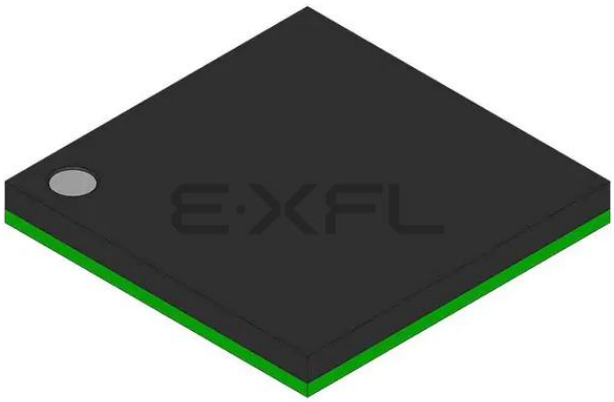


Welcome to E-XFL.COM

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	66MHz
Connectivity	EBI/EMI, I²C, IrDA, Memory Card, SPI, SSC, UART/USART, USB OTG
Peripherals	Brown-out Detect/Reset, DMA, POR, WDT
Number of I/O	88
Program Memory Size	128KB (128K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	1.75V ~ 3.6V
Data Converters	A/D 8x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	100-VFBGA
Supplier Device Package	100-VFBGA (7x7)
Purchase URL	https://www.e-xfl.com/product-detail/atmel/at32uc3a4128-c1ur

2.2 Configuration Summary

The table below lists all AT32UC3A3/A4 memory and package configurations:

Table 2-1. Configuration Summary

Feature	AT32UC3A3256/128/64	AT32UC3A4256/128/64
Flash	256/128/64 KB	
SRAM	64 KB	
HSB RAM	64 KB	
EBI	Full	Nand flash only
GPIO	110	70
External Interrupts	8	
TWI	2	
USART	4	
Peripheral DMA Channels	8	
Generic DMA Channels	4	
SPI	2	
MCI slots	2 MMC/SD slots	1 MMC/SD slot + 1 SD slot
High Speed USB	1	
AES (S option)	1	
SSC	1	
Audio Bitstream DAC	1	
Timer/Counter Channels	6	
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/OSC1) Crystal Oscillator 32 KHz (OSC32K) RC Oscillator 115 kHz (RCSYS)	
10-bit ADC number of channels	1 8	
JTAG	1	
Max Frequency	84 MHz	
Package	LQFP144, TFBGA144	VFBGA100

