mCube
MC36XX
Programming Guide

Rev. 1.1.1
Date 2018/09/07

9F-2, No.432, Sec. 1, Keelung Rd.,
Xinyi Dist., Taipei City 110, Taiwan
mCube-mems.
T +886-2-8780-2585 F +886-2-8780-2586

Content

1. Preface

This document describes the software package to support mCube Accelerometer – MC36XX on MCU platform. Following the introduction, software engineers can easily control MC36XX sensor for applications.

2. Software package

2.1. Source Code File Structure

FOLDER	FILE	DESCRIPTION
drv\sensor\accel\mc36xx		
	m_drv_mc36xx	Accelerometer sensor driver – MC36XX.
drv\sensor		
	m_drv_mc_utility	Driver utility to re-map orientation coordinates. (optional)
platform\bus		
	m_drv_interface	Interface to import platform SDK for I2C/SPI.
platform\console		
	m_drv_console	Interface to import platform SDK for debug message.

3. Configuration

To make MC36XX work well, customers have to do proper configuration, and hook necessary functions following the descriptions in this document.

3.1. Driver Configuration

【FILE】m_drv_mc36xx.c

1. Select the bus interface connected to MC36XX IC:
   - Only one bus interface should be selected. (exclusively)
   - Compiler would terminate with error, if improper configuration is made.

```
/*****************************************************************************
 *** CONFIGURATION
 *****************************************************************************/
/** !!! DO NOT use both I2C and SPI at the same time !!! */
//#define M_DRV_MC36XX_CFG_BUS_I2C
#define M_DRV_MC36XX_CFG_BUS_SPI
```

2. Configure I2C Address, if I2C is selected:

```
#define M_DRV_MC36XX_CFG_I2C_ADDR    (0x4C)
//#define M_DRV_MC36XX_CFG_I2C_ADDR    (0x6C)
```

3. Configure default Range and Resolution:
   - Range and Resolution can be re-configured after initialization.

```
typedef enum
{
    E_M_DRV_MC36XX_RANGE_2G = 0,
    E_M_DRV_MC36XX_RANGE_4G,
    E_M_DRV_MC36XX_RANGE_8G,
    E_M_DRV_MC36XX_RANGE_16G,
    E_M_DRV_MC36XX_RANGE_12G,
    E_M_DRV_MC36XX_RANGE_24G,
    E_M_DRV_MC36XX_RANGE_END
}   e_m_drv_mc36xx_range_t;

typedef enum
{
    E_M_DRV_MC36XX_RESOLUTION_6BIT = 0,
    E_M_DRV_MC36XX_RESOLUTION_7BIT,
    E_M_DRV_MC36XX_RESOLUTION_8BIT,
    E_M_DRV_MC36XX_RESOLUTION_10BIT,
    E_M_DRV_MC36XX_RESOLUTION_12BIT,
    E_M_DRV_MC36XX_RESOLUTION_14BIT,
    E_M_DRV_MC36XX_RESOLUTION_END,
}   e_m_drv_mc36xx_res_t;
```

```
#define M_DRV_MC36XX_CFG_RANGE                          \
        E_M_DRV_MC36XX_RANGE_4G
#define M_DRV_MC36XX_CFG_RESOLUTION                         \
        E_M_DRV_MC36XX_RESOLUTION_12BIT
```

4. Configure default sample rate for CWAKE modes:
   - Sample Rate can be re-configured after initialization.

```
typedef enum
{
   E_M_DRV_MC36XX_CWAKE_SR_LP_DUMMY_BASE = 0x00,
   E_M_DRV_MC36XX_CWAKE_SR_LP_0p4Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_0p8Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_1p7Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_7Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_14Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_27Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_54Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_106Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_210Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_411Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_606Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_DUMMY_END,

 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_DUMMY_BASE = 0x40,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p2Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p4Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p9Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_7p2Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_14Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_28Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_55Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_81Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_DUMMY_END,
}  e_m_drv_mc36xx_cwake_sr_t;
```

```
#define M_DRV_MC36XX_CFG_SAMPLE_RATE_CWAKE_DEFAULT                  \
        E_M_DRV_MC36XX_CWAKE_SR_LP_54Hz
```

5. Configure default sample rate for SNIFF modes:
   - Sample Rate can be re-configured after initialization.

```
typedef enum
{
    E_M_DRV_MC36XX_SNIFF_SR_DEFAULT_6Hz = 0,
    E_M_DRV_MC36XX_SNIFF_SR_0p4Hz,
    E_M_DRV_MC36XX_SNIFF_SR_0p8Hz,
    E_M_DRV_MC36XX_SNIFF_SR_2Hz,
    E_M_DRV_MC36XX_SNIFF_SR_6Hz,
    E_M_DRV_MC36XX_SNIFF_SR_13Hz,
    E_M_DRV_MC36XX_SNIFF_SR_26Hz,
    E_M_DRV_MC36XX_SNIFF_SR_50Hz,
    E_M_DRV_MC36XX_SNIFF_SR_100Hz,
    E_M_DRV_MC36XX_SNIFF_SR_200Hz,
    E_M_DRV_MC36XX_SNIFF_SR_400Hz,
    E_M_DRV_MC36XX_SNIFF_SR_END,
}  e_m_drv_mc36xx_sniff_sr_t;
```

```
#define M_DRV_MC36XX_CFG_SAMPLE_RATE_SNIFF_DEFAULT                    \
            E_M_DRV_MC36XX_SNIFF_SR_6Hz
```

6. Configure orientation coordinates mapping:
   - Optional. Customer can decide to remove relevant code.

```
#define M_DRV_MC36XX_CFG_ORIENTATION_MAP                              \
            E_M_DRV_UTIL_ORIENTATION_TOP_RIGHT_UP
```

