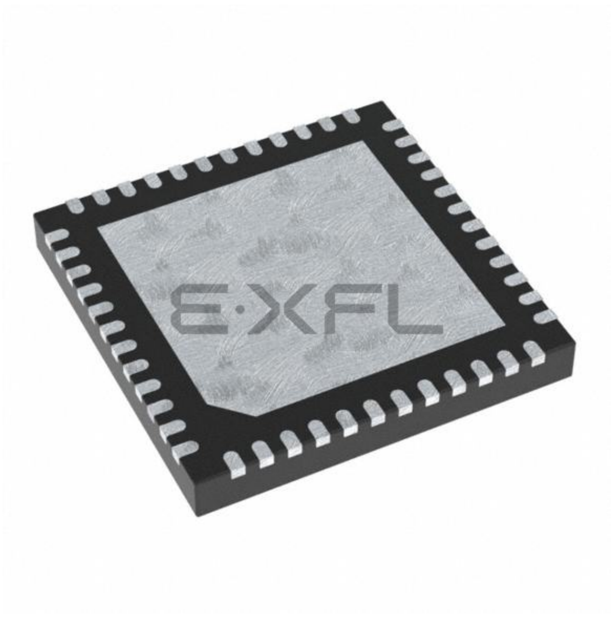
Microchip Technology - AT32UC3B1128-Z1UR Datasheet



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	28
Program Memory Size	128KB (128K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	32K x 8
Voltage - Supply (Vcc/Vdd)	1.65V ~ 3.6V
Data Converters	A/D 6x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	48-VFQFN Exposed Pad
Supplier Device Package	48-QFN (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at32uc3b1128-z1ur

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

Refer to Section 9.3 on page 38 for decoupling capacitors values and regulator characteristics.

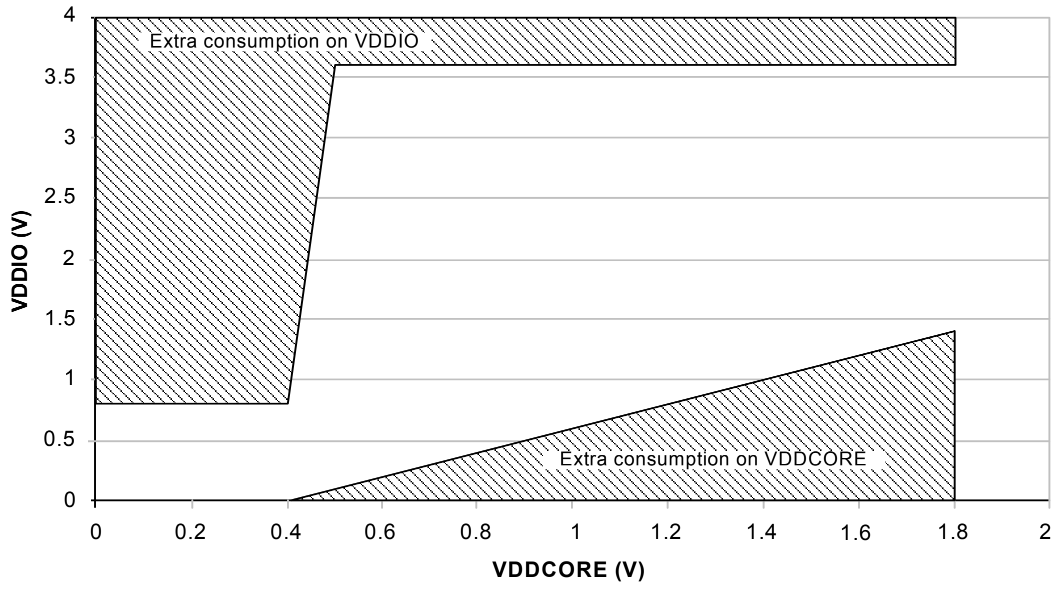
For decoupling recommendations for VDDIO, VDDANA, VDDCORE and VDDPLL, please refer to the Schematic checklist.

