Bluetooth Low Energy Heart Rate Monitor Reference Design

MKW40

0.1
Oct 2015

freescale™

Contents

Chapter 1
Introduction

Chapter 2
Application Layer

Section number	Title	Page

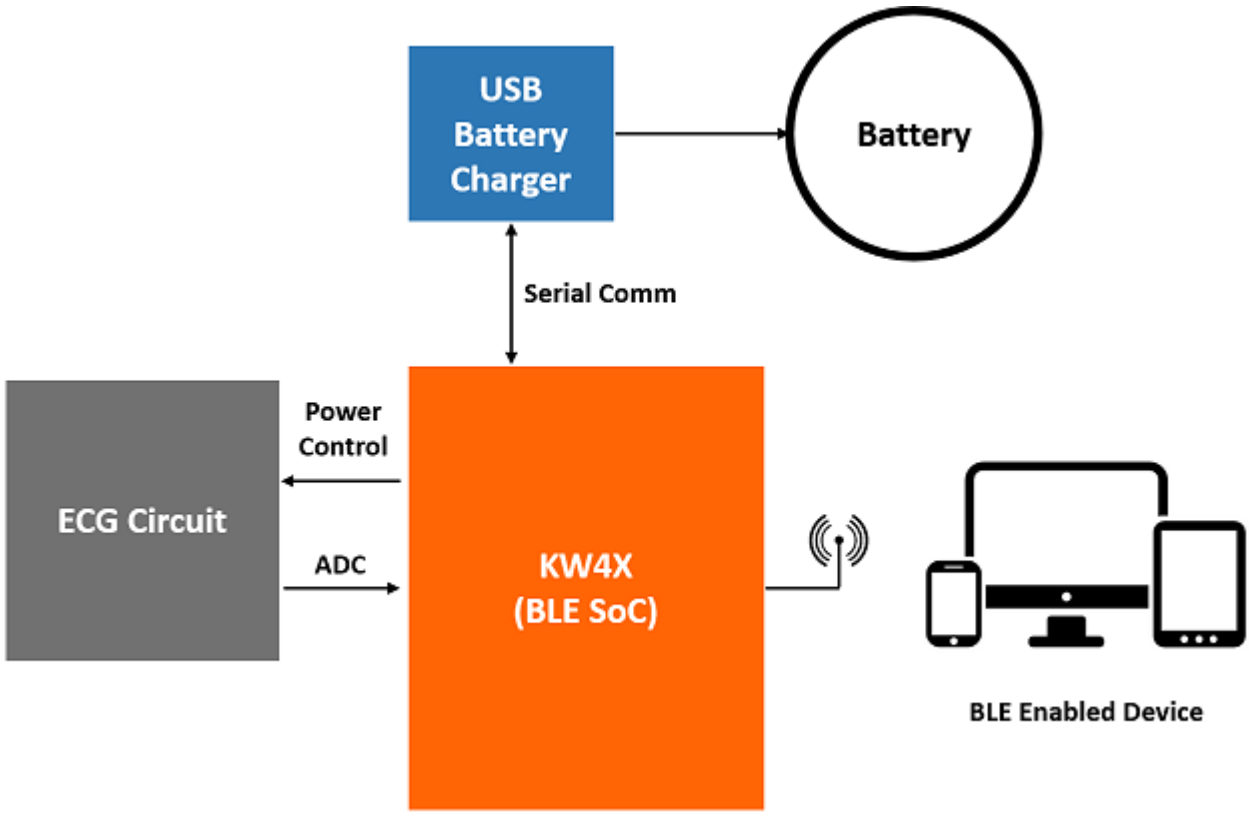
Chapter 5
Transport Layer

Chapter 6
File Documentation

Chapter 1
Introduction

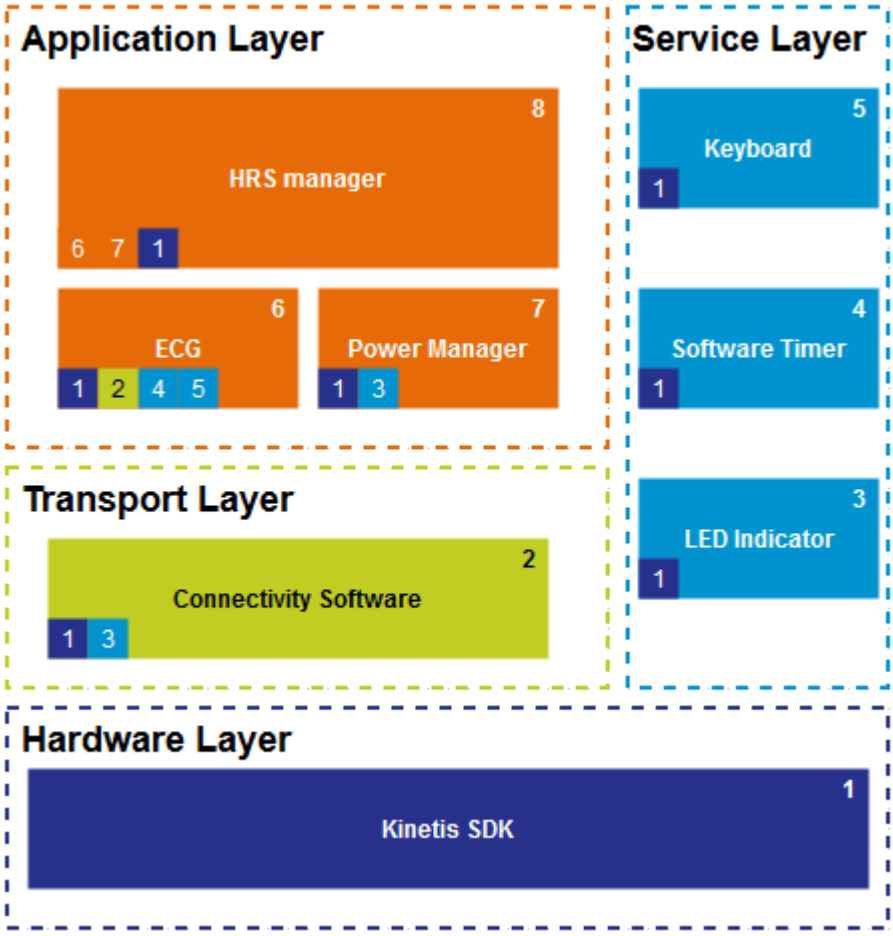
1.1 Introduction

The BLE Heart Rate Sensor is a portable device operated by a rechargeable battery. It includes the proper circuitry to obtain and process ECG signals for heart rate calculation. A SoC that includes a M0+ core microcontroller and a BLE radio acquires, processes and reports the gathered information via BLE to an enabled smartphone or device.



1.2 Software Architecture

The software architecture contemplates four main layers; Application, Service, Transport and Hardware. API documentation is organized in groups accordingly with this architecture. A graphic description is shown below.



1.3 Reference documents

1.4 Revision history

Version	Date	Updates
0	05/2015	Initial release.

Chapter 2
Application Layer

2.1 Overview

The Application Layer comprises the files and functions that define the Heart Rate Monitor application behaviour. This layer is divided in three main modules; The Heart Rate Sensor Manager which controls the overall behaviour for the application across all the states. The ECG Application which takes care of the ECG acquisition, processing and heart rate calculation. And the Power Manager which controls the power consumption for all the states in the application.

