Wireless Connectivity for MCU-based IoT Designs

Class 3: BlueTooth

101/01/2017

Warren Miller

Presented by:

This Week's Agenda

10/30/17	Wireless Connectivity for IoT Designs
10/31/17	The Renesas Synergy Platform
11/01/17	BlueTooth
11/02/17	Wi-Fi
11/03/17	Cellular and More

Presented by:

Course Description

- This course will focus on three important wireless IoT connectivity methods- BlueTooth LE, WiFi and Cellular.
- A short description of each technology will be provided, along with hands-on example implementations.
- The Renesas Synergy Platform will be used as the target for the hands-on implementations and interested students can optionally download the free software, which includes the popular ThreadX RTOS and associated networking stacks.
- Additionally, students can optionally purchase a Synergy hardware kit to test out the hands-on designs used in the course.

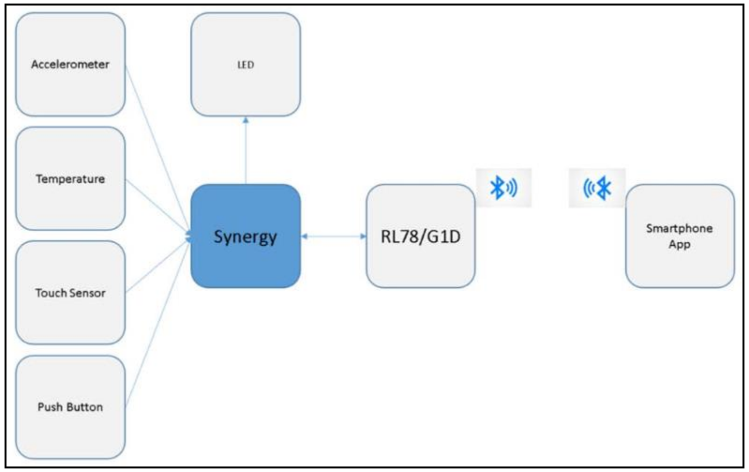
Today's Topics

- BlueTooth Implementation Example
- Hardware
- Software architecture
- Software flow
- Selected code review
- Resources

BlueTooth Implementation

Required Resources

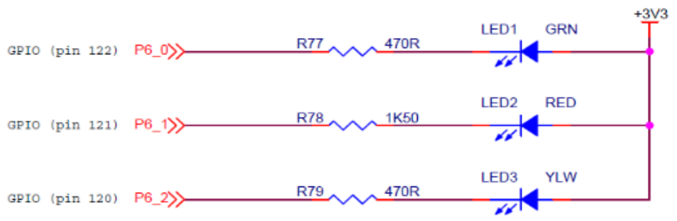
- Renesas Synergy SK-S7G2 v 2.0 and above
- Digilent PmodACL2 accelerometer
- iOS or Android smart device
- Windows PC
- E2 studio ISDE or IAR EW for Synergy
- Synergy Software Package (SSP) or Synergy Stand Alone Configurator (SSC)



Application Project

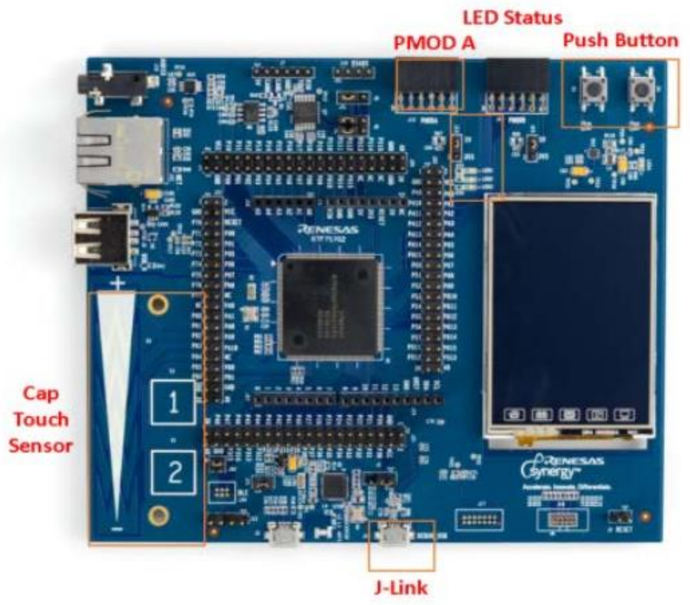
- Typical BLE IoT application using sensors
    - Accelerometer, Temp, Touch, Button
    - LED
    - Smartphone
- Control sensor operation via the smart phone

Hardware



Renesas Synergy SK-S7G2 v 2.0 and above

- S7G2 microprocessor with 176 LQFP package
- Four connectors that provide access to all S7G2 mic
signals
- Low cost QVGA TFT touch screen
- Three user LEDs
- Arduino Shield Uno compatible socket
- Two mechanical switches connected directly to micr
interrupt pins
- Two capacitive touch-buttons connected to pins that
interrupts
- One capacitive slider
- Audio output
- QSPI memory (8MB)
- SPI, IIC, CAN, and SCI interface



Hardware used with application project

Other types of example projects

Resources

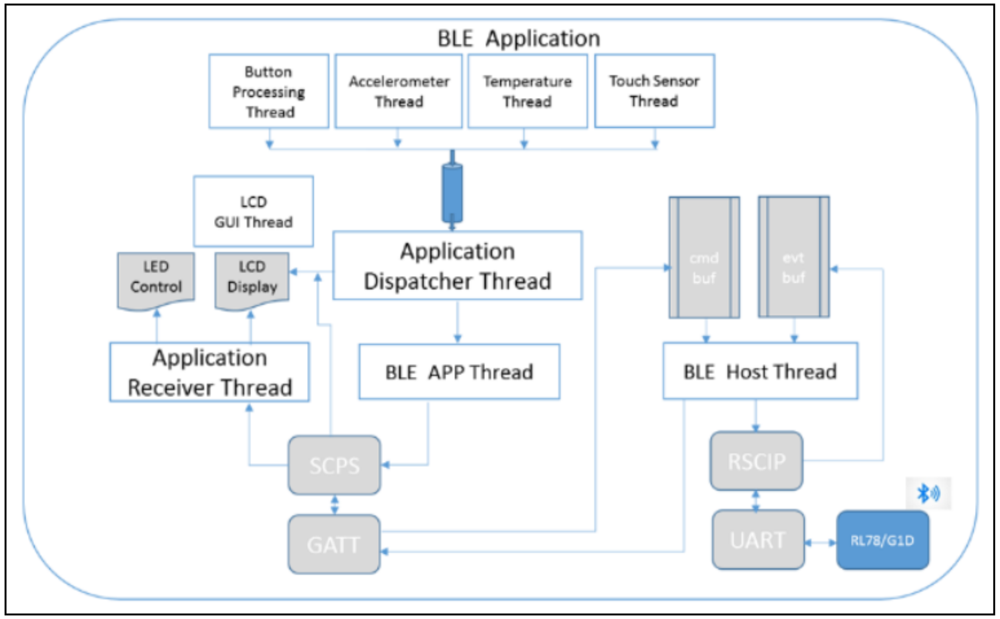
https://www.renesas.com/en-us/doc/products/renesas-synergy/doc/r12um0004eu0100_synergy_sk_s7g2.pdf

