Welcome to E-XFL.COM

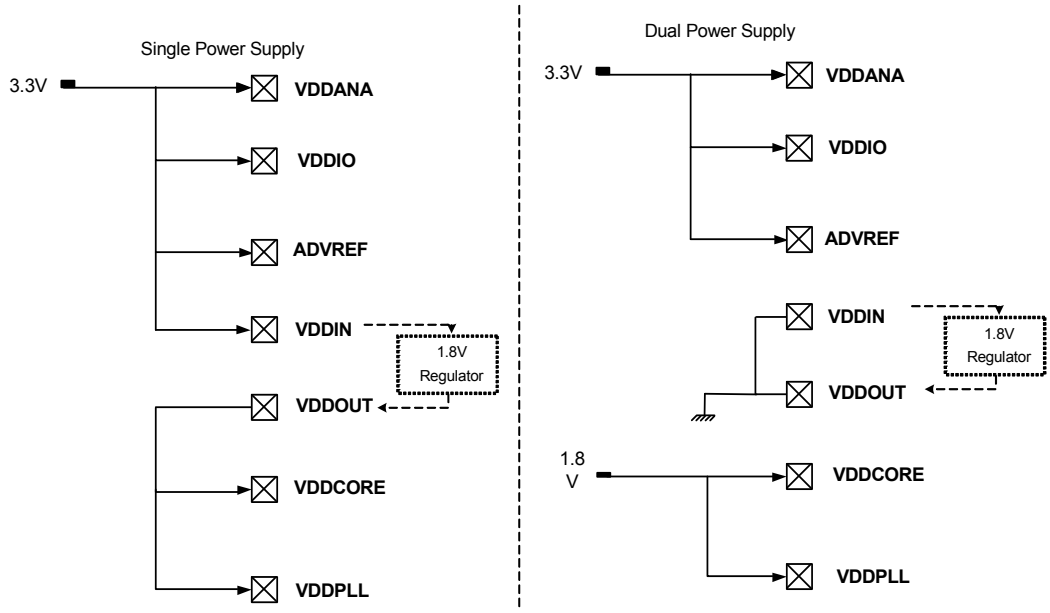
What is "Embedded - Microcontrollers"?

"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	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	16K 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-TQFP
Supplier Device Package	64-TQFP (10x10)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at32uc3b064-a2ut

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

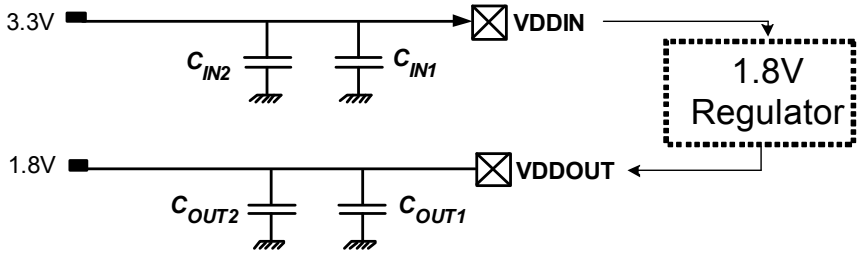
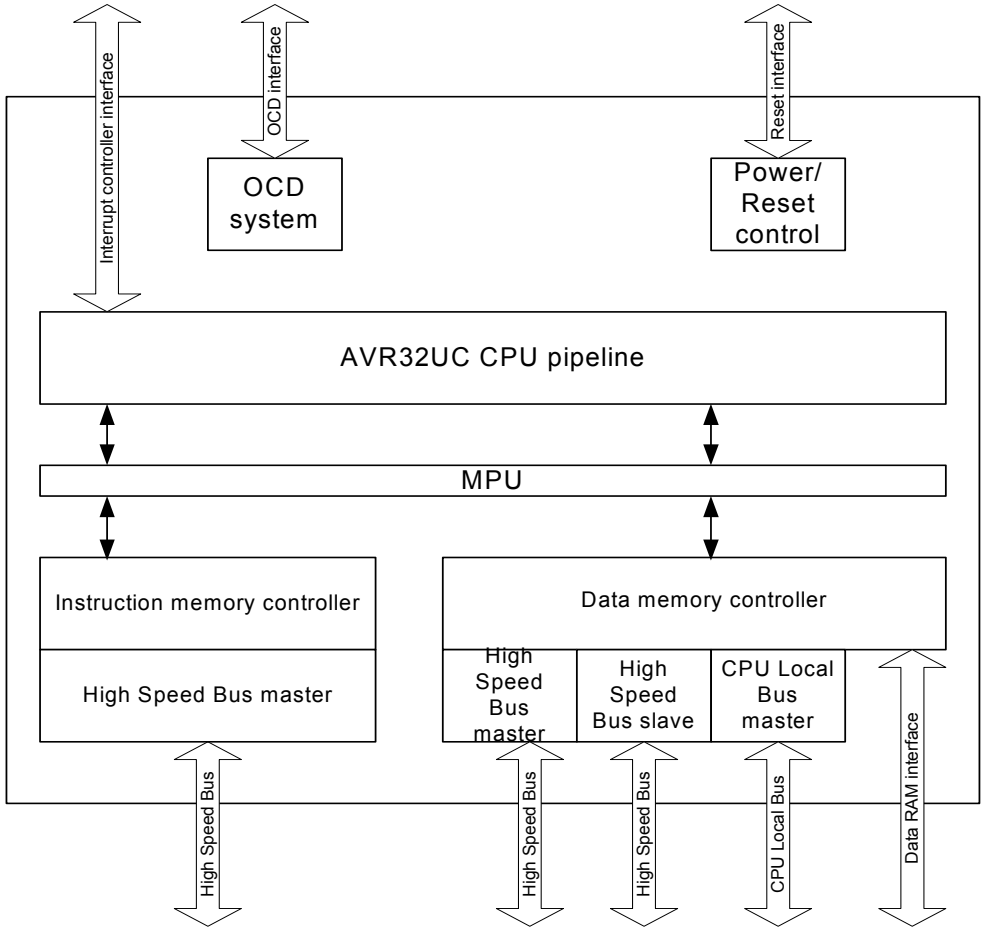


