STM8AL3xx6/8 STM8Lx5xx4/6 Errata sheet

STM8AL31xx, STM8AL3Lxx, STM8L052C6, STM8L151xx4/6 and STM8L152xx4/6 device limitations

Silicon identification

This errata sheet applies to revisions B, Z and Y of the STMicroelectronics STM8AL31xx, STM8AL3Lxx, STM8L052C6, STM8L151xx4/6 and STM8L152xx4/6 devices. Please refer to products datasheets for peripherals availability related to limitations.

The full list of part numbers is shown in *Table 2*. The products are identifiable as shown in *Table 1*:

- by the revision code marked below the product identification area on the package
- by the last three digits of the Internal order code printed on the box label

Table 1. Device identification

Order code	Revision code marked on device		
STM8L052C6	“Z”		“Y”
STM8AL3168, STM8AL3166, STM8AL3148, STM8AL3146, STM8AL3138, STM8AL3136, STM8AL3L68, STM8AL3L66, STM8AL3L48, STM8AL3L46	“Z”		
STM8L152C6, STM8L152C4, STM8L152K6, STM8L152K4	“B”	“Z”	“Y”
STM8L151C6, STM8L151C4, STM8L151K6, STM8L151K4, STM8L151G6, STM8L151G4	“B”	“Z”	“Y”

Table 2. Device summary

Reference	Part number
STM8L052xx	STM8L052C6
STM8L152xx	STM8L152C6, STM8L152C4, STM8L152K6, STM8L152K4
STM8L151xx	STM8L151C6, STM8L151C4, STM8L151K6, STM8L151K4, STM8L151G6, STM8L151G4
STM8AL31xx	STM8AL3168, STM8AL3166, STM8AL3148, STM8AL3146, STM8AL3138, STM8AL3136
STM8AL3Lxx	STM8AL3L68, STM8AL3L66, STM8AL3L48, STM8AL3L46

Contents

1 Silicon limitations

Table 3 gives a summary of the fix status.

Legend for *Table 3*: A = workaround available; N = no workaround available; P = partial workaround available; N/A: not applicable; ‘-’ and grayed = fixed.

Table 3. Summary of STM8L05/15xx4/6 and STM8AL silicon limitations

Section	Limitation	STM8L 152x4/6 rev. B	STM8L 151x4/6 rev. B	STM8L 152x4/6 and STM8AL 3Lxx rev. Z/Y	STM8L 151x4/6 and STM8AL 31xx rev. Z/Y	STM8L 052C6 rev. Z/Y
Section 1.1: Core limitations	*Section 1.1.1: Interrupt service routine (ISR) executed with priority of main process*	N	N	N	N	N
	Section 1.1.2: Main CPU execution is not resumed after an ISR resets the AL bit	A	A	A	A	A
	Section 1.1.3: Unexpected DIV/DIVW instruction result in ISR	A	A	A	A	A
	Section 1.1.4: Incorrect code execution when WFE execution is interrupted by ISR or event	A	A	A	A	A
	Section 1.1.5: Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM	A	A	A	A	A
	Section 1.1.6: Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode	A	A	A	A	A
Section 1.2: System limitations	*Section 1.2.1: PA0, PB0 and PB4 configuration “at reset state” and “under reset”*	N	N	N	N	N
	Section 1.2.2: Restart issue when BOR0 is selected and the system clock source is not HSI at power-down	N	N	–	–	–
	Section 1.2.3: LSE and HSE in bypass mode cannot be used as RTC clock	N	N	–	–	–
	Section 1.2.4: CCO (configurable clock output) may not be functional after a reset	N	N	–	–	–
	Section 1.2.5: 32.768 kHz LSE crystal accuracy may be disturbed by the use of adjacent I/Os	N	N	N	N	N
	Section 1.2.6: Program memory are write unprotected after reset when embedded bootloader enabled	A	A	A	A	A
	Section 1.2.7: Data memory are write unprotected after reset when embedded bootloader enabled	A	A	A	A	A

Table 3. Summary of STM8L05/15xx4/6 and STM8AL silicon limitations (continued)

