mCube
Mensa
Programming Guide

Rev. 1.0.1
Date 2017/2/2

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

0. Document Information

0.1. Revision History:

Version	Date	Changes/Revision History
1.0.0	2016/12/13	Initial alpha release. Documented first known errata.
1.0.1	2017/2/2	Update figure 39.

0.2. Content

0.3. Figures

1. Preface

This document describes the software package which used to support mCube accelerometer - Mensa on MCU platform. Following the introduction, software engineers can easily control Mensa sensor for different applications.

2. Software package

2.1. Source Code File Structure

FOLDER	FILE	DESCRIPTION
drv		
	m_drv_mc34x6	Accelerometer sensor driver - Mensa.
	m_drv_mc_utility	Driver utility to re-map orientation coordinates. (optional)
scenario		
	scen_mc34x6_example	Sample code to verify (1) Configuration of Range / Resolution / INT / Motion mode / LPF (2) Motion mode function – Tilt/Flip (3) Motion mode function – Tilt35 (4) Motion mode function – Any motion (5) Motion mode function – Shake (6) LPF function
root		
	main	Sample code to demonstrate how to utilize the driver.

3. Configuration

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

3.1. Driver Configuration

【FILE】/drv/m_drv_mc34x6.c

1. Configure I2C address:
   - Based on the connection of VPP pin, Mensa supports 2 groups of I2C address. Each group has 4 possible I2C address based on OTP. Default I2C address is 0x4C when VPP pin connected to Ground, or 0x6C when the VPP pin connected VDD.

```
/** I2C device address for MC34X6 */
#define M_DRV_MC34X6_I2C_ADDR_DEFAULT    (MC34X6_VPPGND_ADDR | E_M_DRV_MC34X6_ADDR_4C_6C)
```

Figure 1: Setting default I2C address.

```
#define MC34X6_VPPGND_ADDR              (0x4C)
#define MC34X6_VPPVDD_ADDR              (0x6C)

/* Available I2C address for MC34X6 sensor */
typedef enum
{
    E_M_DRV_MC34X6_ADDR_4C_6C = 0,
    E_M_DRV_MC34X6_ADDR_4D_6D,
    E_M_DRV_MC34X6_ADDR_4E_6D,
    E_M_DRV_MC34X6_ADDR_4F_6D,
    E_M_DRV_MC34X6_ADDR_END,
}   E_M_DRV_MC34X6_I2C_ADDR;
```

Figure 2: Enumeration of I2C address.

2. Configure default Range and Resolution:
   - Range and Resolution could be re-configured after initialization.
   - MC3436 could support 8-bit resolution only.

```
#define M_DRV_MC34X6_CFG_RANGE          E_M_DRV_MC34X6_RANGE_2G
#define M_DRV_MC34X6_CFG_RESOLUTION     E_M_DRV_MC34X6_RESOLUTION_16BIT
```

Figure 3: Configuration of default range and resolution.

```
/* Sensor Range */
typedef enum
{
    E_M_DRV_MC34X6_RANGE_2G = 0,
    E_M_DRV_MC34X6_RANGE_4G,
    E_M_DRV_MC34X6_RANGE_8G,
    E_M_DRV_MC34X6_RANGE_16G,
    E_M_DRV_MC34X6_RANGE_12G,
    E_M_DRV_MC34X6_RANGE_24G,
    E_M_DRV_MC34X6_RANGE_END
}   E_M_DRV_MC34X6_RANGE;

/* Sensor Resolution */
typedef enum
{
    E_M_DRV_MC34X6_RESOLUTION_16BIT = 0,
    E_M_DRV_MC34X6_RESOLUTION_8BIT,
    E_M_DRV_MC34X6_RESOLUTION_END,
}   E_M_DRV_MC34X6_RESOLUTION;
```

Figure 4: Enumeration of sensor resolution.

3. Configure default sample rate for WAKE modes:
   - Sample rate could be re-configured after initialization.
   - Sample rate is combined by RATE, CLK and OSR.

```
#define M_DRV_MC34X6_CFG_SAMPLE_RATE        (E_M_DRV_MC34X6_SR_DIVIDE_BY2 | E_M_DRV_MC34X6_SR_CLK_2P56M | E_M_DRV_MC34X6_SR_OSR_64)
```

Figure 5: Configuration of default sampling rate.

```
/* Sensor Sample Rate configuration */
typedef enum
{
    E_M_DRV_MC34X6_SR_DIVIDE_BY32 =         0,  //RATE1
    E_M_DRV_MC34X6_SR_DIVIDE_BY16           ,  //RATE2
    E_M_DRV_MC34X6_SR_DIVIDE_BY8            ,  //RATE3
    E_M_DRV_MC34X6_SR_DIVIDE_BY4            ,  //RATE4
    E_M_DRV_MC34X6_SR_DIVIDE_BY2            ,  //RATE5
    E_M_DRV_MC34X6_SR_DIVIDE_BY1            ,  //RATE6
    E_M_DRV_MC34X6_SR_DIVIDE_END            ,

    E_M_DRV_MC34X6_SR_CLK_2P56M  = (0x00 << 3),
    E_M_DRV_MC34X6_SR_CLK_1P28M  = (0x01 << 3),
    E_M_DRV_MC34X6_SR_CLK_640K   = (0x10 << 3),
    E_M_DRV_MC34X6_SR_CLK_320K   = (0x11 << 3),
    E_M_DRV_MC34X6_SR_CLK_END,

    E_M_DRV_MC34X6_SR_OSR_64     = (0x00 << 5),
    E_M_DRV_MC34X6_SR_OSR_32     = (0x01 << 5),
    E_M_DRV_MC34X6_SR_OSR_128    = (0x02 << 5),
    E_M_DRV_MC34X6_SR_OSR_256    = (0x03 << 5),
    E_M_DRV_MC34X6_SR_OSR_512    = (0x04 << 5),
    E_M_DRV_MC34X6_SR_OSR_END,
}   E_M_DRV_MC34X6_SR;  //REG RATE_0x08[2:0],CLK_0x08[4:3],OSR_0x08[7:5]
```

Figure 6: Enumeration of Sample RATE, CLK and OSR.

4. Configure default bandwidth of Low-Pass Filter (LPF) mode:
   - LPF bandwidth could be re-configured after initialization.

```
#define M_DRV_MC34X6_CFG_LPF                    E_M_DRV_MC34X6_LPF_DISABLE
```

Figure 7: Configuration of default eanbling of LPF.

```
/* LPF BW */
typedef enum
{
    E_M_DRV_MC34X6_LPF_DISABLE =    0x00,   //LPF DISABLE
    E_M_DRV_MC34X6_LPF_128Hz   =    0x01,   //BW1 ~ 128Hz@2048Hz ODR
    E_M_DRV_MC34X6_LPF_16Hz    =    0x04,   //BW4 ~  16Hz@2048Hz ODR
    E_M_DRV_MC34X6_LPF_ENABLE  =    0x08,   //LPF ENABLE
    E_M_DRV_MC34X6_LPF_END,
}   E_M_DRV_MC34X6_LPF_MODE;
```

Figure 8: Enumeration of LPF Bandwidth.

