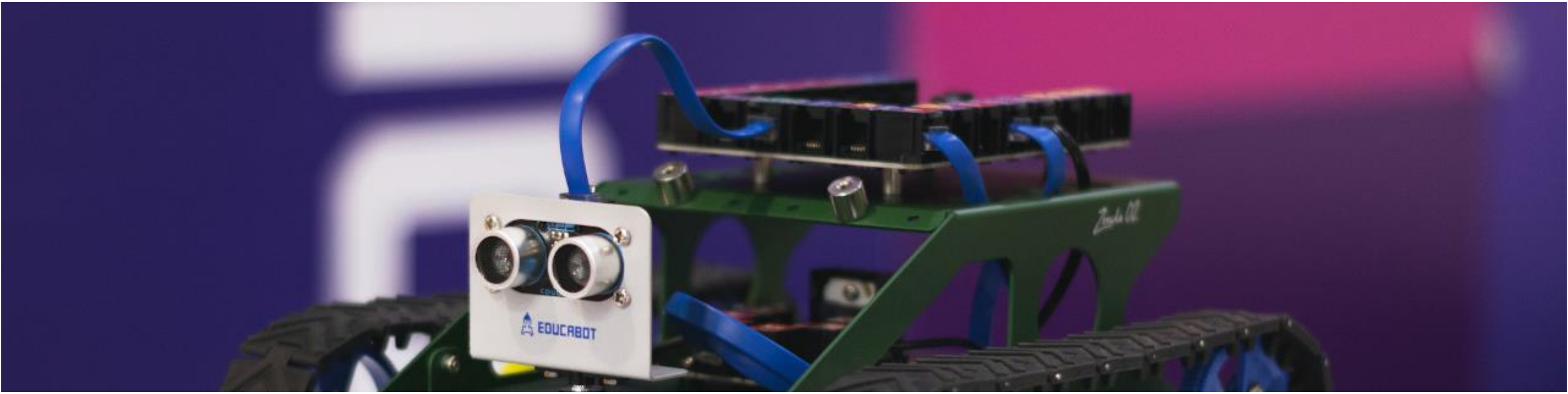


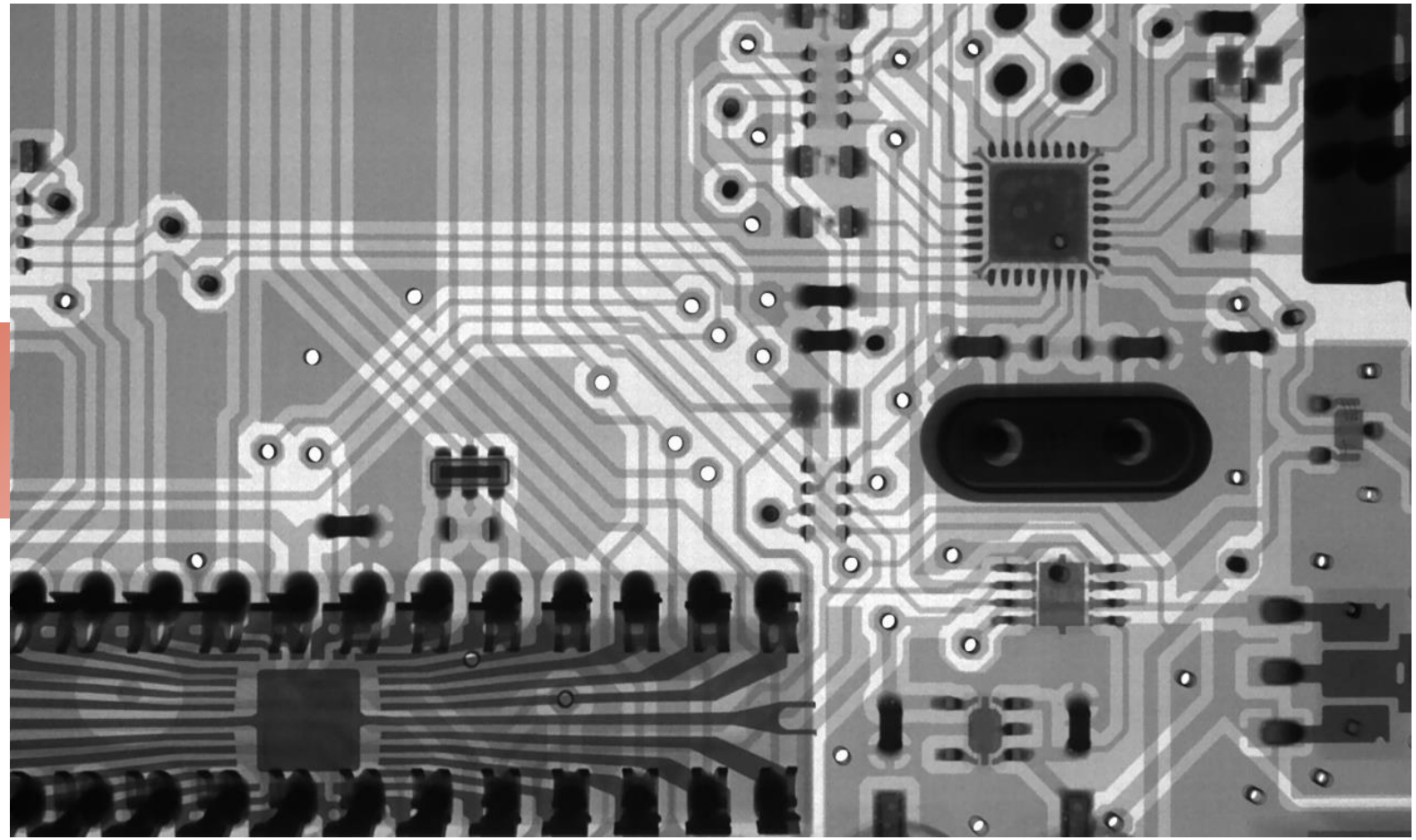
Arduino SYS-STEM for Schools

Training Methodology

SYS-STEM – Arduino SYS-STEM for Schools
Erasmus + Key Action 2 Strategic partnership - 2019-1-ES01-KA201-064454

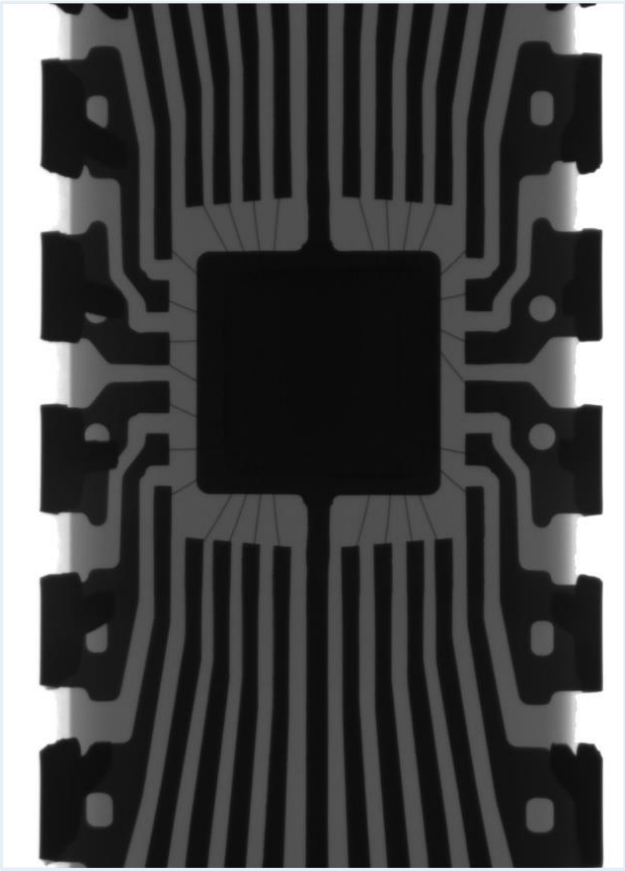
MODULE 5

Decision Making and Control Functions



TRAINING MODULE CONTENTS

- Objectives
- Learning Outcomes
- Unit 1 – Logical conditions
  - Comparison operators
  - Boolean operators
- Unit 2 – Decision Making
  - if and if else statements
  - Switch-case statement
- Unit 3 – Repetitive statements
  - For loop
  - While loop
  - Do While loop
  - Loop flow control
- Additional Reading Materials
- Exercises



OBJECTIVE

Module 5 will present the use of decision making and loop instructions in Arduino (C/C++). The boolean and comparison operators are also presented together with instructions to control the loop execution flow.

This module helps to understand flow control inside a Arduino program, how to repeat blocks of code and define alternative code .

EXPECTED LEARNING OUTCOMES

Knowledge

Upon completion of this module, the student will be able to:

- Understand the difference between decision and loop instructions and where they are commonly applied.
- Understand how logical operators work and their truth tables.

