

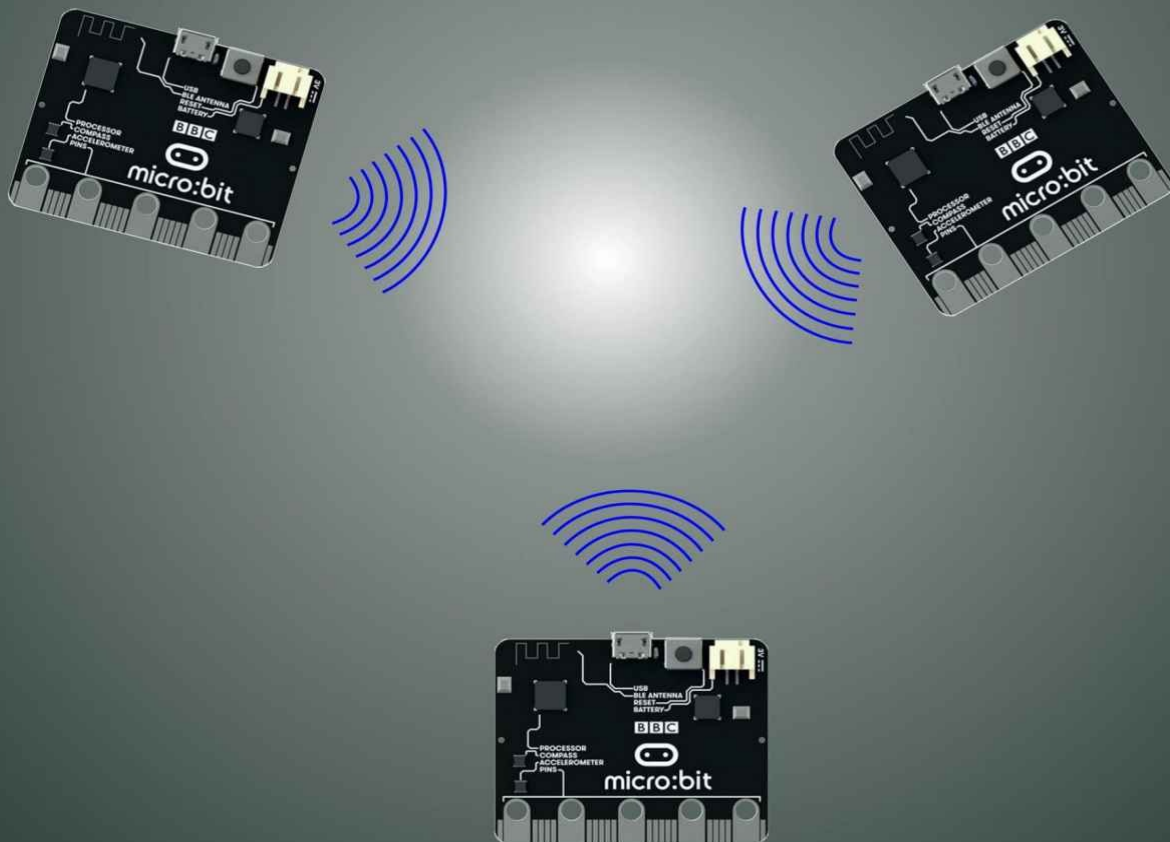
Yury Magda

BBC micro:bit

Wireless

Projects

Book



BBC micro:bit Wireless Projects Book

By Yury Magda

Copyright © 2016 by Yury Magda. All rights reserved.

The programs, examples, and applications presented in this book have been included for their instructional value. The author offers no warranty implied or express, including but not limited to implied warranties of fitness or merchantability for any particular purpose and do not accept any liability for any loss or damage arising from the use of any information in this book, or any error or omission in such information, or any incorrect use of these programs, procedures, and applications.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author.

Contents

[Introduction](#)

[Disclaimer](#)

[BBC micro:bit features](#)

[Using the Microsoft PXT](#)

[Basic wireless applications](#)

[Project 1](#)

[Project 2](#)

[Project 3](#)

[Project 4](#)

[Project 5](#)

[Project 6](#)

[Project 7](#)

[Project 8](#)

[Advanced wireless applications](#)

[Project 1](#)

[Project 2](#)

[Project 3](#)

[Project 4](#)

[Project 5](#)

[Project 6](#)

[Project 7](#)

Introduction

The BBC micro:bit is a pocket-sized computer that you can code, customize and control to bring your ideas to life. This small board allows to build complicated applications, even by beginners. This book is thought as a highly practical guide that is aimed at getting students and hobbyists up-and-running with the BBC micro:bit. The guide describes various wireless applications developed in the Microsoft Programming Experience Toolkit (PXT) environment. The Programming Experience Toolkit editor provides a programming experience supporting both a block-based editor and JavaScript, along with great new features like peer-to-peer radio.

Each project from this book accompanied by a brief description which helps to make things clear. All projects described in this guide can be easily improved or modified if necessary.

Disclaimer

The design techniques described in this book have been tested on the BBC micro:bit boards without damage of the equipment. I will not accept any responsibility for damages of any kind due to actions taken by you after reading this book.

BBC micro:bit features

This section contains a brief description of the BBC micro:bit board. The basic built-in components available on the board are as follows:

- LED Screen and Status LED;
- Buttons;
- Compass;
- Accelerometer;
- Light sensor.

The **5 x 5 LED Screen** is formed by the red LEDs, each of them can be set to ON/OFF form a 5 x 5 LED Screen. The brightness of the LEDs can also be controlled. The yellow LED on the back of the BBC micro:bit is the status LED. It flashes yellow when the system informs a user that something has happened.

Buttons A and B can be used as digital inputs. The BBC micro:bit can detect either of its two buttons being pressed/released and be programmed to process these events.

The **button** on the back of the BBC micro:bit is a system button. It uses for restarting and running the program code that has been downloaded onto the BBC micro:bit. This button after pressing and releasing allows to restart and run your program from the beginning.

The **compass** can detect magnetic fields such as the Earth's magnetic field. As the BBC micro:bit has this compass, it is possible to detect the direction it is moving in. The BBC micro:bit can detect where it is facing and movement in degrees. This data can be used by the BBC micro:bit in a program or be sent to another device.

The **accelerometer** on the BBC micro:bit allows to detect changes in the BBC micro:bit's speed. It converts analogue information into digital form that can be used in BBC micro:bit programs. The output of the accelerometer is in milli-g (mg). The device will also detect a small number of standard actions e.g. shake, tilt and free-fall.

The **light sensor** uses the LED Screen in photodiode mode that allows to measure light intensity.

The BBC micro:bit provides a USB and serial interfaces for data transfer. The BBC micro:bit USB can be used for connecting a device to the computer via a micro USB cable. Data can be sent and received between the BBC micro:bit and the computer so programs can be downloaded from Windows, Macs and Chromebooks onto the BBC micro:bit via this USB data connection.

Power to the BBC micro:bit may be fed from USB when the BBC micro:bit is connected to the

PC. When the BBC micro:bit isn't connected to the computer, tablet or mobile, we can use 2 x AAA 1.5 V batteries to power the board.

The pins labelled **3V** and **GND** are the power supply pins that can be used for powering external circuitry. **Be careful when connecting external circuitry to the BBC micro:bit pins. Remember that the load attached to any pin has to draw a very low current (a few milliamperes), otherwise the BBC micro:bit will be damaged!**

Using the Microsoft PXT

All examples from this book are developed using the Microsoft Programming Experience Toolkit (PXT). The Microsoft PXT is a framework for creating special-purpose programming experiences for beginners, especially focused on computer science education. PXT's underlying programming language is a subset of TypeScript (leaving out JavaScript dynamic features). Below is a brief description of the PXT JavaScript Editor.

The JavaScript editor of PXT extends the features and functionality of the Monaco Editor, the editor that powers Visual Studio Code. The Editor contains a built-in JavaScript/Typescript language service that provides complete code intelligence. Additionally, the JavaScript Editor automatically colors each of function and its respective namespace to match the colors of the respective block in the Block Editor. Also known as **IntelliSense**, the editor supports automatic word completion. If the language service knows possible completions, the **IntelliSense** suggestions will pop up as you type. You can always manually trigger it with **Ctrl-Space**.

Hovering over namespaces, functions and function parameters will show useful information describing the purposes of the function, namespace, or parameter.

The Editor supports **Find**, as well as **Find** and **Replace** in order to search for a particular keyword, or search and replace a particular keyword. You can get to the **Find** and **Replace** widget with **Ctrl-F**, or you can also get to it via the **All Commands** window.

This is only a few features of the PXT JavaScript Editor. Much more information about programming in the Microsoft PXT can be found on <https://www.pxt.io/docs>.

Basic wireless applications

The BBC micro:bit modules can interact with each other using radio channels. This capability is supported by the set of the functions included in the **radio** block of the Microsoft PXT. The interaction between BBC micro:bit modules via radio channels is illustrated by the following simple diagram (**Fig.1**).

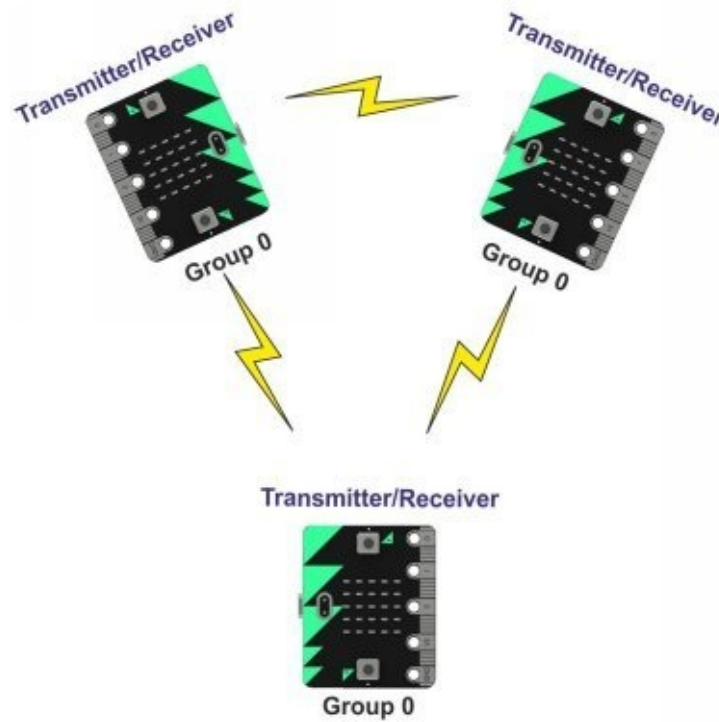


Fig.1

According to this diagram, each of three BBC micro:bit modules can act either as a transmitter or a receiver. The interaction between modules is possible if they belong to the same group (**Group 0**, in our case). There may be up to 256 groups (0...255) in a wireless system.

To assign the group ID to the BBC micro:bit we will use the function **radio.setGroup()** from the **radio** block of PXT. A group is like a cable channel (a BBC micro:bit can only send or receive in one group at a time). A group ID is like the cable channel number.

If we don't tell the program which group ID to use with the **radio.setGroup()** function, it will figure out its own group ID by itself. If we use two different BBC micro:bit modules, they will be able to talk to each other after they have been assigned the same group ID.

The projects from this guide are very simple – they include two BBC micro:bit modules, one of them serves as a radio transmitter, while other serves as a radio receiver. This configuration is shown in **Fig.2**.

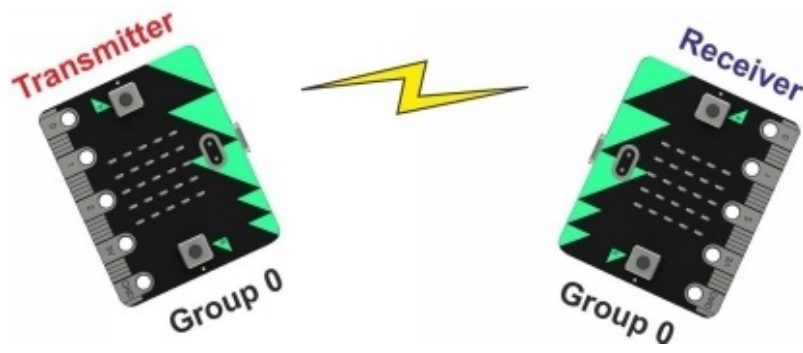


Fig.2

Let's go to developing our first wireless system.

Project 1

In this project, the BBC micro:bit that is a transmitter will send numbers 0 – 9 via a radio channel to the BBC micro:bit module serving as a receiver. The receiver then outputs the received number incremented by 1 on the LED screen.

The block diagram of the application running on the receiver is shown in **Fig.3**.

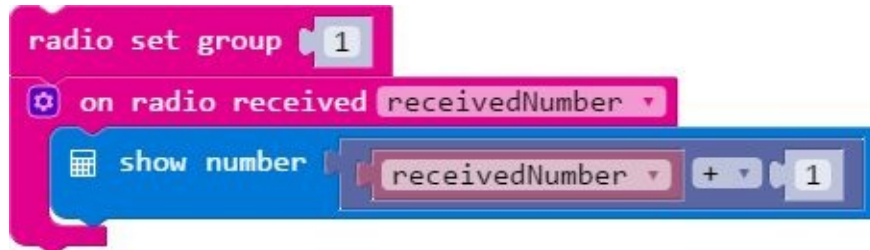


Fig.3

This block diagram is associated with the following JavaScript source code (**Listing 1**).