   - 8 orientation mappings are enumerated in m_drv_mc_utility.h,

```
typedef enum
{
    E_M_DRV_UTIL_ORIENTATION_TOP_LEFT_DOWN = 0,
    E_M_DRV_UTIL_ORIENTATION_TOP_RIGHT_DOWN,
    E_M_DRV_UTIL_ORIENTATION_TOP_RIGHT_UP,
    E_M_DRV_UTIL_ORIENTATION_TOP_LEFT_UP,
    E_M_DRV_UTIL_ORIENTATION_BOTTOM_LEFT_DOWN,
    E_M_DRV_UTIL_ORIENTATION_BOTTOM_RIGHT_DOWN,
    E_M_DRV_UTIL_ORIENTATION_BOTTOM_RIGHT_UP,
    E_M_DRV_UTIL_ORIENTATION_BOTTOM_LEFT_UP,
    E_M_DRV_UTIL_ORIENTATION_TOTAL_CONFIG
}  E_M_DRV_UTIL_OrientationConfig;
```

- Orientation Definition:

3.2. Customized Functions

Two kinds of functions should be implemented to comply with the target platform and to complete control flow,

1. **Platform device functions:**

   Driver needs to access hardware resources by platform-dependent drivers, e.g. timer, I2C, SPI, UART, etc.

2. **MC36XX required functions:**

   Customer needs to link must functions in MC36XX driver with corresponding mechanism on system, e.g. Interrupt Service Routine. (ISR)

3.2.1. Sensor Communication Interface

【FILE】m_drv_interface.c

1. Link **Delay Function** with system delay/timer function,

```
/** Delay required milliseconds */
void mcube_delay_ms(unsigned int ms)
{
    /** Please implement delay function from platform SDK */
}
```

2. Link **m_drv_i2c_init** with system i2c initial process.

```
/** I2C init function */
int m_drv_i2c_init(void)
{
    /** Please implement I2C initial function from platform SDK */
}
```

3. Link **m_drv_spi_init** with system spi initial process.

```
/** SPI init function */
int m_drv_spi_init(void)
{
    /** Please implement SPI initial function from platform SDK */
}
```

4. Link **mcube_write_regs** with system i2c/spi write functions.

```
unsigned char mcube_write_regs(bool bSpi, unsigned char cs_pin, unsigned char register_address, unsigned char *value, unsigned char number_of_bytes)
{
    /** Please implement I2C/SPI write function from platform SDK */
    /** 0 = SPI, 1 = I2C */
    if(!bSpi) {
        /** SPI write function */
    } else {
        /** I2C write function */
    }
}
```

5. Link **mcube_read_regs** with system i2c/spi read functions.

```
unsigned char mcube_read_regs(bool bSpi, unsigned char cs_pin, unsigned char register_address, unsigned char *destination, unsigned char number_of_bytes)
{
    /** Please implement I2C/SPI read function from platform SDK */
    /** 0 = SPI, 1 = I2C */
    if(!bSpi) {
        /** SPI read function */
    } else {
        /** I2C read function */
    }
}
```

6. Link **Interrupt Handler Function** with system interrupt handler function (ISR),

- MUST!
- Refer to Chapter 4.6 and Chapter 5.2 for detailed control flow.

```
/*****************************************************************
 *** M_DRV_MC36XX_HandleINT
 *****************************************************************/
int M_DRV_MC36XX_HandleINT(S_M_DRV_MC36XX_InterruptEvent *ptINT_Event)
{
    uint_dev _bRegStatus2 = 0;

    _M_DRV_MC36XX_REG_READ(E_M_DRV_MC36XX_REG_STATUS_2, &_bRegStatus2, 1);

    ptINT_Event->bWAKE =                                    \
        _M_DRV_MC36XX_REG_STATUS_2_INT_WAKE(_bRegStatus2);
    ptINT_Event->bACQ =                                     \
        _M_DRV_MC36XX_REG_STATUS_2_INT_ACQ(_bRegStatus2);
    ptINT_Event->bFIFO_EMPTY =                                  \
        _M_DRV_MC36XX_REG_STATUS_2_INT_FIFO_EMPTY(_bRegStatus2);
    ptINT_Event->bFIFO_FULL =                                   \
        _M_DRV_MC36XX_REG_STATUS_2_INT_FIFO_FULL(_bRegStatus2);
    ptINT_Event->bFIFO_THRESHOLD =                                  \
        _M_DRV_MC36XX_REG_STATUS_2_INT_FIFO_THRESH(_bRegStatus2);
    ptINT_Event->bSWAKE_SNIFF =                                 \
        _M_DRV_MC36XX_REG_STATUS_2_INT_SWAKE_SNIFF(_bRegStatus2);

/** clear interrupt flag */
#ifdef    M_DRV_MC36XX_CFG_BUS_SPI
    _M_DRV_MC36XX_REG_WRITE(E_M_DRV_MC36XX_REG_STATUS_2, &_bRegStatus2, 1);
#endif

    return (M_DRV_MC36XX_RETCODE_SUCCESS);
}
```

3.2.1. Debug Message Interface

【FILE】m_drv_console.c

1. Link **mcube_printf** with system printf function,

```
/** Print function */
void mcube_printf(const char *format, ...)
{
    /** Please implement printf function from platform SDK */
}
```

3.3. APIs

Driver exports application interfaces (APIs) in m_drv_mc36xx.h

The present chapter describes APIs for applications to control MC36XX.

3.3.1. Driver Version

To query the version of Driver, call function M_DRV_MC36XX_GetVersion.