5. Configure orientation coordination mappings:
   - Optional. Customer can decide to remove relevant code.
   - 8 orientation mappings are enumerated in /drv/m_drv_mc_utility.h,

```
#define M_DRV_MC34X6_CFG_ORIENTATION_MAP    E_M_DRV_UTIL_ORIENTATION_TOP_RIGHT_UP
```

Figure 9: Configuration of default orientation coordinate.

```
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;
```

Figure 10: Enumeration of orientation coordinations.

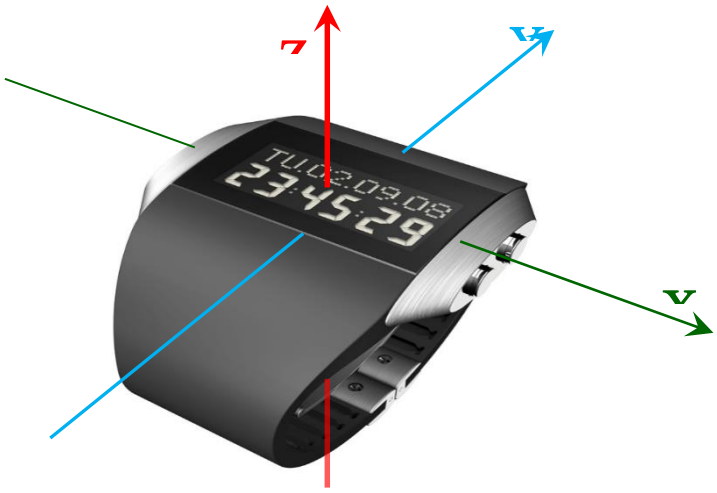


Figure 11: Example of orientation coordinations.

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, UART, I2C, etc.

2. **Mensa required functions:**

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

3.2.1. Customized Driver Function

【FILE】/drv/m_drv_mc34x6.c

1. Hook **Read / Write Function of I2C** for driver to access Mensa registers:

- Replace "***M_DRV_I2C_Write***" and "***M_DRV_I2C_Read***" by I2C – Read / Write function on platform. (marked in read rectangle)

```
#define _M_DRV_MC34X6_REG_WRITE(bRegAddr, pbDataBuf, bLength)      M_DRV_I2C_Write(M_DRV_MC34X6_I2C_WRITE_ADDR, bRegAddr, pbDataBuf, bLength)
#define _M_DRV_MC34X6_REG_READ(bRegAddr, pbDataBuf, bLength)       M_DRV_I2C_Read(M_DRV_MC34X6_I2C_READ_ADDR, bRegAddr, pbDataBuf, bLength)
```

Figure 12: Hook I2C function for driver.

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

```
/*****************************************************************************
 *** _M_DRV_MC34X6_Delay
 *****************************************************************************/
int _M_DRV_MC34X6_Delay(uint32_t dwMs) {

    // hook by custom

    return (M_DRV_TIMER_DelayMs(dwMs));
}
```

Figure 13: Delay function for driver.

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

- Do not use GetMotion and this function at same time.

```
/*****************************************************************
 *** M_DRV_MC34X6_HandleINT
 *** Do not use GetMotion and this function at same time
 *****************************************************************/
int M_DRV_MC34X6_HandleINT(S_M_DRV_MC34X6_InterruptEvent *ptINT_Event)
{

    unsigned char   _bPrjCfgResolution          = M_DRV_MC34X6_CFG_RESOLUTION;
    unsigned char   _bRegStatus2                = 0;

    if (_bPrjCfgResolution > E_M_DRV_MC34X6_RESOLUTION_16BIT)
    {
        _M_DRV_MC34X6_REG_READ(E_M_DRV_MC34X6_REG_INTR_STAT_1 , &_bRegStatus2 , 1);
        //M_PRINTF("0x%02x:%02x.", E_M_DRV_MC34X6_REG_INTR_STAT_1, _bRegStatus2);
    }
    else
    {
        _M_DRV_MC34X6_REG_READ(E_M_DRV_MC34X6_REG_INTR_STAT_2 , &_bRegStatus2 , 1);
        //M_PRINTF("0x%02x:%02x.", E_M_DRV_MC34X6_REG_INTR_STAT_2, _bRegStatus2);
    }

    ptINT_Event->bACQ_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_ACQ_INT(_bRegStatus2);
    ptINT_Event->bTILT35_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_TILT35_INT(_bRegStatus2);
    ptINT_Event->bSHAKE_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_SHAKE_INT(_bRegStatus2);
    ptINT_Event->bANYM_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_ANYM_INT(_bRegStatus2);
    ptINT_Event->bFLIP_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_FLIP_INT(_bRegStatus2);
    ptINT_Event->bTILT_INT = _M_DRV_MC34X6_REG_INTR_STAT_2_TILT_INT(_bRegStatus2);
    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 14: Interrupt Handler function for driver.

3.3. APIs

Driver exports application interfaces (APIs) in /lib/m_drv_mc34x6.h

The present chapter describes APIs for applications to control Mensa.

3.3.1. Driver Version

To query the version of Driver, read the below define in m_drv_mc34x6.c,

```
#define M_DRV_MC34X6_VERSION     "1.0.2"
```

Figure 15: Driver version.

3.3.2. Return Codes

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

```
#define M_DRV_MC34X6_RETCODE_SUCCESS                    (0)
#define M_DRV_MC34X6_RETCODE_ERROR_BUS                  (-1)
#define M_DRV_MC34X6_RETCODE_ERROR_NULL_POINTER         (-2)
#define M_DRV_MC34X6_RETCODE_ERROR_STATUS               (-3)
#define M_DRV_MC34X6_RETCODE_ERROR_SETUP                (-4)
#define M_DRV_MC34X6_RETCODE_ERROR_GET_DATA             (-5)
#define M_DRV_MC34X6_RETCODE_ERROR_IDENTIFICATION       (-6)
#define M_DRV_MC34X6_RETCODE_ERROR_NO_DATA              (-7)
#define M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT       (-8)
```

Figure 16: Definition of return codes.

DEFINE NAME (ignore prefix "M_DRV_MC34X6_")	VALUE	DESCRIPTION
RETCODE_SUCCESS	0	On SUCC.
RETCODE_ERROR_BUS	-1	Error of I2C communication.
RETCODE_ERROR_NULL_POINTER	-2	Error of access memory address at zero.
RETCODE_ERROR_STATUS	-3	Error of improper modes.
RETCODE_ERROR_SETUP	-4	Error of configure fail to Mensa registers.
RETCODE_ERROR_GET_DATA	-5	Error of read data fail.
RETCODE_ERROR_IDENTIFICATION	-6	Error of unsupported sensors.
RETCODE_ERROR_NO_DATA	-7	Error of data is not ready to read.
RETCODE_ERROR_WRONG_ARGUMENT	-8	Error of wrong parameters for function.

Table 1: Description of return codes.

3.3.3. APIs

1. **M_DRV_MC34X6_Init**

- Initialize the Mensa driver.
- Application should invoke this API when device is powered on, or reset.

▼ Parameters

None.

▼ Return Value

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

2. **M_DRV_MC34X6_SetMode**

- Switch the mode of Mensa.

▼ Parameters

eNextMode (input)

- Specify the next mode for Mensa to switch to.
- Specify the reaction of INT pin.
- All modes are enumerated as below:

```
/* Available mode in MC34X6 sensor */
typedef enum
{
    E_M_DRV_MC34X6_MODE_SLEEP = 0x00,
    E_M_DRV_MC34X6_MODE_WAKE,
    E_M_DRV_MC34X6_MODE_RESERVED,       //Reserve, not use
    E_M_DRV_MC34X6_MODE_STANDBY,
    E_M_DRV_MC34X6_MODE_END,

    E_M_DRV_MC34X6_INTR_IPP_OPEN_DRAIN_IAH_LOW =        0,  //External pullup to VDD for INT and active low
    E_M_DRV_MC34X6_INTR_IPP_MODE_PUSH_PULL_IAH_LOW = 0x40,  //Logic high drive level is VDD for INT and active low
    E_M_DRV_MC34X6_INTR_IPP_OPEN_DRAIN_IAH_HIGH =    0x80,  //External pullup to VDD for INT and active high
    E_M_DRV_MC34X6_INTR_IPP_MODE_PUSH_PULL_IAH_HIGH =0xC0,  //Logic high drive level is VDD for INT and active high
    E_M_DRV_MC34X6_INTR_END,
}   E_M_DRV_MC34X6_MODE;
```

Figure 17: Definition of sersor and inturrput modes.

▼ Return Value

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

3. **M_DRV_MC34X6_ConfigRegResRngLPFCtrl**

- Configure Resolution, Range and function of LPF.

▼ **Parameters**

eCfgRes (input)

- Specify the resolution of sensor data.

- ***MC3436 could ONLY support 8-bit resolution, 16-bit setting will be ignored.***

- All resolutions are enumerated as below:

```
typedef enum
{
    E_M_DRV_MC34X6_RESOLUTION_16BIT = 0,
    E_M_DRV_MC34X6_RESOLUTION_8BIT,
    E_M_DRV_MC34X6_RESOLUTION_END,
}   E_M_DRV_MC34X6_RESOLUTION;
```

Figure 18: Definition of resolutions.

eCfgRange (input)

- Specify the range for Mensa to detect.

- All ranges are enumerated as below:

```
/* Sensor Range */
typedef enum
{
    E_M_DRV_MC34X6_RANGE_2G = 0,
    E_M_DRV_MC34X6_RANGE_4G,
    E_M_DRV_MC34X6_RANGE_8G,
    E_M_DRV_MC34X6_RANGE_16G,
    E_M_DRV_MC34X6_RANGE_12G,
    E_M_DRV_MC34X6_RANGE_24G,
    E_M_DRV_MC34X6_RANGE_END
}   E_M_DRV_MC34X6_RANGE;
```

Figure 19: Definition of ranges.

eCfgLPF (input)

- Specify the LPF function for Mensa.
- LPF need to be enabled.
- All bandwidths are enumerated as below:

```
/* LPF BW */
typedef enum
{
    E_M_DRV_MC34X6_LPF_DISABLE =    0x00,   //LPF DISABLE
    E_M_DRV_MC34X6_LPF_128Hz   =    0x01,   //BW1 ~ 128Hz@2048Hz ODR
    E_M_DRV_MC34X6_LPF_16Hz    =    0x04,   //BW4 ~  16Hz@2048Hz ODR
    E_M_DRV_MC34X6_LPF_ENABLE  =    0x08,   //LPF ENABLE
    E_M_DRV_MC34X6_LPF_END,
}   E_M_DRV_MC34X6_LPF_MODE;
```

Figure 20: Definition of LPF.

▼ Return Value

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

4. **M_DRV_MC34X6_SetSampleRate**

- Configure sample rate for wake mode.
- The sample rate (Output Data Rate) was controlled by Rate, Clock and Output Sampling Rate (OSR).

▼ **Parameters**

eSR (input)

- Specify the rate, clock and OSR for ODR in WAKE mode.
- All rates, clock and OSR are enumerated as below:

```
/* Sensor Sample Rate configuration */
typedef enum
{
    E_M_DRV_MC34X6_SR_DIVIDE_BY32 =         0,  //RATE1
    E_M_DRV_MC34X6_SR_DIVIDE_BY16            ,  //RATE2
    E_M_DRV_MC34X6_SR_DIVIDE_BY8             ,  //RATE3
    E_M_DRV_MC34X6_SR_DIVIDE_BY4             ,  //RATE4
    E_M_DRV_MC34X6_SR_DIVIDE_BY2             ,  //RATE5
    E_M_DRV_MC34X6_SR_DIVIDE_BY1             ,  //RATE6
    E_M_DRV_MC34X6_SR_DIVIDE_END             ,

    E_M_DRV_MC34X6_SR_CLK_2P56M  = (0x00 << 3),
    E_M_DRV_MC34X6_SR_CLK_1P28M  = (0x01 << 3),
    E_M_DRV_MC34X6_SR_CLK_640K   = (0x10 << 3),
    E_M_DRV_MC34X6_SR_CLK_320K   = (0x11 << 3),
    E_M_DRV_MC34X6_SR_CLK_END,

    E_M_DRV_MC34X6_SR_OSR_64     = (0x00 << 5),
    E_M_DRV_MC34X6_SR_OSR_32     = (0x01 << 5),
    E_M_DRV_MC34X6_SR_OSR_128    = (0x02 << 5),
    E_M_DRV_MC34X6_SR_OSR_256    = (0x03 << 5),
    E_M_DRV_MC34X6_SR_OSR_512    = (0x04 << 5),
    E_M_DRV_MC34X6_SR_OSR_END,
}   E_M_DRV_MC34X6_SR;  //REG RATE_0x08[2:0],CLK_0x08[4:3],OSR_0x08[7:5]
```

Figure 21: Definition of rate, clock and OSR.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

5. **M_DRV_MC34X6_ConfigINT**

- Enable or disable individual interrupt.

▼ **Parameters**

bINTmode (input)

- Specify the interrupt modes are enable or disable.

- All interrupt modes are enumerated as below:

```
/* Sensor INT configuration */
typedef enum
{
    E_M_DRV_MC34X6_INT_DIS          = 0,
    E_M_DRV_MC34X6_INT_TILT_DIS     = 0,
    E_M_DRV_MC34X6_INT_TILT_EN      = 0x01,
    E_M_DRV_MC34X6_INT_FLIP_DIS     = 0,
    E_M_DRV_MC34X6_INT_FLIP_EN      = 0x02,
    E_M_DRV_MC34X6_INT_ANYM_DIS     = 0,
    E_M_DRV_MC34X6_INT_ANYM_EN      = 0x04,
    E_M_DRV_MC34X6_INT_SHAKE_DIS    = 0,
    E_M_DRV_MC34X6_INT_SHAKE_EN     = 0x08,
    E_M_DRV_MC34X6_INT_TILT35_DIS   = 0,
    E_M_DRV_MC34X6_INT_TILT35_EN    = 0x10,
    E_M_DRV_MC34X6_INT_AUTOCLR_DIS  = 0,
    E_M_DRV_MC34X6_INT_AUTOCLR_EN   = 0x40,
    E_M_DRV_MC34X6_INT_ACQ_DIS      = 0,
    E_M_DRV_MC34X6_INT_ACQ_EN       = 0x80,
    E_M_DRV_MC34X6_INT_END,
}   E_M_DRV_MC34X6_INTCfg;
```

Figure 22: Definition of interrupt modes.

▼ **Return Value**

■ M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

▼ **Remark**

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

Interrupt pin mode and polarity could be defined by function SetMode.

6. **M_DRV_MC34X6_ConfigMotionCtl**

- Enable or disable individual motion modes.

▼ **Parameters**

bIMotionCtl (input)

- Specify the motion mode as enable or disable.

- All motion modes are enumerated as below:

```
/* Sensor Motion configuration */
typedef enum
{
    E_M_DRV_MC34X6_MCTL_DIS                 = 0,
    E_M_DRV_MC34X6_MCTL_TF_DIS              = 0,
    E_M_DRV_MC34X6_MCTL_TF_EN               = 0x01,
    E_M_DRV_MC34X6_MCTL_MLATCH_DIS          = 0,
    E_M_DRV_MC34X6_MCTL_MLATCH_EN           = 0x02,
    E_M_DRV_MC34X6_MCTL_ANYM_DIS            = 0,
    E_M_DRV_MC34X6_MCTL_ANYM_EN             = 0x04,
    E_M_DRV_MC34X6_MCTL_SHAKE_DIS           = 0,
    E_M_DRV_MC34X6_MCTL_SHAKE_EN            = 0x08,
    E_M_DRV_MC34X6_MCTL_TILT35_DIS          = 0,
    E_M_DRV_MC34X6_MCTL_TILT35_EN           = 0x10,
    E_M_DRV_MC34X6_MCTL_ZORT_POS            = 0,
    E_M_DRV_MC34X6_MCTL_ZORT_NEG            = 0x20,
    E_M_DRV_MC34X6_MCTL_RAW_PROC_STATUS_DIS = 0,
    E_M_DRV_MC34X6_MCTL_RAW_PROC_STATUS_EN  = 0x40,
    E_M_DRV_MC34X6_MCTL_MOTION_RESET_DIS    = 0,
    E_M_DRV_MC34X6_MCTL_MOTION_RESET_EN     = 0x80,
    E_M_DRV_MC34X6_MCTL_END,
}   E_M_DRV_MC34X6_MotionCtl;
```

Figure 23: Definition of motion modes.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

7. **M_DRV_MC34X6_SetFTThreshold**

- Set threshold for function tilt/flip detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_FTTHRESHOLD (config)

- Specify the detection threshold for mode tilt/flip.

```
#define M_DRV_MC34X6_CFG_FTTHRESHOLD          200
```

Figure 24: Threshold configuration for tilt/flip mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

8. **M_DRV_MC34X6_SetFTDebounce**

- Set debounce for function tilt/flip detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_FTDEBOUNCE (config)

- Specify the detection debounce for mode tilt/flip.

```
#define M_DRV_MC34X6_CFG_FTDEBOUNCE           50
```

Figure 25: Debounce configuration for tilt/flip mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

9. **M_DRV_MC34X6_SetTILT35Threshold**

- Set threshold for tilt35 event detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_TILT35THRESHOLD (config)

- Specify the detection threshold for mode tilt35.

```
#define M_DRV_MC34X6_CFG_TILT35THRESHOLD     20
```

Figure 26: Threshold configuration for tilt35 mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

10. **M_DRV_MC34X6_SetTILT35Timer**

- Set timer for function tilt35 detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_TILT35TIMER (config)

- Specify the detection timer for mode tilt35.

```
#define M_DRV_MC34X6_CFG_TILT35TIMER          E_M_DRV_MC34X6_TILT35_2p0s
```

Figure 27: Timer configuration for tilt35 mode.

- All timer modes are enumerated as below:

```
/* Tilt35 timer configuration */
typedef enum
{
    E_M_DRV_MC34X6_TILT35_1p6s      = 0,
    E_M_DRV_MC34X6_TILT35_1p8s,
    E_M_DRV_MC34X6_TILT35_2p0s,
    E_M_DRV_MC34X6_TILT35_2p2s,
    E_M_DRV_MC34X6_TILT35_2p4s,
    E_M_DRV_MC34X6_TILT35_2p6s,
    E_M_DRV_MC34X6_TILT35_2p8s,
    E_M_DRV_MC34X6_TILT35_3p0s,
    E_M_DRV_MC34X6_TILT35_END,
}   E_M_DRV_MC34X6_TILT35_TIMER;
```

Figure 28: Definition of timer for tilt35 mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

11. **M_DRV_MC34X6_SetANYMThreshold**

- Set threshold for function any motion detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_ANYMTHRESHOLD (config)

- Specify the detection threshold for any motion.

```
#define M_DRV_MC34X6_CFG_ANYMTHRESHOLD      400
```

Figure 29: Threshold configuration for any motion mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

12. **M_DRV_MC34X6_SetANYMDebounce**

- Set debounce for function any motion detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_ANYMDEBOUNCE (config)

- Specify the detection debounce for any motion.

```
#define M_DRV_MC34X6_CFG_ANYMDEBOUNCE       30
```

Figure 30: Debounce configuration for any motion mode.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

13. **M_DRV_MC34X6_SetShakeThreshold**

- Set threshold for function shake detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_SHAKETHRESHOLD (config)

- Specify the detection threshold for shake.

```
#define M_DRV_MC34X6_CFG_SHAKETHRESHOLD      300
```

Figure 31: Threshold configuration for shake.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

14. **M_DRV_MC34X6_ SetShake_P2P_DUR_THRESH**

- Set peak to peak duration and count for function shake detection.

▼ **Parameters**

M_DRV_MC34X6_CFG_ SHAKEP2PDURATION (config)

- Specify the detection duration and count for shake.

```
#define M_DRV_MC34X6_CFG_SHAKECOUNT          1
#define M_DRV_MC34X6_CFG_SHAKEP2PDURATION    10
```

Figure 32: Duration and count configuration for shake.

▼ **Return Value**

- M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.
- M_DRV_MC34X6_RETCODE_ERROR_WRONG_ARGUMENT, on FAIL.

15. **M_DRV_MC34X6_ReadSensorData**

- Read accelerometer data

▼ **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.

▼ **Return Value**

■ M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

16. **M_DRV_MC34X6_GetMotion**

- Read the motion detection flag.

▼ **Parameters**

MotionStatus (output)

- Application should declare a data buffer to store output data.
- Returned data:

VALUE	DESCRIPTION
1	Tilt motion detection.
2	Flip motion detection.
3	Any motion detection
4	Shake motion detection.
5	Tilt35 motion detection.

Table 2: Description of detected motion.

▼ **Return Value**

■ M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

17. **M_DRV_MC34X6_HandleINT**

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

▼ **Parameters**

ptINT_Event (output)

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

