AVRCubeRev2

Generated by Doxygen 1.9.5

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

File Documentation

2.1 /Users/FabianFranz/Development/Projects/AVRCubeRev2/AvrCube↩ V2Code/src/main.cpp File Reference

Implementation of functionality for the "Cube Project".

```
#include "main.hpp"
```

Functions

- void setLedPins (uint8_t mode)

  Set the LED Pins eigther "INPUT" or "OUTPUT".
- void setSDA (bool mode)

  Set the SDA pin eighter "ON" or "OFF".
- void setSCL (bool mode)

  Set the SCL pin eighter "ON" or "OFF".
- void **init** ()

  Initialises all the AVR hardware E.g. ports, interrupts, timers, etc.
- void **allLedOn** ()

  Turn on all LEDs (LED1 - LED7)
- void **allLedOff** ()

  Turn off all LEDs (LED1 - LED7)
- void deInit ()

  Deeinitialises all the AVR hardware E.g. ports, interrupts, timers, etc.
- void showNumber (uint8_t numberToShow)

  Show a number between 1 and 6 on the LEDs.
- void showCross ()

  Show a cross on the LEDs.
- void showHook ()

  Show a hook on the LEDs.
- void testLeds ()

  Check if all LEDs are working correctly.
- void start ()

I2C BitBang start condition.

- void stop ()

  I2C BitBang stop condition.

- bool tx (uint8_t dat)

  I2C BitBang tx of one byte.

- uint8_t rx (bool ack)

  I2C BitBang rx of one byte.

- uint8_t readRegister (uint8_t reg)

  Read a register from the MMA8653FC.

- void writeRegister (uint8_t reg, uint8_t value)

  Write a value to a register of the MMA8653FC.

- void testI2C_read ()

  Checks, if the I2C connection to the MMA8653FC is working.

- void **MMA8653FC_init** ()

  Initialise the MMA8653FC range ans mode.

- void **MMA8653FC_deInit** ()

  Deinitialise the MMA8653FC to safe energy.

- void getAcceleration (int16_t *x, int16_t *y, int16_t *z)

  Read the acceleration data from the MMA8653FC.

- void dynamicDelay (uint16_t ms)

  Implemetation of a variable delay.

- float getAngle (float axis, float reference)

  Get the Angle between two acceleration values.

- void getRollNick (float *roll, float *nick)

  Get the roll and ncik angle from the x, y, z acceleration.

- int16_t getGradient (int16_t baseValue)

  Get the Gradient from the current values and the last value.

- bool motionDetected (uint8_t threshold)

  Return true or false, depending on the threshold.

- void spritLevel ()

  Shows the sprit level on the LEDs.

- void dice ()

  Realises a random number from 1 - 6 and show it on the LEDs.

- void calibrate ()

  Short calibration of the MMA8653FC values.

- void goToSleep ()

  Set the sleep mode, disable peripherals and go to sleep.

- int **main** ()
- **cli** ()
- **while** (!(PINB &(1<< BUTTON)))
- **if** (force_sleep_counter >=FORCE_SLEEP_TIME)
- **sei** ()

Variables

- volatile uint16_t **counter** = 0
- volatile uint8_t **button_pressed** = 0
- int16_t **x_offset**
- int16_t **y_offset**
- int16_t **z_offset**
- else

2.1.1 Detailed Description

Implementation of functionality for the "Cube Project".

Author

Fabian Franz fabian.franz0596@gmail.com

Version

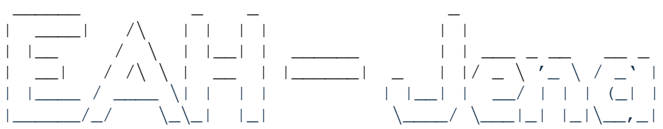
09.22

Date

2022-09-14

Copyright

Copyright (c) 2022



2.1.2 Function Documentation

2.1.2.1 calibrate()

```
void calibrate ( )
```

Short calibration of the MMA8653FC values.

Remarks

The calibration will first check if the device can be reached via the I2C connection. If not, it will show a cross on the LEDs and go to sleep. After that, it will read the current acceleration values and get the offset. The offset will be stored in the non persistant memory of the device. It is used to compensate the offset for angle calculation and motion detection.