3.3.2. Return Codes

Driver API returns a code to indicate the result of the invoked function:

```
#define M_DRV_MC36XX_RETCODE_SUCCESS                    (0)
#define M_DRV_MC36XX_RETCODE_ERROR_BUS                  (-1)
#define M_DRV_MC36XX_RETCODE_ERROR_NULL_POINTER         (-2)
#define M_DRV_MC36XX_RETCODE_ERROR_STATUS               (-3)
#define M_DRV_MC36XX_RETCODE_ERROR_SETUP                (-4)
#define M_DRV_MC36XX_RETCODE_ERROR_GET_DATA             (-5)
#define M_DRV_MC36XX_RETCODE_ERROR_IDENTIFICATION       (-6)
#define M_DRV_MC36XX_RETCODE_ERROR_NO_DATA              (-7)
#define M_DRV_MC36XX_RETCODE_ERROR_WRONG_ARGUMENT       (-8)
```

DEFINE NAME (ignore prefix "M_DRV_MC36XX_")	VALUE	DESCRIPTION
RETCODE_SUCCESS	0	On SUCC.
RETCODE_ERROR_BUS	-1	Error on I2C / SPI.
RETCODE_ERROR_NULL_POINTER	-2	Error to access mem. addr. at zero.
RETCODE_ERROR_STATUS	-3	Warn that mode is not proper.
RETCODE_ERROR_SETUP	-4	Error to configure MC36XX register.
RETCODE_ERROR_GET_DATA	-5	Error to read data.
RETCODE_ERROR_IDENTIFICATION	-6	Error without supported sensor.
RETCODE_ERROR_NO_DATA	-7	Error without ready data to read.
RETCODE_ERROR_WRONG_ARGUMENT	-8	Error on parameters for function.

3.3.3. APIs

1. **M_DRV_MC36XX_Init**

- Initialize the MC36XX driver.
- Application should invoke this API when device is powered on, or reset.
-
```
int M_DRV_MC36XX_Init(void)
```

▼ **Parameters**

None.

▼ **Return Value**

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC36XX_RETCODE_ERROR_IDENTIFICATION: on FAIL.

  (no supported sensor can be found)

2. **M_DRV_MC36XX_SetMode**

- Switch the mode of MC36XX.
-
```
static int _M_DRV_MC36XX_SetMode(E_M_DRV_MC36XX_MODE eNextMode)
```

▼ **Parameters**

eNextMode (input)

- Specify the next mode for MC36XX to switch to.
- All modes are enumerated as below:

```
typedef enum
{
   E_M_DRV_MC36XX_WAKE_SR_MODE_LOW_POWER = 0,
   E_M_DRV_MC36XX_WAKE_SR_MODE_LOW_PRECISION,
   E_M_DRV_MC36XX_WAKE_SR_MODE_PRECISION,
   E_M_DRV_MC36XX_WAKE_SR_MODE_ULTRA_LOW_POWER,
   E_M_DRV_MC36XX_WAKE_SR_MODE_HIGH_PRECISION,
   E_M_DRV_MC36XX_WAKE_SR_MODE_END,
}  e_m_drv_mc36xx_wake_sr_mode_t;
```

▼

Return Value

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.

3. M_DRV_MC36XX_ConfigRegRngResCtrl

- Configure Range and Resolution

-
```
int M_DRV_MC36XX_ConfigRegRngResCtrl( E_M_DRV_MC36XX_RANGE eCfgRange,
                                      E_M_DRV_MC36XX_RESOLUTION eCfgResolution)
```

▼ **Parameters**

eCfgRange (input)

- Specify the range for MC36XX to detect.
- All ranges are enumerated as below:

```
typedef enum
{
   E_M_DRV_MC36XX_RANGE_2G = 0,
   E_M_DRV_MC36XX_RANGE_4G,
   E_M_DRV_MC36XX_RANGE_8G,
   E_M_DRV_MC36XX_RANGE_16G,
   E_M_DRV_MC36XX_RANGE_12G,
   E_M_DRV_MC36XX_RANGE_24G,
   E_M_DRV_MC36XX_RANGE_END
}  e_m_drv_mc36xx_range_t;
```

eCfgResolution (input)

- Specify the resolution of sensor data.
- All resolutions are enumerated as below:

```
typedef enum
{
   E_M_DRV_MC36XX_RESOLUTION_6BIT = 0,
   E_M_DRV_MC36XX_RESOLUTION_7BIT,
   E_M_DRV_MC36XX_RESOLUTION_8BIT,
   E_M_DRV_MC36XX_RESOLUTION_10BIT,
   E_M_DRV_MC36XX_RESOLUTION_12BIT,
   E_M_DRV_MC36XX_RESOLUTION_14BIT,
   E_M_DRV_MC36XX_RESOLUTION_END,
}  e_m_drv_mc36xx_res_t
```

▼ **Return Value**

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.

4. **M_DRV_MC36XX_SetSampleRate**

- Configure sample rate for both wake mode (cwake) and sniff mode

-
```
int M_DRV_MC36XX_SetSampleRate(E_M_DRV_MC36XX_CWAKE_SR eCwakeSR,
                               E_M_DRV_MC36XX_SNIFF_SR eSniffSR)
```

▼ **Parameters**

eCwakeSR (input)

- Specify the output data rate (ODR) in CWAKE mode.

