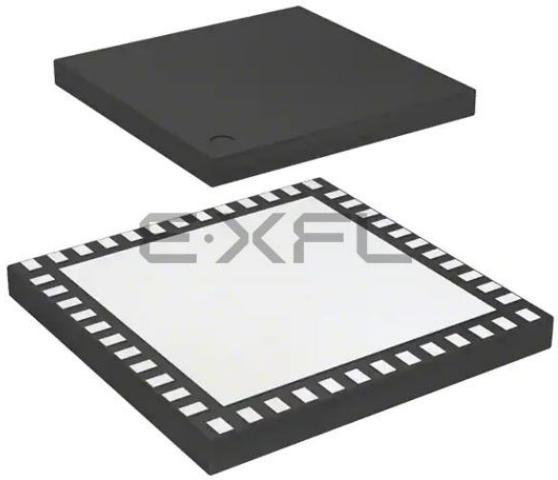


Welcome to E-XFL.COM

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	Obsolete
Core Processor	AVR
Core Size	32-Bit Single-Core
Speed	50MHz
Connectivity	I²C, SPI, UART/USART
Peripherals	Brown-out Detect/Reset, DMA, POR, PWM, WDT
Number of I/O	36
Program Memory Size	64KB (64K x 8)
Program Memory Type	FLASH
EEPROM Size	-
RAM Size	16K x 8
Voltage - Supply (Vcc/Vdd)	1.62V ~ 3.6V
Data Converters	A/D 8x12b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	48-UFLGA Exposed Pad
Supplier Device Package	48-TLLGA (5.5x5.5)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at32uc3l064-d3ht

7.7.30 PDCA Version Register

Name: VERSION

Access Type: Read-only

Offset: 0x834

Reset Value: -

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-

23	22	21	20	19	18	17	16
-	-	-	-	VARIANT			

15	14	13	12	11	10	9	8
-	-	-	-	VERSION[11:8]			

7	6	5	4	3	2	1	0
VERSION[7:0]							

- **VARIANT: Variant Number**
  Reserved. No functionality associated.
- **VERSION: Version Number**
  Version number of the module. No functionality associated.

To change from one kind of default master to another, the Bus Matrix user interface provides the Slave Configuration Registers, one for each slave, that set a default master for each slave. The Slave Configuration Register contains two fields: DEFMSTR_TYPE and FIXED_DEFMSTR. The 2-bit DEFMSTR_TYPE field selects the default master type (no default, last access master, fixed default master), whereas the 4-bit FIXED_DEFMSTR field selects a fixed default master provided that DEFMSTR_TYPE is set to fixed default master. Please refer to the Bus Matrix user interface description.

10.4.1.1 No Default Master

At the end of the current access, if no other request is pending, the slave is disconnected from all masters. No Default Master suits low-power mode.

10.4.1.2 Last Access Master

At the end of the current access, if no other request is pending, the slave remains connected to the last master that performed an access request.

10.4.1.3 Fixed Default Master

At the end of the current access, if no other request is pending, the slave connects to its fixed default master. Unlike last access master, the fixed master does not change unless the user modifies it by a software action (field FIXED_DEFMSTR of the related SCFG).

10.4.2 Arbitration

The Bus Matrix provides an arbitration mechanism that reduces latency when conflict cases occur, i.e. when two or more masters try to access the same slave at the same time. One arbiter per HSB slave is provided, thus arbitrating each slave differently.

The Bus Matrix provides the user with the possibility of choosing between 2 arbitration types for each slave:

1. Round-Robin Arbitration (default)
2. Fixed Priority Arbitration

This is selected by the ARBT field in the Slave Configuration Registers (SCFG).

Each algorithm may be complemented by selecting a default master configuration for each slave.

When a re-arbitration must be done, specific conditions apply. This is described in “Arbitration Rules” .

