Smart Lock on RV32M1-VEGA Board

1. Introduction

This document provides an example application for smart lock demo on RV32M1-VEGA board based on BLUETOOTH LE 4.2 stack shipped with RV32M1SDK. The code of demo runs on the RISC-V ZERO_RISCY in the RV32M1 device. The example application can be employed as: 1) A demonstration using the supplied pre-built binaries that can be run on nodes of the RV32M1 hardware kits. 2) A starting point for custom application development using the supplied C source files and associated project files.

This document summarizes how the demo works on RV32M1-VEGA board, needed software environment and necessary hardware kits, an introduction to the BLUETOOTH LE, and the smart lock block diagram. The document also describes the source code of the demo including its workflow and related source files. Finally, the document demonstrates how to set up the demo and control the lock through a smart phone App.

Contents

2 Smart Lock Overview

The demo implements a smart lock application that the lock can be controlled by a smart phone app using manual mode or auto mode. It comprises a RV32M1-VEGA board, the lock driven by a DC electric motor, and Bluetooth-based smart phone App.

The communication protocol between App and smart lock is BLUETOOTH LE 4.2 that's enabled in the RV32M1 SDK. The code including BLUETOOTH LE stack run on the RISC-V Zero_riscy core. The RV32M1-VEGA board receives the command from the App through BLUETOOH LE and controls two GPIOs according to the received command. Then the GPIOs drive the DC motor to lock or unlock the lock. In manual mode, the App controls the lock using the buttons of "LOCKED" and "UNLOCKED". User must operate the App to touch the "LOCKED" or "UNLOCKED" button to lock or unlock the lock. In auto mode, the App controls the lock through Received Signal Strength Indicator (RSSI), when the value is higher than a predefined value, that means the person who's holding the smart phone is close to the lock, the lock will be un-locked. When the value is smaller than a predefined value, that means the person having the smart phone is leaving the lock, the lock will be locked automatically.

To save power, after reset, the demo is in fast advertisement with advertise from 20 milliseconds to 30 milliseconds. After 30 seconds, the demo will turn in slow advertisement with advertise every 1 second

3 Development Environment

This section describes the needed software environment and necessary hardware kits to create a smart lock system.

3.1 Software Preparation

Follow the guide *Setting Up RISC-V Development Environment for RV32M1-VEGA* from www.open-isa.org website to install bellow software and tools on Windows.

- Eclipse IDE
- GNU MCU Eclipse Windows Build Tools (Optional)
- RV32M1 GNU GCC Toolchain
- OpenOCD Debugger
- RV32M1 SDK

3.2 Hardware Preparation

The following hardware kits are needed to run the example application demo:

- RV32M1-VEGA Board, which is from www.open-isa.org.
- Lock with H-bridge drive and DC motor.
- Smart phone with app installed and BLE enabled.

4 Smart Lock System

The section gives a brief introduction to the BLUETOOTH LE that's the communication protocol between smart phone app and the smart lock as well as the system block diagram of the demo.

4.1 Introduction to BLUETOOTH LE

The demo is based on Bluetooth low energy (BLUETOOTH LE) 4.2. BLUETOOTH LE is a new technology that has been designed as both a complementary technology to classic Bluetooth as well as the lowest possible power wireless technology that can be designed and built. BLUETOOTH LE stack is split into three basic parts: controller, host, and applications. The controller is typically a physical device that can transmit and receive radio signals and understand how these signals can be interpreted as packets with information within them. The host is typically a software stack that manages how two or more devices communicate with one another and how several different services can be provided at the same time over the radios. The application uses the software stack, and therefore the controller, to enable a use case.