Modules

- ECG Application
- Heart Rate Sensor Manager
- Power Manager

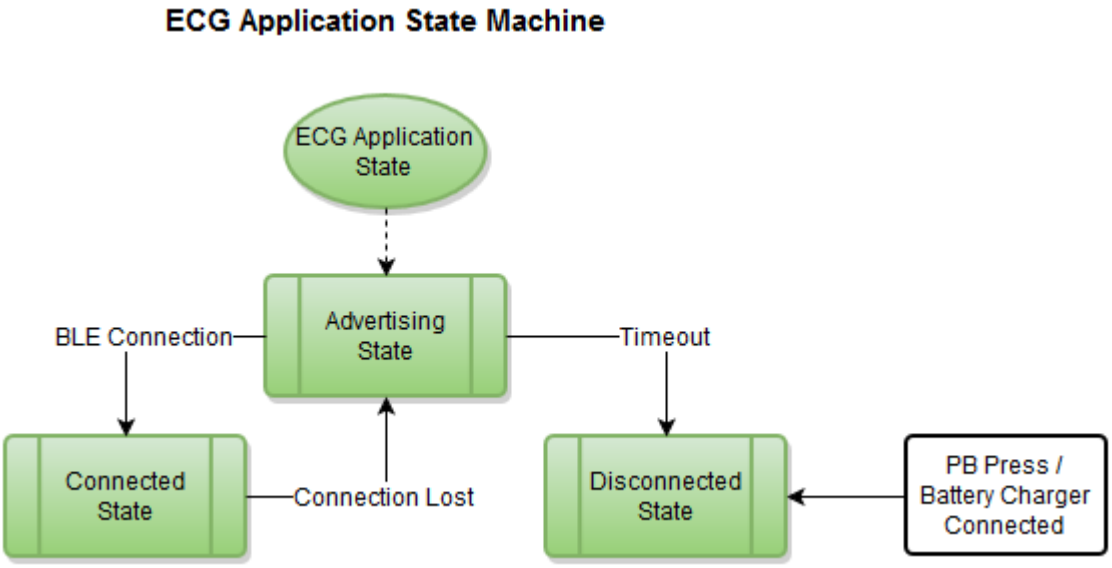
2.2 ECG Application

2.2.1 Overview

The ECG Application module encloses the files and functions that allow the acquisition, processing and report of the electrocardiograph (ECG) signal in the application. It manages the state machines that execute the heart rate acquisition process which consists in the following steps:

1. Digitalize the ECG signal using the ADC.
2. Perform digital filtering to the signal.
3. Calculate the heart rate value.
4. Report heart rate measurements.

Following diagram illustrates the functionality of the ECG application.



Functional Description

The ECG Application module bases its functionality in the different connection states for a Bluetooth Low Energy connection. Depending on the connection state, a specific action is taken.

During the Advertising State, the Transport (see Transport Layer) is set to report the presence of the device. All the ECG processes are stopped at this time and the device waits for a connection.

During the Connected State, communications have been stablished with an enabled smartphone. The ECG circuitry and algorithms are enabled and the device starts reporting the obtained ECG measurements.