Figure 3-2. LQFP144 Pinout

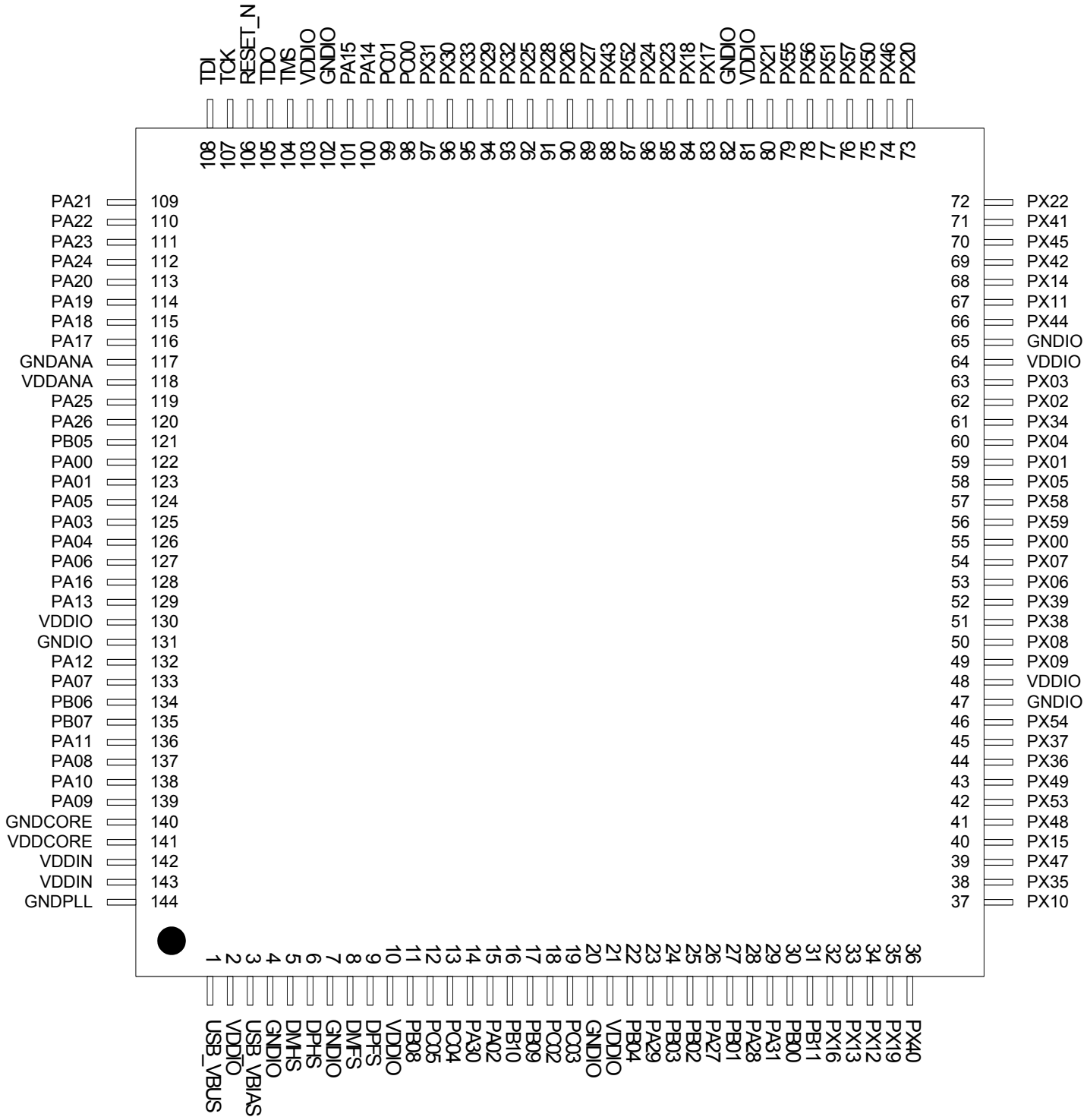
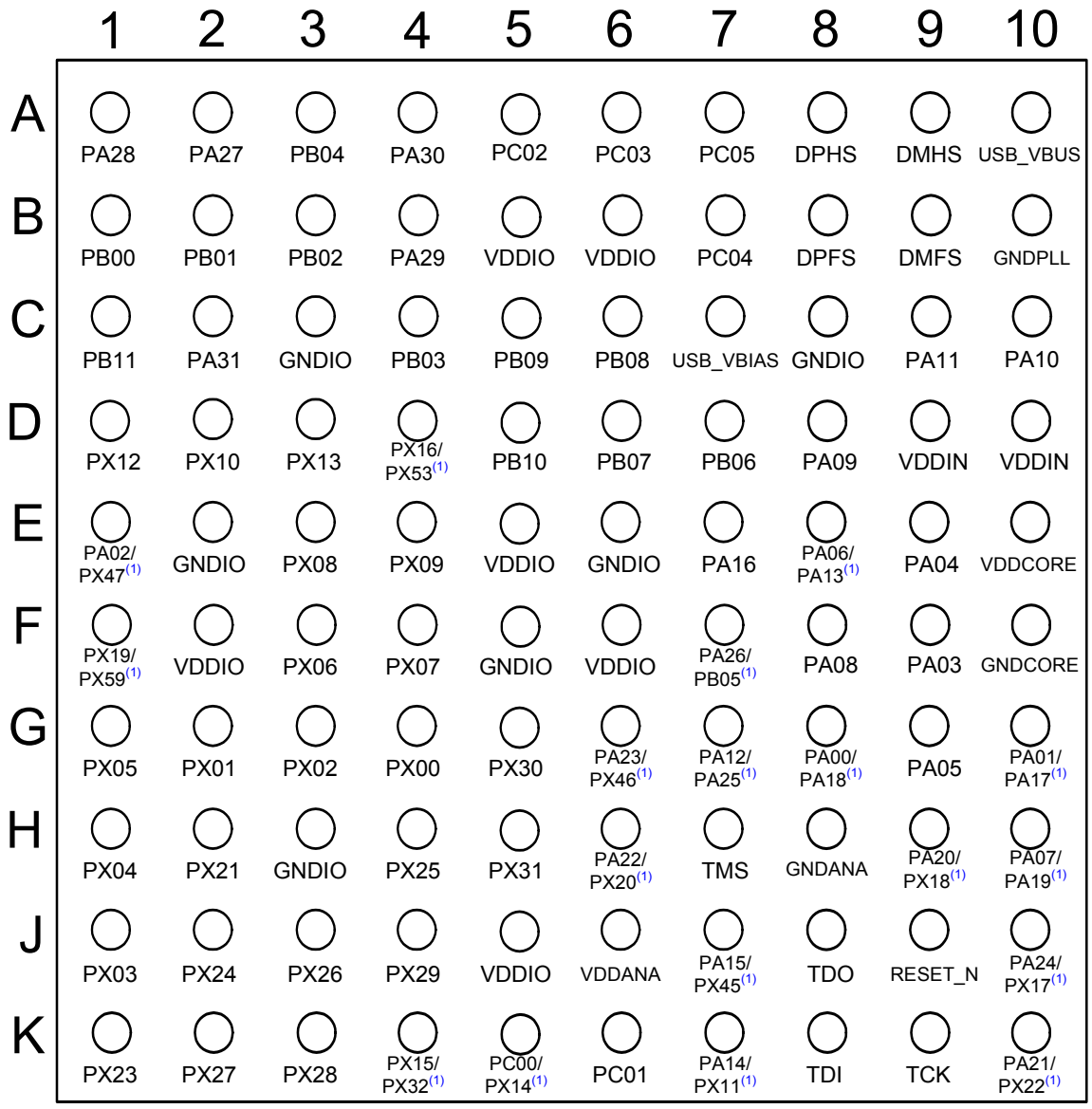


