Mask Set Errata for Mask 2N95G

This report applies to mask 2N95G for these products:

- S12ZVML32
- S12ZVML64
- S12ZVML12
- S12ZVMC64
- S12ZVMC12

Mask Specific Information

JTAG identifier	No

Table 1. Errata and Information Summary

Erratum ID	Erratum Title
e6181	ADC: ADCSTS[RVL_SEL] is not changed as specified in rare corner case if ADC disabled while flow control request SEQA ongoing
e6975	ADC: Conversion Command List (CSL) swapping not consistent in corner case of Restart and LoadOK overrun scenario
e6964	ADC: Flag RSTAR_EIF is set unexpectedly in corner cases
e6965	ADC: Flag TRIG_EIF not set if erroneous TRIG request occurs short before STOP Mode entry or WAIT Mode entry with bit SWAI set
e8188	ADC: High current in Stop Mode
e9318	PMF: Glitch visible in PWM asymmetric operation mode
e8233	PMF: PWM signals after fault recovery incorrect
e7606	PMF: Writes to CINVn affect PWM output if generator disabled
e6167	S12Z_BDC: OVRUN bit is set unexpectedly after GO_UNTIL command (ACK enabled)
e6722	S12Z_BDC: WRITE_MEM, FILL_MEM commands issued during STOP with the core clock disabled can write incorrect data to the addressed location.
e8346	SCI: RXEDGIF interrupt miss while enter STOP

Table 2. Revision History

Revision	Changes
17 JUNE 2015	Initial revision
17 March 2016	The following errata were removed. • e7733 The following errata were added. • e9318 • e6866
24 June 2016	The following errata were removed. • e6866

e6181: ADC: ADCSTS[RVL_SEL] is not changed as specified in rare corner case if ADC disabled while flow control request SEQA ongoing

Description: In case a Sequence Abort Request is in process (ongoing) and the ADC gets disabled (ADCEN = 1'b0) before:

The Sequence Abort Request is finished which terminates any possible ongoing data bus access to store a conversion result.

In this case the RVL select is not changed as specified. Instead it is unchanged.

Workaround: The ADC should not be disabled before any flow control request (SEQA, TRIG, RSTA) is finished (all flow control bits are cleared).

It is recommended to issue a Sequence Abort Request and to poll bit SEQA to be clear before ADC is disabled via bit ADCEN.

e6975: ADC: Conversion Command List (CSL) swapping not consistent in corner case of Restart and LoadOK overrun scenario

Description: The CSL list swapping is not consistent when the conditions below are true:

- The CSL is configured for double buffer mode
- A Restart Event is in process (Conversion Command of a CSL is loaded)
- Then a Restart Request/LoadOK overrun situation happens just within a few cycles (previous Restart Request not finished)
- While the current CSL is aborted, an additional Restart Request/LoadOK overrun situation occurs

Expected behavior description:

If Restart/LoadOK overrun situations occur, then those requests should be stored in the background. So each Restart Request overrun with LoadOK should swap the selected/tracked CSL into the background. The ongoing list should be aborted and the last selected/tracked CSL must be started.

Error behavior description:

In the current case the second Restart Event overrun with LoadOK actually misses the LoadOk and the selected/tracked CSL is not swapped in the background. This leads to differently used command lists between modules e.g. for devices with PTU, the PTU and ADC command lists are no longer in sync.

Single Restart Request overruns with LoadOK within a few cycles after the previous Restart Request are tracked correctly.

NOTE:

A single Restart/LoadOK overrun scenario can happen in a motor control application, but not a multiple overrun situation within a few cycles.

Workaround: If possible, avoid multiple Restart Request overrun scenarios.

e6964: ADC: Flag RSTAR_EIF is set unexpectedly in corner cases

Description: In case of repeated Restart Request overruns (Restart overrun occurs again before previous Restart overrun could be handled) the RSTAR_EIF is erroneously set if:

- The first Restart Request overrun occurs correctly with a simultaneous Sequence Abort Request AND
- The second Restart Request overrun occurs without Sequence Abort Request, which is a flow control failure but for overrun situation the issue should not be flagged.

Misbehavior:

The ADC flags the flow control failure (RSTAR_EIF bit is set) in the second overrun which should not happen. Overruns are tracked quietly and executed as soon as Sequence Abort turnaround time allows.

Workaround: If possible, avoid multiple Restart Request overrun scenarios.

e6965: ADC: Flag TRIG_EIF not set if erroneous TRIG request occurs short before STOP Mode entry or WAIT Mode entry with bit SWAI set

Description: At Low power Mode entry (STOP or WAIT with bit SWAI set) all current flow control information as well as all pending flow control information gets deleted immediately.

In case of a flow control error, the detection can take several cycles until it is flagged by TRIG_EIF.

