Maria Tudor Ioana Culic Alexandru Radovici

Getting started guide for Intel® Galileo and Intel® Edison* using Wyliodrin

*compatible with Arduino Breakout Board

Getting started guide for Intel® Galileo and Intel® Edison* using Wyliodrin

*compatible with Arduino breakout board

Maria Tudor, Ioana Culic, Alexandru Radovici

We would like to thank Cristian Rusu, Andrei Decu, Valeriu Moldovan and Răzvan Matei.

Illustrations by Ovidiu Stoica.

This report has been commissioned and paid for by Intel®.

[1] http://creativecommons.org/licenses/by-sa/4.0/

Contents

What is Intel® Galileo?

Now that you own an Intel® Galileo here are some interesting things you should know about it.

Intel® Galileo simulates a micro-controller board that enables you to quickly and easily bring your ideas to life. In other words it is a tiny computer, with about the same computing power as an Intel® Pentium 2 processor. It is specifically designed for makers, students, educators, and Do It Yourself (DIY) electronics enthusiasts.

Galileo is an Embedded Computing platform specially designed to interact with its environment.

For a better understanding, with Intel® Galileo you can make your own devices, for instance a lamp that turns on when you clap your hands, a flower that waters itself or any other DIY projects. You can do all this while benefiting from the power of a fully operating system and any programming language.

Intel® Galileo provides all the features of a micro-controller, while offering the possibility of connecting it to the Internet. You will be able to build systems that can communicate with each other and access online services.

Intel® Galileo gives you the opportunity to create the gadgets you always dreamed of.

On a more technical approach, Galileo is an embedded computer board based on the Intel® Quark SoC X1000 Application Processor, a 32-bit Intel® Pentium-class system on a chip. It is the first board based on Intel® architecture designed to be hardware and software pin-compatible with Arduino shields designed for the Uno R3. It is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the I/O pins.

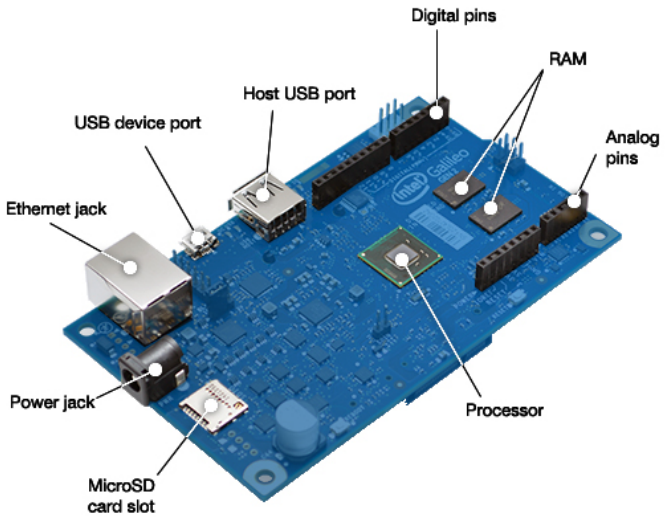

Figure 1: Intel® Galileo board

Key features of Intel® Galileo

Let's take a look at Intel® Galileo's design. It consists of:

- Intel® Quark™ SoC X1000 application processor, a 32-bit, single-core, single-thread, Intel® Pentium® processor instruction set architecture (ISA)-compatible, operating at speeds up to 400 MHz.

- 256 MB DDR3, 512 kb embedded SRAM, 8 MB NOR Flash, and 8 kb EEPROM standard on the board, plus support for microSD card up to 32 GB.
- Support for a wide range of industry standard I/O interfaces, including a full-sized mini-PCI Express* slot, 100 Mb Ethernet port, microSD* slot, USB host port, and USB client port.
- Hardware and pin compatibility with a wide range of Arduino Uno R3 shields.
- Programmable through the Arduino integrated development environment (IDE) that is supported on Microsoft Windows*, Mac OS*, and Linux host operating systems, Intel® XDK, Eclipse and *Wyliodrin*.
- Support for Yocto 1.4 Poky* Linux release.

What is new with the Intel® Galileo Gen 2 board?

Intel® Galileo is already at its second generation. Compared to its predecessor, it has:

- 6-pin 3.3V USB TTL UART header replaces the 3.5 mm jack RS-232 console port for Linux debug. New 6-pin connector mates with standard FTDI* USB serial cable (TTL-232R-3V3) and popular USB-to-Serial breakout boards. 12 GPIOs now fully native for greater speed and improved drive strength.
- 12-bit pulse-width modulation (PWM) for more precise control of servos and smoother response.
- Console UART1 can be redirected to Arduino headers in sketches, elim-

inating the need for soft-serial in many cases.

- 12V power-over-Ethernet (PoE) capable (PoE module installation required).
- Power regulation system changed to accept power supplies from 7V to 15V.
- Software support for C, C++, Python, and Node.js (Javascript).
- In addition to open source Yocto Linux, Intel® Galileo Gen 2 supports VxWorks (RTOS), and now Microsoft Windows is supported directly from Microsoft.

What does the board consist of to help you build amazing projects?

As previously described, the Intel® Galileo board is compatible with the Arduino Uno layout.

In Figure 1 you can visualize the following components:

1. 20 Digital Pins (0 -> 13, A0 -> A5).
2. 6 PWM Pins (3, 5, 6, 9, 10, 11).
3. 6 Analog Pins (A0 -> A5).
4. 2 Ground (GND) Pins.
5. 1 5V Pin.
6. 1 3.3V Pin.
7. 1 MicroSD Card Slot (on the left side).

8. 1 Power Jack (on the left side).
9. 1 Ethernet Jack (located above the power jack).
10. 1 USB device port (next to the ethernet jack).
11. 1 Host USB Port (after the USB device port).

What is Intel® Edison?

Let's see what you can do with your Intel® Edison board.

Intel® Edison is a tiny computer that allows you to easily create the Internet of Things projects you always dreamed of.

It aims at enabling beginners, students, teachers, makers and any technology enthusiast to ease their everyday lives by using systems they've built themselves.

The Intel® Edison was designed with wearable devices in mind. Its small dimensions and the integrated Wi-Fi and Bluetooth make it a great component for any wearable device.

Figure 2 depicts the emplacement and the numbering of the pins on the Intel® Edison breakout board.

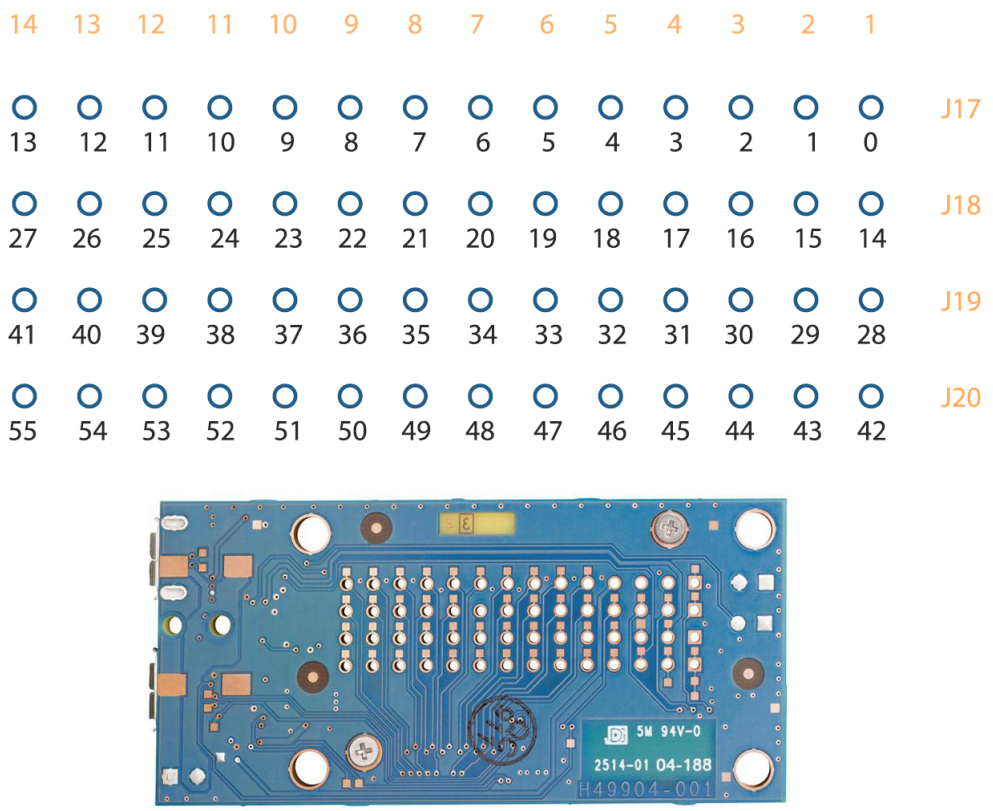

Figure 2: Pins on Intel® Edison breakout board

The Edison can be connected to an expansion board that gives you the possibility to access up to 40 pins. What is important to know is that the board itself has no ADC, however the expansion board does and they are connected via the SPI pins. Thus with the help of the expansion board you can create a system that actively interacts with its environment.

The board consists of a dual-core Intel® Atom CPU and 32-bit Intel® Quark™ micro-controller.

Key features of Intel® Edison

Let's take a look at Intel® Edison's design. It consists of:

- an Intel® SoC that includes a dual-core, dual-threaded Intel® Atom CPU at 500 MHz and a 32-bit Intel® Quark micro-controller at 100 MHz
- 1 GB RAM memory
- Integrated Wi-Fi Broadcom 43340 802.11 a/b/g/n
- Integrated Bluetooth 4.0
- Programmable through the Arduino integrated development environment (IDE) that is supported on Microsoft Windows*, Mac OS*, and Linux host operating systems, Intel® XDK, Eclipse and *Wyliodrin*.
- Support for Yocto 1.6.

What does the board consist of ?

The Intel® Edison Arduino board is compatible with the Arduino Uno layout, except that is has 4 PWM pins instead of 6). You can use a jumper to change the PWM mapping, by default the 11 and 12 pins are not PWM.

In order to access the Edison's pins and other capabilities, you can connect it to the expansion board it has attached when you buy it.

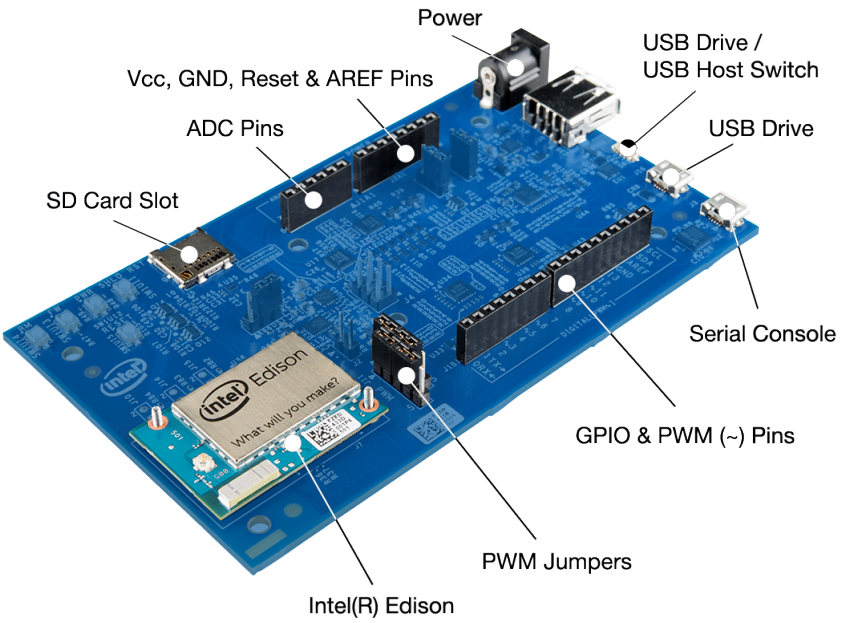

Figure 3: Intel® Edison together with the Arduino expansion board

The board consists of the following components (figure 3):

1. 20 Digital Pins (0 -> 13, A0 -> A5).
2. 4 PWM Pins.
3. 6 Analog Pins (A0 -> A5).
4. 1 I2C Pin.
5. 1 ICSP 6-pin header (SPI).
6. 3 Ground (GND) Pins.
7. 1 5V Pin.
8. 1 3.3V Pin.
9. 1 MicroSD Card Slot (on the left side).

10. 1 Power Jack (on the left side).
11. 1 USB device port for console connection.
12. 1 USB device port for mounting the disk.
13. 1 Host USB Port (if it is used, the USB device for for mounting the disk will not work)

Introduction to Electronics

You are going to build IoT projects around the Intel® Galileo board. However, the board is only one part of the projects: it does all the computing, but you also need I/O devices to connect to it. The devices are mainly sensors, buttons, LEDs and LCDs. In order to correctly connect the peripherals, you need to be acquainted to basic electronics notions, otherwise, you risk to burn the I/O devices and even the board.

Ohm's Law

Ohm's law states that in a circuit the current (I) is directly proportional to the applied voltage (U) and inversely proportional to the resistance (R) of the circuit.

$$I = \frac{U}{R} \tag{1}$$

Kirchhoff's Circuit Laws

Before the two laws will be stated, you need to understand the following notions:

- junction/node - the place where at least three conducting branches

meet

- loop - a closed path, including at least two nodes

Kirchhoff's First Law

Kirchhoff's First Law states that in a node, the sum of the currents is 0.

$$\sum_k i_k = 0 \tag{2}$$

Please keep in mind that currents have directions. Currents incoming have negative values, while currents outgoing have positive values.

Kirchhoff's Second Law

Kirchhoff's Second Law states that the sum of the voltage in a circuit loop is equal to the power source voltage.

$$\sum_k E_k = \sum_k R_k I_k \tag{3}$$

Example:

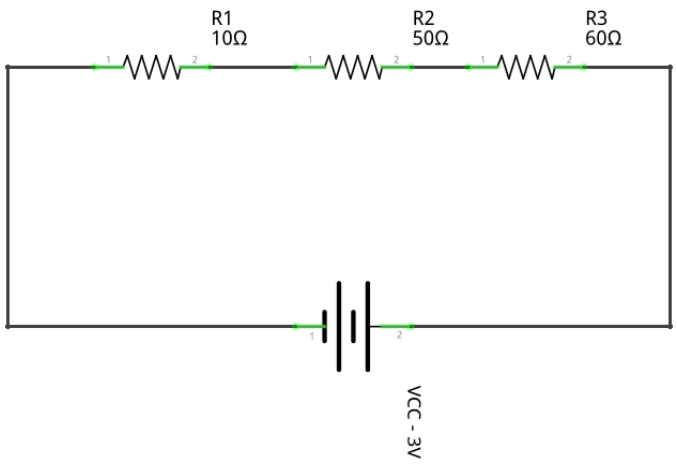

Figure 4: Kirchhoff's second law example

You have a 3V source and three resistors of different resistance (figure 4). The sum of voltage drops on each of them is equal to the source voltage.

$$I * (R1 + R2 + R3) = VCC1 => 0.25V + 1.25V + 1.5V = 3V \quad (4)$$

LED

This chapter explains how to correctly connect a LED to an Intel® Galileo board.

First of all, you need to know what a diode is.

A diode is an electronic component that has a positive and a negative side and it basically allows the current to flow only in one direction, from positive to negative.

The LED is also a diode. When current is flowing through the LED, it lights up. So in order to light up a LED you need to put the high voltage at the anode and the low voltage at the cathode.

Schematics

Taking into account the theory previously stated, you would build a circuit like the one in figure 5 to light up a LED.

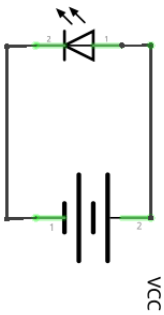

Figure 5: LED schematics example

You must take into account that in the projects you are going to build, the power source depicted will be replaced with an Intel® Galileo board.

There is only one tiny problem with the schematics in figure 5: it is a short circuit. That means there is no resistance to limit the current because the diode does not have any resistance at all. It just allows the current to flow. That can cause big problems (you can damage your Intel® Galileo, for example). To fix this, you need a resistor to limit the current flow (figure 6).