2.1.2.2 deInit()

```
void deInit ( )
```

Deeinitialises all the AVR hardware E.g. ports, interrupts, timers, etc.

Remarks

Calling deInit(); is necessary to save power before the CPU goes to sleep.

2.1.2.3 dice()

```
void dice ( )
```

Realises a random number from 1 - 6 and show it on the LEDs.

Remarks

It will first generate and show some randome numbers with a long delay between them. Then the delay gets shorter. Finally, it will peristently show a random number on the LEDs. The function can only be stopped by external interrupt.

2.1.2.4 dynamicDelay()

```
void dynamicDelay (
            uint16_t ms )
```

Implemetation of a variable delay.

Parameters

ms	The time you want the core to not execute any code.

2.1.2.5 getAcceleration()

```
void getAcceleration (
            int16_t * x,
            int16_t * y,
            int16_t * z )
```

Read the acceleration data from the MMA8653FC.

Parameters

x	The x-axis acceleration data (call by reference).
y	The y-axis acceleration data (call by reference).
z	The z-axis acceleration data (call by reference).

```
// usage example
int16_t x, y, z;
MMA8653FC_read(&x, &y, &z);
```

2.1.2.6 getAngle()

```
float getAngle (
```

```
            float axis,
            float reference )
```

Get the Angle between two acceleration values.

Parameters

axis	The orthogonal axis of device rotation.
reference	The reference acceleration value of the other axis.

Returns

float Resulting angle in degree.

Remarks

The function is used to calculate the angle between a reference acceleration value and the acceleration value of the axis where the amount of acceleration is changing. A best practise to calculate the angle would be:

1. Read all the acceleration values from the sensor (x, y, z).
2. Calculate the amount of motion on those axis where the acceleration doesn't change significantly. E.g.:
   `amount = sqrt(y * y + z * z);`
3. Calculate the angle between the amount of motion and the acceleration value of the axis where the amount of motion is changing. E.g.:
   `angle = getAngle(x, amount);`
4. Use the angle to calculate the rotation of the device. For further information read: https://www.nxp.com/docs/en/application-note/AN3461.pdf

2.1.2.7 getGradient()

```
int16_t getGradient (
            int16_t baseValue )
```

Get the Gradient from the current values and the last value.

Parameters

baseValue	The currently measured value.

Returns

int16_t The gradient of the current value and the last value.

2.1.2.8 getRollNick()

```
void getRollNick (
            float * roll,
            float * nick )
```

Get the roll and ncik angle from the x, y, z acceleration.

Parameters

roll	The roll angle the get (call by reference).
nick	The nick angle the get (call by reference).

2.1.2.9 goToSleep()

```
void goToSleep ( )
```

Set the sleep mode, disable peripherals and go to sleep.

2.1.2.10 motionDetected()

```
bool motionDetected (
            uint8_t threshold )
```

Return true or false, depending on the threshold.

Parameters

threshold	The amount of angle which has to change in one to get the function to return true.

Returns

True if the threshold is reached, false if not.

2.1.2.11 readRegister()

```
uint8_t readRegister (
            uint8_t reg )
```

Read a register from the MMA8653FC.

Parameters

reg	The register to read.

Returns

The value of the register.

2.1.2.12 rx()

```
uint8_t rx (
            bool ack )
```

I2C BitBang rx of one byte.

Parameters

ack	ACK or NACK (acknowledge or no acknowledge)

Returns

The received byte.

2.1.2.13 setLedPins()

```
void setLedPins (
            uint8_t mode )
```

Set the LED Pins eigther "INPUT" or "OUTPUT".

Parameters

mode	OUTPUT = 0 or INPUT = 1

2.1.2.14 setSCL()

```
void setSCL (
            bool mode )
```

Set the SCL pin eighter "ON" or "OFF".

Parameters

mode	"ON" (input pullup) or "OFF" (output low)

2.1.2.15 setSDA()

```
void setSDA (
            bool mode )
```

Set the SDA pin eighter "ON" or "OFF".