- All rates are enumerated as below:

```
typedef enum
{
   E_M_DRV_MC36XX_CWAKE_SR_LP_DUMMY_BASE = 0x00,
   E_M_DRV_MC36XX_CWAKE_SR_LP_0p4Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_0p8Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_1p7Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_7Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_14Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_27Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_54Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_106Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_210Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_411Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_606Hz,
   E_M_DRV_MC36XX_CWAKE_SR_LP_DUMMY_END,

 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_DUMMY_BASE = 0x40,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p2Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p4Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_0p9Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_7p2Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_14Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_28Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_55Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_81Hz,
   E_M_DRV_MC36XX_CWAKE_SR_HIGH_PR_DUMMY_END,
}  e_m_drv_mc36xx_cwake_sr_t;
```

eSniffSR (input)

- Specify the output data rate (ODR) in SNIFF mode.
- *E_M_DRV_MC36XX_SNIFF_SR_0p4Hz*, recommended,
- All rates are enumerated as below:

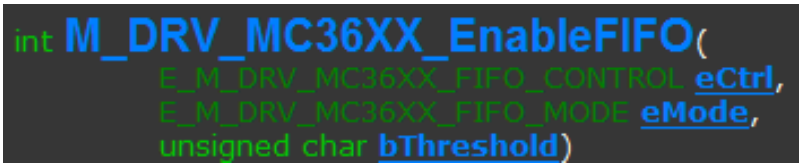
```
typedef enum
{
    E_M_DRV_MC36XX_SNIFF_SR_DEFAULT_6Hz = 0,
    E_M_DRV_MC36XX_SNIFF_SR_0p4Hz,
    E_M_DRV_MC36XX_SNIFF_SR_0p8Hz,
    E_M_DRV_MC36XX_SNIFF_SR_2Hz,
    E_M_DRV_MC36XX_SNIFF_SR_6Hz,
    E_M_DRV_MC36XX_SNIFF_SR_13Hz,
    E_M_DRV_MC36XX_SNIFF_SR_26Hz,
    E_M_DRV_MC36XX_SNIFF_SR_50Hz,
    E_M_DRV_MC36XX_SNIFF_SR_100Hz,
    E_M_DRV_MC36XX_SNIFF_SR_200Hz,
    E_M_DRV_MC36XX_SNIFF_SR_400Hz,
    E_M_DRV_MC36XX_SNIFF_SR_END,
}   e_m_drv_mc36xx_sniff_sr_t;
```

▼ Return Value

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC36XX_RETCODE_ERROR_WRONG_ARGUMENT: on FAIL.

5. **M_DRV_MC36XX_EnableFIFO**

- Configure FIFO parameter and control FIFO enabled or disabled.

- 

▼ **Parameters**

eCtrl (input)

- Enable or disable FIFO,

```
typedef enum
{
   E_M_DRV_MC36XX_FIFO_CTL_DISABLE = 0,
   E_M_DRV_MC36XX_FIFO_CTL_ENABLE,
   E_M_DRV_MC36XX_FIFO_CTL_END,
}  e_m_drv_mc36xx_fifo_ctl_t;
```

eMode (input)

- FIFO supports two modes:

. *NORMAL*: FIFO accepts new sample as long as there is available space.

. *WATERMARK*: FIFO drops new sample, once amount of samples reaches or exceeds the configured *threshold*.

```
typedef enum
{
   E_M_DRV_MC36XX_FIFO_MODE_NORMAL = 0,
   E_M_DRV_MC36XX_FIFO_MODE_WATERMARK,
   E_M_DRV_MC36XX_FIFO_MODE_END,
}  e_m_drv_mc36xx_fifo_mode_t;
```

bThreshold (input)

- Set the amount of samples for FIFO to trigger INT to notify Application.
- **0**: FIFO triggers INT_FIFO_FULL, when samples in FIFO reach 32. (FULL)
- **1 ~ 31**: FIFO triggers INT_FIFO_THRESH, when samples in FIFO reach or exceed the configured threshold.

▼ Return Value

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC36XX_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

6. M_DRV_MC36XX_ConfigINT

- Enable or disable individual interrupt.

```
int M_DRV_MC36XX_ConfigINT(
        unsigned char bFifoThreshEnable,
        unsigned char bFifoFullEnable,
        unsigned char bFifoEmptyEnable,
        unsigned char bACQEnable,
        unsigned char bWakeEnable)
```

▼ Parameters

bFifoThreshEnable (input)

- Enable (1) or disable (0) "*FIFO Threshold Interrupt"*.

eFifoFullEnable (input)

- Enable (1) or disable (0) "*FIFO Full Interrupt"*.

eFifoEmptyEnable (input)

- Enable (1) or disable (0) "*FIFO Empty Interrupt"*.

eACQEnable (input)

- Enable (1) or disable (0) "*New Sample or Acquisition Interrupt"*.

eWakeEnable (input)

- Enable (1) or disable (0) "Wake (*SNIFF to CWAKE) Interrupt"*.

▼ Return Value

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.

▼ Remark

Customer may need to configure INT according to the H/W design (schematic),

- ACTIVE LOW / HIGH,

- PUSH PULL / OPEN DRAIN

```
int M_DRV_MC36XX_ConfigINT(
        unsigned char bFifoThreshEnable,
        unsigned char bFifoFullEnable,
        unsigned char bFifoEmptyEnable,
        unsigned char bACQEnable,
        unsigned char bWakeEnable)
{

    if(s_debug) M_PRINTF("[%s] ", __func__);

    unsigned char _bPreMode = 0;

    _M_DRV_MC36XX_REG_READ(M_DRV_MC36XX_REG_MODE_C, &_bPreMode, 1);
    _M_DRV_MC36XX_SetMode(E_M_DRV_MC36XX_MODE_STANDBY);

    s_bCfgINT = (((bFifoThreshEnable & 0x01) << 6)
              | ((bFifoFullEnable & 0x01) << 5)
              | ((bFifoEmptyEnable & 0x01) << 4)
              | ((bACQEnable & 0x01) << 3)
              | ((bWakeEnable & 0x01) << 2)
              | M_DRV_MC36XX_INTR_C_IAH_ACTIVE_LOW
              | M_DRV_MC36XX_INTR_C_IPP_MODE_PUSH_PULL);

    _M_DRV_MC36XX_REG_WRITE(M_DRV_MC36XX_REG_INTR_C, &s_bCfgINT, 1);
    _M_DRV_MC36XX_SetMode(_M_DRV_MC36XX_REG_MODE_C_MODE(_bPreMode));

    return (M_DRV_MC36XX_RETCODE_SUCCESS);
} ? end M_DRV_MC36XX_ConfigINT ?
```

