AN14266

I2C Monitor Mode Using i.MX RT500

Rev. 1.0 — 1 April 2024 Application note

Document information

Information	Content
Keywords	AN14266, I2C, Monitor mode, i.MX RT500
Abstract	This application note introduces how to enable the Monitor mode of the I2C to facilitate debugging of the I2C during application development.

1 Introduction

The i.MX RT500 is a family of dual-core microcontrollers for embedded applications. It features an Arm Cortex-M33 CPU combined with a Cadence Xtensa Fusion F1 Audio Digital Signal Processor CPU. The Cortex-M33 includes two hardware coprocessors providing enhanced performance for an array of complex algorithms. The family offers a rich set of peripherals and a low-power consumption.

The device consists of the following:

- 5 MB SRAM
- Two FlexSPIs (octal/quad SPI Interfaces) each with 32 kB cache, one with dynamic decryption, high-speed USB device/host + PHY, 12-bit 1 MS/s ADC
- Analog comparator
- Audio subsystems supporting up to eight DMIC channels
- 2.5 D vector GPU and LCD controller with MIPI DSI PHY
- Two SDIO/eMMC
- FlexIO
- AES/SHA/Crypto M33 coprocessor
- PUF key generation

The i.MX RT500 provides connectivity interfaces such as UART, SPI, I2C, and I2S. This application note introduces how to enable the Monitor mode of the I2C. It also provides information about events on the I2C-bus to facilitate debugging of the I2C during application development.

To achieve the I2C monitor feature, perform the following steps:

- Enable the Monitor mode in the CFG register
- Enable I2C status flags to generate the interrupt when I2C even occurs
- To discover the event that triggered the interrupt and information about the event, read the INTSTAT/STAT registers

2 I2C monitor mode

The Monitor mode provides information about events on the I2C-bus, which includes data movement, data acknowledgment, start/stop events, and so on.

The time-out function provides information about I2C time-out events as follows:

- SCL time-out: Indicates when SCL has remained low for a longer time than specified by the TIMEOUT register.
- Event time-out: Indicates when the time between events has remained longer than the time specified by the TIMEOUT.

The Monitor mode and time-out function are complementary features that can be enabled independently.

When functions are enabled, the user must specify the status flags that generate an interrupt. Once done, each time an event occurs, an interrupt gets generated and can be processed in the proper interrupt handler by reading the INTSTAT register.

2.1 Registers modification

This section describes how to program the registers to enable the I2C functions required for the Monitor mode as follows:

- To enable the desired functions, one must modify the CFG register of the I2C module by enabling the corresponding bit.

Table 1. Offset

Register	Offset
CFG	800h

Table 2. Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	Reserved															
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	Reserved										HSCA PAB...	MONC LKS...	TIME OUT...	MONE N	SLVE N	MSTE N
Reset	u	u	u	u	u	u	u	u	u	u	0	0	0	0	0	0

- Therefore, to enable the monitor and time-out function, enable the bit position 2 and 3.

Table 3. Fields

Field	Function
3 TIMEOUTEN	I2C bus Time-out Enable 0b - Disabled. The time-out function is disabled. When disabled, the time-out function is internally reset. 1b - Enabled. The time-out function is enabled. If those flags are enabled, both types of time-out flags are generated and causes interrupts. Typically, only one time-out flag is used in a system.
2 MONEN	Monitor Enable 0b - Disabled. The I2C Monitor function is disabled. When disabled, the Monitor function configuration settings are not changed, but the Monitor function is internally reset. 1b - Enabled. The I2C Monitor function is enabled.

- To enable status flags that generate an interrupt, modify the INTENSET register of the I2C module by writing "1" to the corresponding bit.
- The user can overwrite the `FLEXCOMMx_IRQHandler` function to process the event each time an interrupt is generated.
- The INTSTAT register of the I2C module provides information on the active flag, that is, the source of the interrupt and the STAT register for state information. For instance, if INSTAT and 0x10000 = 1, then the MONRDY generates the interrupt.

Table 4. Offset

Register	Offset
INTSTAT	818h

Table 5. Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved						SCLTIME...	EVENTTI...	Reserved				MONIDLE	Reserved	MONOV	MONRDY
W																
Reset	u	u	u	u	u	u	0	0	u	u	u	u	0	u	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SLVDESEL	Reserved			SLVNOTS...	Reserved		SLVPEND...	Reserved	MSTSTST...	Reserved	MSTARBL...	Reserved			MSTPEND...
W																
Reset	0	u	u	u	1	u	u	0	u	0	u	0	u	u	u	1

Table 6. Offset

Register	Offset
STAT	804h

Table 7. Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved						SCLTIME...	EVENTTI...	Reserved				MONIDLE	MONACTI...	MONOV	MONRDY
W																
Reset	u	u	u	u	u	u	0	0	u	u	u	u	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SLVDESEL	SLVSEL	SLVIDX		SLVNOTS...	SLVSTATE		SLVPEND...	Reserved	MSTSTST...	Reserved	MSTARBL...	MSTSTATE			MSTPEND...
W																
Reset	0	0	0	0	1	0	0	0	u	0	u	0	0	0	0	1

- For more information on MONRDY, read the corresponding bit (position 16) in the STAT register. Therefore, the Monitor mode generates the interrupt because data is waiting to be read in the MONRXDAT register.