This leads to the corner case in which TRIG_EIF does not get set because the device enters STOP or WAIT with SWAI set .

Workaround: The ADC should be idle (no conversion or conversion sequence or Command Sequence List ongoing) when entering STOP mode or WAIT with bit SWAI set.

e8188: ADC: High current in Stop Mode

Description: The ADC can take a higher current in Stop Mode (500µA increasing over time to about 1mA per ADC instance) if the ADC is enabled and conversions have been done.

There is a software workaround available.

Workaround: This software workaround has been validated.

The subroutine ADC0_stop_current_workaround takes about 105 bus clock cycles (ECLK) on S12ZVM128.

The subroutine must be executed for each ADC instance performing conversions before Stop Mode entry.

Before the workaround subroutine is executed, the ADC must be initialized (command list pointers and result list pointers have valid addresses).

Please note:

- All the conversion commands of List 0 must be valid (no illegal channel, no illegal SMP value)
- ADCFMT[SRES] value must not be illegal
- The command fetch must not cause an illegal access flag

```
void main(void) {

...

DisableInterrupts; // Shutdown sequence is not interruptible until STOP

...

ADC0_stop_current_workaround(); // Subroutine must be executed with interrupt protection
asm(andcc #0x6f); /* CCW settings: S = 0, I = 0 */ asm(stop); /* MCU enters Stop mode if S =
0 in CCW */ /* I-mask interrupts will be enabled because I = 0 in CCW */ ... }

//Workaround to avoid ADC high current during STOP mode

//ADC will be disabled in this function

//Therefore following register are cleared:

//Address | Register | Description

//----------+----------+---------------------------

//0x02 | ADCSTS | CSL_SEL, RVL_SEL will be restored at the end of the function

//0x08 | ADCEIF | Error interrupt flags

//0x09 | ADCIF | Interrupt flags (SEQAD_IF, CONIF_OIF)

//0x0C-0x0D:| ADCCONIF | Conversion interrupt flags, EOL (end of list) interrupt flag

//0x0E-0x0F:| ADCIMDRI | Intermediate result information (must be stored by the customer
before,

// | | if this information is still needed after the function execution)

//0x10 | ADCEOLRI | EOL result information (must be stored by the customer before,

// | | if this information is still needed after the function execution)

//0x1C | ADCCIDX |

//0x20 | ADCRIDX |

void ADC0_stop_current_workaround (void) {

byte tmp_ADCxCTL_0 = ADC0CTL_0; // Save customer settings, these values will be restored
afterwards byte tmp_ADCxTIM = ADC0TIM; byte tmp_ADCxSTS = ADC0STS; ADC0CTL_0 =
0x00; // Disable ADC ADC0TIM = 0x00; // ADC is set to maximum frequency // Device
```

specification of allowed frequency is ignored, // the ADC conversion is stopped when reaching error state // There is no conversion result generated. ADC0CTL_0 = 0x88; // ADC is enabled, single access mode data bus, restart mode ADC0CTL_0 = 0x88; // Re-do in order to guarantee ADC is ready for requests before Restart Event occurs ADC0FLWCTL = 0x20; // ADC is restarted, RSTA bit is set: the first command of list 0 is // loaded from memory. The command type does not matter, the conversion // is immediately stopped while (ADC0FLWCTL_RSTA == 1) {} // Wait for restart completion (within a few clock cycles) ADC0FLWCTL = 0x40; // Start conversion (TRIG) ADC0FLWCTL = 0x40; // The second TRIG immediately generates a TRIG_EIF, ERROR state is entered while (ADC0EIF_TRIG_EIF == 0) {} // Wait for trigger error interrupt flag (within a few clock cycles) ADC0CTL_0_ADC_SR = 1; // Execute ADC soft-reset (SR), ADC enters IDLE state while (ADC0STS_READY == 0) {} // Wait for ADC soft-reset done (within a few clock cycles) ADC0CTL_0 = 0x00; // ADC is disabled ADC0TIM = tmp_ADCxTIM; //Restore previous customer settings ADC0STS = tmp_ADCxSTS; ADC0CTL_0 = tmp_ADCxCTL_0; // ADC0CTL_0 is the last one to be restored, in case ADC was enabled before... }

e9318: PMF: Glitch visible in PWM asymmetric operation mode

Description: When any of the PWM pairs in the PMF module is operating in asymmetric complementary center-aligned mode, with half cycle reload enabled.

PMF configuration:

- Complementary mode: PMFCFG0_INDEP{A,B,C}=0
- Center aligned outputs: PMFCFG0_EDGE{A,B,C}=0
- Asymmetric mode: PMFICCTL_ICC{A,B,C}=1
- Normal pulse edge control: PMFICCTL_PEC{A,B,C}=0
- Half cycle reload enabled: PMFFQC{A,B,C}_HALF{A,B,C}=1

And any of the following two conditions below (A or B) occur, an unexpected pulse with a width of “dead time” will be visible in the corresponding odd PWM channel output (PWM1,3 or 5)

Condition A.

