# MAXQ610
# ERRATA SHEET

## Revision B1 Errata

*The errata listed below describe situations where MAXQ610 revision B1 components perform differently than expected or differently than described in the data sheet. Maxim Integrated Products, Inc., intends to correct these errata when the opportunity to redesign the product presents itself.*

*This errata sheet only applies to MAXQ610 revision B1 components. Revision B1 components are branded on the topside of the package with a six-digit code in the form yywwB1, where yy and ww are two-digit numbers representing the year and work week of manufacture, respectively. To obtain an errata sheet on another MAXQ610 die revision, visit our website at **www.maximintegrated.com/errata**.*

### 1) CANNOT EXECUTE A RETURN AT FLASH WORD ADDRESS 0x7FFF

**Description:**
A return instruction executed at word address 0x7FFF in flash memory does not return correctly to the calling function.

**Workaround:**
Make sure this location does not hold a return instruction.

### 2) $I_{DD}$ CAN EXCEED SPECIFIED VALUES WITH SLOW $V_{DD}$ RISE TIME OR STEADY STATE VOLTAGE BETWEEN $V_{POR}$ AND $V_{RST}$

**Description:**
$I_{DD}$ can exceed the specified maximum value by as much as 500µA if $V_{DD}$ is held at a voltage between $V_{POR}$ and $V_{RST}$. This condition only occurs if $V_{DD}$ starts at a voltage lower than $V_{POR}$ and then rises above $V_{POR}$ but remains below $V_{RST}$. A standard brownout condition will not cause this errata behavior.

**Workaround:**
Guarantee that the $V_{DD}$ rise time is faster than 1.5V/s to minimize the time spent in the region between $V_{RST}$ and $V_{PFW}$. The system designer must ensure that $V_{DD}$ cannot remain in the region between $V_{RST}$ and $V_{PFW}$ during power-up as a result of a low-voltage condition such as a discharged battery.

*19-5464; Rev 3; 1/13*

maxim integrated™

## 3) INTERRUPTS IMMEDIATELY PRECEEDING ENTRY INTO STOP MODE CAN AFFECT INTERRUPT PRIORITY HANDLING

**Description:**
If an interrupt is received on the cycle preceding entry into stop mode, that interrupt is handled as if it were high priority, regardless of the original priority. If the interrupt was not originally a high-priority interrupt, the interrupt priority logic erroneously believes that another interrupt of the original priority is still in progress. The existence of this false interrupt can prevent the activation of valid interrupts of the same or lower priority.

**Workaround:**
There are several workarounds. A software example that performs workaround #2 of erratum #3 as well as the fixes required for workaround #2 of erratum #4 is contained in the workaround section for erratum #4.

1) Disable low and medium priority interrupts before entering stop mode. Re-enable them as desired immediately after exiting stop mode. If low and medium priority interrupts are required in the interrupt service routines, however, they can be re-enabled at the start of the interrupt service routines.
2) Immediately following the NOP after stop mode, perform a POPI and matching PUSH instruction to clear the interrupt logic and remove the false interrupts.

## 4) INTERRUPTS IMMEDIATELY PRECEEDING ENTRY INTO STOP MODE CAN AFFECT STACK INTEGRITY

**Description:**
If an interrupt is received on the cycle preceding entry into stop mode, the stop-mode logic pushes an additional return address onto the stack in addition to the one pushed by the interrupt logic. The additional write to the stack offsets all stack values by one word.

**Workaround:**
There are several workarounds. A software example that performs workaround #2 of erratum #3 as well as the fixes required for workaround #2 of erratum #4 is below.

