

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA

PROJETO DE SISTEMAS EMBARCADOS

Dispositivo para encontrar sinais de Bluetooth

Felipe Seitenfus , Leandro Frazzon e William Azevedo

Objetivo :

O projeto tem proposta usar os recursos da placa SAMD21, em conjunto com o periférico BLTC1000. Esse periférico permite que a SAMD21 trate sinais de Bluetooth. O programa desenvolvido define o comportamento quando um botão é pressionado em um dispositivo externo, como um celular para causar um sinal de alerta em um terminal no computador.

Características do Projeto :



Figura 1: Ilustração do funcionamento do projeto em alto nível

A placa SAMD21, da Atmel, será utilizada por permitir o tratamento de sinais Bluetooth via código. O periférico BTLC1000, é do tipo BLE (*Bluetooth Low Energy*). Essa tecnologia tem como característica ser um modelo mais económico e mais eficiente em termos de consumo energético do que a tecnologia Bluetooth padrão. Suas características são:

- Conexão sem fios de curto alcance cobrindo uma área num raio de 10 metros,
- Menor consumo energético

- Transferência de dados entre equipamentos da mesma tecnologia e compatível ao padrão Bluetooth
- Implantação de novas utilizações em dispositivos de menor autonomia energética.

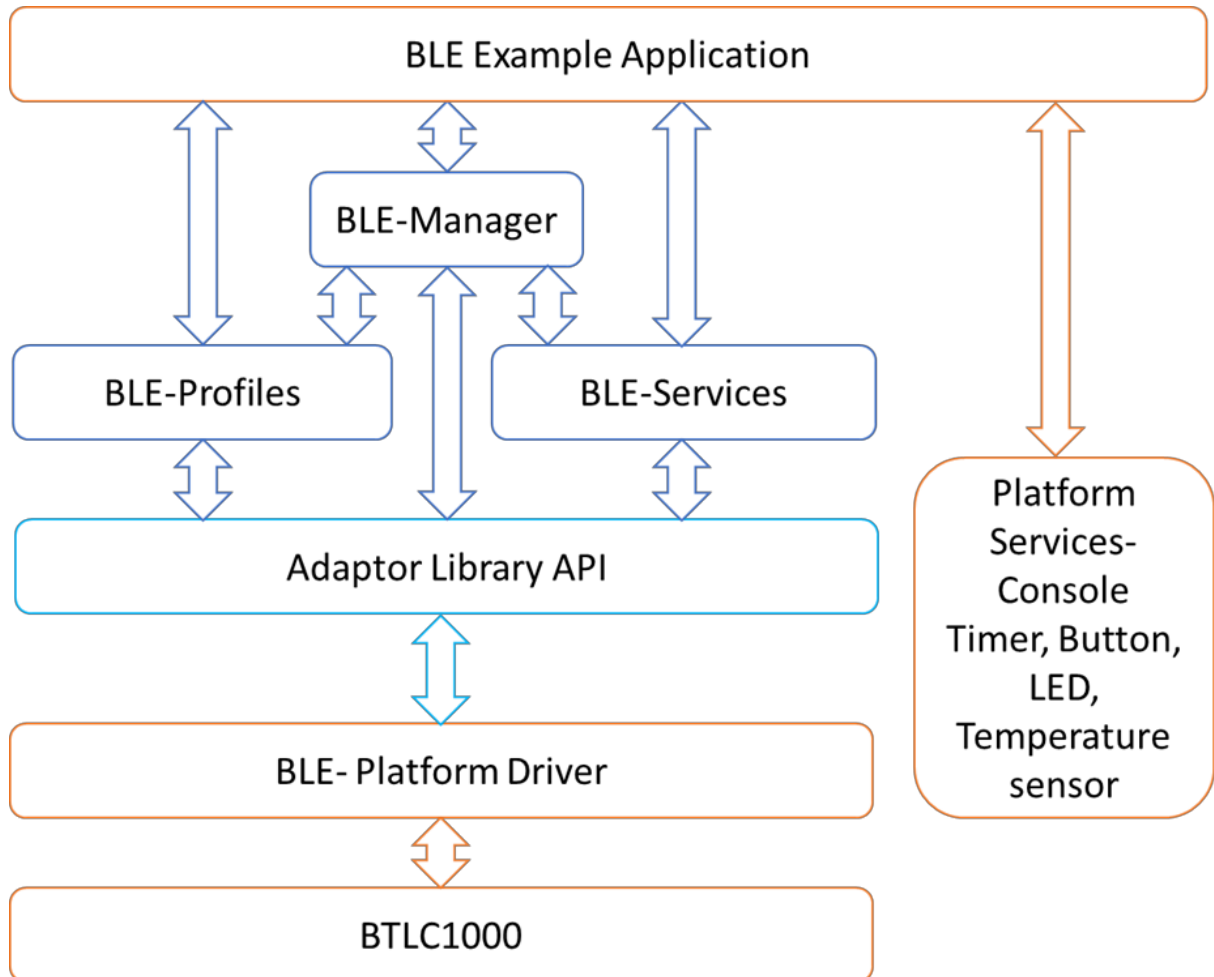


Figura 2: Arquitetura da comunicação entre o BLTC1000 e a placa SAMD21 (*BLE Example Application*)

Por ter um consumo menor de energia, possibilita que novas aplicações sejam utilizadas em dispositivos de tamanho reduzido, nomeadamente relógios de pulso, brinquedos, sensores desportivos, dispositivos de monitorização médica, alarmes e sensores automotivos além da troca de arquivos entre aparelhos diferentes, como PCs, celulares e impressoras.

O programa define o comportamento quando um botão é pressionado em um dispositivo externo para causar um alerta imediato em um terminal. Isso pode ser usado para permitir que os usuários encontrem dispositivos que foram extraviados.

Quando o dispositivo externo deseja causar um alerta no terminal, ele deve indicar o nível de alerta (alto, médio ou baixo). Esse nível é gravado na memória da placa SAMD21, e, toda vez que o programa é reexecutado, o último sinal gravado na memória é mostrado.



Figura 3: placa SAMD21 (acima) com o BTLC1000 conectado (abaixo)

Metodologia :

