STM8S103xx and STM8S903xx Errata sheet

STM8S103xx and STM8S903xx device limitations

Silicon identification

This errata sheet applies to the STMicroelectronics STM8S103xx and STM8S903xx devices.

The full list of root part numbers is given in *Table 2*.

The products can be identified as shown in *Table 1*:

- By the revision code marked below the sales type on the device package
- By the last three digits of the Internal sales type printed on the box label

Table 1. Device identification

Sales type	Revision code marked on the device(1)
STM8S103xxxx	Z and Y/6
STM8S903xxxx	Z and Y/6

1. Refer to *Appendix A: Revision code on device marking* for details on how to identify the revision code according to the packages.

Table 2. Device summary

Part number	Part number
STM8S103xx	STM8S103K3, STM8S103F3, STM8S103F2
STM8S903xx	STM8S903K3, STM8S903F3

Contents

1 Product evolution

Table 3 gives a summary of the fix status.

Legend for *Table 3*: A = workaround available; N = no workaround available; P = partial workaround available, '-' and grayed = fixed.

Table 3. Product evolution summary

Section	Limitation	Rev Z	Rev Y/6
Section 2.1: Core limitations	*Section 2.1.1: Activation level (AL bit) not functional in Halt mode*	N	N
	Section 2.1.2: JRIL and JRIH instructions not available	N	N
	Section 2.1.3: Interrupt service routine (ISR) executed with priority of main process	N	N
	Section 2.1.4: Unexpected DIV/DIVW instruction result in ISR	A	A
Section 2.2: Clock controller	*Section 2.2.1: HSI RC oscillator cannot be switched off in Run mode*	N	N
	Section 2.2.2: LSI oscillator remains on in Active halt mode when the AWU unit uses the HSE as input clock	N	N
Section 2.3: UART peripheral limitations	*Section 2.3.1: UART PE flag cannot be cleared during the reception of the first half of Stop bit*	A	A
Section 2.4: I^2C peripheral limitations	*Section 2.4.1: I^2C event management*	A	A
	Section 2.4.2: Corrupted last received data in I^2C Master Receiver mode	A	A
	Section 2.4.3: Wrong behavior of I^2C peripheral in Master mode after misplaced STOP	A	A
	Section 2.4.4: Violation of I^2C "setup time for repeated START condition" parameter	A	A
	Section 2.4.5: In I^2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors	A	A
	Section 2.4.6: I^2C pulse missed	A	-

2 Silicon limitations

2.1 Core limitations

2.1.1 Activation level (AL bit) not functional in Halt mode

Description

As described in the STM8S microcontroller family reference manual (RM0016), an IRET instruction should switch the device into WFI mode or Halt mode when the AL bit is set.

In STM8S103/903xx devices, the IRET instruction only causes the CPU to enter the WFI mode.

Workaround

No workaround available.

2.1.2 JRIL and JRIH instructions not available

Description

JRIL (jump if port INT pin = 0) and JRIH (jump if port INT pin = 1) are not supported by the devices covered by this errata sheet. These instructions perform a conditional jump: JRIL and JRIH jump if one of the external interrupt lines is low and high, respectively. JRIL is equivalent to an unconditional jump and JRIH is equivalent to a NOP.

For further details on these instructions, refer to the STM8 CPU programming manual (PM0044) on http://www.st.com.

Workaround

No workaround available.

2.1.3 Interrupt service routine (ISR) executed with priority of main process

Description

If an interrupt is cleared or masked when the context saving has already started, the corresponding ISR is executed with the priority of the main process.

Workaround

No workaround available.

2.1.4 Unexpected DIV/DIVW instruction result in ISR

Description

In very specific conditions, a DIV/DIVW instruction may return a false result when executed inside an interrupt service routine (ISR). This error occurs when the DIV/DIVW instruction is interrupted and a second interrupt is generated during the execution of the IRET instruction of the first ISR. Under these conditions, the DIV/DIVW instruction executed inside the second ISR, including function calls, may return an unexpected result.

