The Architecture of SW

WISOL

August 25, 2016

Index of Contents

Introduction

Device Model & Firmware Version

Model	Firmware
SFM20R	EVBSFM20R_V200

The architecture of SW documentation includes descriptions to help you understand and develop SW in the module. Scenario mode and test mode are provided for development purposes only and should always be tested with your design.

Mode

Introduction

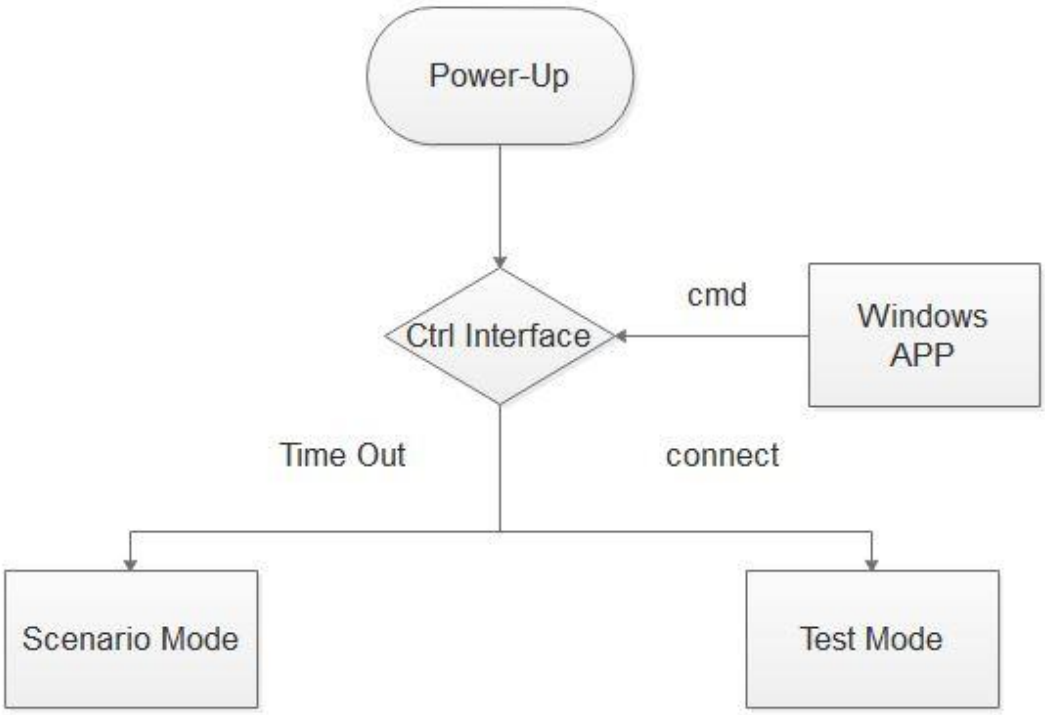
Examples are consist of two modes.

Scenario mode is based on Software Marketing Specification by SIGFOX.

Test mode enables user to test and debug each modules like a GPS module, a WIFI module, and so on.