Listing 1.

```
radio.setGroup(1)
radio.onDataPacketReceived(({receivedNumber}) => {
  basic.showNumber(receivedNumber + 1)
})
```

In this source code, the **radio.setGroup()** function sets the group with ID = 0 for radio communication channel (the same ID should be assigned to the transmitter). The function **radio.onDataPacketReceived()** registers code to run when the **radio** receives a packet. In our case, a received number incremented by 1 is output on the LED display by invoking the **basic.showNumber(receivedNumber + 1)** function.

The block diagram of the application running on the BBC micro:bit serving as a transmitter is shown in **Fig.4**.

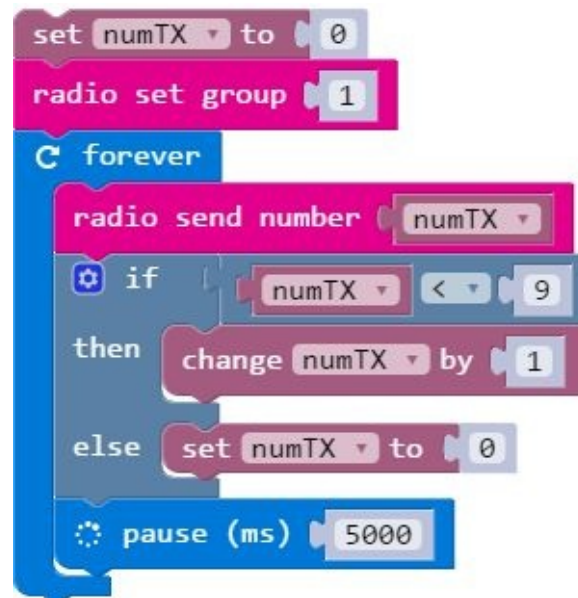


Fig.4

The JavaScript source code associated with the above block diagram is shown in **Listing 2**.

Listing 2.

```

let numTX = 0
radio.setGroup(1)
basic.forever(() => {
  radio.sendNumber(numTX)
  if (numTX < 9) {
    numTX += 1
  } else {
    numTX = 0
  }
  basic.pause(5000)
})

```

This program code invokes the **radio.sendNumber()** function that periodically broadcasts numbers from 0 to 9 to the BBC micro:bit receiver connected via **radio**. The function is called within an endless **basic.forever()** loop every 5 s (function **basic.pause(5000)**).

Project 2

This project illustrates how to remotely check if button **A** on the BBC micro:bit serving as a transmitter is pressed down. The program code on the BBC micro:bit module that is the transmitter detects when button **A** is pressed down and then transfers a warning message to the BBC micro:bit module that acts as a receiver. The receiver outputs the message just received on the LED screen.

The block diagram of the application running on the transmitter is shown in **Fig.5**.

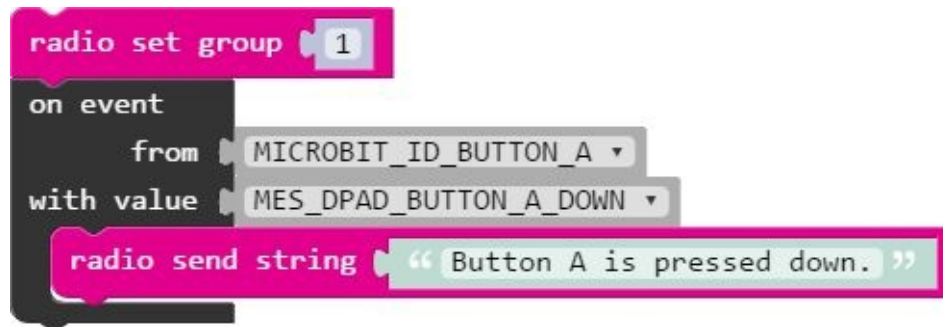


Fig.5

The JavaScript source code associated with the above block diagram is shown in **Listing 3**.

Listing 3.

```
radio.setGroup(1)
control.onEvent(EventBusSource.MICROBIT_ID_BUTTON_A,
EventBusValue.MES_DPAD_BUTTON_A_DOWN, () => {
    radio.sendString("Button A is pressed down.")
})
```

In this source code, the **control.onEvent()** function raises an event in the event bus. This function takes two parameters, **EventBusSource** and **EventBusValue**. The **EventBusSource** parameter determines what component raised an event. In our case, this is a button **A**. The type of the event (pressing the button **A** down) is determined by **MES_DPAD_BUTTON_A_DOWN** constant assigned to **EventBusValue**. To handle this event we use the **radio.sendString()** function that simply transfers a warning message to the receiver.

The block diagram of the application running on the receiver is shown in **Fig.6**.

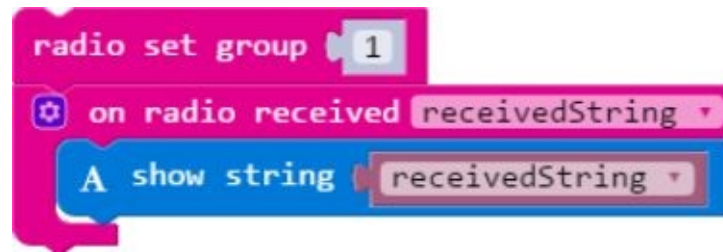


Fig.6

The JavaScript source code associated with the above block diagram is shown in **Listing 4**.

Listing 4.

```
radio.setGroup(1)
radio.onDataPacketReceived(({receivedString}) => {
  basic.showString(receivedString)
})
```

In this code, the data received from the transmitter raises the event registered by the **radio.onDataPacketReceived()** function. This event is handled by the function **basic.showString()** that outputs the string just received on to the LED screen.

Project 3

This project illustrates how to drive digital output **P0** on the BBC micro:bit receiver using the commands sent by the BBC micro:bit transmitter. The digital output **P0** on the receiver board drives the LED network consisting of the LED itself and resistor R1 (**Fig.7**). When pin **P0** goes HIGH, the LED is driven ON. Conversely, when pin **P0** goes LOW, the LED is driven OFF.

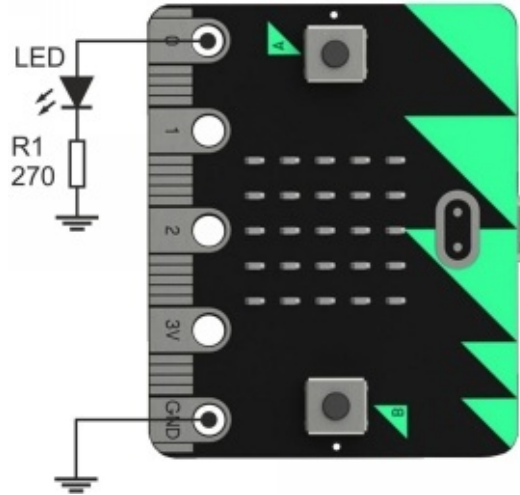


Fig.7

The block diagram of the application running on the BBC micro:bit board serving as a transmitter is shown in **Fig.8**.

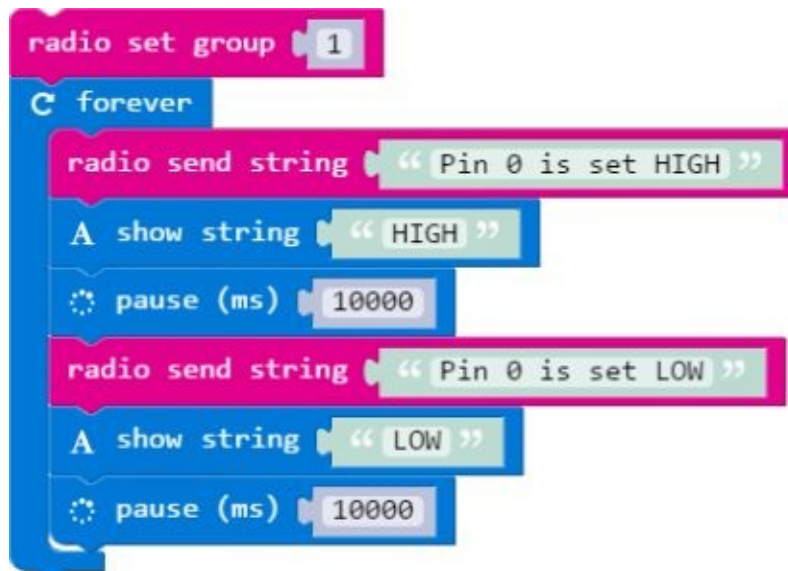


Fig.8

The JavaScript source code associated with the above block diagram is shown in **Listing 5**.

Listing 5.

```

radio.setGroup(1)
basic.forever() => {
    radio.sendString("Pin 0 is set HIGH")
    basic.showString("HIGH")
    basic.pause(10000)
    radio.sendString("Pin 0 is set LOW")
    basic.showString("LOW")
    basic.pause(10000)
})

```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.9**.

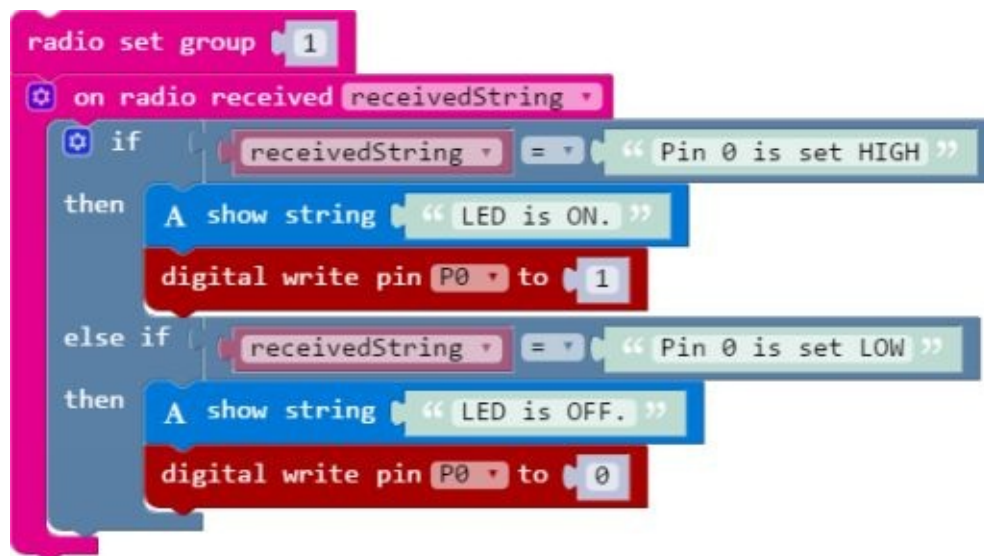


Fig.9

The JavaScript source code associated with the above block diagram is shown in **Listing 6**.

Listing 6.

```

radio.setGroup(1)
radio.onDataPacketReceived(({receivedString}) => {
    if (receivedString == "Pin 0 is set HIGH") {
        basic.showString("LED is ON.")
        pins.digitalWritePin(DigitalPin.P0, 1)
    } else if (receivedString == "Pin 0 is set LOW") {
        basic.showString("LED is OFF.")
        pins.digitalWritePin(DigitalPin.P0, 0)
    }
})

```

To drive the BBC micro:bit pin **P0** we use function **pins.digitalWritePin()**. This function writes a digital (0 or 1) signal to a pin on the BBC micro:bit board. This function allows to bring pin **P0** HIGH/LOW depending on the command being held in the **receivedString** variable.

Project 4

This project demonstrates the design of a simple alarm system using two BBC micro:bit modules. The BBC micro:bit transmitter when shaken transmits the alarm message to the BBC micro:bit that is a receiver. The receiver then outputs the “Alarm!” string on the LED screen. The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.10**.

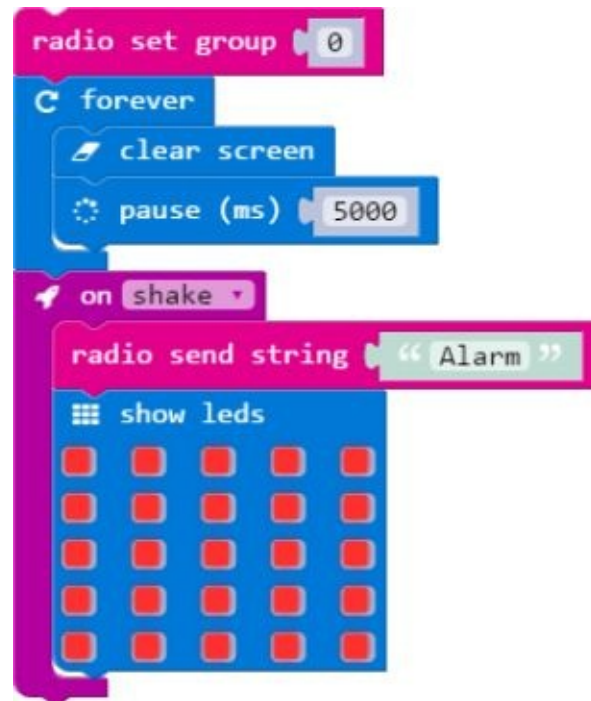


Fig.10

The JavaScript source code associated with the above diagram is shown in **Listing 7**.

Listing 7.

```
radio.setGroup(0)
basic.forever(() => {
  basic.clearScreen()
  basic.pause(5000)
})
input.onGesture(Gesture.Shake, () => {
  radio.sendString("Alarm")
  basic.showLeds(`
    # # # # #
    # # # # #
    # # # # #
    # # # # #
  `)
```

```

#####
`)
})

```

In this source code, we use the **input.onGesture()** event handler that is invoked when you do a gesture (like shaking the BBC micro:bit). The parameter passed to the handler may be one of **shake**, **logo up**, **logo down**, **screen up**, **screen down**, **tilt left**, **tilt right**, **free fall**, **3g** or **6g**. In our example, this parameter is assigned **Shake**; you can test this application with other values assigned to the **Gesture** parameter.