Section	Limitation	STM8L 152x4/6 rev. B	STM8L 151x4/6 rev. B	STM8L 152x4/6 and STM8AL 3Lxx rev. Z/Y	STM8L 151x4/6 and STM8AL 31xx rev. Z/Y	STM8L 052C6 rev. Z/Y
Section 1.2: System limitations	*Section 1.2.8: Program memory are write protected when embedded bootloader enabled and ROP is enabled*	A	A	A	A	A
Section 1.3: I^2C peripheral limitations	*Section 1.3.1: I^2C event management*	A	A	A	A	A
	Section 1.3.2: Corrupted last received data in I^2C Master Receiver mode	A	A	A	A	A
	Section 1.3.3: Wrong behavior of the I^2C peripheral in Master mode after misplaced STOP	A	A	A	A	A
	Section 1.3.4: Violation of I^2C “setup time for repeated START condition” parameter	A	A	A	A	A
	Section 1.3.5: In I^2C slave “NOSTRETCH” mode, underrun errors may not be detected and may generate bus errors	A	A	A	A	A
	Section 1.3.6: SMBus standard not fully supported in I^2C peripherals	A	A	A	A	A
Section 1.4: USART peripheral limitations	*Section 1.4.1: USART IDLE frame detection not supported in the case of a clock deviation*	N	N	N	N	N
	Section 1.4.2: PE flag can be cleared in USART Duplex mode by writing to the data register	A	A	A	A	A
	Section 1.4.3: PE flag is not set in USART Mute mode using address mark detection	N	N	N	N	N
	Section 1.4.4: IDLE flag is not set using address mark detection in the USART peripheral	N	N	N	N	N
Section 1.5: DAC peripheral limitations	*Section 1.5.1: Default DAC output level when output buffer is enabled*	N	N	–	–	N/A
Section 1.6: ADC peripheral limitations	*Section 1.6.1: ADC VREF+ voltage level disturbed when the PF0 voltage is higher than VREF+*	N	N	–	–	–
Section 1.7: LCD peripheral limitations	*Section 1.7.1: Overvoltage on VLCD when using internal step-up converter*	A	N/A	–	–	–
	Section 1.7.2: LCD voltage levels are disturbed when VIN > 3.9 V on PA7 and PE0 5 V tolerant pins	N	N/A	N	N/A	N/A
Section 1.8: Timer peripheral limitations	*Section 1.8.1: TIM1 advanced timer: Bad regulation for 100% PWM*	N	N	N	N	N

1.1 Core limitations

1.1.1 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

None.

No fix is planned for this limitation.

1.1.2 Main CPU execution is not resumed after an ISR resets the AL bit

Description

If the CPU is in wait for interrupt state and the AL bit is set, the CPU returns to wait for interrupt state after executing an ISR. To continue executing the main program, the AL bit must be reset by the ISR. When AL is reset just before exiting the ISR, the CPU may remain stalled.

Workaround

Reset the AL bit at least two instructions before the IRET instruction.

No fix is planned for this limitation.

1.1.3 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.

1.1.4 Incorrect code execution when WFE execution is interrupted by ISR or event

Description

Two types of failures can occur:

Case 1:

In case WFE instruction is placed in the two MSB of the 32-bit word within the memory, an event which occurs during the WFE execution cycle or re-execution cycle (when returning from ISR handler) will cause an incorrect code execution.

Case 2:

An interrupt request, which occurs during the WFE execution cycle will lead to incorrect code execution. This is also valid for the WFE re-execution cycle, while returning from an ISR handler.

The above failures have no impact on the core behavior when the ISR request or events occur in Wait for Event mode itself, out of the critical single cycle of WFE instruction execution.

Workaround

General solution is to ensure no interrupt request or event occurs during WFE instruction execution or re-execution cycle by proper application timing.

Dedicated workarounds:

Case 1:

Replace the WFE instruction with

```
  WFE
  JRA next
next:
```

Case 2:

It is recommended to avoid any interrupts before WFE mode is entered. This can be done by disabling all interrupts before the device enters Wait for event mode.

```
SIM
WFE
RIM
```

This workaround also prevents WFE re-execution in case 1.

No fix is planned for this limitation.

1.1.5 Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM

Description

When the MCU performs EEPROM program/erase operation, the core is stalled during data transfer to the memory controller, which occurs at the beginning of the program/erase operation. If a DMA request servicing starts while the core is stalled, the core does not return from stall mode to program execution.

The core is stalled for 11 cycles during byte program/erase, 8 cycles during word program/erase and 3 cycles during each word transfer in block programming mode. For block erase, the core is stalled for 127 cycles.

When a DMA request arises, it is only served if the DMA priority is higher than the core access priority.

If the current DMA priority is lower than the core one, the DMA service is delayed until the core access becomes idle.

The DMA also includes a programmable timeout function, configurable by DMA_GCSR register. If the core does not release the bus during this timeout, the DMA automatically increases its own priority and forces the core to release the bus for DMA service.

No fix is planned for this limitation.

Several workarounds are available for this limitation.

Workaround 1

Disable all DMA requests during data transfer to the EEPROM.

This workaround is applicable for all program/erase operations.

Workaround 2

Configure DMA programmable timeout in the DMA_GCSR register to exceed the number of stall cycles required during the transfer. DMA priority must never be configured to a very high level.

This workaround is applicable for all program/erase operations except block erase.

In order to apply this workaround to block erase, use block programming to 0x00 instead of block erase. This takes ~6 ms instead of ~3 ms.

Workaround 3

This workaround can be used if block erase cannot be replaced by block programming.