5.6.2.2 *Dual Power Supply*

In case of dual power supply, VDDIN and VDDOUT should be connected to ground to prevent from leakage current.

To avoid over consumption during the power up sequence, VDDIO and VDDCORE voltage difference needs to stay in the range given Figure 5-3.

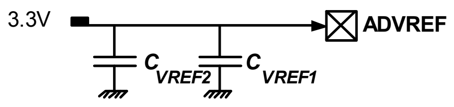
Figure 5-3. VDDIO versus VDDCORE during power up sequence



5.6.3 Analog-to-Digital Converter (ADC) reference.

The ADC reference (ADVREF) must be provided from an external source. Two decoupling capacitors must be used to insure proper decoupling.

Figure 5-4. ADVREF Decoupling



Refer to Section 9.4 on page 38 for decoupling capacitors values and electrical characteristics.

In case ADC is not used, the ADVREF pin should be connected to GND to avoid extra consumption.

The register file is organized as sixteen 32-bit registers and includes the Program Counter, the Link Register, and the Stack Pointer. In addition, register R12 is designed to hold return values from function calls and is used implicitly by some instructions.

6.3 The AVR32UC CPU

The AVR32UC CPU targets low- and medium-performance applications, and provides an advanced OCD system, no caches, and a Memory Protection Unit (MPU). Java acceleration hardware is not implemented.

AVR32UC provides three memory interfaces, one High Speed Bus master for instruction fetch, one High Speed Bus master for data access, and one High Speed Bus slave interface allowing other bus masters to access data RAMs internal to the CPU. Keeping data RAMs internal to the CPU allows fast access to the RAMs, reduces latency, and guarantees deterministic timing. Also, power consumption is reduced by not needing a full High Speed Bus access for memory accesses. A dedicated data RAM interface is provided for communicating with the internal data RAMs.

A local bus interface is provided for connecting the CPU to device-specific high-speed systems, such as floating-point units and fast GPIO ports. This local bus has to be enabled by writing the LOCEN bit in the CPUCR system register. The local bus is able to transfer data between the CPU and the local bus slave in a single clock cycle. The local bus has a dedicated memory range allocated to it, and data transfers are performed using regular load and store instructions. Details on which devices that are mapped into the local bus space is given in the Memories chapter of this data sheet.

Figure 6-1 on page 19 displays the contents of AVR32UC.

The user must also make sure that the system stack is large enough so that any event is able to push the required registers to stack. If the system stack is full, and an event occurs, the system will enter an UNDEFINED state.

6.5.2 Exceptions and Interrupt Requests

When an event other than *scall* or debug request is received by the core, the following actions are performed atomically:

1. The pending event will not be accepted if it is masked. The I3M, I2M, I1M, I0M, EM, and GM bits in the Status Register are used to mask different events. Not all events can be masked. A few critical events (NMI, Unrecoverable Exception, TLB Multiple Hit, and Bus Error) can not be masked. When an event is accepted, hardware automatically sets the mask bits corresponding to all sources with equal or lower priority. This inhibits acceptance of other events of the same or lower priority, except for the critical events listed above. Software may choose to clear some or all of these bits after saving the necessary state if other priority schemes are desired. It is the event source's responsability to ensure that their events are left pending until accepted by the CPU.
2. When a request is accepted, the Status Register and Program Counter of the current context is stored to the system stack. If the event is an INT0, INT1, INT2, or INT3, registers R8-R12 and LR are also automatically stored to stack. Storing the Status Register ensures that the core is returned to the previous execution mode when the current event handling is completed. When exceptions occur, both the EM and GM bits are set, and the application may manually enable nested exceptions if desired by clearing the appropriate bit. Each exception handler has a dedicated handler address, and this address uniquely identifies the exception source.
3. The Mode bits are set to reflect the priority of the accepted event, and the correct register file bank is selected. The address of the event handler, as shown in Table 6-4, is loaded into the Program Counter.

The execution of the event handler routine then continues from the effective address calculated.

