Introduction to IoT

School Year 2023-2024

Valsalice

Course Structure

1. Introduction and Basics
2. Basic Data Types and Operators
3. Control Structures Pt. 1
4. Control Structures Pt. 2
5. Functions and Scope
6. Arrays

10. Preprocessor and Macros — **DEC 19**
11. Custom Data Types — **JAN 9**

7. Introduction to Contiki-NG and nRF52840
8. Sensing and Actuating with Contiki-NG
9. Timers and Concurrency — **DEC 12**
12. Basic Communication and Networking — **JAN 16**
13. Introduction to RPL and Network Routing — **JAN 23**
14. Advanced Protocols: TSCH and 6TiSCH — **JAN 30**
15. Advanced Topics in Wireless Communication — **FEB 6**
16. Reliable Data Transfer Challenge — **FEB 20** **FEB 27** **MAR 5**

■ = Core Topics   ▢ = Optional Topics

Valsalice

Open your Virtual Machines

1. Turn on your Laptops

2. Login to Windows using "User"

3. Open the **Virtual Box** program

4. Add a new Virtual Machine (**Ctrl + A**)

5. Open the **VirtualBox** folder   ⚠️ (**NOT** the .VirtualBox)

6. Select the **nRF52840LAB** file

7. Click **Start**
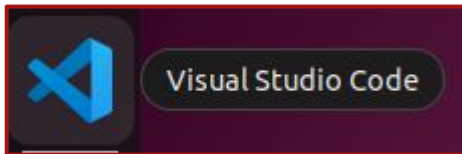
Prepare the Coding Environment

1 Start the Virtual Machine **nRF52840LAB**

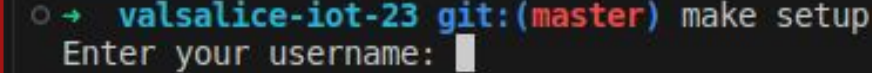**2** Log-in using credentials:

Username: **ubuntu**

Password: **ubuntu**
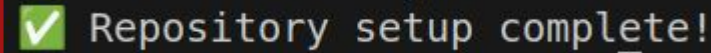
3 Open **Visual Studio Code** (use the App bar on the left)

Prepare the Coding Environment

4 From the Terminal:

```
make setup
```

5


```
○ → valsalice-iot-23 git:(master) make setup
  Enter your username: ▮
```

6


```
Password
```

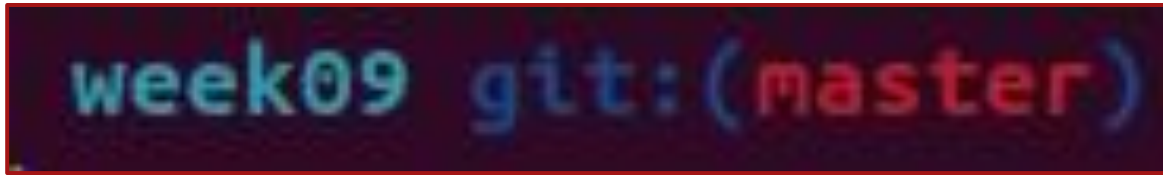**7**


```
✅ Repository setup complete!
```

⚠️ If you see **any (yellow) errors** input the credentials again

Prepare the Coding Environment

7 Open the **week09** folder in the terminal

Right click on the left + "**Open in Integrated terminal**"

8 You should see the following in the terminal:

Recap: Data Types

C has a number of primitive data types:

int   `42`   `1200`   `1_200`   `-3`

float   `3.14`   `0.00001`   `-2.1`

char   `'A'`   `'@'`   `'\n'`

bool   `true`   `false`

Strings are *NOT* a primitive data type, and have special syntax.

strings   `"Hello"`   `"A"`   `"I am a full sentence!"`

Recap: Variables

A variable is a named container that stores data or values.

```c
int x = 42;

float y = -0.12;

char w = 'A';

char z[50] = "Full sentence";
```

Booleans require a custom include statement:

```c
#include <stdbool.h>

bool hello = true;
```

Recap: Boolean Operators

Greater than	>
Greater or equal than	>=
Less than	<
Less or equal than	<=
Equals	==
Not equals	!=
Not	!

Recap: Chaining Comparisons

- **and** (both must be true)

  `true && false`   `(5 < 6) && (5 < 10)`

- **or** (either must be true)

  `true || false`   `(5 < 3) || (5 < 10)`

- **not** (negation)

  `!true`   `!(5 < 3)`

You can chain multiple conditions with **else if**.