In this workaround, DMA is used to transfer data to the EEPROM instead of the core. All other DMA transfers are delayed once the core is stalled due to data transfer to memory controller.

```
/* start of the workaround in user code, using FW Library */
#ifdef USE_EVENT_MODE
  DMA1_Channel3->CCR= DMA_CCR_MEM | DMA_CCR_IDM | DMA_CCR_TCIE;   /*
Config DMA Chn3 Mem, incr, disable, interrupt) */
#else
  DMA1_Channel3->CCR= DMA_CCR_MEM | DMA_CCR_IDM;   /* Config DMA
Chn3 (Mem, incr,disable) */
#endif
   DMA1_Channel3->CM0ARH= (uint8_t)0;    /* Source address */
  DMA1_Channel3->CM0ARL= (uint8_t)0;
  DMA1_Channel3->CPARH= (uint8_t)(addr_begin >> 8); /* Destination
address */
  DMA1_Channel3->CPARL= (uint8_t)(addr_begin);
  DMA1_Channel3->CNBTR= 2; /* Number of data to be transferred */
  DMA1_Channel3->CSPR= 8; /* Low priority, 16 bit mode */
  DMA1_Channel3->CSPR &= ~DMA_CSPR_TCIF;/* Clear TCIF */
  DMA1->GCSR|= 1; /* Global DMA enable */

#ifdef USE_EVENT_MODE
  WFE->CR3 = WFE_CR3_DMA1CH23_EV; /* Enable event generation on
   DMA */
#endif
  FLASH->DUKR = 0xAE; /* Unprotect data memory */
  FLASH->DUKR = 0x56;
  while((FLASH->IAPSR & FLASH_IAPSR_DUL) == 0)
  {} /* Polling DUL */
  FLASH_Block_Load();
/* end of the workaround in user code */

/* following routine has to be placed in the RAM */
void FLASH_Block_Load(){
  __asm("sim\n"); /* Disable interrupts */

  FLASH->CR2 |= FLASH_CR2_ERASE; /* Enable erase block mode */
  DMA1_Channel3->CCR|= DMA_CCR_CE; /* Enable DMA MEM transfer */
#ifdef USE_EVENT_MODE
  __asm("wfe"); /* Wait for end of DMA operation */
#else
  while((DMA1_Channel3->CSPR & DMA_CSPR_TCIF) == 0)
  {} /* Polling for end of DMA operation */
#endif

  __asm("rim\n"); /* Enable interrupts */
}
```

1.1.6 Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode

Description

In case FLASH/EEPROM memory is put in power down mode (I_{DDQ}), first read after wakeup could return an incorrect content when F_{CPU} is above 8 MHz + 5%.

FLASH/EEPROM memory is put in I_{DDQ} mode by default during Halt mode and could be forced to I_{DDQ} mode by software for wait mode and during RAM execution.

As a consequence, 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 incorrect read of vector table.
- Code running from RAM read an incorrect value from FLASH/EEPROM memory, when forced in I_{DDQ} mode.
- Program execution gets corrupted when returning from RAM execution to FLASH memory execution in case FLASH memory is forced in I_{DDQ} mode.

Workaround

Slow down F_{SYSCLK} before entering Low power mode to ensure correct FLASH memory wakeup. This could be done using clock divider (CLK_CKDIVR) or by activation of fast wakeup feature by setting FHWU bit in CLK_ICKCR register. Original clock setting can be reconfigured back by software after wakeup.

Code example, assuming no divider is used in application by default.

```
CLK_CKDIVR = 0x01;
_asm(“HALT”);
CLK_CKDIVR = 0x00;
```

The interrupt service routine executed after wakeup could either stay at slower clock speed, or reconfigure clock setting. Care has to be taken to restore previous clock divider at the end of interrupt routines when modifying clock divider.

No fix is planned for this limitation.

1.2 System limitations

1.2.1 PA0, PB0 and PB4 configuration “at reset state” and “under reset”

Description

When a reset occurs, PA0, PB0 and PB4 configurations differ from the other pins:

- PA0 is configured as input with pull-up under reset (i.e. during the reset phase) and at reset state (i.e. after internal reset release).
- A pull-up is applied to PB0 and PB4 under reset (i.e. during the reset phase). These two pins are input floating at reset state (i.e. after internal reset release).

Workaround

None.

No fix is planned for this limitation.

1.2.2 Restart issue when BOR0 is selected and the system clock source is not HSI at power-down

Description

In case the system clock source is not the HSI and the brown-out reset is enabled by option byte with the minimum threshold value V_{BOR0} (BOR_TH[2:0]=000), the following event occurs: if the power supply drops below V_{BOR0} (1.7 V typical) but remains higher than the power-down reset threshold V_{PDR} (1.5 V typical), the MCU does not restart when the power supply voltage rises above V_{BOR0}.

The NRST pin is high, but all internal clocks remain disabled. Consequently, the MCU cannot restart.

An external reset has no effect. The MCU will restart normally after a power-on reset.

This limitation does not occur if:

- The system clock is the HSI
- The BOR is disabled
- The BOR is enabled with a higher threshold (from V_{BOR1} to V_{BOR4}).

This limitation is not present in silicon revision Z.

Workaround

None.

1.2.3 LSE and HSE in bypass mode cannot be used as RTC clock

Description

Low speed external (LSE) and high speed external (HSE) oscillators can be configured in bypass mode if an external oscillator is used. In that case, the LSE/HSE oscillator is bypassed and the external clock signal is directly transmitted to the MCU.

If the LSE bypass or HSE bypass is used as RTC clock, the bypass configuration is lost during a reset and cannot be set again by software after reset. The external clock signal is no longer transmitted to the MCU and only a power-on reset can allow a recovery from this state.