Selecting Mode

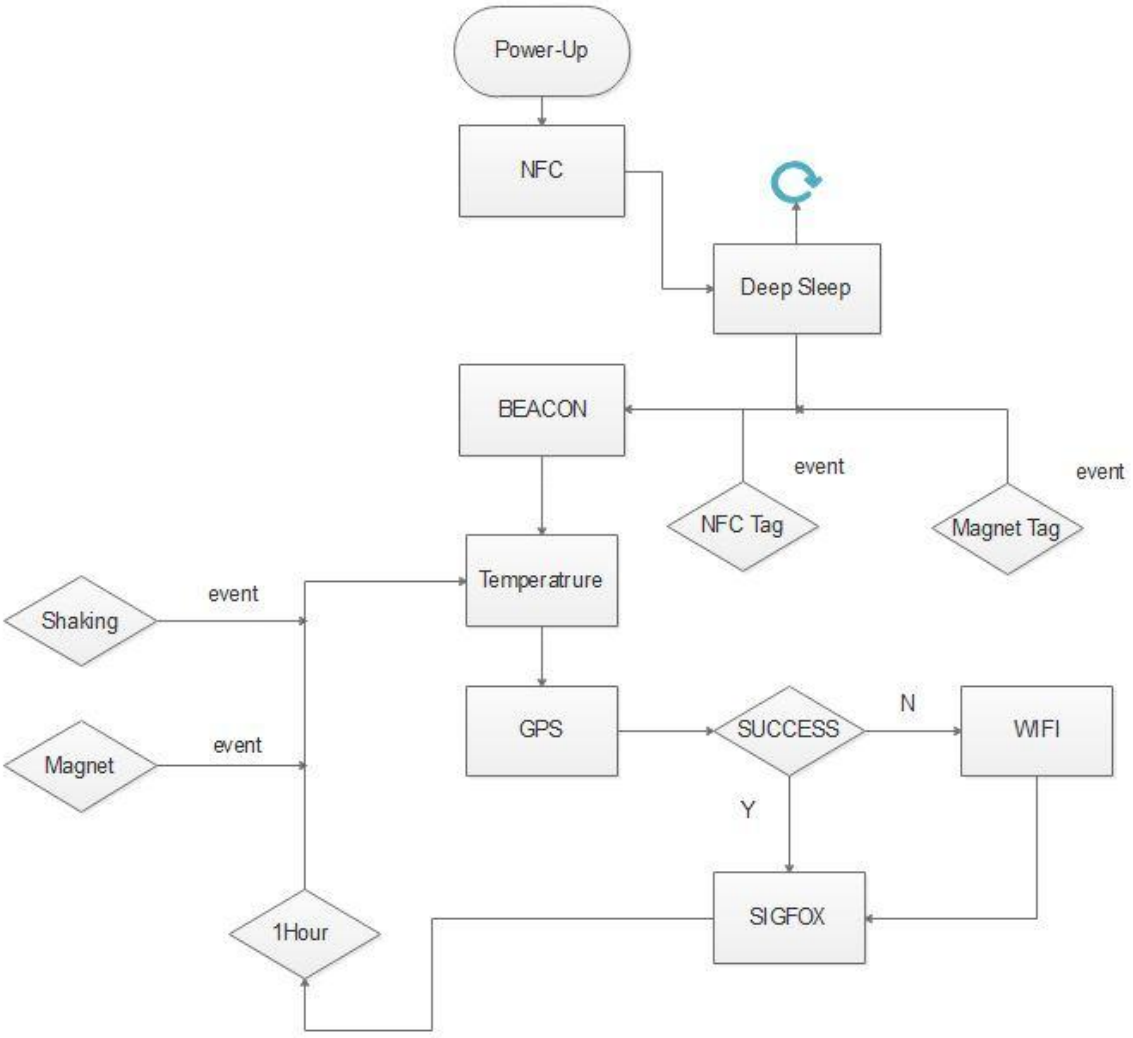
- Scenario mode is automatically selected in power-up.
- Test mode needs pc tool and connecting to the used usb port.



Scenario

Introduction

The target application for the scenario mode is a low power tracking device.
The application use WIFI or GPS to determine location. It will then transmit the location information via SIGFOX. It also will transmit other information like temperature, accelerometer, and so on.



Wake-up Device

- Touching the NFC antenna with a smartphone or a tablet can wake up device.
- Touching magnetic sensor with magnetic tag can also wake up it.

Bluetooth

- The default payload for the beacon advertisement will be the Bluetooth address and SIGFOX ID.
- In Power-up or touching the NFC antenna with a smartphone, application will change non-connectable advertising to connectable advertising.
- A smartphone app can connect to device via BLE service. User can download firmware via FOTA, and get the information of device.

Determining location

- The application can turn on the GPS radio and attempt to get a fix for 60s.
- The application can run a scan for access points for 10s with WIFI.

Measurement frequency

- When power-up
- Every 10 minutes(editable)
- Whenever the accelerometer triggers an interrupt
- Whenever touching magnetic sensor with magnetic tag

API

Bluetooth API

- refer to documents of Nordic SDK

SIGFOX API

- Set RCZ 1-4

  void sigfox_set_rcz(sigfox_rcz which_rcz)
  param rcz : set RCZ_1, RCZ_2, RCZ_3, or RCZ_4

- Define the SIGFOX message payload and pass downlink message

  void sigfox_send_payload(uint8_t * send_data, uint8_t * received_data)
  param[in] uint8_t* send_data : pointer to user data
  parameter[out] uint8_t *received_data : pointer to downlink buffer

- Set the power level (the output power of the radio) and save config

  bool cfg_sigfox_set_powerlevel(int level);
  param[in] int level : power level (0-15 dbm)
  retval true in success

WIFI API

- Initialize wifi driver

  int wifi_drv_init(void);
  retval CWFI_Result_OK in success

- Scan for local WIFI access points

  int start_AP_scan(void);
  retval CWFI_Result_OK in success

- Set the duration of a scan from 1s to 60s

  void set_scan_interval(int interval);
  param interval 1~60 sec

- Report basestation IDs and associated RSSI readings

  int get_AP_scanResult(uint32_t * get_cnt, uint8_t **ssid, int32_t **rssi, uint8_t **bssid);
  param[out] *get_cnt pointer to scan count
  param[out] **ssid pointer to ssid data
  param[out] **rssi pointer to rssi data
  param[out] **bssid pointer to bssid data

ACCELEROMETER API

- Initialize I2C

  void cfg_i2c_master_init(void);

