Thunderboard Sense 2 I2C Drivers

Generated by Doxygen 1.9.2

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 APP_LETIMER_PWM_TypeDef Struct Reference

Data Fields

- bool **debugRun**
- bool **enable**
- uint32_t **out_pin_route0**
- uint32_t **out_pin_route1**
- bool **out_pin_0_en**
- bool **out_pin_1_en**
- float **period**
- float **active_period**
- bool **comp0_irq_enable**
- uint32_t **comp0_cb**
- bool **comp1_irq_enable**
- uint32_t **comp1_cb**
- bool **uf_irq_enable**
- uint32_t **uf_cb**

The documentation for this struct was generated from the following file:

- src/Header Files/letimer.h

3.2 I2C_OPEN_STRUCT Struct Reference

Data Fields

- bool **enable**
- bool **master**
- uint32_t **refFreq**
- uint32_t **freq**
- I2C_ClockHLR_TypeDef **clhr**
- uint32_t **out_pin_route_scl**
- uint32_t **out_pin_route_sda**
- bool **out_pin_enable_scl**
- bool **out_pin_enable_sda**

The documentation for this struct was generated from the following file:

- src/Header Files/i2c.h

3.3 I2C_STATE_MACHINE Struct Reference

Data Fields

- enum DEFINED_STATES **state**
- I2C_TypeDef * **i2c**
- uint32_t **slave_addr**
- uint32_t **slave_reg**
- bool **read**
- uint32_t **bytes_expected**
- uint32_t **bytes_received**
- uint32_t * **rx_data**
- uint32_t **tx_data**
- uint32_t **callback**
- volatile bool **busy**

The documentation for this struct was generated from the following file:

- src/Source Files/i2c.c

Chapter 4

File Documentation

4.1 src/Header Files/app.h File Reference

Contains high-level functions that drive the application.

```
#include "em_cmu.h"
#include "em_assert.h"
#include "cmu.h"
#include "gpio.h"
#include "letimer.h"
#include "brd_config.h"
#include "LEDs_thunderboard.h"
#include "si1133.h"
```

Macros

- #define **PWM_PER** 1.0
- #define **PWM_ACT_PER** 0.002
- #define **LETIMER0_COMP0_CB** 0x00000001
- #define **LETIMER0_COMP1_CB** 0x00000002
- #define **LETIMER0_UF_CB** 0x00000004
- #define **SI1133_REG_READ_CB** 0x00000008
- #define **SI1133_PART_ID** 0x33

Functions

- void app_peripheral_setup (void)

  Function that calls several functions that open and initialize all necessary peripherals.
- void scheduled_letimer0_uf_cb (void)

  Event handler that responds to a LETIMER0_UF_CB event.
- void scheduled_letimer0_comp0_cb (void)

  Event handler that responds to a LETIMER0_COMP0_CB event.
- void scheduled_letimer0_comp1_cb (void)

  Event handler that invokes an I2C read in response to a LETIMER0_COMP1_CB event.
- void scheduled_si1133_reg_read_cb (void)

  Event handler that turns RGB LED red or green depending on result of I2C interaction in response to a SI1133_↩ REG_READ_CB event.

4.1.1 Detailed Description

Contains high-level functions that drive the application.

Author

Mason Milligan

Date

2021-10-24

4.1.2 Function Documentation

4.1.2.1 app_peripheral_setup()

```
void app_peripheral_setup (
            void  )
```

Function that calls several functions that open and initialize all necessary peripherals.

This routine calls the setup processes for the clock and gpio peripherals, gathers PWM period and routing settings, then starts the low energy timer.

Note

This function is typically run once. It calls all necessary functions to prepare peripherals and begin operation.

4.1.2.2 scheduled_letimer0_comp0_cb()

```
void scheduled_letimer0_comp0_cb (
            void  )
```

Event handler that responds to a LETIMER0_COMP0_CB event.

This function contains an assert that is set to always fail.

Note

LETIMER0_COMP0_CB events should not occur in this application. If one does occur, the assert statement may be used to trace the cause.

4.1.2.3 scheduled_letimer0_comp1_cb()

```
void scheduled_letimer0_comp1_cb (
            void  )
```

Event handler that invokes an I2C read in response to a LETIMER0_COMP1_CB event.

This function calls si1133_read to begin I2C communication with Si1133.

Note

Typically, this function should only be called when a LETIMER0_COMP1_CB event occurs.

4.1.2.4 scheduled_letimer0_uf_cb()

```
void scheduled_letimer0_uf_cb (
            void  )
```

Event handler that responds to a LETIMER0_UF_CB event.

This function contains an assert that is set to always fail.

Note

LETIMER0_UF_CB events should not occur in this application. If one does occur, the assert statement may be used to trace the cause.

4.1.2.5 scheduled_si1133_reg_read_cb()

```
void scheduled_si1133_reg_read_cb (
            void  )
```

Event handler that turns RGB LED red or green depending on result of I2C interaction in response to a SI1133_↩ REG_READ_CB event.

This function calls si1133_get_result to collect the data retrieved via I2C for comparison with the Si1133 part ID. If the values match, a green LED is enabled. If they do not match, a red LED is enabled.

Note

Typically, this function should only be called when a SI1133_REG_READ_CB event occurs.

4.2 app.h

Go to the documentation of this file.

```
1
9 //*****************************************************************************
10 // Include files
11 //*****************************************************************************
12 #ifndef APP_HG
13 #define APP_HG
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18 #include "em_cmu.h"
19 #include "em_assert.h"
20
21 /* The developer's include statements */
22 #include "cmu.h"
23 #include "gpio.h"
24 #include "letimer.h"
25 #include "brd_config.h"
26 #include "LEDs_thunderboard.h"
27 #include "si1133.h"
28
29 //*****************************************************************************
30 // defined files
31 //*****************************************************************************
32 #define   PWM_PER          1.0           // PWM period in seconds
33 #define   PWM_ACT_PER      0.002         // PWM active period in seconds
34
35 // Application scheduled events
36 #define LETIMER0_COMP0_CB   0x00000001  // 0b0001
37 #define LETIMER0_COMP1_CB   0x00000002  // 0b0010
38 #define LETIMER0_UF_CB      0x00000004  // 0b0100
39 #define SI1133_REG_READ_CB  0x00000008  // 0b1000
40
41 #define SI1133_PART_ID      0x33        // 51
42
43 //*****************************************************************************
44 // global variables
45 //*****************************************************************************
46
47 //*****************************************************************************
48 // function prototypes
49 //*****************************************************************************
50 void app_peripheral_setup(void);
51 void scheduled_letimer0_uf_cb(void);
52 void scheduled_letimer0_comp0_cb(void);
53 void scheduled_letimer0_comp1_cb(void);
54 void scheduled_si1133_reg_read_cb(void);
55
56 #endif
```

4.3 src/Header Files/brd_config.h File Reference

Contains Thunderboard Sense 2 board definitions.

```
#include "em_gpio.h"
#include "em_cmu.h"
```

Macros

- #define **LED_RED_PORT** gpioPortD
- #define **LED_RED_PIN** 8
- #define **LED_RED_DEFAULT** false
- #define **LED_RED_GPIOMODE** gpioModePushPull
- #define **LED_GREEN_PORT** gpioPortD
- #define **LED_GREEN_PIN** 9

- #define **LED_GREEN_DEFAULT** false
- #define **LED_GREEN_GPIOMODE** gpioModePushPull
- #define **MCU_HFXO_FREQ** cmuHFRCOFreq_26M0Hz
- #define **LED_RED_DRIVE_STRENGTH** gpioDriveStrengthWeakAlternateWeak
- #define **LED_GREEN_DRIVE_STRENGTH** gpioDriveStrengthWeakAlternateWeak
- #define **PWM_ROUTE_0** LETIMER_ROUTELOC0_OUT0LOC_LOC17
- #define **PWM_ROUTE_1** LETIMER_ROUTELOC0_OUT1LOC_LOC0
- #define **RGB_ENABLE_PORT** gpioPortJ
- #define **RGB_ENABLE_PIN** 14
- #define **RGB0_PORT** gpioPortI
- #define **RGB0_PIN** 0
- #define **RGB1_PORT** gpioPortI
- #define **RGB1_PIN** 1
- #define **RGB2_PORT** gpioPortI
- #define **RGB2_PIN** 2
- #define **RGB3_PORT** gpioPortI
- #define **RGB3_PIN** 3
- #define **RGB_RED_PORT** gpioPortD
- #define **RGB_RED_PIN** 11
- #define **RGB_GREEN_PORT** gpioPortD
- #define **RGB_GREEN_PIN** 12
- #define **RGB_BLUE_PORT** gpioPortD
- #define **RGB_BLUE_PIN** 13
- #define **RGB_DEFAULT_OFF** false
- #define **COLOR_DEFAULT_OFF** false
- #define **RED_RGB_LOC** TIMER_ROUTELOC0_CC0LOC_LOC19
- #define **GREEN_RGB_LOC** TIMER_ROUTELOC0_CC1LOC_LOC19
- #define **BLUE_RGB_LOC** TIMER_ROUTELOC0_CC2LOC_LOC19
- #define **SI1133_SENSOR_EN_PORT** gpioPortF
- #define **SI1133_SENSOR_EN_PIN** 9
- #define **SI1133_SENSOR_EN_DEFAULT** true
- #define **SI1133_SENSOR_EN_GPIOMODE** gpioModePushPull
- #define **SI1133_SENSOR_EN_DRIVE_STRENGTH** gpioDriveStrengthWeakAlternateWeak
- #define **SI1133_SCL_PORT** gpioPortC
- #define **SI1133_SCL_PIN** 5
- #define **SI1133_SCL_DEFAULT** true
- #define **SI1133_SCL_GPIOMODE** gpioModeWiredAnd
- #define **SI1133_SCL_LOC** I2C_ROUTELOC0_SCLLOC_LOC17
- #define **SI1133_SDA_PORT** gpioPortC
- #define **SI1133_SDA_PIN** 4
- #define **SI1133_SDA_DEFAULT** true
- #define **SI1133_SDA_GPIOMODE** gpioModeWiredAnd
- #define **SI1133_SDA_LOC** I2C_ROUTELOC0_SDALOC_LOC17

4.3.1 Detailed Description

Contains Thunderboard Sense 2 board definitions.