A metodologia utilizada foi o Test Driven Development (TDD). Cada nova funcionalidade inicia com a criação de um teste. Este teste precisa inevitavelmente falhar porque ele é escrito antes da funcionalidade a ser implementada (se ele falha, então a funcionalidade ou melhoria proposta é óbvia).

Para escrever um teste, o desenvolvedor precisa claramente entender as especificações e requisitos da funcionalidade. O desenvolvedor pode fazer isso através de casos de uso ou user stories que cubram os requisitos e exceções condicionais. Ele torna o desenvolvedor focado nos requisitos antes do código, que é uma sutil mas importante diferença.

Desenvolvimento :

O desenvolvimento do código foi feito com base em um exemplo já existente no Atmel Studio. A partir dele, fomos adicionando novas funções (acesso à memória) e otimizações,

como tornar as configurações menos dependentes de bibliotecas do *ASF Wizard*, por exemplo.

A estrutura do código foi feita utilizando o mecanismo de protothreads. Uma protothread é um mecanismo de baixo custo para programação concorrente.

Protothreads são usadas para realizar uma forma de concorrência não antecipada, conhecida como multitarefa cooperativa e, portanto, não incorre em mudança de contexto ao renderizar-se a outra linha.

Isso permite saltar (retomar) de um retorno em outra chamada de função. Para bloquear threads, esses rendimentos podem ser protegidos por um condicional, de modo que as chamadas sucessivas para a mesma função retorne, a menos que o condicional de proteção seja verdadeiro. Como as protothreads são realizadas por chamadas sucessivas a uma função, eles devem manter seu estado através do uso de variáveis externas, muitas vezes globais.

```
button_init();

/* Inicializa a comunicação serial */
serial_console_init();

/* Inicializa o temporizador */
hw_timer_init();

/* Registra a interrupção */
hw_timer_register_callback(timer_callback_handler);

DBG_LOG("Initializing Find Me Application");

/* inicializa o Bluetooth da placa e seta o endereço do dispositivo no computador */
ble_device_init(NULL);

fmp_target_init(NULL);

/* registro da interrupção para o alerta imediato*/
register_find_me_handler(app_immediate_alert);

/* execução*/
while (!) {
    /* Tarefa do dispositivo Bluetooth */
    ble_event_task();

    /* Tratamento de interrupção */
    if (app_timer_done) {
        LED_Toggle(LED0);
        hw_timer_start(timer_interval);
        app_timer_done = false;
    }
}
```

Figura 4: código original, disponível no exemplo.