Competences and Skills

Upon completion of this module, the student will be able to:

- Correctly select and use the decision and loop instructions
- Correctly build a logical condition to use on the decision and loop instructions

UNIT 1

Logical conditions

Comparison operators

- In conditions, the equal operator (==) is frequently exchanged by mistake, with the assign operator (=) with the following results

Expression	Description
x==10	Evaluates to **true if x is 10**; evaluates to **false otherwise**
10==x	Evaluates to **true if x is 10**; evaluates to **false otherwise**
x=10	**Assigns 10 to x**; Then evaluates the x as a boolean variable (0 is false; different from 0 is true). In this case the result **is always true**
10=x	Renders an **compiler error**, because you can not assign to 10

- When you are comparing equality from a value and a variable, you should place the value always on the left side. This will render an error if you only use on equal sign

Comparison examples

- Considering the following variables:

```
char  c='x', car='A';
int   x=10, v[]={10,3,5};
float pi=3.141592;
```

- Examples

```
'c' == c     // False
c != car     // True
c < car      // False (Ascii x=88, A=65)
65 == car    // True (Ascii A=65)
x <= v[1]    // False (v[1] is 3)
x > pi       // True
```

Boolean operators

- The C/C++ language uses the following boolean operators

Operator	Description
!	Logical NOT (unary operator used before value/condition)
&&	Logical AND
||	Logical OR

- Boolean operators are used to combine several comparisons/Boolean values
- The `&&` result is `true`, only if the all the conditions are `true`
- The `||` result is `true`, if any of the conditions is `true`
- The `&&` operator takes precedence over `||`

Boolean operators

- Considering the following variables:

```
char  c='x', car='A';
int   x=10, y=12345, v[]={10,3,5};
float pi=3.141592;
```

- Examples

```
c == 'X' && x>10                          // False
car >= 'A' && car <= 'Z'                  // True
(x == v[0]) && (pi >= 3) && (car != 'X')  // True
(y > 12300) || (pi == 3.1412)             // True
(x == y) || (x > 10 + 4)                  //False
```

UNIT 2

Decision Making

if statement

```
if (Boolean_expression) {
    // statement(s) will execute if the boolean expression is true
}
```

- The **`if`** statement is used, to run some code only when the **`Boolean_expression`** is **`true`**
- If you only have one statement, you may exclude the **`{}`**
- Example

```
If ( a > 20 ) {
     // if condition is true then turn on builtin LED
     digitalWrite(LED_BUILTIN, HIGH);
}
```

if-else statement

```
if (Boolean_expression) {
    // statement(s) will execute if the boolean expression is true
} else {
    // statement(s) will execute if the boolean expression is false
}
```

- The **if-else** statement is similar to the if statement, but will also run code when the **Boolean_expression** is **false**
- If you don't use **{}**, and have two **if** statements followed by only one **else**, the **else** belongs to the last **if** (the two following examples are equivalent)

```
if (x==10) if (y>x) a=5; else a=10;
if (x==10) {if (y>x) a=5; else a=10;}
```

if-else statement

- Example

```
if ( a > 20 ) {
    /* if condition is true then turn builtin LED on */
     digitalWrite(LED_BUILTIN, HIGH);
} else {
    /* if condition is false then turn  builtin LED off */
     digitalWrite(LED_BUILTIN, LOW);
}
```

- You can nest several **`if`** statements, i.e., you can have one **`if`** or **`if-else`** statement inside another **`if`** or **`if-else`** statement.

switch statement

```
switch(expression) {
    case constant-expression :
            statement(s);
            break; // optional
    case constant-expression :
            statement(s);
            break; // optional
    // you can have any number of case statements
    default : // Optional
            statement(s);
}
```

switch statement

- When the variable being switched on is equal to a **`case`**, the statements following that **`case`** will execute until a **`break`** statement is reached.
- You can have any number of **`case`** statements within a switch. Each **`case`** is followed by the value to be compared to and **`a colon`**
- Not every **`case`** needs to contain a **`break`**. If no **`break`** appears, the flow of control will fall through to subsequent cases until a break is reached.
- A **`switch`** statement can have an optional **`default`** case, which must appear at the end of the **`switch`**.
- It is possible to have a **`switch`** as a part of the statement sequence of an outer **`switch`**. Even if the **`case`** constants of the inner and outer **`switch`** contain common values, no conflicts will arise.

switch statement

```
char grade = 'B';
switch(grade) {
    case 'A' : Serial.println("Excellent!\n" );
               break;
    case 'B' :
    case 'C' : Serial.println("Well done\n" );
               break;
    case 'D' : Serial.println("You passed\n" );
               break;
    case 'F' : Serial.println("Better try again\n" );
               break;
    default : Serial.println("Invalid grade\n" );
}
```