Parameters

mode	"ON" (input pullup) or "OFF" (output low)

2.1.2.16 showCross()

```
void showCross ( )
```

Show a cross on the LEDs.

Remarks

This function is used to indicate some error.

2.1.2.17 showHook()

```
void showHook ( )
```

Show a hook on the LEDs.

Remarks

This function is used to indicate some success.

2.1.2.18 showNumber()

```
void showNumber (
            uint8_t numberToShow )
```

Show a number between 1 and 6 on the LEDs.

Parameters

numberToshow	The number to show on the LEDs.

Lremark If the number is larger than 6, no LED will be turned on.

2.1.2.19 spritLevel()

```
void spritLevel ( )
```

Shows the sprit level on the LEDs.

Remarks

The function internally calculates the roll and the nick angle of the device. It then uses the calculated angles to show in which direction the device is rotated. When it is at rest, only the LED in the middle will be turned on. When the device is rotated, the LEDs on the corresponding side will be turned on.

2.1.2.20 start()

```
void start ( )
```

I2C BitBang start condition.

2.1.2.21 stop()

```
void stop ( )
```

I2C BitBang stop condition.

2.1.2.22 testI2C_read()

```
void testI2C_read ( )
```

Checks, if the I2C connection to the MMA8653FC is working.

Remarks

If the communication is not working, it will be shown on the LEDs as a cross. Furthermore, the device will go to sleep.

2.1.2.23 testLeds()

```
void testLeds ( )
```

Check if all LEDs are working correctly.

2.1.2.24 tx()

```
bool tx (
            uint8_t dat )
```

I2C BitBang tx of one byte.

Parameters

dat	The byte to send.

Returns

ACK or NACK (acknowledge or no acknowledge)

2.1.2.25 writeRegister()

```
void writeRegister (
            uint8_t reg,
            uint8_t value )
```

Write a value to a register of the MMA8653FC.

Parameters

reg	The register to write.
value	The value to write.

2.1.3 Variable Documentation

2.1.3.1 else

```
else
```

Initial value:

```
{
    _delay_ms(10)
```

2.2 /Users/FabianFranz/Development/Projects/AVRCubeRev2/AvrCube↩ V2Code/src/main.hpp File Reference

Includes, defines and function prototypes for the "Cube Project".

```
#include <math.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include <avr/interrupt.h>
```

Macros

- #define **F_CPU** 8000000
- #define **__DELAY_BACKWARD_COMPATIBLE__**
- #define **BUTTON** PB2
- #define **D1** PA0
- #define **D2** PA1
- #define **D3** PA2
- #define **D4** PA3
- #define **D5** PA5
- #define **D6** PA7
- #define **D7** PB1
- #define **SCK** PA4
- #define **MISO** PA5
- #define **MOSI** PA6
- #define **SCL** PA4
- #define **SDA** PA6
- #define **UNCONNECTED** PB0
- #define **OUTPUT** 0
- #define **INPUT** 1
- #define **ENABLE** true
- #define **DISABLE** false
- #define **ON** true
- #define **OFF** false
- #define **SLEEP_THRESHOLD** 60000
- #define **ANGLE_GRADIENT_THRESHOLD** 25
- #define **DICE_STEPS_FIRST_ROUND** 5
- #define **DICE_STEPS_SECOND_ROUND** 5
- #define **DICE_TIME_STEPS** 100
- #define **DICE_TIME_STEPS_INCREASE** 10
- #define **ANGLETHRESHOLD** 1
- #define **CALIBRATION_STEP_DELAY** 500
- #define **OFFSET_CALIBRATION_STEPS** 10
- #define **OFFSET_CALIBRATION_STEP_DELAY** 10
- #define **FORCE_SLEEP_STEPTIME** 500
- #define **FORCE_SLEEP_TIME** 6 ∗ FORCE_SLEEP_STEPTIME
- #define **MOTION_MEASURING_REPETITIONS** 5
- #define **I2C_DELAY** 10
- #define **SDA_ON** (PORTA |= (1 << SDA))
- #define **SDA_OFF** (PORTA &= ∼(1 << SDA))
- #define **SCL_READ** (PINA & (1 << SCL))
- #define **SDA_READ** (PINA & (1 << SDA))
- #define **MMA8653FC_ADD** 0x1D
- #define **MMA8653FC_ADDR_READ** 0x3B
- #define **MMA8653FC_ADDR_WRITE** 0x3A
- #define **MMA8653FC_WHO_AM_I** 0x0D
- #define **MMA8653FC_XYZ_DATA_CFG** 0x0E
- #define **MMA8653FC_CTRL_REG1** 0x2A
- #define **MMA8653FC_SYSMOD** 0x0B
- #define **MMA8653FC_OUT_X_MSB** 0x01
- #define **MMA8653FC_OUT_X_LSB** 0x02
- #define **MMA8653FC_OUT_Y_MSB** 0x03
- #define **MMA8653FC_OUT_Y_LSB** 0x04
- #define **MMA8653FC_OUT_Z_MSB** 0x05
- #define **MMA8653FC_OUT_Z_LSB** 0x06

