# Table of Contents

**1. Programmable Peripheral Interconnect** ..... 2
***1.1. Using PPI with GPIO*** ..... 2
***1.2. Tasks to be done during the laboratory*** ..... 2

# 1. Programmable Peripheral Interconnect

Programmable Peripheral Interconnect (PPI) is the matrix that creates programmable connections between the internal peripherals of the microcontroller. It is possible to connect output signals from peripherals known as EVENTS with inputs of other peripherals known as TASKS. It is possible to connect the input and output of the same peripheral creating so-called SHORTCUT. PPI is used to create direct connections that transmit control signals without the need for program execution.

## 1.1. Using PPI with GPIO

This part of the laboratory shows how to use the PPI to send the activation information from the button (sw0) directly to the LED (led0). It is possible to set up the hardware connection between the button EVENT signal and the TASK signal for LED. While the output is controlled by the GPIOTE (GPIO Tasks and Events) mechanism the normal GPIO writes have no effect. The GPIOTE mechanism is not supported in Zephyr OS, but the driver (NRFX) which is provided directly by Nordic Semiconductor can be used in Zephyr-compatible software. Every connection between EVENT and TASK is done by defining the channel. The number of available channels is limited to 8. The example provided uses the GPIOTE to handle signalling. To make it work properly the config file must be changed, enabling the NRFX driver, and disabling the general GPIO driver:

```
CONFIG_NRFX_GPIOTE=y
CONFIG_GPIO=n
```

Additionally, we can use the logging mechanism to receive log messages from the software. To do this we enable logging with:

```
CONFIG_LOG=y
```

The resulting config file prj.conf looks like this:

```
CONFIG_GPIO=n
CONFIG_SERIAL=y
CONFIG_NRFX_GPIOTE=y
CONFIG_LOG=y
```

## 1.2. Tasks to be done during the laboratory

To do this part of the laboratory take the following steps:

1. Modify the prj.conf file.
2. Replace main.c with the code included below.
3. Modify the code to turn the LED on with one button, and off with another button.

To make point 3 of the exercise it is essential to know that one GPIO can have three TASK signals:

- SET - to write the high state
- CLEAR - to write the low state
- OUT - to specify behaviour in the configuration (set, clear or toggle).

The example provided uses the OUT task only with toggle configuration. In the exercise, while replacing the OUT task with SET and CLEAR there is no need to configure such tasks, but there is a need to create separate PPI channels, one for setting LED with one button, and another for clearing it with a second button. While creating channels you can use the functions for resolving the TASK's handlers addresses:

```
nrfx_gpiote_set_task_addr_get();
nrfx_gpiote_clr_task_addr_get();
```