The *rete* instruction signals the end of the event. When encountered, the Return Status Register and Return Address Register are popped from the system stack and restored to the Status Register and Program Counter. If the *rete* instruction returns from INT0, INT1, INT2, or INT3, registers R8-R12 and LR are also popped from the system stack. The restored Status Register contains information allowing the core to resume operation in the previous execution mode. This concludes the event handling.

6.5.3 Supervisor Calls

The AVR32 instruction set provides a supervisor mode call instruction. The *scall* instruction is designed so that privileged routines can be called from any context. This facilitates sharing of code between different execution modes. The *scall* mechanism is designed so that a minimal execution cycle overhead is experienced when performing supervisor routine calls from time-critical event handlers.

The *scall* instruction behaves differently depending on which mode it is called from. The behaviour is detailed in the instruction set reference. In order to allow the *scall* routine to return to the correct context, a return from supervisor call instruction, *rets*, is implemented. In the AVR32UC CPU, *scall* and *rets* uses the system stack to store the return address and the status register.

6.5.4 Debug Requests

The AVR32 architecture defines a dedicated Debug mode. When a debug request is received by the core, Debug mode is entered. Entry into Debug mode can be masked by the DM bit in the

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

9.5.1 Power Consumtion for Different Sleep Modes

Table 9-10. Power Consumption for Different Sleep Modes for AT32UC3B064, AT32UC3B0128, AT32UC3B0256, AT32UC3B164, AT32UC3B1128, AT32UC3B1256

Mode	Conditions		Typ.	Unit
Active	- CPU running a recursive Fibonacci Algorithm from flash and clocked from PLL0 at f MHz. - Voltage regulator is on. - XIN0: external clock. Xin1 Stopped. XIN32 stopped. - All peripheral clocks activated with a division by 8. - GPIOs are inactive with internal pull-up, JTAG unconnected with external pull-up and Input pins are connected to GND		0.3xf(MHz)+0.443	mA/MHz
	Same conditions at 60 MHz		18.5	mA
Idle	See Active mode conditions		0.117xf(MHz)+0.28	mA/MHz
	Same conditions at 60 MHz		7.3	mA
Frozen	See Active mode conditions		0.058xf(MHz)+0.115	mA/MHz
	Same conditions at 60 MHz		3.6	mA
Standby	See Active mode conditions		0.042xf(MHz)+0.115	mA/MHz
	Same conditions at 60 MHz		2.7	mA
Stop	- CPU running in sleep mode - XIN0, Xin1 and XIN32 are stopped. - All peripheral clocks are desactived. - GPIOs are inactive with internal pull-up, JTAG unconnected with external pull-up and Input pins are connected to GND.		37.8	µA
Deepstop	See Stop mode conditions		24.9	µA
Static	See Stop mode conditions	Voltage Regulator On	13.9	µA
		Voltage Regulator Off	8.9	µA

Notes: 1. Core frequency is generated from XIN0 using the PLL so that 140 MHz < f_{PLL0} < 160 MHz and 10 MHz < f_{XIN0} < 12 MHz.

Table 9-11. Power Consumption for Different Sleep Modes for AT32UC3B0512, AT32UC3B1512

Mode	Conditions	Typ.	Unit
Active	- CPU running a recursive Fibonacci Algorithm from flash and clocked from PLL0 at f MHz. - Voltage regulator is on. - XIN0: external clock. Xin1 Stopped. XIN32 stopped. - All peripheral clocks activated with a division by 8. - GPIOs are inactive with internal pull-up, JTAG unconnected with external pull-up and Input pins are connected to GND	0.359xf(MHz)+1.53	mA/MHz
	Same conditions at 60 MHz	24	mA
Idle	See Active mode conditions	0.146xf(MHz)+0.291	mA/MHz
	Same conditions at 60 MHz	9	mA

