STM8AF6213 STM8AF6223/26 Errata sheet

Device limitations for STM8AF62xx automotive MCUs featuring up to 8 Kbytes of Flash program memory

Silicon identification

This errata sheet applies to the STMicroelectronics STM8AF6213 and STM8AF6223/26. devices.

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

- By the revision code marked 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)
STM8AF6213	A
STM8AF6223/26	A

1. Refer to the device datasheet for details on how to identify the revision code according to the packages.

Contents

List of tables

List of figures

1 Product evolution

Table 2 gives a summary of the fix status.

Legend for *Table 2*: A = workaround available; N = no workaround available; P = partial workaround available, '-' and grayed = fixed.

Table 2. Product evolution summary

Section	Limitation	Rev A
Section 2.1: Core limitations	*Section 2.1.1: Activation level (AL bit) not functional in Halt mode*	N
	Section 2.1.2: JRIL and JRIH instructions not available	N
	Section 2.1.3: Interrupt service routine (ISR) executed with priority of main process	A
	Section 2.1.4: Unexpected DIV/DIVW instruction result in ISR	A
Section 2.2: System limitations	*Section 2.2.1: HSI RC oscillator cannot be switched off in Run mode*	N
	Section 2.2.2: LSI oscillator remains on in Active-halt mode when the AWU unit uses the HSE as input clock	N
	Section 2.2.3: Flash / EEPROM memory is read incorrectly after wakeup from power down mode	A
	Section 2.2.4: VDD rise-time rate for 100mV < VDD < 1V	N
Section 2.3: EXTI limitations	*Section 2.3.1: Possible collision in servicing of external interrupts (EXTI)*	N
Section 2.4: Timer peripheral limitations	*Section 2.4.1: Corruption of read sequence for the 16-bit counter registers*	A
Section 2.5: LINUART (UART4) peripheral limitations	*Section 2.5.1: UART PE flag cannot be cleared during the reception of the first half of Stop bit*	A
	Section 2.5.2: PE testing issue in UART mode	N
	Section 2.5.3: LIN mode: framing error with data byte 0x00	N
	Section 2.5.4: LIN mode: framing error when receiving an identifier (ID)	N
	Section 2.5.5: LIN mode: parity error when receiving an identifier (ID)	N
	Section 2.5.6: LIN mode: OR flag not correctly set in LIN Master mode	N

Table 2. Product evolution summary (continued)

Section	Limitation	Rev A
Section 2.6: I2C peripheral limitations	*Section 2.6.1: I2C event management*	A
	Section 2.6.2: Corrupted last received data in I2C Master Receiver mode	A
	Section 2.6.3: Wrong behavior of I2C peripheral in Master mode after misplaced STOP	A
	Section 2.6.4: Violation of I2C "setup time for repeated START condition" parameter	A
	Section 2.6.5: In I2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors	A

2 Silicon limitations

2.1 Core limitations

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

Description

The AL bit is not supported in Halt mode. In particular, when the AL bit of the CFG_GCR register is set, the CPU does not return to Halt mode after exiting an interrupt service routine (ISR). It returns to the main program and executes the next instruction after the HALT instruction. The AL bit is supported correctly in WFI mode.

Workaround

No workaround available.

No fix is planned for this limitation.

2.1.2 JRIL and JRIH instructions not available

Description

The JRIL (jump if port INT pin = 0) and JRIH (jump if port INT pin = 1) instructions are not supported by the devices covered by this errata sheet. These instructions perform conditional jumps: JRIL and JRIH jump if one of the external interrupt lines is low or high respectively.

In the devices covered by this errata sheet, JRIL is equivalent to an unconditional jump and JRIH is equivalent to NOP. For further details on these instructions, see the STM8 CPU programming manual (PM0044).

Workaround

No workaround available.

No fix is planned for this limitation.

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. The next interrupt request can interrupt execution of the service routine

Workaround

At the beginning of the interrupt routine, change the current priority level in the CCR register by software.

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 the 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, the user 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 System limitations

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.

No fix is planned for this limitation.

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 device operates 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 the MVR on.

Workaround

No workaround available.

No fix is planned for this limitation.

2.2.3 Flash / EEPROM memory is read incorrectly after wakeup from power down mode

Description

If Flash/EEPROM memory has been put in power down mode (I_{DDQ}), the first read access after wakeup could return incorrect content.