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.11**.

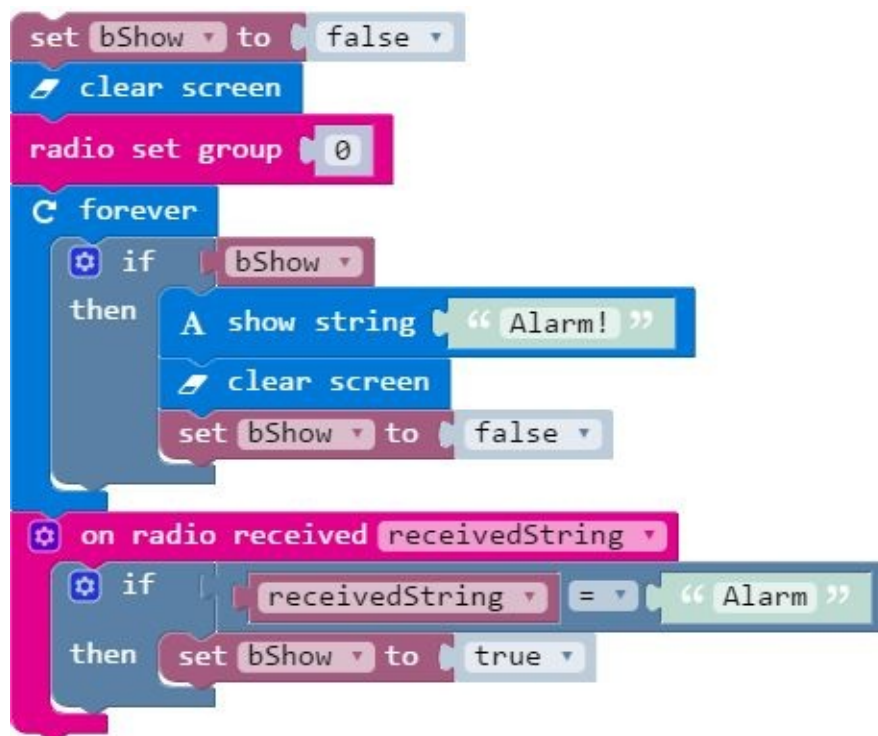


Fig.11

The JavaScript source code associated with this block diagram is shown in **Listing 8**.

Listing 8.

```

let bShow = false
basic.clearScreen()
radio.setGroup(0)
basic.forever(() => {
  if (bShow) {
    basic.showString("Alarm!")
    basic.clearScreen()
  }
})

```

```
        bShow = false
    }
})
radio.onDataPacketReceived(({receivedString}) => {
    if (receivedString == "Alarm") {
        bShow = true
    }
})
```

Project 5

This project illustrates one more programming technique for detecting gestures on a remote BBC micro:bit device. The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.12**.

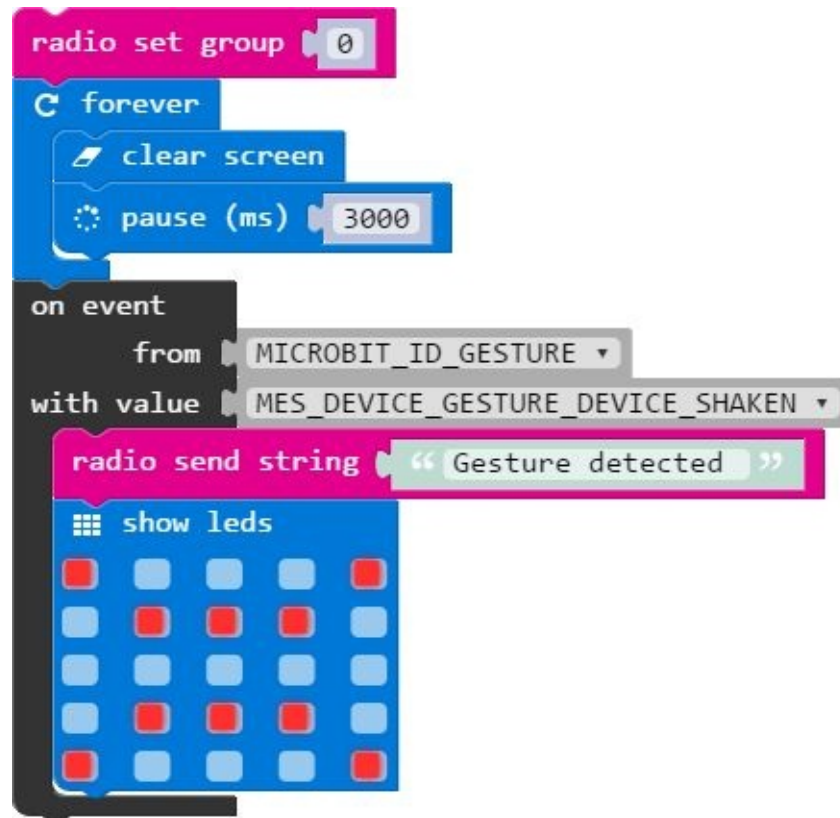


Fig.12

The JavaScript source code associated with the above block diagram is shown in **Listing 9**.

Listing 9.

```
radio.setGroup(0)
basic.forever(() => {
  basic.clearScreen()
  basic.pause(3000)
})
control.onEvent(EventBusSource.MICROBIT_ID_GESTURE,
EventBusValue.MES_DEVICE_GESTURE_DEVICE_SHAKEN, () => {
  radio.sendString("Gesture detected ")
  basic.showLeds (
    # ... #
    .# # # .
```



```

.....
.###.
#...#
`)
})

```

In this source code, the **control.onEvent()** function raises an event in the event bus. This function takes two parameters, **EventBusSource** and **EventBusValue**. The **EventBusSource** parameter determines what component raised an event. In our case, this will be **MICROBIT_ID_GESTURE**. The type of the event is determined by **MES_DEVICE_GESTURE_DEVICE_SHAKEN** constant assigned to **EventBusValue**. To handle this event we use the **radio.sendString()** function that simply transfers a warning message to the receiver. Function **basic.showLeds()** draws the user-defined picture on the LED screen.

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.13**.

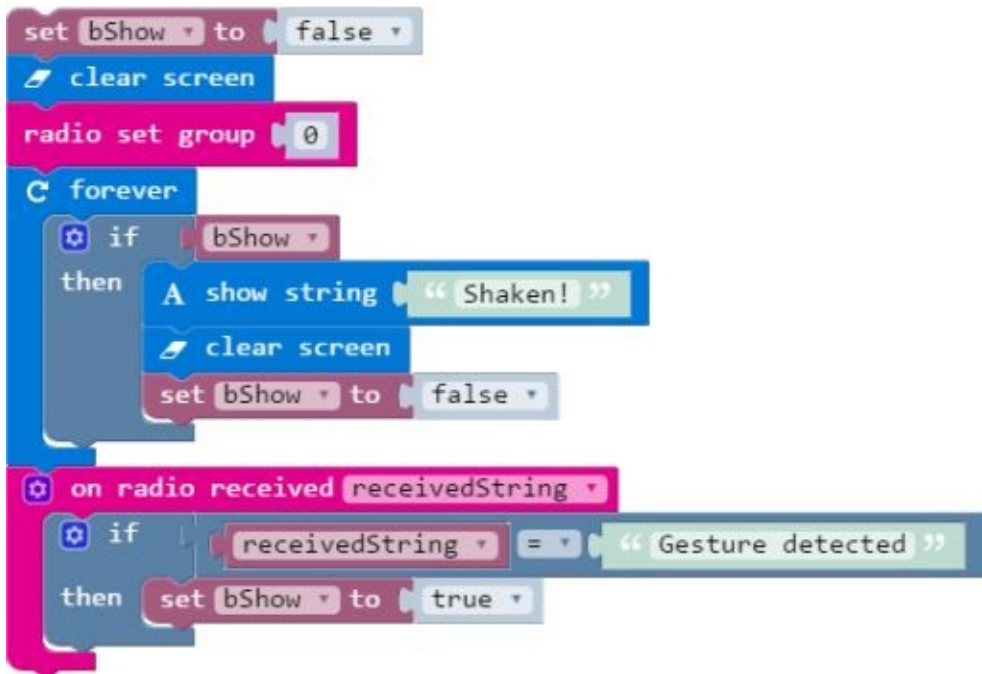


Fig.13

The JavaScript source code associated with the above block diagram is shown in **Listing 10**.

Listing 10.

```

let bShow = false
basic.clearScreen()
radio.setGroup(0)
basic.forever(() => {

```

```
    if (bShow) {  
        basic.showString("Shaken!")  
        basic.clearScreen()  
        bShow = false  
    }  
})  
radio.onDataPacketReceived(({receivedString}) => {  
    if (receivedString == "Gesture detected") {  
        bShow = true  
    }  
})
```

Project 6

This project illustrates how to adjust the brightness of the LED screen on the remote BBC micro:bit that serves as a receiver. The BBC micro:bit transmitter transfers the value between 0 and 255 (light intensity) to the receiver which then adjusts the brightness of its LED screen. The brightness can be increased/reduced by pressing button **A**/button **B** down on the BBC micro:bit transmitter.

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.14**.

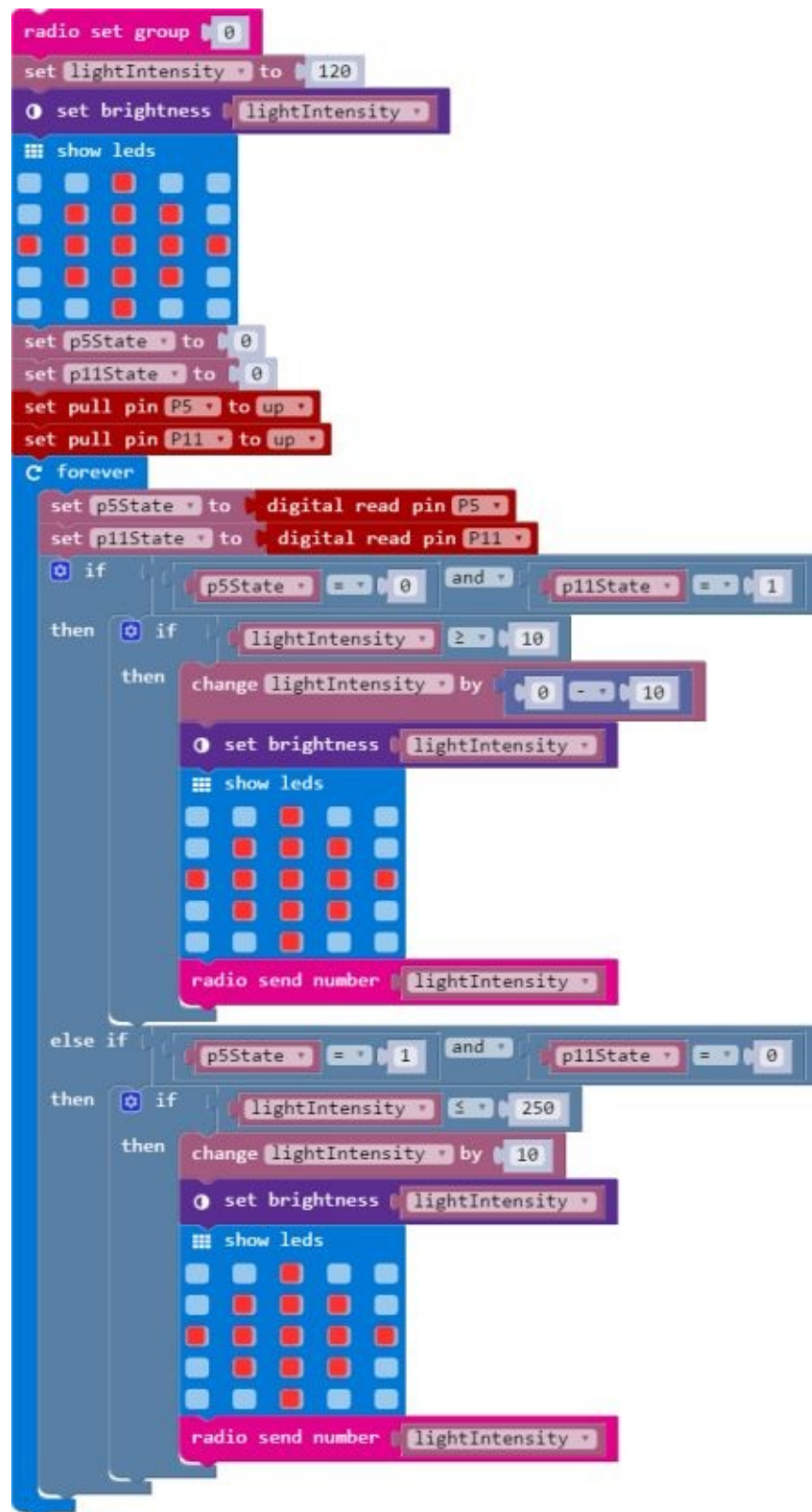


Fig.14

The JavaScript source code associated with the above block diagram is shown in **Listing 11**.