9.10 JTAG Characteristics

9.10.1 JTAG Timing

Figure 9-6. JTAG Interface Signals

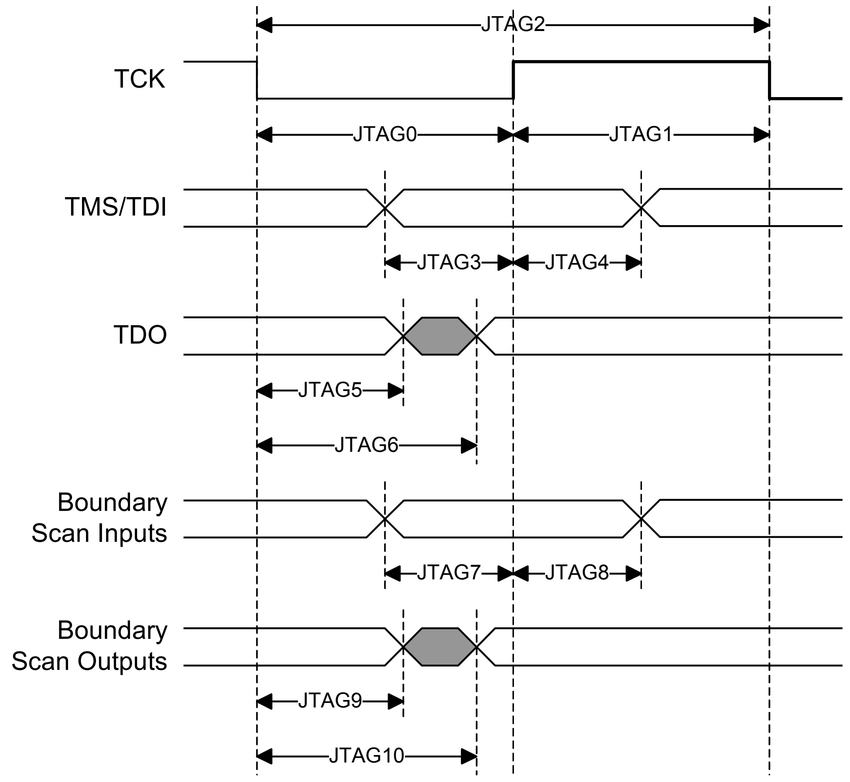


Table 9-26. JTAG Timings[1]

Symbol	Parameter	Conditions	Min	Max	Units
JTAG0	TCK Low Half-period	V_{VDDIO} from 3.0V to 3.6V, maximum external capacitor = 40pF	23.2		ns
JTAG1	TCK High Half-period		8.8		ns
JTAG2	TCK Period		32.0		ns
JTAG3	TDI, TMS Setup before TCK High		3.9		ns
JTAG4	TDI, TMS Hold after TCK High		0.6		ns
JTAG5	TDO Hold Time		4.5		ns
JTAG6	TCK Low to TDO Valid			23.2	ns
JTAG7	Boundary Scan Inputs Setup Time		0		ns
JTAG8	Boundary Scan Inputs Hold Time		5.0		ns
JTAG9	Boundary Scan Outputs Hold Time		8.7		ns
JTAG10	TCK to Boundary Scan Outputs Valid			17.7	ns

Note: 1. These values are based on simulation and characterization of other AVR microcontrollers manufactured in the same pro-cess technology. These values are not covered by test limits in production.

Figure 9-10. SPI Slave mode with (CPOL = NCPHA = 0) or (CPOL= NCPHA= 1)

