STM8L151x2 STM8L151x3
Errata sheet

STM8L151x2 and STM8L151x3 revision Z device limitations

Silicon identification

This errata sheet applies to revision Z of the STMicroelectronics STM8L151x2 and STM8L151x3 devices.

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 order code on the device 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
STM8L151K2, STM8L151G2, STM8L151F2	“Z”
STM8L151C3, STM8L151K3, STM8L151G3, STM8L151F3	“Z”

Table 2. Device summary

Reference	Part number
STM8L151x2	STM8L151K2, STM8L151G2, STM8L151F2
STM8L151x3	STM8L151C3, STM8L151K3, STM8L151G3, STM8L151F3

Contents

1 STM8L151x2/x3 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 STM8L151x2/3 silicon limitations

Section	Limitation	Rev. Z
Section 1.1: Core limitations	*Section 1.1.1: Interrupt service routine (ISR) executed with priority of main process*	N
	Section 1.1.2: Main CPU execution is not resumed after an ISR resets the AL bit	A
	Section 1.1.3: Unexpected DIV/DIVW instruction result in ISR	A
	Section 1.1.4: Incorrect code execution when WFE execution is interrupted by ISR	A
	Section 1.1.5: Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM	A
Section 1.2: System limitations	*Section 1.2.1: PA0, PB0 and PB4 configuration "at reset state" and "under reset"*	N
	Section 1.2.2: 32.768 kHz LSE crystal accuracy may be disturbed by the use of adjacent I/Os	N
Section 1.3: I2C peripheral limitations	*Section 1.3.1: I2C event management*	A
	Section 1.3.2: Corrupted last received data in I2C Master Receiver mode	A
	Section 1.3.3: Wrong behavior of the I2C peripheral in Master mode after misplaced STOP	A
	Section 1.3.4: Violation of I2C "setup time for repeated START condition" parameter	A
	Section 1.3.5: In I2C slave "NOSTRETCH" mode, underrun errors may not be detected and may generate bus errors	A
	Section 1.3.6: SMBus standard not fully supported in I2C peripherals	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
	Section 1.4.2: PE flag can be cleared in USART Duplex mode by writing to the data register	A
	Section 1.4.3: PE flag is not set in USART Mute mode using address mark detection	N
	Section 1.4.4: IDLE flag is not set using address mark detection in the USART peripheral	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.

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

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 program memory, an event which occurs during the WFE 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

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 is also valid for case 1.

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

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.

