impossible after any TX.
   Fix/Workaround
   SOFT_RESET on RX+ Config US_MR + Config_US_CR.

3. **The RTS output does not function correctly in hardware handshaking mode**
   The RTS signal is not generated properly when the USART receives data in hardware handshaking mode. When the Peripheral DMA receive buffer becomes full, the RTS output should go high, but it will stay low.
   Fix/Workaround
   Do not use the hardware handshaking mode of the USART. If it is necessary to drive the RTS output high when the Peripheral DMA receive buffer becomes full, use the normal mode of the USART. Configure the Peripheral DMA Controller to signal an interrupt when the receive buffer is full. In the interrupt handler code, write a one to the RTSDIS bit in the USART Control Register (CR). This will drive the RTS output high. After the next DMA trans-

15. **SSC**

16. **Additional delay on TD output**
    A delay from 2 to 3 system clock cycles is added to TD output when:
    TCMR.START = Receive Start,
    TCMR.STTDLY = more than ZERO,
    RCMR.START = Start on falling edge / Start on Rising edge / Start on any edge,
    RFMR.FSOS = None (input).
    Fix/Workaround
    None.

17. **TF output is not correct**
    TF output is not correct (at least emitted one serial clock cycle later than expected) when:
    TFMR.FSOS = Driven Low during data transfer/ Driven High during data transfer
    TCMR.START = Receive start
    RFMR.FSOS = None (Input)
    RCMR.START = any on RF (edge/level)
    Fix/Workaround
    None.

18. **Frame Synchro and Frame Synchro Data are delayed by one clock cycle**
    The frame synchro and the frame synchro data are delayed from 1 SSC_CLOCK when:
    - Clock is CKDIV
    - The START is selected on either a frame synchro edge or a level
    - Frame synchro data is enabled
    - Transmit clock is gated on output (through CKO field)
    Fix/Workaround
    Transmit or receive CLOCK must not be gated (by the mean of CKO field) when START condition is performed on a generated frame synchro.

19. **USB**

20. **UPCFGn.INTFRQ is irrelevant for isochronous pipe**
    As a consequence, isochronous IN and OUT tokens are sent every 1ms (Full Speed), or every 125uS (High Speed).
    Fix/Workaround
    For higher polling time, the software must freeze the pipe for the desired period in order to prevent any "extra" token.

- *ADC*

1. **Sleep Mode activation needs additional A to D conversion**
   If the ADC sleep mode is activated when the ADC is idle the ADC will not enter sleep mode before after the next AD conversion.
   Fix/Workaround
   Activate the sleep mode in the mode register and then perform an AD conversion.

- *PDCA*

1. **Wrong PDCA behavior when using two PDCA channels with the same PID**
   Wrong PDCA behavior when using two PDCA channels with the same PID.
   Fix/Workaround
   The same PID should not be assigned to more than one channel.

SR.CPCS bit, reconfigure the RA and RC registers for the lower channel with the real values.

- *OCD*

1. **The auxiliary trace does not work for CPU/HSB speed higher than 50MHz**
   The auxiliary trace does not work for CPU/HSB speed higher than 50MHz.
   Fix/Workaround
   Do not use the auxiliary trace for CPU/HSB speed higher than 50MHz.

- *Processor and Architecture*

1. **LDM instruction with PC in the register list and without ++ increments Rp**
   For LDM with PC in the register list: the instruction behaves as if the ++ field is always set, ie the pointer is always updated. This happens even if the ++ field is cleared. Specifically, the increment of the pointer is done in parallel with the testing of R12.
   Fix/Workaround
   None.

2. **RETE instruction does not clear SREG[L] from interrupts**
   The RETE instruction clears SREG[L] as expected from exceptions.
   Fix/Workaround
   When using the STCOND instruction, clear SREG[L] in the stacked value of SR before returning from interrupts with RETE.