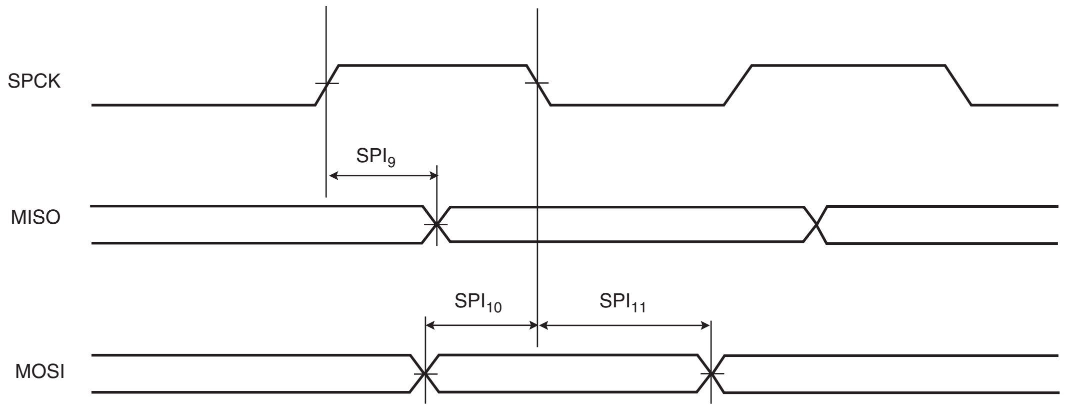


Table 9-27. SPI Timings

Symbol	Parameter	Conditions	Min.	Max.	Unit
SPI_0	MISO Setup time before SPCK rises (master)	3.3V domain(1)	22 + $(t_{CPMCK})/2$(2)		ns
SPI_1	MISO Hold time after SPCK rises (master)	3.3V domain(1)	0		ns
SPI_2	SPCK rising to MOSI Delay (master)	3.3V domain(1)		7	ns
SPI_3	MISO Setup time before SPCK falls (master)	3.3V domain(1)	22 + $(t_{CPMCK})/2$(2)		ns
SPI_4	MISO Hold time after SPCK falls (master)	3.3V domain(1)	0		ns
SPI_5	SPCK falling to MOSI Delay master)	3.3V domain(1)		7	ns
SPI_6	SPCK falling to MISO Delay (slave)	3.3V domain(1)		26.5	ns
SPI_7	MOSI Setup time before SPCK rises (slave)	3.3V domain(1)	0		ns
SPI_8	MOSI Hold time after SPCK rises (slave)	3.3V domain(1)	1.5		ns
SPI_9	SPCK rising to MISO Delay (slave)	3.3V domain(1)		27	ns
SPI_{10}	MOSI Setup time before SPCK falls (slave)	3.3V domain(1)	0		ns
SPI_{11}	MOSI Hold time after SPCK falls (slave)	3.3V domain(1)	1		ns

Notes: 1. 3.3V domain: V_{VDDIO} from 3.0V to 3.6V, maximum external capacitor = 40 pF.
2. t_{CPMCK}: Master Clock period in ns.

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.

10.3 Soldering Profile

Table 10-14 gives the recommended soldering profile from J-STD-20.

Table 10-14. Soldering Profile

Profile Feature	Green Package
Average Ramp-up Rate (217°C to Peak)	3°C/s
Preheat Temperature 175°C ±25°C	Min. 150°C, Max. 200°C
Temperature Maintained Above 217°C	60-150s
Time within 5·C of Actual Peak Temperature	30s
Peak Temperature Range	260°C
Ramp-down Rate	6°C/s
Time 25·C to Peak Temperature	Max. 8mn

Note: It is recommended to apply a soldering temperature higher than 250°C.

A maximum of three reflow passes is allowed per component.