This limitation is not present in silicon revision Z.

Workaround

None.

LSE bypass mode and HSE bypass mode (external clocks) must not be used as RTC clock.

Note: 1 *LSE and HSE crystal oscillators are not impacted by this problem.*

2 *LSE bypass or HSE bypass can be used as system clock.*

1.2.4 CCO (configurable clock output) may not be functional after a reset

Description

When enabled, the configurable clock output (CCO) may be lost after a reset. Once it is lost, enabling the CCO by software has no effect. Only a power-on reset can allow a recovery from this state. Consequently, the CCO functionality cannot be guaranteed in the application.

This limitation is not present in silicon revision Z.

Workaround

None.

1.2.5 32.768 kHz LSE crystal accuracy may be disturbed by the use of adjacent I/Os

Description

The activity on the PC4 and PC7 I/Os (input or output) can lead to missing pulses on the low speed external oscillator (32.768 kHz external crystal).

Workaround

None.

If a high LSE accuracy is required, PC4 and PC7 must be tied to V_{DD} or V_{SS}.

No fix is planned for this limitation.

1.2.6 Program memory are write unprotected after reset when embedded bootloader enabled

Description

If bootloader is enabled then before jumping into application (after 1 second timeout or after GO command) it sets incorrect MASS keys into the FLASH_PUKR register. This operation blocks future write unprotection state setting for the program memory. Because program memory has been set to write unprotected state during bootloader initialization it stays in this state until it is protected.

Workaround

If IAP operation is required in application then application must never write protect the program memory, otherwise write protection remains set to the next device reset.

1.2.7 Data memory are write unprotected after reset when embedded bootloader enabled

Description

If bootloader is enabled then before jumping into application (after 1 second timeout) the data memory remained unprotected (it was unprotected during bootloader initialization).

Workaround

To improve application robustness the application should set write protected state of data memory.

1.2.8 Program memory are write protected when embedded bootloader enabled and ROP is enabled

Description

If ROP is active but bootloader is enabled by option bytes the bootloader sets incorrect MASS keys into the FLASH_PUKR register (immediately after ROP check). This operation permanently locks program memory to write protected state.

Workaround

If IAP operation is required in application (with ROP active) then bootloader must be also disabled by option bytes.

1.2.9 Embedded bootloader entry points change

Description

Bootloader entry points was changed in new device revision. Please refer to updated UM0560 (User manual of STM8 bootloader) to entry point changes information for given device revision.

Workaround

Use entry points from UM0560 user manual according bootloader version. The bootloader version can be read from device by host (see GET bootloader command in UM0560).

1.3 I^{2}C peripheral limitations

1.3.1 I^{2}C event management

Description

As described in the I^{2}C section of the STM8L05xx, STM8L15xx, STM8L162x, STM8AL31xx and STM8AL3Lxx microcontroller family reference manual (RM0031), 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.

Workarounds

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:

1. Use the I^2C with DMA in general, except when the Master is receiving a single byte.
2. Use I^2C interrupts in nested mode and boost their priorities to the highest one in the application to make them uninterruptible.

No fix is planned for this limitation.

1.3.2 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 the 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 I^2C RxNE and TxE events with DMA or interrupts of the highest priority level, so that the condition BTF = 1 never occurs.

No fix is planned for this limitation.

1.3.3 Wrong behavior of the I^{2}C peripheral in Master mode after misplaced STOP

Description

The I^{2}C 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 I^{2}C 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 allowed to send a Stop only at the end of the full byte (8 bits + acknowledge), so this scenario is not allowed. 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 I^{2}C 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.

1.3.4 Violation of I^{2}C “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 I^{2}C specifications) may be slightly violated when the I^{2}C 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 I^{2}C 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 I^{2}C Fast mode if it is supported by the slave.

No fix is planned for this limitation.

1.3.5 In I^2C 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 I^2C 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 I^2C 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 or use DMA.

Using the “NOSTRETCH” mode with a slow I^2C 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.

No fix is planned for this limitation.

1.3.6 SMBus standard not fully supported in I^2C peripherals

Description

The I^2C peripheral is not fully compliant with the SMBus v2.0 standard since it does not support the capability to NACK an invalid byte/command.

Workarounds

A higher-level mechanism should be used to verify that a write operation is being performed correctly at the target device, such as:

- The use of the SMBA pin if supported by the host
- The alert response address (ARA) protocol
- The Host notify protocol.

No fix is planned for this limitation.

1.4 USART peripheral limitations

1.4.1 USART IDLE frame detection not supported in the case of a clock deviation

Description

An idle frame cannot be detected if the receiver clock is deviated.

If a valid idle frame of a minimum length (depending on the M and Stop bit numbers) is followed without any delay by a start bit, the IDLE flag is not set if the receiver clock is deviated from the RX line (only if the RX line switches before the receiver clock).

Consequently, the IDLE flag is not set even if a valid idle frame occurred.

Workaround

None.

No fix is planned for this limitation.

