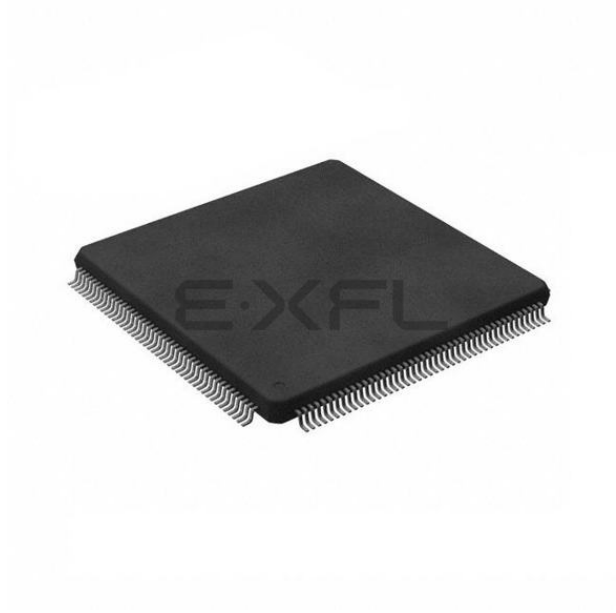

Welcome to E-XFL.COM

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	ARM7®
Core Size	16/32-Bit
Speed	33MHz
Connectivity	EBI/EMI, SPI, UART/USART
Peripherals	POR, WDT
Number of I/O	58
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	8K x 8
Voltage - Supply (Vcc/Vdd)	2.7V ~ 3.6V
Data Converters	A/D 8x10b; D/A 2x10b
Oscillator Type	Internal
Operating Temperature	-40°C ~ 85°C (TA)
Mounting Type	Surface Mount
Package / Case	176-LQFP
Supplier Device Package	176-LQFP (24x24)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/at91m55800a-33au

Edition 2014-10

Published by
Infineon Technologies AG
81726 Munich, Germany

Legal Disclaimer

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation, warranties of non-infringement of intellectual property rights of any third party.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

16-Bit

Architecture

XE166 Derivatives

16-Bit Single-Chip
Real Time Signal Controller
XE166 Family / Alpha Line

Errata Sheet

V1.6 2014-10

Microcontrollers

3 Current Documentation

The Infineon XE166 Family comprises device types from the XE164 Series and the XE167 Series.

Device	XE164x, XE167x
Marking/Step	AB, AC
Package	PG-LQFP-100**,** PG-LQFP-144

This Errata Sheet refers to the following documentation:

- XE166 Derivatives User's Manual
- XE164 Data Sheet
- XE167 Data Sheet
- Documentation Addendum (if applicable)

Make sure you always use the corresponding documentation for this device available in category 'Documents' at **www.infineon.com/xe166** .

The specific test conditions for EES and ES are documented in a separate Status Sheet.

Note: Devices marked with EES or ES are engineering samples which may not be completely tested in all functional and electrical characteristics, therefore they should be used for evaluation only.

Table 2 Functional Deviations (cont'd)