Author

Mason Milligan

Date

2021-10-09

4.4 brd_config.h

Go to the documentation of this file.

```
1
9 #ifndef BRD_CONFIG_HG
10 #define BRD_CONFIG_HG
11
12 //***********************************************************************************
13 // Include files
14 //***********************************************************************************
15 #include "em_gpio.h"
16 #include "em_cmu.h"
17
18 //***********************************************************************************
19 // defined files
20 //***********************************************************************************
21
22 /* LED 0 pin definitions */
23 #define LED_RED_PORT       gpioPortD
24 #define LED_RED_PIN        8
25 #define LED_RED_DEFAULT    false   // Default false (0) = off, true (1) = on
26 #define LED_RED_GPIOMODE   gpioModePushPull
27
28 // LED 1 pin definitions
29 #define LED_GREEN_PORT       gpioPortD
30 #define LED_GREEN_PIN        9
31 #define LED_GREEN_DEFAULT    false // Default false (0) = off, true (1) = on
32 #define LED_GREEN_GPIOMODE   gpioModePushPull
33
34 #define MCU_HFXO_FREQ           cmuHFRCOFreq_26M0Hz
35
36 /* LED drive strength definitions */
37 #ifdef STRONG_DRIVE
38     #define LED_RED_DRIVE_STRENGTH      gpioDriveStrengthStrongAlternateStrong
39     #define LED_GREEN_DRIVE_STRENGTH    gpioDriveStrengthStrongAlternateStrong
40 #else
41 #define LED_RED_DRIVE_STRENGTH      gpioDriveStrengthWeakAlternateWeak
42 #define LED_GREEN_DRIVE_STRENGTH    gpioDriveStrengthWeakAlternateWeak
43 #endif
44
45 /* LETIMER LED pin routes */
46 #define   PWM_ROUTE_0     LETIMER_ROUTELOC0_OUT0LOC_LOC17
47 #define   PWM_ROUTE_1     LETIMER_ROUTELOC0_OUT1LOC_LOC0
48
49 /* Thunderboard RGB LED pin definitions */
50 #define RGB_ENABLE_PORT   gpioPortJ
51 #define RGB_ENABLE_PIN    14
52 #define RGB0_PORT         gpioPortI
53 #define RGB0_PIN          0
54 #define RGB1_PORT         gpioPortI
55 #define RGB1_PIN          1
56 #define RGB2_PORT         gpioPortI
57 #define RGB2_PIN          2
58 #define RGB3_PORT         gpioPortI
59 #define RGB3_PIN          3
60 #define RGB_RED_PORT      gpioPortD
61 #define RGB_RED_PIN       11
62 #define RGB_GREEN_PORT    gpioPortD
63 #define RGB_GREEN_PIN     12
64 #define RGB_BLUE_PORT     gpioPortD
65 #define RGB_BLUE_PIN      13
66 #define RGB_DEFAULT_OFF   false
67 #define COLOR_DEFAULT_OFF false
68 #define RED_RGB_LOC       TIMER_ROUTELOC0_CC0LOC_LOC19
69 #define GREEN_RGB_LOC     TIMER_ROUTELOC0_CC1LOC_LOC19
70 #define BLUE_RGB_LOC      TIMER_ROUTELOC0_CC2LOC_LOC19
71
72 /* Si1133 enable pin definitions */
73 #define SI1133_SENSOR_EN_PORT           gpioPortF
74 #define SI1133_SENSOR_EN_PIN            9
75 #define SI1133_SENSOR_EN_DEFAULT        true    // true = on, false = off
76 #define SI1133_SENSOR_EN_GPIOMODE       gpioModePushPull
77 #define SI1133_SENSOR_EN_DRIVE_STRENGTH gpioDriveStrengthWeakAlternateWeak
78
79 /* Si1133 SCL pin definitions */
80 #define SI1133_SCL_PORT                 gpioPortC
81 #define SI1133_SCL_PIN                  5
82 #define SI1133_SCL_DEFAULT              true
83 #define SI1133_SCL_GPIOMODE             gpioModeWiredAnd
84 #define SI1133_SCL_LOC                  I2C_ROUTELOC0_SCLLOC_LOC17
85
86 /* Si1133 SDA pin definitions */
87 #define SI1133_SDA_PORT                 gpioPortC
88 #define SI1133_SDA_PIN                  4
89 #define SI1133_SDA_DEFAULT              true
```

```
90 #define SI1133_SDA_GPIOMODE             gpioModeWiredAnd
91 #define SI1133_SDA_LOC                  I2C_ROUTELOC0_SDALOC_LOC17
92
93 //*****************************************************************************
94 // function prototypes
95 //*****************************************************************************
96
97 #endif
```

4.5 src/Header Files/cmu.h File Reference

Contains functions that configure clocks and clock routing.

```
#include "em_cmu.h"
#include "em_assert.h"
#include "brd_config.h"
```

Functions

- void cmu_open (void)

  Function to disable unused clocks and route ULFRCO.

4.5.1 Detailed Description

Contains functions that configure clocks and clock routing.

Author

Mason Milligan

Date

2021-09-12

4.5.2 Function Documentation

4.5.2.1 cmu_open()

```
void cmu_open (
            void  )
```

Function to disable unused clocks and route ULFRCO.

Since LFRCO and LFXO are not needed, they are disabled here. Additionally, ULFRCO is routed through CMU_↩ LFACLKSEL.LFA and CMU_LFACLKEN0.LETIMER0.

Note

This function is typically called once to disable unused clocks and prepare ULFRCO to be used.

4.6 cmu.h

Go to the documentation of this file.

```
1
9 //***************************************************************************************
10 // Include files
11 //***************************************************************************************
12 #ifndef CMU_HG
13 #define CMU_HG
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18 #include "em_cmu.h"
19 #include "em_assert.h"
20
21 /* The developer's include statements */
22 #include "brd_config.h"
23
24 //***************************************************************************************
25 // defined files
26 //***************************************************************************************
27
28 //***************************************************************************************
29 // global variables
30 //***************************************************************************************
31
32 //***************************************************************************************
33 // function prototypes
34 //***************************************************************************************
35 void cmu_open(void);
36
37 #endif
```

4.7 src/Header Files/gpio.h File Reference

Contains functions that configure GPIO peripherals.

```
#include "em_cmu.h"
#include "em_gpio.h"
#include "em_assert.h"
#include "brd_config.h"
```

Functions

- void gpio_open (void)

  Function to configure LED pins for use.

4.7.1 Detailed Description

Contains functions that configure GPIO peripherals.

Author

Mason Milligan

Date

2021-09-25

4.7.2 Function Documentation

4.7.2.1 gpio_open()

```
void gpio_open (
            void  )
```

Function to configure LED pins for use.

This function enables the red and green LED pins and sets the appropriate drive strength.

Note

This function is typically run once to prepare the LEDs as an output.

4.8 gpio.h

Go to the documentation of this file.

```
1
9 //***************************************************************************
10 // Include files
11 //***************************************************************************
12 #ifndef GPIO_HG
13 #define GPIO_HG
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18 #include "em_cmu.h"
19 #include "em_gpio.h"
20 #include "em_assert.h"
21
22 /* The developer's include statements */
23 #include "brd_config.h"
24
25 //***************************************************************************
26 // defined files
27 //***************************************************************************
28
29 //***************************************************************************
30 // global variables
31 //***************************************************************************
32
33 //***************************************************************************
34 // function prototypes
35 //***************************************************************************
36 void gpio_open(void);
37
38 #endif
```

4.9 src/Header Files/HW_delay.h File Reference

Contains function that enables a time delay.

```
#include "em_timer.h"
#include "em_cmu.h"
```

Functions

- void timer_delay (uint32_t ms_delay)

  Function to stall a specified number of milliseconds.

4.9.1 Detailed Description

Contains function that enables a time delay.

Author

Keith Graham

Date

2020-04-19

4.9.2 Function Documentation

4.9.2.1 timer_delay()

```
void timer_delay (
            uint32_t ms_delay )
```

Function to stall a specified number of milliseconds.

This function enables stalling for a specified amount of time.