Table 8. Fields

Field	Function
16 MONRDY	Monitor Ready The MONRDY flag is cleared when the MONRXDAT register is read. 0b - No data. The Monitor function does not currently have data available. 1b - Data waiting. The Monitor function has data waiting to be read.

Table 9. Offset

Register	Offset
MONRXDAT	880h

Table 10. Diagram

Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	Reserved															
W																
Reset	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u	u

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	Reserved					MONNACK	MONREST...	MONSTART	MONRXDAT							
W																
Reset	u	u	u	u	u	0	0	0	0	0	0	0	0	0	0	0

Table 11. Fields

Field	Function
31-11 —	Reserved The read value is undefined; only zero must be written.
10 MONNACK	Monitor Received NACK 0b - Acknowledged. At least one master or slave receiver acknowledges the data currently being provided by the Monitor function. 1b - Not acknowledged. None of the receivers has acknowledged the data currently being provided by the Monitor function.
9 MONRESTART	Monitor Received Repeated Start 0b - No repeated start detected. The Monitor function has not detected a Repeated Start event on the I2C bus. 1b - Repeated start detected. The Monitor function has detected a Repeated Start event on the I2C bus.
8 MONSTART	Monitor Received Start 0b - No start detected. The Monitor function has not detected a Start event on the I2C bus. 1b - Start detected. The Monitor function has detected a Start event on the I2C bus.
7-0 MONRXDAT	Monitor function Receiver Data It reflects every data byte that passes on the I2C pins.

- The user can easily know the type of data waiting to be read. If MONRXDAT and 0x200 = 1, a repeated start has been detected in the I2C bus.
- The flag that generates the interrupt must be cleared after being processed. In the case of MONRDY interrupt, the flag is automatically cleared by reading the MONRXDAT register. For other cases, writing "1" to itself in the STAT register clears the flag. For instance, writing 0x1000000 clears the flag at bit position 24, that is, EVENTTIMEOUT.

For more information, refer to *i.MX RT500 Low-Power Crossover MCU Reference Manual* (document IMXRT500RM).

3 Software example

This example is based on the i.MX RT500 SDK demo `evkmimxrt595_i2c_accel_event_trigger`, which uses SDK 2.13.1 version and MCUXpresso IDE 11.7.

Modification is added to the project to enable the Monitor mode with time-out function, without impacting the behavior of the standard demo. Therefore, only related modifications are highlighted in this application note. Also, this software example introduces how to enable and apply the different I2C features.

The standard demo demonstrates how to wake up the main device in Low-power mode with the accelerometer trigger event. The accelerometer can keep working while the main device is in Low-power mode (or Deep-sleep mode). Only when the configured event is captured, the accelerometer trigger the interrupt to wake up the main device. This example uses I2C to configure the accelerometer to work in 800 Hz data rate with Low-noise mode. The main device wakes up when the tap event is triggered and 32 samples around the trigger event are captured.

3.1 Code modification

In this example, the FLEXCOMM4 is programmed as I2C to communicate with the onboard accelerometer.

Monitor and time-out features are enabled by modifying the CFG register of the I2C module as mentioned in Section 2.1 "Registers modification". This example uses I2C4 (0x40122000).

```
void APP_MonitorInit(I2C_Type *base)
{
/* set Monitor enable */
base->CFG = (base->CFG & (uint32_t)I2C_CFG_MASK) | I2C_CFG_MONEN_MASK;
}
void APP_EnableTimeOut(I2C_Type *base)
{
/* set Timeout enable */
base->CFG = (base->CFG & (uint32_t)I2C_CFG_MASK) | I2C_CFG_TIMEOUTEN_MASK;
}
```

Different flags can be set to generate an interrupt. The following flags are set related to I2C master, monitor, and time-out functions:

- Master-related flags: Master arbitration loss enabled (MSTARBLOSSEN) and Master start/stop error interrupt enabled (MSTSTSTPERREN).
- Monitor-related flags: Monitor data ready interrupt enabled (MONRDYEN).
- Time-out related flags: Event time-out interrupt enabled (EVENTTIMEOUTEN) and SCL time-out interrupt enabled (SCLTIMEOUTEN).