What is the difference between these two snippets of code?

```c
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
} else if (num < 10) {
    printf("Medium number\n");
}
```

```c
int num;
scanf("%d", &num);

if (num < 3) {
    printf("Small number\n");
}
if (num < 10) {
    printf("Medium number\n");
}
```

Recap: While-Loops

Repeat parts of your code!

```c
int num;
printf("Input a number greater than 100: ");
scanf("%d", &num);

while (num <= 100) {
    printf("Wrong number, try again: ");
    scanf("%d", &num);
}


printf("Well done!\n");
```

Repeat a **<u>specific</u>** amount of times!

```c
int x;


for (x = 1; x <= 5; x++) {

    printf("Hello %d\n", x);

}
```

```c
int x = 0;


while (x < 5) {

    x += 1;

    printf("Hello %d\n", x);

}
```

Recap: Arrays

Modifiable containers for data.

With **variables**:

```c
int num1 = 42;
int num2 = 100;
int num3 = 10;

printf("%d\n", num1);
printf("%d\n", num2);
printf("%d\n", num3);
```

With a **list**:

```c
int array[] = {42, 100, 10};

for(int i = 0; i < 3; i++)
{
    printf("%d\n", array[i]);
}
```

Recap: Accessing Array Elements

To <u>access</u> array elements you can use the **[index]** operator.

NOTE: List indices start from **0**

index: | 0 | 1 | 2 | 3 | 4 |

```
int array[] = {17, 28, 33, 56, 6};
```

```
printf("%d\n", array[0]);
```

```
printf("%d\n", array[3]);
```

Recap: Assigning Array Elements

To <u>assign</u> array elements you can use the **[index]** operator on the left-hand-side of a statement (like a variable)
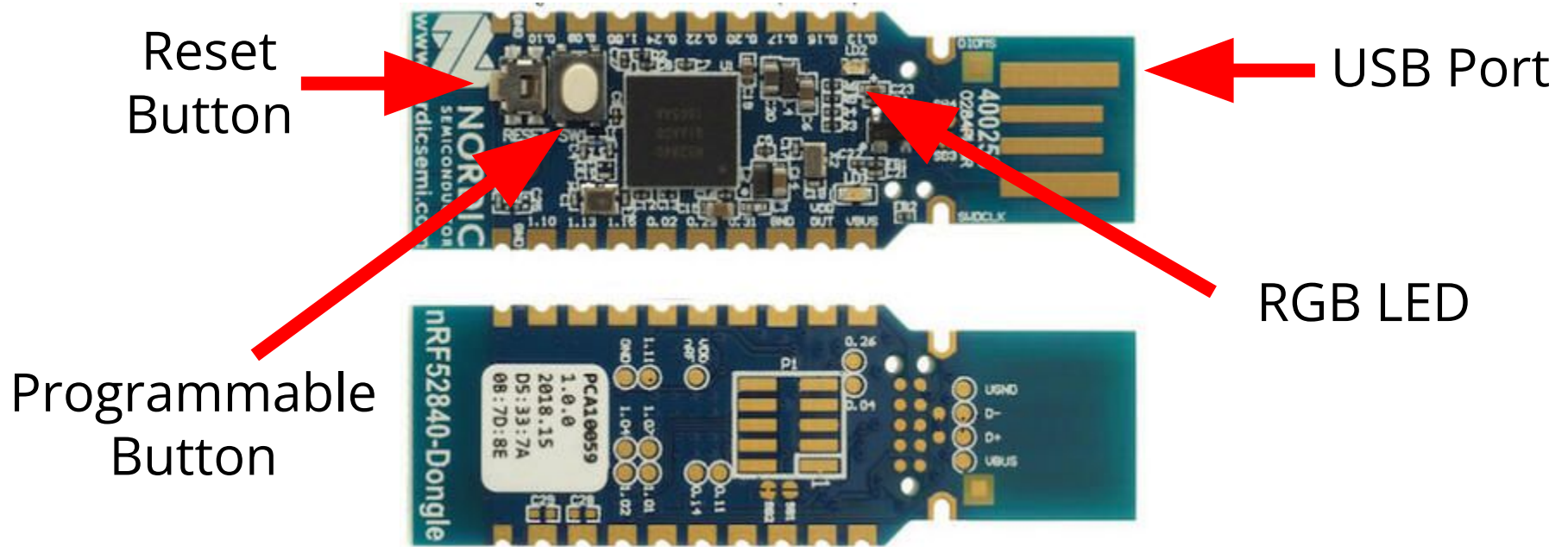
```c
int array[] = {17, 28, 33, 56, 6};
array[3] = 100;
array[2] = -7;
```

```c
printf("%d\n", array[0]);
```

```c
printf("%d\n", array[3]);
```

Recap: What is the nRF52840?