Parameters

in	*ms_delay*	Time to wait in milliseconds

4.10 HW_delay.h

Go to the documentation of this file.

```
1
9 #ifndef SRC_HW_DELAY_H_
10 #define SRC_HW_DELAY_H_
11
12 #include "em_timer.h"
13 #include "em_cmu.h"
14
15 void timer_delay(uint32_t ms_delay);
16
17 #endif /* SRC_HW_DELAY_H_ */
```

4.11 src/Header Files/i2c.h File Reference

Contains functions that enable I2C state machine functionality.

```
#include "em_cmu.h"
#include "em_i2c.h"
#include "scheduler.h"
#include "sleep_routines.h"
```

Data Structures

- struct I2C_OPEN_STRUCT

Macros

- #define **I2C_EM** EM2
- #define **I2C_READ_BIT** 1
- #define **I2C_WRITE_BIT** 0
- #define **BYTE** 8

Functions

- void i2c_open (I2C_TypeDef *i2c, I2C_OPEN_STRUCT *i2c_setup)

  Driver to initialize and configure an I2C peripheral.
- void i2c_start (I2C_TypeDef *i2c, uint32_t slave_addr, uint32_t slave_reg, bool read, uint32_t bytes_↩ expected, uint32_t *rx_data, uint32_t tx_data, uint32_t callback)

  Function that invokes I2C communication with an external device.
- bool i2c_busy (I2C_TypeDef *i2c)

  Function to report the busy status of the I2C state machine associated with the given I2C peripheral.
- void I2C0_IRQHandler (void)

  Interrupt handler to respond to I2C-related interrupts.
- void I2C1_IRQHandler (void)

  Interrupt handler to respond to I2C-related interrupts.

4.11.1 Detailed Description

Contains functions that enable I2C state machine functionality.

Author

Mason Milligan

Date

2021-10-24

4.11.2 Function Documentation

4.11.2.1 I2C0_IRQHandler()

```
void I2C0_IRQHandler (
            void  )
```

Interrupt handler to respond to I2C-related interrupts.

This function determines the nature of the I2C interrupt that occurred and calls the relevant function for the event that occurred.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.11.2.2 I2C1_IRQHandler()

```
void I2C1_IRQHandler (
            void  )
```

Interrupt handler to respond to I2C-related interrupts.

This function determines the nature of the I2C interrupt that occurred and calls the relevant function for the event that occurred.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.11.2.3 i2c_busy()

```
bool i2c_busy (
            I2C_TypeDef * i2c )
```

Function to report the busy status of the I2C state machine associated with the given I2C peripheral.

This function returns the busy status variable of the selected I2C state machine variable.

Note

This function may be called by higher-level routines to check the status of the private I2C state machine variable.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
out	*busy*	Boolean indicating the busy status of the selected I2C state machine. True indicates busy; false indicates not busy.

4.11.2.4 i2c_open()

```
void i2c_open (
            I2C_TypeDef * i2c,
            I2C_OPEN_STRUCT * i2c_setup )
```

Driver to initialize and configure an I2C peripheral.

This routine is a low-level driver that enables usage of an I2C peripheral, enables interrupts from the peripheral, and prepares the I2C state machine.

Note

This function is typically called once per I2C peripheral, as it is used for initial setup.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being opened
in	*i2c_setup*	Struct that the calling routine will use to set the parameters for I2C operation

4.11.2.5 i2c_start()

```
void i2c_start (
            I2C_TypeDef * i2c,
            uint32_t slave_addr,
            uint32_t slave_reg,
            bool read,
            uint32_t bytes_expected,
            uint32_t * rx_data,
            uint32_t tx_data,
            uint32_t callback )
```

Function that invokes I2C communication with an external device.

This function begins an exchange via I2C with an external device and invokes the I2C state machine behavior.

Note

A given I2C state machine may only have one active I2C exchange at any time. If a previous exchange is incomplete, a new one may not yet be started.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*slave_addr*	I2C address of external device that will be communicated with
in	*slave_reg*	Register within external device that will be read or written to via I2C
in	*bytes_expected*	Number of bytes expected to be sent or received via I2C
in	*data*	Pointer to data that will be sent or where received data will be stored
in	*callback*	Event callback for scheduler

4.12 i2c.h

Go to the documentation of this file.

```
1
9 //***********************************************************************************
10 // Include files
11 //***********************************************************************************
12 #ifndef HEADER_FILES_I2C_H_
13 #define HEADER_FILES_I2C_H_
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18 #include "em_cmu.h"
19 #include "em_i2c.h"
20
21 /* The developer's include statements */
22 #include "scheduler.h"
23 #include "sleep_routines.h"
24
25 //***********************************************************************************
26 // defined files
27 //***********************************************************************************
28 #define I2C_EM          EM2 // block from entering energy mode 2
29 #define I2C_READ_BIT    1
30 #define I2C_WRITE_BIT   0
31 #define BYTE            8
32
33 //***********************************************************************************
34 // global variables
35 //***********************************************************************************
36 typedef struct
37 {
38     bool enable;                // enable I2C on completion of i2c_open
39     bool master;                // behave as I2C master
40     uint32_t refFreq;           // I2C reference frequency
41     uint32_t freq;              // I2C frequency
42     I2C_ClockHLR_TypeDef clhr;              // clock low:high ratio
43     uint32_t out_pin_route_scl; // SCL route to GPIO port/pin
44     uint32_t out_pin_route_sda; // SDA route to GPIO port/pin
45     bool out_pin_enable_scl;    // enable SCL route
46     bool out_pin_enable_sda;    // enable SDA route
47 } I2C_OPEN_STRUCT;
48
49 //***********************************************************************************
50 // function prototypes
51 //***********************************************************************************
52 void i2c_open(I2C_TypeDef *i2c, I2C_OPEN_STRUCT *i2c_setup);
53 void i2c_start(I2C_TypeDef *i2c, uint32_t slave_addr, uint32_t slave_reg, bool read, uint32_t
       bytes_expected, uint32_t *rx_data, uint32_t tx_data, uint32_t callback);
54 bool i2c_busy(I2C_TypeDef *i2c);
55 void I2C0_IRQHandler(void);
56 void I2C1_IRQHandler(void);
57
58 #endif /* HEADER_FILES_I2C_H_ */
```

4.13 src/Header Files/LEDs_thunderboard.h File Reference

Contains functions that configure and control the RGB LEDs.

```
#include "stdbool.h"
#include "stdint.h"
#include "em_gpio.h"
#include "brd_config.h"
```

Macros

- #define **COLOR_RED** (0x01 << 0)
- #define **COLOR_GREEN** (0x01 << 1)
- #define **COLOR_BLUE** (0x01 << 2)
- #define **NO_COLOR** (0x00 << 0)
- #define **RGB_LED_0** (0x01 << 0)
- #define **RGB_LED_1** (0x01 << 1)
- #define **RGB_LED_2** (0x01 << 2)
- #define **RGB_LED_3** (0x01 << 3)
- #define **NO_LEDS** (0x00 << 0)
- #define **RGB_PWM_PERIOD** 20
- #define **RGB_PWM_ACTIVE** 1

Functions

- void rgb_init (void)

  Driver to initialize RGB LEDs.
- void leds_enabled (uint32_t leds, uint32_t color, bool enable)

  Driver to control RBG LEDs.

4.13.1 Detailed Description

Contains functions that configure and control the RGB LEDs.

Date

2021-09-12

4.13.2 Function Documentation

4.13.2.1 leds_enabled()

```
void leds_enabled (
            uint32_t leds,
            uint32_t color,
            bool enable )
```

Driver to control RBG LEDs.

This routine calls the necessary functions to turn the red, green, and blue LEDs on and off.

Note

This function may be called one or many times to manipulate the RGB LED outputs.

Parameters

in	*leds*	Parameter to select which RGB LED to manipulate
in	*color*	Parameter to select color of LED that will be manipulated
in	*enable*	Parameter to set the selected LED to either on or off

4.13.2.2 rgb_init()

```
void rgb_init (
              void  )
```

Driver to initialize RGB LEDs.

This routine calls the necessary functions to prepare the RGB LEDs as outputs.

Note

This function is typically called once to initialize the LEDs before use.

4.14 LEDs_thunderboard.h

Go to the documentation of this file.

```
1
8 #ifndef LED_thunderboard_HG
9 #define LED_thunderboard_HG
10
11 //***********************************************************************************
12 // Include files
13 //***********************************************************************************
14 /* System include statements */
15
16 /* Silicon Labs include statements */
17 #include "stdbool.h"
18 #include "stdint.h"
19 #include "em_gpio.h"
20
21 /* The developer's include statements */
22 #include "brd_config.h"
23
24 //***********************************************************************************
25 // defined files
26 //***********************************************************************************
27 #define COLOR_RED       (0x01 « 0)
28 #define COLOR_GREEN (0x01 « 1)
29 #define COLOR_BLUE  (0x01 « 2)
30 #define NO_COLOR        (0x00 « 0)
31
32 #define RGB_LED_0       (0x01 « 0)
33 #define RGB_LED_1       (0x01 « 1)
34 #define RGB_LED_2       (0x01 « 2)
35 #define RGB_LED_3       (0x01 « 3)
36 #define NO_LEDS         (0x00 « 0)
37
38 #define RGB_PWM_PERIOD  20
39 #define RGB_PWM_ACTIVE  1
40
41 //***********************************************************************************
42 // global variables
43 //***********************************************************************************
44
45 //***********************************************************************************
46 // function prototypes
47 //***********************************************************************************
48 void rgb_init(void);
49 void leds_enabled(uint32_t leds, uint32_t color, bool enable);
50
51 #endif
```

4.15 src/Header Files/letimer.h File Reference

Contains functions that configure and use LETIMER peripherals.

```
#include "em_letimer.h"
#include "em_gpio.h"
#include "em_cmu.h"
#include "em_assert.h"
#include "scheduler.h"
#include "sleep_routines.h"
```

Data Structures

- struct APP_LETIMER_PWM_TypeDef

Macros

- #define **LETIMER_HZ** 1000
- #define **LETIMER_EM** EM4

Functions

- void letimer_pwm_open (LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct)

  Driver to open and set an LETIMER peripheral in PWM mode.
- void letimer_start (LETIMER_TypeDef *letimer, bool enable)

  Function to enable/turn-on or disable/turn-off the LETIMER specified.
- void LETIMER0_IRQHandler (void)

  Interrupt handler to schedule events when an interrupt occurs.