Figure 3-3. VFBGA100 Pinout (top view)



Note: 1. Those balls are physically connected to 2 GPIOs. Software must managed carrefully the GPIO configuration to avoid electrical conflict

Table 3-1. GPIO Controller Function Multiplexing

BGA 144	QFP 144	BGA 100	PIN	GPIO	Supply	PIN Type (2)	GPIO function			
							A	B	C	D
J4	78		PX56	107	VDDIO	x2	EBI - ADDR[21]	EIC - SCAN[2]	USART2 - TXD	
H4	76		PX57	108	VDDIO	x2	EBI - ADDR[20]	EIC - SCAN[1]	USART3 - RXD	
H3	57		PX58	109	VDDIO	x2	EBI - NCS[0]	EIC - SCAN[0]	USART3 - TXD	
G3	56	F1(1)	PX59	110	VDDIO	x2	EBI - NANDWE		MCI - CMD[1]	

Note: 1. Those balls are physically connected to 2 GPIOs. Software must managed carrefully the GPIO configuration to avoid electrical conflict.

2. Refer to "Electrical Characteristics" on page 40 for a description of the electrical properties of the pad types used..

3.2.2 Peripheral Functions

Each GPIO line can be assigned to one of several peripheral functions. The following table describes how the various peripheral functions are selected. The last listed function has priority in case multiple functions are enabled on the same pin.

Table 3-2. Peripheral Functions

Function	Description
GPIO Controller Function multiplexing	GPIO and GPIO peripheral selection A to D
Nexus OCD AUX port connections	OCD trace system
JTAG port connections	JTAG debug port
Oscillators	OSC0, OSC1, OSC32

3.2.3 Oscillator Pinout

The oscillators are not mapped to the normal GPIO functions and their muxings are controlled by registers in the Power Mananger (PM). Please refer to the PM chapter for more information about this.

Table 3-3. Oscillator Pinout

TFBGA144	QFP144	VFBGA100	Pin name	Oscillator pin
A7	18	A5	PC02	XIN0
B7	19	A6	PC03	XOUT0
A8	13	B7	PC04	XIN1
A9	12	A7	PC05	XOUT1
K5	98	K5(1)	PC00	XIN32
H6	99	K6	PC01	XOUT32

Note: 1. This ball is physically connected to 2 GPIOs. Software must managed carrefully the GPIO configuration to avoid electrical conflict

Table 3-6. Signal Description List

Signal Name	Function	Type	Active Level	Comments
RESET_N	Reset Pin	Input	Low	
DMA Controller - DMACA (optional)				
DMAACK[1:0]	DMA Acknowledge	Output		
DMARQ[1:0]	DMA Requests	Input		
External Interrupt Controller - EIC				
EXTINT[7:0]	External Interrupt Pins	Input		
SCAN[7:0]	Keypad Scan Pins	Output		
NMI	Non-Maskable Interrupt Pin	Input	Low	
General Purpose Input/Output pin - GPIOA, GPIOB, GPIOC, GPIOX				
PA[31:0]	Parallel I/O Controller GPIO port A	I/O		
PB[11:0]	Parallel I/O Controller GPIO port B	I/O		
PC[5:0]	Parallel I/O Controller GPIO port C	I/O		
PX[59:0]	Parallel I/O Controller GPIO port X	I/O		
External Bus Interface - EBI				
ADDR[23:0]	Address Bus	Output		
CAS	Column Signal	Output	Low	
CFCE1	Compact Flash 1 Chip Enable	Output	Low	
CFCE2	Compact Flash 2 Chip Enable	Output	Low	
CFRNW	Compact Flash Read Not Write	Output		
DATA[15:0]	Data Bus	I/O		
NANDOE	NAND Flash Output Enable	Output	Low	
NANDWE	NAND Flash Write Enable	Output	Low	
NCS[5:0]	Chip Select	Output	Low	
NRD	Read Signal	Output	Low	
NWAIT	External Wait Signal	Input	Low	
NWE0	Write Enable 0	Output	Low	
NWE1	Write Enable 1	Output	Low	
RAS	Row Signal	Output	Low	