7. M_DRV_MC36XX_ReadData

- Read accelerometer data

```
#define M_DRV_MC36XX_AXES_NUM    3
#define M_DRV_MC36XX_FIFO_DEPTH    32
```

```
int M_DRV_MC36XX_ReadData(float faOutput[M_DRV_MC36XX_FIFO_DEPTH][M_DRV_MC36XX_AXES_NUM],
                          int nNumOfSample)
```

▼ **Parameters**

faOutput (output)

- Application should declare data buffer to store output data.
- Returned data unit: (**SI / LSB**), where SI is **m/s^2**.
- The data buffer should allocate an array of three float variables as one Sample.
- For details, refer to the chapter: Sample Code.

nNumOfSampe (input)

- Specify how many samples should be read (one sample = data of three axes)
- No larger than *M_DRV_MC36XX_FIFO_DEPTH*.

▼ **Return Value**

- ON SUCC: positive number: how many samples are read.
- ON FAIL: negative number or zero.

8. **M_DRV_MC36XX_HandleINT**

- When ISR is triggered by MC36XX INT, ISR should invoke this function to clear interrupt status.
- This handler reads individual INT status, and updates data buffer of application.
-

```
int M_DRV_MC36XX_HandleINT(S_M_DRV_MC36XX_InterruptEvent *ptINT_Event)
```

▼ **Parameters**

ptINT_Event (output)

- Application allocates structure buffer to current INT status.
- The structure holds individual event status from MC36XX,

```
typedef struct
{
    unsigned char   bWAKE;
    unsigned char   bACQ;
    unsigned char   bFIFO_EMPTY;
    unsigned char   bFIFO_FULL;
    unsigned char   bFIFO_THRESHOLD;
    unsigned char   bRESV;
    unsigned char   bSWAKE_SNIFF;
    unsigned char   baPadding[2];
} S_M_DRV_MC36XX_InterruptEvent;
```

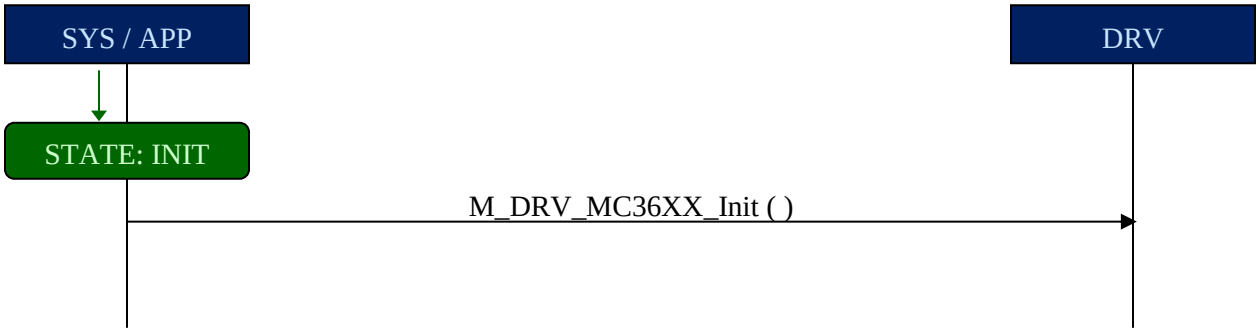
▼ **Return Value**

- M_DRV_MC36XX_RETCODE_SUCCESS, on SUCC.

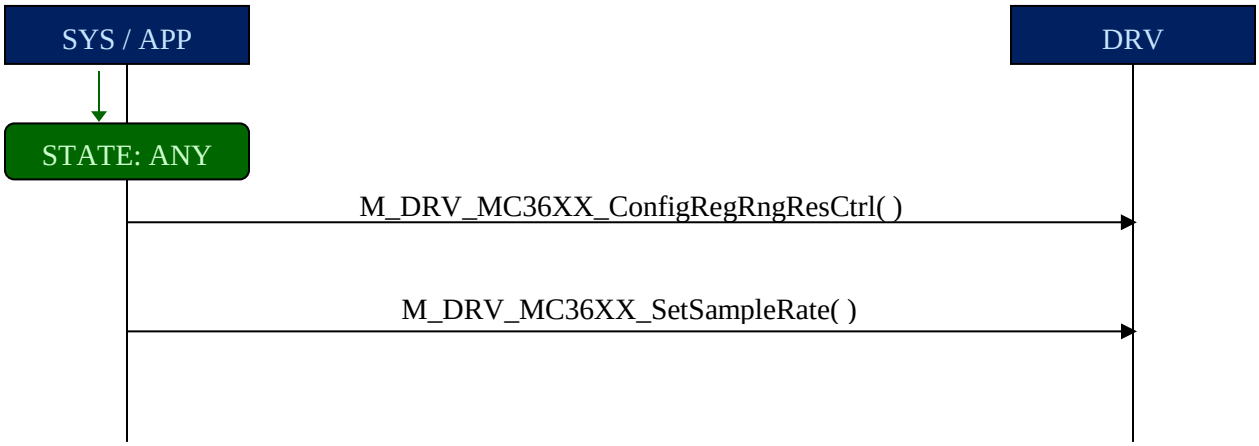
4. Control Sequence

The present chapter describes how to control MC36XX by scenarios.