The applications that do not use the DIV/DIVW instruction within ISRs are not impacted.

Workaround 1

If an ISR or a function called by this routine contains a division operation, the following assembly code should be added inside the ISR before the DIV/DIVW instruction:

```
push cc
pop a
and a,#$BF
push a
pop cc
```

This sequence should be placed by C compilers at the beginning of the ISR using DIV/DIVW. Refer to your compiler documentation for details on the implementation and control of automatic or manual code insertion.

Workaround 2

To optimize the number of cycles added by workaround 1, you can use this workaround instead. Workaround 2 can be used in applications with fixed interrupt priorities, identified at the program compilation phase:

```
push #value
pop cc
```

where bits 5 and 3 of #value have to be configured according to interrupt priority given by I1 and I0, and bit 6 kept cleared.

In this case, compiler workaround 1 has to be disabled by using compiler directives.

No fix is planned for this limitation.

2.2 Clock controller

2.2.1 HSI RC oscillator cannot be switched off in Run mode

Description

The internal 16 MHz HSI RC oscillator cannot be switched off in Run mode even if the HSIEN bit is programmed to 0.

Workaround

No workaround available.

2.2.2 LSI oscillator remains on in Active halt mode when the AWU unit uses the HSE as input clock

Description

When the Auto wake-up unit (AWU) uses the high speed external clock (HSE) divided by the prescaler (clock source enabled by setting the CKAWUSEL option bit), the LSI RC oscillator is not switched off when the devices operate in Active halt mode with the main voltage regulator (MVR) on. This causes negligible extra power consumption compared to the total consumption of the MCU in Active Halt mode with main voltage regulator (MVR) on.

Workaround

No workaround available.

2.3 UART peripheral limitations

2.3.1 UART PE flag cannot be cleared during the reception of the first half of Stop bit

Description

The PE flag is set by hardware when the UART is in reception mode and a parity error (PE) occurs. This flag cannot be cleared during the first half of the Stop bit period. If the software attempts to clear the PE flag at this moment, the flag is set again by hardware, thus generating an unwanted interrupt (assuming the PIEN bit has been set in the UART_CR1 register.

Workaround

1. Disable PE interrupts by setting PIEN to 0.
2. After the RXNE bit is set (received data ready to be read), poll the PE flag to check if it a parity error occurred. For example, this could be done in the RXNE interrupt service routine.

2.4 I²C peripheral limitations

2.4.1 I²C event management

Description

As described in the I²C section of the STM8S and STM8A microcontroller reference manual (RM0016), the application firmware has to manage several software events before the current byte is transferred. If the EV7, EV7_1, EV6_1, EV6_3, EV2, EV8, and EV3 events are not managed before the current byte is transferred, problems may occur such as receiving an extra byte, reading the same data twice, or missing data.

Workaround

When the EV7, EV7_1, EV6_1, EV6_3, EV2, EV8, and EV3 events cannot be managed before the current byte transfer, and before the acknowledge pulse when the ACK control bit changes, it is recommended to use I²C interrupts in nested mode and to make them uninterruptible by increasing their priority to the highest priority in the application.

No fix is planned for this limitation.

2.4.2 Corrupted last received data in I²C Master Receiver mode

Conditions

In Master Receiver mode, when the communication is closed using method 2, the content of the last read data may be corrupted. The following two sequences are concerned by the limitation:

- Sequence 1: transfer sequence for master receiver when N = 2
  a) BTF = 1 (Data N-1 in DR and Data N in shift register)
  b) Program STOP = 1
  c) Read DR twice (Read Data N-1 and Data N) just after programming the STOP bit.
- Sequence 2: transfer sequence for master receiver when N > 2
  a) BTF = 1 (Data N-2 in DR and Data N-1 in shift register)
  b) Program ACK = 0
  c) Read Data N-2 in DR
  d) Program STOP bit to 1
  e) Read Data N-1.