After the connection is lost, eiher because the application button is pressed, or the device times out during the Advertising state without stablishing a connection, the application enters in Disconnected State. During this state, all the ECG acquisition processes are disabled, communications terminated, and the device is prepared to enter in a deep low power mode.

Modules

- ECG Acquisition
- Heart Rate Analysis

Classes

- struct ecg_machine_states_t

Enumerations

- enum ecg_states_t {
  kAdvertising,
  kConnected,
  kDisconnected }

Functions

- void ecg_application (void)

Variables

- static uint8_t kStartAdvFlag
- static uint8_t reAdvertising
- uint8_t gTimeOut
- uint8_t reConnect
- uint8_t gDesconnectAdv

2.2.2 Class Documentation

2.2.2.1 struct ecg_machine_states_t

State Machine possible states !

Class Members

uint8_t	PrevState	Previous state.
uint8_t	ActualState	State that the StateMachine driver uses like the next state.
uint8_t	NextState	Use to suggest possible Actual State.
uint8_t	ResumeState	Resume to a particular state.
uint8_t	TimeoutState	Flag to indicate that a timeout occur on the state machine.

2.2.3 Enumeration Type Documentation

2.2.3.1 enum ecg_states_t

enumeration of the possible states NOTE the states shall be aligned with Pointer_SM function pointer array.

Enumerator

kAdvertising Starts SourceActivityTimer.

kConnected Sends VCONN_Swap message and starts SenderResponeTimer.
kDisconnected Policy Engine (PE) starts the VconnOnTimer.

2.2.4 Function Documentation

2.2.4.1 void ecg_application (void)

Executes the ECG acquisition application.

Parameters

in	*none*	

Returns

none

2.2.5 Variable Documentation

2.2.5.1 uint8_t kStartAdvFlag [static]

This variable is a flag that starts the advertising.

2.2.5.2 uint8_t reAdvertising [static]

This variable is used to reactivate the advertising when the previous state was "connected".

This variable is only used when the low power mode is disabled

2.2.5.3 uint8_t gTimeOut

This variable sends the "ECG state machine" to the "disconnect" state when the timer to send advertising is over.

2.2.5.4 uint8_t reConnect

This variable sends the "ECG state machine" to the "advertising" state when the previous state was disconnected.

This variable is only used when the low power mode is disabled

2.2.5.5 uint8_t gDesconnectAdv

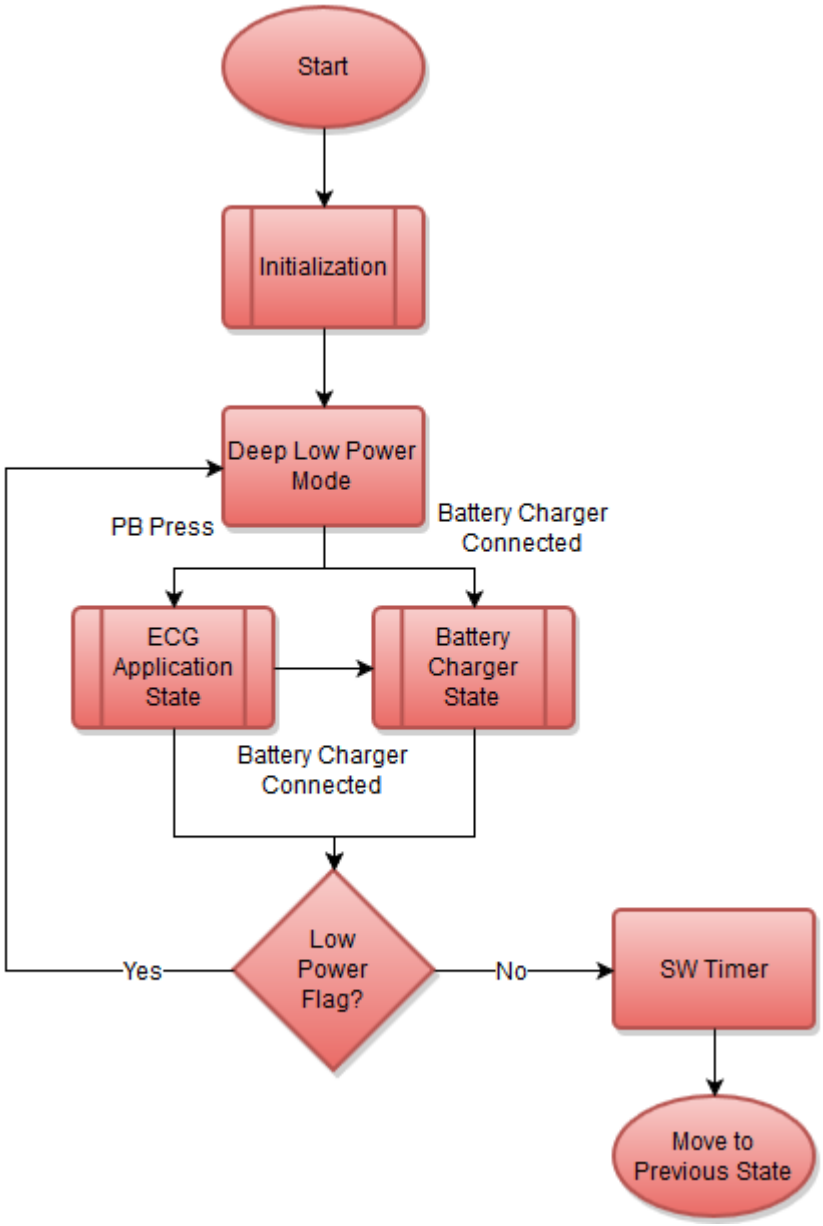
This variable sends the "ECG state machine" to the "disconnected" state when the previous state was "advertising".