4.15.1 Detailed Description

Contains functions that configure and use LETIMER peripherals.

Author

Mason Milligan

Date

2021-09-25

4.15.2 Function Documentation

4.15.2.1 LETIMER0_IRQHandler()

```
void LETIMER0_IRQHandler (
            void  )
```

Interrupt handler to schedule events when an interrupt occurs.

This function determines the type of interrupt that has occurred and schedules the relevant event for each type of interrupt flag.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.15.2.2 letimer_pwm_open()

```
void letimer_pwm_open (
            LETIMER_TypeDef * letimer,
            APP_LETIMER_PWM_TypeDef * app_letimer_struct )
```

Driver to open and set an LETIMER peripheral in PWM mode.

This routine is a low level driver. The application code calls this function to open one of the LETIMER peripherals for PWM operation to directly drive GPIO output pins of the device and/or create interrupts that can be used as a system "heart beat" or by a scheduler to determine whether any system functions need to be serviced.

Note

This function is normally called once to initialize the peripheral and the function letimer_start() is called to turn-on or turn-off the LETIMER PWM operation.

Parameters

in	*letimer*	Pointer to the base peripheral address of the LETIMER peripheral being opened
in	*app_letimer_struct*	Is the STRUCT that the calling routine will use to set the parameters for PWM operation

4.15.2.3 letimer_start()

```
void letimer_start (
            LETIMER_TypeDef * letimer,
            bool enable )
```

Function to enable/turn-on or disable/turn-off the LETIMER specified.

letimer_start uses the lower level API interface of the EM libraries to directly interface to the LETIMER peripheral to turn-on or off its counter

Note

This function should only be called to enable/turn-on the LETIMER once the LETIMER peripheral has been completely configured via its open driver

Parameters

in	*letimer*	Pointer to the base peripheral address of the LETIMER peripheral being opened
in	*enable*	Variable to turn-on the LETIMER if boolean value = true and turn-off the LETIMER if the boolean value = false

4.16 letimer.h

Go to the documentation of this file.

```
1
9 //*****************************************************************************
10 // Include files
11 //*****************************************************************************
12 #ifndef LETIMER_HG
13 #define LETIMER_HG
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18 #include "em_letimer.h"
19 #include "em_gpio.h"
20 #include "em_cmu.h"
21 #include "em_assert.h"
22
23 /* The developer's include statements */
24 #include "scheduler.h"
25 #include "sleep_routines.h"
26
27 //*****************************************************************************
28 // defined files
29 //*****************************************************************************
30 #define LETIMER_HZ  1000    // Utilizing ULFRCO oscillator for LETIMERs
31 #define LETIMER_EM  EM4     // Using the ULFRCO, block from entering Energy Mode 4
32
33 //*****************************************************************************
34 // global variables
35 //*****************************************************************************
36 typedef struct
37 {
38     bool debugRun;          // True = keep LETIMER running will halted
39     bool enable;               // enable the LETIMER upon completion of open
40     uint32_t out_pin_route0;        // out 0 route to gpio port/pin
41     uint32_t out_pin_route1;        // out 1 route to gpio port/pin
42     bool out_pin_0_en;      // enable out 0 route
43     bool out_pin_1_en;      // enable out 1 route
44     float period;               // seconds
45     float active_period;        // seconds
46     bool comp0_irq_enable;   // enable interrupt on comp0 interrupt
47     uint32_t comp0_cb;           // event callback for scheduler
48     bool comp1_irq_enable;   // enable interrupt on comp1 interrupt
49     uint32_t comp1_cb;           // event callback for scheduler
50     bool uf_irq_enable;      // enable interrupt on uf interrupt
51     uint32_t uf_cb;              // event callback for scheduler
52 } APP_LETIMER_PWM_TypeDef;
53
54 //*****************************************************************************
55 // function prototypes
56 //*****************************************************************************
57 void letimer_pwm_open(LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct);
58 void letimer_start(LETIMER_TypeDef *letimer, bool enable);
59 void LETIMER0_IRQHandler(void);
60
61 #endif
```

4.17 src/Header Files/main.h File Reference

Application exploring SLTB004A energy modes through controlling on-board RGB LEDs.

```
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "em_device.h"
#include "em_chip.h"
#include "em_emu.h"
#include "em_assert.h"
#include "app.h"
#include "brd_config.h"
```

4.17.1 Detailed Description

Application exploring SLTB004A energy modes through controlling on-board RGB LEDs.

4.17.2 License

Copyright 2018 Silicon Laboratories Inc. www.silabs.com

The licensor of this software is Silicon Laboratories Inc. Your use of this software is governed by the terms of Silicon Labs Master Software License Agreement (MSLA) available at www.silabs.com/about-us/legal/master-software-license-agreement. This software is distributed to you in Source Code format and is governed by the sections of the MSLA applicable to Source Code.

4.18 main.h

Go to the documentation of this file.

```
1 /***************************************************************************/
19 //***************************************************************************
20 // Include files
21 //***************************************************************************
22 #ifndef MAIN_HG
23 #define MAIN_HG
24
25 /* System include statements */
26 #include <stdint.h>
27 #include <stdbool.h>
28 #include <stdio.h>
29
30 /* Silicon Labs include statements */
31 #include "em_device.h"
32 #include "em_chip.h"
33 #include "em_emu.h"
34 #include "em_assert.h"
35
36 /* The developer's include statements */
37 #include "app.h"
38 #include "brd_config.h"
39
40 //***************************************************************************
41 // defined files
42 //***************************************************************************
43
44 //***************************************************************************
45 // global variables
46 //***************************************************************************
47
48 //***************************************************************************
49 // function prototypes
50 //***************************************************************************
51
52 #endif
```

4.19 src/Header Files/scheduler.h File Reference

Contains functions that manipulate and report the event schedule.

```
#include <stdint.h>
#include "em_assert.h"
#include "sleep_routines.h"
```

Functions

- void scheduler_open (void)

  Function to initialize schedule with no events.
- void add_scheduled_event (uint32_t event)

  Function to add an event to the schedule.
- void remove_scheduled_event (uint32_t event)

  Function to remove an event from the schedule.
- uint32_t get_scheduled_events (void)

  Function to report currently scheduled events.

4.19.1 Detailed Description

Contains functions that manipulate and report the event schedule.

Author

Mason Milligan

Date

2021-09-25

4.19.2 Function Documentation

4.19.2.1 add_scheduled_event()

```
void add_scheduled_event (
            uint32_t event )
```

Function to add an event to the schedule.

This function sets the associated bit for an event to 1, adding the event to the scheduler.

Note

This function is typically, but not necessarily always, called to schedule an event handler after an interrupt occurs.

Parameters

in	*event*	The each bit in this variable represents a different callback event. Bits set to 1 are scheduled.

4.19.2.2 get_scheduled_events()

```
uint32_t get_scheduled_events (
            void  )
```

Function to report currently scheduled events.

This function returns the current schedule.

Note

Note that in some instances, the schedule may be accessible and writable by interrupt handlers. This means the state of the schedule can change immediately after returning.

Parameters

out	*event_scheduled*	This variable has specific events associated with each bit. A 1 indicates that an event is scheduled, a 0 indicates that an event is not scheduled.

4.19.2.3 remove_scheduled_event()

```
void remove_scheduled_event (
            uint32_t event )
```

Function to remove an event from the schedule.

This function sets the associated bit for an event to 0, removing the event from the scheduler.

Note

This function is typically, but not necessarily always, called to remove an an event from the schedule before an event handler is run.

Parameters

in	*event*	The each bit in this variable represents a different callback event. Bits set to 1 are removed from the schedule.

4.19.2.4 scheduler_open()

```
void scheduler_open (
             void  )
```

Function to initialize schedule with no events.

This function sets the static int event_scheduled to 0, ensuring that the scheduler starts with no events scheduled.

Note

This function is intended to be called once before using the scheduler. Running this function at any other time will clear all scheduled events.