Functional Deviation	Short Description	Chg	Pg
MultiCAN_AI.042	Clear MSGVAL during transmit acceptance filtering		29
MultiCAN_AI.043	Dealloc Previous Obj		29
MultiCAN_AI.044	RxFIFO Base SDT		30
MultiCAN_AI.045	OVIE Unexpected Interrupt		30
MultiCAN_AI.046	Transmit FIFO base Object position		31
MultiCAN_TC.025	RXUPD behavior		31
MultiCAN_TC.026	MultiCAN Timestamp Function		32
MultiCAN_TC.027	MultiCAN Tx Filter Data Remote		32
MultiCAN_TC.028	SDT behavior		32
MultiCAN_TC.029	Tx FIFO overflow interrupt not generated		33
MultiCAN_TC.030	Wrong transmit order when CAN error at start of CRC transmission		34
MultiCAN_TC.031	List Object Error wrongly triggered		35
MultiCAN_TC.032	MSGVAL wrongly cleared in SDT mode		35
MultiCAN_TC.035	Different bit timing modes		36
MultiCAN_TC.037	Clear MSGVAL		37
MultiCAN_TC.038	Cancel TXRQ		38
OCDS_X.003	Peripheral Debug Mode Settings cleared by Reset		38
RESET_X.002	Startup Mode Selection is not Valid in SCU_STSTAT.HWCFG		39
RESET_X.003	P2.[2:0] and P10.[12:0] Switch to Input		39
RESET_X.004	Sticky "Register Access Trap" forces device into power-save mode after reset.		40
RTC_X.003	Interrupt Generation in Asynchronous Mode		40
USIC_AI.003	TCSRL.SOF and TCSRL.EOF not cleared after a transmission is started		41
USIC_AI.004	Receive shifter baudrate limitation		41
USIC_AI.005	Only 7 data bits are generated in IIC mode when TBUF is loaded in SDA hold time		41

Table 4 Application Hints (cont'd)

Hint	Short Description	Chg	Pg
USIC_AI.H002	Configuration of USIC Port Pins		56
USIC_AI.H003	PSR.RXIDLE Cleared by Software		57

4.4 Documentation Updates

Table 5 gives a short description of the documentation updates.

Table 5 Documentation Updates

Documentation Updates	Short Description	Chg	Pg
EBC_X.D001	Visibility of Internal LXBus Cycles on External Address Bus		58
ID_X.D001	Identification Register		58
RESET_X.D001	Reset Types of Trap Registers		59
SCU_X.D007	SCU Interrupts Enabled After Reset		59
StartUp_X.D002	External Start-Up Mode Selection by Configuration Pins		59
USIC_X.D003	USIC0 Channel 1 Connection DX0D and DOUT		60
XTAL_X.D001	Input Voltage Amplitude V_{AX1} on XTAL1		60

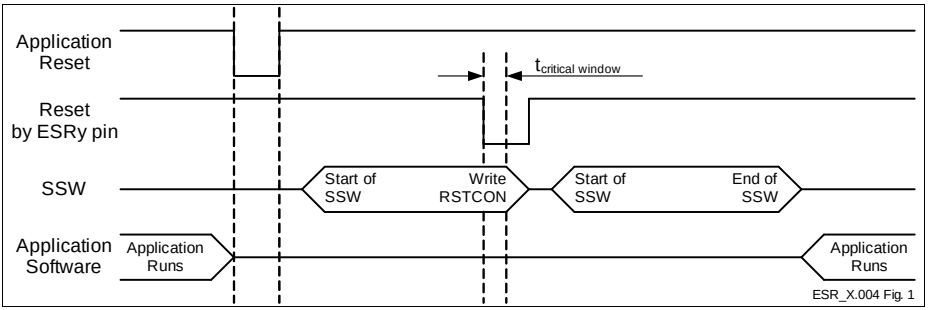

Figure 1 Critical application reset sequence

Workaround

- Initialize SCU_RSTCONx registers by user software after any reset, or
- assure that a second application reset request with an ESR pin does not occur during the critical time window.

FLASH_X.008 Flash Read after Flash Erase Command

Under certain conditions all Flash erase commands do not work correctly. After erasing, all erased bits must be programmed with new data or with all-zero data before reading any data from the addressed sector is allowed.

Workarounds

1. Erase a range of Flash memory and program it completely with new data before reading. This is the fastest solution.
   Additional hint: A Flash driver could implement a programming function that performs first an “Erase Page” and uses directly thereafter “Program Page” to program the data of this page. The Flash driver wouldn’t need any separate erase function.
2. Erase a range of Flash memory and program it completely with all-zero data. Only after this the range may be read. Data can be programmed later[1)].

1) Please note: only in order to implement this workaround for the noted device steps it is allowed to execute two program commands before erasing it.