Functions

- void setLedPins (uint8_t mode)

  Set the LED Pins eigther "INPUT" or "OUTPUT".
- void setSDA (bool mode)

  Set the SDA pin eighter "ON" or "OFF".
- void setSCL (bool mode)

  Set the SCL pin eighter "ON" or "OFF".
- void **init** ()

  Initialises all the AVR hardware E.g. ports, interrupts, timers, etc.
- void **allLedOn** ()

  Turn on all LEDs (LED1 - LED7)
- void **allLedOff** ()

  Turn off all LEDs (LED1 - LED7)
- void deInit ()

  Deeinitialises all the AVR hardware E.g. ports, interrupts, timers, etc.
- void showNumber (uint8_t numberToShow)

  Show a number between 1 and 6 on the LEDs.
- void showCross ()

  Show a cross on the LEDs.
- void showHook ()

  Show a hook on the LEDs.
- void testLeds ()

  Check if all LEDs are working correctly.
- void start ()

  I2C BitBang start condition.
- void stop ()

  I2C BitBang stop condition.
- bool tx (uint8_t dat)

  I2C BitBang tx of one byte.
- uint8_t rx (bool ack)

  I2C BitBang rx of one byte.
- uint8_t readRegister (uint8_t reg)

  Read a register from the MMA8653FC.
- void writeRegister (uint8_t reg, uint8_t value)

  Write a value to a register of the MMA8653FC.
- void testI2C_read ()

  Checks, if the I2C connection to the MMA8653FC is working.
- void **MMA8653FC_init** ()

  Initialise the MMA8653FC range ans mode.
- void **MMA8653FC_deInit** ()

  Deinitialise the MMA8653FC to safe energy.
- void getAcceleration (int16_t *x, int16_t *y, int16_t *z)

  Read the acceleration data from the MMA8653FC.
- void dynamicDelay (uint16_t ms)

  Implemetation of a variable delay.
- float getAngle (float axis, float reference)

  Get the Angle between two acceleration values.
- void getRollNick (float *roll, float *nick)

  Get the roll and ncik angle from the x, y, z acceleration.
- int16_t getGradient (int16_t baseValue)

Get the Gradient from the current values and the last value.

- bool motionDetected (uint8_t threshold)

  Return true or false, depending on the threshold.
- void spritLevel ()

  Shows the sprit level on the LEDs.
- void dice ()

  Realises a random number from 1 - 6 and show it on the LEDs.
- void calibrate ()

  Short calibration of the MMA8653FC values.
- void goToSleep ()

  Set the sleep mode, disable peripherals and go to sleep.

2.2.1 Detailed Description

Includes, defines and function prototypes for the "Cube Project".

Author

Fabian Franz fabian.franz0596@gmail.com

Version

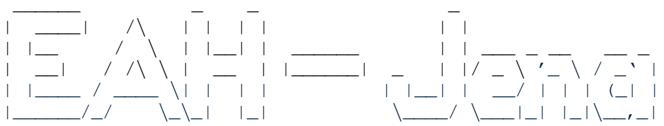
09.22

Date

2022-09-14

Copyright

Copyright (c) 2022



2.2.2 Function Documentation

2.2.2.1 calibrate()

```
void calibrate ( )
```

Short calibration of the MMA8653FC values.