```
typedef struct
{
    unsigned char    bACQ_INT;
    unsigned char    bTILT35_INT;
    unsigned char    bSHAKE_INT;
    unsigned char    bANYM_INT;
    unsigned char    bFLIP_INT;
    unsigned char    bTILT_INT;
}   S_M_DRV_MC34X6_InterruptEvent;
```

Figure 33: Definition of interrupt event.

▼ **Return Value**

■ M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

18. **M_DRV_MC34X6_ReadRegMap**

- Reading all Mensa registers.

▼ **Parameters**

baRegMap (output)

- Application should declare a data buffer to store output data.

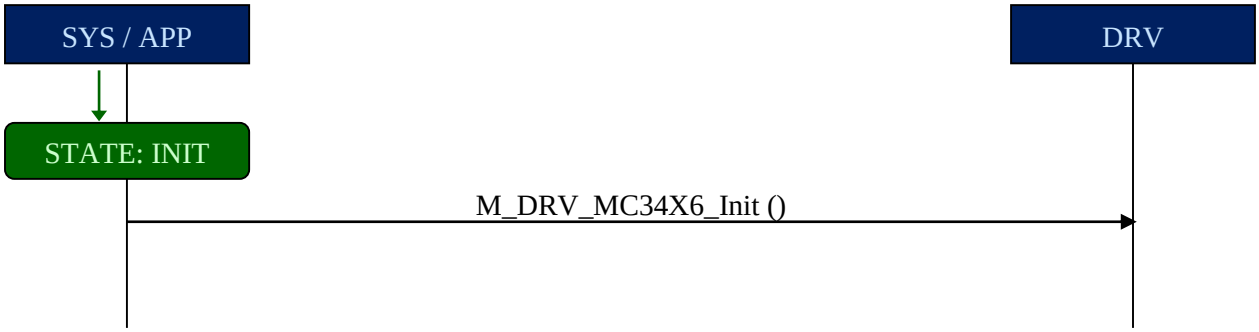
▼ **Return Value**

■ M_DRV_MC34X6_RETCODE_SUCCESS, on SUCC.

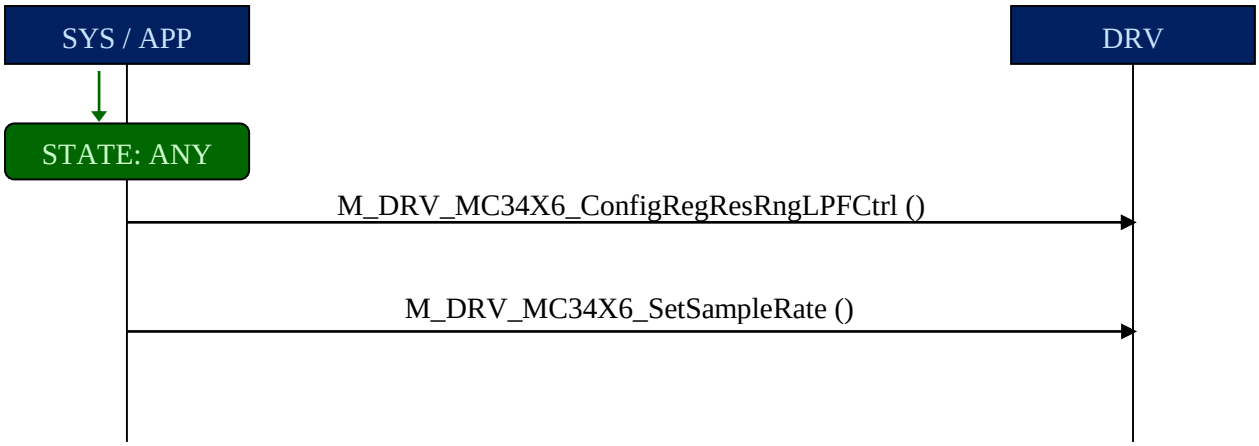
4. Control Sequence

This chapter describes how to control Mensa by scenarios.

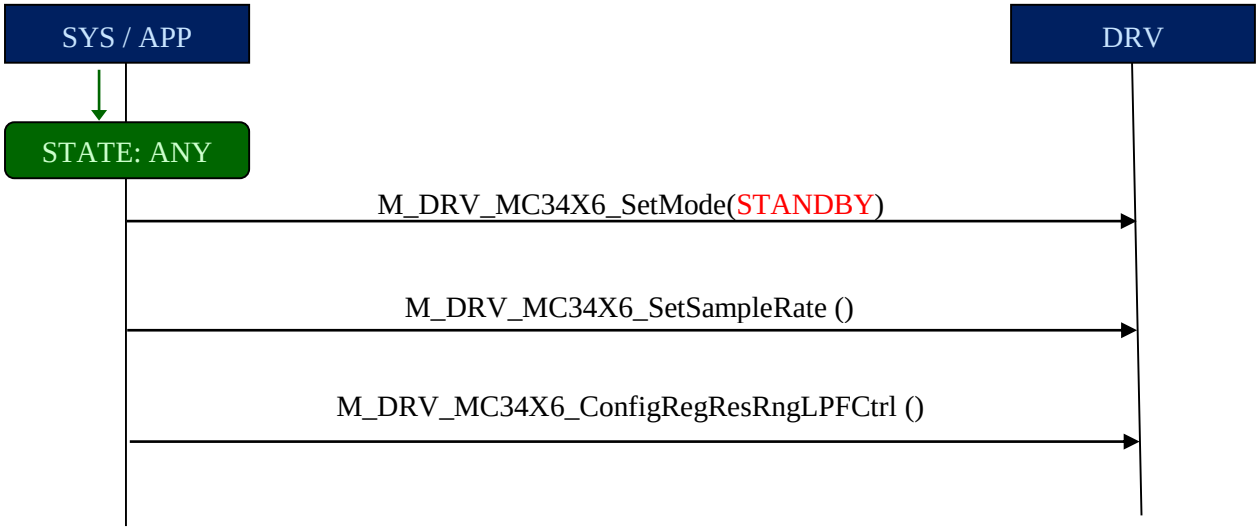
4.1. Power ON



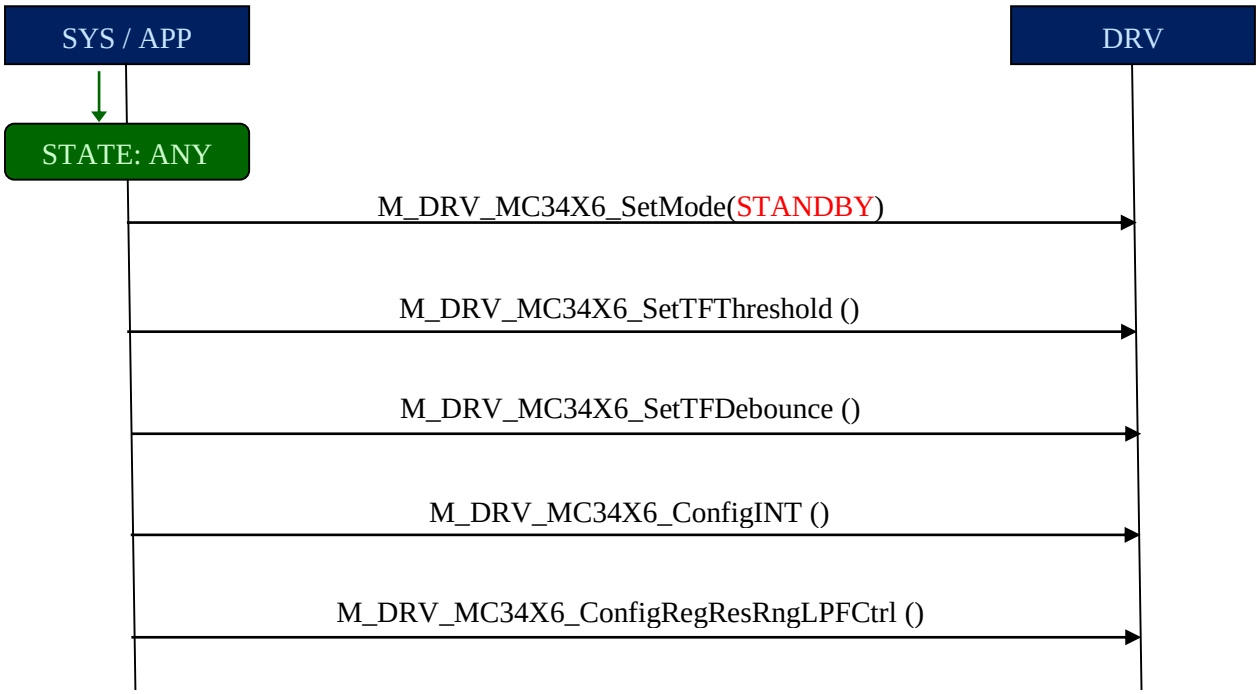
4.2. Any State (Configuration)



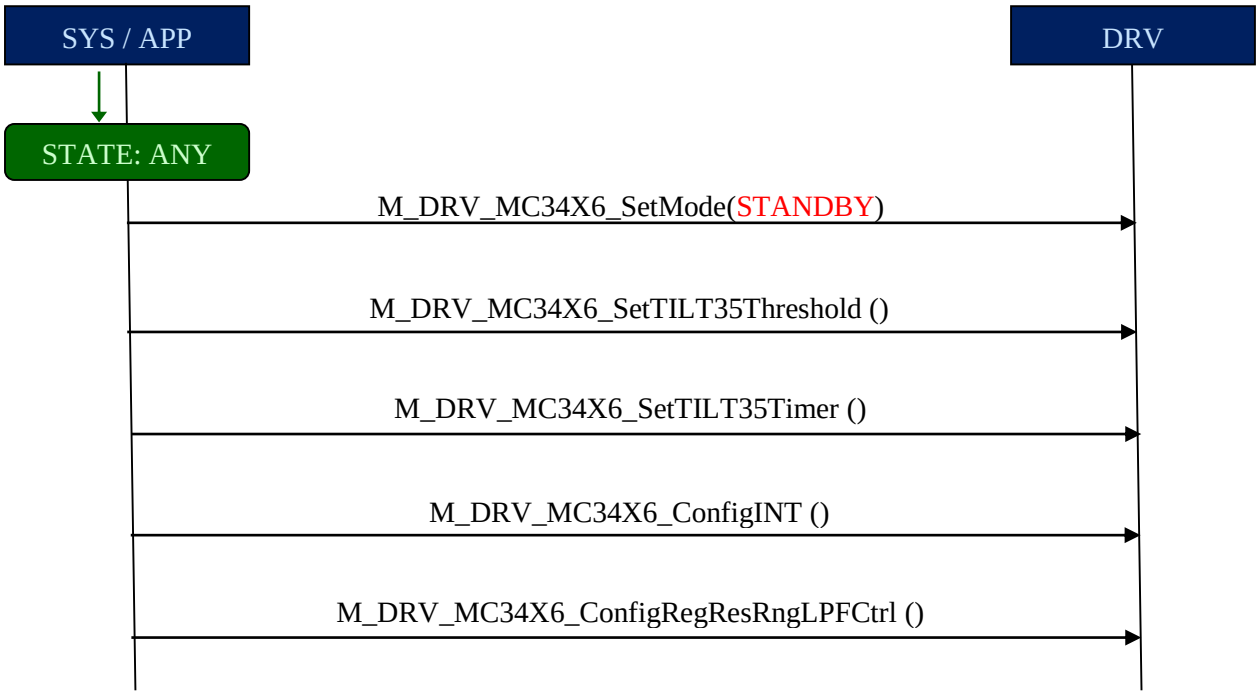
4.3. Enable LPF (Configuration)