Application Architecture

BLE Application Threads

- Dispatcher
- Sensors
- GUI
- Receiver
- BLE App and Host

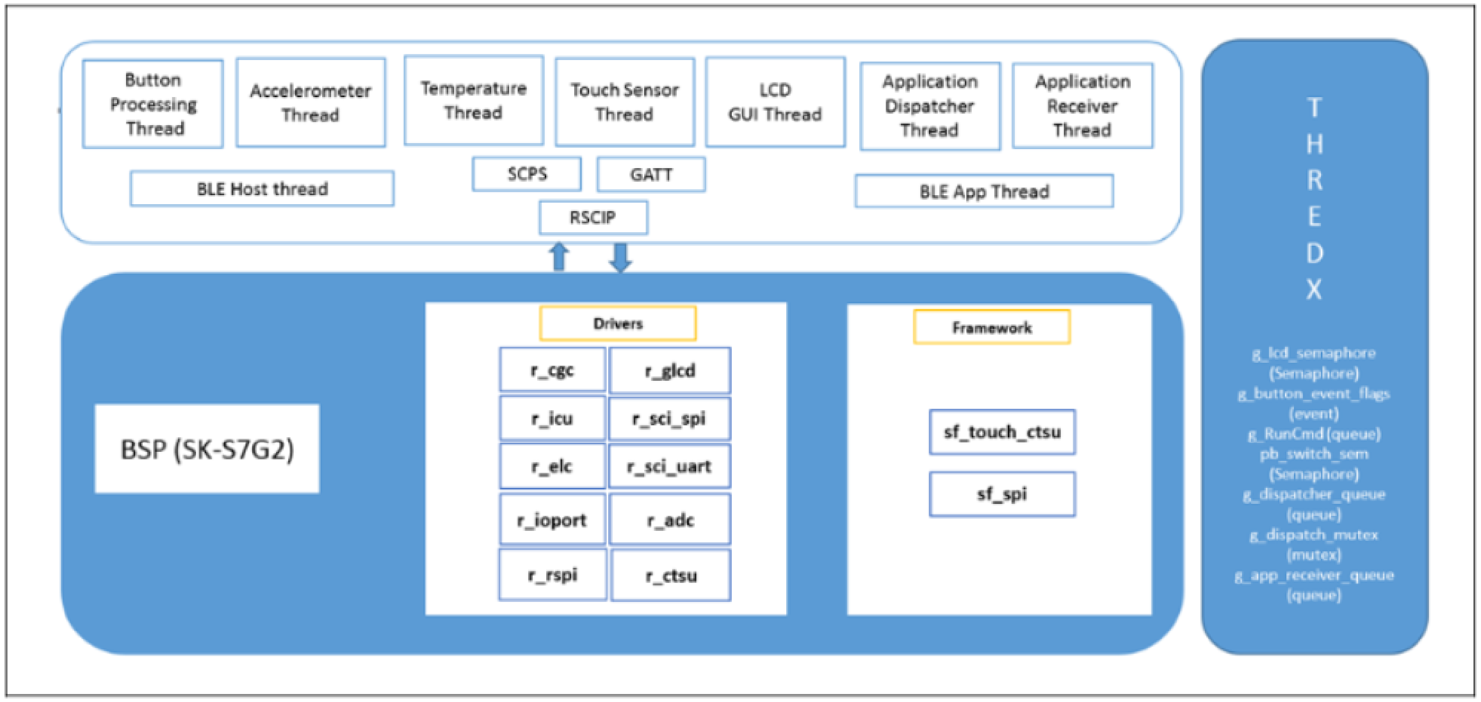
- Generic Access Profile
- Security Manager
- Sample Custom Profile Server



Presented by:

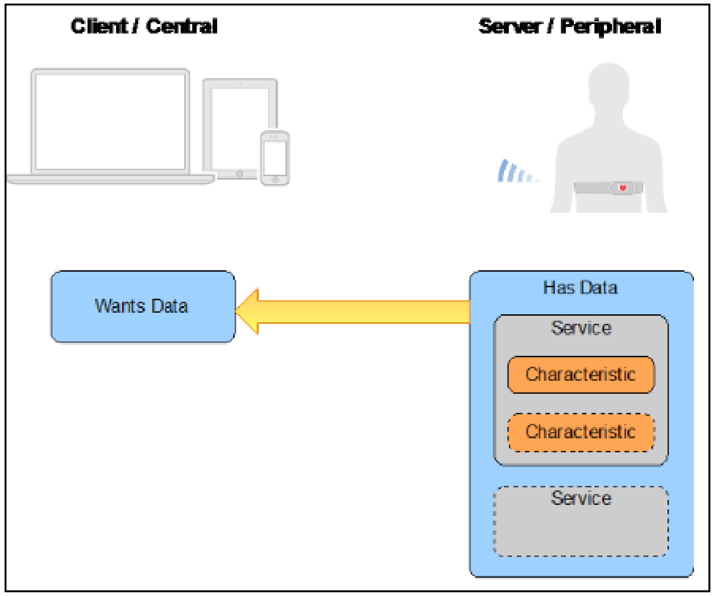
SSP Components

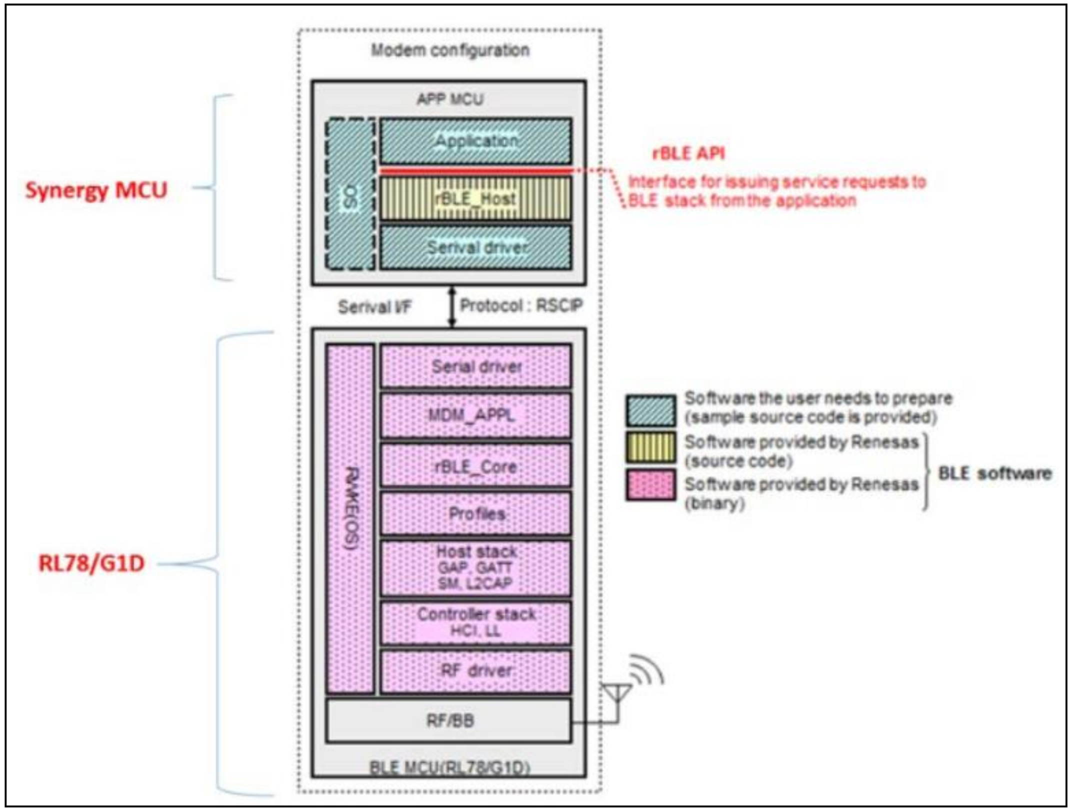
- Application
- Modules (Drivers)
- Frameworks
- BSP
- Thread-X