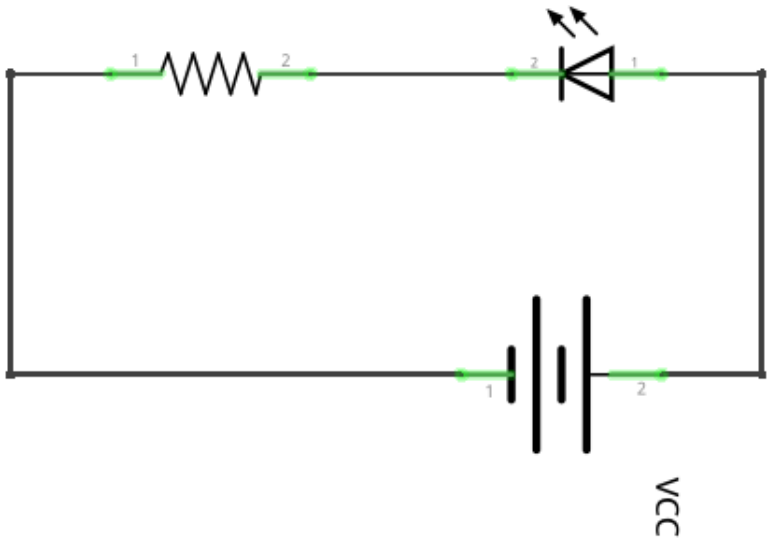

Figure 6: LED correct schematics example

Button

This chapter explains how to correctly connect a button to an Intel® Galileo board.

A button, also called a switch, is an electric component that can break an electrical circuit by interrupting the current.

When used in schematics, there are multiple possible symbols to depict it (figure 7).

(a) (b)

Figure 7: Button symbols

Also, figure 8 depicts an example of circuit that uses a switch.

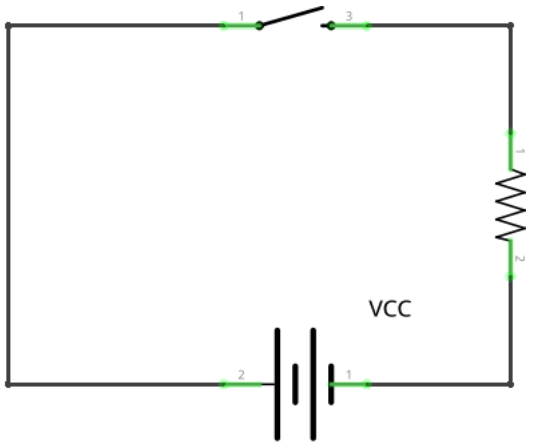

Figure 8: Button example circuit

When the button is pressed, it acts like a wire and it will let the current flow through the circuit. If the button is not pressed, the circuit is interrupted.

When a button is connected to a board, you can tell if the button was pressed by looking at the pin's value.

Let's see how you can connect a button to Intel® Galileo. The first possibility would be the one in figure 9, but it is *wrong*.

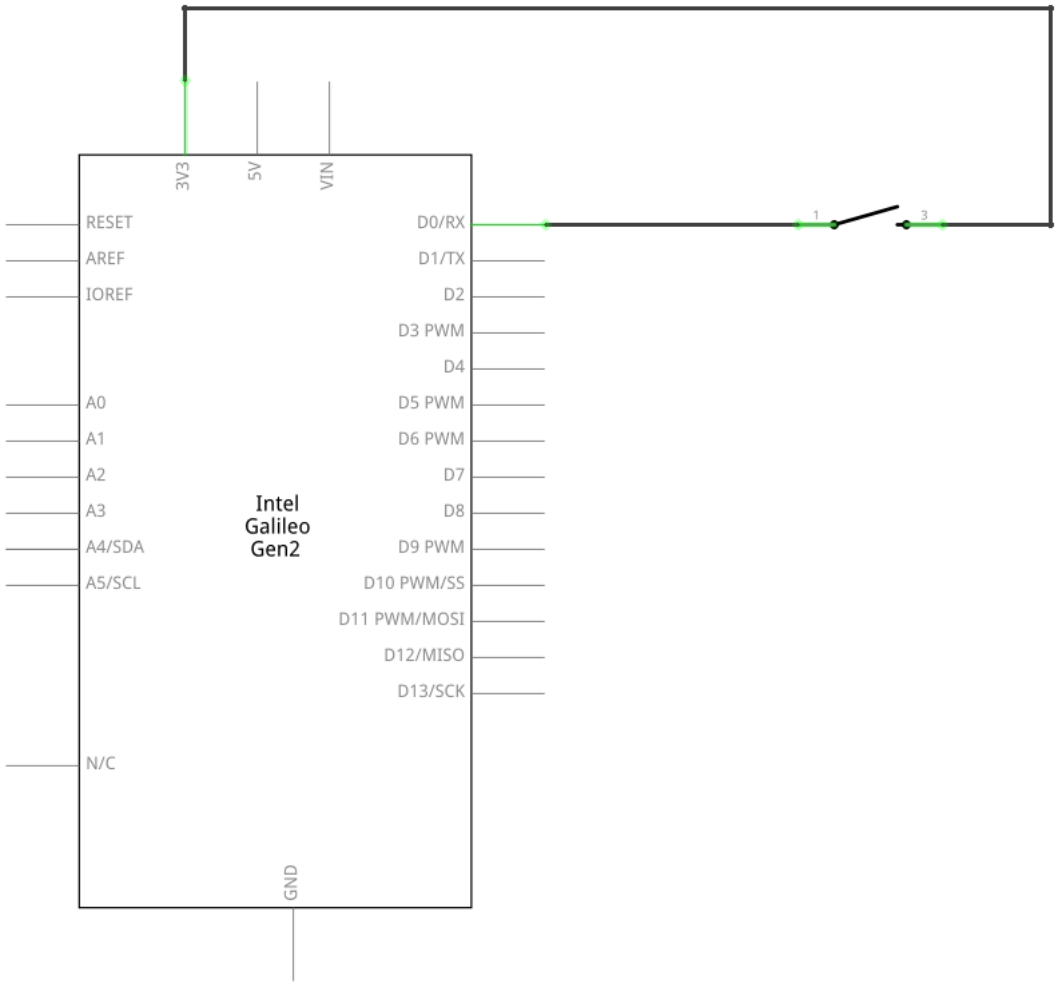

Figure 9: Button incorrectly connected to an Intel® Galileo

Why is it wrong? If the button is pressed, everything works fine. The value of pin would be HIGH and you can say "Yes, the button is definitely pressed". But what happens when the button is off? It is important to know that a logic level can be: LOW (or 0), HIGH (or 1) and also UNKNOWN (or high impedance). When the button is not pressed you cannot say for sure what logical level the pin has: it could be 0 as well as 1 because the wire is not connected neither to Ground nor to a power supply.

Let's give it another try (figure 10).

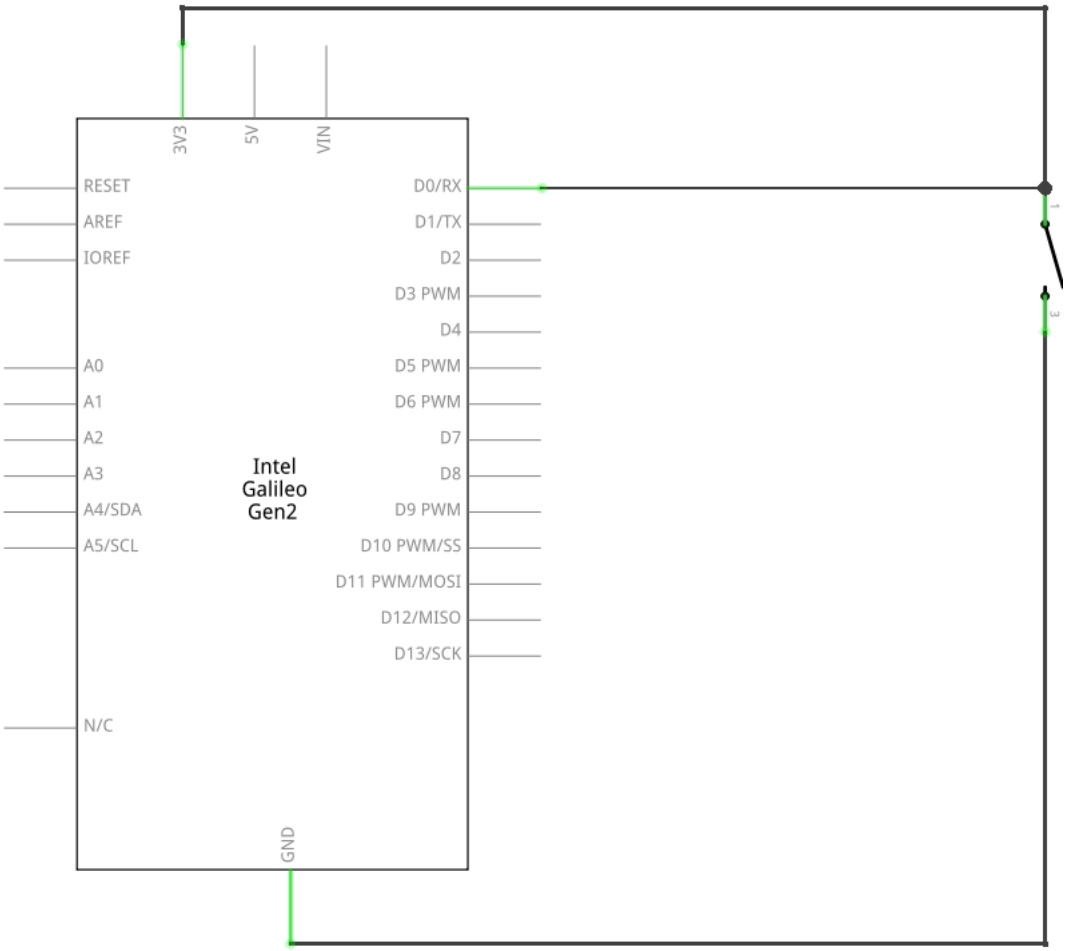

Figure 10: Button incorrectly connected to an Intel® Galileo

Figure 10 is also incorrect. When the switch is off the button's pin value is HIGH. The big problem appears when the button is pressed. It will create a short circuit: the ground is directly connected to VCC which is very bad because you do not have any resistor and the electric current is not limited.

The correct way to connect a button to a board is presented in figure 11.

This time you will not have a short circuit because if the button is pressed there is the R resistor. The R resistor is called a *pull up resistor* and the

whole system is called a *voltage divider*. If the button is pressed our pin's value will be LOW.

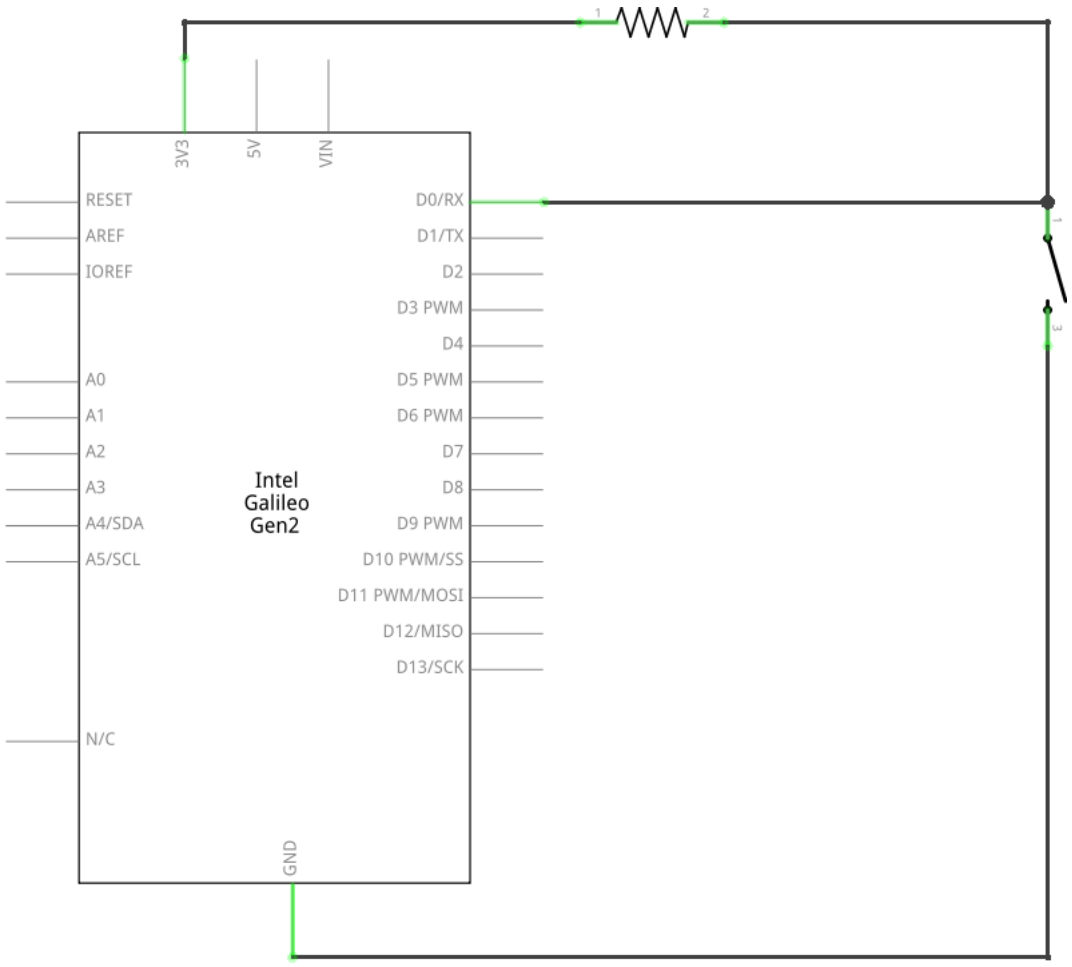

Figure 11: Button correctly connected to an Intel® Galileo

You can also connect the resistor to the Ground. Now you have a pull-down resistor and the pin's value will be HIGH when the button is pressed and low otherwise.

Safety Instructions

Whenever you connect anything to your Intel® Galileo board, you must

ensure that the board is not powered up. Otherwise you might accidentally create a short-circuit and burn it.

Only after you ensured that everything is correctly connected, you may safely power up the board.

Introduction to Linux

The Intel® Galileo board runs Yocto as the operating system. Yocto is not only an operating system (OS), it is also an environment that allows you to build your own custom operating system.

As any other OS, Yocto allows you to control the board via a Shell by using standard Linux commands. Although this sounds intimidating, especially nowadays when the Graphical User Interface (GUI) makes everything intuitive and extremely easy, sometimes it is the only viable option.

We will introduce some of the Shell's characteristics together with a few basic commands and when you might need to use them.

The Shell

The Shell is a window that allows you to interact with the board. It waits for you to enter a command and then executes it. Once you open a Shell, you will see the prompt. This means that everything works fine and the Shell is waiting for input.

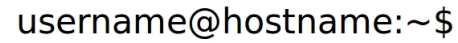

Figure 12: Shell prompt

The prompt also offers you some information. First of all, it shows you the user currently logged in. The user's name is what you see before the @

character. It also shows the hostname of your board after that.

Perhaps the most important information the prompt displays is the working directory. That is the directory you are currently working and entering commands in. It is displayed right after the colon in the prompt. You will notice that the default working directory is `~`. That is the user's home directory and its equivalent is */home/username*.

Paths

In order to access a certain file or directory, you have to take into account the path to it. There are two different paths you can use: absolute and relative.

In Linux, the directories' structure is like a tree. The root directory is `/` and it contains all the other directories and files.

If you use an absolute path to a file or a directory, that means that you build the path to it starting with the root directory. Thus, you can say that any path that starts with `/` is an absolute path.

On the other hand, you can use a relative path, which means that you build it starting from the directory you are working in, our working directory. Thus, all the files and directories are relative to it.

When building paths, there are three symbols you should be familiar with:

- `.` is your current directory
- `..` is its parent directory
- `~` is the home directory

pwd

The *pwd* command makes the Shell print the working directory. It is impor-

tant to know which directory you are working in and sometimes it is difficult to get it from the prompt. So, anytime you feel lost, use *pwd*.

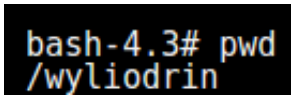

Figure 13: Example of pwd output

ls

ls makes the Shell print all the files and directories located in the working directory. If you want to see the contents of some other directory, you can pass that directory as an argument to the command. For instance, if you want to print all the files and directories in /, you will write: *ls /* .

cd

You already know that once you open a Shell, the working directory is your home directory. However, you will need to work in other directories too. In order to change the working directory, you will have to use *cd* followed by the directory you want to go to. If you don't know which directories are in the current one, take a look at the ls command above.

For example, if our home directory contains a directory called *homework* and you want to have that as the working directory, you use *cd homework*. Please notice that you used an absolute path. Some other alternatives would be *cd /home/wyliodrin/homework* or *cd ~/homework*. In the last two examples you used an absolute path to refer to *homework* directory.

cat

cat asks the Shell to print the contents of a file. However, it must be clear that you can only see its contents, you cannot modify them. For that you need an editor.

Just like with the *cd* command, *cat* gets as an argument the file it should display.