4.20 scheduler.h

Go to the documentation of this file.

```
1
9 //*******************************************************************************************
10 // Include files
11 //*******************************************************************************************
12 #ifndef SCHEDULER_HG
13 #define SCHEDULER_HG
14
15 /* System include statements */
16 #include <stdint.h>
17
18 /* Silicon Labs include statements */
19 #include "em_assert.h"
20
21 /* The developer's include statements */
22 #include "sleep_routines.h"
23
24 //*******************************************************************************************
25 // defined files
26 //*******************************************************************************************
27
28 //*******************************************************************************************
29 // global variables
30 //*******************************************************************************************
31
32 //*******************************************************************************************
33 // function prototypes
34 //*******************************************************************************************
35 void scheduler_open(void);
36 void add_scheduled_event(uint32_t event);
37 void remove_scheduled_event(uint32_t event);
38 uint32_t get_scheduled_events(void);
39
40 #endif
```

4.21 src/Header Files/si1133.h File Reference

Contains functions that enable interaction with the Si1133 sensor.

```
#include "i2c.h"
#include "brd_config.h"
#include "HW_delay.h"
```

Macros

- #define **NULL_CALLBACK** 0
- #define **SI1133_STARTUP_DELAY** 25
- #define **SI1133_I2C_ADDRESS** 0x55
- #define **SI1133_PART_ID_REG** 0x00
- #define **SI1133_PART_ID_REG_BYTES** 1
- #define **SI1133_RESPONSE0_REG** 0x11
- #define **SI1133_RESPONSE0_REG_BYTES** 1
- #define **SI1133_CMD_CTR_MASK** 0xF
- #define **SI1133_RESPONSE0_ERROR_MASK** 0x10
- #define **SI1133_INPUT0_REG** 0x0A
- #define **SI1133_INPUT0_REG_BYTES** 1
- #define **SI1133_ADCMUX_WHITE** 0b01011
- #define **SI1133_CHANNEL_0_ACTIVE** 1
- #define **SI1133_COMMAND_REG** 0x0B
- #define **SI1133_COMMAND_REG_BYTES** 1
- #define **SI1133_COMMAND_FORCE** 0x11
- #define **SI1133_HOSTOUT0_REG** 0x13
- #define **SI1133_HOSTOUT0_REG_BYTES** 2
- #define **SI11333_PARAM_WRITE_MASK** 0x80
- #define **SI11333_PARAM_READ_MASK** 0x40
- #define **SI1133_ADCCONFIG0_ADDRESS** 0x02
- #define **Si1133_CHAN_LIST_ADDRESS** 0x01

Functions

- void si1133_i2c_open (void)

  Function that configures one of the Mighty Gecko's I2C peripherals to interact with the Si1133 sensor.
- void si1133_read (I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback)

  Function that invokes I2C communication with the Si1133 sensor to collect data from the sensor.
- void si1133_write (I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback)

  Function that invokes I2C communication with the Si1133 sensor to write data to the sensor.
- uint32_t si1133_get_result (void)

  Function that returns si1133_read_result, which is where data received via I2C is stored.
- void si1133_force_command (I2C_TypeDef *i2c)

  Function that invokes a FORCE measurement on the Si1133 sensor.

4.21.1 Detailed Description

Contains functions that enable interaction with the Si1133 sensor.

Author

Mason Milligan

Date

2021-10-24

4.21.2 Function Documentation

4.21.2.1 si1133_force_command()

```
void si1133_force_command (
            I2C_TypeDef * i2c )
```

Function that invokes a FORCE measurement on the Si1133 sensor.

This function sends a command to the Si1133 sensor to initiate measurements and output them as specified by CHAN_LIST.

Note

This function should not be run until the CHAN_LIST parameter has been configured on the Si1133 sensor.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used

4.21.2.2 si1133_get_result()

```
uint32_t si1133_get_result (
            void  )
```

Function that returns si1133_read_result, which is where data received via I2C is stored.

This function simply returns the value stored in si1133_read_result.

Note

The static variable being returned by this function is subject to being updated at any time.

Parameters

out	*si1133_read_result*	Variable containing data received via I2C

4.21.2.3 si1133_i2c_open()

```
void si1133_i2c_open (
            void  )
```

Function that configures one of the Mighty Gecko's I2C peripherals to interact with the Si1133 sensor.

This function configures an I2C peripheral to properly communicate with an Si1133 sensor and prepares the I2C state machine for interaction with the sensor.

Note

This function is typically called once to perform initial configurations for interaction with the sensor.

4.21.2.4 si1133_read()

```
void si1133_read (
            I2C_TypeDef * i2c,
            uint32_t reg,
            uint32_t bytes,
            uint32_t callback )
```

Function that invokes I2C communication with the Si1133 sensor to collect data from the sensor.

This function begins I2C state machine operation to gather the part ID from the Si1133 sensor via I2C.

Note

This function results in si1133_read_result being overwritten.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*callback*	Callback value for scheduler so that an event may be triggered upon completion of the I2C interaction

4.21.2.5 si1133_write()

```
void si1133_write (
            I2C_TypeDef * i2c,
            uint32_t reg,
            uint32_t bytes,
            uint32_t callback )
```

Function that invokes I2C communication with the Si1133 sensor to write data to the sensor.

This function begins I2C state machine operation write data to the Si1133.

Note

This function reads si1133_write_data.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*callback*	Callback value for scheduler so that an event may be triggered upon completion of the I2C interaction

4.22 si1133.h

Go to the documentation of this file.

```
1
9 //*************************************************************************************
10 // Include files
11 //*************************************************************************************
12 #ifndef HEADER_FILES_SI1133_H_
13 #define HEADER_FILES_SI1133_H_
14
15 /* System include statements */
16
17 /* Silicon Labs include statements */
18
19 /* The developer's include statements */
20 #include "i2c.h"
21 #include "brd_config.h"
22 #include "HW_delay.h"
23
24 //*************************************************************************************
25 // defined files
26 //*************************************************************************************
27 /* empty scheduler callback */
28 #define NULL_CALLBACK                 0
29
30 /* time to wait between startup and interaction with Si1133 sensor */
31 #define SI1133_STARTUP_DELAY             25      // startup time in ms
32
33 /* Si1133 I2C definitions */
34 #define SI1133_I2C_ADDRESS              0x55
35 #define SI1133_PART_ID_REG              0x00
36 #define SI1133_PART_ID_REG_BYTES        1
37
38 #define SI1133_RESPONSE0_REG            0x11
39 #define SI1133_RESPONSE0_REG_BYTES      1
40 #define SI1133_CMD_CTR_MASK             0xF
41 #define SI1133_RESPONSE0_ERROR_MASK     0x10
42
43 #define SI1133_INPUT0_REG               0x0A
44 #define SI1133_INPUT0_REG_BYTES         1
45 #define SI1133_ADCMUX_WHITE             0b01011
46 #define SI1133_CHANNEL_0_ACTIVE         1
47
48 #define SI1133_COMMAND_REG              0x0B
49 #define SI1133_COMMAND_REG_BYTES        1
50 #define SI1133_COMMAND_FORCE            0x11
51
52 #define SI1133_HOSTOUT0_REG             0x13
53 #define SI1133_HOSTOUT0_REG_BYTES       2
54
55 #define SI11333_PARAM_WRITE_MASK        0x80
56 #define SI11333_PARAM_READ_MASK         0x40
57 #define SI1133_ADCCONFIG0_ADDRESS       0x02
58 #define Si1133_CHAN_LIST_ADDRESS        0x01
59
60 //*************************************************************************************
61 // global variables
62 //*************************************************************************************
63
64 //*************************************************************************************
65 // function prototypes
66 //*************************************************************************************
67 void si1133_i2c_open(void);
68 void si1133_read(I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback);
69 void si1133_write(I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback);
70 uint32_t si1133_get_result(void);
71 void si1133_force_command(I2C_TypeDef *i2c);
72
73 #endif /* HEADER_FILES_SI1133_H_ */
```

4.23 src/Header Files/sleep_routines.h File Reference

Contains functions that control the system's energy mode.

```
#include "em_emu.h"
#include "em_core.h"
#include "em_assert.h"
```

Macros

- #define **EM0** 0
- #define **EM1** 1
- #define **EM2** 2
- #define **EM3** 3
- #define **EM4** 4
- #define **MAX_ENERGY_MODES** 5

Functions

- void sleep_open (void)

  Function to initialize the lowest_energy_mode array with all energy modes unblocked.
- void sleep_block_mode (uint32_t EM)

  Function to block the system from entering a given energy mode.
- void sleep_unblock_mode (uint32_t EM)

  Function to unblock the system from entering a given energy mode.
- void enter_sleep (void)

  Function to enter the deepest allowed sleep state.

4.23.1 Detailed Description

Contains functions that control the system's energy mode.

4.23.2 License

(C) Copyright 2015 Silicon Labs, http://www.silabs.com

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no obligation to support this Software. Silicon Labs is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Silicon Labs will not be liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

4.23.3 Function Documentation

4.23.3.1 enter_sleep()

```
void enter_sleep (
            void  )
```

Function to enter the deepest allowed sleep state.

This function checks lowest_energy_mode for the deepest allowable sleep state, then enters it.

Note

If an energy mode has a value greater than 0, it and all deeper sleep states will not be entered.

4.23.3.2 sleep_block_mode()

```
void sleep_block_mode (
            uint32_t EM )
```

Function to block the system from entering a given energy mode.

This function increases the given energy state's value in lowest_energy_mode by 1, preventing that energy state from being entered.

Note

Multiple peripherals may block the same energy mode. If an energy mode's value is greater than 0, that energy mode and all deeper sleep states are blocked.

Parameters

in	*EM*	Parameter to select which energy mode to block

4.23.3.3 sleep_open()

```
void sleep_open (
            void  )
```

Function to initialize the lowest_energy_mode array with all energy modes unblocked.

This function sets all array elements to 0, unblocking all energy modes.

Note

This function is intended to be run once before configuring most peripherals. Running this function more than once or after peripherals are configured may result in sleep states being inadvertently being unblocked.