This variable is only used when the low power mode is disabled

2.3 Heart Rate Sensor Manager

2.3.1 Overview

The Heart Rate Sensor Manager module controls the behavior for the complete heart rate monitor application. It takes care of calling the appropriate state machines for services, heart rate acquisition and power management. The software state diagram for this module is shown below.



Functional Description

During the initialization process, the application configures all the necessary hardware for the proper application functionality. Right after the initialization, the application enters in a deep low power mode for battery retention.

Only two events can exit the application for the deep low power mode; pressing the application button, which enters the device in the ECG application state (See ECG Application), or connecting the battery charger, which enters the device in battery charging state (see Power Manager).

After the proper state machine flow has been completed, the heart rate sensor manager determines if the device can enter in a deep low power state. Otherwise, it executes the software timer service function and

returns to the previous state machine.

Classes

- struct sSM

Enumerations

- enum pd_vcs_dfp_states_t {
  kLowPowerState,
  kEcgState,
  kBatteryChargerState }

Functions

- void hrs_manager (void)
- static void hrsLowPowerState (void)
- static void hrsEcgAcquisitionState (void)
- static void hrsBatteryChargerState (void)

Variables

- sSM gHrsManagerStateMachine

2.3.2 Class Documentation

2.3.2.1 struct sSM

State Machine possible states.

Class Members

uint8_t	PrevState	Previous state.
uint8_t	ActualState	Current execution state.
uint8_t	NextState	Next execution state.
uint8_t	ResumeState	State to enter in case of exception.
uint8_t	TimeoutState	State to execute on timeout.

2.3.3 Enumeration Type Documentation

2.3.3.1 enum pd_vcs_dfp_states_t

enumeration of the possible states NOTE the states shall be aligned with pe_vconn_swap_dfp function pointer array.

Enumerator

kLowPowerState Executes the application's low power functionality.
kEcgState Executes the ECG acquisition system.
kBatteryChargerState Handles all the battery charger mechanisms.

2.3.4 Function Documentation

2.3.4.1 void hrs_manager (void)

Executes the Heart Rate Sensor main state machine.

Parameters

in	*none*	

Returns

void

2.3.4.2 static void hrsLowPowerState (void) [static]

Executes the application's low power functionality.

Parameters

in	*none*	

Returns

void

2.3.4.3 static void hrsEcgAcquisitionState (void) [static]

Executes the ECG acquisition system.

Parameters

in	*none*	

Returns

void

2.3.4.4 static void hrsBatteryChargerState (void) [static]

Handles all the battery charger mechanisms.

Parameters

in	*none*	

Returns

void

2.3.5 Variable Documentation

2.3.5.1 sSM gHrsManagerStateMachine

Heart Rate Sensor manager state machine.

2.4 Power Manager

2.4.1 Overview

This module includes all the files and functions to manage the power consumption in the device, enable/disable power in different hardware sections and supervise the battery level and charging.

Macros

- #define BATTERY_MEASUREMENT_ADC_INSTANCE
- #define BATTERY_MEASUREMENT_ADC_RESOLUTION
- #define BATTERY_MEASUREMENT_PERIOD_MS
- #define BATTERY_MEASUREMENT_MAX_VOLTAGE_MV
- #define BATTERY_MEASUREMENT_MIN_VOLTAGE_MV
- #define BATTERY_MEASUREMENT_CORRELATION_SLOPE

Enumerations

- enum power_manager_batt_meas_error_t { kPowerManagerBattMeasError }

Functions

- void power_manager (void)
- void power_manager_enter_low_power (void)
- void power_manager_init (void)
- void power_manager_battery_level_timer_callback (void)
- void power_manager_ppr_handler (void)

Variables

- uint8_t gpowerManagerCurrentBatteryLevel
- uint8_t batteryMeasurementTimerId

2.4.2 Macro Definition Documentation

2.4.2.1 #define BATTERY_MEASUREMENT_ADC_INSTANCE

Battery measurement ADC.

2.4.2.2 #define BATTERY_MEASUREMENT_ADC_RESOLUTION

ADC resolution used for battery measurement.

2.4.2.3 #define BATTERY_MEASUREMENT_PERIOD_MS

Period in ms for the battery measurement execution.

2.4.2.4 #define BATTERY_MEASUREMENT_MAX_VOLTAGE_MV

Maximum voltage in mV of the battery.

When the battery reaches this voltage, it reports 100% of capacity

2.4.2.5 #define BATTERY_MEASUREMENT_MIN_VOLTAGE_MV

Minimum voltage in mV of the battery.

When the battery reaches this voltage, it reports 0% of capacity

2.4.2.6 #define BATTERY_MEASUREMENT_CORRELATION_SLOPE

Correlation slope for battery calculation.