Modem Operation

- APP MCU
- Serial Interface
  - Renesas protocol
- BLE MCU
- Software components
- Functional model- client/server

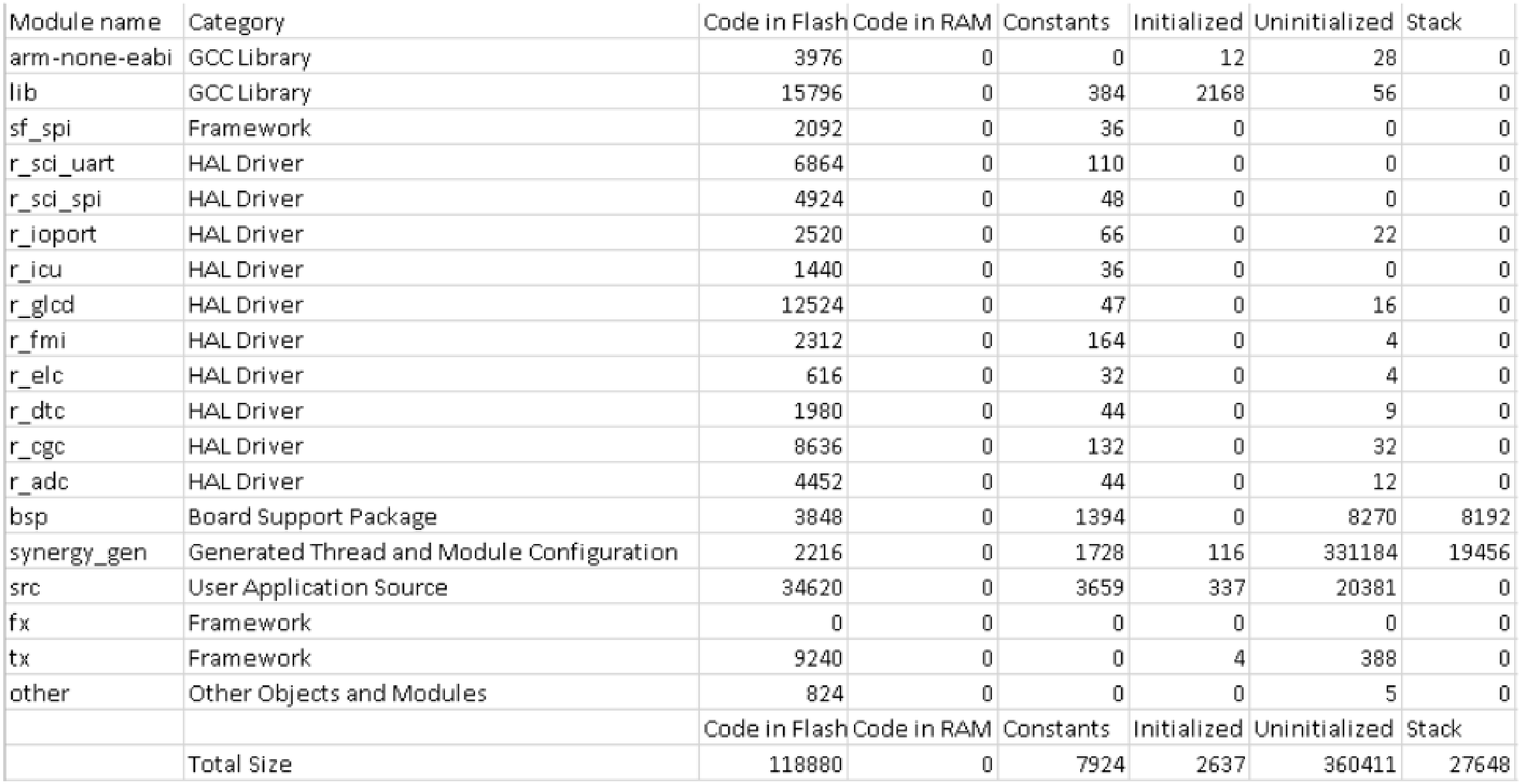



Modem configuration
APP MCU
Application
OS
rBLE_Host
Serival driver
Synergy MCU
rBLE API
Interface for issuing service requests to BLE stack from the application
Serival I/F
Protocol : RSCIP
Serial driver
MDM_APPL
rBLE_Core
Profiles
Host stack GAP, GATT SM, L2CAP
Controller stack HCI, LL
RF driver
RWKE(OS)
RF/BB
BLE MCU(RL78/G1D)
RL78/G1D
Software the user needs to prepare (sample source code is provided)
Software provided by Renesas (source code)
Software provided by Renesas (binary)
BLE software

Presented by:

Memory Usage

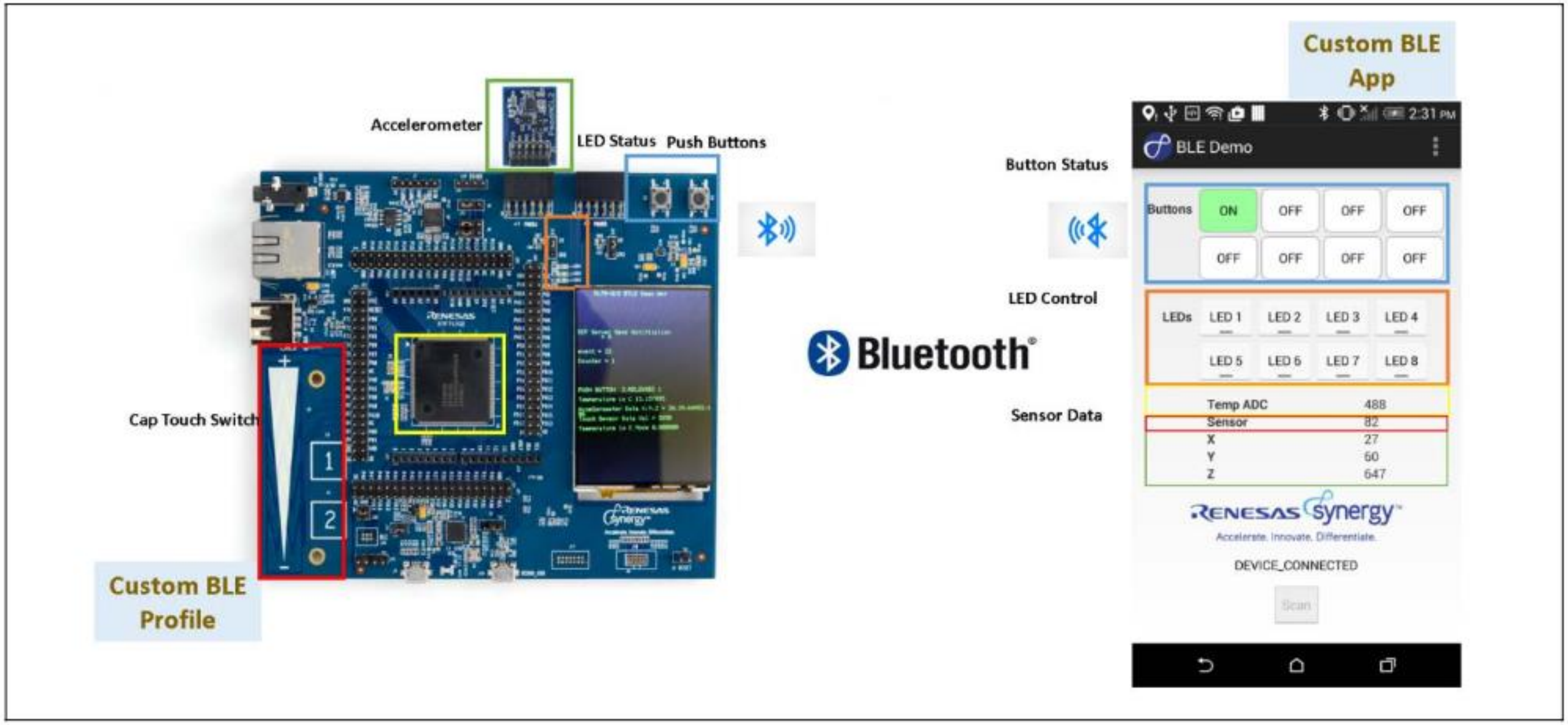
Report on memory use by module



Module name	Category	Code in Flash	Code in RAM	Constants	Initialized	Uninitialized	Stack
arm-none-eabi	GCC Library	3976	0	0	12	28	0
lib	GCC Library	15796	0	384	2168	56	0
sf_spi	Framework	2092	0	36	0	0	0
r_sci_uart	HAL Driver	6864	0	110	0	0	0
r_sci_spi	HAL Driver	4924	0	48	0	0	0
r_ioport	HAL Driver	2520	0	66	0	22	0
r_icu	HAL Driver	1440	0	36	0	0	0
r_glcd	HAL Driver	12524	0	47	0	16	0
r_fmi	HAL Driver	2312	0	164	0	4	0
r_elc	HAL Driver	616	0	32	0	4	0
r_dtc	HAL Driver	1980	0	44	0	9	0
r_cgc	HAL Driver	8636	0	132	0	32	0
r_adc	HAL Driver	4452	0	44	0	12	0
bsp	Board Support Package	3848	0	1394	0	8270	8192
synergy_gen	Generated Thread and Module Configuration	2216	0	1728	116	331184	19456
src	User Application Source	34620	0	3659	337	20381	0
fx	Framework	0	0	0	0	0	0
tx	Framework	9240	0	0	4	388	0
other	Other Objects and Modules	824	0	0	0	5	0
		Code in Flash	Code in RAM	Constants	Initialized	Uninitialized	Stack
	Total Size	118880	0	7924	2637	360411	27648

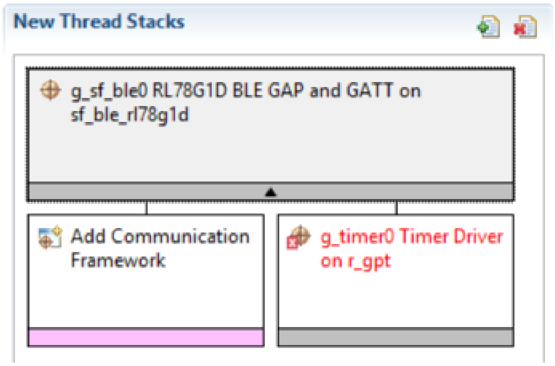
Presented by:

BLE Application in Action



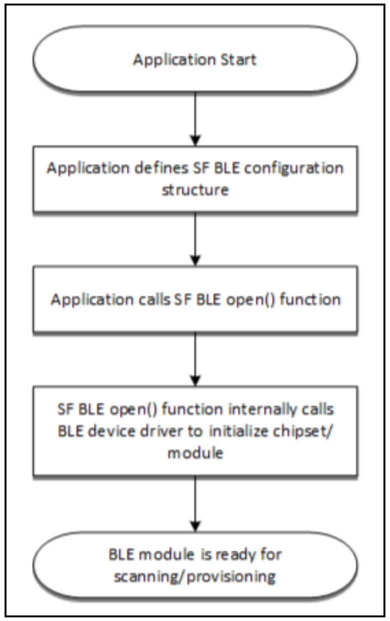
Presented by:

BLE Framework for Synergy

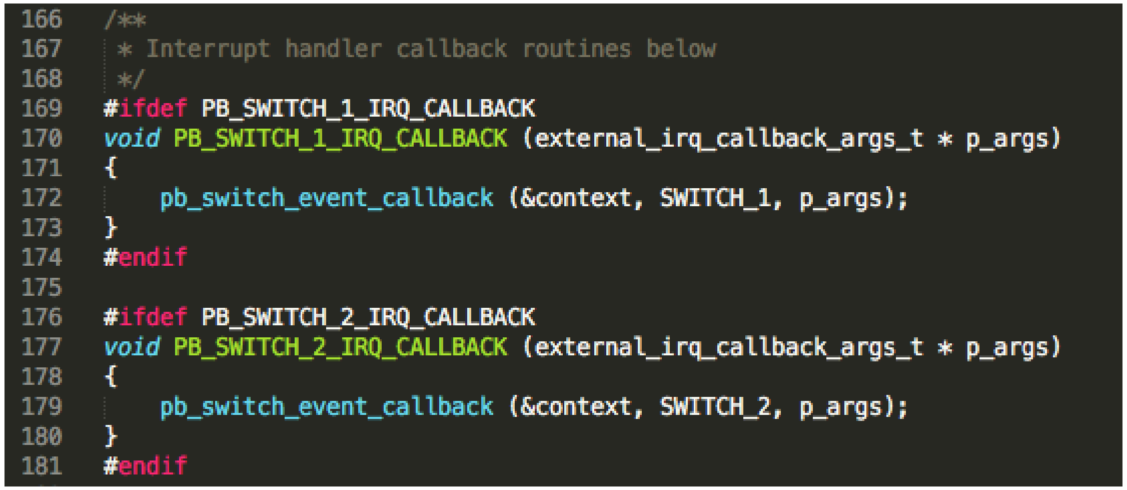

New Thread Stacks
g_sf_ble0 RL78G1D BLE GAP and GATT on sf_ble_rl78g1d
Add Communication Framework
g_timer0 Timer Driver on r_gpt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking.
Heart Rate Profile	Enabled, Disabled (Default: Enabled)	Heart rate profile selection
Alert Notification Profile	Enabled, Disabled (Default: Disabled)	Alert notification profile selection
Blood Pressure Profile	Enabled, Disabled (Default: Disabled	Blood pressure profile selection