4.23.3.4 sleep_unblock_mode()

```
void sleep_unblock_mode (
             uint32_t EM )
```

Function to unblock the system from entering a given energy mode.

This function decreases the given energy state's value in lowest_energy_mode by 1.

Note

Multiple peripherals may block the same energy mode. If an energy mode's value is greater than 0, that energy mode and all deeper sleep states are blocked.

Parameters

in	*EM*	Parameter to select which energy mode to unblock

4.24 sleep_routines.h

Go to the documentation of this file.

```
1 /***************************************************************************/
31 //*****************************************************************************
32 // Include files
33 //*****************************************************************************
34 #ifndef HEADER_FILES_SLEEP_ROUTINES_H_
35 #define HEADER_FILES_SLEEP_ROUTINES_H_
36
37 /* System include statements */
38
39 /* Silicon Labs include statements */
40 #include "em_emu.h"
41 #include "em_core.h"
42 #include "em_assert.h"
43
44 /* The developer's include statements */
45
46 //*****************************************************************************
47 // defined files
48 //*****************************************************************************
49 #define EM0               0
50 #define EM1               1
51 #define EM2               2
52 #define EM3               3
53 #define EM4               4
54 #define MAX_ENERGY_MODES  5
55
56 //*****************************************************************************
57 // global variables
58 //*****************************************************************************
59
60 //*****************************************************************************
61 // function prototypes
62 //*****************************************************************************
63 void sleep_open(void);
64 void sleep_block_mode(uint32_t EM);
```

```
65 void sleep_unblock_mode(uint32_t EM);
66 void enter_sleep(void);
67
68 #endif /* HEADER_FILES_SLEEP_ROUTINES_H_ */
```

4.25 src/main.c File Reference

Application exploring SLTB004A energy modes through controlling on-board RGB LEDs.

```
#include "main.h"
```

Functions

- int **main** (void)

4.25.1 Detailed Description

Application exploring SLTB004A energy modes through controlling on-board RGB LEDs.

4.25.2 License

Copyright 2018 Silicon Laboratories Inc. www.silabs.com

The licensor of this software is Silicon Laboratories Inc. Your use of this software is governed by the terms of Silicon Labs Master Software License Agreement (MSLA) available at www.silabs.com/about-us/legal/master-software-license-agreement. This software is distributed to you in Source Code format and is governed by the sections of the MSLA applicable to Source Code.

4.26 src/Source Files/app.c File Reference

Contains high-level functions that drive the application.

```
#include "app.h"
```

Functions

- static void app_letimer_pwm_open (float period, float act_period, uint32_t out0_route, uint32_t out1_route)

  Function that prepares a struct with necessary information to set LETIMER0 to interact with LED 1.
- void app_peripheral_setup (void)

  Function that calls several functions that open and initialize all necessary peripherals.
- void scheduled_letimer0_uf_cb (void)

  Event handler that responds to a LETIMER0_UF_CB event.
- void scheduled_letimer0_comp0_cb (void)

  Event handler that responds to a LETIMER0_COMP0_CB event.
- void scheduled_letimer0_comp1_cb (void)

  Event handler that invokes an I2C read in response to a LETIMER0_COMP1_CB event.
- void scheduled_si1133_reg_read_cb (void)

  Event handler that turns RGB LED red or green depending on result of I2C interaction in response to a SI1133_↩ REG_READ_CB event.

4.26.1 Detailed Description

Contains high-level functions that drive the application.

Author

Mason Milligan

Date

2021-10-24

4.26.2 Function Documentation

4.26.2.1 app_letimer_pwm_open()

```
void app_letimer_pwm_open (
            float period,
            float act_period,
            uint32_t out0_route,
            uint32_t out1_route )  [static]
```

Function that prepares a struct with necessary information to set LETIMER0 to interact with LED 1.

This function builds a struct with information needed to set the LETIMER to blink LED 1. This includes the timer period, the timer active period, routing information, and pin enables.

Note

This function typically runs once to provide the LETIMER peripheral driver with the desired operating settings.

Parameters

in	*period*	Period, in seconds, to run the LETIMER
in	*act_period*	Time, in seconds, that the output of the LETIMER is active
in	*out0_route*	Route from LETIMER0 to a peripheral, see EFR32MG12 datasheet table 6.8
in	*out1_route*	Used to route LETIMER0 to another peripheral

4.26.2.2 app_peripheral_setup()

```
void app_peripheral_setup (
            void  )
```

Function that calls several functions that open and initialize all necessary peripherals.

This routine calls the setup processes for the clock and gpio peripherals, gathers PWM period and routing settings, then starts the low energy timer.

Note

This function is typically run once. It calls all necessary functions to prepare peripherals and begin operation.

4.26.2.3 scheduled_letimer0_comp0_cb()

```
void scheduled_letimer0_comp0_cb (
            void  )
```

Event handler that responds to a LETIMER0_COMP0_CB event.

This function contains an assert that is set to always fail.

Note

LETIMER0_COMP0_CB events should not occur in this application. If one does occur, the assert statement may be used to trace the cause.

4.26.2.4 scheduled_letimer0_comp1_cb()

```
void scheduled_letimer0_comp1_cb (
            void  )
```

Event handler that invokes an I2C read in response to a LETIMER0_COMP1_CB event.

This function calls si1133_read to begin I2C communication with Si1133.

Note

Typically, this function should only be called when a LETIMER0_COMP1_CB event occurs.

4.26.2.5 scheduled_letimer0_uf_cb()

```
void scheduled_letimer0_uf_cb (
            void  )
```

Event handler that responds to a LETIMER0_UF_CB event.

This function contains an assert that is set to always fail.

Note

LETIMER0_UF_CB events should not occur in this application. If one does occur, the assert statement may be used to trace the cause.

4.26.2.6 scheduled_si1133_reg_read_cb()

```
void scheduled_si1133_reg_read_cb (
            void  )
```

Event handler that turns RGB LED red or green depending on result of I2C interaction in response to a SI1133_↩ REG_READ_CB event.

This function calls si1133_get_result to collect the data retrieved via I2C for comparison with the Si1133 part ID. If the values match, a green LED is enabled. If they do not match, a red LED is enabled.

Note

Typically, this function should only be called when a SI1133_REG_READ_CB event occurs.

4.27 src/Source Files/cmu.c File Reference

Contains functions that configure clocks and clock routing.

```
#include "cmu.h"
```

Functions

- void cmu_open (void)

  Function to disable unused clocks and route ULFRCO.

4.27.1 Detailed Description

Contains functions that configure clocks and clock routing.

Author

Mason Milligan

Date

2021-09-12

4.27.2 Function Documentation

4.27.2.1 cmu_open()

```
void cmu_open (
            void  )
```

Function to disable unused clocks and route ULFRCO.

Since LFRCO and LFXO are not needed, they are disabled here. Additionally, ULFRCO is routed through CMU_↩ LFACLKSEL.LFA and CMU_LFACLKEN0.LETIMER0.

Note

This function is typically called once to disable unused clocks and prepare ULFRCO to be used.

4.28 src/Source Files/gpio.c File Reference

Contains functions that configure GPIO peripherals.

```
#include "gpio.h"
```

Functions

- void gpio_open (void)

  Function to configure LED pins for use.

4.28.1 Detailed Description

Contains functions that configure GPIO peripherals.

Author

Mason Milligan

Date

2021-09-25

4.28.2 Function Documentation

4.28.2.1 gpio_open()

```
void gpio_open (
            void  )
```

Function to configure LED pins for use.

This function enables the red and green LED pins and sets the appropriate drive strength.

Note

This function is typically run once to prepare the LEDs as an output.

4.29 src/Source Files/HW_delay.c File Reference

Contains function that enables a time delay.

```
#include "HW_delay.h"
```

Functions

- void timer_delay (uint32_t ms_delay)
  Function to stall a specified number of milliseconds.

4.29.1 Detailed Description

Contains function that enables a time delay.

Author

Keith Graham

Date

2020-04-19

4.29.2 Function Documentation

4.29.2.1 timer_delay()

```
void timer_delay (
            uint32_t ms_delay )
```

Function to stall a specified number of milliseconds.

This function enables stalling for a specified amount of time.

Parameters

in	*ms_delay*	Time to wait in milliseconds

4.30 src/Source Files/i2c.c File Reference

Contains functions that enable I2C state machine functionality.

```
#include "i2c.h"
```

Data Structures

- struct I2C_STATE_MACHINE

Enumerations

- enum **DEFINED_STATES** {
  idle , **init** , **reg_sel** , **rep_start** ,
  get_data , **send_data** , **stop** }

Functions

- static void i2c_bus_reset (I2C_TypeDef *i2c)

  Function to reset I2C state machine of given I2C peripheral and all connected external I2C devices.
- static void i2c_send (I2C_STATE_MACHINE *i2c_sm)

  Function run during I2C state machine functionality to perform a state's respective sending behavior.
- static void i2c_receive (I2C_STATE_MACHINE *i2c_sm)

  Function run during I2C communication when data is received.
- static void i2c_ACK (I2C_STATE_MACHINE *i2c_sm)

  Function run during I2C communication when an ACK is received.
- static void i2c_NACK (I2C_STATE_MACHINE *i2c_sm)

  Function run during I2C communication when a NACK is received.
- static void i2c_end (I2C_STATE_MACHINE *i2c_sm)

  Function to end an I2C interaction.
- void i2c_open (I2C_TypeDef *i2c, I2C_OPEN_STRUCT *i2c_setup)

  Driver to initialize and configure an I2C peripheral.
- void i2c_start (I2C_TypeDef *i2c, uint32_t slave_addr, uint32_t slave_reg, bool read, uint32_t bytes_↩ expected, uint32_t *rx_data, uint32_t tx_data, uint32_t callback)

  Function that invokes I2C communication with an external device.
- bool i2c_busy (I2C_TypeDef *i2c)

  Function to report the busy status of the I2C state machine associated with the given I2C peripheral.
- void I2C0_IRQHandler (void)

  Interrupt handler to respond to I2C-related interrupts.
- void I2C1_IRQHandler (void)

  Interrupt handler to respond to I2C-related interrupts.