Reset Button

USB Port

Programmable Button
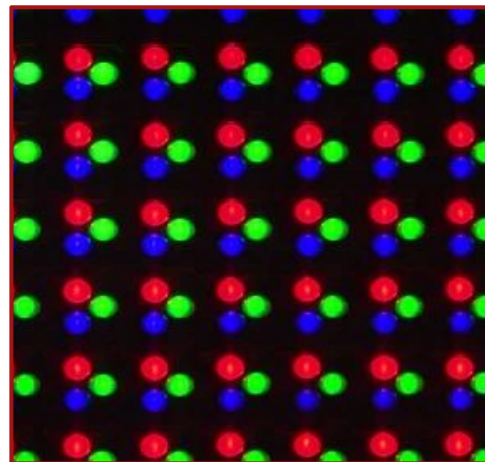
RGB LED

Recap: Anatomy of a Contiki-NG Program

```
1  PROCESS_THREAD(button_hal_example, ev, data) {
2    PROCESS_BEGIN();
3    while (1) {
4      PROCESS_YIELD();

5      if (ev == button_hal_press_event) {
6        printf("Button pressed!\n");

       }

     }

7    PROCESS_END();

   }
```

Recap: RGB LEDs

LEDs are **actuators**, they allow the device to act on the outside world. RGB LEDs have **three configurable color** channels:

1. **Red**
2. **Green**
3. **Blue**

LED displays (such as those of PCs) work the same way

Recap: The LED Library

```c
#define RGB_LED_RED      1
#define RGB_LED_GREEN    2
#define RGB_LED_BLUE     4
#define RGB_LED_MAGENTA (RGB_LED_RED  | RGB_LED_BLUE)
#define RGB_LED_YELLOW  (RGB_LED_RED  | RGB_LED_GREEN)
#define RGB_LED_CYAN    (RGB_LED_GREEN  | RGB_LED_BLUE )
#define RGB_LED_WHITE    (RGB_LED_RED  | RGB_LED_GREEN | RGB_LED_BLUE)
/*------------------------------------------------------------------*/
void rgb_led_off(void);
void rgb_led_set(uint8_t colour);
```

Recap: Buttons

Buttons allow the device to **"sense"** the world around them.

The button allows the device to **receive input and react** to

actions in the world around them.
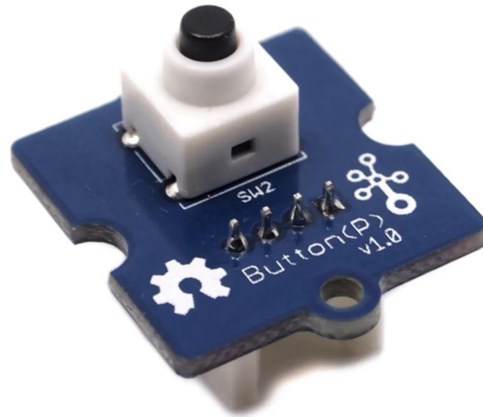
Recap: Button Library

```c
/* Event generated when a button gets pressed */

extern process_event_t button_hal_press_event;

/* Event generated when a button gets released */

extern process_event_t button_hal_release_event;

/* Event generated every second the button is kept pressed */

extern process_event_t button_hal_periodic_event;

/*----------------------------------------------------------------*/

#define BUTTON_HAL_STATE_RELEASED  0

#define BUTTON_HAL_STATE_PRESSED   1

/*----------------------------------------------------------------*/

void button_hal_init(void);

uint8_t button_hal_get_state(button_hal_button_t  *button);
```

Recap: Button Events

```c
while (1) {

    PROCESS_YIELD();

    if (ev == button_hal_press_event) {

        printf("Button pressed!\n");

    } else if (ev == button_hal_release_event) {

        printf("Button released!\n");

    } else if (ev == button_hal_periodic_event) {

        press_seconds = press_seconds + 1;

        printf("Button pressed for %d seconds!\n",
press_seconds);

    }

}
```