2.4.3 Enumeration Type Documentation

2.4.3.1 enum power_manager_batt_meas_error_t

Possible errors during battery measurement.

Enumerator

kPowerManagerBattMeasError Battery measurement error.

2.4.4 Function Documentation

2.4.4.1 void power_manager (void)

Power Manager main application.

This is called from the Heart Rate Sensor Manager

Parameters

in	*None*	

Returns

void

2.4.4.2 void power_manager_enter_low_power (void)

This function prepares the SoC to enter in low power mode.

Parameters

in	*none*	

Returns

void

2.4.4.3 void power_manager_init (void)

Initialize the power manager module.

Parameters

in	*none*	

Returns

void

2.4.4.4 void power_manager_battery_level_timer_callback (void)

This function is executed as a callback for the battery measurement timer.

It reinitializes the timer and reports the current battery level

Parameters

in	*none*	

Returns

void

2.4.4.5 void power_manager_ppr_handler (void)

PPR pin ISR.

This interrupt indicates that a power source has been connected to the battery charger.

Parameters

in	*none*	

Returns

2.4.5 Variable Documentation

2.4.5.1 uint8_t gpowerManagerCurrentBatteryLevel

Current battery level percentage.

2.4.5.2 uint8_t batteryMeasurementTimerId

Timer ID for battery measurement task.

Chapter 3
Hardware Layer

The Hardware Layer includes all the functions necessary to interact with the microcontrollers modules. Access to modules is performed through the Freescale Kinetis SDK APIs. All the documentation describing the hardware layer functionality can be found in the Kinetis SDK documentation.

Chapter 4
Service Layer

4.1 Overview

The Service Layer includes functions that enable other layers to perform some specific functionalities like setting a time base or receiving an input. The Service Layer is divided in three modules.

1. Software Timer: Stablishes a time base for the execution of time-dependent functions.
2. LED Indicator: Provides functions for the control of LED indicators.
3. Keyboard: Provides functions for the management of input methods.

Modules

- Keyboard
- Software Timer
- Status Indicator

4.2 Keyboard

The Keyboard module provides functions for the management of input methods for the user. It includes functionality to handle push button and TSI inputs, detect different pressing methods and report to the upper layers any action performed by the user.

The Keyboard module is reused from the Freescale BLE stack. For documentation on this module please refer to the Freescale BLE stack documentation.

Chapter 5
Transport Layer

The Transport Layer includes all the functions and characteristics that allows the Bluetooth Low Energy communications between the Heart Rate Sensor and an enabled smartphone. The Transport Layer is based on the Freescale Bluetooth Low Energy (BLE) stack. For details on the Transport Layer please review the Freescale BLE stack documentation.

5.0.1 ECG Acquisition

5.0.1.1 Overview

This module contains the required functions to acquire and process the ECG signal.

Macros

- #define EKG_TASK_TIME_MS
- #define ECG_ACQUISITION_ADC_INSTANCE

Enumerations

- enum ecg_acquisition_init_status_t {
  ecgAcquisitionInitOk,
  ecgAcquisitionInitError }

Functions

- uint8_t ecg_acquisition_init (void)
- void ecg_acquisition (void)

Variables

- int16_t i16EkgSample
- uint8_t gEcgAcquisitionTimerId

5.0.1.2 Macro Definition Documentation

5.0.1.2.1 #define EKG_TASK_TIME_MS

Time in ms betwen ECG acquisition task executions.

5.0.1.2.2 #define ECG_ACQUISITION_ADC_INSTANCE

ADC instance used for ecg acquisition.

5.0.1.3 Enumeration Type Documentation

5.0.1.3.1 enum ecg_acquisition_init_status_t

Enumerator

ecgAcquisitionInitOk ECG Acquisition system initialized correctly.
ecgAcquisitionInitError ECG Acquisition system initialization errors.

5.0.1.4 Function Documentation

5.0.1.4.1 uint8_t ecg_acquisition_init (void)

This function initializes the ECG acquisition system.

Parameters

in	*none*	

Returns

ECG acquisition initialization status (see ecg_acquisition_init_status_t)

5.0.1.4.2 void ecg_acquisition (void)

This function calls the necessary mechanisms for ECG acquisition.

Parameters

in	*none*	

Returns

none

5.0.1.5 Variable Documentation

5.0.1.5.1 int16_t i16EkgSample

ADC result for ECG signal.

5.0.1.5.2 uint8_t gEcgAcquisitionTimerId

Timer ID for ECG acquisition sequence timer.

5.0.2 Heart Rate Analysis

5.0.2.1 Overview

This module contains the required functions to obtain the heart rate value based on the acquired ECG signal.

Macros

- #define HR_SAMPLING_PERIOD_MS
- #define HR_TIMEOUT_MS
- #define HR_SIGNAL_AMPLITUDE_THRESHOLD
- #define HR_VALUE_LIMIT
- #define HR_AVERAGE

Enumerations

- enum HR_DETECTOR_STATES {
  HR_FIND_MAX,
  HR_FIND_MIN,
  HR_QRS_DETECTED }

Functions

- void heart_rate_analysis (int16_t i16EcgSample)

Variables

- uint8_t gu8HrValue

5.0.2.2 Macro Definition Documentation

5.0.2.2.1 #define HR_SAMPLING_PERIOD_MS

ECG ADC sampling period in ms.