1.4.2 PE flag can be cleared in USART Duplex mode by writing to the data register

Description

The PE flag can be cleared by a read to the USART_SR register followed by a read or a write to the USART_DR register.

When working in duplex mode, the following event can occur: the PE flag set by the receiver at the end of a reception is cleared by the software transmitter reading the USART_SR (to check TXE or TC flags) and writing a new data into the USART_DR.

The software receiver can also read a PE flag at ‘0’ if a parity error occurred.

Workaround

The PE flag should be checked before writing to the USART_DR.

No fix is planned for this limitation.

1.4.3 PE flag is not set in USART Mute mode using address mark detection

Description

If, when using address mark detection, the receiver recognizes in Mute mode a valid address frame but the parity check fails, it exits from the Mute mode without setting the PE flag.

Workaround

None.

No fix is planned for this limitation.

1.4.4 IDLE flag is not set using address mark detection in the USART peripheral

Description

The IDLE flag is not set when the address mark detection is enabled, even when the USART is in Run mode (not only in Mute mode).

Workaround

None.

No fix is planned for this limitation.

1.5 DAC peripheral limitations

1.5.1 Default DAC output level when output buffer is enabled

Description

When the DAC is enabled in buffered mode configuration, the output is set to a voltage which corresponds to the code 0xFFF, whatever the data output register value. The output recovers the correct voltage as soon as a new data is written into the data holding register.

This limitation is not present in silicon revision Z.

Workaround

None.

The following software sequence must be executed at the highest speed to limit the duration of this transient behavior:

DAC->CR1=01; //Enable DAC

DAC->DHR8 = 0x0; //Update the data holding register with 0 (as an example), or with any other data.

Note: The DAC in unbuffered mode is not affected by this limitation.

1.6 ADC peripheral limitations

1.6.1 ADC V_{REF+} voltage level disturbed when the PF0 voltage is higher than V_{REF+}

Description

When V_{REF+} (ADC reference voltage pin) is connected to a voltage lower than V_{DD}, this reference voltage is disturbed when the voltage applied on PF0 is greater than V_{REF+}. This is due to a leakage from the DAC output on V_{REF+}.

Consequently, due to this V_{REF+} disturbance, the ADC is no longer functional.

This limitation is not present in silicon revision Z.

Workaround

None.

PF0 must not be used as a general purpose I/O when the ADC is used with $V_{REF+} < V_{DD}$.

Note: 1 *Only the STM8L15xC4 or STM8L15xC6 48-pin packages are impacted by this issue.*

2 *This problem cannot occur when PF0 is used as a DAC output (except in value line devices).*

1.7 LCD peripheral limitations

1.7.1 Overvoltage on VLCD when using internal step-up converter

Description

The V_{LCD} voltage delivered by the internal step-up converter can be several hundreds of mV above the programmed value if the current drawn on V_{LCD} is very low (typically below 2 µA). This corresponds to static LCD operation (1 COM only) or multiplexed LCDs with bits PON[2:0]=0 in LCD_FRQ register. This phenomenon appears for high V_{DD} values:

For V_{DD} = 3.6 V, the overvoltage is around 300 mV for all V_{LCD} values.

For V_{DD} = 3.3 V, the overvoltage is around 100 mV for all V_{LCD} values These are worst case values. The overvoltage can vary from one part to another.

This limitation is not present in silicon revision Z.

Workaround

For multiplexed LCDs, the load on V_{LCD} can be artificially increased by increasing the PON[2:0] value in the LCD_CR2 register to 2 (if DIV clock divider =16 in the LCD_FRQ register) or 4 (if DIV clock divider = 31).

For static operation, an external resistor must be added to add an extra 2 µA consumption on V_{LCD} (for instance 1.5 MΩ if V_{LCD} = 3.3 V).

1.7.2 LCD voltage levels are disturbed when V_{IN} > 3.9 V on PA7 and PE0 5 V tolerant pins

Description

Intermediate LCD voltage values on LCD segments and common lines are erroneous when V_{IN} is greater than 3.9 V on PA7 and PE0 5 V tolerant pins.

Workaround

None.

PA7 and PE0 can be used as 5V tolerant pins only when the LCD controller is disabled.
If LCD is enabled, PA7 and PE0 must be used as 3 V pins (V_{IN}<V_{DD}+0.3 V).

Note: Only STM8L052/152C6, STM8L152C4, STM8AL3L68 and STM8AL3L48 48-pin packages are impacted by this issue.

No fix is planned for this limitation.

1.8 Timer peripheral limitations

1.8.1 TIM1 advanced timer: Bad regulation for 100% PWM

Description

When the OCREFCLR functionality is activated, the OCxREF signal becomes deasserted (and consequently OCx is deasserted / OCxN is asserted) when a high level is applied on the OCREF_CLR signal. Then, the PWM restarts (output re-enabled) at the next counter overflow.

But if the PWM is configured at 100% (CCxR->ARR), then it does not restart and OCxREF remains de-asserted.

Consequently, current feedbacks cannot be generated without programming a minimum off-time (there cannot be a 100% PWM for this usage).

Workaround

None.

No fix is planned for this limitation.