1

2

3

Recap: Programming the nRF52840

1 Attach the **nRF52840** chip to your laptops

⚠️ Ensure the device is in **bootloader mode** (blinking red light)

Reset Button ➡️

3 Program the firmware

```
make button.dfu-upload
```

Exercise

Change the code in (**button.c**):

1. Turn the LED to color **GREEN** when the button is <u>pressed</u>

2. Turn the LED **OFF** when the button is <u>released</u>

3. Turn the LED to color **CYAN** when the button is <u>kept pressed</u> for more than five seconds

To flash: `make button.dfu-upload`   **For console:** `make login`

Exercise Solution

```c
if (ev == button_hal_periodic_event) {

  press_seconds = press_seconds + 1;

  printf("Button pressed for %d seconds!\n", press_seconds);


  if (press_seconds >= 5) {

    rgb_led_set(RGB_LED_CYAN);

  }

}
```

Save remotely your Changes

1 `make save`

2

```
Password
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```

3

```
✅ Changes committed and pushed. All done!
```

Recap Exercise

Change the code in (**rainbow.c**):

- You are given the following array of colors. Cycle through the LED colors as the button is kept pressed.

```
static uint8_t colors[] = {
    RGB_LED_CYAN, RGB_LED_YELLOW, RGB_LED_GREEN, RGB_LED_WHITE, RGB_LED_MAGENTA,
};
```

To flash: `make rainbow.dfu-upload`   **For console:** `make login`

Exercise Solution

```c
static int press_count = 0;
while (1) {
  PROCESS_YIELD();
  if (ev == button_hal_release_event) {
    rgb_led_off();
    press_count = 0;
  } else if (ev == button_hal_periodic_event) {
    rgb_led_set(colors[press_count]);
    press_count = (press_count + 1) % 5;
  }
}
```

Save remotely your Changes

1 `make save`

2
```
Password
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```
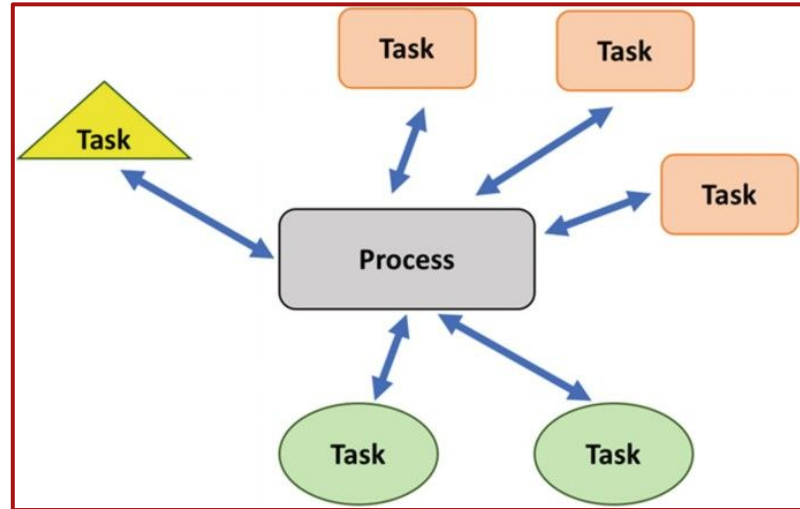
3
```
✅ Changes committed and pushed. All done!
```

Processes

Contiki-NG **Processes** allow for the execution of **multiple tasks** at the same time (i.e. **concurrently**).

We have seen:

1. **PROCESS_BEGIN**

2. **PROCESS_END**

3. **PROCESS_YIELD**

Timers

Timers allow Contiki-NG Processes to execute tasks at **specific points in time**. There are different types of timer:

1. **S-TIMER**: Seconds timer

2. **E-TIMER**: Event timer

3. **C-TIMER**: Callback timer

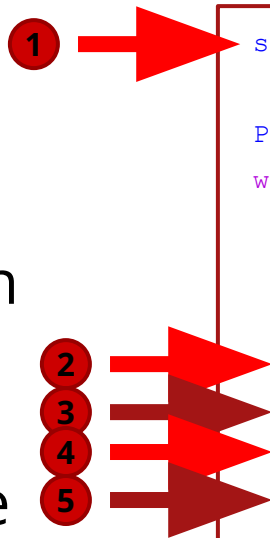In this course we will focus on <u>E-Timers</u>.