Table 4-3. System Registers (Continued)

Reg #	Address	Name	Function
26	104	JAVA_LV3	Unused in AVR32UC
27	108	JAVA_LV4	Unused in AVR32UC
28	112	JAVA_LV5	Unused in AVR32UC
29	116	JAVA_LV6	Unused in AVR32UC
30	120	JAVA_LV7	Unused in AVR32UC
31	124	JTBA	Unused in AVR32UC
32	128	JBCR	Unused in AVR32UC
33-63	132-252	Reserved	Reserved for future use
64	256	CONFIG0	Configuration register 0
65	260	CONFIG1	Configuration register 1
66	264	COUNT	Cycle Counter register
67	268	COMPARE	Compare register
68	272	TLBEHI	Unused in AVR32UC
69	276	TLBELO	Unused in AVR32UC
70	280	PTBR	Unused in AVR32UC
71	284	TLBEAR	Unused in AVR32UC
72	288	MMUCR	Unused in AVR32UC
73	292	TLBARLO	Unused in AVR32UC
74	296	TLBARHI	Unused in AVR32UC
75	300	PCCNT	Unused in AVR32UC
76	304	PCNT0	Unused in AVR32UC
77	308	PCNT1	Unused in AVR32UC
78	312	PCCR	Unused in AVR32UC
79	316	BEAR	Bus Error Address Register
80	320	MPUAR0	MPU Address Register region 0
81	324	MPUAR1	MPU Address Register region 1
82	328	MPUAR2	MPU Address Register region 2
83	332	MPUAR3	MPU Address Register region 3
84	336	MPUAR4	MPU Address Register region 4
85	340	MPUAR5	MPU Address Register region 5
86	344	MPUAR6	MPU Address Register region 6
87	348	MPUAR7	MPU Address Register region 7
88	352	MPUPSR0	MPU Privilege Select Register region 0
89	356	MPUPSR1	MPU Privilege Select Register region 1
90	360	MPUPSR2	MPU Privilege Select Register region 2
91	364	MPUPSR3	MPU Privilege Select Register region 3

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.

4.5.2 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 4-4, 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.

4.5.3 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.

4.5.4 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

status register. Upon entry into Debug mode, hardware sets the SR[D] bit and jumps to the Debug Exception handler. By default, Debug mode executes in the exception context, but with dedicated Return Address Register and Return Status Register. These dedicated registers remove the need for storing this data to the system stack, thereby improving debuggability. The mode bits in the status register can freely be manipulated in Debug mode, to observe registers in all contexts, while retaining full privileges.

Debug mode is exited by executing the *retd* instruction. This returns to the previous context.

4.5.5 Entry Points for Events

Several different event handler entry points exists. In AVR32UC, the reset address is 0x8000_0000. This places the reset address in the boot flash memory area.

TLB miss exceptions and *scall* have a dedicated space relative to EVBA where their event handler can be placed. This speeds up execution by removing the need for a jump instruction placed at the program address jumped to by the event hardware. All other exceptions have a dedicated event routine entry point located relative to EVBA. The handler routine address identifies the exception source directly.

AVR32UC uses the ITLB and DTLB protection exceptions to signal a MPU protection violation. ITLB and DTLB miss exceptions are used to signal that an access address did not map to any of the entries in the MPU. TLB multiple hit exception indicates that an access address did map to multiple TLB entries, signalling an error.

All external interrupt requests have entry points located at an offset relative to EVBA. This autovector offset is specified by an external Interrupt Controller. The programmer must make sure that none of the autovector offsets interfere with the placement of other code. The autovector offset has 14 address bits, giving an offset of maximum 16384 bytes.

Special considerations should be made when loading EVBA with a pointer. Due to security considerations, the event handlers should be located in non-writeable flash memory, or optionally in a privileged memory protection region if an MPU is present.

If several events occur on the same instruction, they are handled in a prioritized way. The priority ordering is presented in Table 4-4. If events occur on several instructions at different locations in the pipeline, the events on the oldest instruction are always handled before any events on any younger instruction, even if the younger instruction has events of higher priority than the oldest instruction. An instruction B is younger than an instruction A if it was sent down the pipeline later than A.

The addresses and priority of simultaneous events are shown in Table 4-4. Some of the exceptions are unused in AVR32UC since it has no MMU, coprocessor interface, or floating-point unit.