- Uninitialize I2C

  void cfg_i2c_master_uninit(void);
- Take an accelerometer reading(x,y,z)

  bool cfg_bma250_read_xyz(struct bma_accel_data *accel);
  param[out] struct bma_accel_data pointer to value of x,y,z
  retval true in success

  struct bma_accel_data {
  int16_t x,/**< accel x data */
  y,/**< accel y data */
  z;/**< accel z data */
  };

- Read the value of register

  bool cfg_bma250_read_reg(uint8_t reg_addr, uint8_t * read);
  param[in] uint8_t reg_addr address of register

param[out] uint8_t * read value of register
retval true in success

- Write the value in register

  bool cfg_bma250_write_reg(uint8_t reg_addr, uint8_t reg_data);
  param[in] uint8_t reg_addr address of register
  param[in] uint8_t reg_data value to write to register

- refer to BST-BMA250E-DS004-06.pdf for knowing registers in detail.

GPS API

- Initialize GPS driver

  void gps_init(void);

- Start tracking

  int start_gps_tracking(void);
  retval 0 in success

- Report the longitude and latitude

  int get_lastLocation(char **ns, char**latitude, char ** ew, char**longitude);
  param[out] char **ns pointer to the data of cardinal point (N or S)
  param[out] char **latitude pointer to the data of latitude
  param[out] char **new pointer to the data of cardinal point (E or W)
  param[out] char **latitude pointer to the data of longitude
  retval 0 in success

TEMPERATURE API

- Read temperature

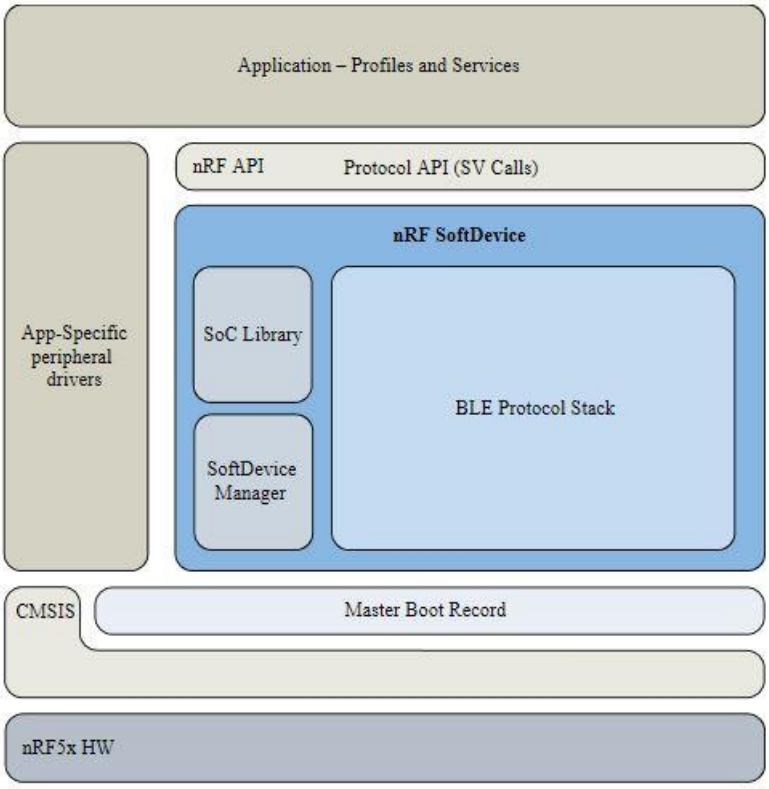
  int get_temperature(int * tmp_int, int *tmp_dec);
  param[out] int *tmp_int pointer to integer of temperature
  param[out] int *tmp_dec pointer to decimals of temperature
  retval 1 in success

- Set threashold(low, high)

  uint32_t set_alert_threshold(u16 tmp_min, u16 tmp_max);
  param[in] u16 tmp_min the low value of temperature
  param[in] u16 tmp_max the high value of temperature
  retval 0 in success