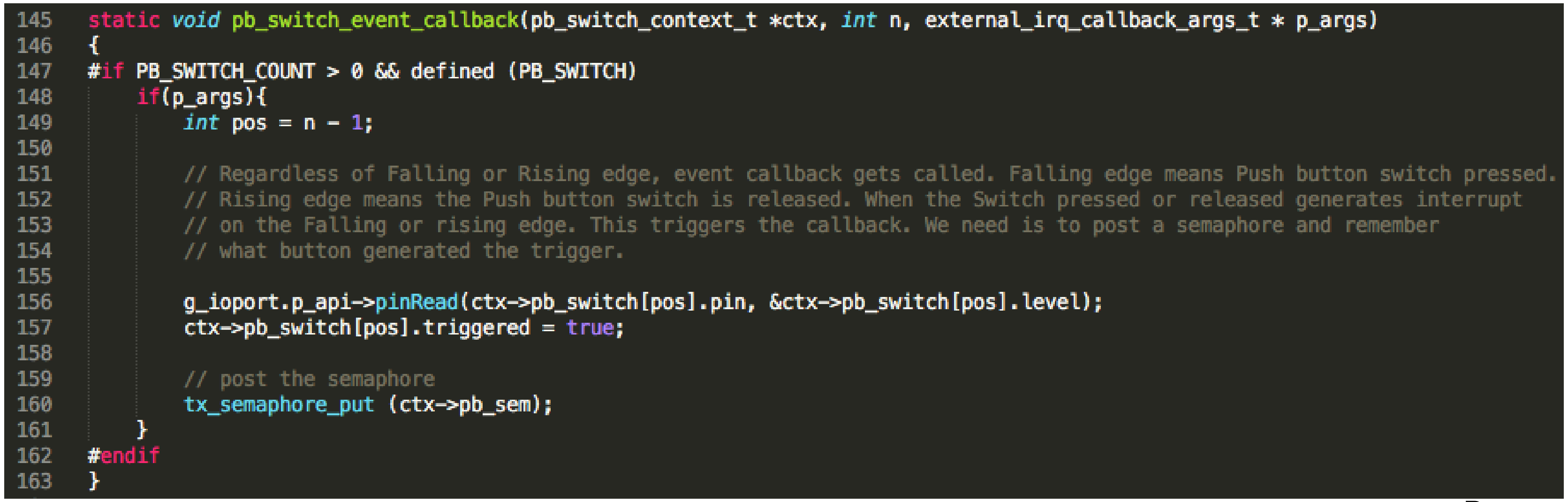
Function Name	Example API Call and Description
.gattServiceDiscovery	g_sf_ble0.p_api->gattServiceDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattServiceDiscovery() function perfors the service discovery.
.gattCharDiscovery	g_sf_ble0.p_api->gattCharDiscovery (g_sf_ble_0.p_cfg, p_handle, p_sf_ble_svc_dscv_req, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); The gattServiceDiscovery() function performs the service discovery operation.
.gattCharDescDiscovery	g_sf_ble0.p_api->gattCharDescDiscovery (g_sf_ble_0.p_cfg, p_handle, start_handle, end_handle, p_sf_ble_svc_dscv_rsp, p_rsp_cnt); Discovers GATT characteristics descriptor on a remote device.
.gattCharRead	g_sf_ble0.p_api->gattCharRead (g_sf_ble_0.p_cfg, p_handle, start_handle, p_char_read_req, p_char_read_rsp); Reads GATT characteristics on a remote device.


Application Start
Application defines SF BLE configuration structure
Application calls SF BLE open() function
SF BLE open() function internally calls BLE device driver to initialize chipset/ module
BLE module is ready for scanning/provisioning

Some Code- Input Callback



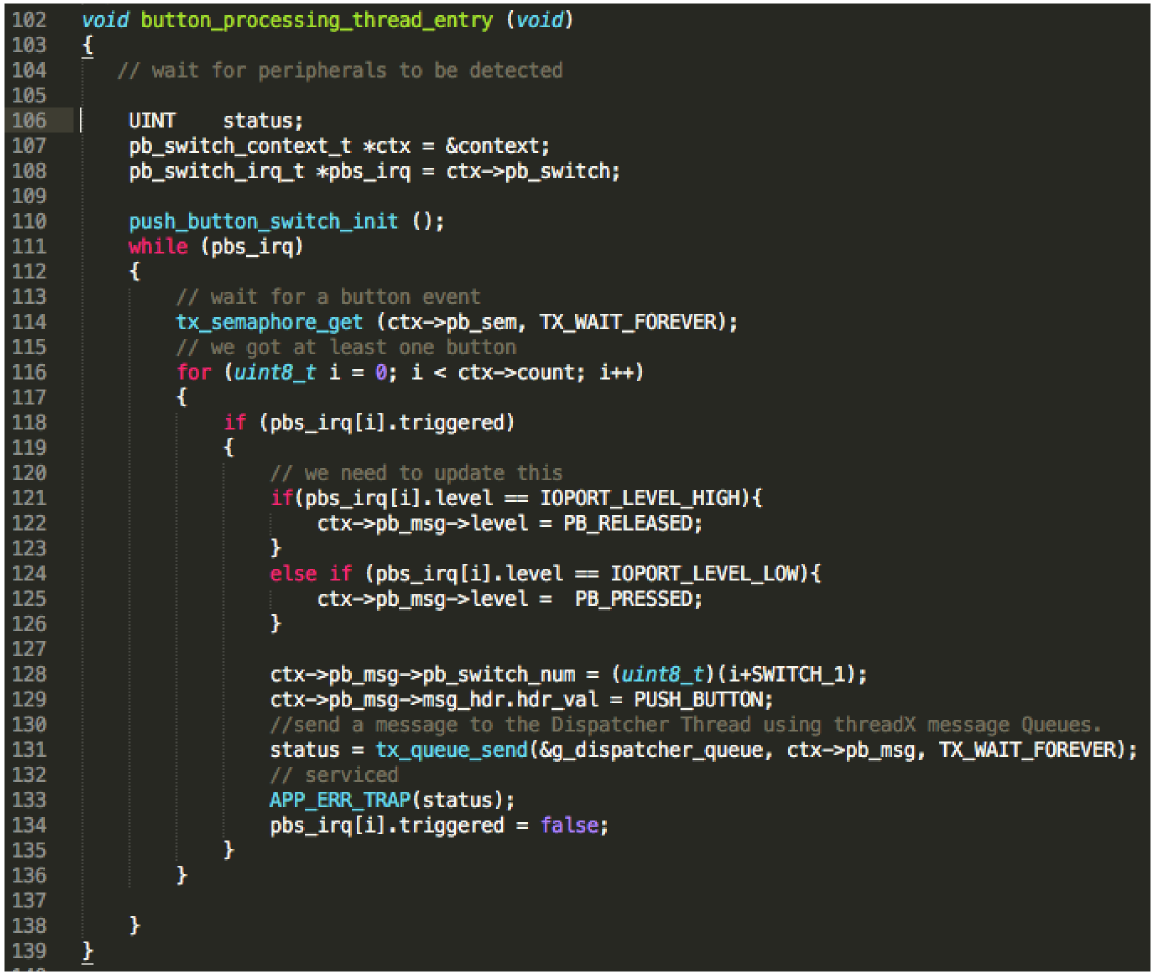
```c
/**
 * Interrupt handler callback routines below
 */
#ifdef PB_SWITCH_1_IRQ_CALLBACK
void PB_SWITCH_1_IRQ_CALLBACK (external_irq_callback_args_t * p_args)
{
    pb_switch_event_callback (&context, SWITCH_1, p_args);
}
#endif

#ifdef PB_SWITCH_2_IRQ_CALLBACK
void PB_SWITCH_2_IRQ_CALLBACK (external_irq_callback_args_t * p_args)
{
    pb_switch_event_callback (&context, SWITCH_2, p_args);
}
#endif
```



```c
static void pb_switch_event_callback(pb_switch_context_t *ctx, int n, external_irq_callback_args_t * p_args)
{
#if PB_SWITCH_COUNT > 0 && defined (PB_SWITCH)
    if(p_args){
        int pos = n - 1;

        // Regardless of Falling or Rising edge, event callback gets called. Falling edge means Push button switch pressed.
        // Rising edge means the Push button switch is released. When the Switch pressed or released generates interrupt
        // on the Falling or rising edge. This triggers the callback. We need is to post a semaphore and remember
        // what button generated the trigger.

        g_ioport.p_api->pinRead(ctx->pb_switch[pos].pin, &ctx->pb_switch[pos].level);
        ctx->pb_switch[pos].triggered = true;

        // post the semaphore
        tx_semaphore_put (ctx->pb_sem);
    }
#endif
}
```