Remarks

The calibration will first check if the device can be reached via the I2C connection. If not, it will show a cross on the LEDs and go to sleep. After that, it will read the current acceleration values and get the offset. The offset will be stored in the non persistant memory of the device. It is used to compensate the offset for angle calculation and motion detection.

2.2.2.2 deInit()

```
void deInit ( )
```

Deeinitialises all the AVR hardware E.g. ports, interrupts, timers, etc.

Remarks

Calling deInit(); is necessary to save power before the CPU goes to sleep.

2.2.2.3 dice()

```
void dice ( )
```

Realises a random number from 1 - 6 and show it on the LEDs.

Remarks

It will first generate and show some randome numbers with a long delay between them. Then the delay gets shorter. Finally, it will peristently show a random number on the LEDs. The function can only be stopped by external interrupt.

2.2.2.4 dynamicDelay()

```
void dynamicDelay (
            uint16_t ms )
```

Implemetation of a variable delay.

Parameters

ms	The time you want the core to not execute any code.

2.2.2.5 getAcceleration()

```
void getAcceleration (
            int16_t * x,
            int16_t * y,
            int16_t * z )
```

Read the acceleration data from the MMA8653FC.

Parameters

x	The x-axis acceleration data (call by reference).
y	The y-axis acceleration data (call by reference).
z	The z-axis acceleration data (call by reference).

```
// usage example
int16_t x, y, z;
MMA8653FC_read(&x, &y, &z);
```

2.2.2.6 getAngle()

```
float getAngle (
            float axis,
            float reference )
```

Get the Angle between two acceleration values.

Parameters

axis	The orthogonal axis of device rotation.
reference	The reference acceleration value of the other axis.

Returns

float Resulting angle in degree.

Remarks

The function is used to calculate the angle between a reference acceleration value and the acceleration value of the axis where the amount of acceleration is changing. A best practise to calculate the angle would be:

1. Read all the acceleration values from the sensor (x, y, z).
2. Calculate the amount of motion on those axis where the acceleration doesn't change significantly. E.g.:
   ```
   amount = sqrt(y * y + z * z);
   ```
3. Calculate the angle between the amount of motion and the acceleration value of the axis where the amount of motion is changing. E.g.:
   ```
   angle = getAngle(x, amount);
   ```
4. Use the angle to calculate the rotation of the device. For further information read: https://www.nxp.com/docs/en/application-note/AN3461.pdf

2.2.2.7 getGradient()

```
int16_t getGradient (
            int16_t baseValue )
```

Get the Gradient from the current values and the last value.

Parameters

baseValue	The currently measured value.

Returns

int16_t The gradient of the current value and the last value.

2.2.2.8 getRollNick()

```
void getRollNick (
            float * roll,
            float * nick )
```

Get the roll and ncik angle from the x, y, z acceleration.

Parameters

roll	The roll angle the get (call by reference).
nick	The nick angle the get (call by reference).

2.2.2.9 goToSleep()

```
void goToSleep ( )
```

Set the sleep mode, disable peripherals and go to sleep.

2.2.2.10 motionDetected()

```
bool motionDetected (
            uint8_t threshold )
```

Return true or false, depending on the threshold.

Parameters

threshold	The amount of angle which has to change in one to get the function to return true.

Returns

True if the threshold is reached, false if not.

2.2.2.11 readRegister()

```
uint8_t readRegister (
            uint8_t reg )
```

Read a register from the MMA8653FC.

Parameters

reg	The register to read.

Returns

The value of the register.

2.2.2.12 rx()

```
uint8_t rx (
            bool ack )
```

I2C BitBang rx of one byte.

Parameters

ack	ACK or NACK (acknowledge or no acknowledge)

Returns

The received byte.

2.2.2.13 setLedPins()

```
void setLedPins (
            uint8_t mode )
```

Set the LED Pins eigther "INPUT" or "OUTPUT".

Parameters

mode	OUTPUT = 0 or INPUT = 1

2.2.2.14 setSCL()

```
void setSCL (
            bool mode )
```

Set the SCL pin eighter "ON" or "OFF".