A figura 5, a seguir, mostra as alterações feitas no código.

```

#include "asf.h"
#include "usart.h"
#include "platform.h"
#include "timer_hw.h"
#include "tc_interrupt.h"
#include "conf_timer.h"
#include "conf_extint.h"
#include "ble_manager.h"
#include "immediate_alert.h"
#include "find_me_app.h"
#include "find_me_target.h"
#include "pt.h"

/* === MACROS ===== */
//Função de configuração do timer.
void tc_cc0_cb(struct tc_module *const module_inst);

void configure_eeprom(void);

//struct que guarda a configuração da porta serial.
static struct usart_module cdc_uart_module;

//struct que guarda a configuração do timer.
struct tc_config config_tc;

struct usart_config usart_conf;

//interrupções aceitas pelo dispositivo na aplicação
static const ble_event_callback_t fmp_gap_handle[] = {
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    fmp_target_connected_state_handler,
    fmp_target_disconnect_event_handler,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL
};

//interrupções aceitas pelo dispositivo na aplicação
static const ble_event_callback_t fmp_gatt_server_handle[] = {
    NULL,
    NULL,
    fmp_target_char_changed_handler,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL
};
};

//recebe o sinal alto ou médio do LED.
uint32_t timeout_count;

//recebe a interrupção do timer.
hw_timer_callback_t timer_callback;

//variável que trata dos eventos do dispositivo Bluetooth.
at_ble_events_t event;

//variável que trata dos eventos do dispositivo Bluetooth.
uint8_t ble_event_params[524];