Listing 11.

```
radio.setGroup(0)
```

```

let lightIntensity = 120
led.setBrightness(lightIntensity)
basic.showLeds(`
  ..# ..
  .###.
  #####
  .###.
  ..# ..
`)
let p5State = 0
let p11State = 0
pins.setPull(DigitalPin.P5, PinPullMode.PullUp)
pins.setPull(DigitalPin.P11, PinPullMode.PullUp)
basic.forever(() => {
  p5State = pins.digitalReadPin(DigitalPin.P5)
  p11State = pins.digitalReadPin(DigitalPin.P11)
  if (p5State == 0 && p11State == 1) {
    if (lightIntensity >= 10) {
      lightIntensity += 0 - 10
      led.setBrightness(lightIntensity)
      basic.showLeds(`
        ..# ..
        .###.
        #####
        .###.
        ..# ..
      `)
      radio.sendNumber(lightIntensity)
    }
  } else if (p5State == 1 && p11State == 0) {
    if (lightIntensity <= 250) {
      lightIntensity += 10
      led.setBrightness(lightIntensity)
      basic.showLeds(`
        ..# ..
        .###.
        #####
        .###.
        ..# ..
      `)
      radio.sendNumber(lightIntensity)
    }
  }
}

```

```
}  
})
```

In this source code, pins P5 (button A) and P11 (button B) are configured as inputs with pull-up resistors by the following sequence:

```
pins.setPull(DigitalPin.P5, PinPullMode.PullUp)  
pins.setPull(DigitalPin.P11, PinPullMode.PullUp)
```

The program code when launched adjusts the initial brightness of the LED screen to 120:

```
let lightIntensity = 120  
led.setBrightness(lightIntensity)
```

Most of the program code is running within an endless **basic.forever()** loop. Each iteration within the loop begins with reading digital inputs **P5** and **P11**:

```
p5State = pins.digitalReadPin(DigitalPin.P5)  
p11State = pins.digitalReadPin(DigitalPin.P11)
```

If **p5State** is 0/1, then button **A** has been pressed/released. Similarly, the state of button **B** can be evaluated through the **P11State**. The program code evaluates two conditions:

- button **A** pressed and button **B** released;
- button **A** released and button **B** pressed.

This is performed by the **if()** logical statement:

```
if (p5State == 0 && p11State == 1)  
...  
else  
...
```

Depending on the comparison, the brightness of the LED screen either increases or reduces.

The application running on a BBC micro:bit receiver is represented by the following block diagram (**Fig.15**).

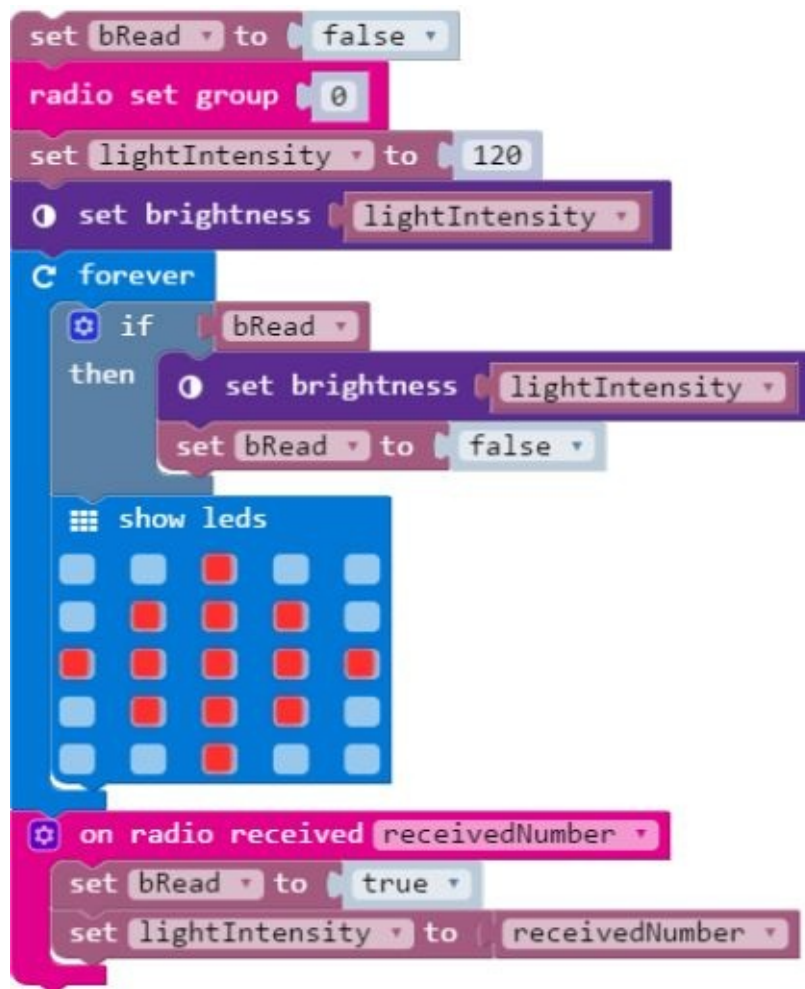


Fig.15

The JavaScript source code associated with the above block diagram is shown in **Listing 12**.

Listing 12.

```

let bRead = false
radio.setGroup(0)
let lightIntensity = 120
led.setBrightness(lightIntensity)
basic.forever(() => {
  if (bRead) {
    led.setBrightness(lightIntensity)
    bRead = false
  }
  basic.showLeds(`
    ..# ..
    .###.
    #####
    .###.
    ..# ..
  `)
})

```

```
    `)  
  })  
  radio.onDataPacketReceived(({receivedNumber}) => {  
    bRead = true  
    lightIntensity = receivedNumber  
  })
```


Project 7

This project allows to build a simple light-control system. This system uses two BBC micro:bit boards, a transmitter and receiver. The application running on the transmitter obtains the light intensity data from a light sensor every 1 s and sends the data via a radio channel to the receiver. The application on the BBC micro:bit receiver reads the data, compares it with a predetermined value and drives the LED screen either ON or OFF. The block diagram of the application running on the transmitter is shown in **Fig.16**.

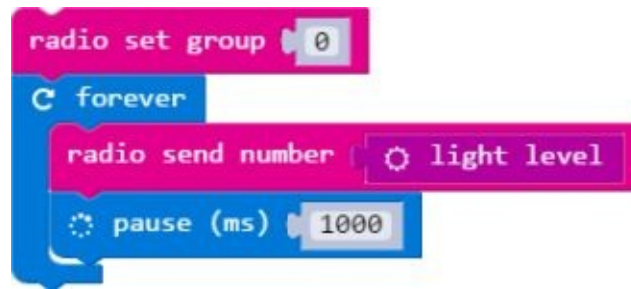


Fig.16

The JavaScript source code associated with the above diagram is shown in **Listing 13**.

Listing 13.

```
radio.setGroup(0)
basic.forever(() => {
    radio.sendNumber(input.lightLevel())
    basic.pause(1000)
})
```

The block diagram of the application running on the receiver is shown in **Fig.17**.

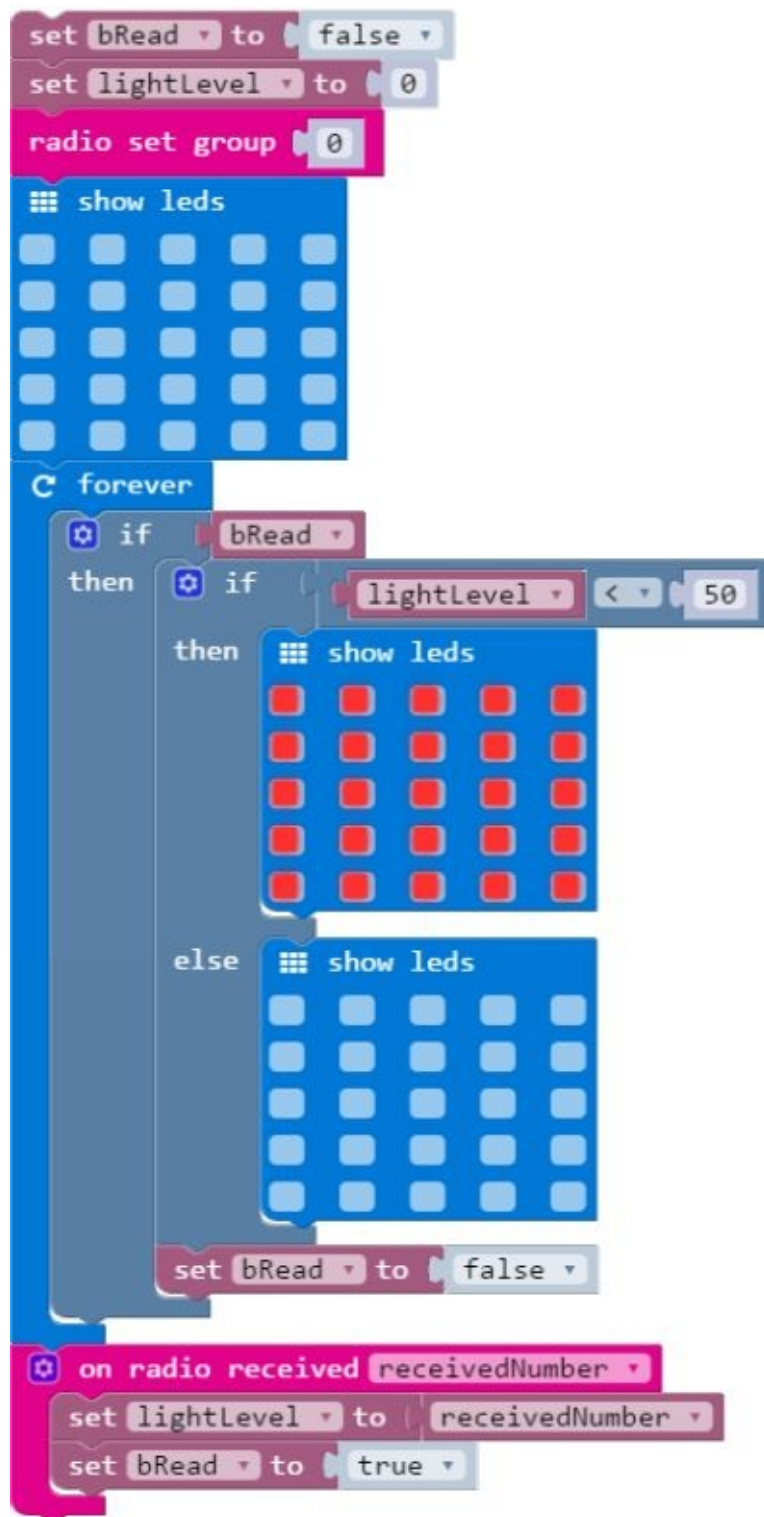


Fig.17

The JavaScript source code associated with the above diagram is shown in **Listing 14**.

Listing 14.

```

let bRead = false
let lightLevel = 0
radio.setGroup(0)

```

```

basic.showLeds(`
  .....
  .....
  .....
  .....
  .....
`)
basic.forever(() => {
  if (bRead) {
    if (lightLevel < 50) {
      basic.showLeds(`
        # # # # #
        # # # # #
        # # # # #
        # # # # #
        # # # # #
      `)
    } else {
      basic.showLeds(`
        .....
        .....
        .....
        .....
        .....
      `)
    }
    bRead = false
  }
})
radio.onDataPacketReceived(({receivedNumber}) => {
  lightLevel = receivedNumber
  bRead = true
})

```

In this code, the value of light intensity received from the transmitter is saved in the variable **lightLevel**. If this value turns out to be smaller than 50 (we can vary this value if needed), then the LEDs on the screen are driven ON. Otherwise, if light intensity is larger than 50, the LEDs will be driven OFF.

Project 8

This project illustrates how to design a simple system that will detect the pitch of a BBC micro:bit board. The system contains two BBC micro:bit boards, a transmitter and receiver. The BBC micro:bit that acts as a transmitter detects a pitch and sends a warning message to the receiver. The value of a pitch is then displayed on the LED screen of the receiver. This project can be a template for building a simple alarm system. The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.18**.

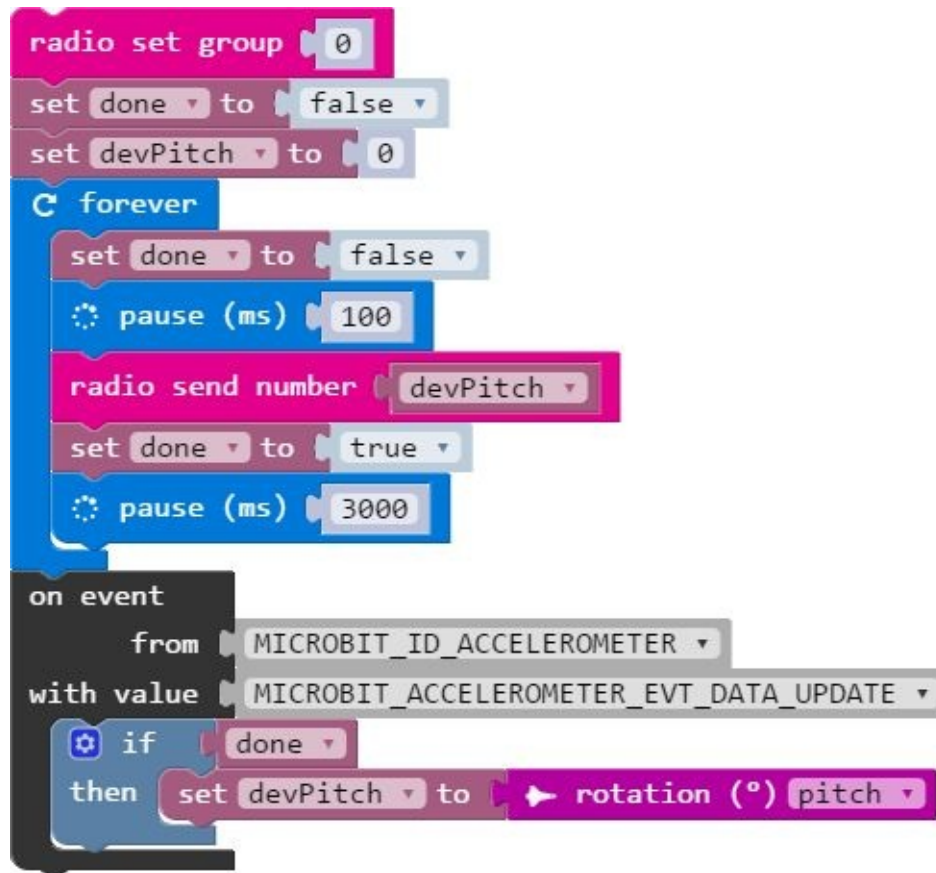


Fig.18

The JavaScript source code associated with the above diagram is shown in **Listing 15**.