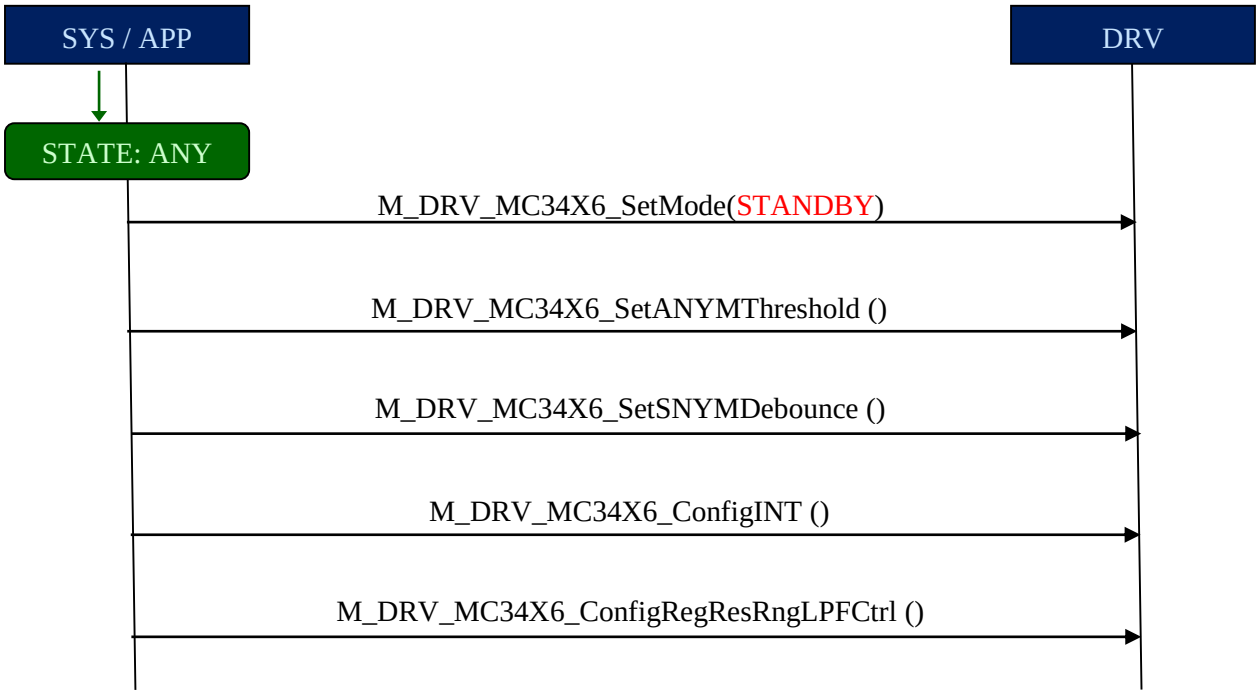
4.4. Enable Tilt/Flip Detection (Configuration)



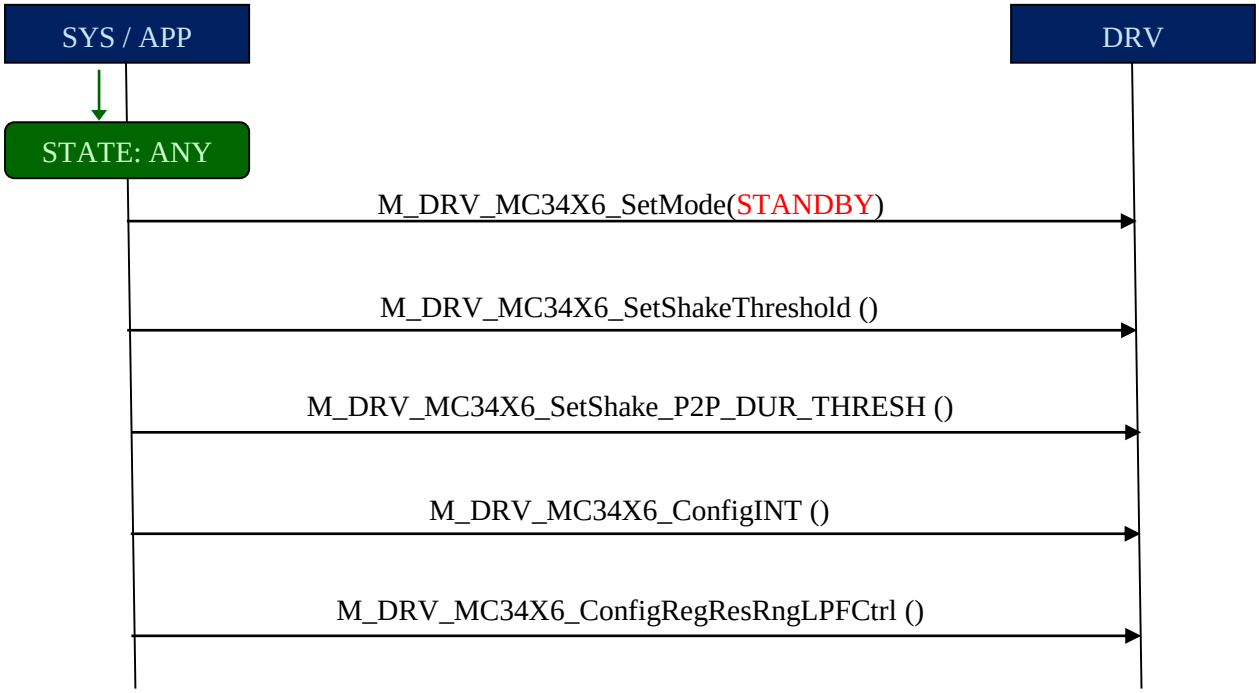
4.5. Enable Tilt35 Detection (Configuration)



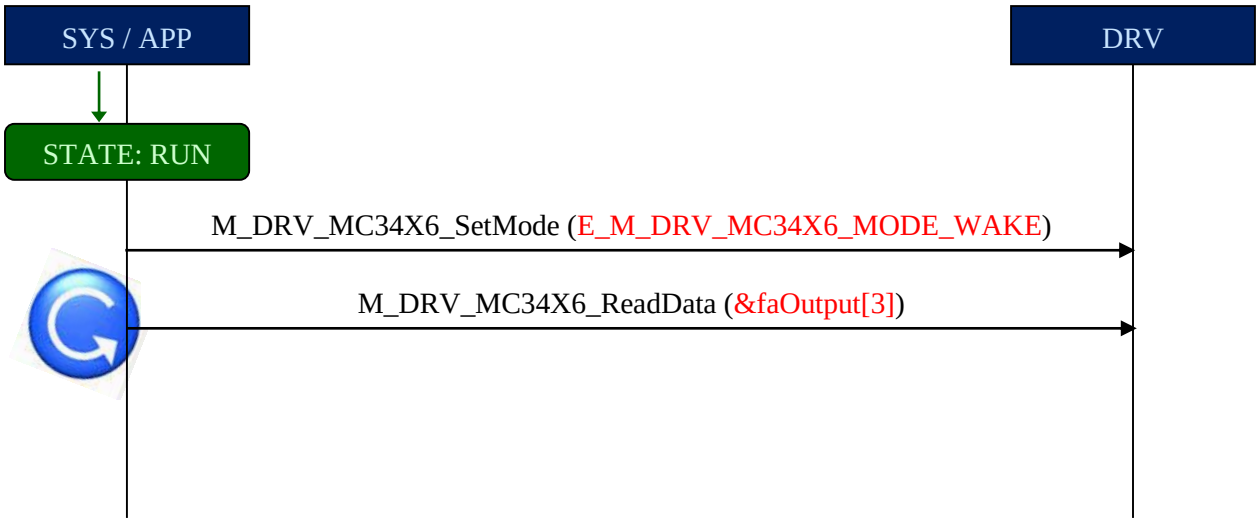
4.6. Enable ANYM Detection (Configuration)



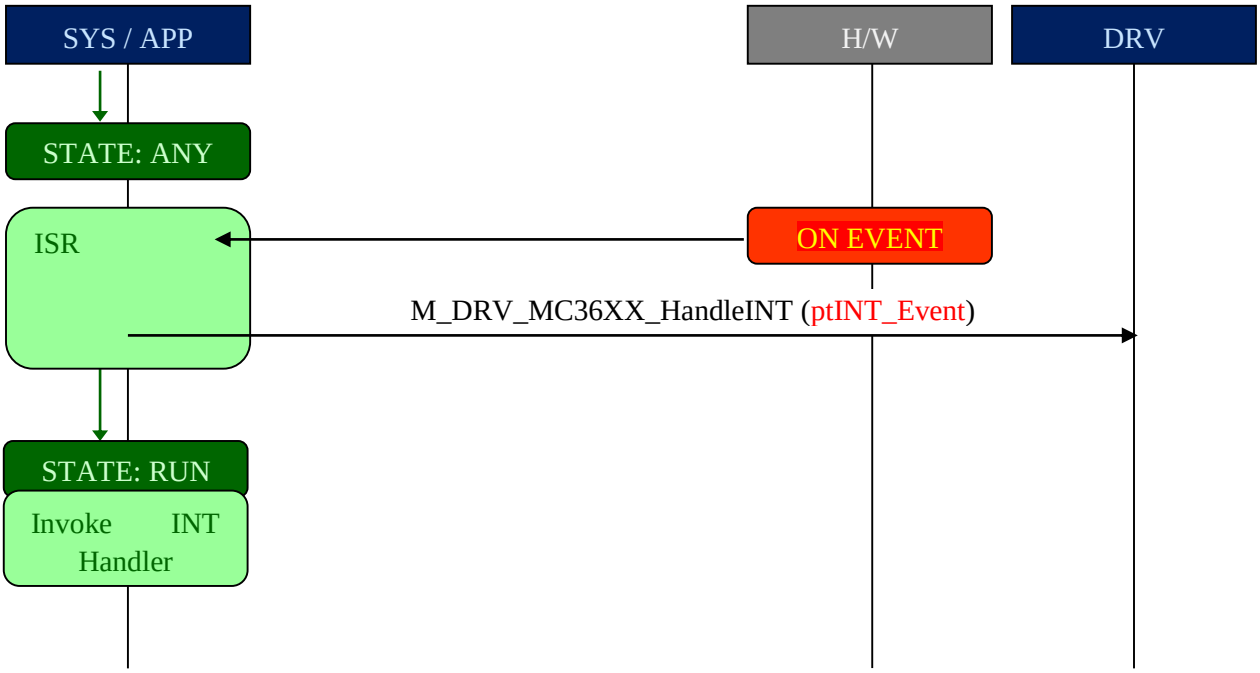
4.7. Enable Shake Detection (Configuration)



4.8. Read Data (Direct Mode)



4.9. Interrupt Handler



5. Sample Code

This chapter demonstrates code for reference.

5.1. Read Data (Direct Mode)