```

```
///variável que recebe o último alerta dado por um dispositivo externo ao periférico.
uint8_t last_alert = 0;
```

```
///variável de comunicação entre a memória e o programa.
uint8_t page_data[EEPROM_PAGE_SIZE];
```

```
///variável de comunicação entre a protothread e a função Main
volatile char i = 0;
///variável de comunicação entre a protothread e a função Main
volatile char buffer;
```

```
///variável que especifica o uso do dispositivo Bluetooth para a função Find Me
gatt_service_handler_t ias_handle;
```

```
///Flag da tarefa do Timer
volatile bool app_timer_done = false;
/// Flag da contagem de tempo
static uint8_t timer_interval = INIT_TIMER_INTERVAL;
```

```
///Interrupção do serviço de alerta imediato
find_me_callback_t immediate_alert_cb;
```

```
void configure_eeprom(void)
```

```
{
    enum status_code error_code = eeprom_emulator_init();
    if (error_code == STATUS_ERR_NO_MEMORY) {
        while (true) {
        }
    }
    else if (error_code != STATUS_OK) {
        printf("Memory error!!!\n");
        eeprom_emulator_erase_memory();
        eeprom_emulator_init();
    }
}
```

```
/**
 * @fn void configure_bod(void)
 * @brief Configuração da interrupção da memória EEPROM, juntamente com a função SYSCTRL_Handler(void).
```

```
*/
#ifdef (SAMD || SAMR21)
void SYSCTRL_Handler(void)
{
    if (SYSCTRL->INTFLAG.reg & SYSCTRL_INTFLAG_BOD33DET) {
        SYSCTRL->INTFLAG.reg = SYSCTRL_INTFLAG_BOD33DET;
        eeprom_emulator_commit_page_buffer();
    }
}
#endif
```

```
static void configure_bod(void)
```

```
{
    #if (SAMD || SAMR21)
    struct bod_config config_bod33;
    bod_get_config_defaults(&config_bod33);
    config_bod33.action = BOD_ACTION_INTERRUPT;
    config_bod33.level = 48;
    bod_set_config(BOD_BOD33, &config_bod33);
    bod_enable(BOD_BOD33);

    SYSCTRL->INTENSET.reg = SYSCTRL_INTENCLR_BOD33DET;
    system_interrupt_enable(SYSTEM_INTERRUPT_MODULE_SYSCTRL);
    #endif
}
```

```
/**
 * @fn void timer_callback_handler(void)
 * @brief Tratamento da interrupção do timer.
```

```

O timer é desabilitado.
Se app_timer_done é true, é uma indicação de que nenhum dispositivo foi conectado no tempo possível, ou não conseguiu.
Então, ele reinicia no modo Advertising Mode.
*/
```

```
static void timer_callback_handler(void)
```

```
{
    tc_disable_callback(&tc_instance, TC_CALLBACK_CC_CHANNEL0);
    app_timer_done = true;
}
```



```

/**/
static void app_immediate_alert(uint8_t alert_val)
{
    if (alert_val == IAS_HIGH_ALERT) {
        DBG_LOG("Find Me : High Alert");
        LED_On(LED0);
        last_alert = 2;
        timeout_count = LED_FAST_INTERVAL;
        tc_set_count_value(&tc_instance, 0);
        tc_enable_callback(&tc_instance, TC_CALLBACK_CC_CHANNEL0);

    } else if (alert_val == IAS_MID_ALERT) {
        DBG_LOG("Find Me : Mild Alert");
        LED_On(LED0);
        last_alert = 1;
        timeout_count = LED_MILD_INTERVAL;
        tc_set_count_value(&tc_instance, 0);
        tc_enable_callback(&tc_instance, TC_CALLBACK_CC_CHANNEL0);

    } else if (alert_val == IAS_NO_ALERT) {
        DBG_LOG("Find Me : No Alert");
        last_alert = 0;
        tc_disable_callback(&tc_instance, TC_CALLBACK_CC_CHANNEL0);
        LED_Off(LED0);
    }
    page_data[0] = last_alert;
    eeprom_emulator_write_page(0, page_data);
    eeprom_emulator_commit_page_buffer();
}

PT_THREAD(pt_find_me(struct pt *pt, char data)){
    PT_BEGIN(pt);
    buffer = i;
    PT_WAIT_UNTIL(pt, buffer == 0);
    system_init();
    PT_YIELD(pt);

    buffer = i;
    PT_WAIT_UNTIL(pt, buffer == 1);
    usart_get_config_defaults(&usart_conf);
    usart_conf.mux_setting = USART_RX_1_TX_0_XCK_1;
    usart_conf.pinmux_pad0 = PINMUX_PA22C_SERCOM3_PAD0;
    usart_conf.pinmux_pad1 = PINMUX_PA23C_SERCOM3_PAD1;
    usart_conf.pinmux_pad2 = PINMUX_UNUSED;
    usart_conf.pinmux_pad3 = PINMUX_UNUSED;
    usart_conf.baudrate = 115200;
    stdio_serial_init(&cdc_uart_module, SERCOM3, &usart_conf);
    usart_enable(&cdc_uart_module);
    PT_YIELD(pt);

    buffer = i;
    PT_WAIT_UNTIL(pt, buffer == 2);
    tc_get_config_defaults(&config_tc);
    config_tc.counter_size = TC_CTRLA_MODE_COUNT32;
    config_tc.clock_source = GCLK_GENERATOR_0;
    config_tc.clock_prescaler = TC_CTRLA_PRESCALER(7);
    config_tc.counter_8_bit.period = 0;
    config_tc.counter_32_bit.compare_capture_channel[0] = (48000000ul/1024ul);
    config_tc.counter_32_bit.compare_capture_channel[1] = 0xFFFF;
    tc_init(&tc_instance, TC3, &config_tc);
    tc_enable(&tc_instance);
    tc_register_callback(&tc_instance, tc_cc0_cb, TC_CALLBACK_CC_CHANNEL0);

    timer_callback = timer_callback_handler;
    PT_YIELD(pt);

    buffer = i;
    PT_WAIT_UNTIL(pt, buffer == 3);
    configure_eeprom();
    configure_bod();
    PT_YIELD(pt);
}