Table 4-4. Priority and Handler Addresses for Events

Priority	Handler Address	Name	Event source	Stored Return Address
1	0x8000_0000	Reset	External input	Undefined
2	Provided by OCD system	OCD Stop CPU	OCD system	First non-completed instruction
3	EVBA+0x00	Unrecoverable exception	Internal	PC of offending instruction
4	EVBA+0x04	TLB multiple hit	MPU	
5	EVBA+0x08	Bus error data fetch	Data bus	First non-completed instruction
6	EVBA+0x0C	Bus error instruction fetch	Data bus	First non-completed instruction
7	EVBA+0x10	NMI	External input	First non-completed instruction
8	Autovectored	Interrupt 3 request	External input	First non-completed instruction
9	Autovectored	Interrupt 2 request	External input	First non-completed instruction
10	Autovectored	Interrupt 1 request	External input	First non-completed instruction
11	Autovectored	Interrupt 0 request	External input	First non-completed instruction
12	EVBA+0x14	Instruction Address	CPU	PC of offending instruction
13	EVBA+0x50	ITLB Miss	MPU	
14	EVBA+0x18	ITLB Protection	MPU	PC of offending instruction
15	EVBA+0x1C	Breakpoint	OCD system	First non-completed instruction
16	EVBA+0x20	Illegal Opcode	Instruction	PC of offending instruction
17	EVBA+0x24	Unimplemented instruction	Instruction	PC of offending instruction
18	EVBA+0x28	Privilege violation	Instruction	PC of offending instruction
19	EVBA+0x2C	Floating-point	UNUSED	
20	EVBA+0x30	Coprocessor absent	Instruction	PC of offending instruction
21	EVBA+0x100	Supervisor call	Instruction	PC(Supervisor Call) +2
22	EVBA+0x34	Data Address (Read)	CPU	PC of offending instruction
23	EVBA+0x38	Data Address (Write)	CPU	PC of offending instruction
24	EVBA+0x60	DTLB Miss (Read)	MPU	
25	EVBA+0x70	DTLB Miss (Write)	MPU	
26	EVBA+0x3C	DTLB Protection (Read)	MPU	PC of offending instruction
27	EVBA+0x40	DTLB Protection (Write)	MPU	PC of offending instruction
28	EVBA+0x44	DTLB Modified	UNUSED	

Table 5-2. Peripheral Address Mapping

0xFFFF0C00	PM	Power Manager - PM
0xFFFF0D00	RTC	Real Time Counter - RTC
0xFFFF0D30	WDT	Watchdog Timer - WDT
0xFFFF0D80	EIC	External Interrupt Controller - EIC
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
0xFFFF2000	USART3	Universal Synchronous/Asynchronous Receiver/Transmitter - USART3
0xFFFF2400	SPI0	Serial Peripheral Interface - SPI0
0xFFFF2800	SPI1	Serial Peripheral Interface - SPI1
0xFFFF2C00	TWIM0	Two-wire Master Interface - TWIM0
0xFFFF3000	TWIM1	Two-wire Master Interface - TWIM1
0xFFFF3400	SSC	Synchronous Serial Controller - SSC
0xFFFF3800	TC0	Timer/Counter - TC0
0xFFFF3C00	ADC	Analog to Digital Converter - ADC
0xFFFF4000	ABDAC	Audio Bitstream DAC - ABDAC
0xFFFF4400	TC1	Timer/Counter - TC1

6. Boot Sequence

This chapter summarizes the boot sequence of the AT32UC3A3/A4. The behavior after power-up is controlled by the Power Manager. For specific details, refer to Section 7. ”Power Manager (PM)” on page 86.

6.1 Starting of Clocks

After power-up, the device will be held in a reset state by the Power-On Reset circuitry, until the power has stabilized throughout the device. Once the power has stabilized, the device will use the internal RC Oscillator as clock source.

On system start-up, the PLLs are disabled. All clocks to all modules are running. No clocks have a divided frequency, all parts of the system receives a clock with the same frequency as the internal RC Oscillator.

6.2 Fetching of Initial Instructions

After reset has been released, the AVR32 UC CPU starts fetching instructions from the reset address, which is 0x8000_0000. This address points to the first address in the internal Flash.

The internal Flash uses VDDIO voltage during read and write operations. BOD33 monitors this voltage and maintains the device under reset until VDDIO reaches the minimum voltage, preventing any spurious execution from flash.

The code read from the internal Flash is free to configure the system to use for example the PLLs, to divide the frequency of the clock routed to some of the peripherals, and to gate the clocks to unused peripherals.

When powering up the device, there may be a delay before the voltage has stabilized, depending on the rise time of the supply used. The CPU can start executing code as soon as the supply is above the POR threshold, and before the supply is stable. Before switching to a high-speed clock source, the user should use the BOD to make sure the VDDCORE is above the minimum-level (1.62V).