Appendix A Revision code on device marking

The following figures show the standard marking compositions for the UFQFPN48, LQFP48, LQFP32, UFQFPN32, UFQFPN28 and WLCSP28 packages, respectively. Only the Additional information field containing the revision code is shown.

Figure 1. UFQFPN48 top package view

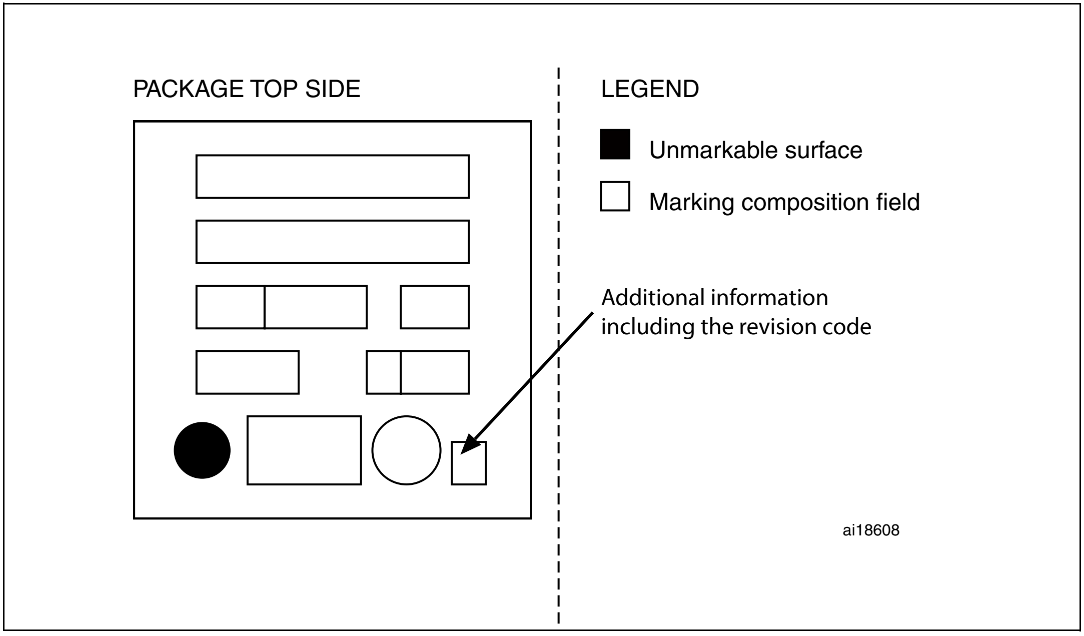


Figure 2. LQFP48 and LQFP32 top package view

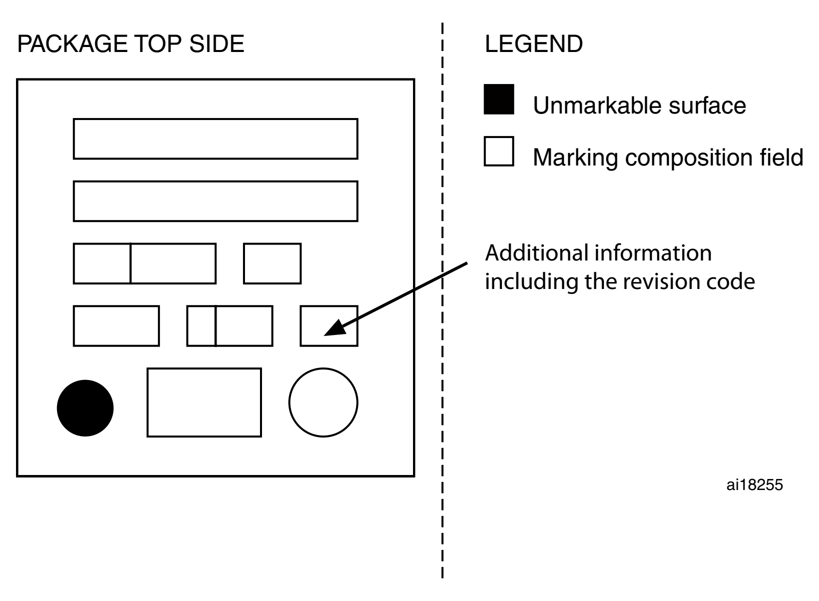


Figure 3. UFQFPN32 top package view

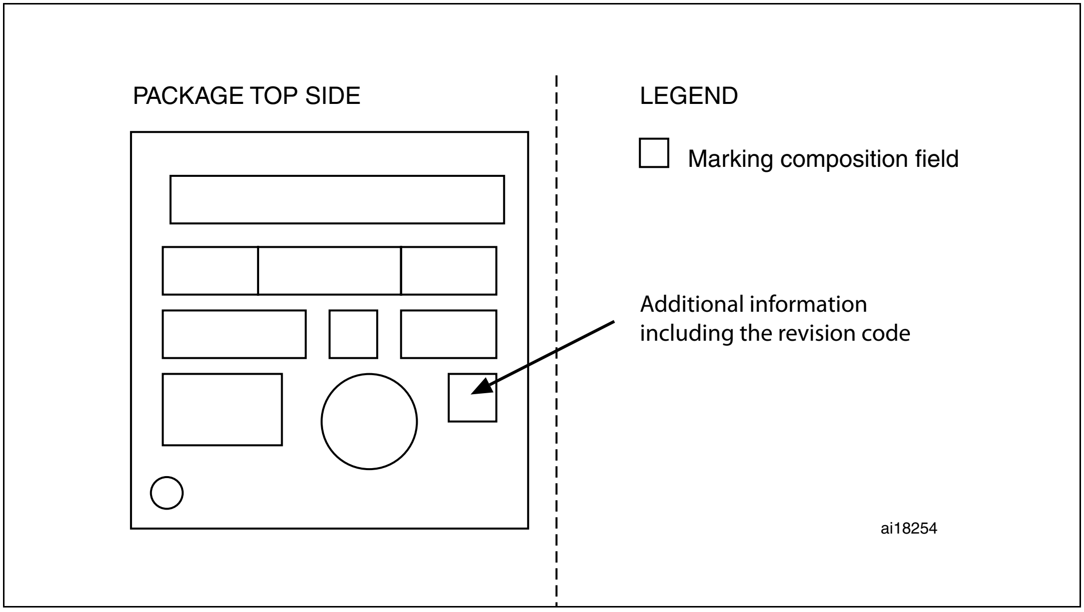


Figure 4. UFQFPN28 top package view

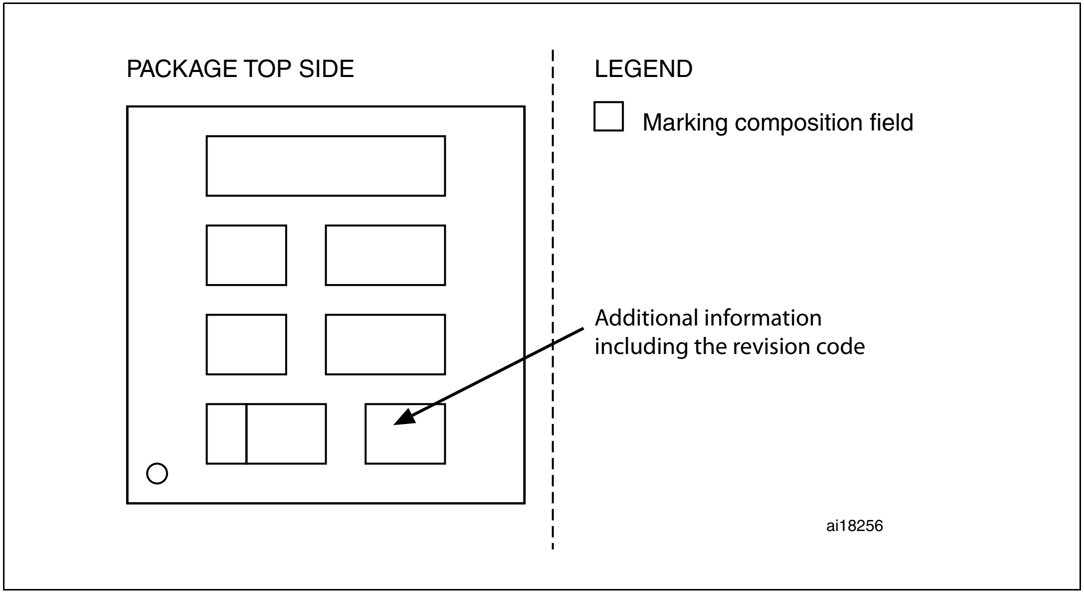
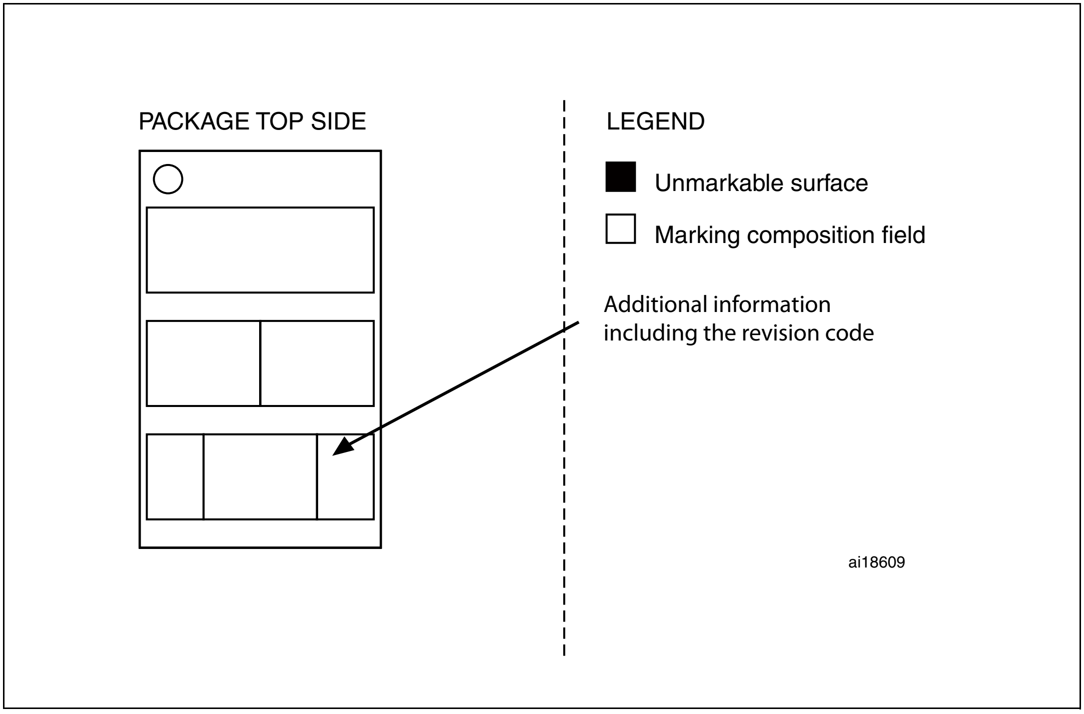