```

```

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 4);
DBG_LOG("Initializing Find Me Application");
ble_device_init(NULL);
PT_YIELD(pt);

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 5);
eeprom_emulator_read_page(0, page_data);
last_alert = page_data[0];
if(last_alert == 2){
    DBG_LOG("Last Alert: High Alert!");
}
else if(last_alert == 1){
    DBG_LOG("Last Alert: Mild Alert!");
}
else{
    DBG_LOG("Last Alert: No Alert!");
}
PT_YIELD(pt);

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 6);
init_immediate_alert_service(&ias_handle);
ias_primary_service_define(&ias_handle);
DBG_LOG("The Supported Services in Find Me Profile are:");
DBG_LOG(" -> Immediate Alert Service");
PT_YIELD(pt);

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 7);
if(!(ble_advertisement_data_set() == AT_BLE_SUCCESS))
{
    DBG_LOG("Fail to set Advertisement data");
}

if (at_ble_adv_start(AT_BLE_ADV_TYPE_UNDIRECTED,
AT_BLE_ADV_GEN_DISCOVERABLE, NULL, AT_BLE_ADV_FP_ANY,
APP_FMP_FAST_ADV, APP_FMP_ADV_TIMEOUT,
0) == AT_BLE_SUCCESS) {
    DBG_LOG("Bluetooth device is in Advertising Mode");
} else {
    DBG_LOG("BLE Adv start Failed");
}
PT_YIELD(pt);

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 8);
ble_mgr_events_callback_handler(REGISTER_CALL_BACK, BLE_GAP_EVENT_TYPE, fmp_gap_handle);
ble_mgr_events_callback_handler(REGISTER_CALL_BACK, BLE_GATT_SERVER_EVENT_TYPE, fmp_gatt_server_handle);
immediate_alert_cb = app_immediate_alert;
PT_YIELD(pt);

buffer = i;
PT_WAIT_UNTIL(pt, buffer == 9);
while (1) {
    if (at_ble_event_get(&event, ble_event_params, BLE_EVENT_TIMEOUT) == AT_BLE_SUCCESS)
    {
        ble_event_manager(event, ble_event_params);
    }

    if (app_timer_done) {
        LED_Toggle(LED0);
        timeout_count = timer_interval;
        tc_set_count_value(&tc_instance, 0);
        tc_enable_callback(&tc_instance, TC_CALLBACK_CC_CHANNEL0);
        app_timer_done = false;
    }
}
PT_YIELD(pt);
PT_END(pt);
}

```



```

int main(void)
{
    PT_INIT(&pt);
    i = 0;
    for(i = 0; i < 10; i++){
        pt_find_me(&pt, i);
    }
    return 0;
}

```

Figura 5: modificações feitas no código.

Maiores detalhes sobre o funcionamento do código podem ser vistos na documentação própria dele.

Execução



Figura 6: Placa SAMD21 com o periférico BLTC1000 conectado.

A comunicação Bluetooth, como outras comunicações wireless, não muito estável, havendo com certa frequência a queda do sinal, como foi verificado em diversas execuções do programa. Para a execução do programa, é usado o terminal TeraTerm. Antes da execução, o terminal deve ser configurado com *baudrate* de 115200 e setado na porta correspondente a placa SAMD21. Na execução para este relatório, a porta usada foi a porta COM7, como visto na figura 7.