Example: *cat /etc/passwd*

top

By using the *top* command you can investigate all the processes that run on your board in real-time. Once you entered the command you will notice that the prompt does not appear, that is because you cannot enter another command until you are finished with displaying the processes. So, if you want to go back to what you were doing, just hit the *q* key.

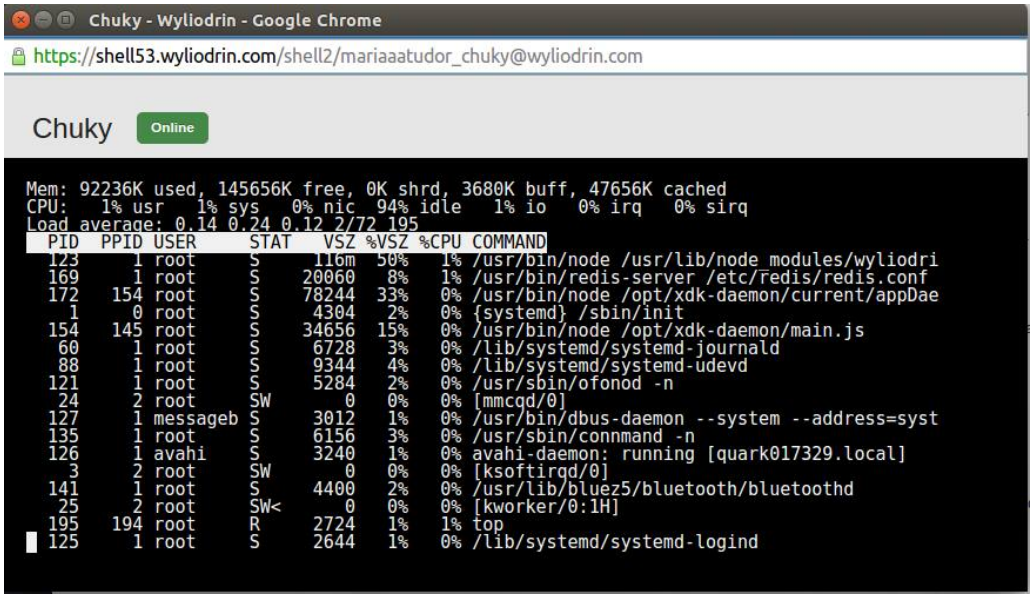

Figure 14: Top command output

For each process displayed, you can see its PID (Process ID), the user who launched the process, how much CPU and memory it is using, which command started the process and other information. What you are most interested in is the PID. That is because each process can be identified by its PID and if you want to interact with it, you have to know its process ID.

kill

You know that you can use *top* to find a process' ID in order to be able to interact with it. *kill* is the command that allows us to interact with another process.

Two processes can interact with one another by using signals. A signal is a number sent by a process to another. Both processes know that each number represents an action. You can refer to a signal either by the number or by its name.

```
ioana@ioana:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Figure 15: List of signals

The format of the *kill* command is the following: *kill -signal pid*, where signal is the number representing the action you want the process to do and pid is the process ID.

The two signals you are most interested in are *SIGTERM* (number 15) and *SIGKILL* (number 9).

SIGTERM tells the process to stop its execution. Normally, the process should save all its data and stop running. However, this signal can be ignored by the process. There are times when you cannot kill a process by using SIGTERM.

On the other hand, SIGKILL, kills the process no matter what. The downside is that the process does not have the opportunity to save its data, so killing it this way can result in loss of data. Neverthless, if something happened and

your process must be forced to stop, you have to use SIGKILL.

In case the running process has a Shell attached and you can access it, you can simply use a key combination to send the SIGTERM signal to it and make it stop, *Ctrl+C* .

killall

killall has the same effect as *kill*, except that you do not have to know the PID of the process, but its name. Instead of passing the process ID as an argument, you have to pass the process name.

What is Wyliodrin?

Here is a question for you: when you were a kid, did you ever want to be able to build robots and all sorts of cool devices? We're sure you did. The two problems you were most likely faced with were how to build the electronics and how to write the software.

Well, now you can! Intel® Galileo provides the hardware you need, and Linux provides the software. You can choose any programming language you want, as long as it has libraries designed to interact with the hardware. Then it all comes down to programming.

Wyliodrin is a platform that allows programming the Intel® Galileo board remotely, from a browser. It supports two visual languages as well as C/C++, Python and Javascript.

Using Wyliodrin Makes It Easier

All you need is a computer, a browser and an Internet connection. Sign up to Wyliodrin and start programming your boards. You can even program and deploy applications using a public computer, as you do not need to install anything on it.

You Do Not Have to Know Any Programming Languages

With Wyliodrin you can program your boards using our

visual & streams programming system. Drag and drop blocks and Wyliodrin will write the code for you.

Embedded Devices Are Not Connected to Your Computer

Usually, when programming embedded devices, you have to either connect them to your computer or dedicate a screen, a keyboard and a mouse to them. With Wyliodrin, your device needs to be connected only to the Internet using a wire or WiFi. This way you can build cars or robots that can move around while still being online.

You Can Choose Your Favorite Programming Language

Programming embedded devices usually meant learning a programming language such as C or C++. With Wyliodrin you can choose any programming language you like from C/C++, Java, Pascal, Shell Script, Perl, PHP, Objective-C, C#, Python, Javascript.

Visualise Sensor Data and Debug Your Programs Using Graphs

Choose from a large graphs library then drag and drop elements on your dashboard to monitor sensors' values. Everything is done using Wyliodrin's platform independent library to display the signals in the dashboard located in your browser.

Embbeded Device Inter-Communication Is Possible and Easy

With Wyliodrin, mobile phone-embedded device communication is now pos-

sible too. This feature allows you to control your embedded device by using apps created especially for your mobile phone.

You May Access Your Board Using a Shell

If you need more advanced tasks, you can access your device's Shell from the browser - no more looking around for the correct IP addresses or setting up port forwarding!

Your Device Can Be Anywhere in the World

Since the embedded device is connected to the Internet, it can be anywhere. You can program it by simply logging into your Wyliodrin account. If you have a weather station or a remote automation device, you can program, control and monitor them from your home or office.

Wyliodrin Setup

In order to setup your Wyliodrin account and connect your board, please go to http://www.wyliodrin.com.

Sign in with one of your Facebook, Google or GitHub accounts. After signing in, right at the top of the projects page shown in Figure 16 you will find the *Add new board* button. Please press the button and continue below to add the embedded board to your account.

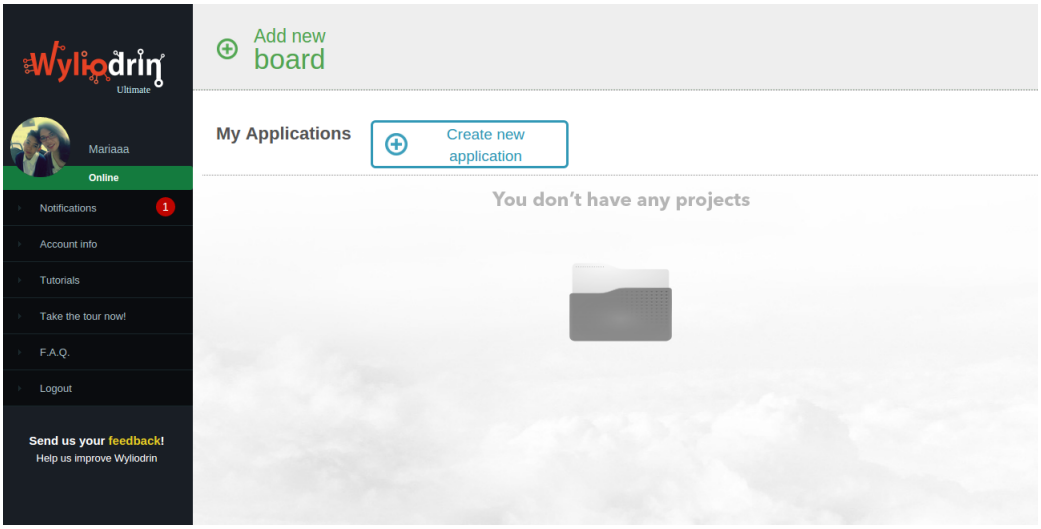

Figure 16: Projects Page

Afterwards, depending on the board you are connecting, you have to follow some different steps.

Intel® Galileo

To get your Intel® Galileo online follow these steps:

1. Name the board and select its type
2. Setup network preferences
3. Write the SD Card
4. Copy configuration file
5. Run updates

Step 1. Name the board and select its type

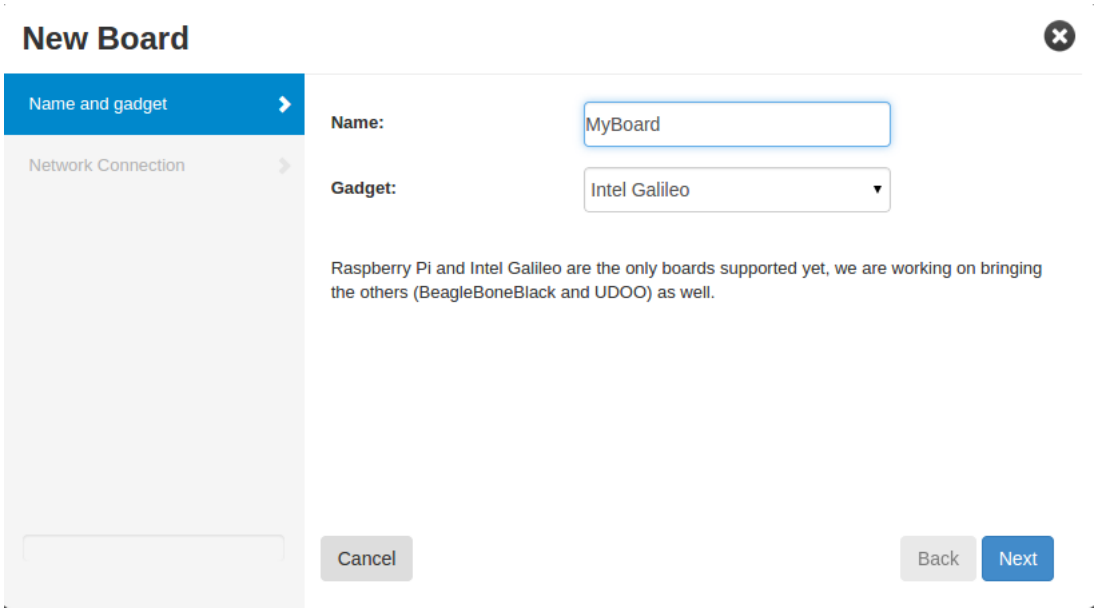

Figure 17: Name and gadget

Please give your Intel® Galileo board a name, select *Intel® Galileo* as the gadget type and press *Next*.

Step 2. Setup network preferences

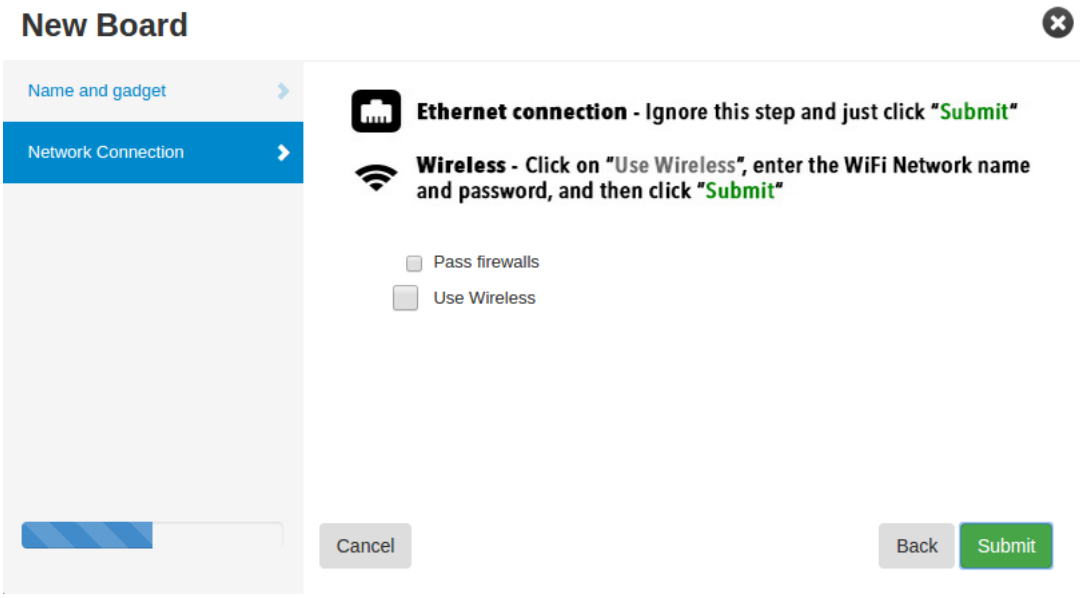

Figure 18: Network Connection

In the network connection setup menu make sure both options (Pass Firewalls and Use Wireless) are *NOT* checked unless needed. You will need to connect a network cable to the board if you are not using a WiFi adapter. Press *Submit*. A tutorial on how to connect the board will appear.

Step 3. Write the SD Card

The board needs to boot an operating system from an SD Card. You will need a class 10 micro SD card with 4 GB or more. If you have a lower than 10 class card, it will still work, but slower.

First you need to download and unzip the Intel® IoT Developer Kit [1] image and write it on an micro SD Card.

In order to write this image, please select and unzip the Intel® IoT Development Kit archive that you have just downloaded and rename it to iot-devkit-

[1] `https://software.Intel.com/sites/landingpage/iotdk/board-boot-image.html`

latest-mmcblkp0.direct.img.

Depending on your system, the following steps may vary.

Windows

Insert the SD card into your SD card reader and check the drive letter that was assigned to it. You can easily see the drive letter (for example E:) by looking in the left column of Windows Explorer. You can use the SD Card slot if you have one or a microSD Adapter in a USB slot.

Download the Win32DiskImager utility [2]. (it is also a zip file).

Extract the executable from the zip file and install the Win32DiskImager utility; you may need to run the utility as Administrator. Right-click on the file, and select *Run as Administrator*

In Win32DiskImager, select the drive letter of the SD card in the device box. *Be careful to select the correct drive; if you get the wrong one you can destroy your data on the computer's hard disk!* If you are using an SD Card slot in your computer and you cannot see the drive in the Win32DiskImager window, try using a microSD USB Adapter in a USB slot.

Click *Write* and wait for the write to complete.

Exit the Win32DiskImager and eject the SD card.

Linux

Insert the SD card into your SD card reader. The card will appear in */dev* usually under the name of mmcblk0.

Open a terminal and enter the following command:

[2] `http://sourceforge.net/projects/win32diskimager`

```
dd if=iot-devkit-latest-mmcblkp0.direct.img\
   of=/dev/device_name
```

Please replace the path */dev/device_name* with the path to the SD Card, usually it will be */dev/mmcblk0*.

Wait until the process has finished.

Eject the SD Card.

Mac OS X

Insert the microSD Card into the Card reader or use a microSD USB Adapter for your computer.

Download the PiWriter[3] utility. This will be used for writing the Intel® IoT Development Kit SD Card Image.

Run PiWriter. You will be prompted for an administrator user and password. You will need to have administrative rights to use PiWriter. If unsure what to do, just type in your password.

Follow the instructions on screen.

When prompted to select a file, select the previously unarchived Intel® IoT Development Kit Card Image and proceed to write it to the microSD card.

Step 4. Copy configuration file

Once you have written the Intel® IoT Development Kit image to the card, you will need to download and copy to the card the Wyliodrin configuration file. This file is *wyliodrin.json* spelled exactly like that. You can download it directly from the setup tutorial page - simply follow the link displayed in

[3] `https://github.com/Wyliodrin/PiWriter`

the previous step.

If you have closed that, just go to the Wyliodrin Projects page, click the icon on the right next to the boards name and click *Download wyliodrin.json* and make a note of where you save it.

In case you want to change the account that the board is associated to, you will only have to update this single file.

Now proceed to copying the wyliodrin.json file directly on the card. Once again make sure the file's name is spelled exactly *wyliodrin.json* with no other characters. On your SD card you should now have exactly the files and folders in Figure 19.

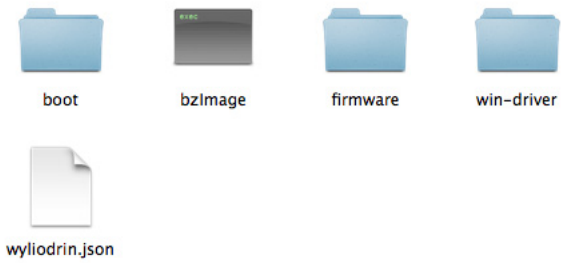

Figure 19: Files in SD Card

Please be aware that each board has a different configuration file. Even if they are all called wyliodrin.json the content is different for each board that you want to activate.

After the setup, please eject the card from you computer and insert it into the Intel® Galileo, connect the network cable to the board and power it on. Within a short time the board will appear as online on the Wyliodrin Projects page.

You have successfully connected your Intel® Galileo to your Wyliodrin account. In the Wyliodrin Projects page you should see your board online just like in Figure 20.

Figure 20: Galileo Online

Step 5. Run updates

When the board appears online, please click the small button next to it and run *Update Image* and *Extra Libraries* (see Figure 21). This will update the operating system on the board to its newest version and install extra libraries for more advanced usage, such as web servers and multimedia projects.

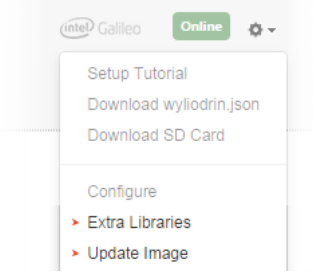

Figure 21: Update board

A full tutorial on how to setup your board is available online [4].

Intel® Edison

To get your Intel® Edison online follow these steps:

1. Flash the Edison
2. Download the board configuration file and write it to the Edison
3. Download the install script file and write it to the Edison
4. Copy configuration file
5. Run updates

[4] `https://wyliodrin.com/wiki/boards_setup/arduinogalileo`

Flash the Edison

To flash the Edison board you just need to follow *Step 1* and *Step 2* in the instructions on Intel's website. You will have to choose the Operating System that runs on the computer you are going to connect the Edison to.

Board activation

Wire up the Intel® Edison to your computer in the same way as in the previous step. Make sure you use the USB adapter closer to the power. You should see a disk called Edison.

Copy the file *wyliodrin.json* on it. Make sure the file is named exactly *wyliodrin.json*. Copy the install script[5] on it. Make sure it is named exactly *install_edison.sh* (figure 22).

Figure 22: Files on the Edison

Connect the USB to the Edison so that you see the console. Run