Figure 6-1. Overview of the AVR32UC CPU



6.3.1 Pipeline Overview

AVR32UC has three pipeline stages, Instruction Fetch (IF), Instruction Decode (ID), and Instruction Execute (EX). The EX stage is split into three parallel subsections, one arithmetic/logic (ALU) section, one multiply (MUL) section, and one load/store (LS) section.

Instructions are issued and complete in order. Certain operations require several clock cycles to complete, and in this case, the instruction resides in the ID and EX stages for the required number of clock cycles. Since there is only three pipeline stages, no internal data forwarding is required, and no data dependencies can arise in the pipeline.

Figure 6-2 on page 20 shows an overview of the AVR32UC pipeline stages.

The following table shows the instructions with support for unaligned addresses. All other instructions require aligned addresses.

Table 6-1. Instructions with Unaligned Reference Support

Instruction	Supported alignment
ld.d	Word
st.d	Word

6.3.6 Unimplemented Instructions

The following instructions are unimplemented in AVR32UC, and will cause an Unimplemented Instruction Exception if executed:

- All SIMD instructions
- All coprocessor instructions if no coprocessors are present
- retj, incjosp, popjc, pushjc
- tlbr, tlbs, tlbw
- cache

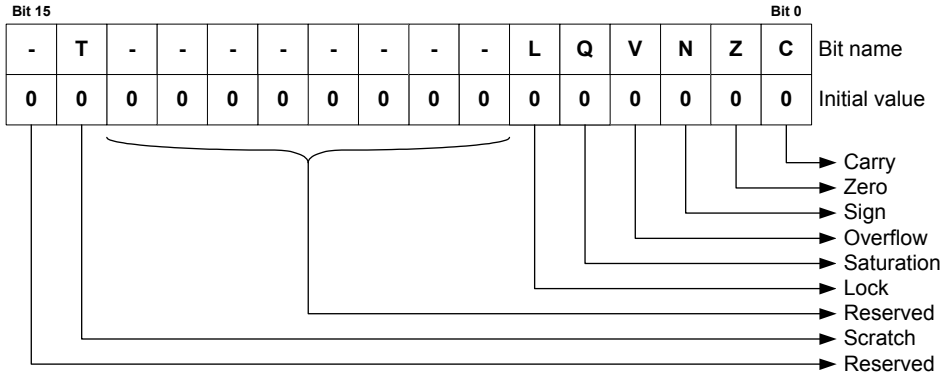
6.3.7 CPU and Architecture Revision

Three major revisions of the AVR32UC CPU currently exist.

The Architecture Revision field in the CONFIG0 system register identifies which architecture revision is implemented in a specific device.

AVR32UC CPU revision 3 is fully backward-compatible with revisions 1 and 2, ie. code compiled for revision 1 or 2 is binary-compatible with revision 3 CPUs.

Figure 6-5. The Status Register Low Halfword



6.4.3 Processor States

6.4.3.1 Normal RISC State

The AVR32 processor supports several different execution contexts as shown in Table 6-2 on page 23.

Table 6-2. Overview of Execution Modes, their Priorities and Privilege Levels.