10.4.2.1 Arbitration Rules

Each arbiter has the ability to arbitrate between two or more different master requests. In order to avoid burst breaking and also to provide the maximum throughput for slave interfaces, arbitration may only take place during the following cycles:

1. Idle Cycles: When a slave is not connected to any master or is connected to a master which is not currently accessing it.
2. Single Cycles: When a slave is currently doing a single access.
3. End of Burst Cycles: When the current cycle is the last cycle of a burst transfer. For defined length burst, predicted end of burst matches the size of the transfer but is managed differently for undefined length burst. This is described below.
4. Slot Cycle Limit: When the slot cycle counter has reached the limit value indicating that the current master access is too long and must be broken. This is described below.

- Undefined Length Burst Arbitration

In order to avoid long slave handling during undefined length bursts (INCR), the Bus Matrix provides specific logic in order to re-arbitrate before the end of the INCR transfer. A predicted end of burst is used as a defined length burst transfer and can be selected among the following five possibilities:

1. Infinite: No predicted end of burst is generated and therefore INCR burst transfer will never be broken.
2. One beat bursts: Predicted end of burst is generated at each single transfer inside the INCP transfer.
3. Four beat bursts: Predicted end of burst is generated at the end of each four beat boundary inside INCR transfer.
4. Eight beat bursts: Predicted end of burst is generated at the end of each eight beat boundary inside INCR transfer.
5. Sixteen beat bursts: Predicted end of burst is generated at the end of each sixteen beat boundary inside INCR transfer.

This selection can be done through the ULBT field in the Master Configuration Registers (MCFG).

- Slot Cycle Limit Arbitration

The Bus Matrix contains specific logic to break long accesses, such as very long bursts on a very slow slave (e.g., an external low speed memory). At the beginning of the burst access, a counter is loaded with the value previously written in the SLOT_CYCLE field of the related Slave Configuration Register (SCFG) and decreased at each clock cycle. When the counter reaches zero, the arbiter has the ability to re-arbitrate at the end of the current byte, halfword, or word transfer.

10.4.2.2 Round-Robin Arbitration

This algorithm allows the Bus Matrix arbiters to dispatch the requests from different masters to the same slave in a round-robin manner. If two or more master requests arise at the same time, the master with the lowest number is first serviced, then the others are serviced in a round-robin manner.

There are three round-robin algorithms implemented:

1. Round-Robin arbitration without default master
2. Round-Robin arbitration with last default master
3. Round-Robin arbitration with fixed default master

- Round-Robin Arbitration without Default Master

This is the main algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to dispatch requests from different masters to the same slave in a pure round-robin manner. At the end of the current access, if no other request is pending, the slave is disconnected from all masters. This configuration incurs one latency cycle for the first access of a burst. Arbitration without default master can be used for masters that perform significant bursts.

- Round-Robin Arbitration with Last Default Master

This is a biased round-robin algorithm used by Bus Matrix arbiters. It allows the Bus Matrix to remove the one latency cycle for the last master that accessed the slave. At the end of the cur-

13.6.23 32kHz RC Oscillator Configuration Register

Name: RC32KCR

Access Type: Read/Write

Reset Value: 0x00000000

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

- **EN: RC32K Enable**
  0: The 32 kHz RC oscillator is disabled.
  1: The 32 kHz RC oscillator is enabled.

Note that this register is protected by a lock. To write to this register the UNLOCK register has to be written first. Please refer to the UNLOCK register description for details.

If ADD is ‘1’, the prescaler frequency is increased:

$$f_{TUNED} = f_0 \left(1 + \frac{1}{\text{roundup}\left(\frac{256}{VALUE}\right) \cdot (2^{EXP}) - 1} \right)$$

Note that for these formulas to be within an error of 0.01%, it is recommended that the prescaler bit that is used as the clock for the counter (selected by CR.PSEL) or to trigger the periodic interrupt (selected by PIRn.INSEL) be bit 6 or higher.