Listing 15.

```
radio.setGroup(0)
let done = false
let devPitch = 0
basic.forever(() => {
  done = false
  basic.pause(100)
  radio.sendNumber(devPitch)
```

```

done = true
basic.pause(3000)
})
control.onEvent(EventBusSource.MICROBIT_ID_ACCELEROMETER,
                EventBusValue.MICROBIT_ACCELEROMETER_EVT_DATA_UPDATE,
()) => {
    if (done) {
        devPitch = input.rotation(Rotation.Pitch)
    }
})

```

In this source code, the **control.onEvent()** function raises an event in the event bus. This function takes two parameters, **EventBusSource** and **EventBusValue**. The **EventBusSource** parameter determines what component raised an event. In our case, this will be **MICROBIT_ID_ACCELEROMETER**. The type of the event is determined by **MICROBIT_ACCELEROMETER_EVT_DATA_UPDATE** constant assigned to **EventBusValue**.

The value of a pitch is saved in the **devPitch** variable. This value is then sent by the **radio.sendNumber()** function to the receiver.

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.19**.

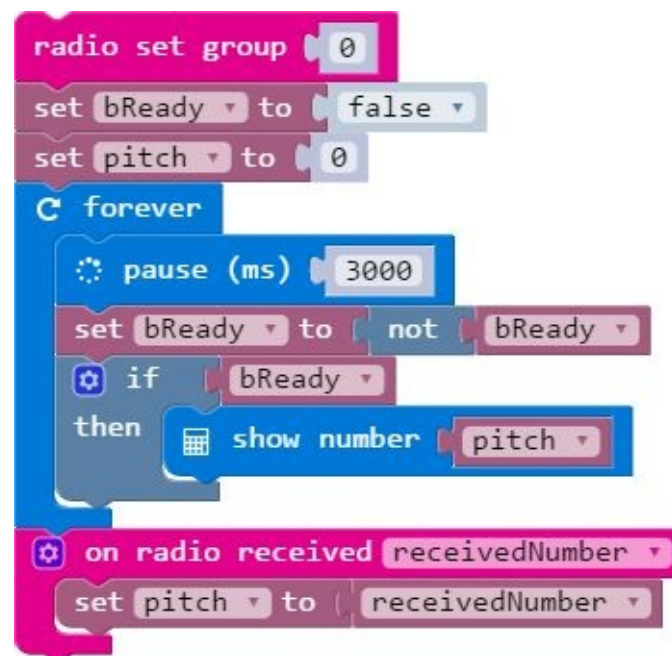


Fig.19

The JavaScript source code associated with the above diagram is shown in **Listing 16**.

Listing 16.

```
radio.setGroup(0)
let bReady = false
let pitch = 0
basic.forever(() => {
  basic.pause(3000)
  bReady = !(bReady)
  if (bReady) {
    basic.showNumber(pitch)
  }
})
radio.onDataPacketReceived(({receivedNumber}) => {
  pitch = receivedNumber
})
```

When data arrives via a radio channel, the **onDataPacketReceived** event arises and the data obtained is saved in the **pitch** variable. This value is then output on the LED screen.

Advanced wireless applications

This chapter contains projects that allow to build BBC micro:bit wireless applications with a simple command interface using a PC serial interface (UART). With UART we can configure the parameters of our applications from either the terminal application such as “PuTTY” or the program written in some popular programming language.

To operate with a serial interface in the PXT environment we can use the functions from the “**Serial**” block on the “**Advanced**” tab in the Microsoft PXT editor. **Note** that applications working with UART will operate properly after resetting the BBC micro:bit. This is done by pressing the “**RESET**” button after writing a binary executable (.hex) file into a board.

Project 1

This project illustrates how to design a simple system that will detect the pitch of a BBC micro:bit board. The system contains two BBC micro:bit boards, a transmitter and receiver. The BBC micro:bit that acts as a transmitter detects a pitch and sends a warning message to the receiver. The value of a pitch is then transferred to the serial port and displayed on the LED screen of the receiver.

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.20**.

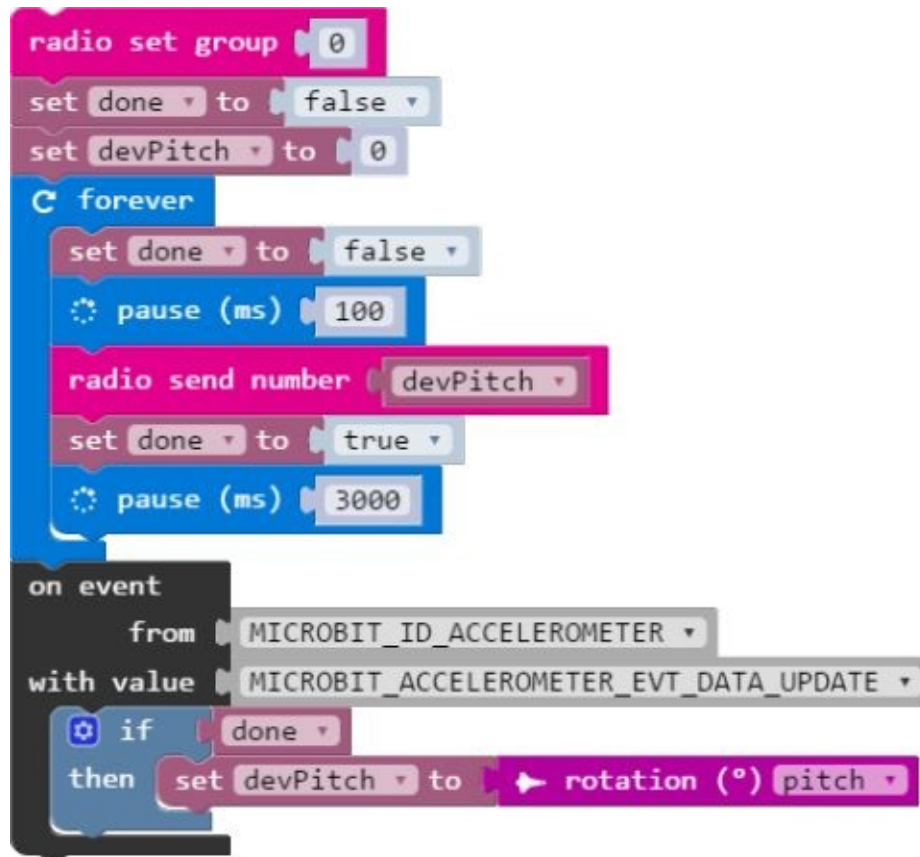


Fig.20

The JavaScript source code associated with the above diagram is shown in **Listing 17**.

Listing 17.

```
radio.setGroup(0)
let done = false
let devPitch = 0
basic.forever(() => {
  done = false
  basic.pause(100)
  radio.sendNumber(devPitch)
```



```

done = true
basic.pause(3000)
})
control.onEvent(EventBusSource.MICROBIT_ID_ACCELEROMETER,
                EventBusValue.MICROBIT_ACCELEROMETER_EVT_DATA_UPDATE,
() => {
    if (done) {
        devPitch = input.rotation(Rotation.Pitch)
    }
})

```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.21**.



Fig.21

The JavaScript source code associated with the above block diagram is shown in **Listing 18**.

Listing 18.

```

radio.setGroup(0)
let bReady = false
let pitch = 0

```

```

basic.forever(() => {
  basic.pause(3000)
  bReady = !(bReady)
  if (bReady) {
    serial.writeString("Pitch: ")
    serial.writeNumber(pitch)
    serial.writeString("\r\n")
    basic.showNumber(pitch)
  }
})
radio.onDataPacketReceived(({receivedNumber}) => {
  pitch = receivedNumber
})

```

In this code, the sequence

```

serial.writeString("Pitch: ")
serial.writeNumber(pitch)
serial.writeString("\r\n")

```

writes the line to the serial port assigned to the BBC micro:bit.

To view the data received by UART, we can launch some terminal emulator application on the PC where the BBC micro:bit is connected to. For this particular example, a popular utility called PuTTY running in Windows 10 was launched. To use PuTTY we should know the serial port assigned to our BBC micro:bit. In Windows, open “**Control Panel**” -> “**Device Manager**” and find the “**Ports (COM & LPT)**” option. Under this option you will see the list of available serial ports. On my PC, the serial port assigned to the BBC micro:bit is COM10 (**Fig.22**).

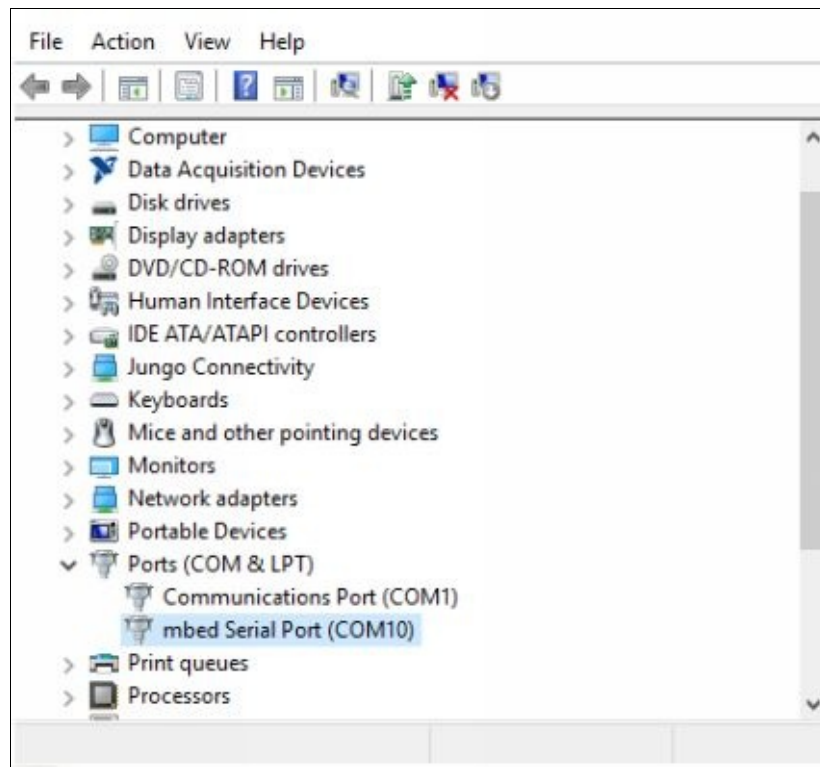


Fig.22

We also need to configure the desired speed of the port by opening the “**Properties**” tab and typing the suitable value (115200, in this example) in the “**Bits per second**” field (**Fig.23**).

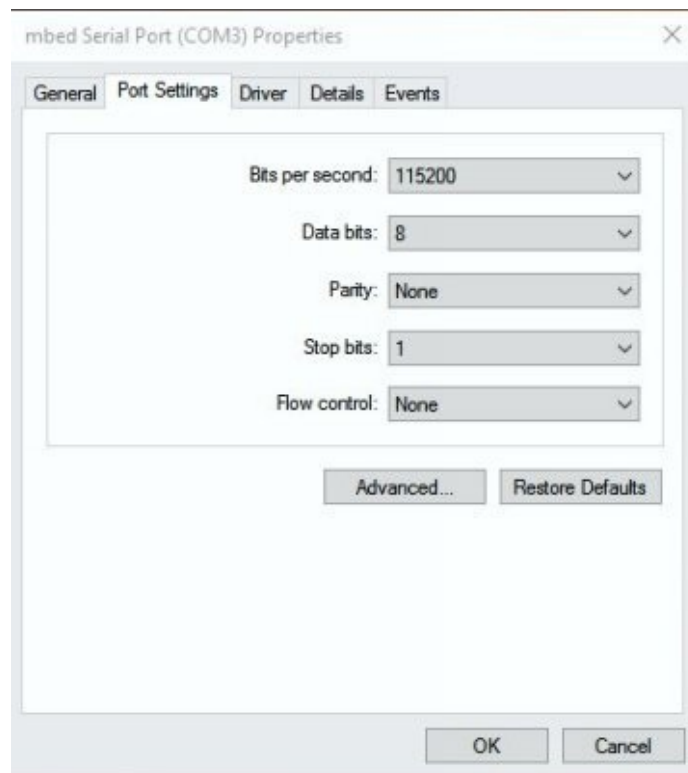


Fig.23

Then press **OK** and start off PuTTY.
In the opened “**PuTTY Configuration**” window assign “**Serial**” to the “**Connection type**”

parameter, then type the name of a serial interface (“**COM3**”, in this example) in the “**Serial line**” field. The last parameter to configure is “**Speed**” (in this example, it is set to 115200). After all parameters has been configured, press “**Open**” (**Fig.24**).