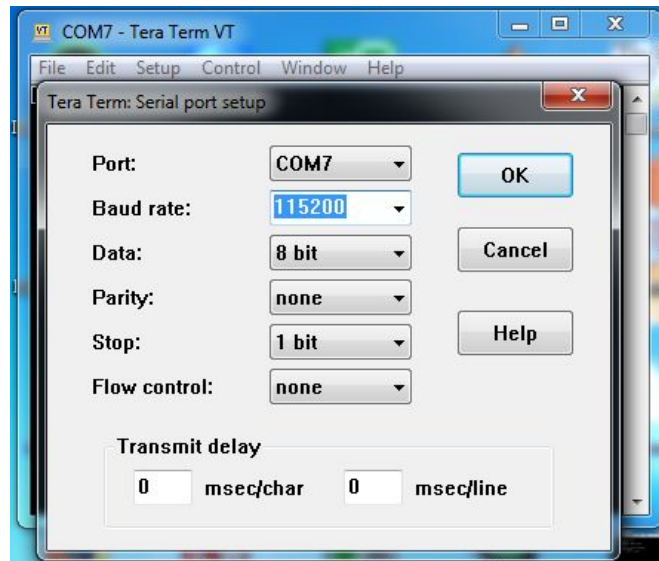


Figura 7: Configuração do TeraTerm.

Quando o programa é inicializado, ele mostra no terminal o último alerta dado em uma execução anterior. Neste caso foi um de nível *High*, como visto na figura 8.

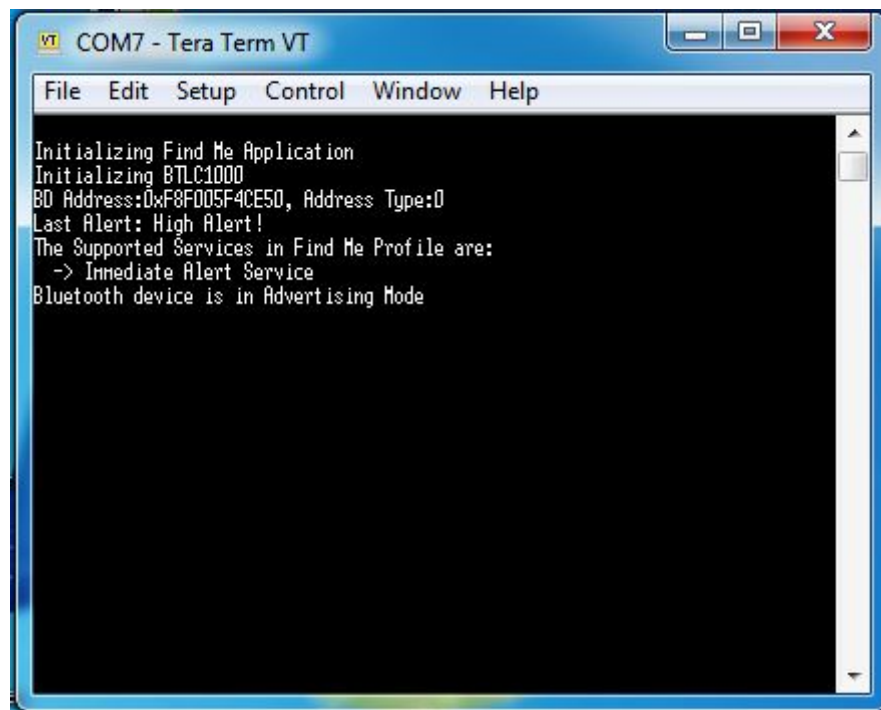


Figura 8: Execução do programa, mostrando o último alerta salvo na EEPROM, do tipo *High*.

Em um celular, é necessário rastrear o BLTC1000 para fazer a conexão, então utilizamos o *Atmel SmartConnect* para isso devido à compatibilidade com a placa. O sinal emitido pelo periférico tem o nome de ATMEL-FTP, como visto na figura 9.