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 */
}
```

1.2 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 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}.

1.3 I²C peripheral limitations

1.3.1 I²C event management

Description

As described in the I²C section of the STM8L15x 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 I²C with DMA in general, except when the Master is receiving a single byte.
2. Use I²C interrupts in nested mode and boost their priorities to the highest one in the application to make them uninterruptible.

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

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

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 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.

1.3.6 SMBus standard not fully supported in I²C peripherals

Description

The I²C 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.

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.

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.

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.

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.

Appendix A Revision code on device marking

The following figures show the standard marking compositions for the LQFP48, UFQFPN28, UFQFPN20, UFQFPN32, and TSSOP20 packages, respectively. Only the Additional information field containing the revision code is shown.

Figure 1. LQFP48 top package view

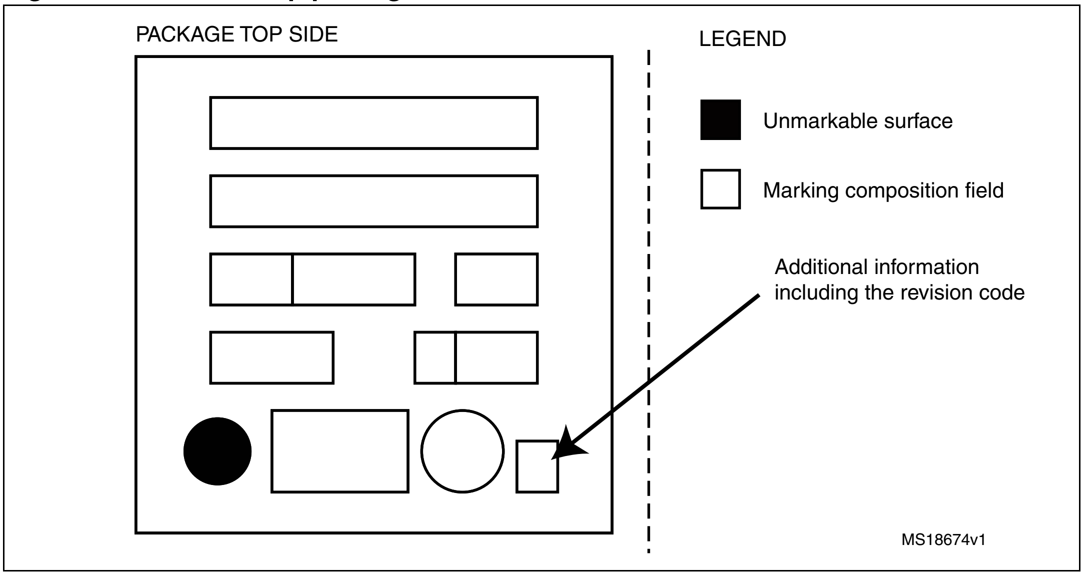


Figure 2. UFQFPN28 top package view

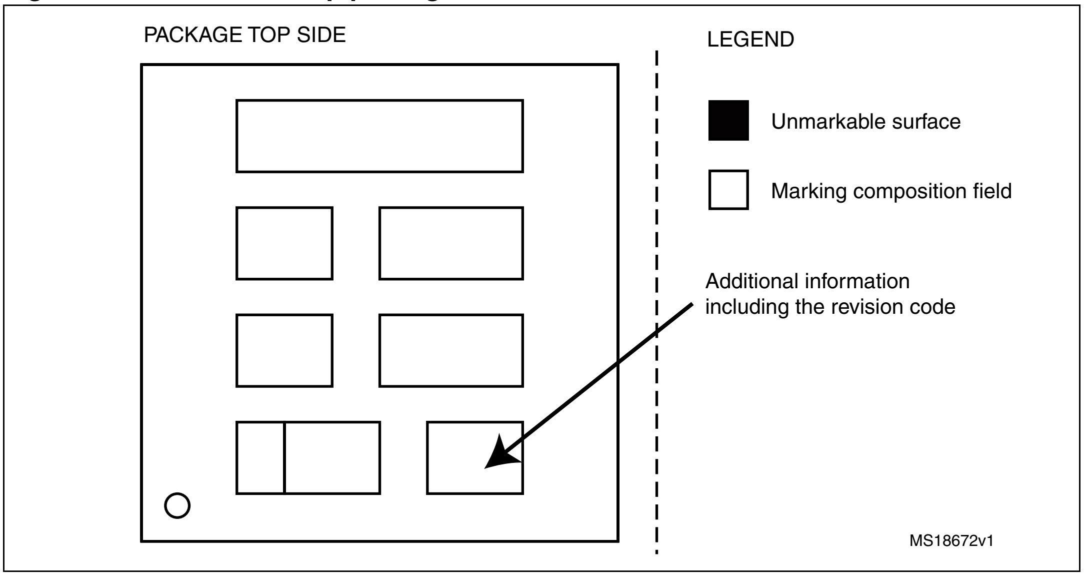


Figure 3. UFQFPN20 top package view

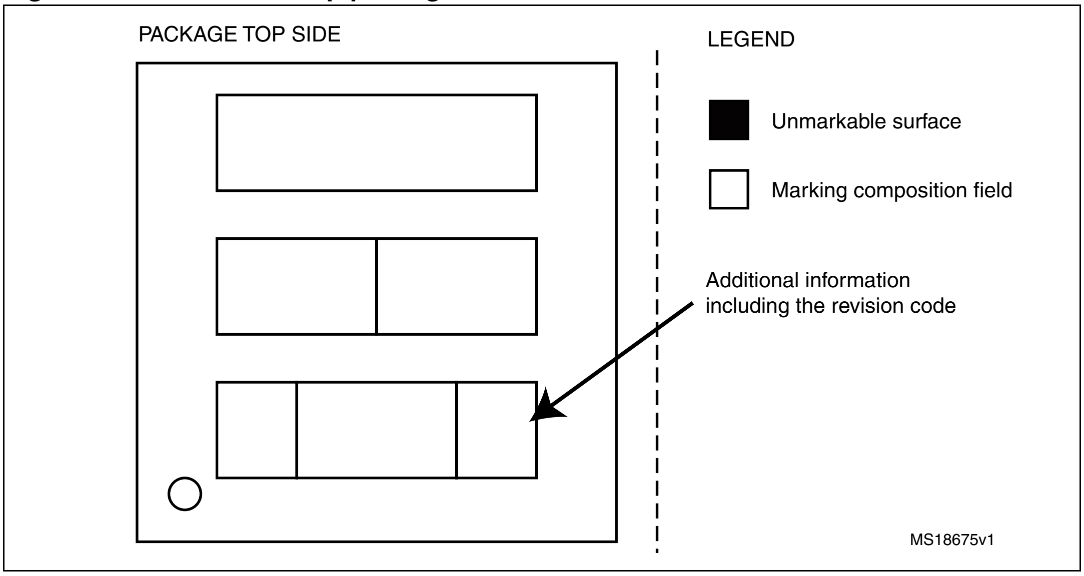


Figure 4. UFQFPN32 top package view

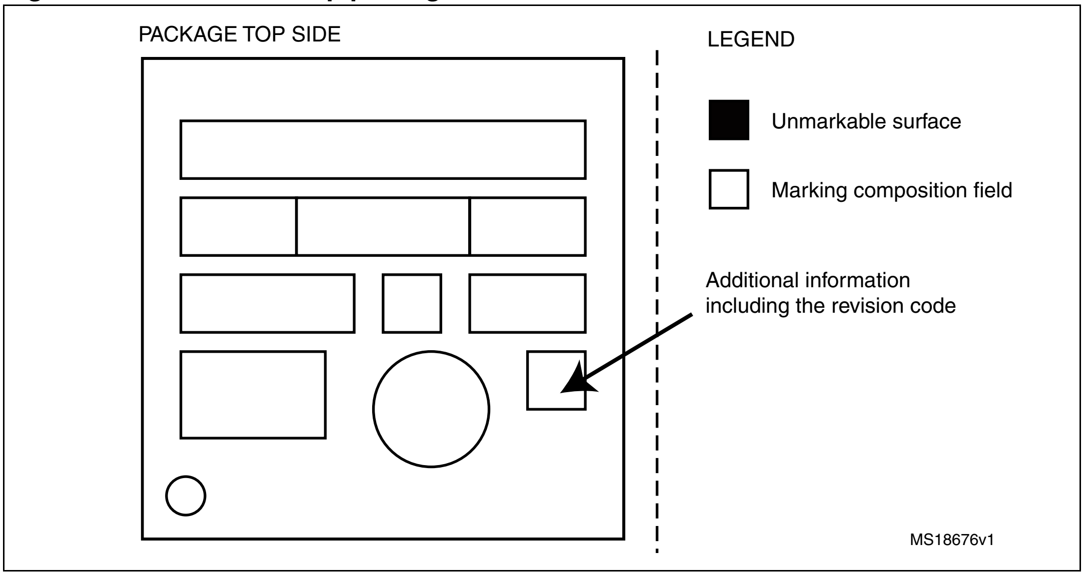
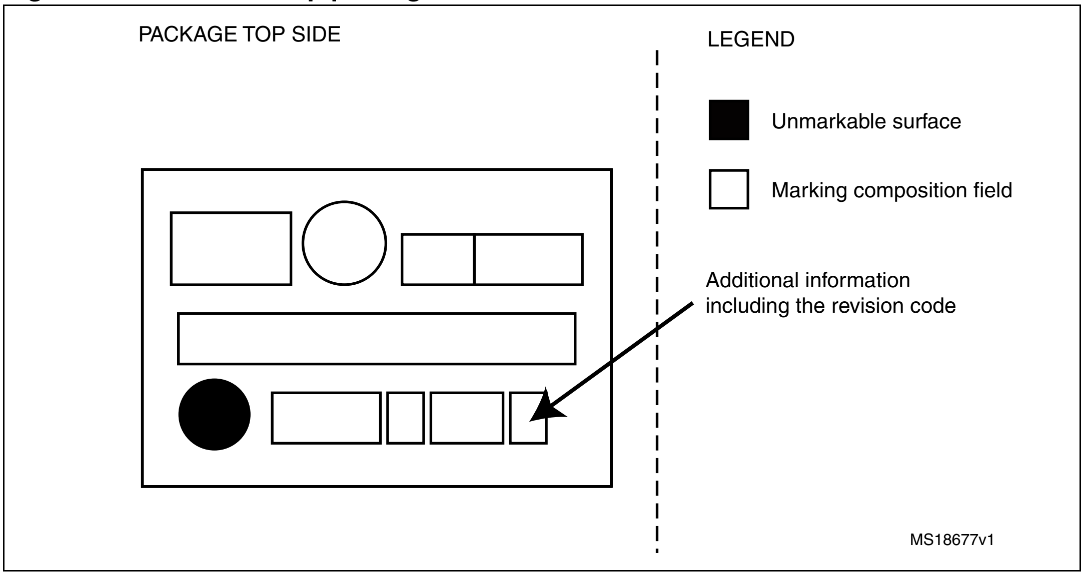


Figure 5. TSSOP20 top package view



Revision history

Table 4. Document revision history

Date	Revision	Changes
30-Jun-2011	1	Initial release.
01-Aug-2011	2	Added *Section 1.1.5: Core kept in stall mode when DMA transfer occurs during program/ erase operation to EEPROM*.