FKS12 MEMS Integration Developer Tests

Daniel Fekete [daniel.fekete@helbling.de](mailto:daniel.fekete@helbling.de)

Created: Mon, 17 Jul, 2017  
Modified: Mon, 17 Jul, 2017

This document describes the methods used to test the code written for the FKS12 MEMS integration. Due to the lack of debugging and ability to read the EEPROM, some of these tests may resemble module or even system tests. However, steps were taken to make these tests as focused as possible given the situation in order to test the functionality of the smallest possible units of code.

# sspi.c - Software SPI driver

## Basic IO Testing

* Pin Initialization
* Pin Output (setting/clearing)
* Pin Input

The previous code was reviewed and basic tests were made to confirm the methods for initializing pins as inputs/outputs as well setting and clearing pins. The CS line (PB3) was tested as an output. The unused pin PB7 was tested as an input. The tests involved simple loops in the main loop that toggled the output pin as well as blocking loops that waited for a change in state of the input.

## Software SPI Testing

* Single byte transfer (transmition only)
* Multi-byte transfer (transmition only)

After the basic IO was confirmed to work, the Software SPI code was tested. This was done by attaching a Logic Analyzer to the MEMS SPI lines, sending bytes out on the SPI bus and checking that the Logic Analyzer decoded the correct values.

### Single byte transfer

Here is the test code used to verify a single byte transfer added to *Uid.c, line 1719*:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* P L A Y G R O U N D : S O F T W A R E S P I  
\*  
\* Tests the sspi driver. To repeat the test, connect a logic analyser or  
\* oscilloscope to the key on the MEMS spi lines. Load this firmware and the key  
\* should write 0x55 to the SPI bus on startup.  
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include "../60\_MEMS/sspi.h"  
  
sspi\_init(); /\* initialize the mems interface \*/  
RegClrBit(RegOutPB, BIT4); /\* Set NCS Low \*/  
RegSetBit(RegIOSelPB, BIT4); /\* Set NCS to output \*/  
  
RegSetBit(RegOutPB, BIT4); /\* start spi transaction \*/  
Nop; Nop; Nop; Nop; Nop; Nop; Nop; Nop; /\* delay for NCS to switch \*/  
sspi\_xfer\_byte(0x55); /\* send single byte \*/  
RegClrBit(RegOutPB, BIT4); /\* end spi transaction \*/  
  
RegSetBit(RegIOSelPB, BIT4); /\* Set NCS to input \*/  
sspi\_deinit(); /\* de-initialize the mems interface \*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* P L A Y G R O U N D : T H E E N D  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### Multi-byte transfer

Once basic functionality was confirmed, the following code was written to test a multi-byte transfer in the same location:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* P L A Y G R O U N D : S O F T W A R E S P I  
\*  
\* Tests the sspi driver. To repeat the test, connect a logic analyser or  
\* oscilloscope to the key on the MEMS spi lines. Load this firmware and the key  
\* should write the sequence 0x00-0xFF to the SPI bus on startup.  
\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include "../60\_MEMS/sspi.h"  
  
sspi\_init(); /\* initialize the mems interface \*/  
RegClrBit(RegOutPB, BIT4); /\* Set NCS Low \*/  
RegSetBit(RegIOSelPB, BIT4); /\* Set NCS to output \*/  
  
RegSetBit(RegOutPB, BIT4); /\* start spi transaction \*/  
Nop; Nop; Nop; Nop; Nop; Nop; Nop; Nop; /\* delay for NCS to switch \*/  
u8 x = 0;  
do {  
 sspi\_xfer\_byte(x); /\* send single byte \*/  
 x++;  
} while (x);  
RegClrBit(RegOutPB, BIT4); /\* end spi transaction \*/  
  
RegSetBit(RegIOSelPB, BIT4); /\* Set NCS to input \*/  
sspi\_deinit(); /\* de-initialize the mems interface \*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\* P L A Y G R O U N D : T H E E N D  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

After the multi-byte transfer was confirmed, transmission of data using the software SPI driver was considered tested and verified.

Verification and testing of data reception using the software SPI driver was postponed until the MEMS driver was written.

# mems.c - MEMS Driver

The MEMS driver is an extension of the software SPI driver, so only the extended functionality was tested.

## mems\_read\_reg() Testing

* Software SPI transfer (Tx/Rx)
* Reading register of MEMS device

The MEMS register reading function was tested and confirmed with the following snippet added to *Uid.c, line 1718*:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : M E M S  
 \*  
 \* Reads the WHO\_AM\_I register in mems. When firmware is run, there should be a  
 \* 3 byte transaction bewtween the mctx and mems. Attach a logic analyzer to the  
 \* mems spi bus to verify.  
 \* - In the first byte the mctx sends 0x93 (read register 0x13).  
 \* - The second byte is a don't care byte, but should be 0xFF from mctx.  
 \* - The thrid byte should be the response from the mems: 0x62 (according to  
 \* the datasheet)  
 \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include "../60\_MEMS/mems.h"  
  
mems\_init();  
mems\_read\_reg(MEMS\_REG\_WHO\_AM\_I); /\* should return 0x62 \*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : E N D  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

A logic analyzer connected to the MEMS SPI bus confirmed the correct register access protocol and the received byte was as expected.

## mems\_write\_register() Testing

* Writing register of MEMS device

The MEMS register writing function was tested and confirmed with the following snippet added to *Uid.c, line 1718*:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : M E M S  
 \*  
 \* Tests mems\_write\_reg by writing a value to the SDCD\_LTHS\_LSB register  
 \* and reading it back. Attach a logic analyzer to the mems spi bus to verify  
 \* results.  
 \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include "../60\_MEMS/mems.h"  
  
mems\_init();  
mems\_write\_reg(MEMS\_REG\_SDCD\_LTHS\_LSB, 0x55);  
mems\_read\_reg(MEMS\_REG\_SDCD\_LTHS\_LSB); /\* should return 0x55 \*/  
mems\_write\_reg(MEMS\_REG\_SDCD\_LTHS\_LSB, 0x00);  
mems\_read\_reg(MEMS\_REG\_SDCD\_LTHS\_LSB); /\* should return 0x00 \*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : E N D  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

A logic analyzer connected to the MEMS SPI bus confirmed the correct register access protocol and the written/returned bytes were as expected.

## mems\_irq\_handler() and mems\_config\_motion\_detect() Testing

* Configuration of the MEMS device for motion detection
* Proper handling of MEMS IRQ events

The MEMS configuration and IRQ handling functions were tested and confirmed with the following snippet added to *Uid.c, line 1718*:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : M E M S  
 \*  
 \* Tests mems\_config\_motion\_detect() and mems\_irq\_handler(). Attach a logic  
 \* analyzer to the mems spi bus to verify results. See function code for debug  
 \* messages emitted. LED should flash on motion detection.  
 \*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
#include "../60\_MEMS/mems.h"  
  
mems\_init();  
  
if (MEMS\_IRQ\_IS\_ACTIVE) {  
 /\* mems irq is high, we need to see what happened \*/  
 mems\_irq\_handler();  
} else if (mems\_read\_reg(MEMS\_REG\_SYS\_MODE) == 0) {  
 /\* mems is just booted and/or unconfigured \*/  
 mems\_config\_motion\_detect();  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* P L A Y G R O U N D : E N D  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

A logic analyzer connected to the MEMS SPI bus confirmed the correct register access protocols and the written/returned bytes were as expected.