5.0.2.2.2 #define HR_TIMEOUT_MS

Timeout time in ms.

If a heartbeat is not detected during this time, the HR count goes to zero

5.0.2.2.3 #define HR_SIGNAL_AMPLITUDE_THRESHOLD

Minimum amplitude (in ADC counts) to consider a slope a QRS complex.

5.0.2.2.4 #define HR_VALUE_LIMIT

Maximum HR value that can be reported.

Any value higher than this is considered noise and set to zero

5.0.2.2.5 #define HR_AVERAGE

Number of HR samples to average before reporting a value.

5.0.2.3 Enumeration Type Documentation

5.0.2.3.1 enum HR_DETECTOR_STATES

Heart Rate detection state machine states.

Enumerator

HR_FIND_MAX Find maximum peak.
HR_FIND_MIN Find minumum peak.
HR_QRS_DETECTED QRS complex detected.

5.0.2.4 Function Documentation

5.0.2.4.1 void heart_rate_analysis (int16_t *i16EcgSample*)

Analizes heart rate.

Parameters

in	*int16_t*	ECG acquired sample.

Returns

none

5.0.2.5 Variable Documentation

5.0.2.5.1 uint8_t gu8HrValue

Heart rate value.

5.1 Software Timer

5.1.1 Overview

This module handles an array of software timers and trigger the timer event when the timer has elapsed.

Classes

- struct SwTimerObj_t
- struct SwCounter_t

Macros

- #define MAX_TIMER_OBJECTS
- #define MAX_COUNTER_OBJECTS
- #define HW_TIMER_DECREMENT_VALUE_MS
- #define INACTIVE_TIMER
- #define INVALID_TIMER_ID

Typedefs

- typedef uint8_t **SwTimerId_t**

Functions

- void SwTimer_Init (void)
- void SwTimer_PeriodicTask (void)
- void SwTimer_StartTimer (uint8_t timerId, uint16_t tickPeriod_ms)
- void SwTimer_StopTimer (uint8_t timerId)
- uint8_t SwTimer_CreateTimer (pFunc_t callBackFunc)
- uint8_t SwTimer_CreateCounter (void)
- void SwTimer_StartCounter (uint8_t counterId)
- void SwTimer_StopCounter (uint8_t counterId)
- uint16_t SwTimer_ReadCounter (uint8_t counterId)

Variables

- uint8_t advertisingTimerId

5.1.2 Class Documentation

5.1.2.1 struct SwTimerObj_t

Structure to define a timer object.

Class Members

uint16_t	timerCount	Current timer count.
pFunc_t	timerEvent	Event to execute on timeout.

5.1.2.2 struct SwCounter_t

Structure to define a counter object.

Class Members

uint16_t	timerCount	Current count value for counter.

5.1.3 Macro Definition Documentation

5.1.3.1 #define MAX_TIMER_OBJECTS

Maximum number of timers that the application can have.

5.1.3.2 #define MAX_COUNTER_OBJECTS

Maximum number of counters that the application can have.

5.1.3.3 #define HW_TIMER_DECREMENT_VALUE_MS

Time in ms to decrement on every hardware timer trigger (HW timer period)

5.1.3.4 #define INACTIVE_TIMER

Indicates an inactive timer.

5.1.3.5 #define INVALID_TIMER_ID

Indicates an error while creating a timer.

5.1.4 Function Documentation

5.1.4.1 void SwTimer_Init (void)

Initializes SwTimer module.

Disables all timers.

Parameters

in	*none*	

Returns

none

5.1.4.2 void SwTimer_PeriodicTask (void)

This function must be called periodically in the main loop.

It executes the software timer main functionality.

Parameters

in	*none*	

Returns

none

5.1.4.3 void SwTimer_StartTimer (uint8_t *timerId*, uint16_t *tickPeriod_ms*)

Starts a timer with a given period.

Parameters

in	*timerId*	Number of the timer to start
in	*tickPeriod_ms*	Timer period in ms

Returns

none

5.1.4.4 void SwTimer_StopTimer (uint8_t *timerId*)

Stops a timer.

Parameters

in	*timerId*	Number of the timer to stop

Returns

none

5.1.4.5 uint8_t SwTimer_CreateTimer (pFunc_t *callBackFunc*)

Creates a timer and assigns it call-back function.

Parameters

in	*callBackFunc*	Function to be executed when timer has elapsed

Returns

timerId The ID of the timer that was created. It returns INVALID_TIMER_ID (0xFF) if the timer was not created (because MAX_TIMER_OBJECTS was reached)

5.1.4.6 uint8_t SwTimer_CreateCounter (void)

Creates a counter.

Parameters

in	*none*	

Returns

The counter ID. This ID will be used to start, stop and read the counter. Returns INVALID_TIMER_ID (0xFF= if the counter was not created due to memory limitations (because MAX_TIMER_OBJECTS was reached)

5.1.4.7 void SwTimer_StartCounter (uint8_t *counterId*)

Starts a counter.

Parameters

in	*counterId*	Id of the counter to start.

Returns

none

5.1.4.8 void SwTimer_StopCounter (uint8_t *counterId*)

Stops a counter.

Parameters

in	*counterId*	Id of the counter to stop.