```
configure_edison --setup
```

This will configure the name of the board and WiFi. Please make sure that your WiFi is sending IP addresses in a network different than 192.168.2.0/24. To install Wyliodrin, run

```
mkdir /media/storage
mount -o loop,ro,offset=8192 /dev/mmcblk0p9 /media/storage
```

[5] https://www.wyliodrin.com/public/scripts/install_edison.sh

```
cd /media/storage
sh install_edison.sh
```

In a few moments you should see the board online.

A full tutorial on how to setup your board is available online [6].

You can watch a tutorial on how to add a new board online [7].

Connect an already flashed board

If your Intel® Edison is already flashed, all you need to do is to copy the wyliodrin.json file onto the board.

Wire up the Intel® Edison to your computer in the same way as described in *Flash the Edison*. Make sure you use the USB adapter closer to the power. You should see a disk called Edison.

Copy the file *wyliodrin.json* on it. Make sure the file is named exactly *wyliodrin.json*.

Unplug all the cables form the board and plug them back.

Your board should get online.

[6] https://projects.wyliodrin.com/wiki/boards_setup/inteledison

[7] https://www.youtube.com/watch?v=HHzWRNAYY4O&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=1

Switch an LED on and off

In this project you will learn how to turn an LED (a Light-Emitting Diode) on and off.

What You Need

1. Your Intel® Galileo board
2. One LED
3. One Resistor rated 150 Ω or more
4. One breadboard
5. Two jumper wires

Building The System

First of all, you will setup the wiring. It is important to first make sure that your board is powered off at this stage. Then, connect everything according to the schematic in Figure 23.

The LED is connected according to the explanations in the *Introduction to Electronics* chapter. There it was stated that in order to correctly connect an LED you also need a resistor that will reduce the current flow.

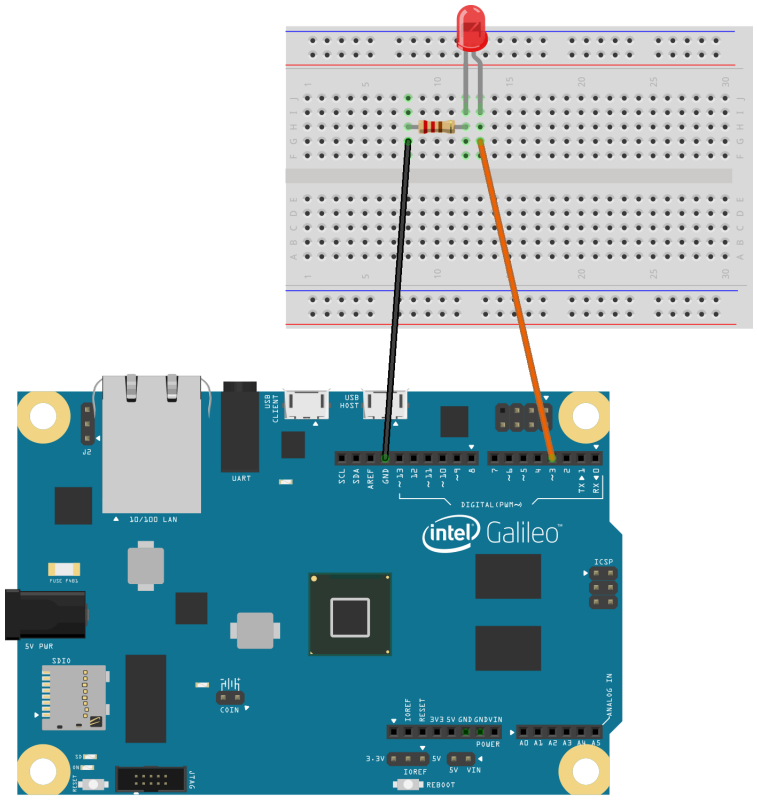

Figure 23: Blinking LED wiring

It does not matter if you use different colored wires or different pin columns on the breadboard as long as the components and the wires are connected in the same manner as in the picture.

Also, please make sure that the LEDs you will be using in this tutorial and in the following are connected the right way, with the longer leg (+) connected to the Digital Pins (nr. 3 in this case) while the shorter one (-) must go to the ground pin (GND).

When you are sure that everything is connected correctly, power up your board.

The Code

Now that you wired everything, let's make it work. Please go to your Wyliodrin account. On the Projects page you will see a *Create new application* button. Press the button. You should see the *New Project* form in Figure 24. You will be asked to name your application and select the programming language. Insert a name and choose *Visual Programming*.

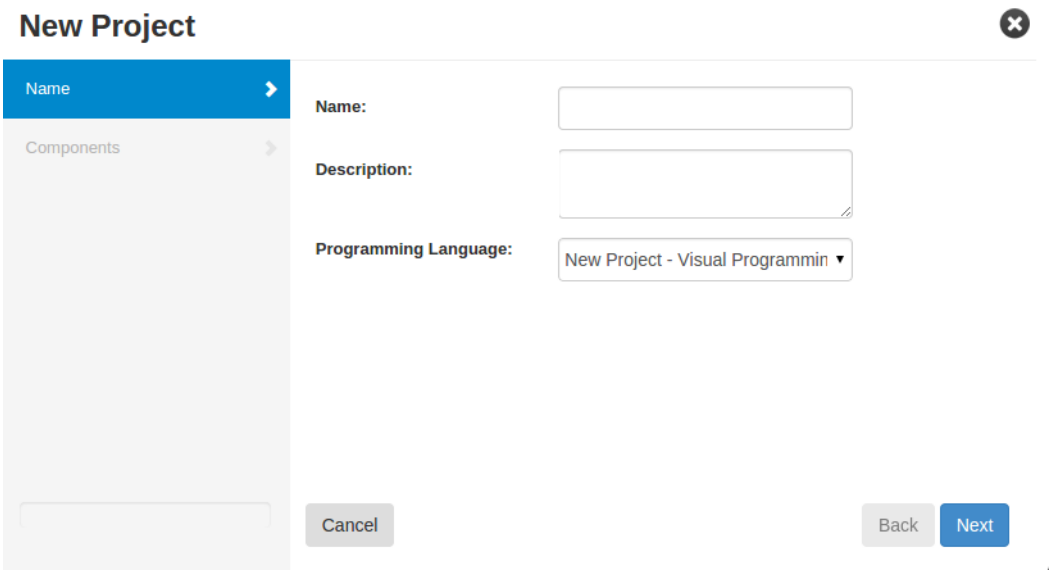

Figure 24: New Project form

Once created, click on the new application's name to open it.

Now let's make the LED blink. Your program must turn the LED on, wait for a small amount of time, turn the LED off, wait, turn the LED back on and so on. The Visual Programming blocks required for the blinking LED are presented in Figure 25.

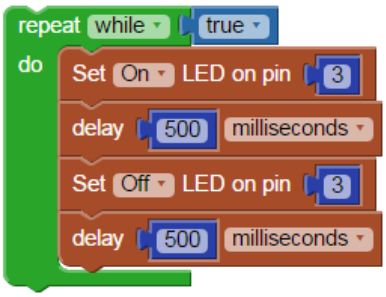

Figure 25: Blinking LED in Visual Programming

The *Set On LED on pin* block will turn on the LED. The *Set Off LED on pin* block will turn it off (the LED is connected to pin 3 on the board).

The *delay* block will stop the program for a period of seconds, milliseconds or microseconds.

You must use the *repeat while []* block as you want your program to repeat the instructions inside the block indefinitely.

If you have programming knowledge you may also want to see the code generated by Wyliodrin. On the right you will see a *Show Code* button. The code is generated in Python and Javascript. Below is the Python code:

Python

```
1 from wyliodrin import *
2 from time import *
3 pinMode (3, 1)
4 while True:
5   digitalWrite (3, 1)
6   sleep ((500)/1000.0)
7   digitalWrite (3, 0)
8   sleep ((500)/1000.0)
```

The first two lines of code import the libraries used in the program. The * means ”import all the functions from this library”. Actually, from the

wyliodrin library only the *pinMode* and *digitalWrite* functions are used, and from the *time* library only the *sleep* function is actually used in this particular example.

The *pinMode(pin, mode)* function allows us to set a pin as either Input or Output. The first parameter is the pin number and the second parameter is its mode (0 for Input and 1 for Ouput).

On line 6 is a *while True* loop corresponding to the outer green block in the Visual Programming block.

The *digitalWrite(pin, value)* function allows us to turn an LED on or off. The first parameter is the pin number the LED is connected to and the second parameter is the value (0 for off and 1 for on). Therefore, line 7 turns the LED on and line 9 turns it off.

Finally, the *sleep(seconds)* function corresponds to the *Sleep* block in the Timing tab. There are 1000 milliseconds in 1 second, therefore, 500 milliseconds are equivalent to 0.5 seconds (half a second).

Below is the Javascript code.

Javascript

```
1 var wyliodrin = require("wyliodrin");
2 wyliodrin.pinMode (3, 1);
3 while (true) {
4   wyliodrin.digitalWrite (3, 1);
5   wyliodrin.delay (500);
6   wyliodrin.digitalWrite (3, 0);
7   wyliodrin.delay (500);
8 }
```

The *pinMode* and *digitalWrite* have the same functionality as explained above. The *sleep(seconds)* is now replaced by *delay(milliseconds)*.

To run the application, click the name of your board located on the left.

You can watch an online tutorial on how to create this project [8].

[8] https://www.youtube.com/watch?v=9UZjaqEEaXQ&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=2

Pulsating LED

In this project you will learn how to make a LED fade.

What You Need

- Your Intel® Galileo board
- One LED
- One 220 Ω Resistor
- One breadboard
- Two jumper wires

Building The System

How can you make a LED fade? Look at your board. Some of the digital pins have a tilde (~). To obtain the fade effect you must use those pins. But let's talk a little about those pins with tilde (Pulse Width Modulation). Digital control is used to create a signal that switches between 0 and 1. Practically you have a clock that measures a quanta of time. The signal has the value 1 during a part of that quanta of time while for the rest of time it is equal to 0 (Figure 26). The period while the signal has the value 1 is called the pulse width. To simulate analog values, you have to change the pulse width. The more the signal equals 1 in a certain time interval, the higher the simulated

analog value gets. You can say that it makes the average of the current in a time interval and that is why the LED's intensity changes. What's important to know is that the LED changes the light intensity so fast that the human eye can't perceive the change so you and I see it as a continuous light. In order to control those pins you must use the *analogWrite* function. The values you write on the pin must range from 0 to 255, 0 being the equivalent of digital 0 and 255 the equivalent of digital 1. The percentage of the time it takes to skip from 0 to 1 is the value divided by 255 so we must create a mapping for those values.

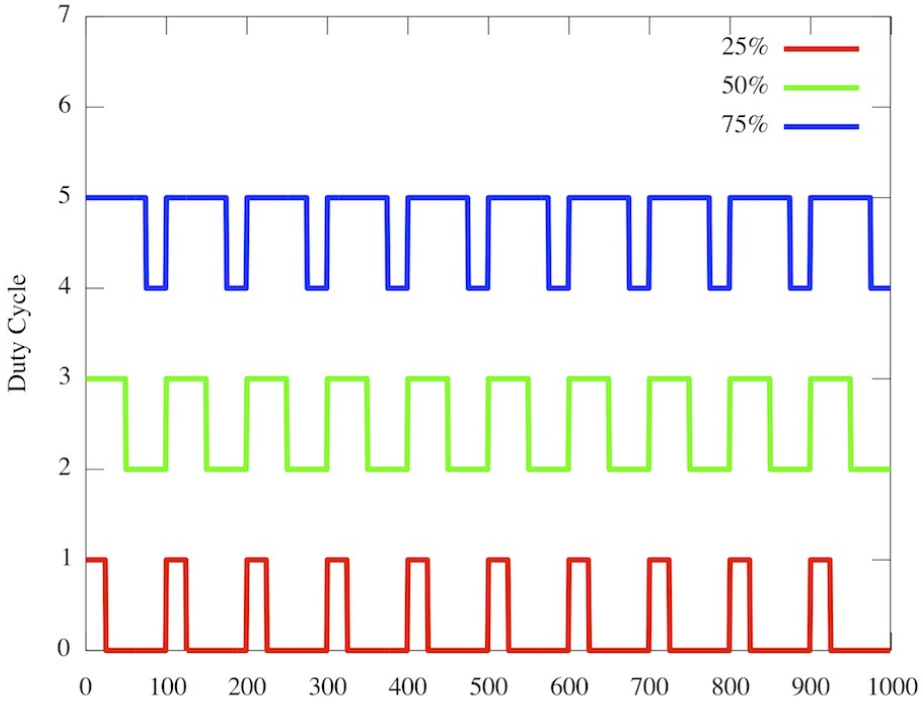

Figure 26: PWM Plot[9]

Now that you cleared this thing let's make the wiring (see Figure 27).

[9] `http://coactionos.com/images/pwm\discretionary{-}{}{}plot.jpg`

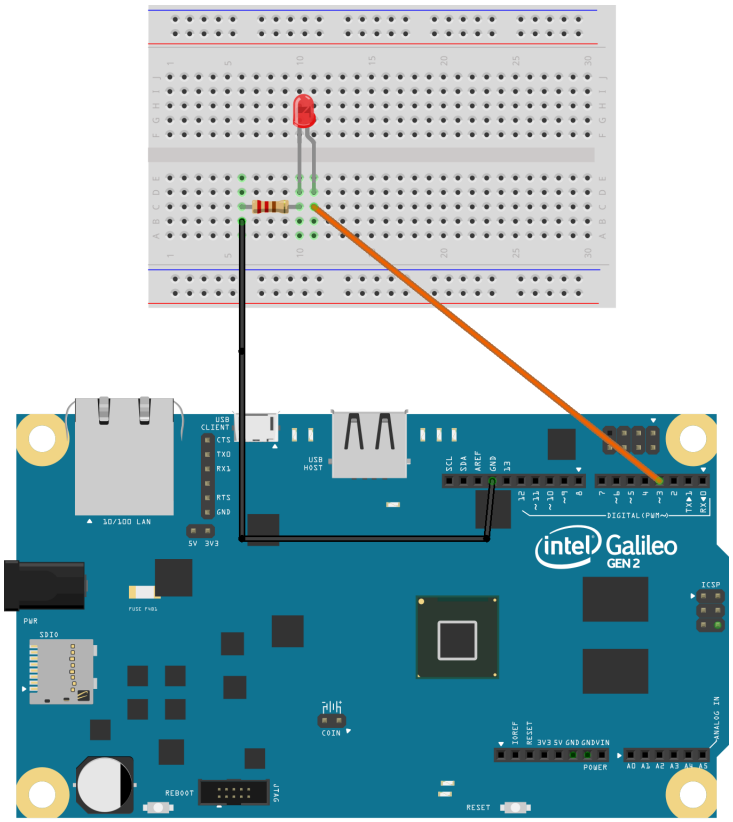

Figure 27: Pulsating LED wiring

The Code

You will have the LED turned off, wait a little bit, turn the LED on at almost half its maximum intensity, wait a little bit again and then proceed to turning the LED on at maximum intensity. This process will be repeating continuously. Figure 28 displays the Visual Programming code that does that.

The *analogWrite pin [] value []* block is used to turn on the LED on the specified pin at a certain intensity. If the *value* equal 0, then the LED will be light off, while for the *value* 255 the LED lights at the maximum intensity.

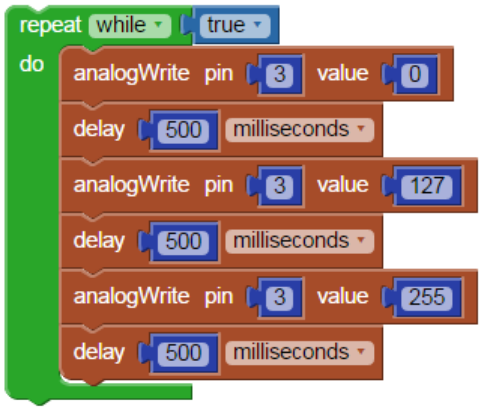

Figure 28: Pulsating LED in Visual Programming

You can see the Python and JavaScript code here:

Python

