#include "led\_control.h"

#include <zephyr/kernel.h>

#include <zephyr/drivers/gpio.h>

#include <zephyr/logging/log.h>

LOG\_MODULE\_REGISTER(main, LOG\_LEVEL\_DBG);

static const struct gpio\_dt\_spec buttons[] = {

GPIO\_DT\_SPEC\_GET(DT\_NODELABEL(button0), gpios),

GPIO\_DT\_SPEC\_GET(DT\_NODELABEL(button1), gpios),

GPIO\_DT\_SPEC\_GET(DT\_NODELABEL(button2), gpios),

GPIO\_DT\_SPEC\_GET(DT\_NODELABEL(button3), gpios)

};

static struct gpio\_callback cb\_data;

static void button\_callback(const struct device \*dev,

struct gpio\_callback \*cb, uint32\_t pins)

{

LOG\_INF("Pin mask 0x%08x", pins);

if (pins & BIT(buttons[0].pin)) {

led\_send((led\_msg){ LED\_STATE, 0 });

} else if (pins & BIT(buttons[1].pin)) {

led\_send((led\_msg){ LED\_STATE, 1 });

} else if (pins & BIT(buttons[2].pin)) {

led\_send((led\_msg){ LED\_BLINK, 500 });

} else if (pins & BIT(buttons[3].pin)) {

led\_send((led\_msg){ LED\_BLINK, 100 });

}

}

int main(void) {

gpio\_init\_callback(&cb\_data, button\_callback,

BIT(buttons[0].pin) | BIT(buttons[1].pin) |

BIT(buttons[2].pin) | BIT(buttons[3].pin));

for (int i = 0; i < ARRAY\_SIZE(buttons); ++i) {

gpio\_pin\_configure\_dt(&buttons[i], GPIO\_INPUT);

gpio\_pin\_interrupt\_configure\_dt(&buttons[i],

GPIO\_INT\_EDGE\_FALLING);

gpio\_add\_callback(buttons[i].port, &cb\_data);

}

int counter = 0;

while (1) {

LOG\_INF("Tick %d", counter++);

k\_msleep(1000);

}

return 0;

}

Listing 1. Plik src/main.c

#pragma once

typedef struct {

enum {

LED\_STATE, // param: 0 (off) or 1 (on)

LED\_BLINK // param: time [ms] for every toggle

} command;

unsigned int param;

} led\_msg;

void led\_send(led\_msg msg);

Listing 2. Plik src/led\_control.h

#include "led\_control.h"

#include <zephyr/kernel.h>

#include <zephyr/drivers/gpio.h>

#include <zephyr/logging/log.h>

LOG\_MODULE\_REGISTER(led\_control, LOG\_LEVEL\_DBG);

static const struct gpio\_dt\_spec led =

GPIO\_DT\_SPEC\_GET(DT\_NODELABEL(led0), gpios);

K\_MSGQ\_DEFINE(led\_queue, sizeof(led\_msg), 10, 1);

void led\_send(led\_msg msg) {

k\_msgq\_put(&led\_queue, &msg, K\_NO\_WAIT);

}

static void led\_worker(void \*a, void \*b, void \*c) {

gpio\_pin\_configure\_dt(&led, GPIO\_OUTPUT\_ACTIVE);

led\_msg msg;

k\_timeout\_t timeout = K\_FOREVER;

while (1) {

if (k\_msgq\_get(&led\_queue, &msg, timeout) == 0) {

if (msg.command == LED\_STATE) {

LOG\_INF("LED\_STATE %d received", msg.param);

gpio\_pin\_set\_dt(&led, msg.param);

timeout = K\_FOREVER;

} else if (msg.command == LED\_BLINK) {

LOG\_INF("LED\_BLINK %dms received", msg.param);

timeout = K\_MSEC(msg.param);

}

} else {

gpio\_pin\_toggle\_dt(&led);

}

}

}

K\_THREAD\_DEFINE(led\_thread, 500, led\_worker, NULL, NULL, NULL,

K\_HIGHEST\_THREAD\_PRIO, 0, 0);

#ifdef CONFIG\_SHELL

#include <zephyr/shell/shell.h>

#include <stdlib.h>

static int cmd\_led\_on(const struct shell \*shell, size\_t argc, char \*\*argv) {

led\_send((led\_msg){ LED\_STATE, 1 });

shell\_print(shell, "LED on");

return 0;

}

static int cmd\_led\_off(const struct shell \*shell, size\_t argc, char \*\*argv) {

led\_send((led\_msg){ LED\_STATE, 0 });

shell\_print(shell, "LED off");

return 0;

}

static int cmd\_led\_blink(const struct shell \*shell, size\_t argc, char \*\*argv) {

if (argc != 2) {

shell\_print(shell, "led blink <time in ms>");

return -EINVAL;

}

uint32\_t period = strtoul(argv[1], NULL, 10);

led\_send((led\_msg){ LED\_BLINK, period });

shell\_print(shell, "LED blinking period %d ms:",period);

return 0;