Presented by:

Some Code- Button Processing



```c
void button_processing_thread_entry (void)
{
   // wait for peripherals to be detected

    UINT    status;
    pb_switch_context_t *ctx = &context;
    pb_switch_irq_t *pbs_irq = ctx->pb_switch;

    push_button_switch_init ();
    while (pbs_irq)
    {
        // wait for a button event
        tx_semaphore_get (ctx->pb_sem, TX_WAIT_FOREVER);
        // we got at least one button
        for (uint8_t i = 0; i < ctx->count; i++)
        {
            if (pbs_irq[i].triggered)
            {
                // we need to update this
                if(pbs_irq[i].level == IOPORT_LEVEL_HIGH){
                    ctx->pb_msg->level = PB_RELEASED;
                }
                else if (pbs_irq[i].level == IOPORT_LEVEL_LOW){
                    ctx->pb_msg->level =  PB_PRESSED;
                }

                ctx->pb_msg->pb_switch_num = (uint8_t)(i+SWITCH_1);
                ctx->pb_msg->msg_hdr.hdr_val = PUSH_BUTTON;
                //send a message to the Dispatcher Thread using threadX message Queues.
                status = tx_queue_send(&g_dispatcher_queue, ctx->pb_msg, TX_WAIT_FOREVER);
                // serviced
                APP_ERR_TRAP(status);
                pbs_irq[i].triggered = false;
            }
        }

    }
}
```

DesignNews

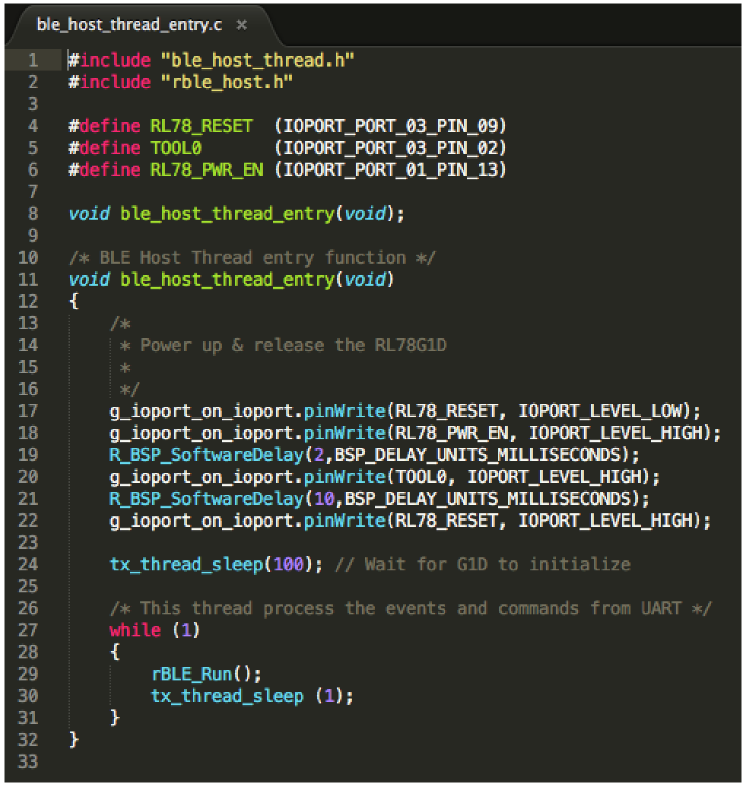
Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

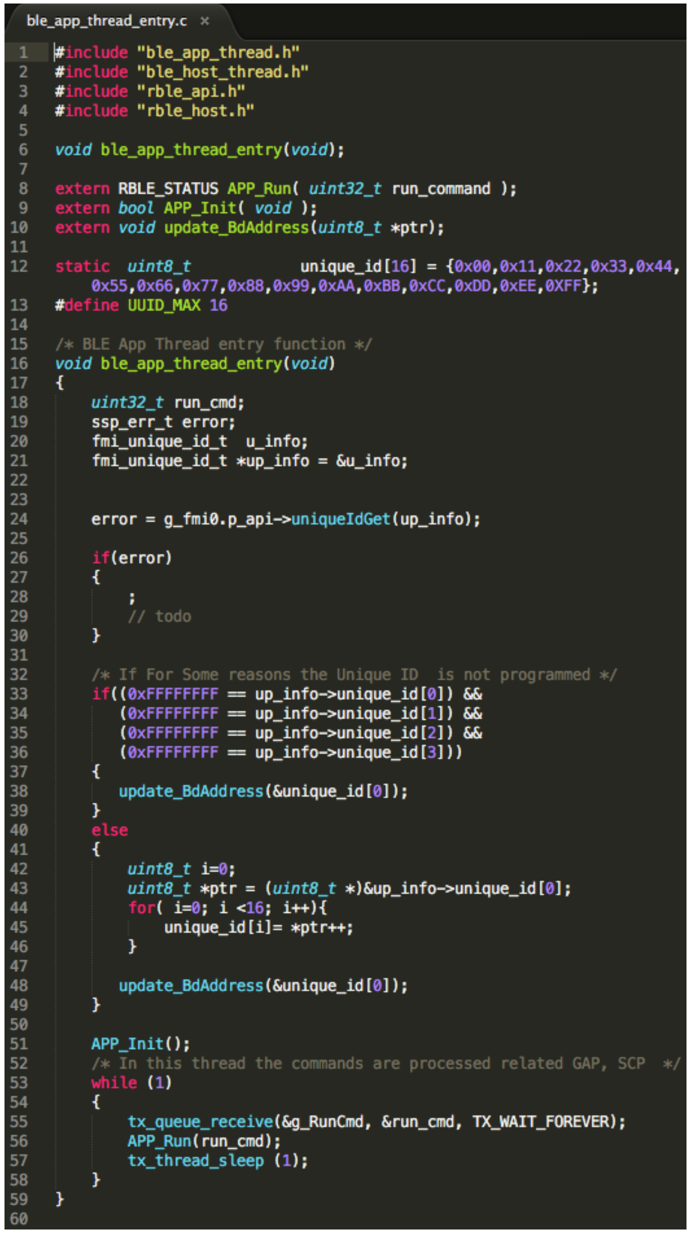
BLE Code

- Host and Application Entry
- rBLE_Run and APP_Run do the work
  - R_BLE/src/host/rble_host.c, rble_if_app_cb.c