14.5.8 Synchronization

As the prescaler and counter operate asynchronously from the user interface, the AST needs a few clock cycles to synchronize the values written to the CR, CV, SCR, WER, EVE, EVD, PIRn, ARn, and DTR registers. The Busy bit in the Status Register (SR.BUSY) indicates that the synchronization is ongoing. During this time, writes to these registers will be discarded and reading will return a zero value.

Note that synchronization takes place also if the prescaler is clocked from CLK_AST.

21.9.14 Version Register (VR)

Name: VR

Access Type: Read-only

Offset: 0x34

Reset Value: -

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-

23	22	21	20	19	18	17	16
-	-	-	-	VARIANT			

15	14	13	12	11	10	9	8
-	-	-	-	VERSION [11:8]			

7	6	5	4	3	2	1	0
VERSION [7:0]							

- **VARIANT: Variant Number**
  Reserved. No functionality associated.
- **VERSION: Version Number**
  Version number of the module. No functionality associated.

22. Two-wire Slave Interface (TWIS)

Rev.: 1.1.2.1

22.1 Features

- **Compatible with I²C standard**
  - **Transfer speeds of 100 and 400 kbit/s**
  - **7 and 10-bit and General Call addressing**
- **Compatible with SMBus standard**
  - **Hardware Packet Error Checking (CRC) generation and verification with ACK response**
  - **SMBALERT interface**
  - **25 ms clock low timeout delay**
  - **25 ms slave cumulative clock low extend time**
- **Compatible with PMBus**
- **DMA interface for reducing CPU load**
- **Arbitrary transfer lengths, including 0 data bytes**
- **Optional clock stretching if transmit or receive buffers not ready for data transfer**
- **32-bit Peripheral Bus interface for configuration of the interface**

22.2 Overview

The Atmel Two-wire Slave Interface (TWIS) interconnects components on a unique two-wire bus, made up of one clock line and one data line with speeds of up to 400 kbit/s, based on a byte-oriented transfer format. It can be used with any Atmel Two-wire Interface bus, I²C, or SMBus-compatible master. The TWIS is always a bus slave and can transfer sequential or single bytes.

Below, Table 22-1 lists the compatibility level of the Atmel Two-wire Slave Interface and a full I²C compatible device.

Table 22-1. Atmel TWIS Compatibility with I²C Standard

I²C Standard	Atmel TWIS
Standard-mode (100 kbit/s)	Supported
Fast-mode (400 kbit/s)	Supported
7 or 10 bits Slave Addressing	Supported
START BYTE(1)	Not Supported
Repeated Start (Sr) Condition	Supported
ACK and NAK Management	Supported
Slope control and input filtering (Fast mode)	Supported
Clock stretching	Supported

Note: 1. START + b000000001 + Ack + Sr

In I²C mode:

- The address in CR.ADR is checked for address match if CR.SMATCH is one.
- The General Call address is checked for address match if CR.GCMATCH is one.

In SMBus mode:

- The address in CR.ADR is checked for address match if CR.SMATCH is one.
- The Alert Response Address is checked for address match if CR.SMAL is one.
- The Default Address is checked for address match if CR.SMDA is one.
- The Host Header Address is checked for address match if CR.SMHH is one.

22.8.2.4 Clock Stretching

Any slave or bus master taking part in a transfer may extend the TWCK low period at any time. The TWIS may extend the TWCK low period after each byte transfer if CR.STREN is one and:

- Module is in slave transmitter mode, data should be transmitted, but THR is empty, or
- Module is in slave receiver mode, a byte has been received and placed into the internal shifter, but the Receive Holding Register (RHR) is full, or
- Stretch-on-address-match bit CR.SOAM=1 and slave was addressed. Bus clock remains stretched until all address match bits in the Status Register (SR) have been cleared.

If CR.STREN is zero and:

- Module is in slave transmitter mode, data should be transmitted but THR is empty: Transmit the value present in THR (the last transmitted byte or reset value), and set SR.URUN.
- Module is in slave receiver mode, a byte has been received and placed into the internal shifter, but RHR is full: Discard the received byte and set SR.ORUN.