Parameters

mode	"ON" (input pullup) or "OFF" (output low)

2.2.2.15 setSDA()

```
void setSDA (
            bool mode )
```

Set the SDA pin eighter "ON" or "OFF".

Parameters

mode	"ON" (input pullup) or "OFF" (output low)

2.2.2.16 showCross()

```
void showCross ( )
```

Show a cross on the LEDs.

Remarks

This function is used to indicate some error.

2.2.2.17 showHook()

```
void showHook ( )
```

Show a hook on the LEDs.

Remarks

This function is used to indicate some success.

2.2.2.18 showNumber()

```
void showNumber (
            uint8_t numberToShow )
```

Show a number between 1 and 6 on the LEDs.

Parameters

numberToshow	The number to show on the LEDs.

Lremark If the number is larger than 6, no LED will be turned on.

2.2.2.19 spritLevel()

```
void spritLevel ( )
```

Shows the sprit level on the LEDs.

Remarks

The function internally calculates the roll and the nick angle of the device. It then uses the calculated angles to show in which direction the device is rotated. When it is at rest, only the LED in the middle will be turned on. When the device is rotated, the LEDs on the corresponding side will be turned on.

2.2.2.20 start()

```
void start ( )
```

I2C BitBang start condition.

2.2.2.21 stop()

```
void stop ( )
```

I2C BitBang stop condition.

2.2.2.22 testI2C_read()

```
void testI2C_read ( )
```

Checks, if the I2C connection to the MMA8653FC is working.

Remarks

If the communication is not working, it will be shown on the LEDs as a cross. Furthermore, the device will go to sleep.

2.2.2.23 testLeds()

```
void testLeds ( )
```

Check if all LEDs are working correctly.

2.2.2.24 tx()

```
bool tx (
            uint8_t dat )
```

I2C BitBang tx of one byte.

Parameters

dat	The byte to send.

Returns

ACK or NACK (acknowledge or no acknowledge)

2.2.2.25 writeRegister()

```
void writeRegister (
            uint8_t reg,
            uint8_t value )
```

Write a value to a register of the MMA8653FC.

Parameters

reg	The register to write.
value	The value to write.