Priority	Mode	Security	Description
1	Non Maskable Interrupt	Privileged	Non Maskable high priority interrupt mode
2	Exception	Privileged	Execute exceptions
3	Interrupt 3	Privileged	General purpose interrupt mode
4	Interrupt 2	Privileged	General purpose interrupt mode
5	Interrupt 1	Privileged	General purpose interrupt mode
6	Interrupt 0	Privileged	General purpose interrupt mode
N/A	Supervisor	Privileged	Runs supervisor calls
N/A	Application	Unprivileged	Normal program execution mode

Mode changes can be made under software control, or can be caused by external interrupts or exception processing. A mode can be interrupted by a higher priority mode, but never by one with lower priority. Nested exceptions can be supported with a minimal software overhead.

When running an operating system on the AVR32, user processes will typically execute in the application mode. The programs executed in this mode are restricted from executing certain instructions. Furthermore, most system registers together with the upper halfword of the status register cannot be accessed. Protected memory areas are also not available. All other operating modes are privileged and are collectively called System Modes. They have full access to all privileged and unprivileged resources. After a reset, the processor will be in supervisor mode.

6.4.3.2 Debug State

The AVR32 can be set in a debug state, which allows implementation of software monitor routines that can read out and alter system information for use during application development. This implies that all system and application registers, including the status registers and program counters, are accessible in debug state. The privileged instructions are also available.

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

9. Electrical Characteristics

9.1 Absolute Maximum Ratings*

Operating Temperature	-40°C to +85°C
Storage Temperature	-60°C to +150°C
Voltage on GPIO Pins with respect to Ground for TCK, RESET_N, PA03, PA04, PA05, PA06, PA07, PA08, PA11, PA12, PA18, PA19, PA28, PA29, PA30, PA31	-0.3 to 3.6V
Voltage on GPIO Pins with respect to Ground except for TCK, RESET_N, PA03, PA04, PA05, PA06, PA07, PA08, PA11, PA12, PA18, PA19, PA28, PA29, PA30, PA31	-0.3 to 5.5V
Maximum Operating Voltage (VDDCORE, VDDPLL)	1.95V
Maximum Operating Voltage (VDDIO,VDDIN,VDDANA)	3.6V
Total DC Output Current on all I/O Pin	
for 48-pin package	200 mA
for 64-pin package	265 mA

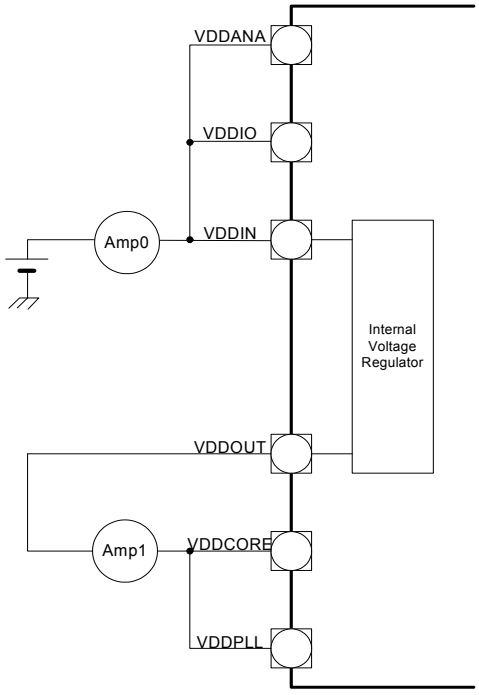
*NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

9.5 Power Consumption

The values in Table 9-10, Table 9-11 on page 43 and Table 9-12 on page 44 are measured values of power consumption with operating conditions as follows:

- $V_{DDIO} = V_{DDANA} = 3.3V$
- $V_{DDCORE} = V_{DDPLL} = 1.8V$
- $T_A = 25°C$, $T_A = 85°C$
- I/Os are configured in input, pull-up enabled.

Figure 9-5. Measurement Setup



The following tables represent the power consumption measured on the power supplies.

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.

9.8 ADC Characteristics

Table 9-20. Channel Conversion Time and ADC Clock

Parameter	Conditions	Min.	Typ.	Max.	Unit
ADC Clock Frequency	10-bit resolution mode			5	MHz
ADC Clock Frequency	8-bit resolution mode			8	MHz
Startup Time	Return from Idle Mode			20	µs
Track and Hold Acquisition Time		600			ns
Track and Hold Input Resistor			350		Ω
Track and Hold Capacitor			12		pF
Conversion Time	ADC Clock = 5 MHz			2	µs
	ADC Clock = 8 MHz			1.25	µs
Throughput Rate	ADC Clock = 5 MHz			384(1)	kSPS
	ADC Clock = 8 MHz			533(2)	kSPS

Notes: 1. Corresponds to 13 clock cycles: 3 clock cycles for track and hold acquisition time and 10 clock cycles for conversion.
2. Corresponds to 15 clock cycles: 5 clock cycles for track and hold acquisition time and 10 clock cycles for conversion.