3. **RETS behaves incorrectly when MPU is enabled**
   RETS behaves incorrectly when MPU is enabled and MPU is configured so that system stack is not readable in unprivileged mode.
   Fix/Workaround
   Make system stack readable in unprivileged mode, or return from supervisor mode using rete instead of rets. This requires:
   1. Changing the mode bits from 001 to 110 before issuing the instruction. Updating the mode bits to the desired value must be done using a single mtsr instruction so it is done atomically. Even if this step is generally described as not safe in the UC technical reference manual, it is safe in this very specific case.
   2. Execute the RETE instruction.

4. **Privilege violation when using interrupts in application mode with protected system stack**
   If the system stack is protected by the MPU and an interrupt occurs in application mode, an MPU DTLB exception will occur.
   Fix/Workaround
   Make a DTLB Protection (Write) exception handler which permits the interrupt request to be handled in privileged mode.

5. **USART**

6. **ISO7816 info register US_NER cannot be read**
   The NER register always returns zero.
   Fix/Workaround
   None.

7. **ISO7816 Mode T1: RX impossible after any TX**
   RX impossible after any TX.
   Fix/Workaround
   SOFT_RESET on RX+ Config US_MR + Config_US_CR.

8. **The RTS output does not function correctly in hardware handshaking mode**
   The RTS signal is not generated properly when the USART receives data in hardware handshaking mode. When the Peripheral DMA receive buffer becomes full, the RTS output should go high, but it will stay low.
   Fix/Workaround
   Do not use the hardware handshaking mode of the USART. If it is necessary to drive the RTS output high when the Peripheral DMA receive buffer becomes full, use the normal mode of the USART. Configure the Peripheral DMA Controller to signal an interrupt when the receive buffer is full. In the interrupt handler code, write a one to the RTSDIS bit in the USART Control Register (CR). This will drive the RTS output high. After the next DMA transfer is started and a receive buffer is available, write a one to the RTSEN bit in the USART CR so that RTS will be driven low.

9. **Corruption after receiving too many bits in SPI slave mode**
   If the USART is in SPI slave mode and receives too much data bits (ex: 9bitsinstead of 8 bits) by the SPI master, an error occurs. After that, the next reception may be corrupted even if the frame is correct and the USART has been disabled, reset by a soft reset and re-enabled.
   Fix/Workaround
   None.

10. **USART slave synchronous mode external clock must be at least 9 times lower in frequency than CLK_USART**
    When the USART is operating in slave synchronous mode with an external clock, the frequency of the signal provided on CLK must be at least 9 times lower than CLK_USART.
    Fix/Workaround
    When the USART is operating in slave synchronous mode with an external clock, provide a signal on CLK that has a frequency at least 9 times lower than CLK_USART.

11. **HMATRIX**

12. **In the PRAS and PRBS registers, the MxPR fields are only two bits**
    In the PRAS and PRBS registers, the MxPR fields are only two bits wide, instead of four bits. The unused bits are undefined when reading the registers.
    Fix/Workaround
    Mask undefined bits when reading PRAS and PRBS.

- *FLASHC*

1. **Reading from on-chip flash may fail after a flash fuse write operation (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands).**
   After a flash fuse write operation (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands), the following flash read access may return corrupted data. This erratum does not affect write operations to regular flash memory.
   Fix/Workaround
   The flash fuse write operation (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands) must be issued from internal RAM. After the write operation, perform a dummy flash page write operation (FLASHC WP). Content and location of this page is not important

and filling the write buffer with all one (FFh) will leave the current flash content unchanged. It is then safe to read and fetch code from the flash.

- *DSP Operations*

1. **Hardware breakpoints may corrupt MAC results**
   Hardware breakpoints on MAC instructions may corrupt the destination register of the MAC instruction.
   Fix/Workaround
   Place breakpoints on earlier or later instructions.

it is done atomically. Even if this step is described in general as not safe in the UC technical reference guide, it is safe in this very specific case.
2. Execute the RETE instruction.