GPT12E_X.001 T5/T6 in Counter Mode with BPS2 = $1X_B$

When T5 and/or T6 are configured for counter mode (bit field TxM = 001_B in register GPT12E_TxCON, x = 5, 6), **and** bit field **BPS2 = $1X_B$** in register GPT12E_T6CON, then edge detection for the following count input and control signals does not work correctly:

T5IN, T6IN, T5EUD, T6EUD.

Note: The configuration where T5 counts the overflow/underflow events of T6 is not affected by this problem.

Workaround

Do not set bit field BPS2 = $1X_B$ in register GPT12E_T6CON when T5 and/or T6 are configured for counter mode. Use only settings BPS2 = $0X_B$ when T5 and/or T6 are configured for counter mode.

GPT12E_X.002 Effects of GPT Module Microarchitecture

The present GPT module implementation provides some enhanced features (e.g. block prescalers BPS1, BPS2) while still maintaining timing and functional compatibility with the original implementation in the C166 Family of microcontrollers.

Both of the GPT1 and GPT2 blocks use a finite state machine to control the actions within each block. Since multiple interactions are possible between the timers (T2 .. T6) and register CAPREL, these elements are processed sequentially within each block in different states. However, all actions are normally completed within one basic clock cycle.

The GPT2 state machine has 4 states (2 states when BPS2 = 01_B) and processes T6 before T5. The GPT1 state machine has 8 states (4 states when BPS1 = 01_B) and processes the timers in the order T3 - T2 (all actions except capture) - T4 - T2 (capture).

In the following, two effects of the internal module microarchitecture that may require special consideration in an application are described in more detail.

1.) Reading T3 by Software with T2/T4 in Reload Mode

When T2 or T4 are used to reload T3 on overflow/underflow, and T3 is read by software on the fly, the following unexpected values may be read from T3:

- when T3 is counting **up**, 0000_H or 0001_H may be read from T3 directly after an overflow, although the reload value in T2/T4 is higher (0001_H may be read in particular if BPS1 = 01_B and T3I = 000_B),

Workaround 2

Do not use local register banks, use only global register banks.

Workaround 3

Locate the system stack in a memory other than the DPRAM, e.g. in DSRAM.

INT_X.008 HW Trap during Context Switch in Routine using a Local Bank

When a hardware trap occurs under specific conditions in a routine using a local register bank, the CPU may stall, preventing further code execution. Recovery from this condition can only be made through a hardware or watchdog reset.

All of the following conditions must be present for this problem to occur:

- The routine that is interrupted by the hardware trap is using one of the **local** register banks (bit field PSW.BANK = 10_B or 11_B)
- The system stack is located in the internal dual-ported RAM (DPRAM, locations $0F600_H$ ... $0FDFF_H$)
- The hardware trap occurs in the second half (load phase) of a context switch operation triggered by one of the following actions:
  - a) Execution of the IDLE instruction, or
  - b) Execution of an instruction writing to the Context Pointer register CP (untypical case, because this would mean that the routine using one of the local banks modifies the CP contents of a global bank)

Workaround 1

Locate the system stack in a memory other than the DPRAM, e.g. in DSRAM.

Workaround 2

Do not use local register banks, use only global register banks.

Workaround 3

Condition b) (writing to CP while a local register bank context is selected) is not typical for most applications. If the application implementation already eliminates the possibility for condition b), then only a workaround for condition a) is required.
The workaround for condition a) is to make sure that the IDLE instruction is executed within a code sequence that uses a global register bank context.