22.8.2.5 Bus Errors

If a bus error (misplaced START or STOP) condition is detected, the SR.BUSERR bit is set and the TWIS waits for a new START condition.

22.8.3 Slave Transmitter Mode

If the TWIS matches an address in which the R/$\overline{W}$ bit in the TWI address phase transfer is set, it will enter slave transmitter mode and set the SR.TRA bit (note that SR.TRA is set one CLK_TWIS cycle after the relevant address match bit in the same register is set).

After the address phase, the following actions are performed:

1. If SMBus mode and PEC is used, NBYTES must be set up with the number of bytes to transmit. This is necessary in order to know when to transmit the PEC byte. NBYTES can also be used to count the number of bytes received if using DMA.
2. Byte to transmit depends on I²C/SMBus mode and CR.PEC:
   - If in I²C mode or CR.PEC is zero or NBYTES is non-zero: The TWIS waits until THR contains a valid data byte, possibly stretching the low period of TWCK. After THR contains a valid data byte, the data byte is transferred to a shifter, and then SR.TXRDY is changed to one because the THR is empty again.
   - SMBus mode and CR.PEC is one: If NBYTES is zero, the generated PEC byte is automatically transmitted instead of a data byte from THR. TWCK will not be stretched by the TWIS.
3. The data byte in the shifter is transmitted.

Writing a one to this bit resets the TWIS.

- **STREN: Clock Stretch Enable**
  0: Disables clock stretching if RHR/THR buffer full/empty. May cause over/underrun.
  1: Enables clock stretching if RHR/THR buffer full/empty.
- **GCMATCH: General Call Address Match**
  0: Causes the TWIS not to acknowledge the General Call Address.
  1: Causes the TWIS to acknowledge the General Call Address.
- **SMATCH: Slave Address Match**
  0: Causes the TWIS not to acknowledge the Slave Address.
  1: Causes the TWIS to acknowledge the Slave Address.
- **SMEN: SMBus Mode Enable**
  0: Disables SMBus mode.
  1: Enables SMBus mode.
- **SEN: Slave Enable**
  0: Disables the slave interface.
  1: Enables the slave interface.

24.7.5 Channel Register A

Name: RA

Access Type: Read-only if CMRn.WAVE = 0, Read/Write if CMRn.WAVE = 1

Offset: 0x14 + n * 0X40

Reset Value: 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	-	-	-	-

23	22	21	20	19	18	17	16
-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8
RA[15:8]							

7	6	5	4	3	2	1	0
RA[7:0]							

- **RA: Register A**
  RA contains the Register A value in real time.

26.9.3 ADC Configuration Register

Name: ACR

Access Type: Read/Write

Offset: 0x08

Reset Value: 0x00000000

31	30	29	28	27	26	25	24
-	-	-	-	SHTIM			

23	22	21	20	19	18	17	16
-	STARTUP						

15	14	13	12	11	10	9	8
-	-	PRESCAL					

7	6	5	4	3	2	1	0
-	-	RES		-	-	-	SLEEP

- **SHTIM: Sample & Hold Time for ADC Channels**

  $T_{SAMPLE\&HOLD} = (SHTIM + 3) \cdot T_{CLK\_ADC}$

- **STARTUP: Startup Time**

  $_{TARTUP} = (STARTUP + 1) \cdot 8 \cdot T_{CLK\_A}$

- **PRESCAL: Prescaler Rate Selection**

  $T_{CLK\_ADC} = (PRESCAL + 1) \cdot 2 \cdot T_{CLK\_ADCIFB}$

- **RES: Resolution Selection**

  0: 8-bit resolution.

  1: 10-bit resolution.

  2: 11-bit resolution, interpolated.

  3: 12-bit resolution, interpolated.

- **SLEEP: ADC Sleep Mode**

  0: ADC Sleep Mode is disabled.

  1: ADC Sleep Mode is enabled.