The E-Timer Library

```c
/* Event generated when a timer expires */
#define PROCESS_EVENT_TIMER          0x88
/*----------------------------------------------------------*/
/* Set the amount of time on the timer. Also start the timer */
void etimer_set(struct etimer *et, clock_time_t interval);
/* Restart the timer with the previously set amount of time */
void etimer_restart(struct etimer *et);
void etimer_stop(struct etimer *et);
/*----------------------------------------------------------*/
/* Check if the timer has completed */
bool etimer_expired(struct etimer *et)
```

Exercise

Change the code in (**simple_timer.c**)

1) Pressing the button turns the LED Blue.

2) After 2 seconds the LED is turned off

(1)

(2)

(3)

(4)

(5)

```c
static struct etimer timer1;


PROCESS_BEGIN();

while (1) {

  PROCESS_YIELD();

  if (ev == button_hal_press_event) {

    rgb_led_set(RGB_LED_BLUE);

    etimer_set(&timer1, 2 * CLOCK_SECOND);

  } else if (ev == PROCESS_EVENT_TIMER) {

    if (etimer_expired(&timer1)) {

      rgb_led_off();

    }

  }

}
```

To flash: `make simple_timer.dfu-upload`   **For console:** `make login`

Save remotely your Changes

1 `make save`

2
```
Password

Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```

3
```
✅ Changes committed and pushed. All done!
```

Exercise

Change the code in
(**simple_timer.c**)

1) Pressing the button turns the LED BLUE.

2) After 2 seconds the LED turns GREEN

3) After 5 seconds the LED turns off

```
static struct etimer timer1;
static struct etimer timer2;      ← 1
PROCESS_BEGIN();
while (1) {
  PROCESS_YIELD();
  if (ev == button_hal_press_event) {
    rgb_led_set(RGB_LED_BLUE);
    etimer_set(&timer1, 2 * CLOCK_SECOND);
    etimer_set( ??? );            ← 2
  } else if (ev == PROCESS_EVENT_TIMER) {
    if (etimer_expired(&timer1)) {
      ???                         ← 3
    }
    if (etimer_expired(&timer2)) {   ← 4
      ???                         ← 5
    }
  }
```

To flash: `make simple_timer.dfu-upload`   **For console:** `make login`

Exercise Solution

```c
static struct etimer timer1;
static struct etimer timer2;
PROCESS_BEGIN();
while (1) {
  PROCESS_YIELD();
  if (ev == button_hal_press_event) {
    rgb_led_set(RGB_LED_BLUE);
    etimer_set(&timer1, 2 * CLOCK_SECOND);
    etimer_set(&timer2, 5 * CLOCK_SECOND);
  } else if (ev == PROCESS_EVENT_TIMER) {
    if (etimer_expired(&timer1)) {
      rgb_led_set(RGB_LED_GREEN);
    }
    if (etimer_expired(&timer2)) {
      rgb_led_off();
    }
  }
}
```

Save remotely your Changes

1 `make save`

2


```
│Password
Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```
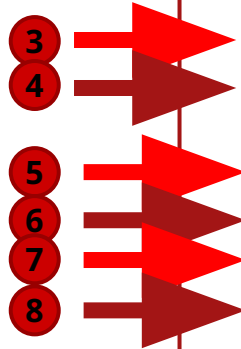
3


```
✅ Changes committed and pushed. All done!
```

Exercise

Change the code in
(**blinker.c**)

1) Turn the LED GREEN
for 1 second

2) Turn the LED OFF
for 1 second

```
PROCESS_BEGIN();

① etimer_set(&timer1, CLOCK_SECOND);

② etimer_set(&timer2, 2 * CLOCK_SECOND);

while (1) {

  PROCESS_YIELD();

  if (ev == PROCESS_EVENT_TIMER) {

③   if (etimer_expired(&timer1)) {

④     rgb_led_set(RGB_LED_GREEN);

    }

⑤   if (etimer_expired(&timer2)) {

⑥     rgb_led_off();

⑦     etimer_restart(&timer1);

⑧     etimer_restart(&timer2);

    }

  }

}
```

To flash: `make blinker.dfu-upload`   **For console:** `make login`

Save remotely your Changes

1   `make save`

2

```
Password

Git: https://aspina@git.spina.me (Press 'Enter' to confirm or
'Escape' to cancel)
```

3

```
✅ Changes committed and pushed. All done!
```

End of Class

See you all next week!