```python
from wyliodrin import *
from time import *
pinMode (3, 1)
while True:
  analogWrite (3, 0)
  sleep (0.5)
  analogWrite (3, 127)
  sleep (0.5)
  analogWrite (3, 255)
  sleep (0.5)
```

Javascript

```javascript
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (3, 1);
while (true) {
  wyliodrin.analogWrite (3, 0);
  wyliodrin.delay (500);
  wyliodrin.analogWrite (3, 127);
  wyliodrin.delay (500);
```

```
8   wyliodrin.analogWrite (3, 255);
9   wyliodrin.delay (500);
10 }
```

For those of you that have some electronics background, instead of writing values on the pin, you can obtain the fading effect by using a rotary angle sensor.

What You Need

1. One potentiometer (rotary angle sensor) RM065
2. One 220 Resistor
3. One LED
4. Six jumper wires

In Figure 29 you can see what a potentiometer looks like. The potentiometer provides a variable resistance, which can be read as an analog value. In this example, that value controls the LED intensity.

Figure 29: Potentiometer [10]

Figure 30 presents the wiring for this project. The first from the three pins of the potentiometer goes to ground (GND), the second goes to VCC (5V) and the third goes to an analog input.

[10] http://www.farnell.com/datasheets/1734496.pdf

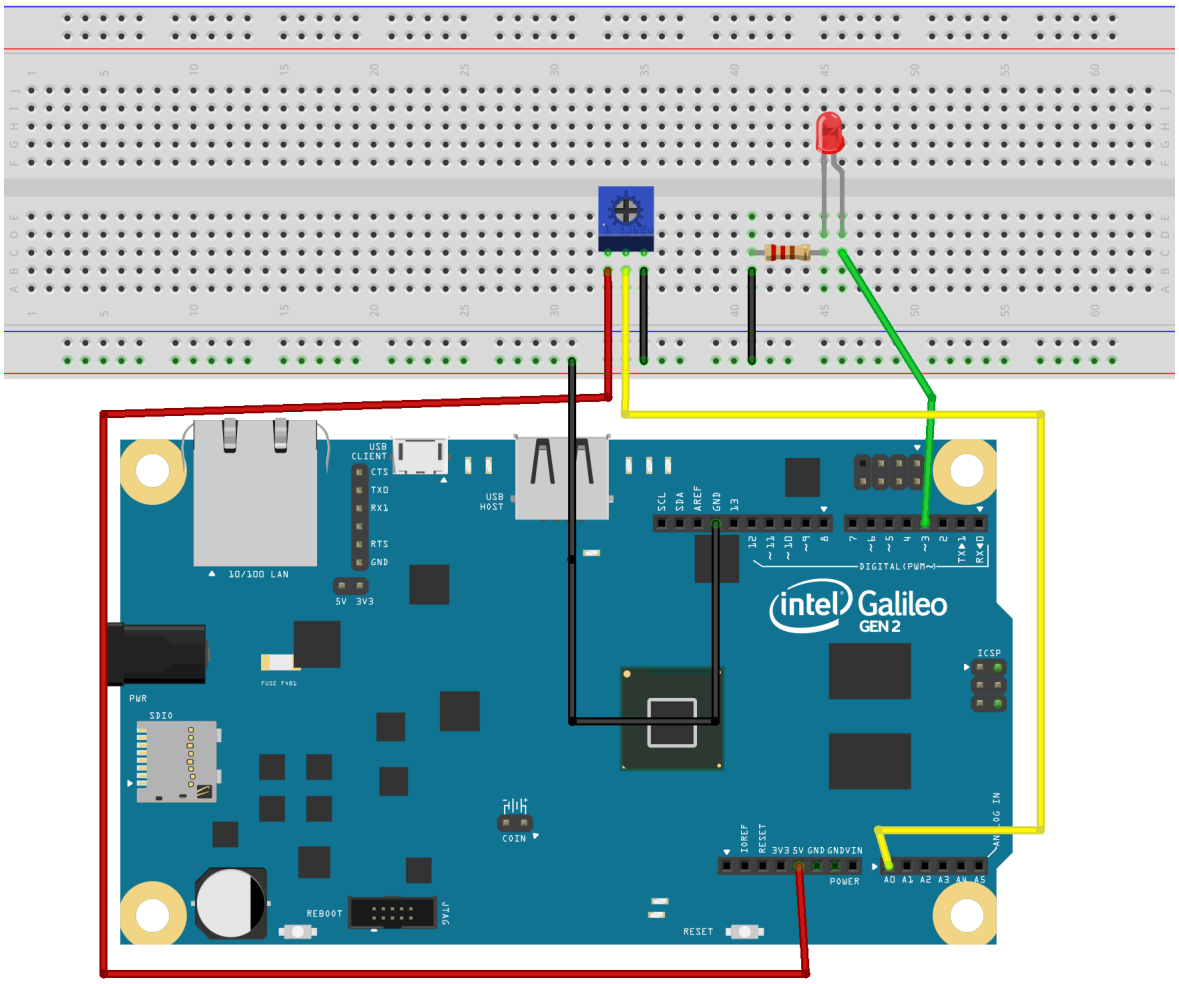

Figure 30: Pulsating LED with potentiometer wiring

The potentiometer spits values ranging from 0 to 1023. The LED takes values ranging from 0 to 255. What you need to do is to map the 0 - 1023 output range of the potentiometer to the 0 - 255 input range of the LED. Figure 31 depicts the Visual Programming code that does that.

If you want to see the variation better you can plot a graph. Go to the *Signals* tab and chose the *Send signal* block. Fill in the name of the signal as *gauge*. This block will send a signal to the graph that consists of a name and a value. That is why the signal you send and the name of the signal received by the graph must be the same.

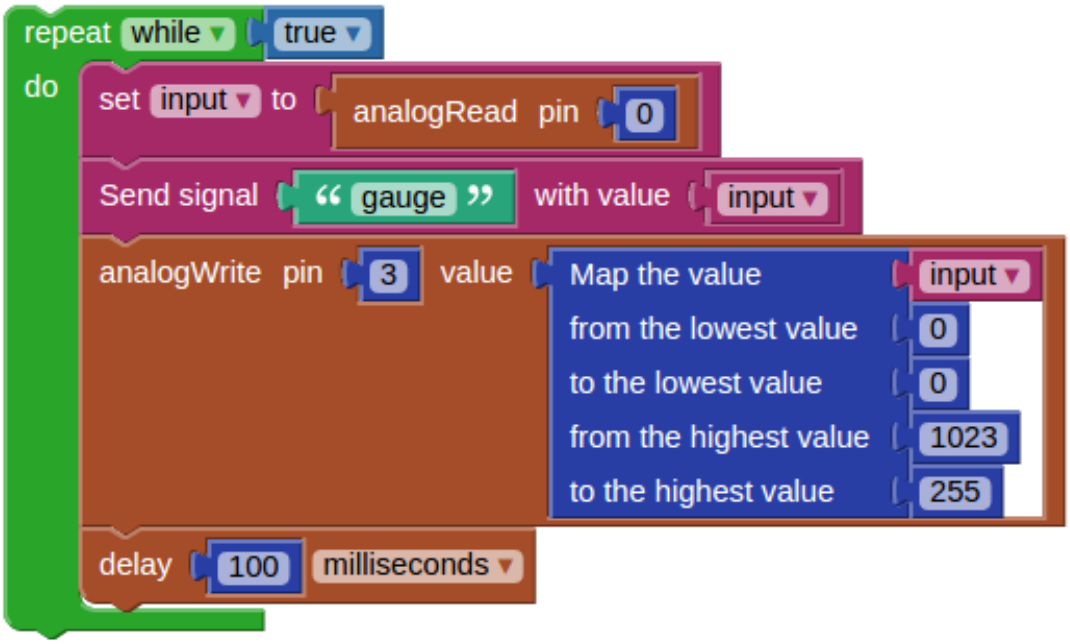

Figure 31: Pulsating LED with potentiometer in Visual Programming

The graph receives each signal that has the corresponding name and it updates the displayed value as it receives a new one.

After that, click the *Dashboard* button. You will find it in the top right corner of the page.(Figure 32)

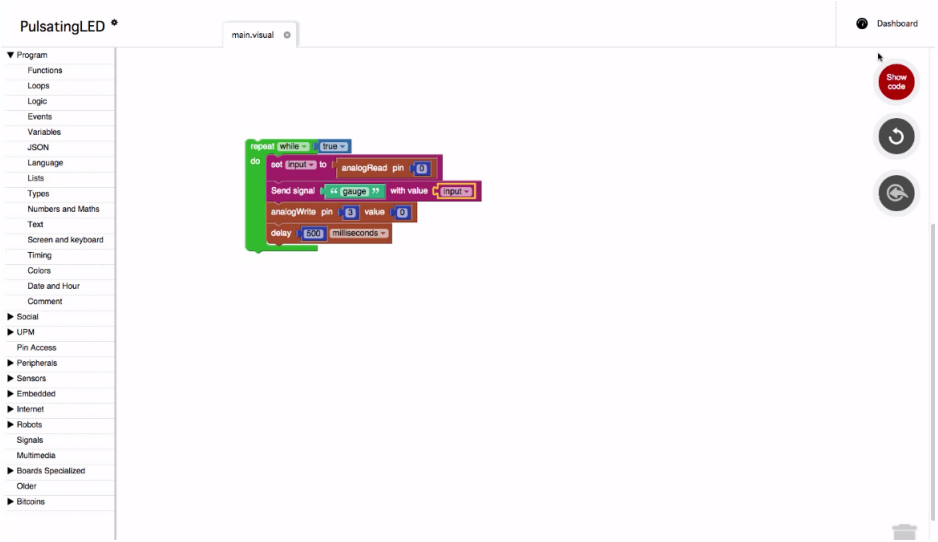

Figure 32: Dashboard

Choose the Solid Gauge(Figure 33) from the list (Figure 34).

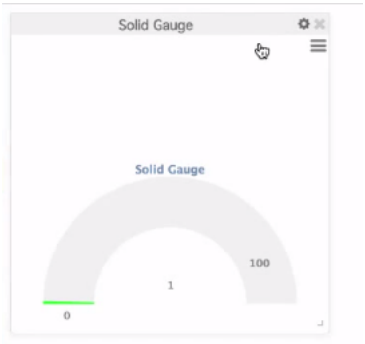

Figure 33: Solid Gauge

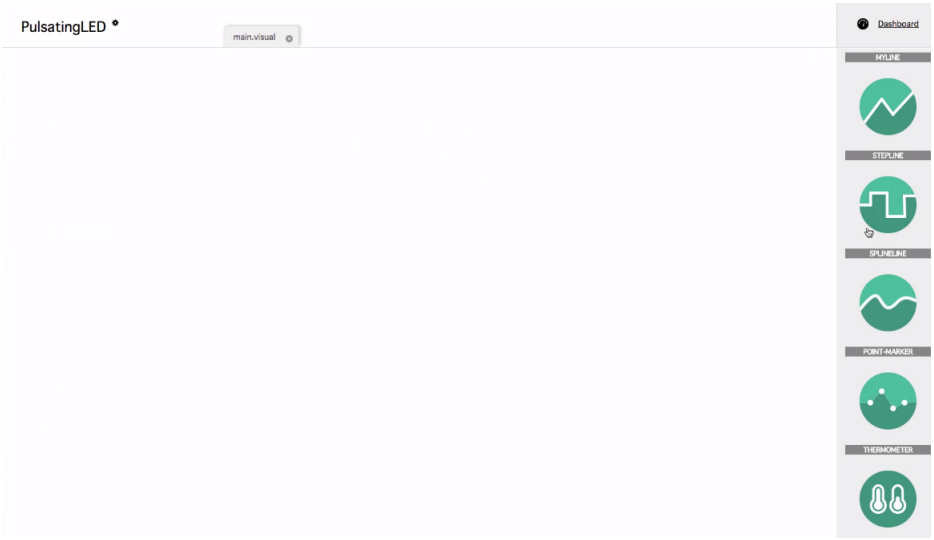

Figure 34: Plot types

Next to the graph you will see a settings button (Figure 35). Click on it. Name the signal *gauge*, just like you did in the block that sends that signal. You can also choose the graph's color. Its lowest value will be 0, 400 for the mid value, 800 for the highest and the maximum value will be 1023.

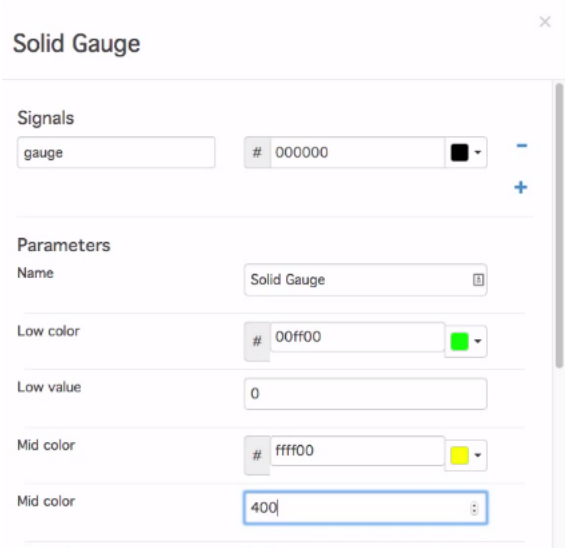

Figure 35: Signal Settings

Below you can see the generated code.

Python

```python
from wyliodrin import *
from time import *
input2 = None
pinMode (0, 0)
pinMode (3, 1)
while True:
  input2 = analogRead (0)
  sendSignal('gauge', input2)
  analogWrite (3, map(input2, 0, 1023, 0, 255))
  sleep (100/1000.0)
```

Javascript