26.9.13 Parameter Register

Name: PARAMETER

Access Type: Read-only

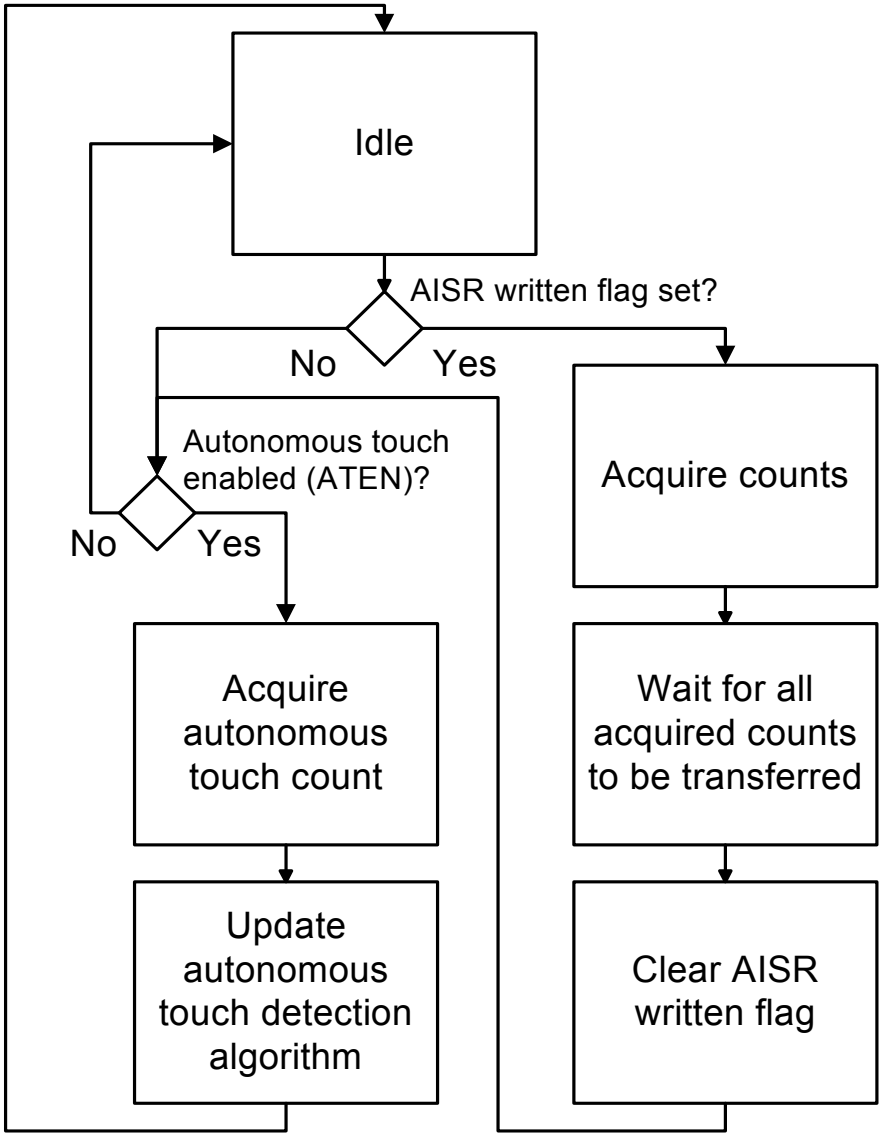
Offset: 0x30

Reset Value: 0x00000000

31	30	29	28	27	26	25	24
CH31	CH30	CH29	CH28	CH27	CH26	CH25	CH24
23	22	21	20	19	18	17	16
CH23	CH22	CH21	CH20	CH19	CH18	CH17	CH16
15	14	13	12	11	10	9	8
CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8
7	6	5	4	3	2	1	0
CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0

- **CHn: Channel n Implemented**
  0: The corresponding channel is not implemented.
  1: The corresponding channel is implemented.