}

SHELL\_STATIC\_SUBCMD\_SET\_CREATE(led\_menu,

SHELL\_CMD(on, NULL, "LED on", cmd\_led\_on),

SHELL\_CMD(off, NULL, "LED off", cmd\_led\_off),

SHELL\_CMD(blink, NULL, "LED blink with period [ms]", cmd\_led\_blink),

SHELL\_SUBCMD\_SET\_END

);

SHELL\_CMD\_REGISTER(led, &led\_menu, "LED", NULL);

#endif

Listing 3. Plik src/led\_control.c

cmake\_minimum\_required(VERSION 3.20.0)

find\_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR\_BASE})

project(thread\_test)

target\_sources(app PRIVATE

src/main.c

src/led\_control.c

)

Listing 4. Plik /CMakeLists.txt

CONFIG\_GPIO=y

CONFIG\_SERIAL=y

CONFIG\_CONSOLE=y

CONFIG\_UART\_CONSOLE=y

CONFIG\_LOG=y

CONFIG\_LOG\_PROCESS\_THREAD\_SLEEP\_MS=100

CONFIG\_USE\_SEGGER\_RTT=y

CONFIG\_LOG\_BACKEND\_RTT=y

CONFIG\_SHELL=y

CONFIG\_SHELL\_BACKEND\_SERIAL=y

CONFIG\_GPIO\_SHELL=y

Listing 5. Plik /prj.conf

CONFIG\_BT=y

CONFIG\_BT\_PERIPHERAL=y

CONFIG\_BT\_DEVICE\_NAME="Led control"

Listing 6. Włączanie BLE w pliku prj.conf

#include <zephyr/bluetooth/bluetooth.h>

#include <zephyr/bluetooth/conn.h>

#include "bt\_control.h"

#include <zephyr/logging/log.h>

LOG\_MODULE\_REGISTER(bt\_control, LOG\_LEVEL\_DBG);

// advertisement data (AdvData)

static const struct bt\_data adv\_data[] = {

BT\_DATA\_BYTES(BT\_DATA\_FLAGS, (BT\_LE\_AD\_GENERAL | BT\_LE\_AD\_NO\_BREDR)),

BT\_DATA(BT\_DATA\_NAME\_COMPLETE, CONFIG\_BT\_DEVICE\_NAME, (sizeof(CONFIG\_BT\_DEVICE\_NAME)-1)),

};

static void connected\_cb(struct bt\_conn \*conn, uint8\_t err)

{

if (err) {

LOG\_ERR("Connection failed. error:%u", err);

return;

}

struct bt\_conn\_info info;

int ret = bt\_conn\_get\_info(conn, &info);

if (ret == 0) {

char addr\_dst[BT\_ADDR\_LE\_STR\_LEN];

bt\_addr\_le\_to\_str(info.le.dst, addr\_dst, sizeof(addr\_dst));

LOG\_INF("Connected to: %s", addr\_dst);

} else {

LOG\_ERR("MAC read failed");

}

}

static void disconnected\_cb(struct bt\_conn \*conn, uint8\_t reason)

{

LOG\_INF("Disconnected. Reason: %u", reason);

}

static struct bt\_conn\_cb conn\_callbacks = {

.connected = connected\_cb,

.disconnected = disconnected\_cb

};

void bt\_ready\_cb(int status) {

//MAC

bt\_addr\_le\_t addrs[CONFIG\_BT\_ID\_MAX];

size\_t count = CONFIG\_BT\_ID\_MAX;

bt\_id\_get(addrs, &count);

//print MAC

if (count > 0) {

char addr\_str[BT\_ADDR\_LE\_STR\_LEN];

bt\_addr\_le\_to\_str(&addrs[0], addr\_str, sizeof(addr\_str));

LOG\_DBG("BLE MAC: %s", addr\_str);

} else {

LOG\_ERR("Error while getting MAC");

}

//Start advertising

int err = bt\_le\_adv\_start(BT\_LE\_ADV\_CONN, adv\_data, ARRAY\_SIZE(adv\_data), NULL, 0);

if (err) {

LOG\_ERR("BLE advertisement error: %d", err);

} else {

LOG\_DBG("BLE advertisement started");

}

//Register connected/disconnected callbacks

bt\_conn\_cb\_register(&conn\_callbacks);

}

int bt\_control\_init(void) {

LOG\_DBG("BLE init...");

int ret = bt\_enable(bt\_ready\_cb);

if (ret) {

LOG\_ERR("BLE init error: %d\n", ret);

return ret;

}

return ret;

}

Listing 7. Plik src/bt\_control.c

#pragma once

int bt\_control\_init(void);

Listing 8. Plik nagłówkowy src/bt\_control.h

[00:00:03.084,899] <dbg> bt\_control: bt\_control\_init: BLE init...