Variables

- static I2C_STATE_MACHINE **i2c_sm_0**
- static I2C_STATE_MACHINE **i2c_sm_1**

4.30.1 Detailed Description

Contains functions that enable I2C state machine functionality.

Author

Mason Milligan

Date

2021-10-24

4.30.2 Function Documentation

4.30.2.1 I2C0_IRQHandler()

```
void I2C0_IRQHandler (
            void  )
```

Interrupt handler to respond to I2C-related interrupts.

This function determines the nature of the I2C interrupt that occurred and calls the relevant function for the event that occurred.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.30.2.2 I2C1_IRQHandler()

```
void I2C1_IRQHandler (
            void  )
```

Interrupt handler to respond to I2C-related interrupts.

This function determines the nature of the I2C interrupt that occurred and calls the relevant function for the event that occurred.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.30.2.3 i2c_ACK()

```
void i2c_ACK (
            I2C_STATE_MACHINE * i2c_sm )  [static]
```

Function run during I2C communication when an ACK is received.

This function is run when an ACK has been received via I2C. The following behavior is determined based on the current state of the I2C state machine.

Note

This function is typically only called in response to an ACK interrupt.

Parameters

in	*i2c_sm*	Struct containing I2C state machine information

4.30.2.4 i2c_bus_reset()

```
void i2c_bus_reset (
            I2C_TypeDef * i2c )  [static]
```

Function to reset I2C state machine of given I2C peripheral and all connected external I2C devices.

This function resets the Mighty Gecko's internal I2C state machine and the I2C state machine of any external peripheral connected via I2C.

Note

This function is typically called at the beginning of any I2C interaction.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being reset

4.30.2.5 i2c_busy()

```
bool i2c_busy (
            I2C_TypeDef * i2c )
```

Function to report the busy status of the I2C state machine associated with the given I2C peripheral.

This function returns the busy status variable of the selected I2C state machine variable.

Note

This function may be called by higher-level routines to check the status of the private I2C state machine variable.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
out	*busy*	Boolean indicating the busy status of the selected I2C state machine. True indicates busy; false indicates not busy.

4.30.2.6 i2c_end()

```
void i2c_end (
            I2C_STATE_MACHINE * i2c_sm )  [static]
```

Function to end an I2C interaction.

This function ends an I2C interaction by resetting the I2C state machine and unblocking sleep modes that were restricted by I2C.

Note

This function is typically only called in response to an MSTOP interrupt.

Parameters

in	*i2c_sm*	Struct containing I2C state machine information

4.30.2.7 i2c_NACK()

```
void i2c_NACK (
            I2C_STATE_MACHINE * i2c_sm )  [static]
```

Function run during I2C communication when a NACK is received.

This function is run when a NACK has been received via I2C. The following behavior is determined based on the current state of the I2C state machine.

Note

This function is typically only called in response to a NACK interrupt.

Parameters

in	*i2c_sm*	Struct containing I2C state machine information

4.30.2.8 i2c_open()

```
void i2c_open (
            I2C_TypeDef * i2c,
            I2C_OPEN_STRUCT * i2c_setup )
```

Driver to initialize and configure an I2C peripheral.

This routine is a low-level driver that enables usage of an I2C peripheral, enables interrupts from the peripheral, and prepares the I2C state machine.

Note

This function is typically called once per I2C peripheral, as it is used for initial setup.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being opened
in	*i2c_setup*	Struct that the calling routine will use to set the parameters for I2C operation

4.30.2.9 i2c_receive()

```
void i2c_receive (
            I2C_STATE_MACHINE * i2c_sm )  [static]
```

Function run during I2C communication when data is received.

This function is run when data has been received via I2C and is in the receive buffer.

Note

This function is typically only called in response to an RXDATAV interrupt.

Parameters

in	*i2c_sm*	Struct containing I2C state machine information

4.30.2.10 i2c_send()

```
void i2c_send (
            I2C_STATE_MACHINE * i2c_sm )  [static]
```

Function run during I2C state machine functionality to perform a state's respective sending behavior.

This function sends data via I2C that varies based on the current status of the I2C state machine.

Note

This function is typically run when initiating an exchange via I2C or when responding to a message via I2C.

Parameters

in	*i2c_sm*	Struct containing I2C state machine information

4.30.2.11 i2c_start()

```
void i2c_start (
            I2C_TypeDef * i2c,
            uint32_t slave_addr,
            uint32_t slave_reg,
            bool read,
            uint32_t bytes_expected,
            uint32_t * rx_data,
            uint32_t tx_data,
            uint32_t callback )
```

Function that invokes I2C communication with an external device.

This function begins an exchange via I2C with an external device and invokes the I2C state machine behavior.

Note

A given I2C state machine may only have one active I2C exchange at any time. If a previous exchange is incomplete, a new one may not yet be started.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*slave_addr*	I2C address of external device that will be communicated with
in	*slave_reg*	Register within external device that will be read or written to via I2C
in	*bytes_expected*	Number of bytes expected to be sent or received via I2C
in	*data*	Pointer to data that will be sent or where received data will be stored
in	*callback*	Event callback for scheduler

4.31 src/Source Files/LEDs_thunderboard.c File Reference

Contains functions that configure and control the RGB LEDs.

```
#include "LEDs_thunderboard.h"
```

Functions

- void rgb_init (void)

  Driver to initialize RGB LEDs.
- void leds_enabled (uint32_t leds, uint32_t color, bool enable)

  Driver to control RBG LEDs.

Variables

- bool **rgb_enabled_status**

4.31.1 Detailed Description

Contains functions that configure and control the RGB LEDs.

Date

2021-09-12

4.31.2 Function Documentation

4.31.2.1 leds_enabled()

```
void leds_enabled (
            uint32_t leds,
            uint32_t color,
            bool enable )
```

Driver to control RBG LEDs.

This routine calls the necessary functions to turn the red, green, and blue LEDs on and off.

Note

This function may be called one or many times to manipulate the RGB LED outputs.

Parameters

in	*leds*	Parameter to select which RGB LED to manipulate
in	*color*	Parameter to select color of LED that will be manipulated
in	*enable*	Parameter to set the selected LED to either on or off

4.31.2.2 rgb_init()

```
void rgb_init (
            void  )
```

Driver to initialize RGB LEDs.

This routine calls the necessary functions to prepare the RGB LEDs as outputs.

Note

This function is typically called once to initialize the LEDs before use.

4.32 src/Source Files/letimer.c File Reference

Contains functions that configure and use LETIMER peripherals.

```
#include "letimer.h"
```

Functions

- void letimer_pwm_open (LETIMER_TypeDef *letimer, APP_LETIMER_PWM_TypeDef *app_letimer_struct)

  Driver to open and set an LETIMER peripheral in PWM mode.
- void letimer_start (LETIMER_TypeDef *letimer, bool enable)

  Function to enable/turn-on or disable/turn-off the LETIMER specified.
- void LETIMER0_IRQHandler (void)

  Interrupt handler to schedule events when an interrupt occurs.

Variables

- static uint32_t **scheduled_comp0_cb**
- static uint32_t **scheduled_comp1_cb**
- static uint32_t **scheduled_uf_cb**

4.32.1 Detailed Description

Contains functions that configure and use LETIMER peripherals.

Author

Mason Milligan

Date

2021-09-25

4.32.2 Function Documentation

4.32.2.1 LETIMER0_IRQHandler()

```
void LETIMER0_IRQHandler (
            void  )
```

Interrupt handler to schedule events when an interrupt occurs.

This function determines the type of interrupt that has occurred and schedules the relevant event for each type of interrupt flag.

Note

This function should not be called manually, as it is intended to only be called by the interrupt controller when an interrupt occurs.

4.32.2.2 letimer_pwm_open()

```
void letimer_pwm_open (
            LETIMER_TypeDef * letimer,
            APP_LETIMER_PWM_TypeDef * app_letimer_struct )
```

Driver to open and set an LETIMER peripheral in PWM mode.

This routine is a low level driver. The application code calls this function to open one of the LETIMER peripherals for PWM operation to directly drive GPIO output pins of the device and/or create interrupts that can be used as a system "heart beat" or by a scheduler to determine whether any system functions need to be serviced.

Note

This function is normally called once to initialize the peripheral and the function letimer_start() is called to turn-on or turn-off the LETIMER PWM operation.