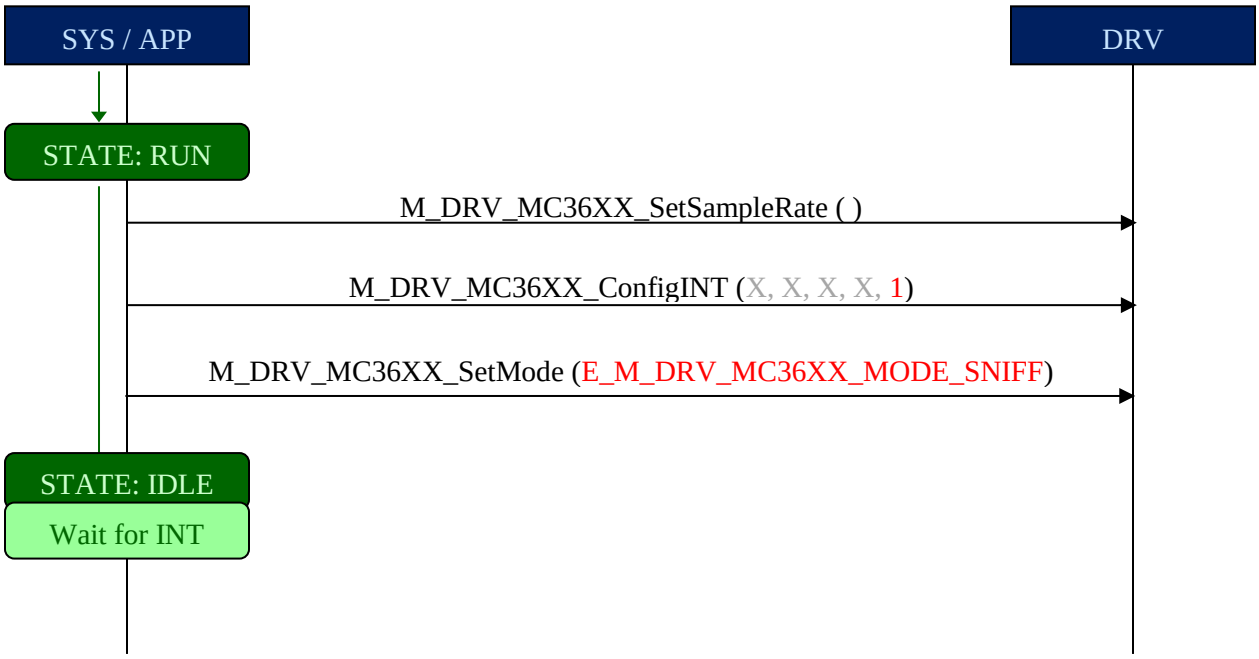
4.1. Power ON



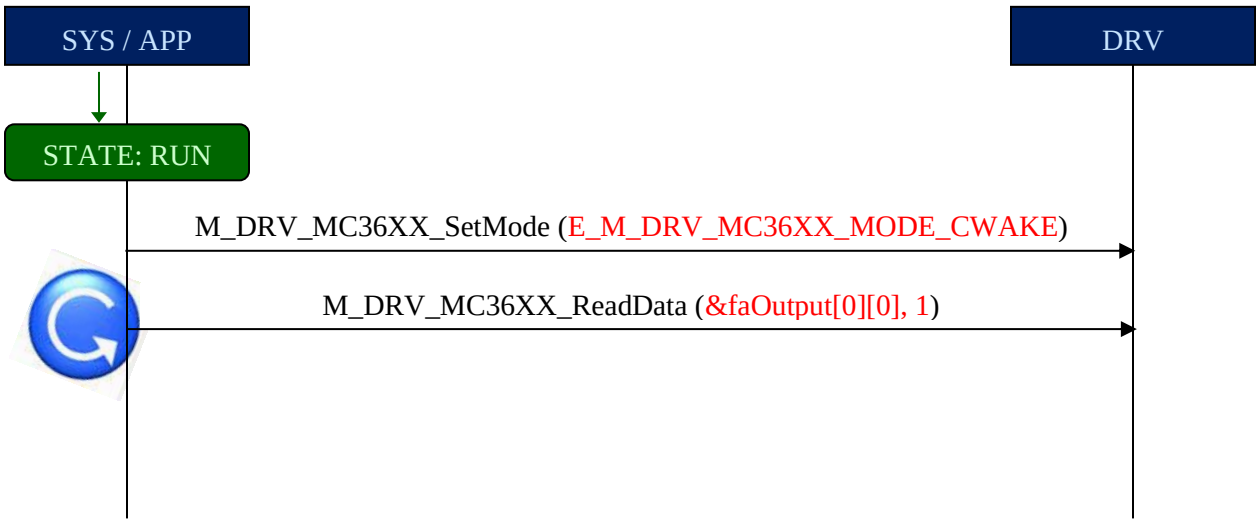
4.2. Any State (Configuration)