Table 7-13. Typical Cuurent Consumption by Peripheral

Peripheral	Typ.	Unit
ADC	7	µA/MHz
AES	80	
ABDAC	10	
DMACA	70	
EBI	23	
EIC	0.5	
GPIO	37	
INTC	3	
MCI	40	
MSI	10	
PDCA	20	
SDRAM	5	
SMC	9	
SPI	6	
SSC	10	
RTC	5	
TC	8	
TWIM	2	
TWIS	2	
USART	10	
USBB	90	
WDT	2	

Table 7-34. SMC Write Signals with No Hold Settings (NWE Controlled only)

Symbol	Parameter	Min.	Unit
SMC_{43}	Data Out Valid before NWE Rising	(nwe pulse length - 1) * t_{CPSMC} - 1.2	ns
SMC_{44}	Data Out Valid after NWE Rising	5	ns
SMC_{45}	NWE Pulse Width	nwe pulse length * t_{CPSMC} - 0.9	ns

Figure 7-7. SMC Signals for NCS Controlled Accesses.

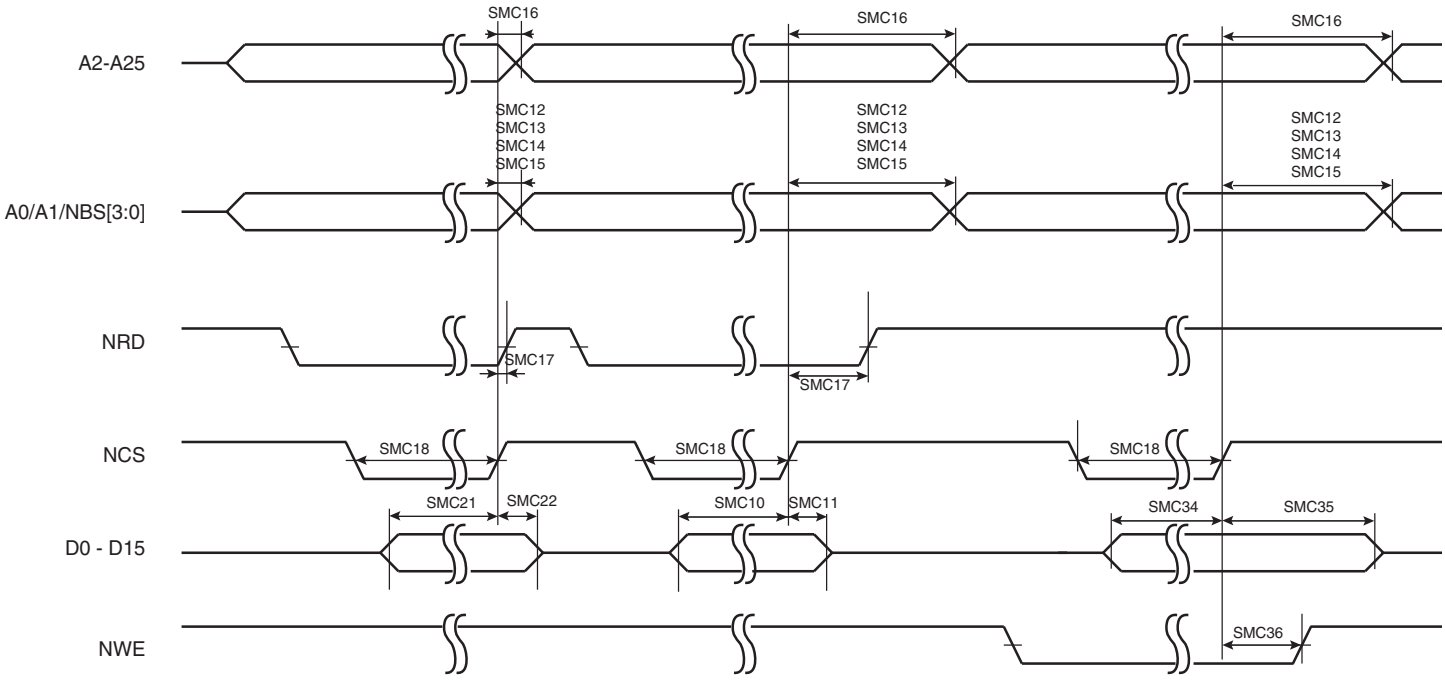
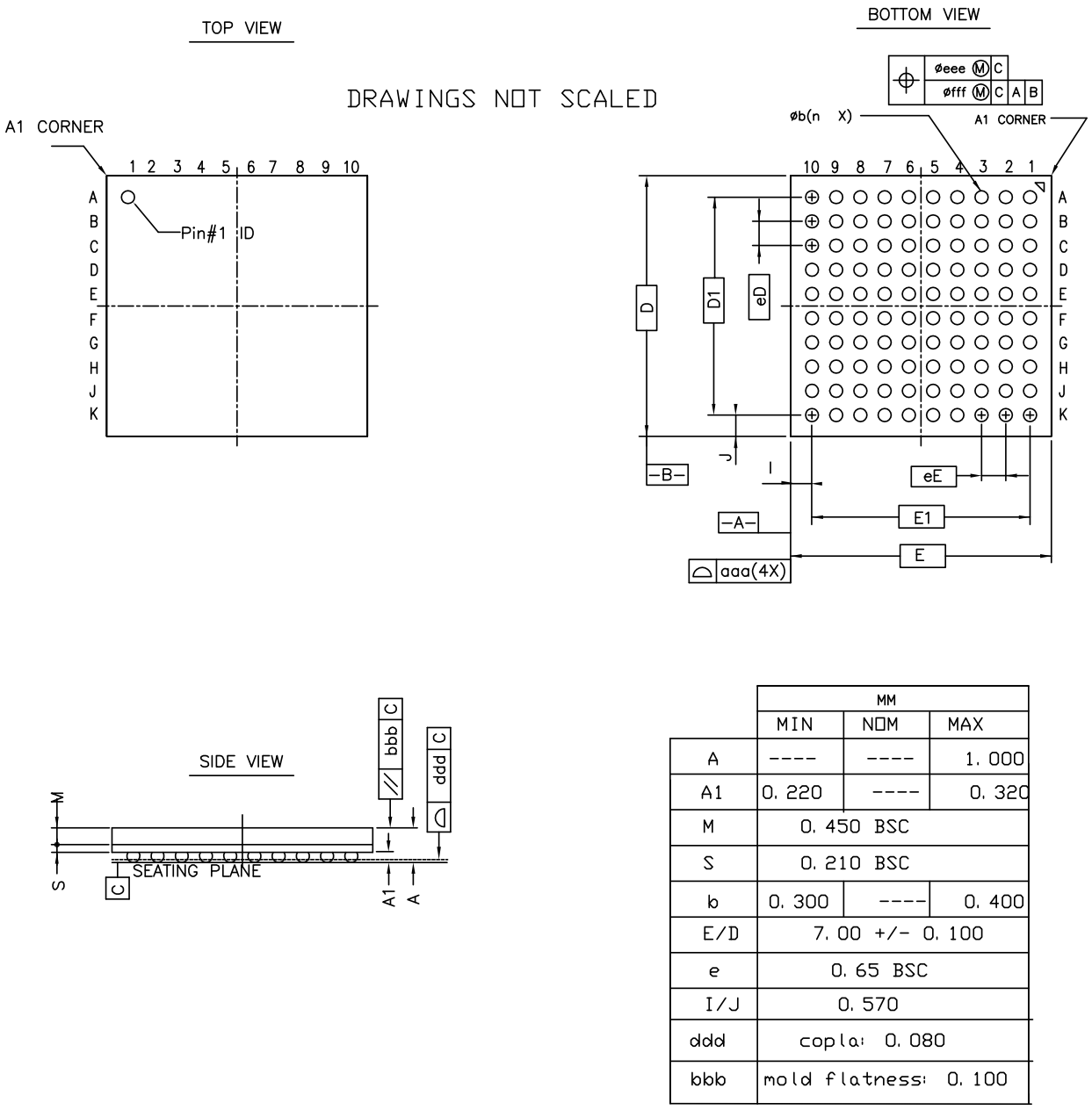