Parameters

in	*letimer*	Pointer to the base peripheral address of the LETIMER peripheral being opened
in	*app_letimer_struct*	Is the STRUCT that the calling routine will use to set the parameters for PWM operation

4.32.2.3 letimer_start()

```
void letimer_start (
```

```
LETIMER_TypeDef * letimer,
bool enable )
```

Function to enable/turn-on or disable/turn-off the LETIMER specified.

letimer_start uses the lower level API interface of the EM libraries to directly interface to the LETIMER peripheral to turn-on or off its counter

Note

This function should only be called to enable/turn-on the LETIMER once the LETIMER peripheral has been completely configured via its open driver

Parameters

in	*letimer*	Pointer to the base peripheral address of the LETIMER peripheral being opened
in	*enable*	Variable to turn-on the LETIMER if boolean value = true and turn-off the LETIMER if the boolean value = false

4.33 src/Source Files/scheduler.c File Reference

Contains functions that manipulate and report the event schedule.

```
#include "scheduler.h"
#include "em_assert.h"
#include "em_core.h"
#include "em_emu.h"
```

Functions

- void scheduler_open (void)

  Function to initialize schedule with no events.
- void add_scheduled_event (uint32_t event)

  Function to add an event to the schedule.
- void remove_scheduled_event (uint32_t event)

  Function to remove an event from the schedule.
- uint32_t get_scheduled_events (void)

  Function to report currently scheduled events.

Variables

- static unsigned int **event_scheduled**

4.33.1 Detailed Description

Contains functions that manipulate and report the event schedule.

Author

Mason Milligan

Date

2021-09-25

4.33.2 Function Documentation

4.33.2.1 add_scheduled_event()

```
void add_scheduled_event (
            uint32_t event )
```

Function to add an event to the schedule.

This function sets the associated bit for an event to 1, adding the event to the scheduler.

Note

This function is typically, but not necessarily always, called to schedule an event handler after an interrupt occurs.

Parameters

in	*event*	The each bit in this variable represents a different callback event. Bits set to 1 are scheduled.

4.33.2.2 get_scheduled_events()

```
uint32_t get_scheduled_events (
            void  )
```

Function to report currently scheduled events.

This function returns the current schedule.

Note

Note that in some instances, the schedule may be accessible and writable by interrupt handlers. This means the state of the schedule can change immediately after returning.

Parameters

out	*event_scheduled*	This variable has specific events associated with each bit. A 1 indicates that an event is scheduled, a 0 indicates that an event is not scheduled.

4.33.2.3 remove_scheduled_event()

```
void remove_scheduled_event (
             uint32_t event )
```

Function to remove an event from the schedule.

This function sets the associated bit for an event to 0, removing the event from the scheduler.

Note

This function is typically, but not necessarily always, called to remove an an event from the schedule before an event handler is run.

Parameters

in	*event*	The each bit in this variable represents a different callback event. Bits set to 1 are removed from the schedule.

4.33.2.4 scheduler_open()

```
void scheduler_open (
             void  )
```

Function to initialize schedule with no events.

This function sets the static int event_scheduled to 0, ensuring that the scheduler starts with no events scheduled.

Note

This function is intended to be called once before using the scheduler. Running this function at any other time will clear all scheduled events.

4.34 src/Source Files/si1133.c File Reference

Contains functions that enable interaction with the Si1133 sensor.

```
#include "si1133.h"
```

Functions

- void si1133_configure (I2C_TypeDef *i2c)

  Function to configure the Si1133 sensor to be used by the I2C state machine.
- void si1133_i2c_open (void)

  Function that configures one of the Mighty Gecko's I2C peripherals to interact with the Si1133 sensor.
- void si1133_read (I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback)

  Function that invokes I2C communication with the Si1133 sensor to collect data from the sensor.
- void si1133_write (I2C_TypeDef *i2c, uint32_t reg, uint32_t bytes, uint32_t callback)

  Function that invokes I2C communication with the Si1133 sensor to write data to the sensor.
- uint32_t si1133_get_result (void)

  Function that returns si1133_read_result, which is where data received via I2C is stored.
- void si1133_force_command (I2C_TypeDef *i2c)

  Function that invokes a FORCE measurement on the Si1133 sensor.

Variables

- static uint32_t **si1133_read_result**
- static uint32_t **si1133_write_data**

4.34.1 Detailed Description

Contains functions that enable interaction with the Si1133 sensor.

Author

Mason Milligan

Date

2021-10-24

4.34.2 Function Documentation

4.34.2.1 si1133_configure()

```
void si1133_configure (
            I2C_TypeDef * i2c )
```

Function to configure the Si1133 sensor to be used by the I2C state machine.

This function sets the Si1133 sensor's parameters to measure white ambient light and function with the I2C state machine.

Note

This function is typically called once by si1133_i2c_open to prepare the sensor for proper I2C communication.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used

4.34.2.2 si1133_force_command()

```
void si1133_force_command (
            I2C_TypeDef * i2c )
```

Function that invokes a FORCE measurement on the Si1133 sensor.

This function sends a command to the Si1133 sensor to initiate measurements and output them as specified by CHAN_LIST.

Note

This function should not be run until the CHAN_LIST parameter has been configured on the Si1133 sensor.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used

4.34.2.3 si1133_get_result()

```
uint32_t si1133_get_result (
            void  )
```

Function that returns si1133_read_result, which is where data received via I2C is stored.

This function simply returns the value stored in si1133_read_result.

Note

The static variable being returned by this function is subject to being updated at any time.

Parameters

out	*si1133_read_result*	Variable containing data received via I2C

4.34.2.4 si1133_i2c_open()

```
void si1133_i2c_open (
```

```
            void  )
```

Function that configures one of the Mighty Gecko's I2C peripherals to interact with the Si1133 sensor.

This function configures an I2C peripheral to properly communicate with an Si1133 sensor and prepares the I2C state machine for interaction with the sensor.

Note

This function is typically called once to perform initial configurations for interaction with the sensor.

4.34.2.5 si1133_read()

```
void si1133_read (
            I2C_TypeDef * i2c,
            uint32_t reg,
            uint32_t bytes,
            uint32_t callback )
```

Function that invokes I2C communication with the Si1133 sensor to collect data from the sensor.

This function begins I2C state machine operation to gather the part ID from the Si1133 sensor via I2C.

Note

This function results in si1133_read_result being overwritten.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*callback*	Callback value for scheduler so that an event may be triggered upon completion of the I2C interaction

4.34.2.6 si1133_write()

```
void si1133_write (
            I2C_TypeDef * i2c,
            uint32_t reg,
            uint32_t bytes,
            uint32_t callback )
```

Function that invokes I2C communication with the Si1133 sensor to write data to the sensor.

This function begins I2C state machine operation write data to the Si1133.

Note

This function reads si1133_write_data.

Parameters

in	*i2c*	Pointer to the base peripheral address of the I2C peripheral being used
in	*callback*	Callback value for scheduler so that an event may be triggered upon completion of the I2C interaction

4.35 src/Source Files/sleep_routines.c File Reference

Contains functions that control the system's energy mode.

```
#include "sleep_routines.h"
```

Functions

- void sleep_open (void)

  Function to initialize the lowest_energy_mode array with all energy modes unblocked.
- void sleep_block_mode (uint32_t EM)

  Function to block the system from entering a given energy mode.
- void sleep_unblock_mode (uint32_t EM)

  Function to unblock the system from entering a given energy mode.
- void enter_sleep (void)

  Function to enter the deepest allowed sleep state.

Variables

- static int **lowest_energy_mode** [MAX_ENERGY_MODES]

4.35.1 Detailed Description

Contains functions that control the system's energy mode.

4.35.2 License

(C) Copyright 2015 Silicon Labs, http://www.silabs.com

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

DISCLAIMER OF WARRANTY/LIMITATION OF REMEDIES: Silicon Labs has no obligation to support this Software. Silicon Labs is providing the Software "AS IS", with no express or implied warranties of any kind, including, but not limited to, any implied warranties of merchantability or fitness for any particular purpose or warranties against infringement of any proprietary rights of a third party.

Silicon Labs will not be liable for any consequential, incidental, or special damages, or any other relief, or for any claim by any third party, arising from your use of this Software.

4.35.3 Function Documentation

4.35.3.1 enter_sleep()

```
void enter_sleep (
            void  )
```

Function to enter the deepest allowed sleep state.

This function checks lowest_energy_mode for the deepest allowable sleep state, then enters it.

Note

If an energy mode has a value greater than 0, it and all deeper sleep states will not be entered.

4.35.3.2 sleep_block_mode()

```
void sleep_block_mode (
            uint32_t EM )
```

Function to block the system from entering a given energy mode.

This function increases the given energy state's value in lowest_energy_mode by 1, preventing that energy state from being entered.

Note

Multiple peripherals may block the same energy mode. If an energy mode's value is greater than 0, that energy mode and all deeper sleep states are blocked.

Parameters

in	*EM*	Parameter to select which energy mode to block

4.35.3.3 sleep_open()

```
void sleep_open (
            void  )
```

Function to initialize the lowest_energy_mode array with all energy modes unblocked.

This function sets all array elements to 0, unblocking all energy modes.

Note

This function is intended to be run once before configuring most peripherals. Running this function more than once or after peripherals are configured may result in sleep states being inadvertently being unblocked.

4.35.3.4 sleep_unblock_mode()

```
void sleep_unblock_mode (
             uint32_t EM )
```

Function to unblock the system from entering a given energy mode.

This function decreases the given energy state's value in lowest_energy_mode by 1.

Note

Multiple peripherals may block the same energy mode. If an energy mode's value is greater than 0, that energy mode and all deeper sleep states are blocked.

Parameters

in	*EM*	Parameter to select which energy mode to unblock

Index