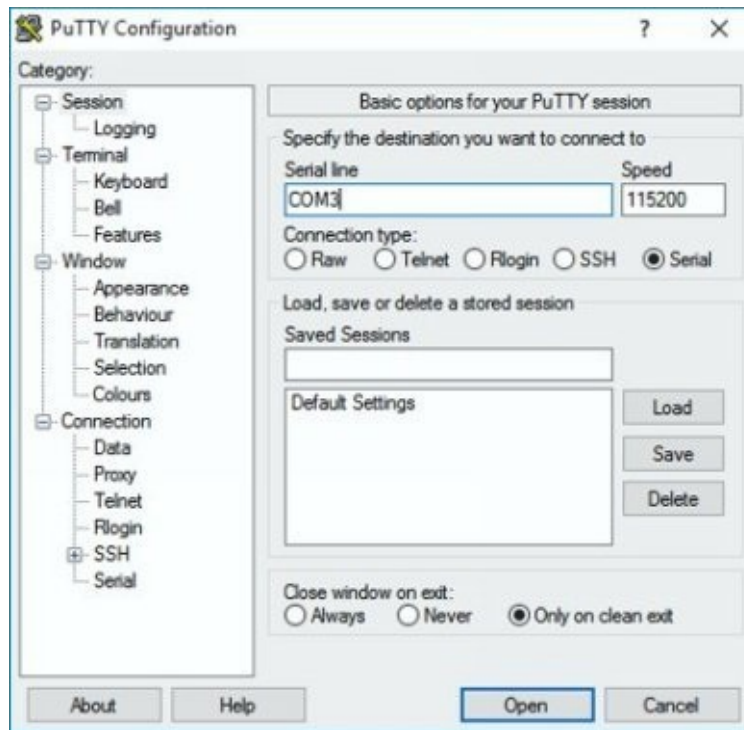


Fig.24

The terminal window when opened produces the output like the following (**Fig.25**).

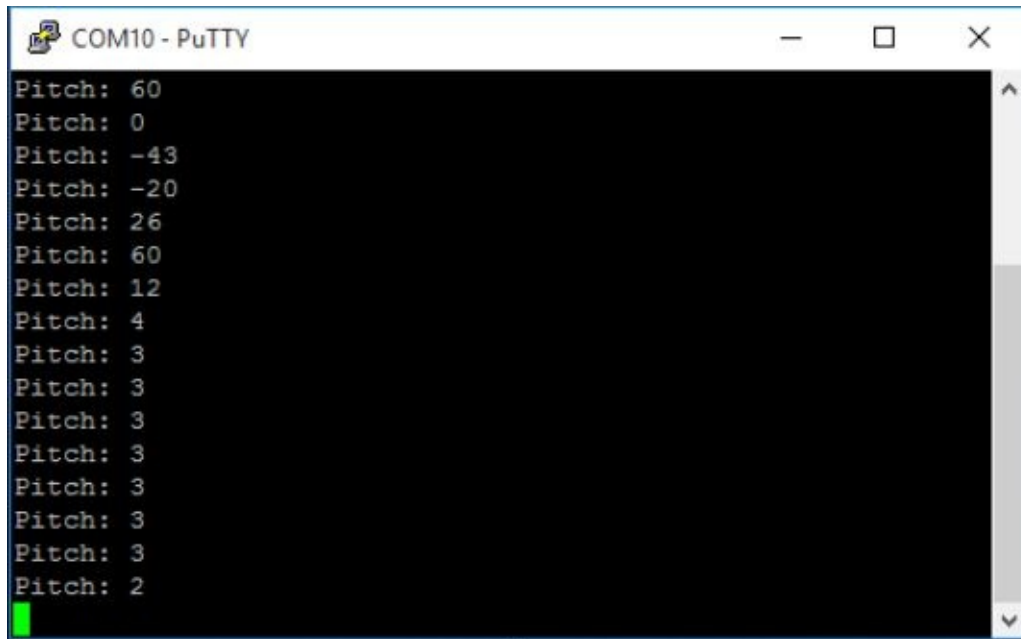


Fig.25

All projects that follow were developed in Linux OS (Ubuntu 16.04 LTS). We will discuss operating with serial interfaces in Ubuntu in the next project.

Project 2

This project illustrates how to pass the data from a serial interface (UART) to the remote BBC micro:bit board. In this system, the BBC micro:bit serving as a transmitter takes the text string from a serial port and transfers the data obtained via a radio channel to a BBC micro:bit receiver. The receiver then outputs the text string just obtained onto the LED screen.

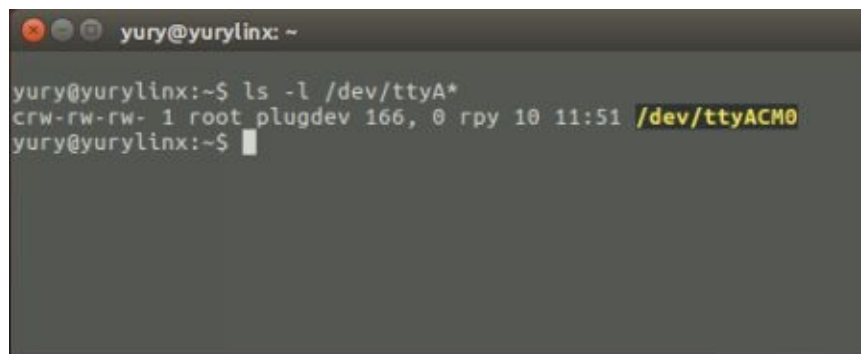
In Ubuntu 16.04 the device files associated with a serial interface appears as **/dev/ttyACMx**. To view the device file associated with the BBC micro:bit serial interface we should open the Terminal application and enter the command:

```
$ ls -l /dev/ttyA*
```

The output may look like the following:

```
$ ls -l /dev/ttyA*  
crw-rw-rw- 1 root plugdev 166, 0 rpy 10 11:51 /dev/ttyACM0
```

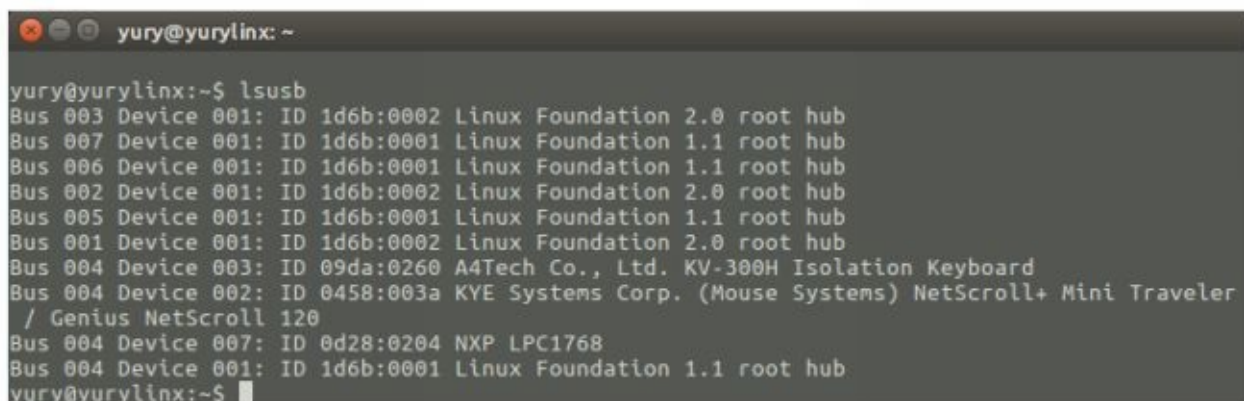
On my PC running Ubuntu 16.04, the above command produces the following output (**Fig.26**).



```
yury@yurylinx: ~  
yury@yurylinx:~$ ls -l /dev/ttyA*  
crw-rw-rw- 1 root plugdev 166, 0 rpy 10 11:51 /dev/ttyACM0  
yury@yurylinx:~$
```

Fig.26

You can also check new devices after BBC micro:bit has been attached by typing the **lsusb** command. On my PC, this command produces the following output (**Fig.27**).



```
yury@yurylinx: ~  
yury@yurylinx:~$ lsusb  
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub  
Bus 006 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub  
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 005 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub  
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub  
Bus 004 Device 003: ID 09da:0260 A4Tech Co., Ltd. KV-300H Isolation Keyboard  
Bus 004 Device 002: ID 0458:003a KYE Systems Corp. (Mouse Systems) NetScroll+ Mini Traveler / Genius NetScroll 120  
Bus 004 Device 007: ID 0d28:0204 NXP LPC1768  
Bus 004 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub  
yury@yurylinx:~$
```

Fig.27

The BBC micro:bit is represented by the following line:

Bus 004 Device 007: ID 0d28:0204 NXP LPC1768

To test our system we should connect the BBC micro:bit (“transmitter”) to the USB port of the PC and check if a new device **/dev/ttyACMx** has appeared. On my PC, this will be **/dev/ttyACM0**. Then we can enter, for example, the **echo** command as is shown below:

```
$ echo 1234 > /dev/ttyACM0
```

This command passes the string “1234” to the BBC micro:bit receiver which outputs this sequence on the LED screen.

One more way to test our system is to develop a simple Python application operating with a serial interface (ensure that a **pySerial** module has been installed). The simple Python script developed in the IDLE Python 3 environment looks like the following (**Listing 19**).

Listing 19.

```
import serial

port = serial.Serial("/dev/ttyACM0", baudrate=115200, timeout=3.0)
port.write("RADIO UART Test!\r\n".encode('utf-8'))
port.close()
```

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.28**.

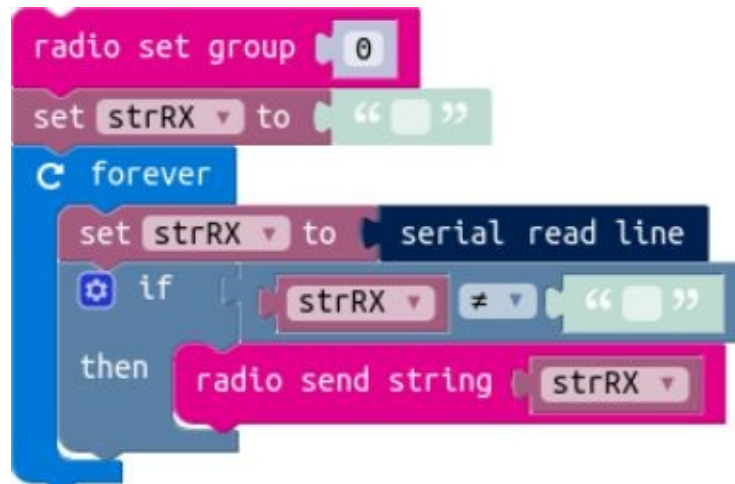


Fig.28

The JavaScript source code associated with the above block diagram is shown in **Listing 20**.

Listing 20.

```
radio.setGroup(0)
let strRX = ""
basic.forever(() => {
    strRX = serial.readLine()
    if (strRX != "") {
        radio.sendString(strRX)
    }
})
```

The block diagram of the application running on the BBC micro:bit receiver is shown **Fig.29**.

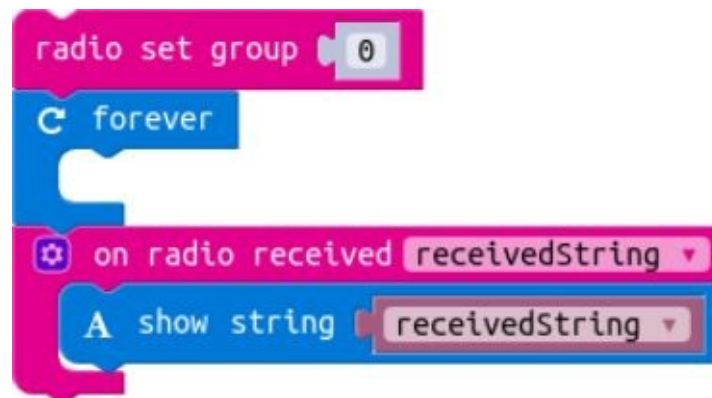


Fig.29

The JavaScript source code associated with the above block diagram is shown in **Listing 21**.

Listing 21.

```
radio.setGroup(0)
basic.forever(() => {

})
radio.onDataPacketReceived(({receivedString}) => {
    basic.showString(receivedString)
})
```


Project 3

This project demonstrates how to control the brightness of a remote LED screen from UART. The application running on the BBC micro:bit transmitter reads the command line from the serial interface. The command line itself looks like the “Brightness value”, where the value is between 0 – 255.

The application running on the BBC micro:bit receiver receives the data, parses the command line and set the brightness of the LED screen according to the “value” passed in the command line.

The block diagram of the application running on the BBC micro:bit transmitter is the same as that described in the previous project (see **Fig.28** and **Listing 20**).

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig. 30**.