11. Ordering Information

Device	Ordering Code	Package	Conditioning	Temperature Operating Range
AT32UC3B0512	AT32UC3B0512-A2UES	TQFP 64	-	Industrial (-40°C to 85°C)
	AT32UC3B0512-A2UR	TQFP 64	Reel	Industrial (-40°C to 85°C)
	AT32UC3B0512-A2UT	TQFP 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B0512-Z2UES	QFN 64	-	Industrial (-40°C to 85°C)
	AT32UC3B0512-Z2UR	QFN 64	Reel	Industrial (-40°C to 85°C)
	AT32UC3B0512-Z2UT	QFN 64	Tray	Industrial (-40°C to 85°C)
AT32UC3B0256	AT32UC3B0256-A2UT	TQFP 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B0256-A2UR	TQFP 64	Reel	Industrial (-40°C to 85°C)
	AT32UC3B0256-Z2UT	QFN 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B0256-Z2UR	QFN 64	Reel	Industrial (-40°C to 85°C)
AT32UC3B0128	AT32UC3B0128-A2UT	TQFP 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B0128-A2UR	TQFP 64	Reel	Industrial (-40°C to 85°C)
	AT32UC3B0128-Z2UT	QFN 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B0128-Z2UR	QFN 64	Reel	Industrial (-40°C to 85°C)
AT32UC3B064	AT32UC3B064-A2UT	TQFP 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B064-A2UR	TQFP 64	Reel	Industrial (-40°C to 85°C)
	AT32UC3B064-Z2UT	QFN 64	Tray	Industrial (-40°C to 85°C)
	AT32UC3B064-Z2UR	QFN 64	Reel	Industrial (-40°C to 85°C)
AT32UC3B1512	AT32UC3B1512-Z1UT	QFN 48	-	Industrial (-40°C to 85°C)
	AT32UC3B1512-Z1UR	QFN 48	-	Industrial (-40°C to 85°C)
AT32UC3B1256	AT32UC3B1256-AUT	TQFP 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B1256-AUR	TQFP 48	Reel	Industrial (-40°C to 85°C)
	AT32UC3B1256-Z1UT	QFN 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B1256-Z1UR	QFN 48	Reel	Industrial (-40°C to 85°C)
AT32UC3B1128	AT32UC3B1128-AUT	TQFP 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B1128-AUR	TQFP 48	Reel	Industrial (-40°C to 85°C)
	AT32UC3B1128-Z1UT	QFN 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B1128-Z1UR	QFN 48	Reel	Industrial (-40°C to 85°C)
AT32UC3B164	AT32UC3B164-AUT	TQFP 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B164-AUR	TQFP 48	Reel	Industrial (-40°C to 85°C)
	AT32UC3B164-Z1UT	QFN 48	Tray	Industrial (-40°C to 85°C)
	AT32UC3B164-Z1UR	QFN 48	Reel	Industrial (-40°C to 85°C)

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

3. Set the polarity CPOL of the line in the opposite value of the required one.
4. Set the polarity CPOL to the required one.
5. Read the RXHOLDING register.
Transfers can now begin and RXREADY will now behave as expected.

8. **SPI disable does not work in SLAVE mode**
SPI disable does not work in SLAVE mode.
Fix/Workaround
Read the last received data, then perform a software reset by writing a one to the Software Reset bit in the Control Register (CR.SWRST).

9. **SPI data transfer hangs with CSR0.CSAAT==1 and MR.MODFDIS==0**
When CSR0.CSAAT==1 and mode fault detection is enabled (MR.MODFDIS==0), the SPI module will not start a data transfer.
Fix/Workaround
Disable mode fault detection by writing a one to MR.MODFDIS.

10. **Disabling SPI has no effect on the SR.TDRE bit**
Disabling SPI has no effect on the SR.TDRE bit whereas the write data command is filtered when SPI is disabled. Writing to TDR when SPI is disabled will not clear SR.TDRE. If SPI is disabled during a PDCA transfer, the PDCA will continue to write data to TDR until its buffer is empty, and this data will be lost.
Fix/Workaround
Disable the PDCA, add two NOPs, and disable the SPI. To continue the transfer, enable the SPI and PDCA.

11. **Power Manager**

12. **If the BOD level is higher than VDDCORE, the part is constantly reset**
If the BOD level is set to a value higher than VDDCORE and enabled by fuses, the part will be in constant reset.
Fix/Workaround
Apply an external voltage on VDDCORE that is higher than the BOD level and is lower than VDDCORE max and disable the BOD.

1. **When the main clock is RCSYS, TIMER_CLOCK5 is equal to PBA clock**
When the main clock is generated from RCSYS, TIMER_CLOCK5 is equal to PBA Clock and not PBA Clock / 128.
Fix/Workaround
None.