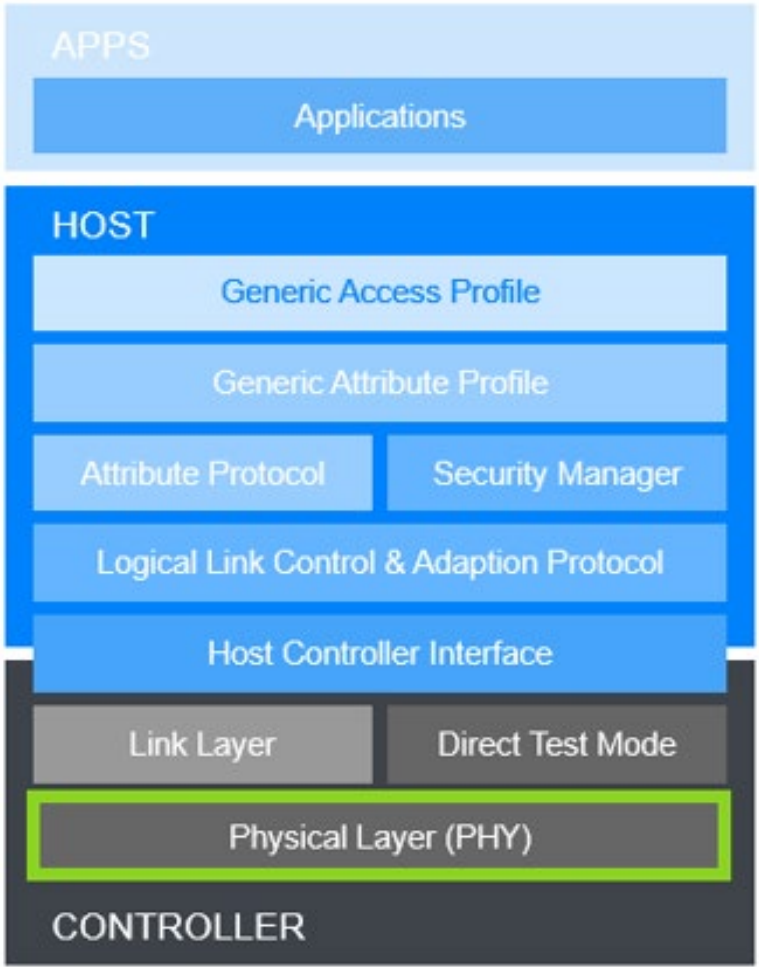


Figure 1 The BLUETOOTH LE stack

There are the Physical Layer, Link Layer, Direct Test Mode, and the lower layer of the Host Controller Interface in the controller. There are three protocols in the host: logical link control and adaptation protocol, attribute protocol, and the security manager protocol. The generic attribute profile, the generic access profile, and modes are in the host too. The BLUETOOTH LE transmits and receives the bits using the 2.4GHz radio. On the RV32M1-VEGA board, it uses a modulation scheme called Gaussian Frequency Shift Keying (GFSK) that has a symbol rate of 1 mega symbol per second (Ms/s).

4.2 System Block Diagram

The demo's system block diagram is shown as below:

