Errata

SLAZ037C–June 2007–Revised January 2010

MSP430CG461x Device Erratasheet

1 Current Version

Devices	Rev:	ADC18	ADC25	CPU8	CPU16	CPU19	DMA3	DMA4	FLL3	FLL6	LCDA5	RTC1	TA12	TA16	TA18	TAB22	TB2	TB16	TB18	USCI19	USCI20	USCI21
MSP430CG4616	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4617	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4618	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4619	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Devices	Rev:	USCI22	USCI23	USCI24	USCI25	USCI26	USCI27	WDG2	XOSC5	XOSC8	XOSC9
MSP430CG4616	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4617	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4618	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MSP430CG4619	B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

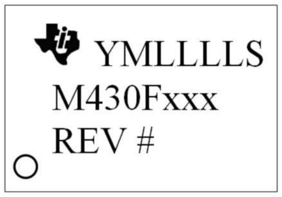
Note: See Appendix A for prior versions.

✓ The checkmark means that the issue is present in that revision.

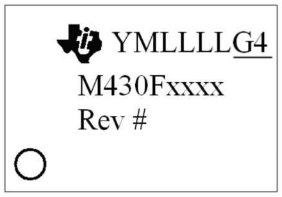
2 Package Markings

PZ100

LQFP (PZ) 100 Pin



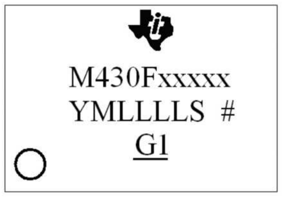
YM	= Year and Month Date Code
LLLL	= LOT Trace Code
S	= Assembly Site Code
#	= DIE Revision
o	= PIN 1



YM	= Year and Month Date Code
LLLL	= LOT Trace Code
S	= Assembly Site Code
#	= DIE Revision
o	= PIN 1

ZQW113

BGA (ZQW), 113 pin



YM	= Year and Month Date Code
LLLL	= LOT Trace Code
S	= Assembly Site Code
#	= DIE Revision
o	= PIN 1

3 Detailed Bug Description

ADC18 — *ADC12 Module*

Function
Incorrect conversion result in extended sample mode

Description
The ADC12 conversion result can be incorrect if the extended sample mode is selected (SHP = 0), the conversion clock is not the internal ADC12 oscillator (ADC12SSEL > 0), and one of the following two conditions is true:

- The extended sample input signal SHI is asynchronous to the clock source used for ADC12CLK and the undivided ADC12 input clock frequency exceeds 3.15 MHz.
  or
- The extended sample input signal SHI is synchronous to the clock source used for ADC12CLK and the undivided ADC12 input clock frequency exceeds 6.3 MHz.

Workaround

- Use the pulse sample mode (SHP = 1).
  or
- Use the ADC12 internal oscillator as the ADC12 clock source.
  or
- Limit the undivided ADC12 input clock frequency to 3.15 MHz.
  or
- Use the same clock source (such as ACLK or SMCLK) to derive both SHI and ADC12CLK, to achieve synchronous operation, and also limit the undivided ADC12 input clock frequency to 6.3 MHz.

ADC25 — *ADC12 Module*

Function
Write to ADC12CTL0 triggers ADC12 when CONSEQ = 00

Description
If ADC conversions are triggered by the Timer_B module and the ADC12 is in single-channel single-conversion mode (CONSEQ = 00), ADC sampling is enabled by write access to any bit(s) in the ADC12CTL0 register. This is contrary to the expected behavior that only the ADC12 enable conversion bit (ADC12ENC) triggers a new ADC12 sample.

Workaround
When operating the ADC12 in CONSEQ = 00 and a Timer_B output is selected as the sample and hold source, temporarily clear the ADC12ENC bit before writing to other bits in the ADC12CTL0 register. The following capture trigger can then be re-enabled by setting ADC12ENC = 1.

CPU8 — *CPU Module*

Function
Using odd values in the SP register

Description
The SP can be written with odd values. In the original CPU, an odd SP value could be combined with an odd offset (for example, `mov. #value, 5(SP)`). In the new CPU, the SP can be written with an odd value, but the first time the SP is used, the LSB is forced to 0.

Workaround
Do not use odd values with the SP.

CPU16 — *CPU Module*

Function

Indexed addressing with instructions `calla`, `mova`, and `bra`

Description

With indexed addressing mode and instructions `calla`, `mova`, and `bra`, it is not possible to reach memory above 64k if the register content is < 64k.

Example: Assume R5 = FFFEh. The instruction `calla 0004h(R5)` results in a 20-bit call of address 0002h instead of 10002h.

Workaround

- Use different addressing mode to reach memory above 64k.
- First use `adda [index],[Rx]` to calculate address in upper memory and then use `calla [Rx]`.

CPU19 — *CPU Module*

Function

CPUOFF can change register values

Description

If a CPUOFF command is followed by an instruction with an indirect addressed operand (for example, `mov @R8, R9, and RET`), an unintentional register-read operation can occur during the wakeup of the CPU. If the unintentional read occurs to a read-sensitive register (for example, UCB0RXBUF or TAIV), which changes its value or the value of other registers (IFGs), the bug leads to lost interrupts or wrong register read values.

Workaround

Insert a NOP instruction after each CPUOFF instruction.

DMA3 — *DMA Module*

Function

Read-modify-write instructions may corrupt DMA address registers

Description

When a 16-bit-wide read-modify-write instruction (such as add.w and sub.w) is directly used on a DMA address register (DMAxSA or DMAxDA), the register contents are corrupted.

Workaround

- Do not use 16-bit-wide read-modify-write instructions on DMA address registers. Instead, if address calculations are necessary, do the calculations first, and then assign the result to the DMA address registers.

  or
- Use 20-bit-wide read-modify-write instructions (such as addx.a, subx.a) on the DMA address registers, if needed.