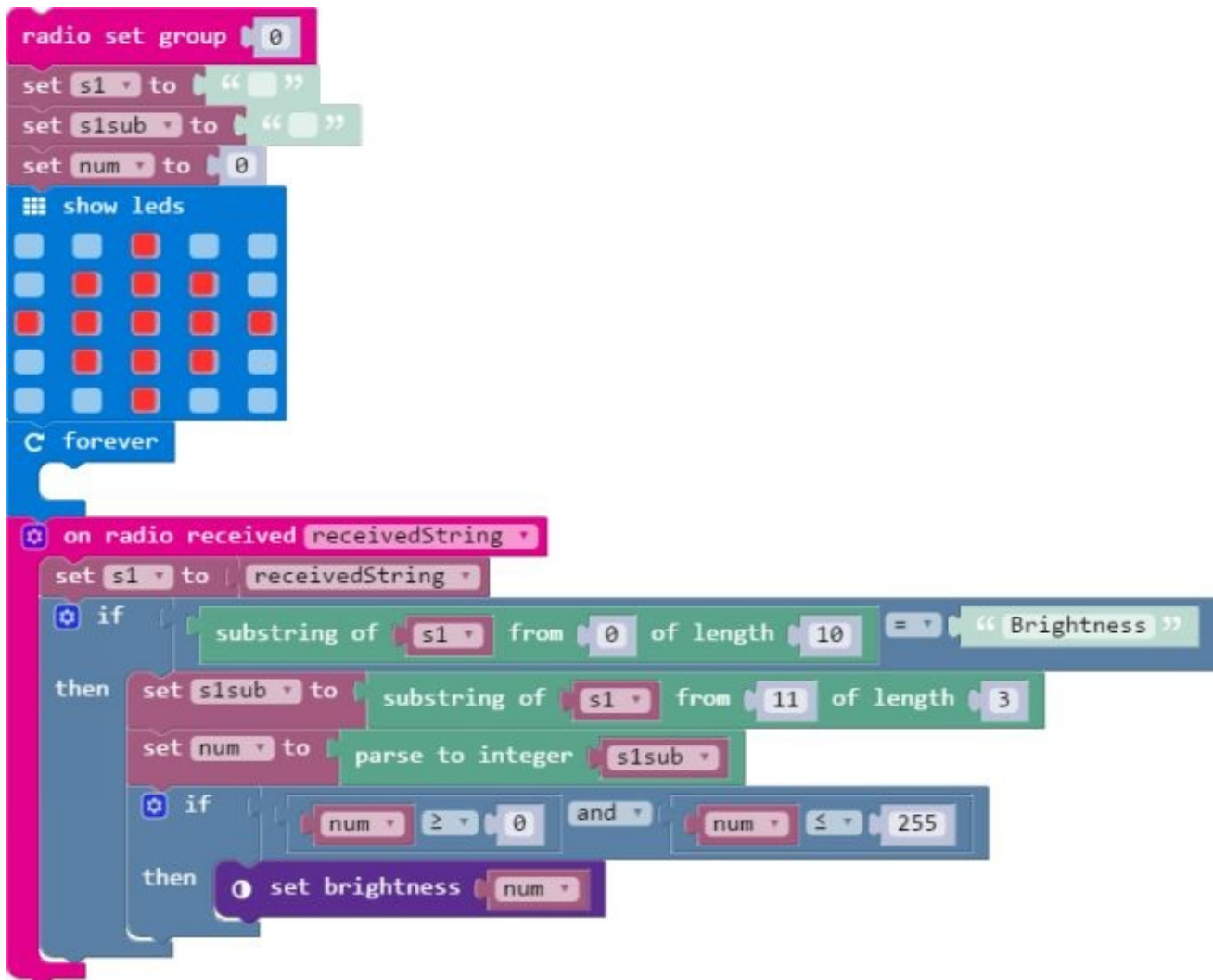


Fig.30

The JavaScript source code associated with the above block diagram is shown in **Listing 22**.

Listing 22.

```
radio.setGroup(0)
let s1 = ""
let s1sub = ""
let num = 0
basic.showLeds(`
  ..# ..
  .###.
  #####
  .###.
  ..# ..
  `)
basic.forever(() => {

})
radio.onDataPacketReceived(({receivedString}) => {
  s1 = receivedString
  if (s1.substr(0, 10) == "Brightness") {
    s1sub = s1.substr(11, 3)
    num = parseInt(s1sub)
    if (num >= 0 && num <= 255) {
      led.setBrightness(num)
    }
  }
})
```

To test the system, we can enter the following commands in the Terminal window:

```
$ echo "Brightness:100" > /dev/ttyACM0
$ echo "Brightness:60" > /dev/ttyACM0
$ echo "Brightness: 6" > /dev/ttyACM0
$ echo "Brightness:250" > /dev/ttyACM0
$ echo "Brightness:25" > /dev/ttyACM0
```

Equally, we can test the system using the following Python script (**Listing 23.**)

Listing 23.

```
import serial
```

```
port = serial.Serial("/dev/ttyACM0", baudrate=115200, timeout=3.0)
port.write("Brightness:49\r\n".encode('utf-8'))
port.close()
```

By changing the parameter of the **write()** function we can set the desired brightness.

Project 4

This project demonstrates how to set the duty cycle of a PWM signal on digital pin **P0** via a radio channel. The application running on the BBC micro:bit transmitter passes the value of a duty cycle (between 10 and 255) to the receiver. The BBC micro:bit receiver adjusts the duty cycle of the PWM signal on pin **P0** of the BBC micro:bit receiver. Pin **P0**, in turn, drives the network consisting of the LED and resistor of 270 Ohms (**Fig.31**). The brightness of the LED will increase/reduce as the duty cycle of PWM is increased/reduced.

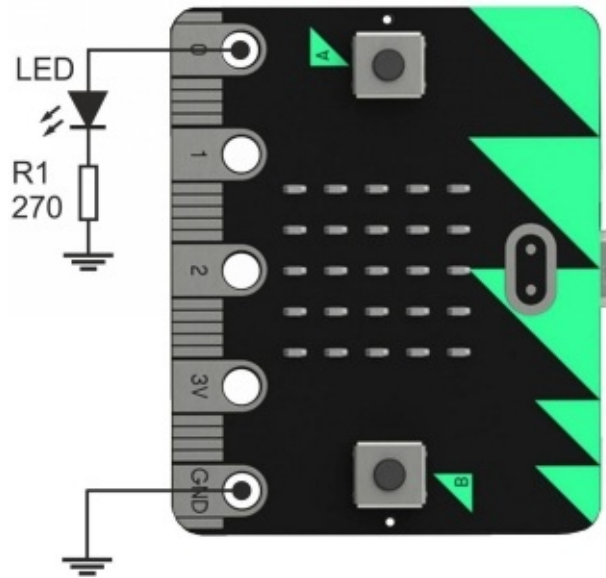


Fig.31

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.32**.

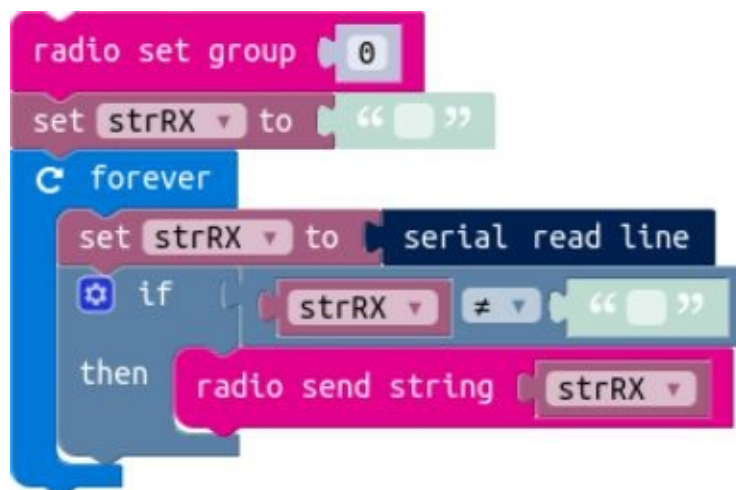


Fig.32

The JavaScript source code associated with the above diagram is shown in **Listing 24**.

Listing 24.

```

radio.setGroup(0)
let strRX = ""
basic.forever(() => {
    strRX = serial.readLine()
    if (strRX != "") {
        radio.sendString(strRX)
    }
})

```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.33**.

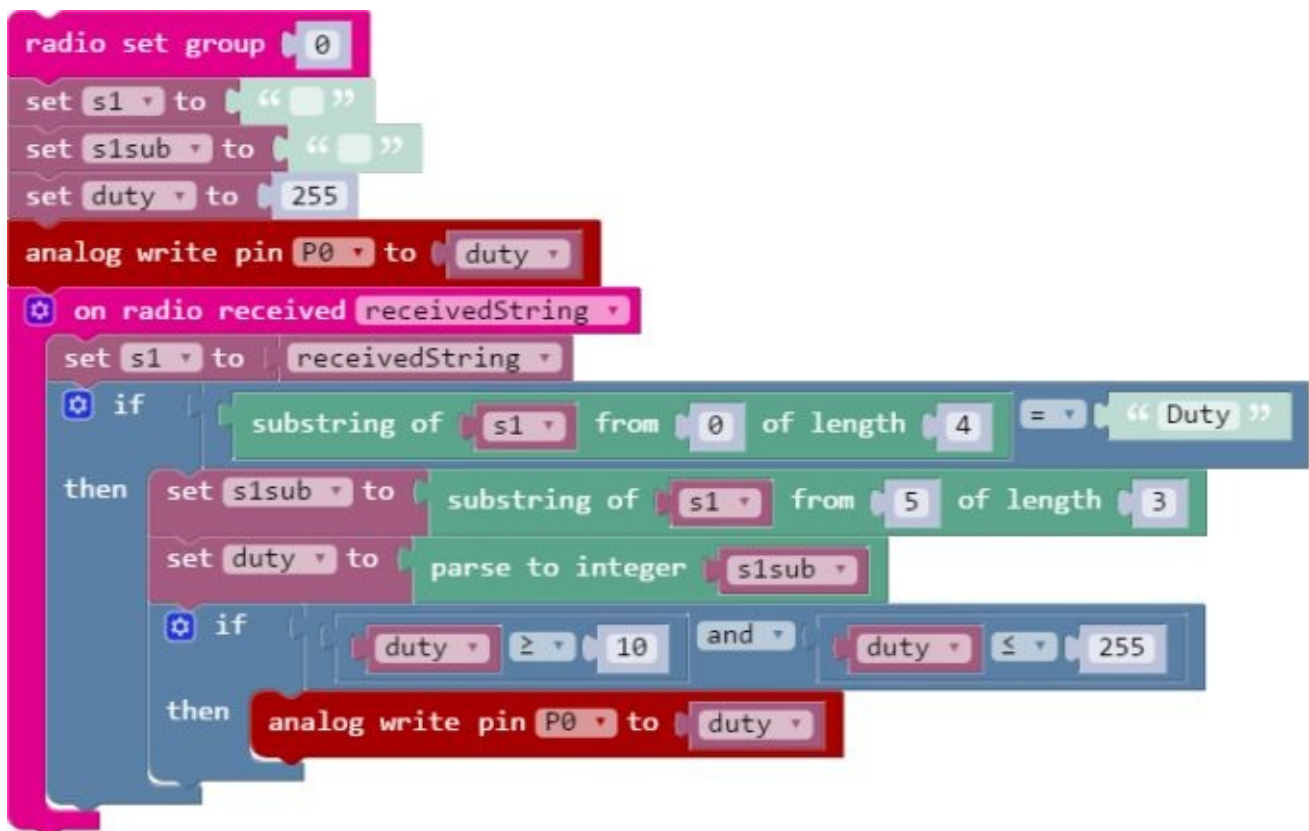


Fig.33

The JavaScript source code associated with the above block diagram is shown in **Listing 25**.

Listing 25.

```

radio.setGroup(0)
let s1 = ""
let s1sub = ""
let duty = 255
pins.analogWritePin(AnalogPin.P0, duty)

```

```
radio.onDataPacketReceived(({receivedString}) => {
  s1 = receivedString
  if (s1.substr(0, 4) == "Duty") {
    s1sub = s1.substr(5, 3)
    duty = parseInt(s1sub)
    if (duty >= 10 && duty <= 255) {
      pins.analogWritePin(AnalogPin.P0, duty)
    }
  }
})
```

To test the system, we should write a command to the device file corresponding to the serial interface where the BBC micro:bit transmitter is connected to. In Ubuntu 16.04, this may look like the following:

```
$ echo "Duty 20" > /dev/ttyACM0
$ echo "Duty 200" > /dev/ttyACM0
$ echo "Duty 20" > /dev/ttyACM0
$ echo "Duty 50" > /dev/ttyACM0
$ echo "Duty 250" > /dev/ttyACM0
$ echo "Duty 25" > /dev/ttyACM0
$ echo "Duty 111" > /dev/ttyACM0
```

We can also use a Python application to configure the duty cycle of PWM (**Listing 26**).

Listing 26.

```
import serial

port = serial.Serial("/dev/ttyACM0", baudrate=115200, timeout=3.0)
port.write("Duty:200\r\n".encode('utf-8'))
port.close()
```

This code allows to set the duty cycle of the PWM signal on pin **P0** equal to 200.

Project 5

This project demonstrates how to drive the LED connected to pin **P0** ON/OFF. The application running on the BBC micro:bit transmitter passes the command to the receiver that, in turn, drives the LED on pin **P0** ON/OFF.

The command line to be transferred to the receiver looks like the following:

P0 {ON|OFF}

For example, to drive the LED on pin **P0** ON we should enter

P0 ON

Conversely, to drive the LED OFF we should use

P0 OFF

The LED connected to pin **P0** on the BBC micro:bit receiver (**Fig.34**) will be ON/OFF depending on the command received.

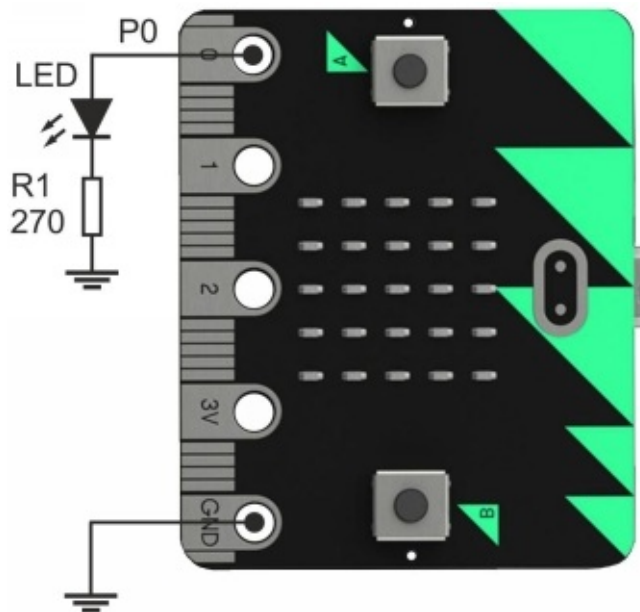


Fig.34

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.35**.