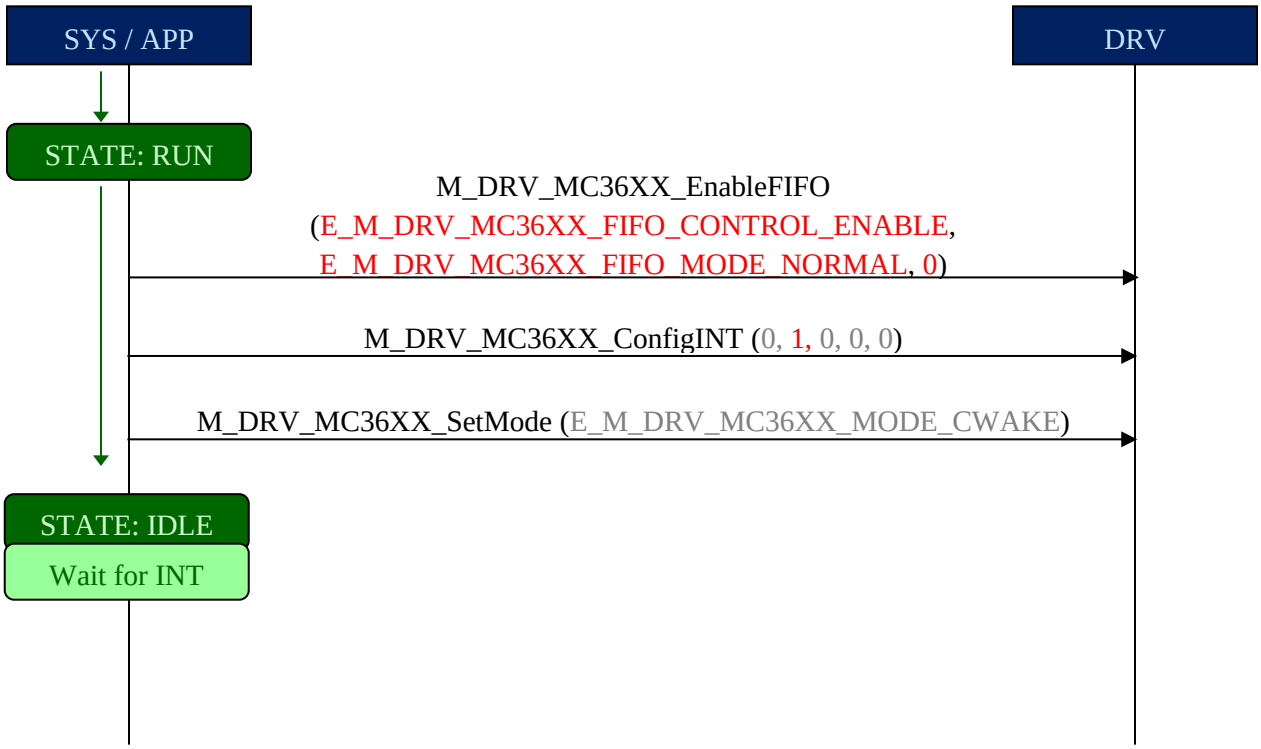
4.3. Enter SNIFF (Power Saving Mode)



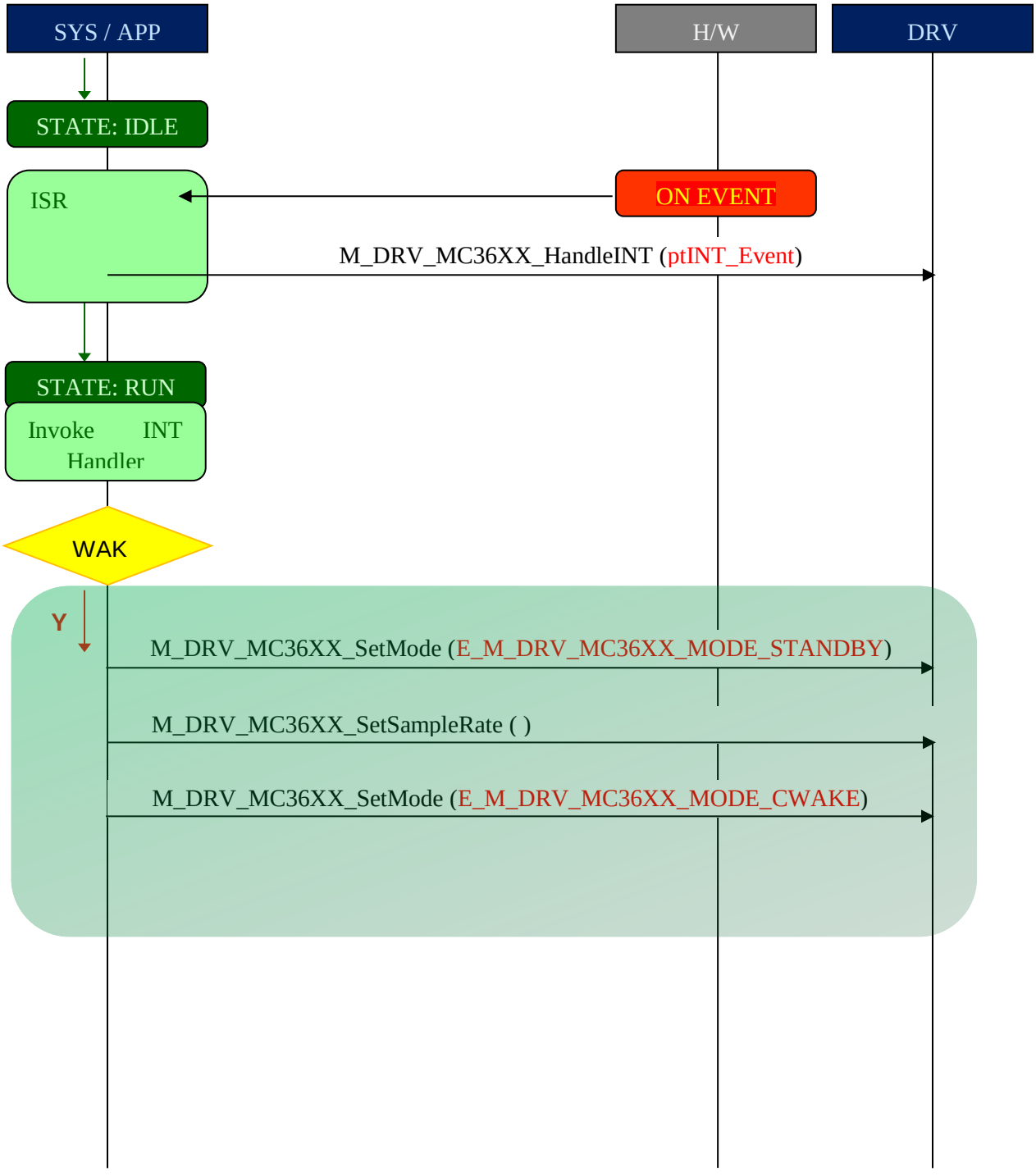
4.4. Read Data (Direct Mode)



4.5. Read Data (FIFO Mode, FULL Case)



4.6. Interrupt Handler



5. Sample Code

This chapter demonstrates code for reference.