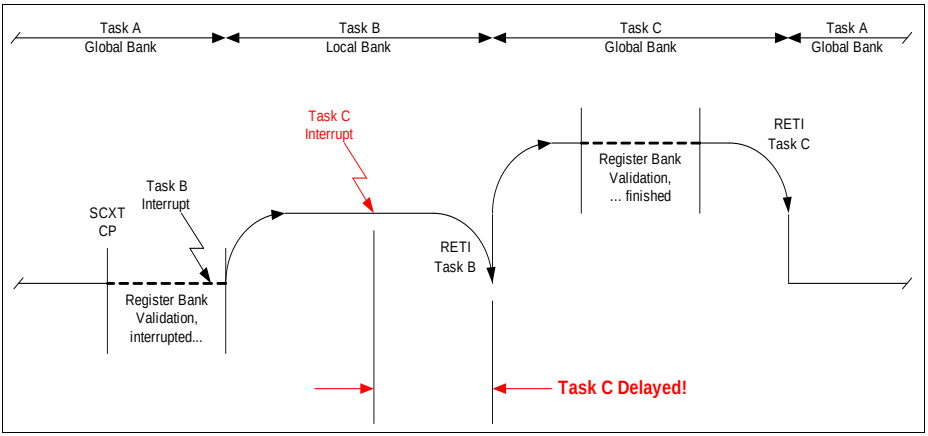

Figure 2 Example for Case 2: Interrupt Service for Task C delayed

Workaround for Case 1

Do not write to the CP register (i.e. modify the context of a global bank) while a local register bank context is selected.

Workaround for Case 2

When using both local and global register banks via the bank selection registers BNKSEL0...3 for interrupts on levels ≥12, ensure that there is no interrupt using a global register bank that has a higher priority than an interrupt using a local register bank.

Example 1:
Local bank interrupts are used on levels 14 and 15, no local bank interrupts on level 12 and 13. In this case, global bank interrupts on level 15 must not be used.

Example 2:
Local bank interrupts are used on level 12. In this case, no global bank interrupts must be used on levels 13, 14, 15.

INT_X.010 HW Traps and Interrupts may get postponed

Under the special conditions described below, a hardware trap (HWTx) and subsequent interrupts, PEC transfers, OCDS service requests (on priority level < 11_H) or class B and class A traps (if HWTx also was class A) may get postponed until the next RETI instruction is executed. If no RETI is executed, these requests may get postponed infinitely.

Both of the following conditions must be fulfilled at the same time when the trigger for the hardware trap HWTx occurs in order to cause the problem:

1. The pipeline is cancelled due to one of the following reasons:
   a) a multiply or divide instruction is followed by a mispredicted conditional (zero-cycle) jump.
   b) a class A hardware trap is triggered quasi-simultaneously with the request for a class B trap (= HWTx), i.e. the trigger for the class A trap arrives before the previously injected TRAP instruction for the class B trap has reached the Execute stage of the pipeline.
      In this case, the class A trap is entered, but when the RETI instruction at the end of the class A trap routine is executed, the pending class B trap (HWTx) is **not** entered, and subsequent interrupts/PECs/class B traps are postponed until the next RETI.
   c) a break is requested by the debugger.
2. The pipeline is stalled in the Execute or Write Back stage due to consecutive writes, or due to a multi-cycle write that is performed to a memory area with wait states (PSRAM, external memory).

Workaround

Disable overrun of pipeline bubbles by setting bit OVRUN (CPUCON2.4) = 0.

MultiCAN_AI.040 Remote frame transmit acceptance filtering error

Correct behaviour:

Assume the MultiCAN message object receives a remote frame that leads to a valid transmit request in the same message object (request of remote answer), then the MultiCAN module prepares for an immediate answer of the remote request. The answer message is arbitrated against the winner of transmit acceptance filtering (without the remote answer) with a respect to the priority class (`MOARn.PRI`).

Wrong behaviour:

Assume the MultiCAN message object receives a remote frame that leads to a valid transmit request in the same message object (request of remote answer), then the MultiCAN module prepares for an immediate answer of the remote request. The answer message is arbitrated against the winner of transmit acceptance filtering (without the remote answer) with a respect to the CAN arbitration rules and not taking the PRI values into account.

If the remote answer is not sent out immediately, then it is subject to further transmit acceptance filtering runs, which are performed correctly.

Workaround

Set MOFCRn.FRREN=1_B and MOFGPRn.CUR to this message object to disable the immediate remote answering.