Architecture

Introduction

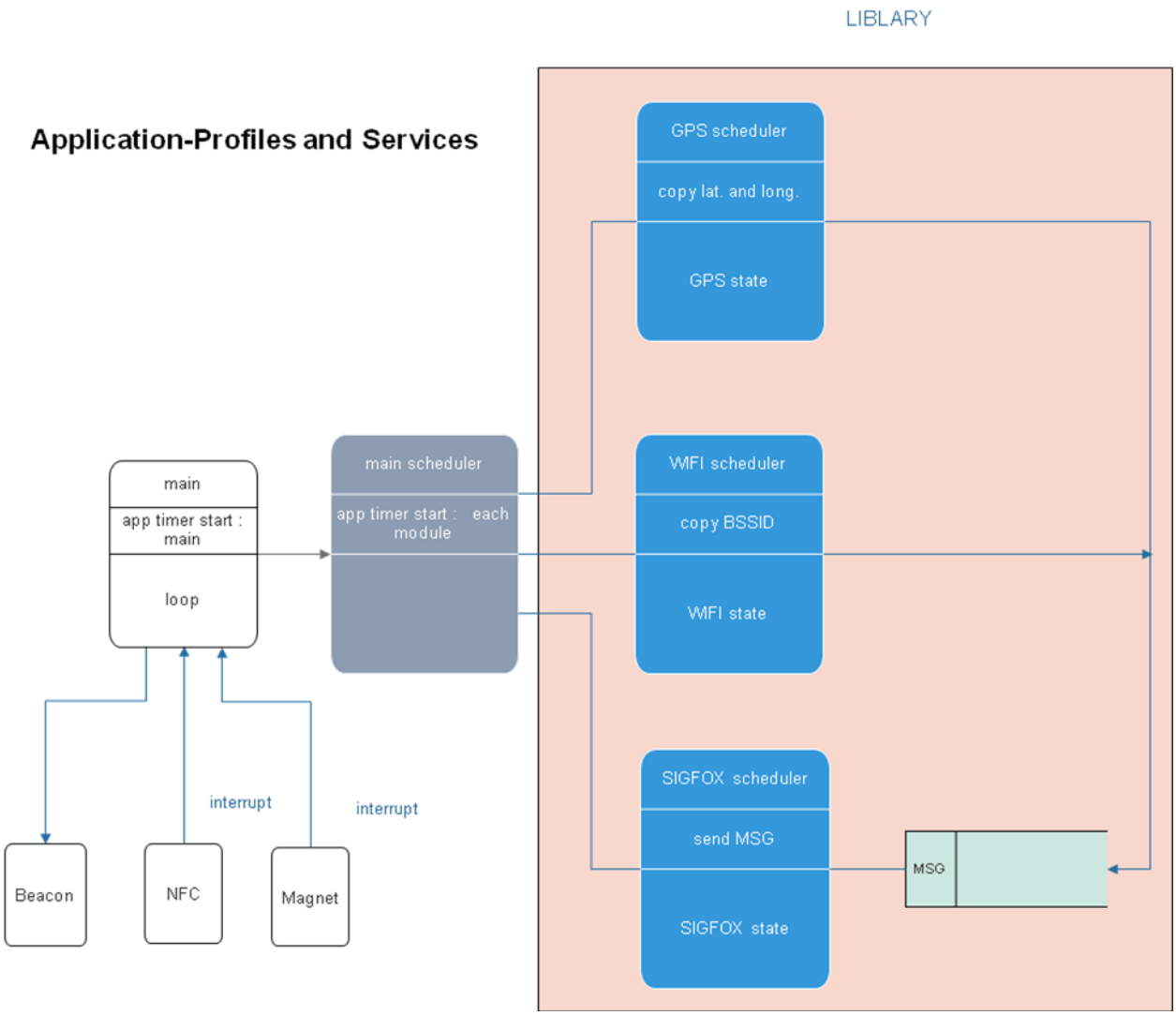


The application is based on nRF52832 platform. The application manages schedule using app timer.

The application has two kinds of scheduler, one is main scheduler and the other is module scheduler.

Application – Profiles and Services

- The tracking-application runs on "application-Profiles and Services" layer.
- The developer can only use the API of Modules like create, start, stop.
- When "start" of API is called, each module automatically will runs and get the results.
- Each module is provided in the form of a library.



Create Scheduler

- The timer library in nRF52832 platform enables the application to create multiple timer instances based on the RTC1 peripheral.
- Checking for time-outs and invoking the user time-out handlers is performed in the RTC1 interrupt handler.
- The main scheduler operates at a 20 ms cycle.
- Ex)
  err_code = app_timer_create(&main_timer_id, mode, timeout_handler)

  mode : either single shot or repeated
  timeout_handler : manage and execute each module

State machine

- Scheduler uses state machine for managing each module.
- Ex)

```
typedef enum
{
    NONE,
    WAIT_CTRL_MODE_CHANGE,
    ACC,
    MAIN_SCENARIO_LOOP,
    TMP,
    BLE,
    GPS,
    WIFI,
    WIFI_BYPASS,
    SIGFOX,
    SIGFOX_BYPASS,
    MANUAL_MODE,
    SCENARIO_MODE,
    GPS_BYPASS,
    BLE_CONTROL,
    IDLE
}module_mode_t
```

- When user time-out, scheduler determines whether to work in current state or move to another state.

Inter-Communication

- Main scheduler calls create Function to execute any module.
- After each module work, the result is copied into sending buffer in SIGFOX.
- After main scheduler checking module's work, it move to another module.

● ex) between main scheduler and SIGFOX module