1a. Setting the odd PWM channel to 0 (PMFVAL{1,3,5}=0) and loaded into the internal buffer (LDOKA=1) before next half cycle start, and

2a. Setting the even PWM channel to 0 (PMFVAL{0,2,4}=0)) and loaded into the internal buffer (LDOKA=1) before next full cycle start.

Condition B.

1b. Setting the odd PWM channel to 0 (PMFVAL{1,3,5}=0) and loaded into the internal buffer (LDOKA=1) before next full cycle start, and

2b. Setting the even PWM channel to 0 (PMFVAL{0,2,4}=0)) before next full cycle start and loaded into the internal buffer (LDOKA=1) before next full cycle start

Workaround: Set both VAL registers of each complementary pair, PMFVAL{1,3,5} and PMFVAL{0,2,4}, to zero before the next half cycle start to disable the PMF output and correct the unexpected pulse

e8233: PMF: PWM signals after fault recovery incorrect

Description: After the fault deassertion the PMF needs additional time before it fully recovers. In center aligned mode with half cycle enabled, this time can be 0.5 or 1 PMF reload cycles, in all other modes the PMF needs 1 PMF reload cycle to recover. This behavior is visible in all modes when recovering from faults 0, 1, 2, or 3. During this recovery time the PWM signals are driven by the state of the PMFOUTB register bit even if the corresponding PMFOUTC register bit is cleared. After this recovery time the PWM signals are controlled by the internal PWM generator. In complementary mode, before driving the PMF outputs by the PWM generator, the configured dead time is inserted, thus guaranteeing no overlap of the two channel outputs.

Workaround: To avoid an additional pulse on the PWM signals during the recovery from fault 0, 1, 2 or 3, the PMFOUTB register bit should only be set if the PMF module is used in software controlled mode (PMFOUTC register bit set).

e7606: PMF: Writes to CINVn affect PWM output if generator disabled

Description: Write accesses to compare invert bits CINVn altering the generator output polarity also take effect on the related PWM outputs if the generator is disabled (PWMENx=0).

Workaround: Configure CINVn after enabling PWM generator.

e6167: S12Z_BDC: OVRUN bit is set unexpectedly after GO_UNTIL command (ACK enabled)

Description: When the GO_UNTIL command with ACK enabled is executed, and BDC is not in active BDM mode, then the ILLCMD flag is set as specified.

But the command following GO_UNTIL sets the OVRUN flag, which is unexpected.

This is not a normal use case because GO_UNTIL is not an active BDM command.

Workaround: Use the SYNC pulse to clear OVRUN.

Development tool vendors can ensure the debugger does not allow GO_UNTIL unless the device is in active BDM.

e6722: S12Z_BDC: WRITE_MEM, FILL_MEM commands issued during STOP with the core clock disabled can write incorrect data to the addressed location.

Description: With the device core clock disabled during STOP mode debugging (BDCCIS=0) then WRITE_MEM and FILL_MEM commands can cause incorrect data to be written to the addressed location. This can happen if the device leaves STOP mode during execution of the WRITE_MEM, FILL_MEM command.

Workaround: Three workarounds are proposed. The appropriate workaround depends on the individual debugging requirements.

1. Do not disable the device core clock when debugging through STOP (set BDCCIS).

2. If BDCCIS=0 then only use WRITE_MEM, FILL_MEM whilst in active BDM (rather than during code execution).

3. If BDCCIS=0 then use WRITE_MEM.WS/FILL_MEM.WS which indicate that STOP mode occurred. If STOP=1 then read data back to verify correct write execution.

OR Use ACK and then check BDCCSR if a long ACK occurs. If STOP=1 then read data back to verify.

Note that the data could have been overwritten by execution code because the device has since left STOP mode. This is only of use if the user can be sure that the application code does not write to the same location.

e8346: SCI: RXEDGIF interrupt miss while enter STOP

Description: If an active edge (falling if RXPOL=0, rising if RXPOL=1) on the RXD input occurs shortly after the execution of the STOP instruction the RXEDGIF is not asserted and the CPU is not woken up. The time window in during which the edge is missed starts about 10 bus cycles after the STOP instruction and is 2-3 bus cycles wide.

Workaround: (1) If more than one edge with a minimum distance of 4 bus cycles occur, the 2nd edge will wake up the CPU. This is the case for instance in a LIN bus system. “The Wake Up Signal consists of a dominant pulse minimum 250 microseconds and maximum 5 milliseconds in length, and it may be sent by any LIN node.”

(2) Use the API to enforce a periodic wake up and check the level of the LIN input.

(3) Reduce the likelihood of occurrence by increasing the bus frequency.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support