2.3 main.hpp

Go to the documentation of this file.

```
1
21 #ifndef MAIN_HPP
22 #define MAIN_HPP
23
24 // ---------------------------------------------------------------- //
25 // -- Includes ------------------------------------------------- //
26 // ---------------------------------------------------------------- //
27 #include <math.h>
28 #include <stdlib.h>
```

```
29 #include <avr/io.h>
30 #include <avr/sleep.h>
31 #include <util/delay.h>
32 #include <avr/interrupt.h>
33
34 // ------------------------------------------------------------- //
35 // -- Defines ---------------------------------------------------- //
36 // ------------------------------------------------------------- //
37 #define F_CPU 8000000
38 #define __DELAY_BACKWARD_COMPATIBLE__
39 // Defines for the button pin
40 #define BUTTON PB2
41 // Defines for the connected LEDs
42 #define D1 PA0
43 #define D2 PA1
44 #define D3 PA2
45 #define D4 PA3
46 #define D5 PA5
47 #define D6 PA7
48 #define D7 PB1
49 // Defines for the SPI interface.
50 #define SCK PA4
51 #define MISO PA5
52 #define MOSI PA6
53 // Defines for the I2C interface (no hardware interface).
54 #define SCL PA4
55 #define SDA PA6
56 // Defines for unconnected pins
57 #define UNCONNECTED PB0
58
59 // General defines
60 #define OUTPUT 0
61 #define INPUT 1
62 #define ENABLE true
63 #define DISABLE false
64 #define ON true
65 #define OFF false
66
67 // Defines for control flow
68 #define SLEEP_THRESHOLD 60000                // Millieconds until the device go to sleep if no action
      occurs
69 #define ANGLE_GRADIENT_THRESHOLD 25  // The threshold for the absolute motion value in LSB registers
70 #define DICE_STEPS_FIRST_ROUND 5    // The number of steps the dice has to roll
71 #define DICE_STEPS_SECOND_ROUND 5   // The number of steps the dice has to roll
72 #define DICE_TIME_STEPS 100         // The time step between dice rolls in ms
73 #define DICE_TIME_STEPS_INCREASE 10 // The time step increase between dice rolls in ms in secon round
74 #define ANGLETHRESHOLD 1            // The threshold for the angle in degrees
75 #define CALIBRATION_STEP_DELAY 500  // The delay between calibration steps in ms
76 #define OFFSET_CALIBRATION_STEPS 10      // The number of calibration steps
77 #define OFFSET_CALIBRATION_STEP_DELAY 10 // The delay between offset calibration steps in ms
78 #define FORCE_SLEEP_STEPTIME 500         // The delay until new number
79 #define FORCE_SLEEP_TIME 6 * FORCE_SLEEP_STEPTIME // When button is hold this amout of milliseconds the
      device will go to sleep
80 #define MOTION_MEASURING_REPETITIONS 5 // The number of repetitions for the motion measuring
81
82 // Defines for I2C
83 #define I2C_DELAY 10 // 10 us -> 100 kHz
84 #define SDA_ON (PORTA |= (1 « SDA))
85 #define SDA_OFF (PORTA &= ~(1 « SDA))
86 #define SCL_READ (PINA & (1 « SCL))
87 #define SDA_READ (PINA & (1 « SDA))
88 // Defines for Acceleration Sensor
89 #define MMA8653FC_ADD 0x1D
90 #define MMA8653FC_ADDR_READ 0x3B
91 #define MMA8653FC_ADDR_WRITE 0x3A
92 // Defines for Acceleration Sensor Registers
93 #define MMA8653FC_WHO_AM_I 0x0D
94 #define MMA8653FC_XYZ_DATA_CFG 0x0E
95 #define MMA8653FC_CTRL_REG1 0x2A
96 #define MMA8653FC_SYSMOD 0x0B    // System Mode, to control STANDBY, WAKE and SLEEP
97 #define MMA8653FC_OUT_X_MSB 0x01 // Most significant byte of X-axis acceleration data
98 #define MMA8653FC_OUT_X_LSB 0x02 // Least significant byte of X-axis acceleration data
99 #define MMA8653FC_OUT_Y_MSB 0x03 // Most significant byte of Y-axis acceleration data
100 #define MMA8653FC_OUT_Y_LSB 0x04 // Least significant byte of Y-axis acceleration data
101 #define MMA8653FC_OUT_Z_MSB 0x05 // Most significant byte of Z-axis acceleration data
102 #define MMA8653FC_OUT_Z_LSB 0x06 // Least significant byte of Z-axis acceleration data
103
104 // ------------------------------------------------------------- //
105 // -- Function Prototypes ---------------------------------------- //
106 // ------------------------------------------------------------- //
112 void setLedPins(uint8_t mode);
113
119 void setSDA(bool mode);
120
126 void setSCL(bool mode);
127
131 void init();
```

```
132
137 void allLedOn();
138
142 void allLedOff();
143
149 void deInit();
150
158 void showNumber(uint8_t numberToShow);
164 void showCross();
165
171 void showHook();
172
177 void testLeds();
178
183 void start();
184
189 void stop();
190
198 bool tx(uint8_t dat);
199
207 uint8_t rx(bool ack);
208
216 uint8_t readRegister(uint8_t reg);
217
224 void writeRegister(uint8_t reg, uint8_t value);
225
233 void testI2C_read();
234
238 void MMA8653FC_init();
239
243 void MMA8653FC_deInit();
244
258 void getAcceleration(int16_t* x, int16_t* y, int16_t* z);
259
265 void dynamicDelay(uint16_t ms);
266
291 float getAngle(float axis, float reference);
292
299 void getRollNick(float* roll, float* nick);
300
307 int16_t getGradient(int16_t baseValue);
308
316 bool motionDetected(uint8_t threshold);
317
328 void spritLevel();
329
338 void dice();
339
352 void calibrate();
353
358 void goToSleep();
359
371 #endif /* MAIN_HPP */
```

Index