The initial, unmodified code of the main.c file looks like this:

```
/*
 * Copyright (c) 2019 Nordic Semiconductor ASA
 *
 * SPDX-License-Identifier: Apache-2.0
 */
#include <zephyr.h>

#include <nrfx_gpiote.h>
#include <helpers/nrfx_gppi.h>
#if defined(DPPI_PRESENT)
#include <nrfx_dppi.h>
#else
#include <nrfx_ppi.h>
#endif

#include <logging/log.h>
LOG_MODULE_REGISTER(nrfx_sample, LOG_LEVEL_INF);

#define INPUT_PIN   DT_GPIO_PIN(DT_ALIAS(sw0), gpios)
#define INPUT_PIN1  DT_GPIO_PIN(DT_ALIAS(sw1), gpios)
#define OUTPUT_PIN  DT_GPIO_PIN(DT_ALIAS(led0), gpios)
#define OUTPUT_PIN1 DT_GPIO_PIN(DT_ALIAS(led1), gpios)

static void button_handler(nrfx_gpiote_pin_t pin, nrf_gpiote_polarity_t
action)
{
    LOG_INF("GPIO input event callback %d", pin);
}

void main(void)
{
    LOG_INF("nrfx_gpiote sample on %s", CONFIG_BOARD);

    nrfx_err_t err;

    /* Connect GPIOTE_0 IRQ to nrfx_gpiote_irq_handler */
```

```
    IRQ_CONNECT(DT_IRQN(DT_NODELABEL(gpiote)),
            DT_IRQ(DT_NODELABEL(gpiote), priority),
            nrfx_isr, nrfx_gpiote_irq_handler, 0);

    /* Initialize GPIOTE (the interrupt priority passed as the parameter
     * here is ignored, see nrfx_glue.h).
     */
    err = nrfx_gpiote_init(0);
    if (err != NRFX_SUCCESS) {
        LOG_ERR("nrfx_gpiote_init error: %08x", err);
        return;
    }

    nrfx_gpiote_in_config_t const in_config = {
        .sense = NRF_GPIOTE_POLARITY_HITOLO,
        .pull = NRF_GPIO_PIN_PULLUP,
        .is_watcher = false,
        .hi_accuracy = true,
        .skip_gpio_setup = false,
    };

    /* Initialize input pin to generate event on high to low transition
     * (falling edge) and call button_handler()
     */
    err = nrfx_gpiote_in_init(INPUT_PIN, &in_config, button_handler);
    if (err != NRFX_SUCCESS) {
        LOG_ERR("nrfx_gpiote_in_init error: %08x", err);
        return;
    }

    err = nrfx_gpiote_in_init(INPUT_PIN1, &in_config, button_handler);
    if (err != NRFX_SUCCESS) {
        LOG_ERR("nrfx_gpiote_in_init error: %08x", err);
        return;
    }

    nrfx_gpiote_out_config_t const out_config = {
        .action = NRF_GPIOTE_POLARITY_TOGGLE,
        //.action = NRF_GPIOTE_POLARITY_HITOLO,
        .init_state = 1,
        .task_pin = true,
    };

    nrfx_gpiote_out_config_t const out_config1 = {
        .action = NRF_GPIOTE_POLARITY_TOGGLE,
        //.action = NRF_GPIOTE_POLARITY_LOTOHI,
        .init_state = 0,
        .task_pin = true,
    };

    /* Initialize output pin. SET task will turn the LED on,
```

```
     * CLR will turn it off and OUT will toggle it.
     */
    err = nrfx_gpiote_out_init(OUTPUT_PIN, &out_config);
    if (err != NRFX_SUCCESS) {
        LOG_ERR("nrfx_gpiote_out_init error: %08x", err);
        return;
    }

    err = nrfx_gpiote_out_init(OUTPUT_PIN1, &out_config1);
    if (err != NRFX_SUCCESS) {
        LOG_ERR("nrfx_gpiote_out1_init error: %08x", err);
        return;
    }

    nrfx_gpiote_in_event_enable(INPUT_PIN, true);
    nrfx_gpiote_in_event_enable(INPUT_PIN1, true);
    nrfx_gpiote_out_task_enable(OUTPUT_PIN);
    nrfx_gpiote_out_task_enable(OUTPUT_PIN1);

    LOG_INF("nrfx_gpiote initialized %d", 0);

    /* Allocate a (D)PPI channel. */
#if defined(DPPI_PRESENT)
    uint8_t channel;
    err = nrfx_dppi_channel_alloc(&channel);
#else
    nrf_ppi_channel_t channel;
    nrf_ppi_channel_t channel1;
    err = nrfx_ppi_channel_alloc(&channel);
    err = nrfx_ppi_channel_alloc(&channel1);
#endif
    if (err != NRFX_SUCCESS) {
        LOG_ERR("(D)PPI channel allocation error: %08x", err);
        return;
    }

    /* Configure endpoints of the channel so that the input pin event is
     * connected with the output pin OUT task. This means that each time
     * the button is pressed, the LED pin will be toggled.
     */
    nrfx_gppi_channel_endpoints_setup(channel,
        nrfx_gpiote_in_event_addr_get(INPUT_PIN),
        nrfx_gpiote_out_task_addr_get(OUTPUT_PIN));

    nrfx_gppi_channel_endpoints_setup(channel1,
        nrfx_gpiote_in_event_addr_get(INPUT_PIN1),
        nrfx_gpiote_out_task_addr_get(OUTPUT_PIN1));

    /* Enable (D)PPI channel. */
#if defined(DPPI_PRESENT)
    err = nrfx_dppi_channel_enable(channel);
```

```
#else
    err = nrfx_ppi_channel_enable(channel);
    err = nrfx_ppi_channel_enable(channel1);
#endif
    if (err != NRFX_SUCCESS) {
        LOG_ERR("Failed to enable (D)PPI channel, error: %08x", err);
        return;
    }

    LOG_INF("(D)PPI configured, leaving main() %d", 0);
}
```

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
**https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:aih:lab:nrf52gpiote**

Last update: **2023/12/01 13:51**