MultiCAN_AI.041 Dealloc Last Obj

When the last message object is deallocated from a list, then a false list object error can be indicated.

Workaround

- Ignore the list object error indication that occurs after the deallocation of the last message object.

or

- Avoid deallocating the last message object of a list.

MultiCAN_AI.042 Clear MSGVAL during transmit acceptance filtering

Assume all CAN nodes are idle and no writes to MOCTRn of any other message object are performed. When bit MOCTRn.MSGVAL of a message object with valid transmit request is cleared by software, then MultiCAN may not start transmitting even if there are other message objects with valid request pending in the same list.

Workaround

- Do not clear MOCTRn.MSGVAL of any message object during CAN operation. Use bits MOCTRn.RXEN, MOCTRn.TXEN0 instead to disable/reenable reception and transmission of message objects.

or

- Take a dummy message object, that is not allocated to any CAN node. Whenever a transmit request is cleared, set MOCTRm.TXRQ of the dummy message object thereafter. This retriggers the transmit acceptance filtering process.

MultiCAN_AI.043 Dealloc Previous Obj

Assume two message objects m and n (message object n = MOCTRm.PNEXT, i.e. n is the successor of object m in the list) are allocated. If message m is reallocated to another list or to another position while the transmit or receive acceptance filtering run is

MultiCAN_TC.026 MultiCAN Timestamp Function

The timestamp functionality does not work correctly.

Workaround

Do not use timestamp.

MultiCAN_TC.027 MultiCAN Tx Filter Data Remote

Message objects of priority class 2 (MOAR.PRI = 2) are transmitted in the order as given by the CAN arbitration rules. This implies that for 2 message objects which have the same CAN identifier, but different DIR bit, the one with DIR = 1 (send data frame) shall be transmitted before the message object with DIR = 0, which sends a remote frame. The transmit filtering logic of the MultiCAN leads to a reverse order, i.e the remote frame is transmitted first. Message objects with different identifiers are handled correctly.

Workaround

None.

MultiCAN_TC.028 SDT behavior

Correct behavior

Standard message objects:

MultiCAN clears bit MOCTR.MSGVAL after the successful reception/transmission of a CAN frame if bit MOFCR.SDT is set.

Transmit Fifo slave object:

MultiCAN clears bit MOCTR.MSGVAL after the successful reception/transmission of a CAN frame if bit MOFCR.SDT is set. After a transmission, MultiCAN also looks at the respective transmit FIFO base object and clears bit MSGVAL in the base object if bit SDT is set in the base object and pointer MOFGPR.CUR points to MOFGPR.SEL (after the pointer update).

Gateway Destination/Fifo slave object:

MultiCAN clears bit MOCTR.MSGVAL after the storage of a CAN frame into the object (gateway/FIFO action) or after the successful transmission of a CAN frame if bit MOFCR.SDT is set. After a reception, MultiCAN also looks at the respective FIFO base/Gateway source object and clears bit MSGVAL in the base object if bit SDT is set in

Workaround

If Tx FIFO overflow interrupt needed, take the FIFO base object out of the circular list of the Tx message objects. That is to say, just use the FIFO base object for FIFO control, but not to store a Tx message.