Description

The content of the shift register (data N) is corrupted (data N is shifted 1 bit to the left) if the user software is not able to read data N-1 before the STOP condition is generated on the bus. In this case, reading data N returns a wrong value.

Workarounds

- Workaround 1
  – Sequence 1
    When sequence 1 is used to close communication using method 2, mask all active interrupts between STOP bit programming and Read data N-1.
  – Sequence 2
    When sequence 2 is used to close communication using method 2, mask all active interrupts between Read data N-2, STOP bit programming and Read data N-1.
- Workaround 2
  Manage I2C RxNE and TxE events with interrupts of the highest priority level, so that the condition BTF = 1 never occurs.

2.4.3 Wrong behavior of I²C peripheral in Master mode after misplaced STOP

Description

The I²C peripheral does not enter Master mode properly if a misplaced STOP is generated on the bus. This can happen in the following conditions:

- If a void message is received (START condition immediately followed by a STOP): the BERR (bus error) flag is not set, and the I²C peripheral is not able to send a START condition on the bus after writing to the START bit in the I2C_CR2 register.
- In the other cases of a misplaced STOP, the BERR flag is set in the IC2_CR2 register. If the START bit is already set in I2C_CR2, the START condition is not correctly generated on the bus and can create bus errors.

Workaround

In the I²C standard, it is not allowed to send a STOP before the full byte is transmitted (8 bits + acknowledge). Other derived protocols like CBUS allow it, but they are not supported by the I²C peripheral.

In case of noisy environment in which unwanted bus errors can occur, it is recommended to implement a timeout to ensure that the SB (start bit) flag is set after the START control bit is set. In case the timeout has elapsed, the peripheral must be reset by setting the SWRST bit in the I2C_CR2 control register. The I²C peripheral should be reset in the same way if a BERR is detected while the START bit is set in I2C_CR2.

No fix is planned for this limitation.

2.4.4 Violation of I²C “setup time for repeated START condition” parameter

Description

In case of a repeated Start, the “setup time for repeated START condition” parameter (named $t_{SU(STA)}$ in the datasheet and Tsu:sta in the I²C specifications) may be slightly violated when the I²C operates in Master Standard mode at a frequency ranging from 88 to 100 kHz. $t_{SU(STA)}$ minimum value may be 4 µs instead of 4.7 µs.

The issue occurs under the following conditions:

1. The I²C peripheral operates in Master Standard mode at a frequency ranging from 88 to 100 kHz (no issue in Fast mode)
2. and the SCL rise time meets one of the following conditions:
   - The slave does not stretch the clock and the SCL rise time is more than 300 ns (the issue cannot occur when the SCL rise time is less than 300 ns).
   - or the slave stretches the clock.

Workaround

Reduce the frequency down to 88 kHz or use the I²C Fast mode if it is supported by the slave.

2.4.5 In I²C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors

Description

The data valid time ($t_{VD;DAT}$, $t_{VD;ACK}$) described by the I²C specifications may be violated as well as the maximum current data hold time ($t_{HD;DAT}$) under the conditions described below. In addition, if the data register is written too late and close to the SCL rising edge, an error may be generated on the bus: SDA toggles while SCL is high. These violations cannot be detected because the OVR flag is not set (no transmit buffer underrun is detected).

This issue occurs under the following conditions:

1. The I²C peripheral operates In Slave transmit mode with clock stretching disabled (NOSTRETCH=1)
2. and the application is late to write the DR data register, but not late enough to set the OVR flag (the data register is written before the SCL rising edge).

Workaround

If the master device supports it, use the clock stretching mechanism by programming the bit NOSTRETCH=0 in the I2C_CR1 register.

If the master device does not support it, ensure that the write operation to the data register is performed just after TXE or ADDR events. You can use an interrupt on the TXE or ADDR flag and boost its priority to the higher level.

Using the "NOSTRETCH" mode with a slow I²C bus speed can prevent the application from being late to write the DR register (second condition).