13. **Clock sources will not be stopped in STATIC sleep mode if the difference between CPU and PBx division factor is too high**
If the division factor between the CPU/HSB and PBx frequencies is more than 4 when going to a sleep mode where the system RC oscillator is turned off, then high speed clock sources will not be turned off. This will result in a significantly higher power consumption during the sleep mode.
Fix/Workaround
Before going to sleep modes where the system RC oscillator is stopped, make sure that the factor between the CPU/HSB and PBx frequencies is less than or equal to 4.

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.

- *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.

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

- *SSC*

1. **SSC does not trigger RF when data is low**
   The SSC cannot transmit or receive data when CKS = CKDIV and CKO = none, in TCMR or RCMR respectively.
   Fix/Workaround
   Set CKO to a value that is not "none" and bypass the output of the TK/RK pin with the GPIO.

- *USB*

1. **USB No end of host reset signaled upon disconnection**
   In host mode, in case of an unexpected device disconnection whereas a usb reset is being sent by the usb controller, the UHCON.RESET bit may not been cleared by the hardware at the end of the reset.
   Fix/Workaround
   A software workaround consists in testing (by polling or interrupt) the disconnection (UHINT.DDISCI == 1) while waiting for the end of reset (UHCON.RESET == 0) to avoid being stuck.

2. **USBFSM and UHADDR1/2/3 registers are not available**
   Do not use USBFSM register.
   Fix/Workaround
   Do not use USBFSM register and use HCON[6:0] field instead for all the pipes.

- *Cycle counter*

1. **CPU Cycle Counter does not reset the COUNT system register on COMPARE match.**
   The device revision B does not reset the COUNT system register on COMPARE match. In this revision, the COUNT register is clocked by the CPU clock, so when the CPU clock stops, so does incrementing of COUNT**.**
   Fix/Workaround
   None.

- *ADC*

1. **ADC possible miss on DRDY when disabling a channel**
   The ADC does not work properly when more than one channel is enabled.
   Fix/Workaround
   Do not use the ADC with more than one channel enabled at a time.

2. **ADC OVRE flag sometimes not reset on Status Register read**
   The OVRE flag does not clear properly if read simultaneously to an end of conversion.
   Fix/Workaround
   None.

3. **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.

Figure 12-1. Timer/Counter clock connections on RevB

Source	Name	Connection
Internal	TIMER_CLOCK1	32KHz Oscillator
	TIMER_CLOCK2	PBA Clock / 4
	TIMER_CLOCK3	PBA Clock / 8
	TIMER_CLOCK4	PBA Clock / 16
	TIMER_CLOCK5	PBA Clock / 32
External	XC0	
	XC1	
	XC2	

7. **Spurious interrupt may corrupt core SR mode to exception**
If the rules listed in the chapter `Masking interrupt requests in peripheral modules' of the AVR32UC Technical Reference Manual are not followed, a spurious interrupt may occur. An interrupt context will be pushed onto the stack while the core SR mode will indicate an exception. A RETE instruction would then corrupt the stack.
Fix/Workaround
Follow the rules of the AVR32UC Technical Reference Manual. To increase software robustness, if an exception mode is detected at the beginning of an interrupt handler, change the stack interrupt context to an exception context and issue a RETE instruction.

8. **CPU cannot operate on a divided slow clock (internal RC oscillator)**
CPU cannot operate on a divided slow clock (internal RC oscillator).
Fix/Workaround
Do not run the CPU on a divided slow clock.

9. **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, i.e. 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.

10. **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.

11. **Exceptions when system stack is protected by MPU**
RETS behaves incorrectly when MPU is enabled and MPU is configured so that system stack is not readable in unprivileged mode.
Fix/Workaround
Workaround 1: Make system stack readable in unprivileged mode,
or
Workaround 2: Return from supervisor mode using rete instead of rets. This requires: 1. Changing the mode bits from 001b to 110b before issuing the instruction.
Updating the mode bits to the desired value must be done using a single mtsr instruction so

Table of Contents