5.1. Read Data (Direct Mode)

- FIFO should be disabled.

```
void main_read_single_data(void)
{
      float    _faOutput[1][M_DRV_MC36XX_AXES_NUM];

      if(1 == M_DRV_MC36XX_ReadData(_faOutput, 1))
      {
         M_PRINTF("%f   %f   %f   ",
                  _faOutput[0][M_DRV_MC36XX_AXIS_X],
                  _faOutput[0][M_DRV_MC36XX_AXIS_Y],
                  _faOutput[0][M_DRV_MC36XX_AXIS_Z]);
      }
      else
      {
         M_PRINTF("Read Fail (no new sample ready)" );
      }
}
```

5.2. Read Data (FIFO Mode, FULL Case)

- FIFO should be enabled.

```
void main_read_fifo_data(void)
{
    int   _nLoopIndex = 0;
    int   _nSampleCount = -1;
    float _faOutput[M_DRV_MC36XX_FIFO_DEPTH][M_DRV_MC36XX_AXES_NUM];


    _nSampleCount = M_DRV_MC36XX_ReadData(_faOutput, M_DRV_MC36XX_FIFO_DEPTH);

    if( 0 < _nSampleCount)
    {
        for( _nIndex = 0; _nIndex < _nSampleCount; _nIndex++)
        {
           M_PRINTF("[%2d] %f   %f   %f   ",
                    _nSampleCount,
                    _faOutput[_nSampleCount][M_DRV_MC36XX_AXIS_X],
                    _faOutput[_nSampleCount][M_DRV_MC36XX_AXIS_Y],
                    _faOutput[_nSampleCount][M_DRV_MC36XX_AXIS_Z]);
        }
    }
    else
    {
         M_PRINTF("Read Fail (FIFO)" );
    }
} ? end main_read_fifo_data ?
```

5.2. ISR + INT Handler

1. ISR triggered by INT,
   (1) Invoke *M_DRV_MC36XX_HandleINT* ( ),
   (2) Application should keep INT status, e.g. *main_set_int_evet* ( ).

```
void GPIO_EVEN_IRQHandler(void)
{
   uint32_t  _dwFlags = GPIO_IntGet();
   S_M_DRV_MC36XX_InterruptEvent  _tINTevent;

   NVIC_DisableIRQ(GPIO_EVEN_IRQn);

   M_DRV_MC36XX_HandleINT(&_tINTevent);

   GPIO_IntClear(_dwFlags);

   main_set_int_event(&_tINTevent);

   NVIC_EnableIRQ(GPIO_EVEN_IRQn);
}
```

2. main_set_int_event ( ),

```
void main_set_int_event(S_M_DRV_MC36XX_InterruptEvent *ptINTevent)
{
   M_PRINTF("[%s] bACQ(%d), bWAKE(%d), bFIFO_EMPTY(%d), bFIFO_FULL(%d), bFIFO_THRESHOLD(%d)",
         __func__          ,
         ptINTevent->bACQ      , ptINTevent->bWAKE     ,
         ptINTevent->bFIFO_EMPTY, ptINTevent->bFIFO_FULL, ptINTevent->bFIFO_THRESHOLD     );

   s_tINTeventW.bACQ          += ptINTevent->bACQ;
   s_tINTeventW.bWAKE         += ptINTevent->bWAKE;
   s_tINTeventW.bFIFO_EMPTY   += ptINTevent->bFIFO_EMPTY;
   s_tINTeventW.bFIFO_FULL    += ptINTevent->bFIFO_FULL;
   s_tINTeventW.bFIFO_THRESHOLD += ptINTevent->bFIFO_THRESHOLD;
}
```

3. Application handles INTs,

```
/*****************************************************************************
 *** main
 *****************************************************************************/
int     main(void)
{
    ...

    while (1)
    {
        if (s_tINTeventR.bWAKE != s_tINTeventW.bWAKE)
        {
            // Device Wakes...

            s_tINTeventR.bWAKE++;
        }

        if (    (s_tINTeventR.bFIFO_FULL      != s_tINTeventW.bFIFO_FULL     )
            || (s_tINTeventR.bFIFO_THRESHOLD != s_tINTeventW.bFIFO_THRESHOLD))
        {
            // Data in FIFO...

            if (s_tINTeventR.bFIFO_FULL != s_tINTeventW.bFIFO_FULL)
                s_tINTeventR.bFIFO_FULL++;

            if (s_tINTeventR.bFIFO_THRESHOLD != s_tINTeventW.bFIFO_THRESHOLD)
                s_tINTeventR.bFIFO_THRESHOLD++;
        }

        if (s_tINTeventR.bACQ != s_tINTeventW.bACQ)
        {
            // New sample ready...

            s_tINTeventR.bACQ++;
        }

        if (s_tINTeventR.bFIFO_EMPTY != s_tINTeventW.bFIFO_EMPTY)
        {
            // FIFO is empty...

            s_tINTeventR.bFIFO_EMPTY++;
        }
    }

    return (0);
}
```