

# MC9S12NE64, Mask 1L19S

---

## Introduction

This errata sheet applies to the following devices:

- MC9S12NE64

---

## MCU Device Mask Set Identification

The mask set is identified by a 5-character code consisting of a version number, a letter, two numerical digits, and a letter, for example 1L19S. All standard devices are marked with a mask set number and a date code.

---

## MCU Device Date Codes

Device markings indicate the week of manufacture and the mask set used. The date is coded as four numerical digits where the first two digits indicate the year and the last two digits indicate the work week. For instance, the date code "0201" indicates the first week of the year 2002.

---

## MCU Device Part Number Prefixes

Some MCU samples and devices are marked with an SC, PC, or XC prefix. An SC prefix denotes special/custom device. A PC prefix indicates a prototype device which has undergone basic testing only. An XC prefix denotes that the device is tested but is not fully characterized or qualified over the full range of normal manufacturing process variations. After full characterization and qualification, devices will be marked with the MC or SC prefix.

---

## Errata System Tracking Numbers

MUCtsXXXXXX is the tracking number for device errata. It can be used with the mask set and date code to identify a specific erratum.

---

## Errata Summary

MUCts01084: DBG	
BDM firmware code execution may erroneously cause forced trigger .....	2
MUCts01493: CPU	
Tagged breakpoints missed if tag attach and interrupt are simultaneous .....	3
MUCts01898: EPHY	
Active LED flashes when data received .....	3
MUCts01899: EPHY	
NOT accepting valid LTP when greater than 100 ns .....	4
MUCts01900: EPHY	
MDIO write requires three MDC clocks to complete transaction .....	4
MUCts01967: BDM	
Possible manipulation of return address when exiting BDM active mode .....	4

---

### MUCts01084: DBG

#### BDM firmware code execution may erroneously cause forced trigger

##### Description

Breakpoints are temporarily disabled while the MCU is executing BDM firmware code when operating in active BDM mode. It logically follows that debug module triggers are disabled in the same manner. While tagged triggers are disabled, forced triggers are not and therefore may cause the debug module to trigger erroneously.

In most circumstances this will only be a problem in outside range trigger mode. In order to see an erroneous trigger in another trigger mode, a forced trigger must be configured in the BDM firmware address range (\$FF00-\$FF80) and an exact address bus match must occur. This memory area would typically contain interrupts, vectors or program code, and would therefore be a very unlikely location for the configuration of a forced trigger address.

##### Workaround

Outside range trigger mode should not be used when configuring forced triggers if the trigger range contains the memory area where the BDM firmware code resides (\$FF00-\$FF80) and the user intends to operate the MCU in active BDM mode.

---

## **MUCts01493: CPU**

### **Tagged breakpoints missed if tag attach and interrupt are simultaneous**

#### **Description**

The erratum concerns the DBG-CPU interface in DBG mode whilst configured for tagging. If an interrupt occurs at the moment that a tag is attached to an opcode being loaded into the instruction queue, the flag will get set, but the part may not enter active BDM mode. Using the DBG configuration BDM=DBGBRK=1, BEGIN=0, an event causing a flag to be set should cause a break to BDM. The flag gets set, but the part does not enter active BDM mode. The CPU executes the interrupt service routine, instead, and returns to the correct position in the program flow, but the breakpoint to BDM is missed. The problem does not occur if the DBG module is configured for operation in BKP mode (BKABEN=1). This is because, even if the flag bit is set, the BKABEN bit is not cleared. On returning from the interrupt service routine, the tag is re-applied when the PC is fetched after the interrupt service routine, and the part enters BDM after the interrupt service routine. In BKP mode with TRGSEL=0, no flags are set when a taghit occurs. In BKP mode with TRGSEL=1, the flag is also set erroneously on entering the interrupt service routine. However, it is unlikely that a user would be affected by the flag being set early (unless the service routine were exceptionally long), due to the length of time needed to read out the DBGSR (flag bits) over the BKGD pin; typically, during this time, the part would enter active BDM when the tag is re-applied.

#### **Workaround**

None.

---

## **MUCts01898: EPHY**

### **Active LED flashes when data received**

#### **Description**

When the EPHY is enabled and receiving data, the active LED (ACTLED) blinks too fast to be visible.

#### **Workaround**

The LED pin can be controlled via software.

The low level driver provided by Freescale disables LED control from the EPHY and uses software to control the port pins directly.

---

## **MUCts01899: EPHY**

### **NOT accepting valid LTP when greater than 100 ns**

#### **Description**

If a link partner's LTP is greater than 100 ns, using auto-negotiation, then the EPHY does not recognize it. This results in a failure to auto-negotiate.

#### **Workaround**

Disable auto-negotiation and configure the EPHY or link partner to operate in 100TX or 10BaseT mode.

---

## **MUCts01900: EPHY**

### **MDIO write requires three MDC clocks to complete transaction**

#### **Description**

At the end of a write cycle, three additional MDC clocks must be supplied before the completion of the transaction.

#### **Workaround**

Follow an MDIO write with an MDIO read of any MDIO register.

---

## **MUCts01967: BDM**

### **Possible manipulation of return address when exiting BDM active mode**


#### **Description**

Upon leaving BDM active mode, the CPU return address is stored temporarily for a few cycles in the BDM shift register. If a BDM command transmission is detected during this time, the return address will be manipulated in the BDM shift register. This situation is likely to occur when a CPU BGND instruction is executed in user code during debugging under the following conditions:

1. The BDM module is not enabled

AND

2. BDM commands are sent from the host



If this situation occurs, the CPU will execute BDM firmware and will check the status of the ENBDM bit in the BDMSTS register. If the BDM is disabled, the ENBDM bit will be clear, and hence the BDM firmware will be exited and the shift register manipulation described above will occur.

### **Workaround**

Avoid using the BGND instruction when the ENBDM bit in the BDMSTS register is cleared.

---