[00:00:03.086,303] <inf> main: Tick 0

[00:00:03.097,869] <inf> bt\_hci\_core: HW Platform: Nordic Semiconductor (0x0002)

[00:00:03.097,900] <inf> bt\_hci\_core: HW Variant: nRF53x (0x0003)

[00:00:03.097,900] <inf> bt\_hci\_core: Firmware: Standard Bluetooth controller (0x00) Version 54.58864 Build 1214809870

[00:00:03.099,426] <inf> bt\_hci\_core: Identity: F5:20:E7:1F:38:EF (random)

[00:00:03.099,456] <inf> bt\_hci\_core: HCI: version 5.4 (0x0d) revision 0x218f, manufacturer 0x0059

[00:00:03.099,487] <inf> bt\_hci\_core: LMP: version 5.4 (0x0d) subver 0x218f

[00:00:06.844,604] <dbg> bt\_control: bt\_ready\_cb: BLE MAC: F5:20:E7:1F:38:EF (random)

[00:00:06.846,313] <dbg> bt\_control: bt\_ready\_cb: BLE advertisement started

Listing 9. Log procesu włączania i rozpoczynanie rozgłaszania.

#include <zephyr/bluetooth/bluetooth.h>

#include <zephyr/bluetooth/uuid.h>

#include <zephyr/bluetooth/gatt.h>

#include "bt\_led\_svc.h"

#include <zephyr/logging/log.h>

LOG\_MODULE\_REGISTER(bt\_led\_svc, LOG\_LEVEL\_DBG);

// UUID for LED Service

#define BT\_UUID\_LED\_SERVICE BT\_UUID\_DECLARE\_128 \

(BT\_UUID\_128\_ENCODE(0x7c8482f0, 0x1104, 0x44b8, 0xa354, 0x3f5b8488083d))

// UUID for LED Characteristics

#define BT\_UUID\_LED\_STATE\_CHR BT\_UUID\_DECLARE\_128 \

(BT\_UUID\_128\_ENCODE(0x7c8482f1, 0x1104, 0x44b8, 0xa354, 0x3f5b8488083d))

#define BT\_UUID\_LED\_BLINK\_CHR BT\_UUID\_DECLARE\_128 \

(BT\_UUID\_128\_ENCODE(0x7c8482f2, 0x1104, 0x44b8, 0xa354, 0x3f5b8488083d))

// Callback function pointers for BLE write operations

static led\_cb\_t write\_state\_cb = NULL;

static led\_cb\_t write\_blink\_cb = NULL;

static ssize\_t write\_chr\_cb(struct bt\_conn \*conn,

const struct bt\_gatt\_attr \*attr,

const void \*buf, uint16\_t len,

uint16\_t offset, uint8\_t flags) {

LOG\_HEXDUMP\_INF(buf, len, "Received buffer:");

if (len != sizeof(uint32\_t)) {

return BT\_GATT\_ERR(BT\_ATT\_ERR\_INVALID\_ATTRIBUTE\_LEN);

}

unsigned int received = \*(unsigned int\*)buf;

if (bt\_uuid\_cmp(attr->uuid,BT\_UUID\_LED\_STATE\_CHR) == 0) {

LOG\_INF("Led state update: %u", received);

if (write\_state\_cb != NULL) {

write\_state\_cb(received);

}

}

if (bt\_uuid\_cmp(attr->uuid,BT\_UUID\_LED\_BLINK\_CHR) == 0) {

LOG\_INF("Led blink update: %u", received);

if (write\_blink\_cb != NULL) {

write\_blink\_cb(received);

}

}

return len;

}

// Define the GATT service and characteristics

BT\_GATT\_SERVICE\_DEFINE(led\_service,

BT\_GATT\_PRIMARY\_SERVICE(BT\_UUID\_LED\_SERVICE),

BT\_GATT\_CHARACTERISTIC(BT\_UUID\_LED\_STATE\_CHR,

BT\_GATT\_CHRC\_WRITE, BT\_GATT\_PERM\_WRITE,

NULL, write\_chr\_cb, NULL),

BT\_GATT\_CHARACTERISTIC(BT\_UUID\_LED\_BLINK\_CHR,

BT\_GATT\_CHRC\_WRITE, BT\_GATT\_PERM\_WRITE,

NULL, write\_chr\_cb, NULL)

);

void bt\_led\_svc\_set\_callbacks(led\_cb\_t write\_state,

led\_cb\_t write\_blink) {

write\_state\_cb = write\_state;

write\_blink\_cb = write\_blink;

}

Listing 10. Plik src/bt\_led\_svc.c - serwis BLE

#pragma once

#include <zephyr/types.h>

typedef void (\*led\_cb\_t)(unsigned int param);

void bt\_led\_svc\_set\_callbacks(led\_cb\_t write\_state, led\_cb\_t write\_blink);

Listing 11. Plik src/bt\_led\_svc.h - serwis BLE

void led\_state\_set(unsigned int state) {

led\_send((led\_msg){ LED\_STATE, state});

}

void led\_blink\_set(unsigned int param) {

led\_send((led\_msg){ LED\_BLINK, param});

}

Listing 12. Nowe funkcje w main.c - funkcje przekazujące parametr do sterowania LED.

bt\_control\_init();

bt\_led\_svc\_set\_callbacks(led\_state\_set,led\_blink\_set);

Listing 13. Włączenie BLE i inicjacja serwisu LED.