Returns

none

5.1.4.9 uint16_t SwTimer_ReadCounter (uint8_t *counterId*)

Reads a counter.

Parameters

in	*counterId*	Id of the counter to read.

Returns

uint16_t Current counter value.

5.1.5 Variable Documentation

5.1.5.1 uint8_t advertisingTimerId

Variable that is used to save the ID of the timer used to disconnect the device when the time is out.

5.2 Status Indicator

5.2.1 Overview

This module provides functions for the control of LED status indicators.

Macros

- #define STATUS_INDICATOR_TPM_INSTANCE
- #define STATUS_INDICATOR_TPM_CHANNEL
- #define STATUS_INDICATOR_FLASHER_PERIOD_MS
- #define STATUS_INDICATOR_FADER_PERIOD_S

Enumerations

- enum status_indicator_error_t {
  kStatusIndicatorErrorOk,
  kStatusIndicatorErrorFlasherBusy,
  kStatusIndicatorErrorTimerInitializationError }
- enum status_indicator_flasher_t {
  kStatusIndicatorFlasherFree,
  kStatusIndicatorFlasherBusy }

Functions

- void status_indicator_fade_init (void)
- void status_indicator_fade_led (void)
- void status_indicator_fade_off (void)
- status_indicator_error_t status_indicator_flash_led (uint32_t pinName)

5.2.2 Macro Definition Documentation

5.2.2.1 #define STATUS_INDICATOR_TPM_INSTANCE

TPM hardware instance for PWM control.

5.2.2.2 #define STATUS_INDICATOR_TPM_CHANNEL

TPM hardware channel for PWM control.

5.2.2.3 #define STATUS_INDICATOR_FLASHER_PERIOD_MS

Period in ms to remain the LED on in every flashing.

5.2.2.4 #define STATUS_INDICATOR_FADER_PERIOD_S

Period in seconds for LED fading.

5.2.3 Enumeration Type Documentation

5.2.3.1 enum status_indicator_error_t

Possible errors for status indicator functions.

Enumerator

kStatusIndicatorErrorOk No error occurred.
kStatusIndicatorErrorFlasherBusy Flasher is busy and cannot be used.
kStatusIndicatorErrorTimerInitializationError Error initializing timer hardware.

5.2.3.2 enum status_indicator_flasher_t

Current status for the indicator flasher.

Enumerator

kStatusIndicatorFlasherFree Flasher is free for use.
kStatusIndicatorFlasherBusy Flasher is currently bussy.

5.2.4 Function Documentation

5.2.4.1 void status_indicator_fade_init (void)

This function initializes the LED fade functionality.

it must be executed before status_indicator_fade_led is called

Parameters

in	*none*	

Returns

void

5.2.4.2 void status_indicator_fade_led (void)

This functions starts fading an LED.

Parameters

in	*none*	

Returns

void

5.2.4.3 void status_indicator_fade_off (void)

This function turns off a LED that is already fading.

Parameters

in	*none*	

Returns

void

5.2.4.4 status_indicator_error_t status_indicator_flash_led (uint32_t *pinName*)

This function starts a flash indication on a selected LED.

Flash time is defined by STATUS_INDICATOR_FLASHER_PERIOD_MS.

Parameters

in	*pinName*	Name of the pin connected to the LED to flash. Pin must be defined using the macros in the KSDK GPIO driver.

Returns

Code indicating initialization status. See status_indicator_error_t for possible error values.

Chapter 6
File Documentation

6.0.5 ecg_acquisition.h File Reference

```
#include "SSD_Types.h"
#include "fsl_adc16_driver.h"
#include <stdint.h>
```

Macros

- #define EKG_TASK_TIME_MS
- #define ECG_ACQUISITION_ADC_INSTANCE

Enumerations

- enum ecg_acquisition_init_status_t {
  ecgAcquisitionInitOk,
  ecgAcquisitionInitError }

Functions

- uint8_t ecg_acquisition_init (void)
- void ecg_acquisition (void)

Variables

- int16_t i16EkgSample
- uint8_t gEcgAcquisitionTimerId

6.0.5.1 Detailed Description

ECG acquisition functions.

Author

Version

1.0

Date

Sep-11-2015

6.0.6 ecg_application.h File Reference

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

Classes

- struct ecg_machine_states_t

Enumerations

- enum ecg_states_t {
  kAdvertising,
  kConnected,
  kDisconnected }

Functions

- void ecg_application (void)

Variables

- static uint8_t kStartAdvFlag
- static uint8_t reAdvertising
- uint8_t gTimeOut
- uint8_t reConnect
- uint8_t gDesconnectAdv

6.0.6.1 Detailed Description

This file contains functions to acquire and process the ECG signal and determine the heart rate value.

Author

Atzel Collazo

Version

1.0

Date

APR-24-2015

6.0.7 fsl_types.h File Reference

Macros

- #define **FALSE**
- #define **TRUE**
- #define **NULL**
- #define **ON**
- #define **OFF**
- #define **EVENT**(gu8Status, bit)
- #define **COMPARE**(gu8Status, bit)
- #define **CLEAR**(gu8Status, bit)

Typedefs

- typedef void(∗ **pFunc_t**)(void)

6.0.7.1 Detailed Description

Freescale types definitions.