```javascript
var input;
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (0, 0);
wyliodrin.pinMode (3, 1);
while (true) {
  input = (wyliodrin.analogRead (0));
  wyliodrin.sendSignal('gauge', input);
  wyliodrin.analogWrite (3, wyliodrin.map(input, 0, 1023, 0, 255));
  wyliodrin.delay (100);
}
```

You can watch an online tutorial on how to create this project [11].

[11] https://www.youtube.com/watch?v=NAsPspM8MXA&index=3&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

SOS Morse Code Signaler

For this project you are going to use the exact same circuit as in the previous chapter. You will use some different code to make the LED display a Morse Code message. In this case, you are going to make the LED signal the letters S.O.S. - the international Morse code distress signal.

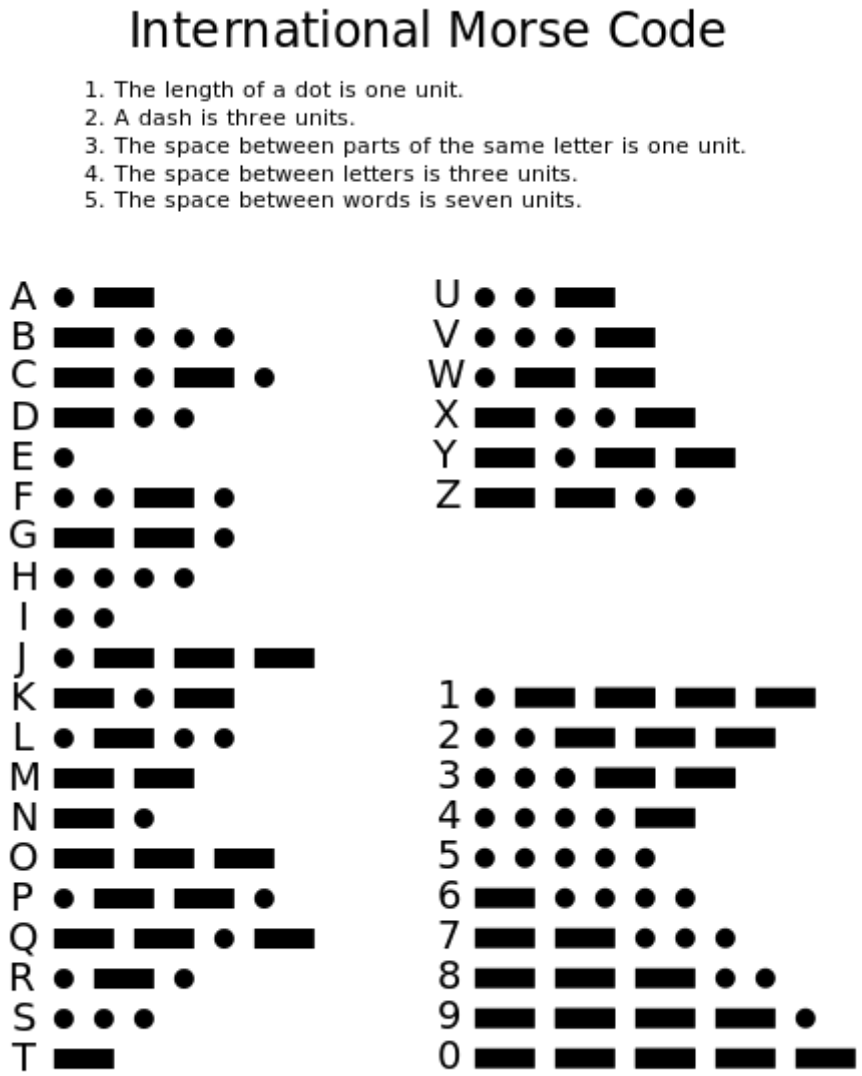

Figure 36: International Morse Code

You will use the same wiring as you used in the *Switch a LED on and off* chapter.

The Code

Please create a new visual programming project.

For this application you will continuously transmit the S.O.S. signal by using the LED from the previous project. It can be seen in Figure 36 that the S letter is signaled as 3 dots and the O letter is signaled as 3 dashes.

Figure 37 presents the code in Visual Programming.

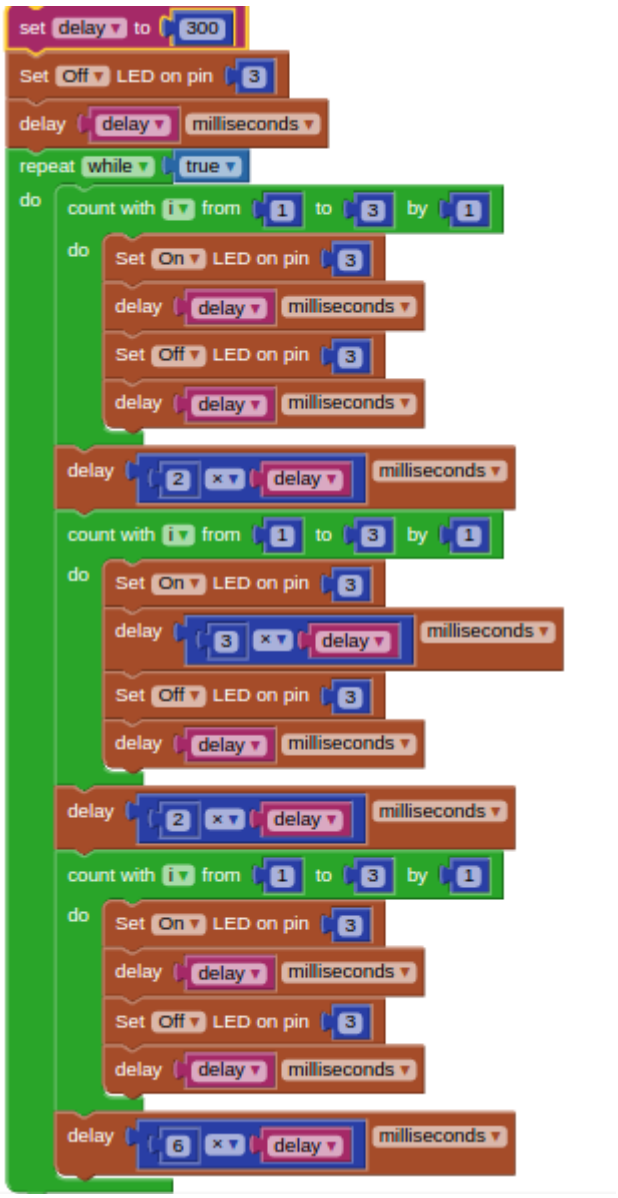

Figure 37: SOS Morse Code in Visual Programming

The first purple block sets the *delay* variable to 300. This is the unit length. This variable can be modified for faster or slower blinks. The next two brown blocks ensure that the LED is off before the first blink. The outer green *repeat* block continuously signals S.O.S. In the first and third inner green *count* block the letter S is signaled as three dots, and in the second inner green block the letter O is signaled as three dashes. Between the three inner blocks there are two units of delay which together with the delay from the previous block form the space of three units between letters. Before the loop starts again, a delay of 6 units is imposed, which together with the delay from the last S block form the space of seven units required between words as standard Morse practice.

Run the project and see what happens.

Below is the code generated in Python and Javascript.

Python

```
from wyliodrin import *
from time import *
delay = None
i = None
pinMode (3, 1)
delay = 300
digitalWrite (3, 0)
sleep ((delay)/1000.0)
while True:
  for i in range(1, 4):
    digitalWrite (3, 1)
    sleep ((delay)/1000.0)
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((2 * delay)/1000.0)
  for i in range(1, 4):
    digitalWrite (3, 1)
```

```
    sleep ((3 * delay)/1000.0)
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((2 * delay)/1000.0)
  for i in range(1, 4):
    digitalWrite (3, 1)
    sleep ((delay)/1000.0)
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((6 * delay)/1000.0)
```

Javascript

```
var delay;
var i;
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (3, 1);
delay = 300;
wyliodrin.digitalWrite (3, 0);
wyliodrin.delay (delay);
while (true) {
  for (i = 1; i <= 3; i++) {
    wyliodrin.digitalWrite (3, 1);
    wyliodrin.delay (delay);
    wyliodrin.digitalWrite (3, 0);
    wyliodrin.delay (delay);
  }
  wyliodrin.delay (2 * delay);
  for (i = 1; i <= 3; i++) {
    wyliodrin.digitalWrite (3, 1);
    wyliodrin.delay (3 * delay);
    wyliodrin.digitalWrite (3, 0);
    wyliodrin.delay (delay);
  }
```

```
22    wyliodrin.delay (2 * delay);
23    for (i = 1; i <= 3; i++) {
24      wyliodrin.digitalWrite (3, 1);
25      wyliodrin.delay (delay);
26      wyliodrin.digitalWrite (3, 0);
27      wyliodrin.delay (delay);
28    }
29    wyliodrin.delay (6 * delay);
30  }
```

Now let's jazz this project up a little bit by using a buzzer. A buzzer is an audio signaling device that, instead of emitting light as an LED, emits sound. Instead of just seeing the LED going on and off, you will now also hear the buzzer.

Figure 38 presents the wiring for connecting the buzzer. In this circuit the buzzer is connected to pin 9. You could have connected the buzzer to any Pulse Width Modulation (PWM) pin. Quick reminder: these pins are marked with a tilde (~) on the board.

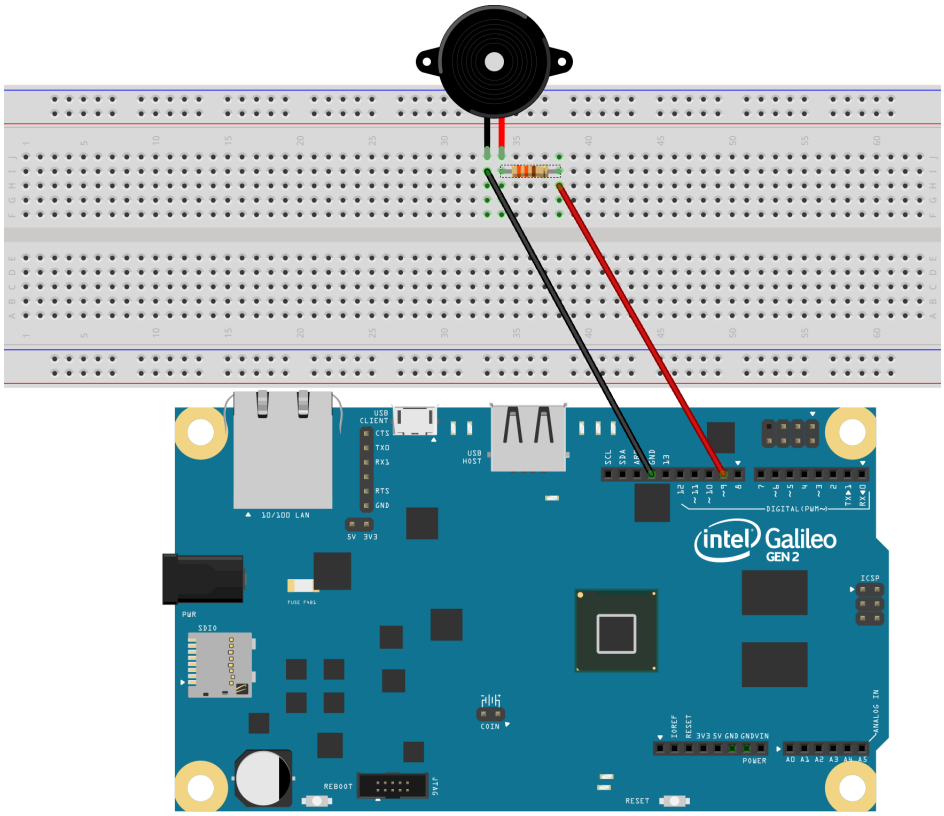

Figure 38: Buzzer wiring

In Figure 39 is presented the code adaptation for the buzzer.

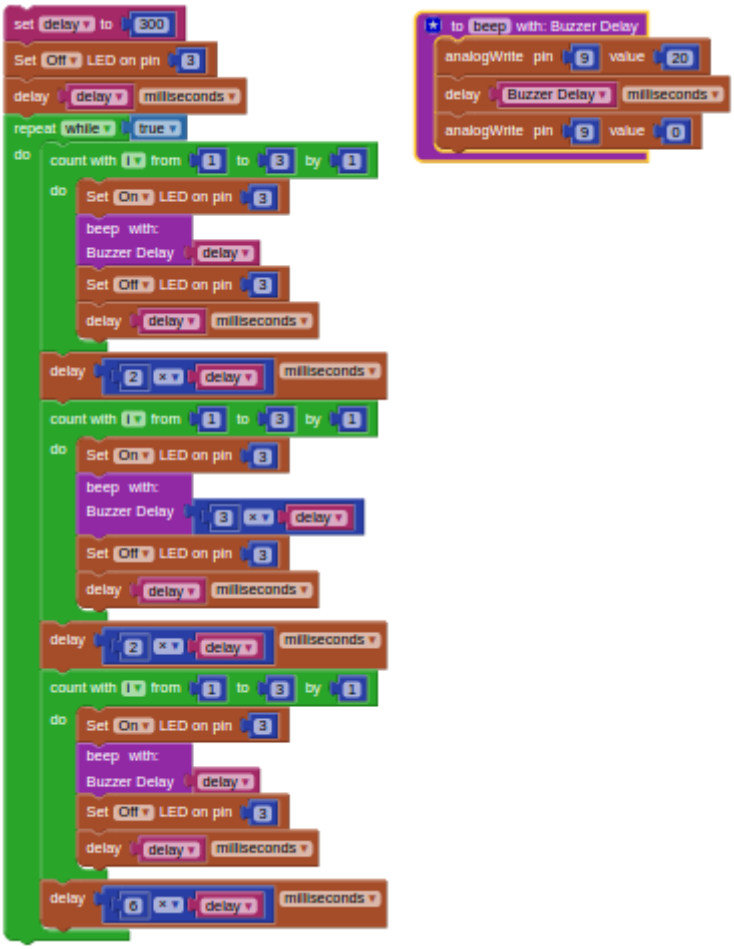

Figure 39: SOS Morse Code with buzzer

The delay between blinks has been replaced by the *beep* function. This function takes an input parameter named *Buzzer Delay* that represents the amount of time the buzzer is beeping. Another difference is that if the LED is turned on and off with the *Set* function, the buzzer is turned on and off using the *analogWrite* function. For this buzzer, 20 is the ON value, and 0 is the OFF value.

Python

```
1 from wyliodrin import *
2 from time import *
```

```
delay = None
Buzzer_Delay = None
i = None
pinMode (3, 1)
pinMode (9, 1)

def beep(Buzzer_Delay):
  analogWrite (9, 20)
  sleep ((Buzzer_Delay)/1000.0)
  analogWrite (9, 0)

delay = 300
digitalWrite (3, 0)
sleep ((delay)/1000.0)
while True:
  for i in range(1, 4):
    digitalWrite (3, 1)
    beep(delay)
    sleep ((delay)/1000.0)
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((2 * delay)/1000.0)
  for i in range(1, 4):
    digitalWrite (3, 1)
    beep(delay)
    sleep ((3 * delay)/1000.0)
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((2 * delay)/1000.0)
  for i in range(1, 4):
    digitalWrite (3, 1)
    beep(delay)
    sleep ((delay)/1000.0)
```

```
    digitalWrite (3, 0)
    sleep ((delay)/1000.0)
  sleep ((6 * delay)/1000.0)
```

Javascript

```
var delay;
var Buzzer_Delay;
var i;

var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (3, 1);
wyliodrin.pinMode (9, 1);

function beep(Buzzer_Delay) {
  wyliodrin.analogWrite (9, 20);
  wyliodrin.delay (Buzzer_Delay);
  wyliodrin.analogWrite (9, 0);
}

delay = 300;
wyliodrin.digitalWrite (3, 0);
wyliodrin.delay (delay);
while (true) {
  for (i = 1; i <= 3; i++) {
    wyliodrin.digitalWrite (3, 1);
    beep(delay);
    wyliodrin.delay (delay);
    wyliodrin.digitalWrite (3, 0);
    wyliodrin.delay (delay);
  }
  wyliodrin.delay (2 * delay);
  for (i = 1; i <= 3; i++) {
    wyliodrin.digitalWrite (3, 1);
```