By default, the Flash/EEPROM memory is put in I_{DDQ} mode when the MCU enters Halt mode and depending on the FLASH_CR1 register settings made by software, the Flash/EEPROM may be forced to I_{DDQ} mode during active halt mode.

As a consequence, the following behavior may be seen on some devices:

- After wakeup from Low power mode, with Flash memory in I_{DDQ} mode, program execution gets lost due to an incorrect read of the vector table.
- Code reads an incorrect value from Flash/EEPROM memory, when forced in I_{DDQ} mode.
- Reset could be forced by an illegal opcode execution due to incorrect read of instruction.

Note: The use of the watchdog helps the application to recover in case of failure.

Workaround

Keep the Flash/EEPROM in operating mode when MCU is put in Halt mode or Active-halt mode. This is done by configuring both the HALT and AHALT bits in the Flash_CR1 register before executing a HALT instruction to prevent the Flash/EEPROM entering I_{DDQ} mode.

Set HALT (bit 3) to ‘1’:

0: Flash in power-down mode when MCU is in Halt mode

1: Flash in operating mode when MCU is in Halt mode

Keep AHALT (bit 2) at ‘0’:

0: Flash in operating mode when MCU is in Active-halt mode

1: Flash in power-down when MCU is in Active-halt mode

Please refer to the datasheet for details on the impact on current consumption and wakeup time.

2.2.4 V_{DD} rise-time rate for 100mV < V_{DD} < 1V

Description

The product datasheet did not specify the V_{DD} rise-time initial conditions as the V_{DD} rise-time was implicitly specified for a V_{DD} starting from 0V. Nevertheless, we observed that some very specific applications could have a V_{DD} starting from a residual voltage already above 0V and thus require that we explicitly specify these conditions.

The t_{VDD} parameter must stay below 50µs/V when V_{DD} is rising from 100mV to 1V.

Table 3. V_{DD} rise-time and fall-time rates

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
t_{VDD}	V_{DD} rise-time rate	V_{DD} < 100mV	2(1)	-	∞	µs/V
		100mV < V_{DD} < 1V	2(1)	-	50(1)	
		V_{DD} > 1V	2(1)	-	∞	
	V_{DD} fall-time rate	-	2(1)	-	∞	

1. Guaranteed by design, not tested in production.

Workaround

Not applicable.

2.3 EXTI limitations

2.3.1 Possible collision in servicing of external interrupts (EXTI)

Description

When an interrupt handler starts executing a service routine and an external interrupt (EXTI) request is pending or arrives during the same cycle, the external interrupt is not executed.

In addition, in nested interrupt mode, when the EXTI arrives between the 1st and the 2nd cycles before an interrupt handler with lower software priority starts executing its service routine, this EXTI interrupt tries to nest it. However, the EXTI request is cleared before fetching the interrupt vector and the previous handler is fetched instead. As a result the previous handler is executed twice and the EXTI service routine is not executed.

The limitation described above is valid for interrupts with address differing by 16, as shown in *Table 4*.

Table 4. Potential interrupt conflicts

EXTI source	Conflicting vector
EXTI0 – Port A	I2C interrupt
EXTI1 – Port B	No conflict – reserved vector
EXTI2 – Port C	No conflict – reserved vector

Table 4. Potential interrupt conflicts (continued)

EXTI source	Conflicting vector
EXTI3 – Port D	ADC interrupt
EXTI 4 – Port E	TIM4 interrupt

Workaround

No software workaround is available. It is recommended to choose the EXTI source to avoid conflicts.

No fix is planned for this limitation.