Table 7-36. SDRAM Clock Signal

Symbol	Parameter	Conditions	Min.	Max.	Unit
$SDRAMC_{13}$	Bank Change before SDCK Rising Edge		6.3		ns
$SDRAMC_{14}$	Bank Change after SDCK Rising Edge		2.4		ns
$SDRAMC_{15}$	CAS Low before SDCK Rising Edge		7.4		ns
$SDRAMC_{16}$	CAS High after SDCK Rising Edge		1.9		ns
$SDRAMC_{17}$	DQM Change before SDCK Rising Edge		6.4		ns
$SDRAMC_{18}$	DQM Change after SDCK Rising Edge		2.2		ns
$SDRAMC_{19}$	D0-D15 in Setup before SDCK Rising Edge		9		ns
$SDRAMC_{20}$	D0-D15 in Hold after SDCK Rising Edge		0		ns
$SDRAMC_{23}$	SDWE Low before SDCK Rising Edge		7.6		ns
$SDRAMC_{24}$	SDWE High after SDCK Rising Edge		1.8		ns
$SDRAMC_{25}$	D0-D15 Out Valid before SDCK Rising Edge		7.1		ns
$SDRAMC_{26}$	D0-D15 Out Valid after SDCK Rising Edge		1.5		ns

Figure 8-3. VFBGA-100 package drawing



	MM		
	MIN	NOM	MAX
A	----	----	1.000
A1	0.220	----	0.320
M	0.450 BSC		
S	0.210 BSC		
b	0.300	----	0.400
E/D	7.00 +/- 0.100		
e	0.65 BSC		
I/J	0.570		
ddd	copla: 0.080		
bbb	mold flatness: 0.100		

For higher polling time, the software must freeze the pipe for the desired period in order to prevent any "extra" token.