```
29      beep(delay);
30      wyliodrin.delay (3 * delay);
31      wyliodrin.digitalWrite (3, 0);
32      wyliodrin.delay (delay);
33    }
34    wyliodrin.delay (2 * delay);
35    for (i = 1; i <= 3; i++) {
36      wyliodrin.digitalWrite (3, 1);
37      beep(delay);
38      wyliodrin.delay (delay);
39      wyliodrin.digitalWrite (3, 0);
40      wyliodrin.delay (delay);
41    }
42    wyliodrin.delay (6 * delay);
43  }
```

You can watch an online tutorial on how to create this project [12].

[12] https://www.youtube.com/watch?v=pfV38gTTa1g&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=4

Temperature Sensor Application

You can now make your own thermometer using only an Intel® Galileo, a thermistor (temperature sensor) and a few other readily available components.

What You Need

- Your Intel® Galileo board
- One Thermistor - TTC05103
- One 220 Ω Resistor
- One Breadboard
- Three jumper wires

Building The System

You will measure the temperature with a thermistor, a resistor that changes its resistance depending on the temperature. As the measured temperature increases, the resistance of the thermistor decreases. You will use a model TTC05103 thermistor (Figure 40).

In the *Introduction to Electronics* chapter you learned how to connect a button by building a voltage divider. Basically, the thermistor works just like a button. If its resistance is high, it is the equivalent of a button that is not pressed, while if the resistance is low, it acts like a pressed button. This is why you have to connect it similar to the button connection.

Connect one leg of the thermistor to the 5V pin and the other leg to a resistor and to an analog pin. The other leg of the resistor must be connected to GND. For a better understanding please go to *Button* to the *Introduction to Electronics* chapter.

Figure 40: TTC05103 Thermistor [13]

Here is an image showing how you should connect it. (Figure 41)

[13] `http://www.thinking.com.tw/documents/en-TTC05.pdf`

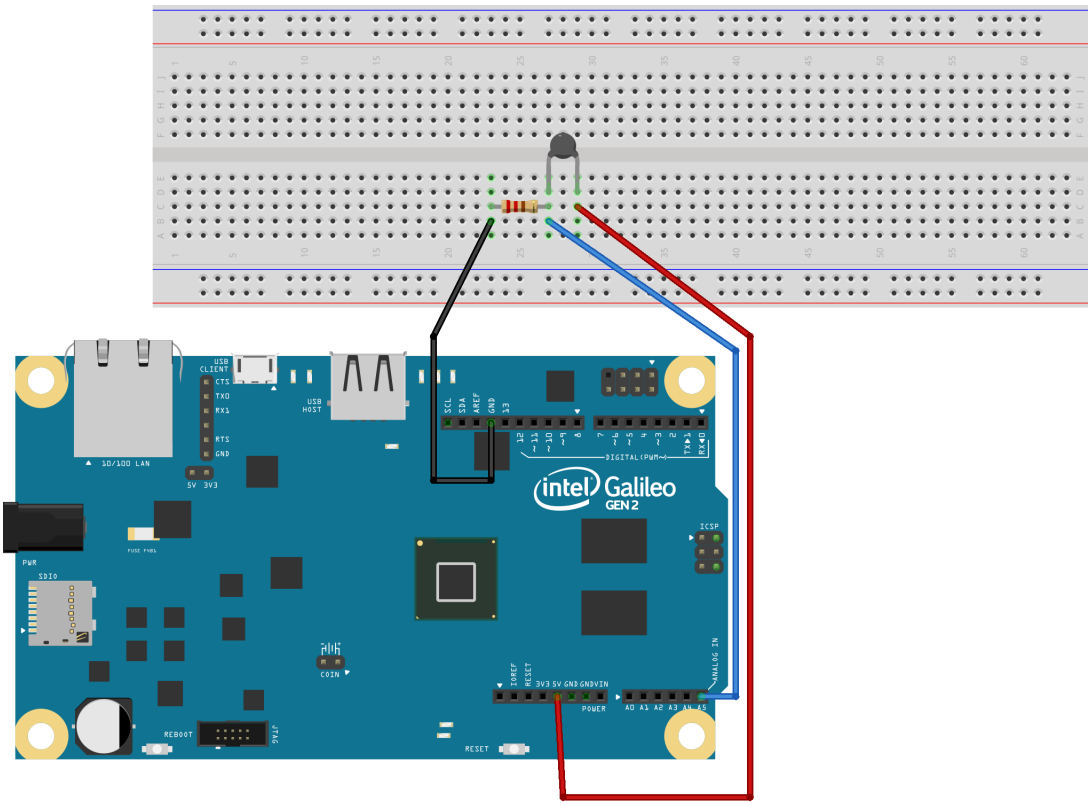

Figure 41: Connecting a thermistor

The Code

Here is the code that will make it work(Figure 42). Basically, what you have to do is read an analog value from the pin you connected the sensor to. That value is stored into the *temperature* variable. However, the read value does not represent a measured temperature in Celsius degrees, it is just a value between 0 and 1023. It's up to you to convert it to a certain measuring unit.

If you want to know the temperature in Celsius degrees you must use some of these formulas that yield the value.

$$V_m = \frac{analogRead(pin) * 5}{1023} \tag{5}$$

$$R_t = (\frac{5}{V_m} * 10000) \tag{6}$$

$$Ratio = \frac{1}{B} * ln\frac{R_t}{10000} \tag{7}$$

$$temperature = \frac{1}{\frac{1}{298} + Ratio} \tag{8}$$

B = 4050, is one of the constants that comes with the sensor's specifications.

Vm is the voltage that you get in your analog input.

Please note that every thermistor or temperature sensor has a specific resistance. The value of that resistance is different for every type of sensor.

How can you find out the resistance's value? Use it as one leg in a voltage divider circuit, while the other leg being a known resistance. Measure the voltage at the midpoint of the divider. Infer the thermistor resistance from the measured voltage.

Replace the resistance's value in the temperature formula you see in Figure 43:

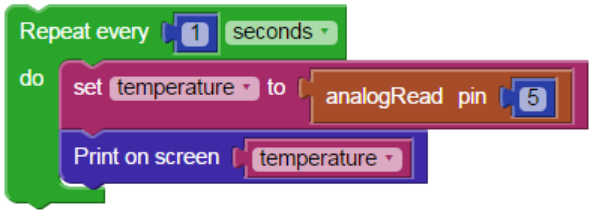

Figure 42: Reading the sensor value

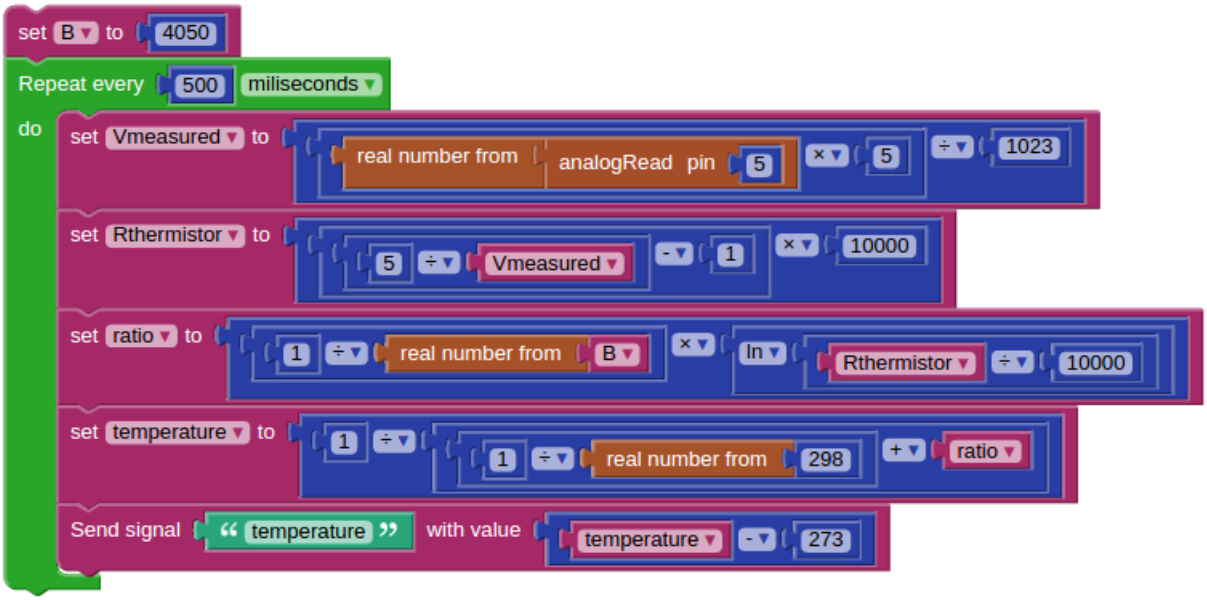

Figure 43: Temperature - Visual Programming

Also, do not forget to declare your variable inside the loop, otherwise you will only see its first value.

After you declare it, you can also print it on the screen. However, if you want to witness the variation better and faster, you can plot a graph.

Go to the *Signals* tab and chose the *Send signal* block. The formula you used will return the temperature in Kelvin degrees so make sure you send the signal with the value *temperature - 273* in order to obtain Celsius degrees. The block will send a signal with a name and value to the dashboard.

Fill in the name of the signal as *temperature*. Use a thermometer just like you see in Figure 44. The thermometer will receive the signal *temperature* and it will display the computed values.

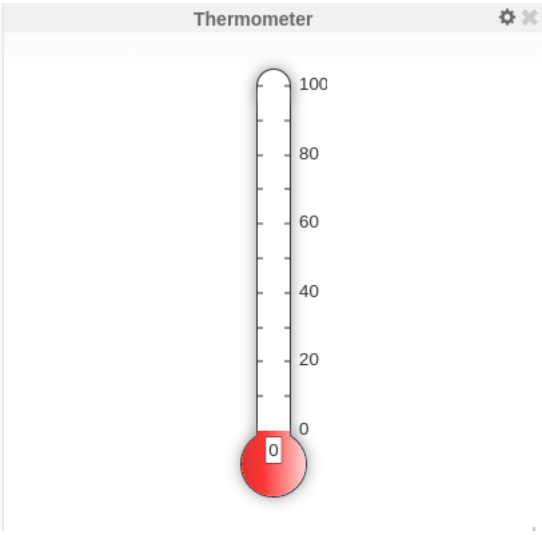

Figure 44: Thermometer

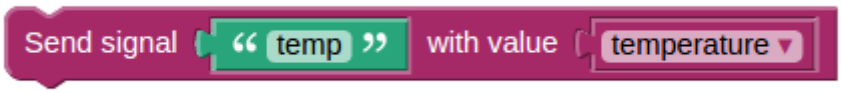

Figure 45: Sending a signal in Visual Programming

Choose the Thermometer from the list.

Name the signal *temperature*, just like you did in the block that sends that signal. (Figure 45)

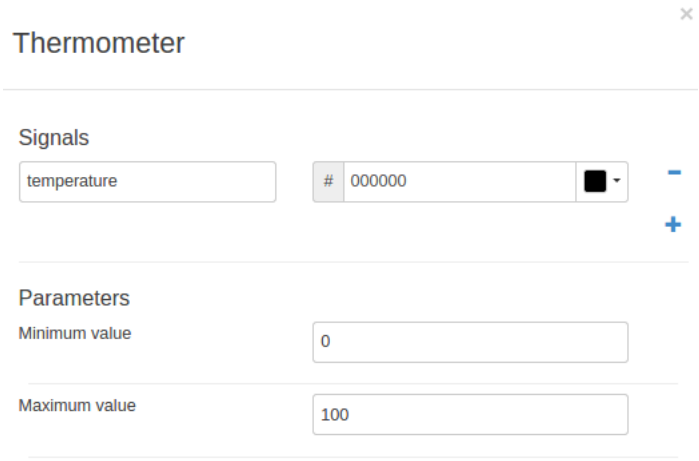

Figure 46: Signal Settings

Now run the project.

If you prefere the Python or JavaScript code, you have it here:

Python

```python
from wyliodrin import *
import math
from threading import Timer
B = None
Vmeasured = None
Rthermistor = None
ratio = None
temperature = None
pinMode (5, 0)
B = 4050
def loopCode():
  global Vmeasured, Rthermistor, ratio, B, temperature
  Vmeasured = (float (analogRead (5))) * 5 / 1023
  Rthermistor = (5 / Vmeasured - 1) * 10000
  ratio = 1 / (float (B)) * math.log(Rthermistor / 10000)
  temperature = 1 / (1 / (float (298)) + ratio)
```

```
  sendSignal('temperature', temperature - 273)
  Timer(0.5, loopCode).start()
loopCode()

math.log(0)
```

Javascript

```
var B;
var Vmeasured;
var Rthermistor;
var ratio;
var temperature;
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (5, 0);

B = 4050;
function loopCode()
{
  Vmeasured = (parseFloat (wyliodrin.analogRead (5))) * 5 / 1023;
  Rthermistor = (5 / Vmeasured - 1) * 10000;
  ratio = 1 / (parseFloat (B)) * Math.log(Rthermistor / 10000);
  temperature = 1 / (1 / (parseFloat (298)) + ratio);
  wyliodrin.sendSignal('temperature', temperature - 273);
}
setInterval(loopCode, 500);
Math.log(0);
```

You can watch an online tutorial on how to create this project [14].

[14] https://www.youtube.com/watch?v=kb8MXZsyZF4&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=5

LEDs Line

You are going to use a string of LEDs (10 in total) to make a LED chase effect controlled by a potentiometer.

What You Need

- Your Intel® Galileo board
- Ten LEDs
- Ten 220 Ω Resistors
- One breadboard
- One potentiometer RM065
- Fourteen jumper wires

Building The System

Connect everything like you see in Figure 47.

Attention! You may start at any pin you like as long as you connect 10 LEDs. In our example they are connected using pins 3 to 14. If you want to start from another pin you have to update the code. Should you want to use fewer cables, simply connect all the ground pins of the LEDs on one *ground line*, then wire that up with a simple cable to the GND pin.

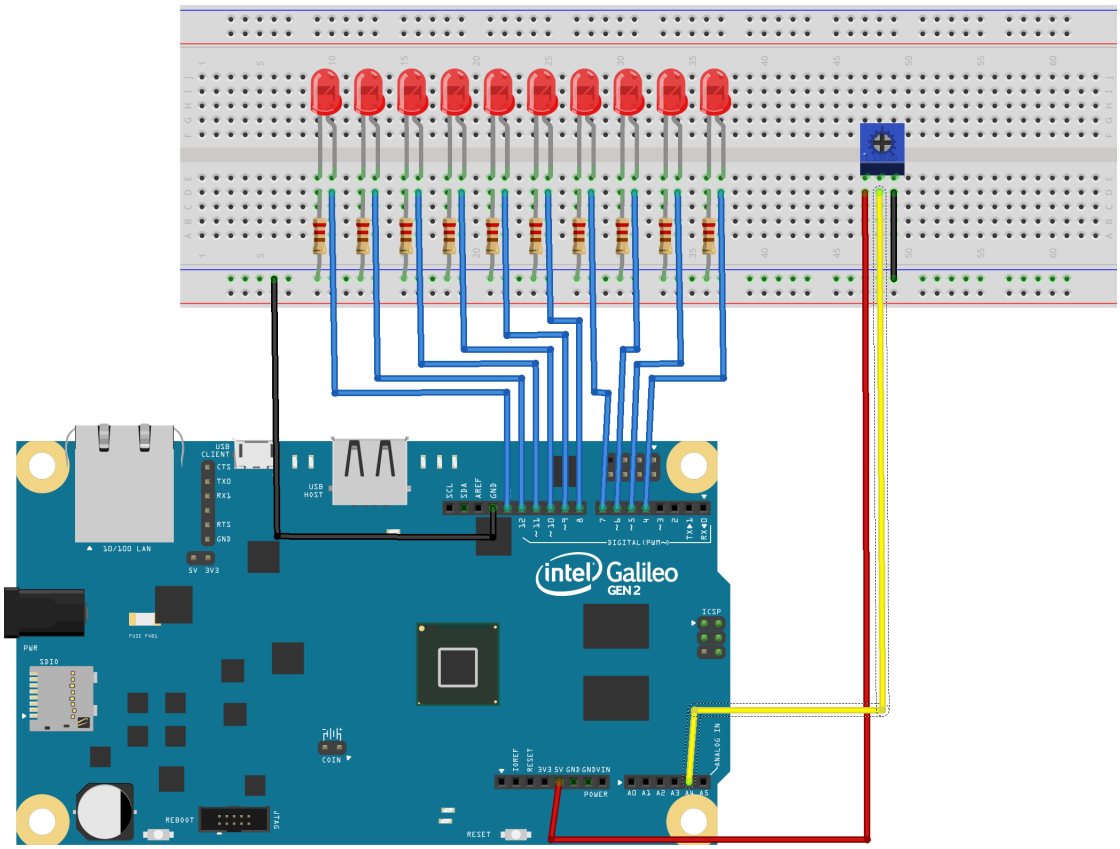

Figure 47: Connecting Leds

The Code

The 10 LEDs must turn on and off one by one from the first to the last and back. With the potentiometer you control the change speed.

First you will make the LEDs turn on and off one by one at a certain speed to get used with how you should write the program. (Figure 48). Basically, you have to turn LEDs on and off one by one. The *currentLED* variable stores the currently lighting LED. You also have to store the direction (left to right or right to left).

The *change currentLED by direction* changes the value of the currentLED with the direction's value. The *currentLED* can either increase or decrease

its value by one, depending on the *direction's* value.

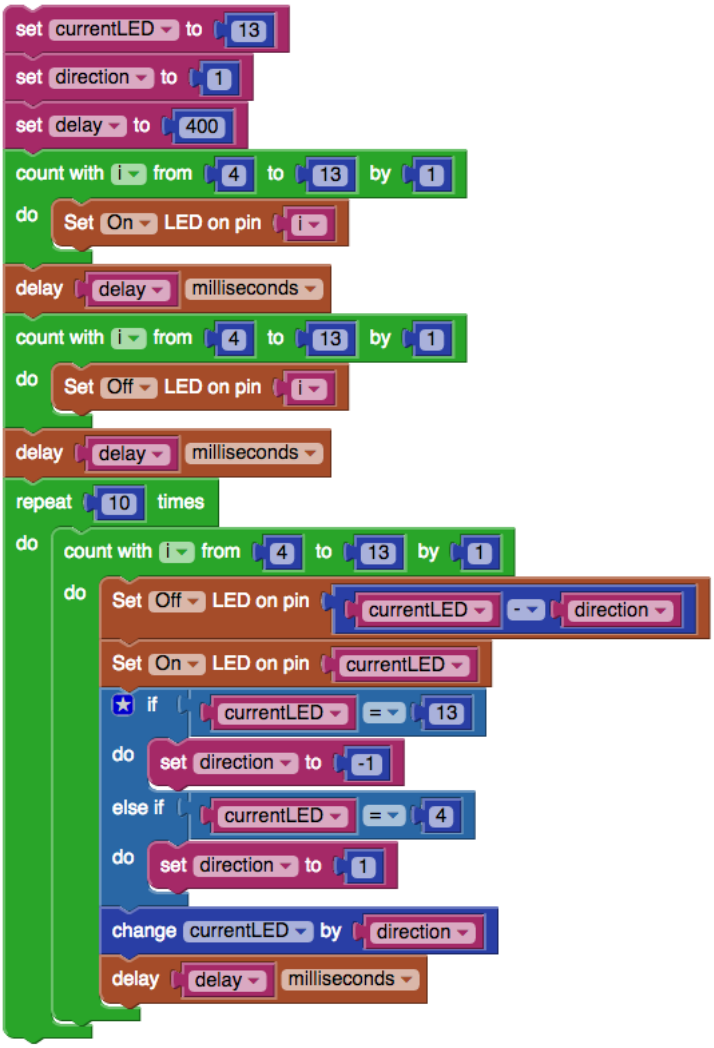

Figure 48: LED chase in Visual Programming without potentiometer

Now that you know how this works, let's use a potentiometer for adjusting the speed. This will simply update the *delay* variable used earlier.

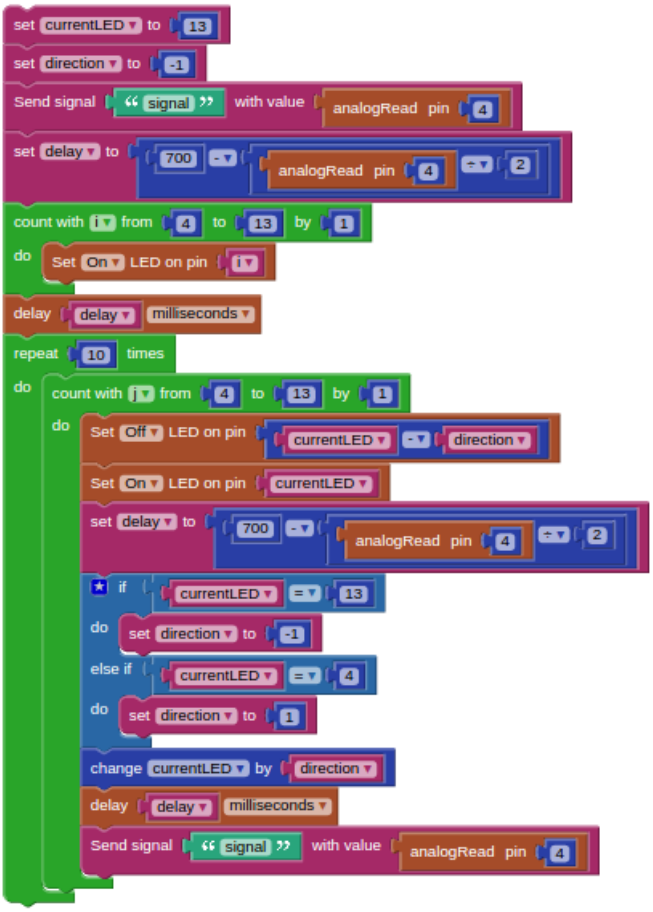

Figure 49: LED chase in Visual Programming with potentiometer

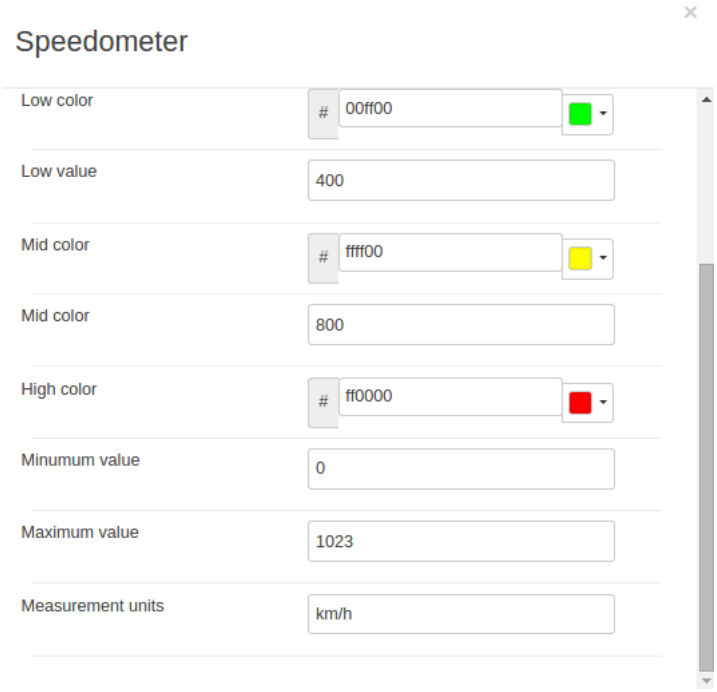

Figure 50: Speedometer settings

You will find the blocks you need in Figure 49.

You can also use the *Send signal* block for a better view of the potentiometer (rotary angle sensor) being updated. Please drag a Speedometer from the Dasboard. Make sure that the maximum allowed is 1023. You can update the colors like in Figure 50. Don't forget to update the signal's name in the dashboard to *signal*.

You will make a list of all the pins and set the direction. When the direction is 1, the LEDs will turn on in ascending order. When the direction is -1 the LEDs will turn on in descending order. When the last LED will turn on and off, the direction will change its status from 1 to -1. After the direction becomes -1 and the first LED will turn on and off, the direction will become 1 and so on. In that way the LEDs will turn on and off from the first one to de last one and vice versa. It's like you count from 1 to 10 and after 10 you turn the other way around from 10 to 1. If you don't want to use blocks you

have the code:

Python