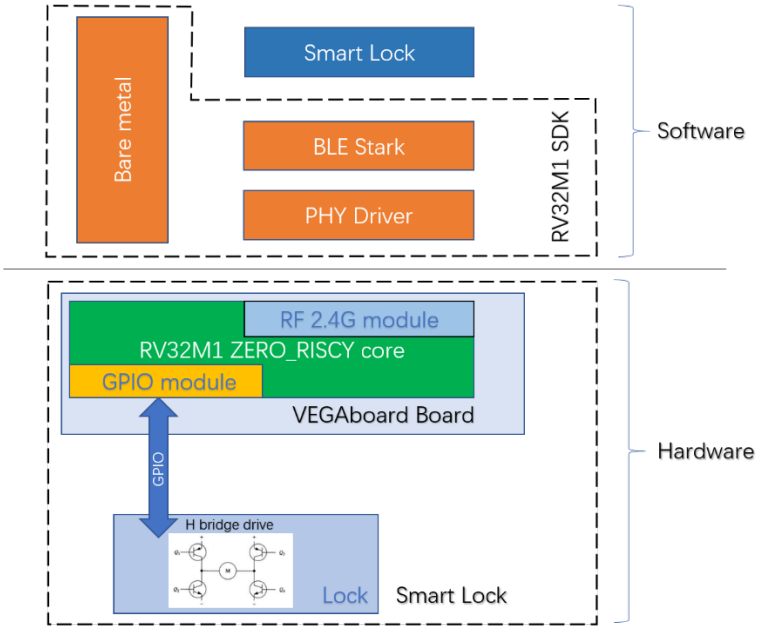


Figure 2 The system diagram

This demo is based on BLUETOOTH LE 4.2. It uses bare metal with ISR as multitask control. The lock is driven by a DC motor that's driven by H-bridge directly. RV32M1-VEGA board uses two GPIOs to control the DC motor to move forward or backward. One GPIO signal controls two semiconductor switches on the diagonal of the H-bridge. The demo works on 2.4GHz Radio.

5 Smart Lock Software

The section describes the workflow of the source code and how their files are organized in the project. The content needed a modification for the customized application is given too.

5.1 Workflow of Source Code

The demo starts from a *main* function, that is in *fsl_os_abstraction_bm.c* under the *framework/OSAbstraction* folder.

```
int main (void)
{
    extern void BOARD_InitHardware(void);

    OSA_Init();
    /* Initialize MCU clock */
    BOARD_InitHardware();
    OSA_TimeInit();
    OSA_TaskCreate(OSA_TASK(main_task),NULL);
    OSA_Start();

    return 0;
}
```

The `main` function initializes hardware modules and related resources on the board, then calls the `OSA_TaskCreate` function to create a `main_task`. At the end of the `main_task`, the `App_Thread` function is called.

```
void main_task(uint32_t param)
{
  ……
  ……
   /* Call application task */
    App_Thread( param );
}
```

The `App_Thread` function processes all events for the task including the timers, messages, and any other user defined events. In the function, it calls `OSA_EventWait` to wait for events. If any event occurs, it calls `App_HandleHostMessageInput` function to process it.

```
void App_Thread (uint32_t param)
{
  ……
  OSA_EventWait(mAppEvent, osaEventFlagsAll_c, FALSE, osaWaitForever_c , &event);
  ……
   /* Process it */
  App_HandleHostMessageInput(pMsgIn);
  ……
  }
```

The `App_HandleHostMessageInput` function has a switch-case structure, the application code is called when the received the message is `gAppGattServerMsg_c`.

```
static void App_HandleHostMessageInput(appMsgFromHost_t* pMsg)
{
  ……
  case gAppGattServerMsg_c:
        {
          if (pfGattServerCallback)
            ……

         }
  ……
  }
```

The function pointer `pfGattServerCallback` is set to `BleApp_GattServerCallback` function during registering it with `App_RegisterGattServerCallback` in `BleApp_Config`.

```
static void BleApp_Config()
{
  ……
  App_RegisterGattServerCallback(BleApp_GattServerCallback);
  ……
  }
```

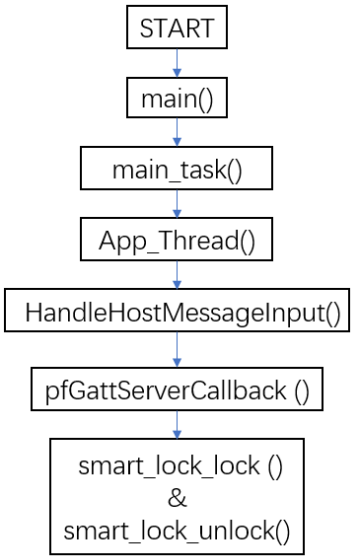
In the function `BleApp_GattServerCallback`, it calls the lock/unlock function according to the received packet.

```
static void BleApp_GattServerCallback (deviceId_t deviceId, gattServerEvent_t*
pServerEvent)
{
  ……
  if (handle == value_smart_lock_ctrl_point)
   {
   if (SMART_LOCK_STATUS_LOCK == val)
     smart_lock_lock(450);
   else
     smart_lock_unlock(450);
  ……
   }
  ……
}
```

In `main` function, after initializing and creating task, it calls `OSA_Start` function. The function is implemented as an infinite loop to sequentially execute multitasking.