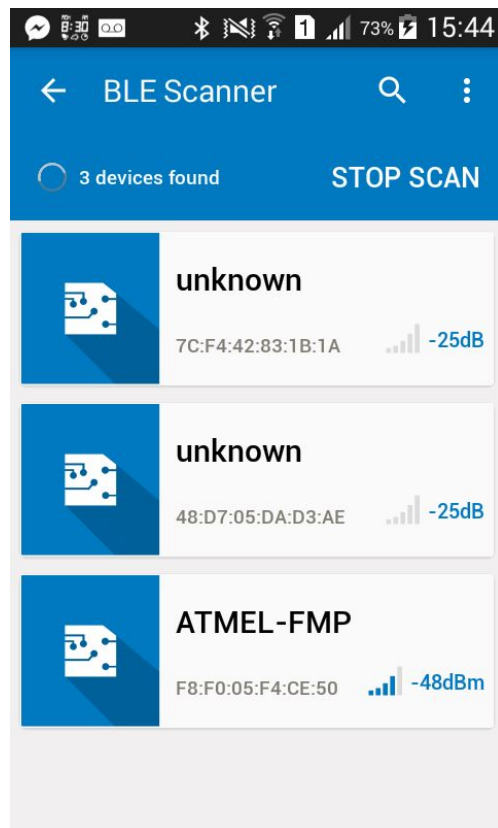


Figura 9: Sinais rastreados pelo *Atmel SmartConnect*, no celular.

Após o pareamento, o celular passa a mandar sinais, através da interface, como visto na figura 10.

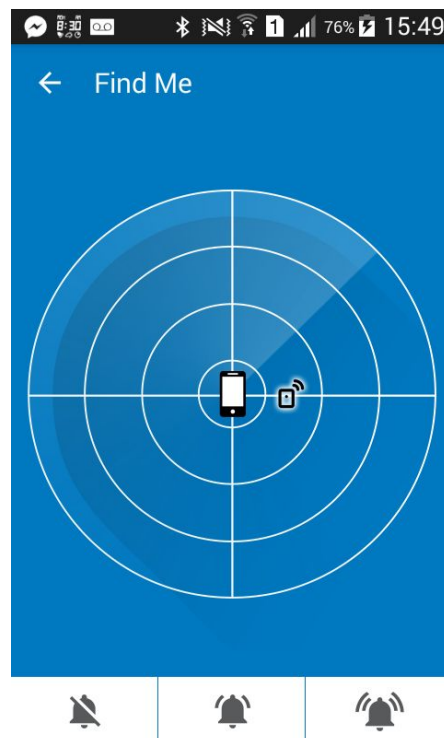
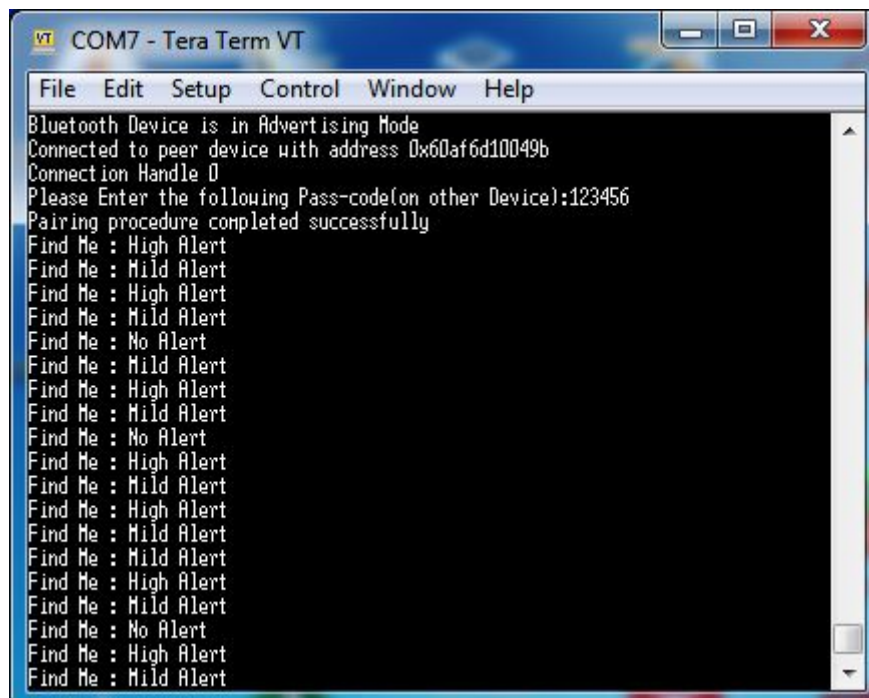


Figura 10: Celular disponível para mandar alertas ao dispositivo, com os botões de *High*, *Mild* e *No* na parte inferior.