ble_host_thread_entry.c

```c
#include "ble_host_thread.h"
#include "rble_host.h"

#define RL78_RESET  (IOPORT_PORT_03_PIN_09)
#define TOOL0       (IOPORT_PORT_03_PIN_02)
#define RL78_PWR_EN (IOPORT_PORT_01_PIN_13)

void ble_host_thread_entry(void);

/* BLE Host Thread entry function */
void ble_host_thread_entry(void)
{
    /*
     * Power up & release the RL78G1D
     *
     */
    g_ioport_on_ioport.pinWrite(RL78_RESET, IOPORT_LEVEL_LOW);
    g_ioport_on_ioport.pinWrite(RL78_PWR_EN, IOPORT_LEVEL_HIGH);
    R_BSP_SoftwareDelay(2,BSP_DELAY_UNITS_MILLISECONDS);
    g_ioport_on_ioport.pinWrite(TOOL0, IOPORT_LEVEL_HIGH);
    R_BSP_SoftwareDelay(10,BSP_DELAY_UNITS_MILLISECONDS);
    g_ioport_on_ioport.pinWrite(RL78_RESET, IOPORT_LEVEL_HIGH);

    tx_thread_sleep(100); // Wait for G1D to initialize

    /* This thread process the events and commands from UART */
    while (1)
    {
        rBLE_Run();
        tx_thread_sleep (1);
    }
}

```



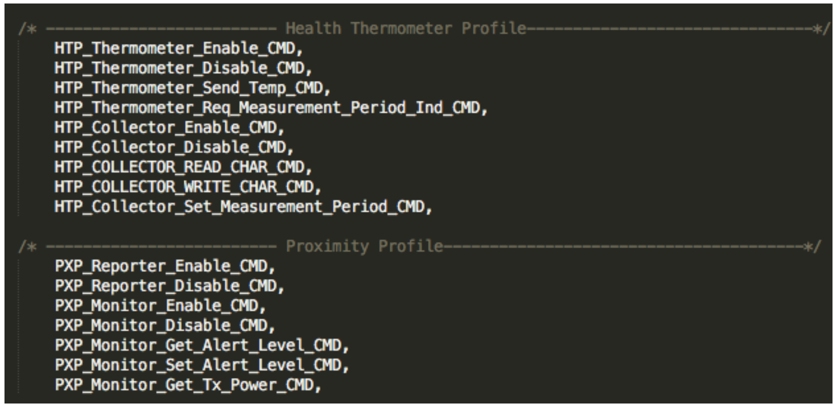
ble_app_thread_entry.c

```c
#include "ble_app_thread.h"
#include "ble_host_thread.h"
#include "rble_api.h"
#include "rble_host.h"

void ble_app_thread_entry(void);

extern RBLE_STATUS APP_Run( uint32_t run_command );
extern bool APP_Init( void );
extern void update_BdAddress(uint8_t *ptr);

static  uint8_t             unique_id[16] = {0x00,0x11,0x22,0x33,0x44,
    0x55,0x66,0x77,0x88,0x99,0xAA,0xBB,0xCC,0xDD,0xEE,0XFF};
#define UUID_MAX 16

/* BLE App Thread entry function */
void ble_app_thread_entry(void)
{
    uint32_t run_cmd;
    ssp_err_t error;
    fmi_unique_id_t  u_info;
    fmi_unique_id_t *up_info = &u_info;


    error = g_fmi0.p_api->uniqueIdGet(up_info);

    if(error)
    {
        ;
        // todo
    }

    /* If For Some reasons the Unique ID  is not programmed */
    if((0xFFFFFFFF == up_info->unique_id[0]) &&
       (0xFFFFFFFF == up_info->unique_id[1]) &&
       (0xFFFFFFFF == up_info->unique_id[2]) &&
       (0xFFFFFFFF == up_info->unique_id[3]))
    {
       update_BdAddress(&unique_id[0]);
    }
    else
    {
        uint8_t i=0;
        uint8_t *ptr = (uint8_t *)&up_info->unique_id[0];
        for( i=0; i <16; i++){
            unique_id[i]= *ptr++;
        }

       update_BdAddress(&unique_id[0]);
    }

    APP_Init();
    /* In this thread the commands are processed related GAP, SCP  */
    while (1)
    {
        tx_queue_receive(&g_RunCmd, &run_cmd, TX_WAIT_FOREVER);
        APP_Run(run_cmd);
        tx_thread_sleep (1);
    }
}

```

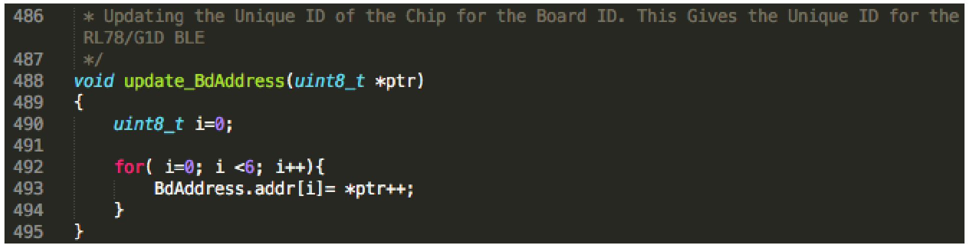
Presented by:

BLE Code- App Run

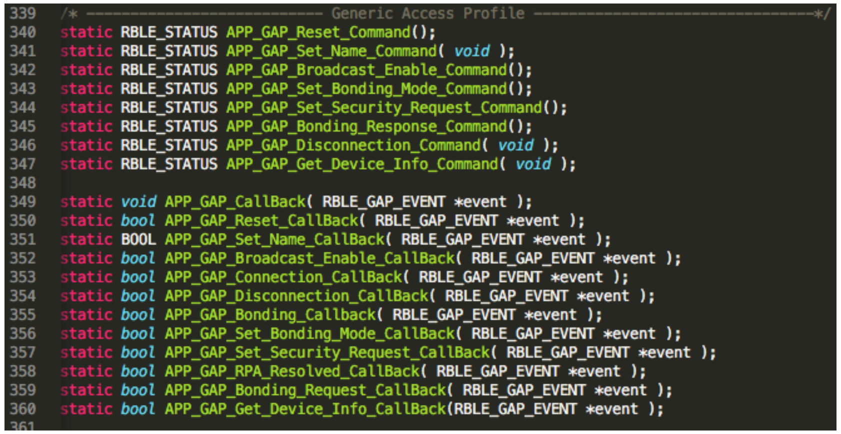
- rble_if_app_cb.c



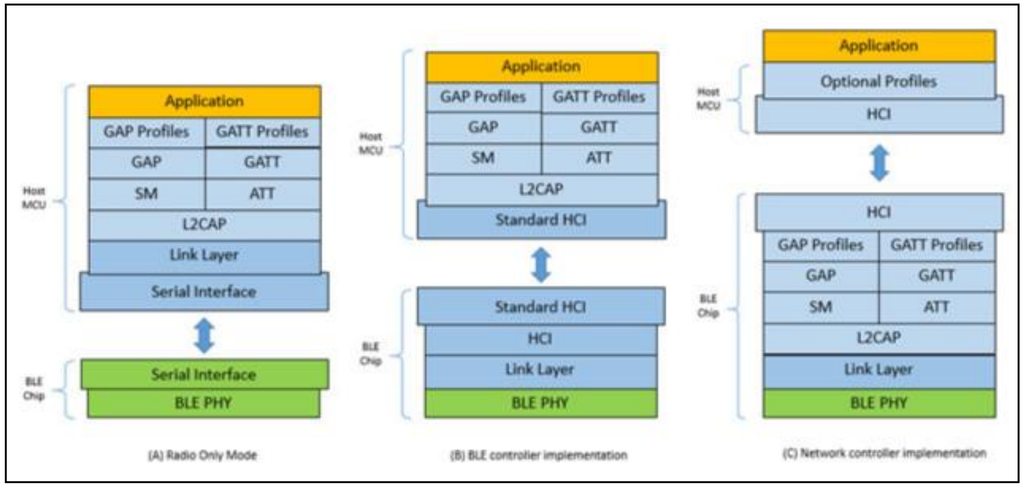
```
/* ------------------------ Health Thermometer Profile------------------------------*/
    HTP_Thermometer_Enable_CMD,
    HTP_Thermometer_Disable_CMD,
    HTP_Thermometer_Send_Temp_CMD,
    HTP_Thermometer_Req_Measurement_Period_Ind_CMD,
    HTP_Collector_Enable_CMD,
    HTP_Collector_Disable_CMD,
    HTP_COLLECTOR_READ_CHAR_CMD,
    HTP_COLLECTOR_WRITE_CHAR_CMD,
    HTP_Collector_Set_Measurement_Period_CMD,

/* ------------------------ Proximity Profile-----------------------------------*/
    PXP_Reporter_Enable_CMD,
    PXP_Reporter_Disable_CMD,
    PXP_Monitor_Enable_CMD,
    PXP_Monitor_Disable_CMD,
    PXP_Monitor_Get_Alert_Level_CMD,
    PXP_Monitor_Set_Alert_Level_CMD,
    PXP_Monitor_Get_Tx_Power_CMD,
```



```
 * Updating the Unique ID of the Chip for the Board ID. This Gives the Unique ID for the
 RL78/G1D BLE
 */
void update_BdAddress(uint8_t *ptr)
{
    uint8_t i=0;

    for( i=0; i <6; i++){
        BdAddress.addr[i]= *ptr++;
    }
}
```



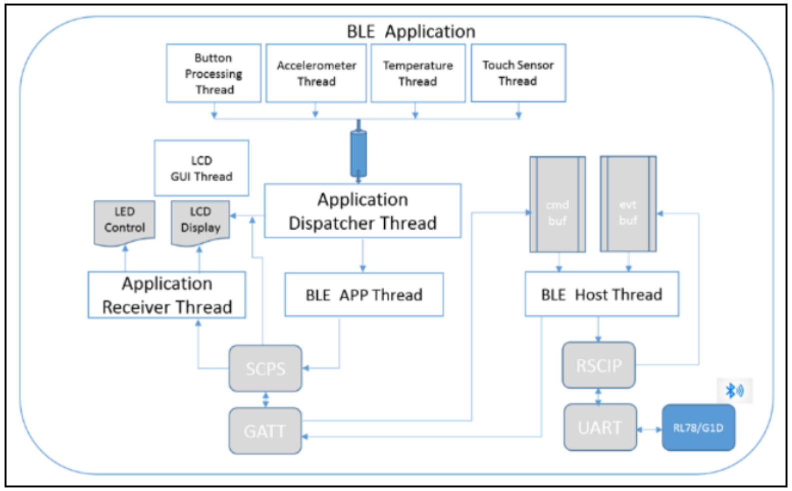
```
/* ------------------------ Generic Access Profile ------------------------------*/
static RBLE_STATUS APP_GAP_Reset_Command();
static RBLE_STATUS APP_GAP_Set_Name_Command( void );
static RBLE_STATUS APP_GAP_Broadcast_Enable_Command();
static RBLE_STATUS APP_GAP_Set_Bonding_Mode_Command();
static RBLE_STATUS APP_GAP_Set_Security_Request_Command();
static RBLE_STATUS APP_GAP_Bonding_Response_Command();
static RBLE_STATUS APP_GAP_Disconnection_Command( void );
static RBLE_STATUS APP_GAP_Get_Device_Info_Command( void );

static void APP_GAP_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Reset_CallBack( RBLE_GAP_EVENT *event );
static BOOL APP_GAP_Set_Name_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Broadcast_Enable_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Connection_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Disconnection_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Bonding_Callback( RBLE_GAP_EVENT *event );
static bool APP_GAP_Set_Bonding_Mode_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Set_Security_Request_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_RPA_Resolved_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Bonding_Request_CallBack( RBLE_GAP_EVENT *event );
static bool APP_GAP_Get_Device_Info_CallBack(RBLE_GAP_EVENT *event );
```



https://www.renesas.com/en-us/software/D6001083.html

Presented by:

Class Resources



- Synergy BLE Project

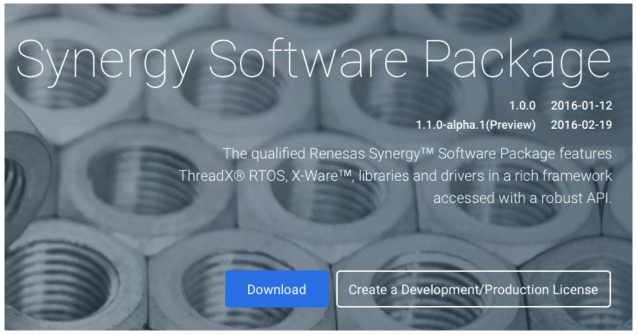
https://www.renesas.com/en-us/doc/products/renesas-synergy/apn/r12an0056eu0112-synergy-ble-rl78-g1d.pdf

- Creating custom profiles

https://en-in.knowledgebase.renesas.com/?title=English_Content/MCUMPU/End_Applications_%26_Key_Technology/Key_Technology/Bluetooth_low_energy/Protocol_Stack/Reference_documents_for_creating_Custom_Profiles

Course Resources

- Express Logic Web Site: www.rtos.com
- Express Logic Articles and White papers Here
- Express Logic RTOS Book (Amazon) Here



Synergy Software Package

1.0.0 2016-01-12

1.1.0-alpha.1(Preview) 2016-02-19

The qualified Renesas Synergy™ Software Package features ThreadX® RTOS, X-Ware™, libraries and drivers in a rich framework accessed with a robust API.

- Renesas Synergy Platform Kits at Digi-Key Here
- Course Kit Resources Here
- Renesas Synergy Platform Here
- Renesas Synergy Gallery (https://synergygallery.renesas.com/auth/login)

This Week's Agenda

10/30/17	Wireless Connectivity for IoT Designs
10/31/17	The Renesas Synergy Platform
11/01/17	BlueTooth
11/02/17	Wi-Fi
11/03/17	Cellular and More