```
if(M_DRV_MC34X6_ReadSensorData(faAccelData) < 0)
    M_PRINTF("Read sensor data error!");
else
{
    int             _baFtoI_d1[3]={0};          //Float to Int
    int             _baFtoI_d2[3]={0};          //Float to Int
    float           _baFtoI_f = 0;              //Float to Int

    //ftoi
    for(int i=0;i<3;i++)
    {
        _baFtoI_d1[i] = faAccelData[i];                 // Get the integer part (678).
        if (faAccelData[i] < 0)
            _baFtoI_f = _baFtoI_d1[i] - faAccelData[i]; // Get fractional part (0.01234567).
        else
            _baFtoI_f = faAccelData[i] - _baFtoI_d1[i]; // Get fractional part (0.01234567).
        _baFtoI_d2[i] = trunc(_baFtoI_f * 10000);       // Turn into integer (123).
    }

    M_PRINTF("[[ACC     %d.%d     %d.%d     %d.%d",_baFtoI_d1[0],_baFtoI_d2[0],_baFtoI_d1[1],_baFtoI_d2[1],_baFtoI_d1[2],_baFtoI_d2[2]);
    //Acc Raw Data
}
```

Figure 34: Sample code for read data.

5.2. Motion Detection

Before start to test with the sample code, must confirm and select the function defined in scen_mc34x6_example.h:

```
/*******************************************************************************
 *** CONSTANT / DEFINE
 ******************************************************************************/
#define TF_TEST         0
#define TILT35_TEST     0
#define ANYM_TEST       1
#define SHAKE_TEST      0
#define LPF_TEST        0
```

Figure 35: Sample code for tilt/flip mode initial.

1. Initial Tilt/Flip mode

```
int _M_DRV_Tilt_Flip_Init(void)
{
    //STANDBY
    M_DRV_MC34X6_SetMode(E_M_DRV_MC34X6_MODE_STANDBY);

    //set TF Threshold
    M_DRV_MC34X6_SetTFThreshold();

    //set TF Debounce
    M_DRV_MC34X6_SetTFDebounce();

    //set TF Int
    M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_TILT_EN | E_M_DRV_MC34X6_INT_FLIP_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN);    //tilt+flip+auto
    //M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_TILT_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN);   //tilt+auto
    //M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_FLIP_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN);   //flip+auto

    //set Motion Control
    M_DRV_MC34X6_ConfigMotionCtl(E_M_DRV_MC34X6_MCTL_TF_EN | E_M_DRV_MC34X6_MCTL_ZORT_POS);

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 36: Sample code for tilt/flip mode initial.

2. Initial Tilt35 mode

```
int _M_DRV_Tilt35_Init(void)
{
    //STANDBY
    M_DRV_MC34X6_SetMode(E_M_DRV_MC34X6_MODE_STANDBY);

    //set Tilt35 Threshold
    M_DRV_MC34X6_SetTILT35Threshold();

    //set Tilt35 Timer
    M_DRV_MC34X6_SetTILT35Timer();

    //set TF Int
    M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_TILT35_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN);   //tilt35+auto

    //set Motion Control (need enable ANYM for tilt35)
    M_DRV_MC34X6_ConfigMotionCtl(E_M_DRV_MC34X6_MCTL_ANYM_EN | E_M_DRV_MC34X6_MCTL_TILT35_EN);

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 37: Sample code for tilt35 mode initial.

3. Initial ANYM mode

```
int _M_DRV_ANYM_Init(void)
{
    //STANDBY
    M_DRV_MC34X6_SetMode(E_M_DRV_MC34X6_MODE_STANDBY);

    //set ANYM Threshold
    M_DRV_MC34X6_SetANYMThreshold();

    //set ANYM Debounce
    M_DRV_MC34X6_SetANYMDebounce();

    //set ANYM Int
    M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_ANYM_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN); //ANYM+auto

    //set Motion Control
    M_DRV_MC34X6_ConfigMotionCtl(E_M_DRV_MC34X6_MCTL_ANYM_EN);

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 38: Sample code for any motion mode initial.

4. Initial Shake mode

```
int _M_DRV_Shake_Init(void)
{
    //STANDBY
    M_DRV_MC34X6_SetMode(E_M_DRV_MC34X6_MODE_STANDBY);

    //set Shake Threshold
    M_DRV_MC34X6_SetShakeThreshold();

    //set Shake counter and peak to peak duration
    M_DRV_MC34X6_SetShake_P2P_DUR_THRESH();

    //set Shake Int
    M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_SHAKE_EN | E_M_DRV_MC34X6_INT_AUTOCLR_EN);    //shake+auto

    //set Motion Control
    M_DRV_MC34X6_ConfigMotionCtl(E_M_DRV_MC34X6_MCTL_ANYM_EN | E_M_DRV_MC34X6_MCTL_SHAKE_EN);  //shake+ANYM_ENABLE

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 39: Sample code for shake mode initial.

5. Get detection result

```
int M_DRV_Mode_Event(void)
{
    unsigned char _baBuf[2] = {0};

    M_DRV_MC34X6_GetMotion(&_baBuf[0]);

    switch(_baBuf[0])
    {
        case 5:
            M_PRINTF("-------------------- TILT35 --------------------");
            break;
        case 4:
            M_PRINTF("-------------------- SHAKE --------------------");
            break;
        case 3:
            M_PRINTF("-------------------- ANYM --------------------");
            break;
        case 2:
            M_PRINTF("-------------------- FLIP --------------------");
            break;
        case 1:
            M_PRINTF("-------------------- TILT --------------------");
            break;
    }

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 40: Sample code for motion detection.

5.3. LPF Mode

```
int _M_DRV_LPF_Init(void)
{

    //STANDBY
    M_DRV_MC34X6_SetMode(E_M_DRV_MC34X6_MODE_STANDBY);

    //Set ODR to 2044Hz
    M_DRV_MC34X6_SetSampleRate(E_M_DRV_MC34X6_SR_DIVIDE_BY2 | E_M_DRV_MC34X6_SR_CLK_2P56M | E_M_DRV_MC34X6_SR_OSR_64);

    //set Int
    M_DRV_MC34X6_ConfigINT(E_M_DRV_MC34X6_INT_DIS);             //Disable

    //set Motion Control
    M_DRV_MC34X6_ConfigMotionCtl(E_M_DRV_MC34X6_MCTL_DIS);      //Disable

    //set LPF BW, according to spec, LPF BW is closed to 128Hz
    M_DRV_MC34X6_ConfigRegResRngLPFCtrl(E_M_DRV_MC34X6_RESOLUTION_16BIT, E_M_DRV_MC34X6_RANGE_2G, \
                                        E_M_DRV_MC34X6_LPF_1 | E_M_DRV_MC34X6_LPF_ENABLE);

    return (M_DRV_MC34X6_RETCODE_SUCCESS);
}
```

Figure 41: Sample code for LPF mode enable.