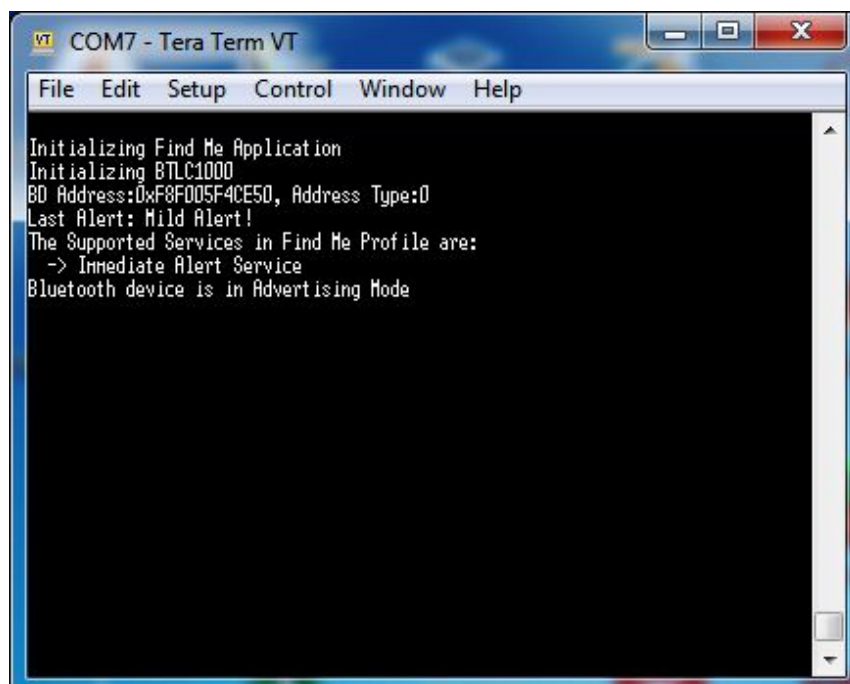
A cada sinal disparado no celular, seu nível (*High*, *Mild* ou *No*) era mostrado no terminal, como visto na figura 11.



```
VT COM7 - Tera Term VT
File Edit Setup Control Window Help
Bluetooth Device is in Advertising Mode
Connected to peer device with address 0x60af6d10049b
Connection Handle 0
Please Enter the following Pass-code(on other Device):123456
Pairing procedure completed successfully
Find Me : High Alert
Find Me : Mild Alert
Find Me : High Alert
Find Me : Mild Alert
Find Me : No Alert
Find Me : Mild Alert
Find Me : High Alert
Find Me : Mild Alert
Find Me : No Alert
Find Me : High Alert
Find Me : Mild Alert
Find Me : High Alert
Find Me : Mild Alert
Find Me : Mild Alert
Find Me : High Alert
Find Me : Mild Alert
Find Me : No Alert
Find Me : High Alert
Find Me : Mild Alert
```

Figura 11: Sinais emitidos pelo celular no terminal.

O último sinal dado na execução na figura 10 foi um sinal Mild. Quando o programa é executado novamente, esse último sinal é lido da EEPROM e é mostrado no terminal, como visto na figura 12.



```
VT COM7 - Tera Term VT
File Edit Setup Control Window Help
Initializing Find Me Application
Initializing BTLC1000
BD Address:0xF8F005F4CE50, Address Type:0
Last Alert: Mild Alert!
The Supported Services in Find Me Profile are:
-> Immediate Alert Service
Bluetooth device is in Advertising Mode
```

Figura 12: Nova execução do programa, mostrando o último alerta salvo na EEPROM, do tipo *Mild*.

Conclusão

O dispositivo se comportou de forma esperada. No entanto, devido a reformulação do código e devido a instabilidade da comunicação Bluetooth, houve um pequeno “atraso”. A atuação do programa TeraTerm foi satisfatória, mostrando os alertas dados pelos dispositivos externos à placa quase ao mesmo tempo. A memória EEPROM também salvou os avisos dados de maneira eficaz.

Link para GitHub

https://github.com/FelipeSeitenfus/Dispositivo_Bluetooth