Note: *Depending on the application, other flags can be set/unset.*

To catch and process the interrupt, the IRQ related to the FLEXCOMM4 must be enabled:

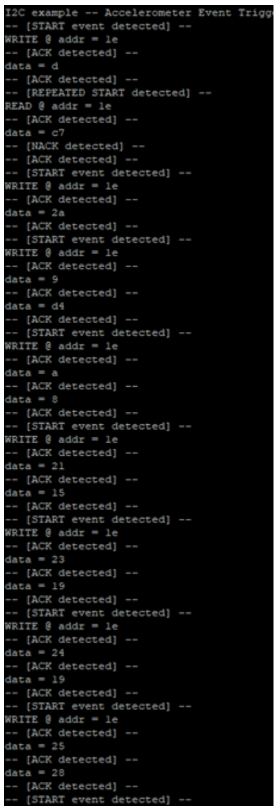
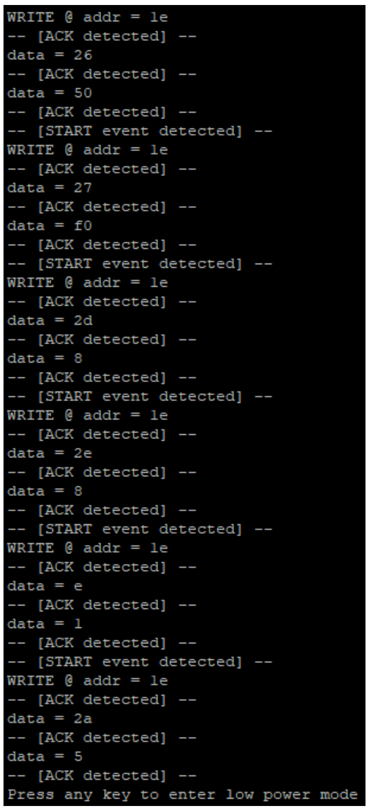
```
void APP_EnableInterrupts(I2C_Type *base)
{
uint32_t all_interrupts;
all_interrupts |= I2C_INTENSET_MONRDYEN_MASK;
all_interrupts |= I2C_INTENSET_MSTARBLOSSEN_MASK |I2C_INTENSET_MSTSTSTPERREN_MASK;
all_interrupts |= I2C_INTENSET_EVENTTIMEOUTEN_MASK | I2C_INTENSET_SCLTIMEOUTEN_MASK;
I2C_EnableInterrupts(base, all_interrupts);
DisableIRQ(FLEXCOMM4_IRQn);
IRQ_ClearPendingIRQ(FLEXCOMM4_IRQn);
EnableIRQ(FLEXCOMM4_IRQn);
}
```

The FLEXCOMM4 IRQ handler has been modified to print flags that have generated the interrupt and any available information about data content.

3.2 Results

Table 12 shows the debug console during the initialization of the accelerometer. The figure shows the correlation between the events captured by the interrupt handler, and the I2C sequence in the code mentioned in Section 3.1 "Code modification".

Table 12. Debug console

```
I2C example -- Accelerometer Event Trigg
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = d
-- [ACK detected] --
-- [REPEATED START detected] --
READ @ addr = le
-- [ACK detected] --
data = c7
-- [NACK detected] --
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 2a
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 9
-- [ACK detected] --
data = d4
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = a
-- [ACK detected] --
data = 8
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 21
-- [ACK detected] --
data = 15
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 23
-- [ACK detected] --
data = 19
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 24
-- [ACK detected] --
data = 19
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 25
-- [ACK detected] --
data = 28
-- [ACK detected] --
-- [START event detected] --
```

```
WRITE @ addr = le
-- [ACK detected] --
data = 26
-- [ACK detected] --
data = 50
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 27
-- [ACK detected] --
data = f0
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 2d
-- [ACK detected] --
data = 8
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 2e
-- [ACK detected] --
data = 8
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = e
-- [ACK detected] --
data = 1
-- [ACK detected] --
-- [START event detected] --
WRITE @ addr = le
-- [ACK detected] --
data = 2a
-- [ACK detected] --
data = 5
-- [ACK detected] --
Press any key to enter low power mode
```

4 Note about the source code in the document

5 Revision history

Table 13 summarizes the revisions to this document.

Table 13. Revision history

Document ID	Release date	Description
AN14266 v.1.0	1 April 2024	Initial public release

Contents

1	**Introduction**	**2**
2	**I2C monitor mode**	**2**
2.1	Registers modification	2
3	**Software example**	**5**
3.1	Code modification	6
3.2	Results	7
4	**Note about the source code in the document**	**7**
5	**Revision history**	**8**
	Legal information	**9**

Date of release: 1 April 2024
Document identifier: AN14266