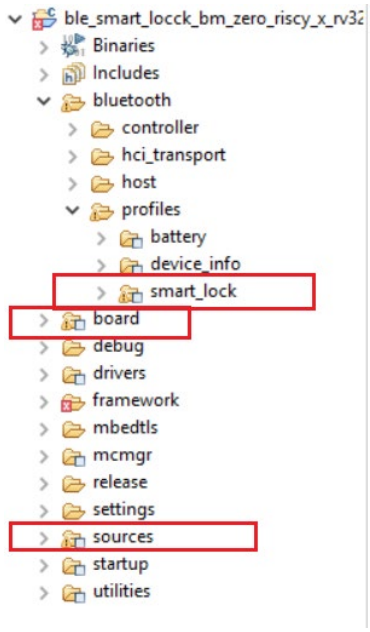
```
void OSA_Start(void)
{
  ……
   for(;;)
   {
  ……
   }
  ……
}
```

The workflow of the code is shown as below:



5.2 Structure of Source Code

The demo source code is divided into three parts, the first is in *sources* folder, the second is in *board* and the last is in *bluetooth/profiles/smart_lock*:



smart_lock:

- *smart_lock_service.c* defines and implements some functions that are called in the *app.c*.
- *smart_lock_interface.h* declares the functions implemented in the *smart_lock_service.c*.

sources:

- *app.c* defines and implements some functions that are called by BLUETOOTH LE stack, such as `BleApp_Init`, `BleApp_GenericCallback`, `BleApp_GattServerCallback`, *etc.*
- *app_config.c* defines some data and information about BLUETOOTH LE, such as scanning and advertising data, default advertising parameters.

```
app_config.c: change the advertise name
   ……
    {
     .adType = gAdShortenedLocalName_c,
     .length = 8,
     .aData = (uint8_t*)"SML_SML"
    }
   ……
```

- *smart_lock_ctrl.c* implements the functions that control the lock and defines the GPIOs that drive the H-bridge to rotate the DC motor to control the lock.

```
smart_lock_ctrl.c: change the GPIOs

/* Declare smart lock control GPIO pins */
gpioOutputPinConfig_t smart_lock_ctrl_pins[2] = {
   ……
 .gpioPort = gpioPort_C_c,
 .gpioPin = 28,

   ……
 .gpioPort = gpioPort_C_c,
 .gpioPin = 29,
   ……
   }
smart_lock_ctrl.c: lock driver
   ……
int32_t smart_lock_lock(uint32_t delay_ms){……}
int32_t smart_lock_unlock(uint32_t delay_ms){……}
   ……
```

- *gatt_db.h* defines the service, characteristic and value including *smart lock service* that has two types. One is writable that is used to control the lock. Another is readable that is used to record the status of lock.

```
gatt_db.h: define the smart lock service
   ……
  PRIMARY_SERVICE_UUID128(service_smart_lock, uuid_service_smart_lock)
   ……
```

- *gatt_uuid128.h* defines the UUID of *smart lock service*.

```
gatt_uuid128.h: define the UUID of smart lock service
   ……
  UUID128(uuid_service_smart_lock,0x00, 0x7a, 0x3c, 0x9a, 0x05, 0x00, 0x96, 0x8a,
   0xe6, 0x11, 0x1c, 0x70, 0x60, 0x6c, 0x84, 0xbc)
   ……
```

board:

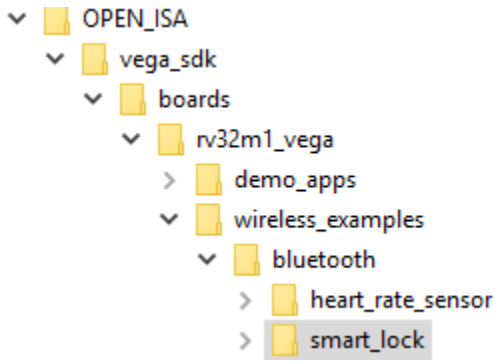
- *board.h* provides some board level interfaces and defines some macros about board.

```
board.h: change the board name
   ……
    /*! @brief The board name */
   #define MANUFACTURER_NAME          "OPEN-ISA"
   #define BOARD_NAME                 "RV32M1-VEGA"
   ……
```

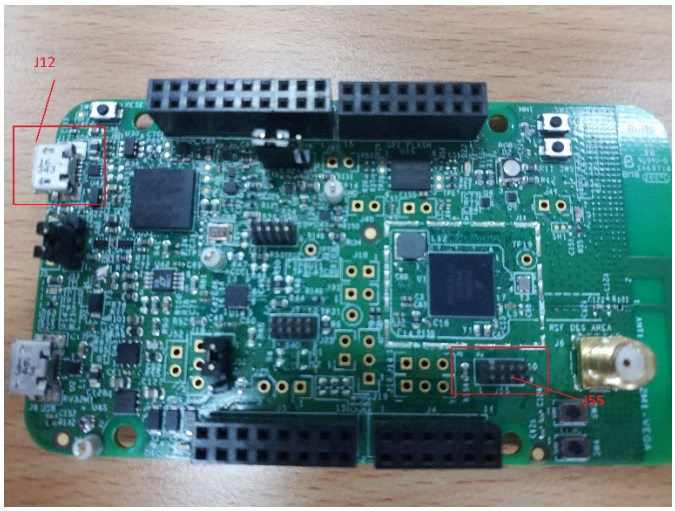
Note: These files in the eclipse project are in virtual folder, their physical location on the hard disk can be seen in project's properties.

6 Demo Setup

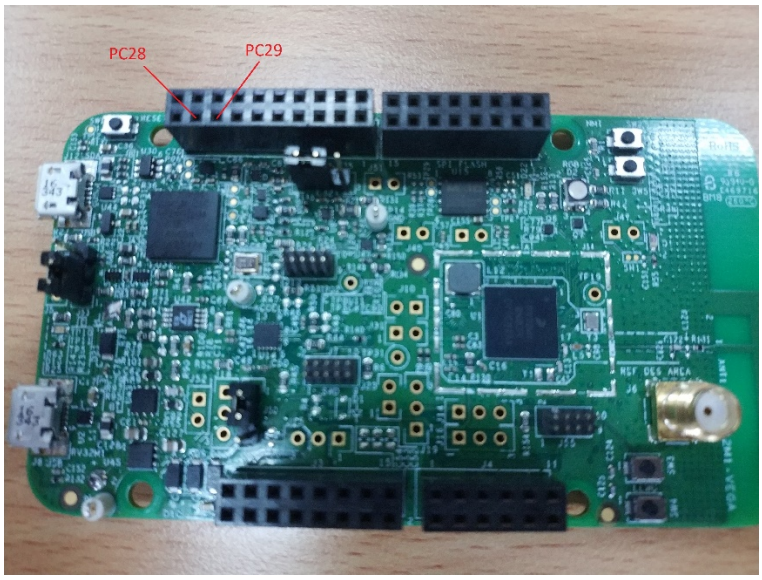
1) Import the ble_smart_locck_bm_zero_riscy_x_rv32m1_vega project into eclipse IDE, the root directory is showed below:



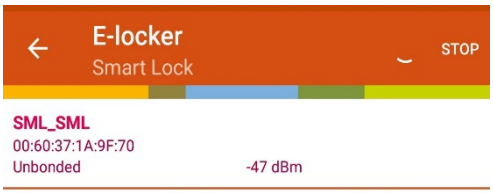
2) Set the release version as the active build project and build it.
3) Set rv32m1 boot from ZERO_RISCY core according to *RV32M1-VEGA Quick Start Guide*.
4) Connect J55 on the board to PC using Jlink and connect J12 on the board to PC using USB cable.