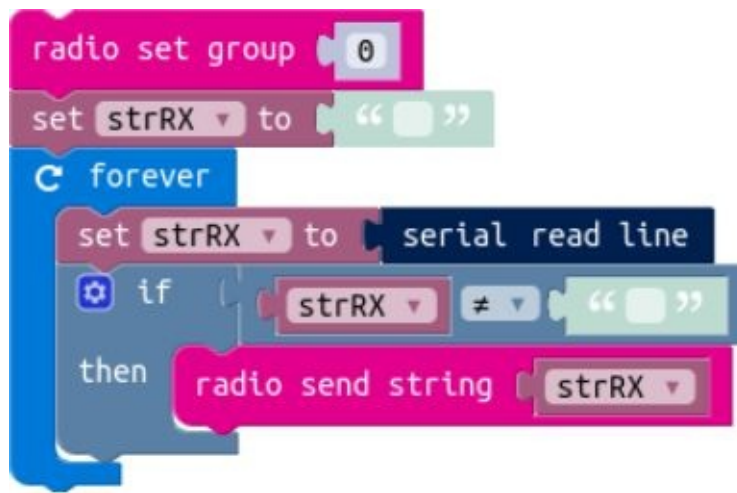


Fig.35

The JavaScript source code associated with the above diagram is shown in **Listing 27**.

Listing 27.

```

radio.setGroup(0)
let strRX = ""
basic.forever(() => {
  strRX = serial.readLine()
  if (strRX != "") {
    radio.sendString(strRX)
  }
})

```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.36**.

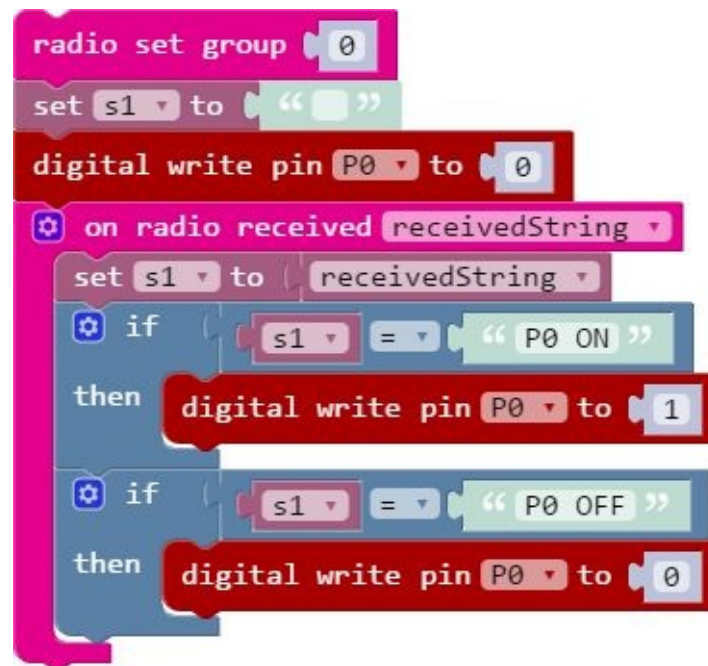


Fig.36

The JavaScript source code associated with the above diagram is shown **Listing 28**.

Listing 28.

```
radio.setGroup(0)
let s1 = ""
pins.digitalWritePin(DigitalPin.P0, 0)
radio.onDataPacketReceived(({receivedString}) => {
  s1 = receivedString
  if (s1 == "P0 ON") {
    pins.digitalWritePin(DigitalPin.P0, 1)
  }
  if (s1 == "P0 OFF") {
    pins.digitalWritePin(DigitalPin.P0, 0)
  }
})
```

To test applications in Ubuntu, we can write two following commands to the serial interface (/dev/ttyACM0 on my PC) on the BBC micro:bit transmitter:

```
$ echo "P0 ON" > /dev/ttyACM0
$ echo "P0 OFF" > /dev/ttyACM0
```

Alternatively, we can drive pin **P0** by launching the Python application whose source code is shown in **Listing 29**.

Listing 29.

```
import serial
import time

cnt = 0
while (cnt < 5):
    port = serial.Serial("/dev/ttyACM0", baudrate=115200, timeout=3.0)
    port.write("P0 ON\n".encode('utf-8'))
    time.sleep(3)
    port.write("P0 OFF\n".encode('utf-8'))
    time.sleep(3)
    cnt += 1
port.close()
```

This application drives the LED ON/OFF 5 times, then exits.

Project 6

This project demonstrates how to read the light intensity of the light sensor located on the remote BBC micro:bit. The application running on the BBC micro:bit transmitter passes the value obtained from the light sensor to the BBC micro:bit receiver. The application running on the receiver then transfers the data obtained via a serial interface.

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.37**.

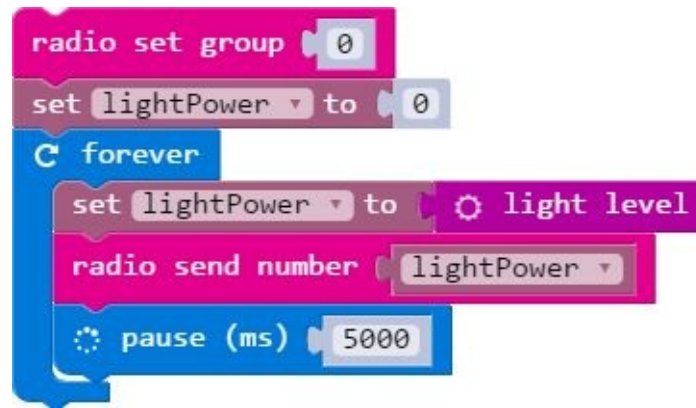


Fig.37

The JavaScript source code associated with the above block diagram is shown in **Listing 30**.

Listing 30.

```
radio.setGroup(0)
let lightPower = 0
basic.forever(() => {
  lightPower = input.lightLevel()
  radio.sendNumber(lightPower)
  basic.pause(5000)
})
```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.38**.

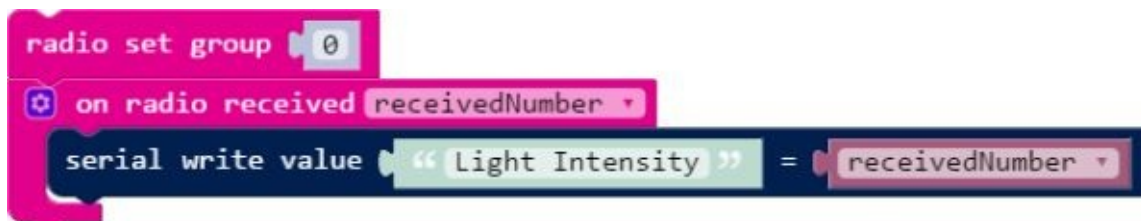


Fig.38

The JavaScript source code associated with the above block diagram is shown in **Listing 31**.

Listing 31.

```
radio.setGroup(0)
radio.onDataPacketReceived(({receivedNumber}) => {
  serial.writeValue("Light Intensity", receivedNumber)
})
```

To read the data received by the serial interface **/dev/ttyACM0** (Ubuntu) we can use the **cat** commands as is shown below:

```
$ cat < /dev/ttyACM0
Light Intensity:65
$ cat < /dev/ttyACM0
Light Intensity:133
$ cat < /dev/ttyACM0
Light Intensity:105
```

We can also use the application written in Python 3 to read the data on the serial interface (**Listing 32**).

Listing 32.

```
import serial

port = serial.Serial('/dev/ttyACM0', 115200)

try:
    while True:
        s1 = port.readline()
        print(s1.decode('utf_8', 'strict'))
except KeyboardInterrupt:
    print ("Serial port is closing...")
    port.close()
```

This application produces the following output (taken from the IDLE Python 3 Shell):

```
Light Intensity:71
Light Intensity:96
Light Intensity:158
Light Intensity:71
```

Light Intensity:64

Project 7

This project allows to build the system measuring analog input voltage provided by some analog sensor on pin **P0** of the remote transmitter. The application running on the BBC micro:bit transmitter transfers the data obtained to the receiver. The receiver, in turn, write the data to its serial interface. In this project, an analog sensor on pin **P0** is simulated by the voltage divider formed by resistor R1 and potentiometer R2 (**Fig.39**).

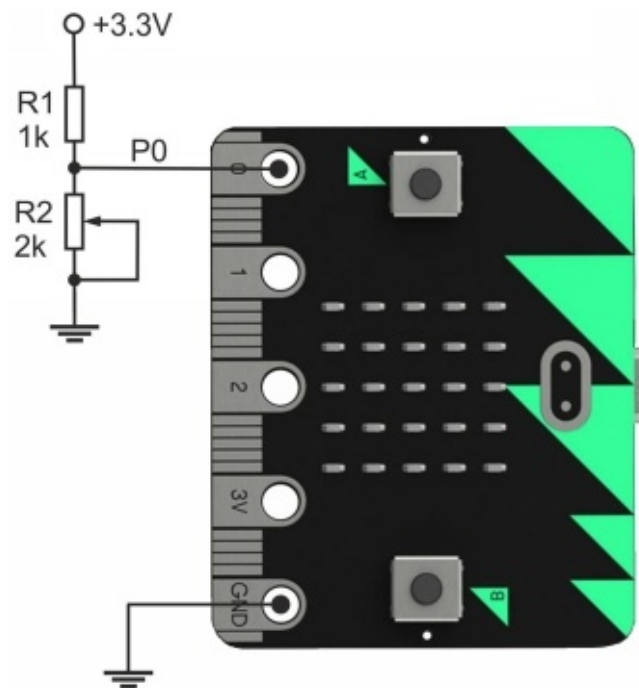


Fig.39

The block diagram of the application running on the BBC micro:bit transmitter is shown in **Fig.40**.

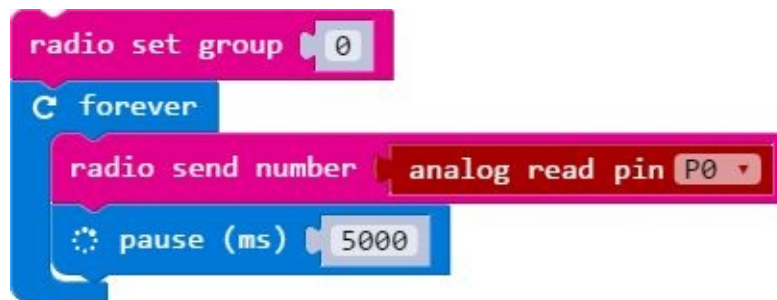


Fig.40

The JavaScript source code associated with the above diagram is shown in **Listing 33**.

Listing 33.

```

radio.setGroup(0)
basic.forever(() => {
    radio.sendNumber(pins.analogReadPin(AnalogPin.P0))
    basic.pause(5000)
})

```

The block diagram of the application running on the BBC micro:bit receiver is shown in **Fig.41**.

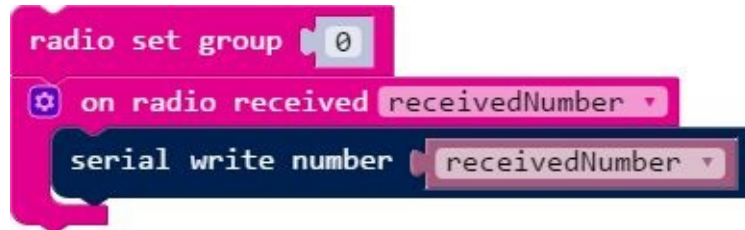


Fig.41

The JavaScript source code associated with the above diagram is shown in **Listing 34**.

Listing 34.

```

radio.setGroup(0)
radio.onDataPacketReceived(({receivedNumber}) => {
    serial.writeNumber(receivedNumber)
})

```

To test the system we can use the application written in Python 3 whose source code is shown in **Listing 35**.

Listing 35.

```

import serial
import time

Dn = 0
vin = 0.0
#LSB = 0.00322 # 3.3V/1024
LSB = 0.003115 # 3.186 / 1024
rx_buff = ""
port = serial.Serial('COM10', 115200)

try:

```

```
port.reset_input_buffer()
while True:
    if (port.in_waiting != 0):
        rx_buff = port.read(port.in_waiting)
        port.reset_input_buffer()
        Dn = int(rx_buff)
        vin = LSB * Dn
        print("Analog Voltage on Pin P0: " + "{:.3f}".format(vin) + " V")
        time.sleep(2)
except KeyboardInterrupt:
    print ("Serial port is closing...")
    port.close()
```

This application produces the following output (taken from IDLE Python 3 shell):

```
Analog Voltage on Pin P0: 2.514 V
Analog Voltage on Pin P0: 2.511 V
Analog Voltage on Pin P0: 2.171 V
Analog Voltage on Pin P0: 2.174 V
Analog Voltage on Pin P0: 1.997 V
Analog Voltage on Pin P0: 1.997 V
Analog Voltage on Pin P0: 1.741 V
Analog Voltage on Pin P0: 1.741 V
Analog Voltage on Pin P0: 2.037 V
```