Author

Version

0.0

Date

Apr-12-2013

6.0.8 hr_analysis.h File Reference

```
#include "SSD_Types.h"
```

Macros

- #define HR_SAMPLING_PERIOD_MS
- #define HR_TIMEOUT_MS
- #define HR_SIGNAL_AMPLITUDE_THRESHOLD
- #define HR_VALUE_LIMIT
- #define HR_AVERAGE

Enumerations

- enum HR_DETECTOR_STATES {
  HR_FIND_MAX,
  HR_FIND_MIN,
  HR_QRS_DETECTED }

Functions

- void heart_rate_analysis (int16_t i16EcgSample)

Variables

- uint8_t gu8HrValue

6.0.8.1 Detailed Description

HR analysis functions.

Author

Version

1.0

Date

Sep-09-2013

6.0.9 hrs_manager.h File Reference

```
#include "SSD_Types.h"
```

Classes

- struct sSM

Enumerations

- enum pd_vcs_dfp_states_t {
  kLowPowerState,
  kEcgState,
  kBatteryChargerState }

Functions

- void hrs_manager (void)
- static void hrsLowPowerState (void)
- static void hrsEcgAcquisitionState (void)
- static void hrsBatteryChargerState (void)

Variables

- sSM gHrsManagerStateMachine

6.0.9.1 Detailed Description

This file contains functions to handle the heart rate sensor main state machine.

Author

Ricardo Olivares

Version

1.0

Date

APR-24-2015

6.0.10 power_manager.h File Reference

```
#include "SSD_Types.h"
#include "gpio_pins.h"
#include "fsl_gpio_driver.h"
```

Macros

- #define BATTERY_MEASUREMENT_ADC_INSTANCE
- #define BATTERY_MEASUREMENT_ADC_RESOLUTION
- #define BATTERY_MEASUREMENT_PERIOD_MS
- #define BATTERY_MEASUREMENT_MAX_VOLTAGE_MV
- #define BATTERY_MEASUREMENT_MIN_VOLTAGE_MV
- #define BATTERY_MEASUREMENT_CORRELATION_SLOPE

Enumerations

- enum power_manager_batt_meas_error_t { kPowerManagerBattMeasError }

Functions

- void power_manager (void)
- void power_manager_enter_low_power (void)
- void power_manager_init (void)
- void power_manager_battery_level_timer_callback (void)
- void power_manager_ppr_handler (void)

Variables

- uint8_t gpowerManagerCurrentBatteryLevel
- uint8_t batteryMeasurementTimerId

6.0.10.1 Detailed Description

This file contains functions to manage the power features for the reference design.

Author

Ricardo Olivares

Version

1.0

Date

APR-24-2015

6.0.11 software_timer.h File Reference

```
#include "ecg_acquisition.h"
#include "fsl_types.h"
#include "SSD_Types.h"
#include "fsl_lptmr_driver.h"
#include "fsl_adc16_driver.h"
```

Classes

- struct SwTimerObj_t
- struct SwCounter_t

Macros

- #define MAX_TIMER_OBJECTS
- #define MAX_COUNTER_OBJECTS
- #define HW_TIMER_DECREMENT_VALUE_MS
- #define INACTIVE_TIMER
- #define INVALID_TIMER_ID

Typedefs

- typedef uint8_t **SwTimerId_t**

Functions

- void SwTimer_Init (void)
- void SwTimer_PeriodicTask (void)
- void SwTimer_StartTimer (uint8_t timerId, uint16_t tickPeriod_ms)
- void SwTimer_StopTimer (uint8_t timerId)
- uint8_t SwTimer_CreateTimer (pFunc_t callBackFunc)
- uint8_t SwTimer_CreateCounter (void)
- void SwTimer_StartCounter (uint8_t counterId)
- void SwTimer_StopCounter (uint8_t counterId)
- uint16_t SwTimer_ReadCounter (uint8_t counterId)

Variables

- uint8_t advertisingTimerId

6.0.11.1 Detailed Description

This file handles an array of software timers and trigger the timer event when the timer has elapsed.

Author

Samuel Quiroz

Version

1.0

Date

SEP-11-2009

6.0.12 status_indicator.h File Reference

```
#include "fsl_tpm_hal.h"
#include "fsl_tpm_driver.h"
#include "TimersManager.h"
```

Macros

- #define STATUS_INDICATOR_TPM_INSTANCE
- #define STATUS_INDICATOR_TPM_CHANNEL
- #define STATUS_INDICATOR_FLASHER_PERIOD_MS
- #define STATUS_INDICATOR_FADER_PERIOD_S

Enumerations

- enum status_indicator_error_t {
  kStatusIndicatorErrorOk,
  kStatusIndicatorErrorFlasherBusy,
  kStatusIndicatorErrorTimerInitializationError }
- enum status_indicator_flasher_t {
  kStatusIndicatorFlasherFree,
  kStatusIndicatorFlasherBusy }

Functions

- void status_indicator_fade_init (void)
- void status_indicator_fade_led (void)
- void status_indicator_fade_off (void)
- status_indicator_error_t status_indicator_flash_led (uint32_t pinName)

6.0.12.1 Detailed Description

This module contains functions to handle the different application indications.

Author

Atzel Collazo

Version

1.0

Date

APR-24-2015

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: freescale.com/SalesTermsandConditions.

freescale™