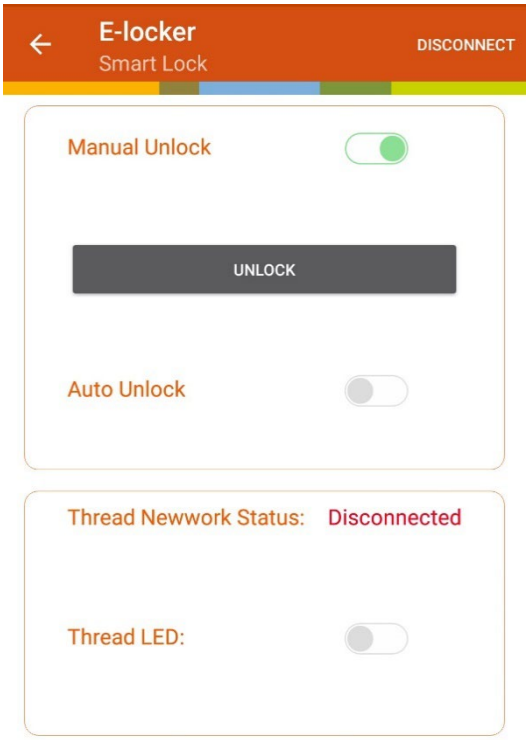


5) Download the demo image to the board;
6) After download, remove the JLINK and shut down the power of the board, then power on the board again;
7) Connect the PC28 (J2-pin17) and PC29(J2-pin15) GPIO on the board to the cable of H-bridge;

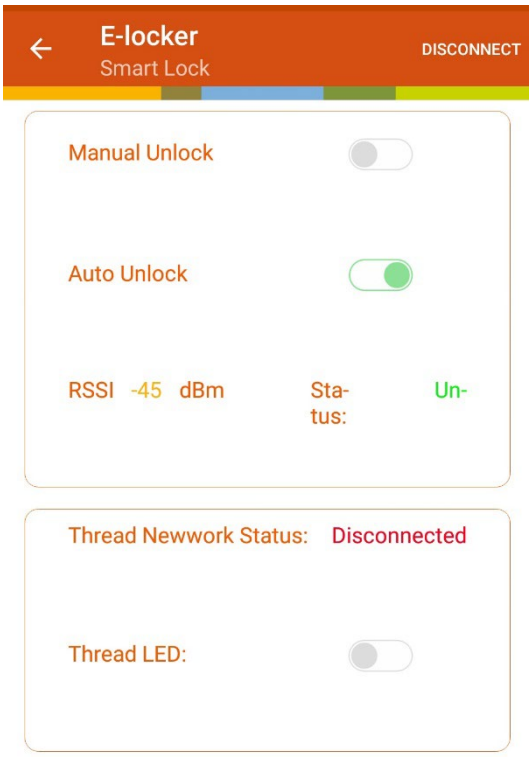


8) Run smart phone app and choose smart lock to start scan, the board will be found, whose name is “SML_SML”, then the lock can be controlled in manual mode or auto mode.





9) In auto mode, the lock’s status is decided by the value of RSSI. When the value is high, the lock will be un-locked. On the contrary, when the value is small, it will be locked.



7 Reference

Following references are available on www.open-isa.org:

- *RV32M1-VEGA-SCH: Schematics*
- *RV32M1-VEGA-LAYOUT: Layout*
- *RV32M1_VEGA_Board_User_Guide*
- *RV32M1RM: Reference Manual*
- *RV32M1DS: Datasheet*
- *RV32M1_Vega_Develop_Environment_Setup*

8 Revision history

Rev.	Date	Substantive change(s)
0	10/2018	Initial release