Figure 5. WLCSP28 top package view



2 Revision history

Table 4. Document revision history

Date	Revision	Changes
16-Jun-2009	1	Initial release.
10-Jul-2009	2	Added STM8L151Gx devices Added *Section 1.5: Wrong DAC maximum total unadjusted error (TUE) value on 32-pin and 28-pin packages on page 5*. and *Section 1.6: Extra consumption when VDD is below the brownout reset 0 (BOR0) threshold on page 5*.
13-Jul-2009	3	Added *Section 1.7: Active-halt consumption is higher than expected on page 5*
11-Sep-2009	4	Removed: *Start of the low-speed external clock*. Added: *Active-halt consumption is higher than expected*, *LCD consumption with internal reference voltage is higher than expected*, *Interrupt service routine (ISR) executed with priority of main process*.
13-Nov-2009	5	Added *Section 1.8: Start of the low speed external clock (LSE)*, *Section 1.11: Invalid code not skipped during RTC, CCO and BEEP clock selection* and *Section 1.12: Wrong package selection programmed in some samples*
02-Apr-2010	6	Added limitations of revision B of the STM8L15xx4 and STM8L15xx6 devices (removed limitations applied to revision A of the STM8L15xx4 and STM8L15xx6 devices).
04-Jun-2010	7	Added *Section 1.2.3: LSE and HSE in bypass mode cannot be used as RTC clock*, *Section 1.3: I^2C peripheral limitations*, *Section 1.4: USART peripheral limitations* and *Section 1.6: Timer limitations*
23-Jul-2010	8	Added Workaround in *Section 1.4.2: PE flag can be cleared in USART Duplex mode by writing to the data register* Modified title in *Section 1.8.1: TIM1 advanced timer: Bad regulation for 100% PWM* Added *Section 1.1.1: Interrupt service routine (ISR) executed with priority of main process*, *Section 1.2.2: Restart issue when BOR0 is selected and the system clock source is not HSI at power-down* and *Section 1.2.3: LSE and HSE in bypass mode cannot be used as RTC clock* and *Section 1.2.3: LSE and HSE in bypass mode cannot be used as RTC clock* Added *Appendix A: Revision code on device marking on page 22*
24-Aug-2010	9	Updated *Table 3*. Added *Section 1.1.2: Main CPU execution is not resumed after an ISR resets the AL bit*
31-Jan-2011	10	Added revision Z data, *Section 1.1.3: Unexpected DIV/DIVW instruction result in ISR*, *Table 2* and *Table 3*. Removed *Temperature sensor TS_Factory_CONV_V90 byte value not programmed* and *Bandgap VREFINT_Factory_CONV byte value not programmed.* Updated description of all limitations under *Section 1.3: I^2C peripheral limitations* and *Section 1.5.1: Default DAC output level when output buffer is enabled*.

Table 4. Document revision history (continued)

Date	Revision	Changes
01-Mar-2011	11	Updated *Table 3* and *Section 1.5.1: Default DAC output level when output buffer is enabled*.
01-Aug-2011	12	Added *Section 1.1.4: Incorrect code execution when WFE execution is interrupted by ISR or event* and *Section 1.1.5: Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM*.
21-Dec-2012	13	Added STM8L052C6 and STM8ALxx part numbers. Updated *Section 1.1.4: Incorrect code execution when WFE execution is interrupted by ISR or event*. Updated cover page.
18-Jun-2013	14	Added *Section 1.1.6: Incorrect code execution when FLASH/EEPROM memory wakes up from power down mode*.
03-Jul-2014	15	Added: – revision “Y” for products STM8Lxxxx devices is functionally identical to revision “Z”. – *Section 1.2.6: Program memory are write unprotected after reset when embedded bootloader enabled* – *Section 1.2.7: Data memory are write unprotected after reset when embedded bootloader enabled* – *Section 1.2.8: Program memory are write protected when embedded bootloader enabled and ROP is enabled* Updated *Table 3: Summary of STM8L05/15xx4/6 and STM8AL silicon limitations* – *Section 1.2.9: Embedded bootloader entry points change*