Figure 28-4. CAT Acquisition and Processing Sequence



28.6.6 Spread Spectrum Sensor Drive

To reduce electromagnetic compatibility issues, the capacitive sensors can be driven with a spread spectrum signal. To enable spread spectrum drive for a specific acquisition type, the user must write a one to the SPREAD bit in the appropriate Configuration Register 1 (MGCFG1, ATCFG1, TGACFG1, or TGBCFG1).

During spread spectrum operation, the length of each pulse within a burst is varied in a deterministic pattern, so that the exact same burst pattern is used for a specific burst length. The maximum spread is determined by the MAXDEV field in the Spread Spectrum Configuration Register (SSCFG) register. The prescaler divisor is varied in a sawtooth pattern from (2(DIV+1))-MAXDEV to (2(DIV+1))+MAXDEV and then back to (2(DIV+1))-MAXDEV. For example, if DIV is 2 and MAXDEV is 3, the prescaler divisor will have the following sequence: 6, 7, 8,

28.8 Module Configuration

The specific configuration the CAT module is listed in the following tables.The module bus clocks listed here are connected to the system bus clocks. Please refer to the Power Manager chapter for details.

Table 28-4. CAT Configuration

Feature	CAT
Number of touch sensors/Size of matrix	Allows up to 17 touch sensors, or up to 16 by 8 matrix sensors to be interfaced.

Table 28-5. CAT Clocks

Clock Name	Description
CLK_CAT	Clock for the CAT bus interface
GCLK	The generic clock used for the CAT is GCLK4

Table 28-6. Register Reset Values

Register	Reset Value
VERSION	0x00000200
PARAMETER	0x0001FFFF

28.8.1 Resistive Drive

By default, the CAT pins are driven with normal I/O drive properties. Some of the CSA and CSB pins can optionally drive with a 1k output resistance for improved EMC.

To enable resistive drive on a pin, the user must write a one to the corresponding bit in the CSA Resistor Control Register (CSARES) or CSB Resistor Control Register (CSBRES) register.

30.7.7 Receive Holding Register

Name: RHR

Access Type: Read-only

Offset: 0x18

Reset Value: 0x00000000

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

- **RXDATA: Received Data**
  The last byte received.

30.7.11 Clock Request Register

Name: CLKR

Access Type: Read/Write

Offset: 0x28

Reset Value: 0x00000000

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

- **CLKEN: Clock Enable**
  0: The aWire clock is disabled.
  1: The aWire clock is enabled.
  Writing a zero to this bit will disable the aWire clock.
  Writing a one to this bit will enable the aWire clock.

31.3.7.1 *Cyclic Redundancy Check (CRC)*

The MSU can be used to automatically calculate the CRC of a block of data in memory. The MSU will then read out each word in the specified memory block and report the CRC32-value in an MSU register.

31.3.7.2 *NanoTrace*

The MSU additionally supports NanoTrace. This is a 32-bit AVR-specific feature, in which trace data is output to memory instead of the AUX port. This allows the trace data to be extracted by the debugger through the SAB, enabling trace features for aWire- or JTAG-based debuggers. The user must write MSU registers to configure the address and size of the memory block to be used for NanoTrace. The NanoTrace buffer can be anywhere in the physical address range, including internal and external RAM, through an EBI, if present. This area may not be used by the application running on the CPU.

31.3.8 AUX-based Debug Features

Utilizing the Auxiliary (AUX) port gives access to a wide range of advanced debug features. Of prime importance are the trace features, which allow an external debugger to receive continuous information on the program execution in the CPU. Additionally, Event In and Event Out pins allow external events to be correlated with the program flow.

Debug tools utilizing the AUX port should connect to the device through a Nexus-compliant Mictor-38 connector, as described in the AVR32UC Technical Reference manual. This connector includes the JTAG signals and the RESET_N pin, giving full access to the programming and debug features in the device.