1) Disable low and medium priority interrupts before entering stop mode. Re-enable them as desired immediately after exiting stop mode. If low and medium priority interrupts are required in the interrupt service routines, however, they can be re-enabled at the start of the interrupt service routines.
2) Save the stack pointer in an intermediate location before entering stop mode. After the interrupt service routine is complete, the code resumes with the instruction following the instruction that activates stop mode. Read the stack pointer and compare it to the saved value. If they are not identical, that means that an extra value was pushed onto the stack. Perform a POP instruction, disregarding the returned value, to remove the extra data from the stack. An example of this procedure is:

```
    move    CORRECT_STACK_VALUE, SP     ; Save current SP value into a temp register

    move    CKCN.4, #1                  ; Enter STOP mode
    nop                                 ; NOP that is necessary for errata #2

    move    SCRATCH_REGISTER, ACC  ; Save contents of ACC to unused register

repairIPS:
    move    ACC, IC                ; Get register holding IPS bits
    and     #0Ch                   ; Mask off everything but IPS bits
    cmp     #0Ch                   ; Check for proper value
    jump    E, ipsFixed            ; If IPS == 0x3, no damage was done
    popi    ACC                    ;   else, do a popi to clear current interrupt level
    push    ACC                    ; Need to put the popped value back
    jump    repairIPS:             ; Repeat until IPS == 0x03
ipsFixed:

repairStack:
    move    ACC, SP                ; Get current stack value
    cmp     CORRECT_STACK_VALUE    ; Compare against pre-stop value
    jump    E, stackFixed          ; If they match, no damage was done
    pop     nul                    ;   else, pop off the extra value that was stored
    jump    repairStack:           ; Repeat until SP is back to its original value
stackFixed:

    move    ACC, SCRATCH_REGISTER  ; Restore contents of ACC
```

### 5) PUSH INSTRUCTION CAUSES CODE EXECUTION ERROR WHEN THE MOD[1:0] BITS HAVE BEEN CHANGED FROM THEIR DEFAULT VALUES

**Description:**
Changing the MOD[1:0] bits (APC[2:0]) from their default value can cause the device to operate incorrectly any time a PUSH instruction is followed by any instruction which reads the active accumulator.

**Workaround:**
The user must ensure that software never changes the MOD[1:0] bits of the AP register from their default value of 00b.

### 6) USE OF INTERRUPT PRIORITY FEATURE CAN CAUSE INCORRECT PROCESSING OF INTERRUPTS

**Description:**
An interrupt that is followed immediately by a higher priority interrupt can cause the CPU to incorrectly service the interrupts.

**Workaround:**
Do not use high-priority interrupts. Do not modify the IPR0 or IPR1 registers from their reset value. This sets all interrupt sources to the lowest (default) priority level.

# MAXQ610
# REV B1 ERRATA

### 7) ERASE COMMAND (0x02) DOES NOT ERASE LAST PAGE WHEN CONTEXT IS SET TO "USER LOADER" OR "USER APPLICATION"

**Description:**
Erase command requires additional operation to erase last page of flash memory when the context is set to 0x01 (user loader area) or 0x02 (user application area). This information is intended only for the use of manufacturers of commercial device programmers.

**Workaround:**
After setting the context, use the command sequence 0x02 0x00 0x00 0xE0, 0x00, 0x7F, 0x00, 0x00 instead of 0x02 0x00 to erase the device.

### 8) IR INPUT CLOCK FREQUENCY IS DEPENDENT ON IRDIV[2:0]

**Description:**
The formula for calculating $f_{IRCLK}$ should be $f_{SYS}/2^{IRDV[2:0]}$ instead of $f_{SYS}/2^{IRDV[1:0]}$.

**Workaround:**
Use the correct formula.

### 9) RET, RETI, POP, AND POPI INSTRUCTIONS INCREMENT SP

**Description:**
The data sheet incorrectly states that the aforementioned instructions decrement SP. The instructions increment SP.

**Workaround:**
None required.

# MAXQ610
# REV B1 ERRATA

**REVISION HISTORY**

| REVISION NUMBER | REVISION DATE | DESCRIPTION | PAGES CHANGED |
|---|---|---|---|
| 0 | 7/10 | Initial release | — |
| 1 | 11/10 | Added erratum #5 (PUSH instruction) | 3 |
| 2 | 11/11 | Added erratum #6 (interrupt priority feature) | 3 |
| 3 | 1/13 | Added erratum #7, #8, and #9 | 4 |