10.1.5 ADC

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.

10.1.6 USART

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

The LIN ID is not transmitted in mode PDCM='0'
Fix/Workaround
Using USART in mode LIN master with the PDCM bit = '0', the LINID written at the first address of the transmit buffer is not used. The LINID must be written in the LINIR register, after the configuration and start of the PDCA transfer. Writing the LINID in the LINIR register will start the transfer whenever the PDCA transfer is ready.

The LINID interrupt is only available for the header reception and not available for the header transmission
Fix/Workaround
None.

USART LIN mode is not functional with the PDCA if PDCM bit in LINMR register is set to 1
If a PDCA transfer is initiated in USART LIN mode with PDCM bit set to 1, the transfer never starts.
Fix/Workaround
Only use PDCM=0 configuration with the PDCA transfer.

10.1.7 SPI

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).

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.

TWIS stretch on Address match error
When the TWIS stretches TWCK due to a slave address match, it also holds TWD low for the same duration if it is to be receiving data. When TWIS releases TWCK, it releases TWD at the same time. This can cause a TWI timing violation.
Fix/Workaround
None.

10.1.14 SSC

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.

10.1.15 FLASHC

Corrupted read in flash may happen after fuses write or erase operations (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands)
After a flash fuse write or erase operation (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands), reading (data read or code fetch) in flash may fail. This may lead to an exception or to other errors derived from this corrupted read access.
Fix/Workaround
Before the flash fuse write or erase operation, enable the flash high speed mode (FLASHC HSEN command). The flash fuse write or erase operations (FLASHC LP, UP, WGPB, EGPB, SSB, PGPFB, EAGPF commands) must be issued from RAM or through the EBI. After these commands, read 3 times one flash page initialized to 00h. Disable the flash high speed mode (FLASHC HSDIS command). It is then possible to safely read or code fetch the flash.

10.2 Rev. E

10.2.1 General

Devices cannot operate with CPU frequency higher than 66MHz in 1WS and 36MHz in 0WS
Fix/Workaround
None

Increased Power Consumption in VDDIO in sleep modes
If the OSC0 is enabled in crystal mode when entering a sleep mode where the OSC0 is disabled, this will lead to an increased power consumption in VDDIO.
Fix/Workaround
Disable the OSC0 through the System Control Interface (SCIF) before going to any sleep mode where the OSC0 is disabled, or pull down or up XIN0 and XOUT0 with 1 Mohm resistor.

Power consumption in static mode The power consumption in static mode can be up to 330µA on some parts (typical at 25°C)

Fix/Workaround
Set to 1b bit CORRS4 of the ECCHRS mode register (MD). In C-code: *((volatile int*) (0xFFFE2404))= 0x400.

DMACA data transfer fails when CTLx.SRC_TR_WIDTH is not equal to CTLx.DST_TR_WIDTH
Fix/Workaround
For any DMACA transfer make sure CTLx.SRC_TR_WIDTH = CTLx.DST_TR_WIDTH.

3.3V supply monitor is not available
FGPFRLO[30:29] are reserved and should not be used by the application.
Fix/Workaround
None.

Service access bus (SAB) can not access DMACA registers
Fix/Workaround
None.

10.2.2 Processor and Architecture

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.

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.

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.

10.2.3 MPU

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.

10.2.4 USB

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).

10.3.5 ADC

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.

10.3.6 USART

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

The LIN ID is not transmitted in mode PDCM='0'
Fix/Workaround
Using USART in mode LIN master with the PDCM bit = '0', the LINID written at the first address of the transmit buffer is not used. The LINID must be written in the LINIR register, after the configuration and start of the PDCA transfer. Writing the LINID in the LINIR register will start the transfer whenever the PDCA transfer is ready.

The LINID interrupt is only available for the header reception and not available for the header transmission
Fix/Workaround
None.

USART LIN mode is not functional with the PDCA if PDCM bit in LINMR register is set to 1
If a PDCA transfer is initiated in USART LIN mode with PDCM bit set to 1, the transfer never starts.
Fix/Workaround
Only use PDCM=0 configuration with the PDCA transfer.

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.

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.

11.7 Rev. B – 08/09

1. Updated the datasheet with new device AT32UC3A4.

11.8 Rev. A – 03/09

1. Initial revision.