Table 31-11. Instruction Description (Continued)

Instruction	Description
DR Size	Shows the number of bits in the data register chain when this instruction is active. Example: 34 bits
DR input value	Shows which bit pattern to shift into the data register in the Shift-DR state when this instruction is active. Multiple such lines may exist, e.g., to distinguish between reads and writes. Example: aaaaaaar xxxxxxxx xxxxxxxx xxxxxxxx xx
DR output value	Shows the bit pattern shifted out of the data register in the Shift-DR state when this instruction is active. Multiple such lines may exist, e.g., to distinguish between reads and writes. Example: xx **xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxeb**

31.5.2 Public JTAG Instructions

The JTAG standard defines a number of public JTAG instructions. These instructions are described in the sections below.

31.5.2.1 IDCODE

This instruction selects the 32 bit Device Identification register (DID) as Data Register. The DID register consists of a version number, a device number, and the manufacturer code chosen by JEDEC. This is the default instruction after a JTAG reset. Details about the DID register can be found in the module configuration section at the end of this chapter.

Starting in Run-Test/Idle, the Device Identification register is accessed in the following way:

1. Select the IR Scan path.
2. In Capture-IR: The IR output value is latched into the shift register.
3. In Shift-IR: The instruction register is shifted by the TCK input.
4. Return to Run-Test/Idle.
5. Select the DR Scan path.
6. In Capture-DR: The IDCODE value is latched into the shift register.
7. In Shift-DR: The IDCODE scan chain is shifted by the TCK input.
8. Return to Run-Test/Idle.

Table 31-12. IDCODE Details

Instructions	Details
IR input value	**00001** (0x01)
IR output value	p0001
DR Size	32
DR input value	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
DR output value	Device Identification Register

31.5.2.2 SAMPLE_PRELOAD

This instruction takes a snap-shot of the input/output pins without affecting the system operation, and pre-loading the scan chain without updating the DR-latch. The boundary-scan chain is selected as Data Register.

Starting in Run-Test/Idle, the Device Identification register is accessed in the following way:

Fix/Workaround
If the target frequency is below 30MHz, use a max step size (DFLL0MAXSTEP.MAXSTEP) of seven or lower.

7. **Generic clock sources are kept running in sleep modes**
If a clock is used as a source for a generic clock when going to a sleep mode where clock sources are stopped, the source of the generic clock will be kept running. Please refer to Power Manager chapter for details about sleep modes.
Fix/Workaround
Disable generic clocks before going to sleep modes where clock sources are stopped to save power.

8. **DFLL clock is unstable with a fast reference clock**
The DFLL clock can be unstable when a fast clock is used as a reference clock in closed loop mode.
Fix/Workaround
Use the 32KHz crystal oscillator clock, or a clock with a similar frequency, as DFLLIF reference clock.

9. **DFLLIF indicates coarse lock too early**
The DFLLIF might indicate coarse lock too early, the DFLL will lose coarse lock and regain it later.
Fix/Workaround
Use max step size (DFLL0MAXSTEP.MAXSTEP) of 4 or higher.

10. **DFLLIF dithering does not work**
The DFLLIF dithering does not work.
Fix/Workaround
None.

11. **DFLLIF might lose fine lock when dithering is disabled**
When dithering is disabled and fine lock has been acquired, the DFLL might lose the fine lock resulting in up to 20% over-/undershoot.
Fix/Workaround
Solution 1: When the DFLL is used as main clock source, the target frequency of the DFLL should be 20% below the maximum operating frequency of the CPU. Don't use the DFLL as clock source for frequency sensitive applications.
Solution 2: Do not use the DFLL in closed loop mode.

12. **GCLK5 is non-functional**
GCLK5 is non-functional.
Fix/Workaround
None.

13. **BRIFA is non-functional**
BRIFA is non-functional.
Fix/Workaround
None.

14. **SCIF VERSION register reads 0x100**
SCIFVERSION register reads 0x100 instead of 0x102.
Fix/Workaround
None.