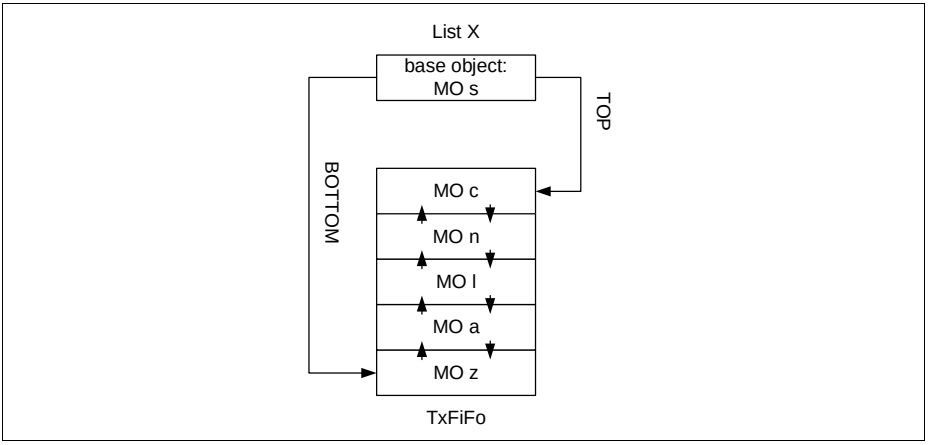

Figure 3 FIFO structure

MultiCAN_TC.030 Wrong transmit order when CAN error at start of CRC transmission

The priority order defined by acceptance filtering, specified in the message objects, define the sequential order in which these messages are sent on the CAN bus. If an error occurs on the CAN bus, the transmissions are delayed due to the destruction of the message on the bus, but the transmission order is kept. However, if a CAN error occurs when starting to transmit the CRC field, the arbitration order for the corresponding CAN node is disturbed, because the faulty message is not retransmitted directly, but after the next transmission of the CAN node.

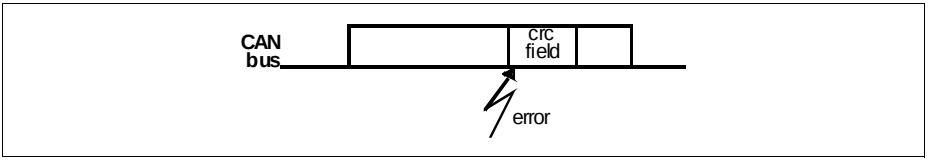

Figure 4

INT_X.H004 SCU Interrupts Enabled After Reset

Following a reset, the SCU interrupts are enabled by default (register SCU_INTDIS = 0000_H). This may lead to interrupt requests being triggered in the SCU immediately, even before user software has begun to execute. In the SCU, multiple interrupt sources are `ORed` to a common interrupt node of the CPU interrupt controller. Due to the "ORing" of multiple interrupt sources, only one interrupt request to the interrupt controller will be generated if multiple sources at the input of this OR gate are active at the same time. If user software enables an interrupt in the interrupt controller (SCU_xIC) which shares the same node as the SCU interrupt request active after reset, it may lead to the effect of suppressing the intended interrupt source. So, for all SCU interrupt sources which will not be used, make sure to disable the interrupt source (SCU_INTDIS) and clear any pending request flags (SCU_xIC.IR) before enabling interrupts in interrupt controller.

JTAG_X.H001 JTAG Pin Routing

In the current device, the pins connected to the JTAG interface can be selected by software (write to register DBGPRR). After a reset, the JTAG interface is connected to position A (see **Table 8**). If connected to these pins, the debugger will work without any restrictions.

Table 8 JTAG Position A

Pin LQFP-100	Pin LQFP-144	Symbol	Signal
23	34	P5.2	TDI_A
6	8	P7.0	TDO_A
57	82	P2.9	TCK_A
28	39	P5.4	TMS_A
5	6	$\overline{\text{TRST}}$	$\overline{\text{TRST}}$

To use other pins for the JTAG interface, the following sequence of steps must be executed:

- $\overline{\text{TRST}}$ must be high at the rising edge of $\overline{\text{PORST}}$. Usually debuggers provide that.
- Debuggers must be set to do a so called `hot attach`. This is connecting the microcontroller without executing a reset.
- Execute a write to DBGPRR register with the desired selection of pins to be used with one of the first instructions out of Flash.

RAM and most of the MultiCAN registers are no longer supported. A normal continuation when the suspend mode is left may not always be possible and may require a reset (e.g. depending on error counters).

PVC_X.H001 PVC Threshold Level 2

The Power Validation Circuits (PVCM, PVC1) compare the supply voltage of the respective domain (DMP_M, DMP_1) with programmable levels (LEV1V and LEV2V in register SCU_PVCMCON0 or SCU_PVC1CON0).