2.4 Timer peripheral limitations

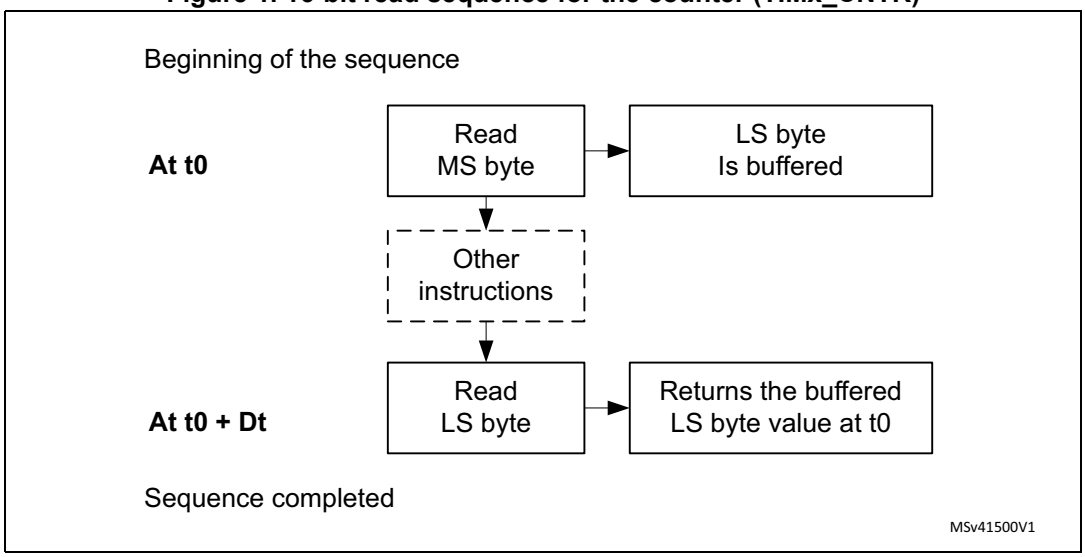
2.4.1 Corruption of read sequence for the 16-bit counter registers

Description

An 8-bit buffer is implemented for reading the 16-bit counter registers. Software must read the MS byte first, after which the LS byte value is buffered automatically (see *Figure 1*). This buffered value remains unchanged until the 16-bit read sequence is completed.

When any multi-cycle instruction precedes the read of the LSB, the content of the buffer is lost and the second read returns the immediate content of the counter directly.

Figure 1. 16-bit read sequence for the counter (TIMx_CNTR)



Workaround

Do not use multi-cycle instructions before reading the LSB.

No fix is planned for this limitation.

2.5 LINUART (UART4) peripheral limitations

2.5.1 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.

2.5.2 PE testing issue in UART mode

Description

When the RXNE flag is not polled, the device is in overrun condition and the PE flag does not rise in case of a parity error. The flag rises only for the last data which have been correctly received.

Workaround

No workaround available.

No fix is planned for this limitation.

2.5.3 LIN mode: framing error with data byte 0x00

Description

If the UART2 interface is configured in LIN slave mode, and the active mode with break detection length is set to 11 (LBDL bit of UART2_CR4 register set to 1), FE and RXNE flags are not set when receiving a 0x00 data byte with a framing error, followed by a recessive state. This occurs only if the dominant state length is between 9.56 and 10.56 times the baud rate.

Workaround

The LIN software driver can handle this exceptional case by implementing frame timeouts to comply with the LIN standard. This method has been implemented in ST LIN 2.1 driver package which passed the LIN compliance tests.

2.5.4 LIN mode: framing error when receiving an identifier (ID)

Description

If an ID framing error occurs when the UART2, configured in LIN mode, is in active mode, both the LHE and LHDF flags are set at the end of the LIN header with an ID framing error.

Workaround

The LIN software driver can handle this case by checking both LHE and LHDF flags upon header reception.

2.5.5 LIN mode: parity error when receiving an identifier (ID)

Description

If an ID parity error occurs, the UART2, configured in LIN mode, wakes up from mute mode and both LHE and LHDF are set at the end of the LIN header with parity error. The PE flag is also set.

Workaround

The LIN software driver can handle this case by checking all flags upon header reception.

No fix is planned for this limitation.

2.5.6 LIN mode: OR flag not correctly set in LIN Master mode

Description

When the UART operates in LIN Master mode, the OR flag is not set if an overrun condition occurs.

Workaround

The LIN software driver can detect this case through a LIN protocol error.

No fix is planned for this limitation.

2.6 I^2C peripheral limitations

2.6.1 I^2C event management

Description

As described in the I^2C 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 I^2C 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.

2.6.2 Corrupted last received data in I^2C 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.

2.6.3 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.

2.6.4 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.

2.6.5 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. The user 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.

2.6.6 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, I^2C 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 I^2C 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 I^2C does not remain in busy state for too long (BUSY bit set in I2C_SR3 register). If so, it should be reinitialized.

Example of I^2C slave code:

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

3 Revision history

Table 5. Document revision history

Date	Revision	Changes
10-Dec-2013	1	Initial release.
12-Apr-2016	2	Added *Section 2.2.4: VDD rise-time rate for 100mV < VDD < 1V*. Deleted Appendix A.