Note: The first data to be transmitted must be written into the data register after the ADDR flag is cleared, and before the next SCL rising edge, so that the time window to write the first data into the data register is less than t_{LOW}.

If this is not possible, a possible workaround can be the following:

1. Clear the ADDR flag
2. Wait for the OVR flag to be set
3. Clear OVR and write the first data.

The time window for writing the next data is then the time to transfer one byte. In that case, the master must discard the first received data.

2.4.6 I²C pulse missed

Description

When the I²C interface is used for long transmit/receive transactions, the MCU may return a NACK somewhere during the transaction instead of returning an ACK for all data. The received data may also be corrupted. In Master mode the I²C may not detect an incoming ACK. This is due to a weakness in the noise filter of the I/O pad which in certain conditions may cause the STM8 I²C to miss a pulse.

The workaround described below is not a clean solution.

Workaround

Since data corruption is caused by noise generated by the CPU, CPU activity should be minimized during data reception and/or transmission. This is done by performing physical data transmission (Master mode) and reception (slave mode) in WFI state (wait for interrupt).

To allow the device to be woken up from WFI, I²C transmission and reception routines must be implemented through interrupt routines instead of polling mechanisms. Receive and transmit interrupts (received data processing) must be triggered only by the BTF bit flag (byte transfer finished) in the I2C_SR1 register. This flag indicates that the I²C is in stretched state (data transfers are stretched on the bus).

Clock stretching must be enabled to allow data transfers from the slave to be stopped and to allow the CPU to be woken up to read the received byte.

To recover from possible errors, periodically check if the I²C does not remain in busy state for too long (BUSY bit set in I2C_SR3 register). If so, it should be reinitialized.

Example of I²C slave code:

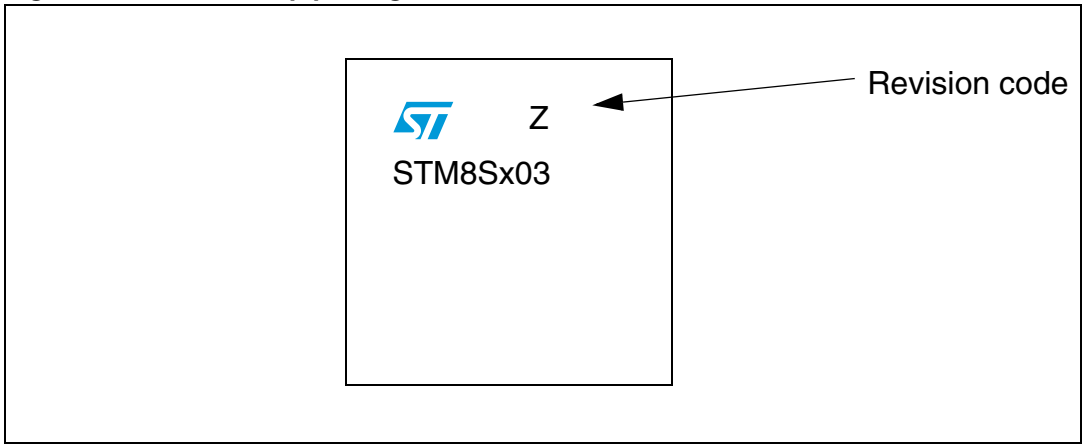
```
//...
//---------------------------------------------------------------
void main()
{
  Init_I2C(); // init I2C to use interrupts: ITBUFEN=0, ITEVTEN=1,
ITERREN=1
  while(1)
```

Appendix A Revision code on device marking

The revision code is marked on the package, except for revision Z which is not present on TSSOP20.

Figure 1 shows the marking for the LQFP32 package.

Figure 1. LQFP32 top package view



Revision history

Table 4. Document revision history

Date	Revision	Changes
01-Apr-2010	1	Initial release.
21-Feb-2011	2	Added revision 6. Added *Section 2.1.4: Unexpected DIV/DIVW instruction result in ISR*. Updated *Table 3* and *Section 2.4: I^2C peripheral limitations*.