The default value of LEV1V is used to generate a reset request in the case of low core voltage.

LEV2V can generate an interrupt request at a higher voltage, to be used as a warning. Due to variations of the tolerance of both the Embedded Voltage Regulators (EVR) and the PVC levels, this interrupt can be triggered inadvertently, even though the core voltage is within the normal range. It is, therefore, recommended not to use this warning level.

LEV2V can be disabled by executing the following sequence:

1. Disable the PVC level threshold 2 interrupt request SCU_PVCMCON0.L2INTEN and SCU_ PVC1CON0.L2INTEN.
2. Disable the PVC interrupt request flag source SCU_INTDIS.PVCMI2 and SCU_INTDIS.PVC1I2.
3. Clear the PVC interrupt request flag source SCU_DMPMITCLR.PVCMI2 and SCU_ DMPMITCLR.PVC1I2.
4. Clear the PVC interrupt request flag by writing to SCU_INTCLR.PVCMI2 and SCU_INTCLR.PVC1I2.
5. Clear the selected SCU request flag (default is SCU_1IC.IR).

The Power Validation Circuits (PVCM) compare the supply voltage of the respective domain (DMP_M) with programmable levels (LEV1V and LEV2V in register SCU_PVCMCON0).

The default value of LEV1V is used to generate a reset request in the case of low core voltage.

LEV2V can generate an interrupt request at a higher voltage, to be used as a warning. Due to variations of the tolerance of both the Embedded Voltage Regulators (EVR) and the PVC levels, this interrupt can be triggered inadvertently, even though the core voltage is within the normal range. It is, therefore, recommended not to use this warning level.

LEV2V can be disabled by executing the following sequence:

1. Disable the PVC level threshold 2 interrupt request SCU_PVCMCON0.L2INTEN.
2. Disable the PVC interrupt request flag source SCU_INTDIS.PVCMI2.

USIC_AI.H001 FIFO RAM Parity Error Handling

A false RAM parity error may be signalled by the USIC module, which may optionally lead to a trap request (if enabled) for the USIC RAM, under the following conditions:

- a receive FIFO buffer is configured for the USIC module, and
- after the last power-up, less data elements than configured in bit field SIZE have been received in the FIFO buffer, and
- the last data element is read from the receiver buffer output register OUTRL (i.e. the buffer is empty after this read access).

Once the number of received data elements is greater than or equal to the receive buffer size configured in bit field SIZE, the effect described above can no longer occur.

To avoid false parity errors, it is recommended to initialize the USIC RAM before using the receive buffer FIFO. This can be achieved by configuring a 64-entry transmit FIFO and writing 64 times the value 0x0 to the FIFO input register IN00 to fill the whole FIFO RAM with 0x0.

USIC_AI.H002 Configuration of USIC Port Pins

Setting up alternate output functions of USIC port pins through Pn.IOCRy registers before enabling the USIC protocol (CCR.MODE = 0001_B, 0010_B, 0011_B or 0100_B) might lead to unintended spikes on these port pins. To avoid the unintended spikes, either of the following two sequences can be used to enable the protocol:

- Sequence 1:
  - Write the initial output value to the port pin through Pn_OMR
  - Enable the output driver for the general purpose output through Pn_IOCRx
  - Enable USIC protocol through CCR.MODE
  - Select the USIC alternate output function through Pn_IOCRx
- Sequence 2:
  - Enable USIC protocol through CCR.MODE
  - Enable the output driver for the USIC alternate output function through Pn_IOCRx

Similarly, after the protocol is established, switching off the USIC channel by reseting CCR.MODE directly might cause undesired transitions on the output pin. The following sequence is recommended:

- Write the passive output value to the port pin through Pn_OMR
- Enable the output driver for the general purpose output through Pn_IOCRx
- Disable USIC protocol through CCR.MODE

w w w . i n f i n e o n . c o m

03266AERRA

Published by Infineon Technologies AG