Table 9-21. External Voltage Reference Input

Parameter	Conditions	Min.	Typ.	Max.	Unit
ADVREF Input Voltage Range	(1)	2.6		VDDANA	V
ADVREF Average Current	On 13 samples with ADC Clock = 5 MHz		200	250	µA
Current Consumption on VDDANA	On 13 samples with ADC Clock = 5 MHz			1	mA

Note: 1. ADVREF should be connected to GND to avoid extra consumption in case ADC is not used.

Table 9-22. Analog Inputs

Parameter	Conditions	Min.	Typ.	Max.	Unit
Input Voltage Range		0		V_{ADVREF}	V
Input Leakage Current				1	µA
Input Capacitance			7		pF

Table 9-23. Transfer Characteristics in 8-bit Mode

Parameter	Conditions	Min.	Typ.	Max.	Unit
Resolution			8		Bit
Absolute Accuracy	ADC Clock = 5 MHz			0.8	LSB
	ADC Clock = 8 MHz			1.5	LSB
Integral Non-linearity	ADC Clock = 5 MHz		0.35	0.5	LSB
	ADC Clock = 8 MHz		0.5	1.0	LSB

7. **SPI Glitch on RXREADY flag in slave mode when enabling the SPI or during the first transfer**
   In slave mode, the SPI can generate a false RXREADY signal during enabling of the SPI or during the first transfer.
   Fix/Workaround
   1. Set slave mode, set required CPOL/CPHA.
   2. Enable SPI.
   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.

13. **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.

14. **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

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.

8. **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.

9. **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.

10. **HMATRIX**

11. **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.

- *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.

12.1.2 Rev C

- *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. **SPI Glitch on RXREADY flag in slave mode when enabling the SPI or during the first transfer**
   In slave mode, the SPI can generate a false RXREADY signal during enabling of the SPI or during the first transfer.
   Fix/Workaround
   1. Set slave mode, set required CPOL/CPHA.
   2. Enable SPI.
   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.

13. **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.

14. **VDDCORE power supply input needs to be 1.95V**
    When used in dual power supply, VDDCORE needs to be 1.95V.
    Fix/Workaround
    When used in single power supply, VDDCORE needs to be connected to VDDOUT, which is configured on revision C at 1.95V (typ.).

15. **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.

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.

- *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 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.

8. **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

12.2.2 Rev. G

- *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. **SPI Glitch on RXREADY flag in slave mode when enabling the SPI or during the first transfer**
   In slave mode, the SPI can generate a false RXREADY signal during enabling of the SPI or during the first transfer.
   Fix/Workaround
   1. Set slave mode, set required CPOL/CPHA.
   2. Enable SPI.
   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.

- *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.

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.

- *USART*

1. **USART Manchester Encoder Not Working**
   Manchester encoding/decoding is not working.
   Fix/Workaround
   Do not use manchester encoding.

2. **USART RXBREAK problem when no timeguard**
   In asynchronous mode the RXBREAK flag is not correctly handled when the timeguard is 0 and the break character is located just after the stop bit.
   Fix/Workaround
   If the NBSTOP is 1, timeguard should be different from 0.

3. **USART Handshaking: 2 characters sent / CTS rises when TX**
   If CTS switches from 0 to 1 during the TX of a character, if the Holding register is not empty, the TXHOLDING is also transmitted.
   Fix/Workaround
   None.

4. **USART PDC and TIMEGUARD not supported in MANCHESTER**
   Manchester encoding/decoding is not working.
   Fix/Workaround
   Do not use manchester encoding.

5. **USART SPI mode is non functional on this revision**
   USART SPI mode is non functional on this revision.
   Fix/Workaround
   Do not use the USART SPI mode.

- *HMATRIX*

1. **HMatrix fixed priority arbitration does not work**
   Fixed priority arbitration does not work.
   Fix/Workaround
   Use Round-Robin arbitration instead.

- *Clock characteristic*

1. **PBA max frequency**
   The Peripheral bus A (PBA) max frequency is 30MHz instead of 60MHz.
   Fix/Workaround
   Do not set the PBA maximum frequency higher than 30MHz.

- *FLASHC*

1. **The address of Flash General Purpose Fuse Register Low (FGPFRLO) is 0xFFFE140C on revB instead of 0xFFFE1410**
   The address of Flash General Purpose Fuse Register Low (FGPFRLO) is 0xFFFE140C on revB instead of 0xFFFE1410.
   Fix/Workaround
   None.

13.12 Rev. A – 05/2007

1. Initial revision.