UNIT 3

Repetitive Instructions

for loop

```
for ( init; condition; increment ) {
    // statement(s);
}
```

1. The **`init`** expression(s) are evaluated first (you can declare variables in **`init`**)
2. Next the **`condition`** is evaluated
   - If it is **`true`**, the body of the loop is executed
   - If it is **`false`**, the body of the loop does not execute and the flow of control jumps to the next statement just after the **`for`** loop
3. After the body of the **`for`** loop executes, the flow of control jumps back up to the **`increment`** statement which is evaluated (variables are incremented, etc.) and we jump to 2 (**`condition`** evaluation)

for loop

- For loops are primary used, when the number of repetitions is known
- If you only have one statement, you may exclude the `{}`
- Be careful not to add a `;` to the end of the `for()` statement (otherwise the `for` has no statements)
- Example

```
for(int i=1; i<10; i++){
    digitalWrite(LED_BUILTIN, HIGH);  // Turns built-in LED on
    delay (150);                      // Pauses the program for 150ms
    digitalWrite(LED_BUILTIN, LOW);   // Turns built-in LED off
    delay (1000);                     // Pauses the program for 1s (1000ms)
}
```

while loop

```
while (condition) {
    // statement(s);
}
```

1. The **condition** is evaluated
   - If it is **true**, the body of the loop is executed
   - If it is **false**, the body of the loop does not execute and the flow of control jumps to the next statement just after the **while** loop
2. After the body of the **while** loop executes, the flow of control jumps to 1 (**condition** evaluation)

- **while** loops are primary used, when the number of repetitions is not known
- If you only have one statement, you may exclude the **{}**

while loop

- Be careful not to add a `;` to the end of the **`while()`** condition (otherwise the **`while`** has no statements)
- Example

```
int n = 6;
while (n > 0) {                          //While n is greater than 0
    digitalWrite(LED_BUILTIN, HIGH);  // Turns built-in LED on
    delay (150);                         // Pauses the program for 150ms
    digitalWrite(LED_BUILTIN, LOW);    // Turns built-in LED off
    delay (1000);                        // Pauses the program for 1s (1000ms)
    n--;                                 // The next value for N ( N = N - 1)
}
```

while loop

- A **`while`** loop is often used to wait until some action happens
- Example

```
while(!digitalRead(4)); // Waiting for pin 4 to get LOW => !(pin 4) is HIGH
```

- A **`while`** loop can also be used to wait run forever
- Example

```
while(1) { // Loop forever
    // Statement(s)
}
```

do-while loop

```
do {
    // statement(s);
} while (condition);
```

1. The body of the **do-while** loop executes
2. The **condition** is evaluated
   - If it is **true**, the flow control jumps back to 1
   - If it is **false**, the body of the loop does not execute and the flow of control jumps to the next statement just after the **for** loop

- **do-while** loops are primary used, when we have to run the loop at least once
- Contrary to the **while** loop, there is a semicolon (;) after the condition in **do-while** loops

do-while loop

- Example

```
bool value;
do {
    value=digitalRead(2);  // Reads PIN2
    delay (150);           // Pauses the program for 150ms
} while (!value);          // While value (PIN2) is LOW
```

continue statement

- Inside a loop, you can use the **`continue`** statement to interrupt the loop execution, and go to the condition evaluation
- Example

```
for(int i=1; i<10; i++){
    digitalWrite(LED_BUILTIN, HIGH);  // Turns built-in LED on
    delay (150);                      // Pauses the program for 150ms
    if (i==9) continue;               // if i is 9, run loop only to here
                                      // and evaluate condition
    digitalWrite(LED_BUILTIN, LOW);   // Turns built-in LED off
    delay (1000);                     // Pauses the program for 1s (1000ms)
}
```

break statement

- Inside a loop (or switch), you can use the **`break`** statement to exit the loop (or switch), and resume flow after the loop (or switch),
- Example

```
for(int i=1; i<10; i++){
    digitalWrite(LED_BUILTIN, HIGH);  // Turns built-in LED on
    delay (150);                      // Pauses the program for 150ms
    if (digitalRead(5)) break;        // if PIN 5 gets HIGH exit the for loop
    digitalWrite(LED_BUILTIN, LOW);   // Turns built-in LED off
    delay (1000);                     // Pauses the program for 1s (1000ms)
}
```

EXTRA READING MATERIALS

Online Resources

- TutorialsPoint C Tutorial Decision Making
  - https://www.tutorialspoint.com/cprogramming/c_decision_making.htm
- TutorialsPoint C Tutorial Loops
  - https://www.tutorialspoint.com/cprogramming/c_loops.htm

EXERCISES / TESTS / QUIZZES

Exercises

1. Turn on the Built-in LED, if the PINS between 2 and 6 are HIGH
2. Simulate a string of lights, turning on sequentially the PINS between 2 and 8 (keep them on during 200ms)
   - The next light only lights after the previous turns off
3. Flash the Built-in LED 10 times (150ms on, 150ms off) after each flash sequence, wait 1 second
4. Change the Built-in LED state (HIGH to LOW or LOW to HIGH), if the PINS 2, 3 and 4 are put HIGH in sequence (only one PIN can be HIGH to pass to next “Level”)
   - The sequence is: PIN2 HIGH; PIN3,4 LOW → PIN3 HIGH; PIN2,4 LOW → PIN4 HIGH; PIN2,3 LOW

CONGRATULATIONS

You have completed SYS-STEM Module 5