```
from wyliodrin import *
from time import *
currentLED = None
direction = None
delay = None
i = None
j = None
pinMode (4, 0)
currentLED = 13
direction = -1
sendSignal('signal', analogRead (4))
delay = 700 - (analogRead (4)) / 2
for i in range(4, 14):
  pinMode (i, 1)
  digitalWrite (i, 1)
sleep ((delay)/1000.0)
for count in range(10):
  for j in range(4, 14):
    pinMode (currentLED - direction, 1)
    digitalWrite (currentLED - direction, 0)
    pinMode (currentLED, 1)
    digitalWrite (currentLED, 1)
    delay = 700 - (analogRead (4)) / 2
    if currentLED == 13:
      direction = -1
    elif currentLED == 4:
      direction = 1
    currentLED = (currentLED if type(currentLED) in (int, float, long) \
    else 0) + direction
    sleep ((delay)/1000.0)
```

```
31      sendSignal('signal', analogRead (4))
```

Javascript

```
1  var currentLED;
2  var direction;
3  var delay;
4  var i;
5  var j;
6  var wyliodrin = require("wyliodrin");
7  wyliodrin.pinMode (4, 0);
8  currentLED = 13;
9  direction = -1;
10 wyliodrin.sendSignal('signal', wyliodrin.analogRead (4));
11 delay = 700 - (wyliodrin.analogRead (4)) / 2;
12 for (i = 4; i <= 13; i++) {
13   wyliodrin.pinMode (i, 1);
14   wyliodrin.digitalWrite (i, 1);
15 }
16 wyliodrin.delay (delay);
17 for (var count = 0; count < 10; count++) {
18   for (j = 4; j <= 13; j++) {
19     wyliodrin.pinMode (currentLED - direction, 1);
20     wyliodrin.digitalWrite (currentLED - direction, 0);
21     wyliodrin.pinMode (currentLED, 1);
22     wyliodrin.digitalWrite (currentLED, 1);
23     delay = 700 - (wyliodrin.analogRead (4)) / 2;
24     if (currentLED == 13) {
25       direction = -1;
26     } else if (currentLED == 4) {
27       direction = 1;
28     }
29     currentLED = (typeof currentLED == 'number' ? currentLED : 0) \
30     + direction;
```

```
31      wyliodrin.delay (delay);
32      wyliodrin.sendSignal('signal', wyliodrin.analogRead (4));
33    }
34  }
```

You can watch an online tutorial on how to create this project [15].

Parking Sensor

Let's make something fun. You will use what you learned and make a parking sensor.

What You Need

1. Your Intel® Galileo
2. One Shift register - 74HC595
3. One breadboard
4. One Distance sensor Sharp GP2D120XJ00F
5. Eight 220 Ω Resistors
6. One buzzer
7. Eighteen jumper wires.

Building The System

It is likely to run out of pins. To avoid this situation you can use a shift register. You will need a 74HC595 piece (Figure 51). You can use it to control 8 outputs at a time using only a few pins, so you will use only 8 LEDs. You can link multiple registers together to extend your output. Actually you will

[15] https://www.youtube.com/watch?v=NSz20ySxD0k&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=6

shift out a byte of data one bit at a time.

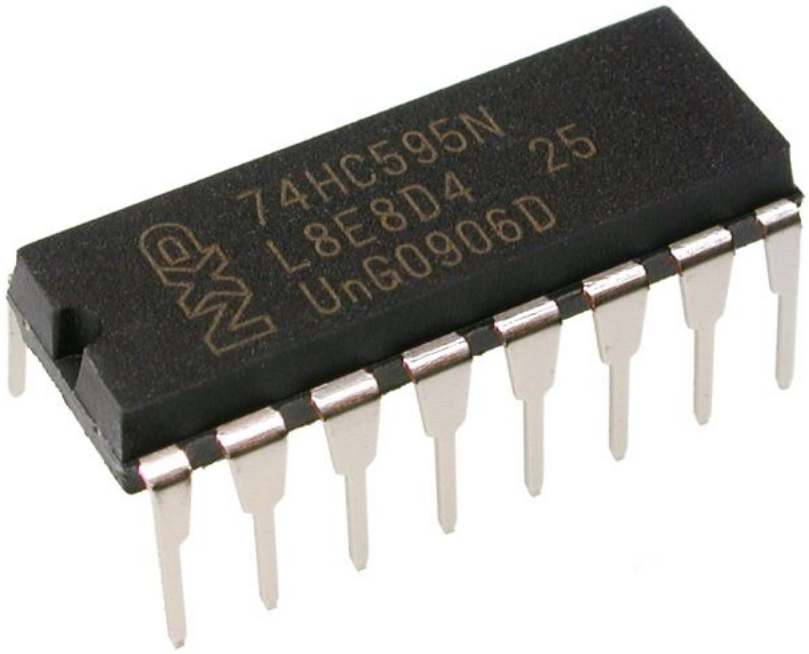

Figure 51: 74HC595 [16]

You can see the scheme in Figure 52.

[16] https://content.solarbotics.com/products/photos/df309280702fe1b526588e932c7ccfa1/lrg/74hc595n-dscn3898.JPG

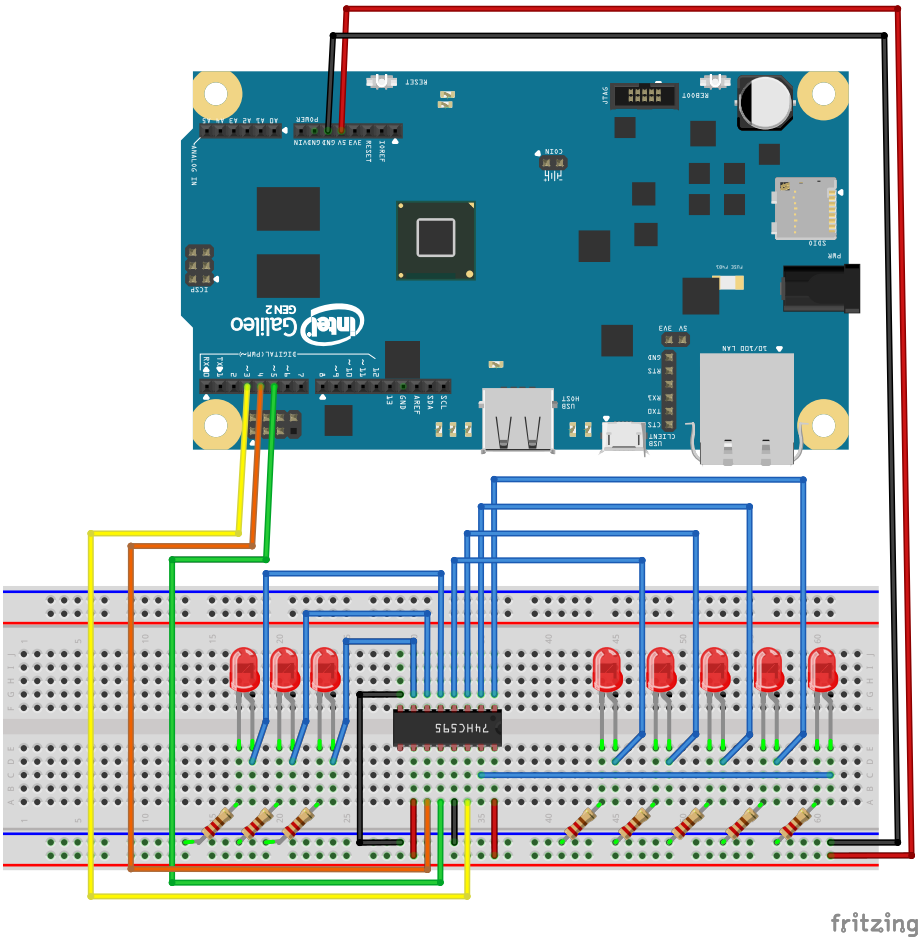

Figure 52: Connecting LEDs with a 595

The distance sensor is an analog one. You can keep the LEDs and the 595 and just add the distance sensor. The PIN connections might vary, but this one has 3 cable connections: the analog input, the ground (GND) and the VCC (5V) - please inspect Figure 53 for more information on this.

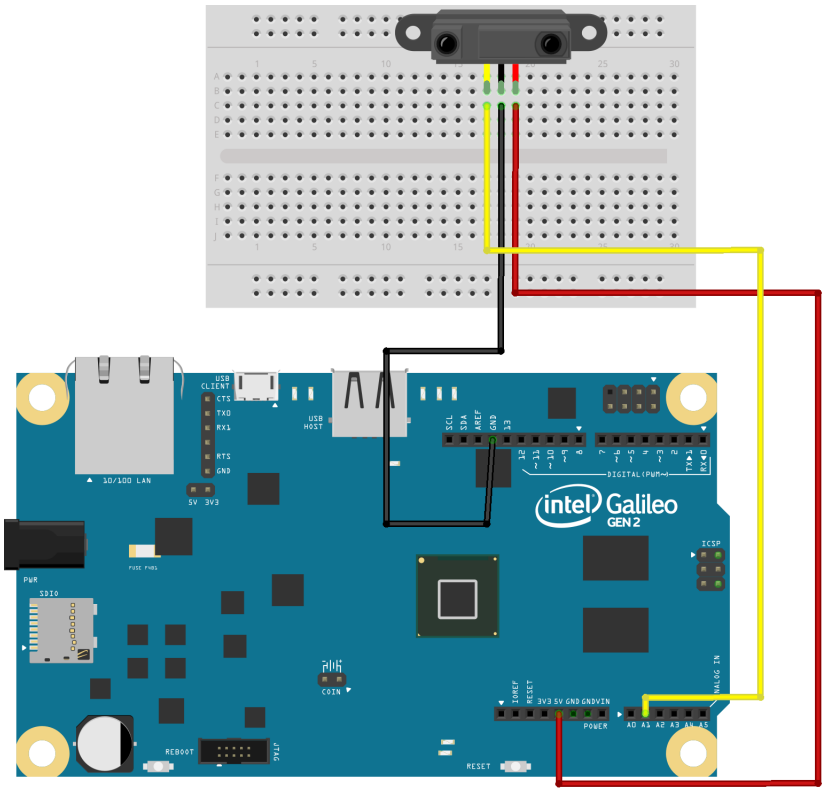

Figure 53: Connecting a Distance sensor

To make it look like a real parking sensor you can connect a buzzer. For connection shematics please go to *S.O.S. Morse Code*. You will remember that it works pretty much like an LED. As you will see in the code, the buzzer is connected on pin 11.

The Code

First, let's get you used with the *shift out* function.

The *Shift out* block will send out a value bit by bit to the data pin and generate a clock signal on the clock pin. It will start with the LSB (Least Significant Bit).

If you want to light up the first LED you have to send 2 to the power of 0, which equals 1. For the second LED you have to send 2 to the power of 1. If you want to ligh up both LEDs at the same time, you have to send 2 to the power 0 plus 2 to the power 1 so $1 + 2 = 3$. If you look closer you can see that 3 is just one short of 2 to power 2, which is exactly the number of LEDs we were trying to turn on. So, for the first n numbers you just have to send 2 to the power n (where n is the number of LEDs) minus 1. Check the code in Figure 54.

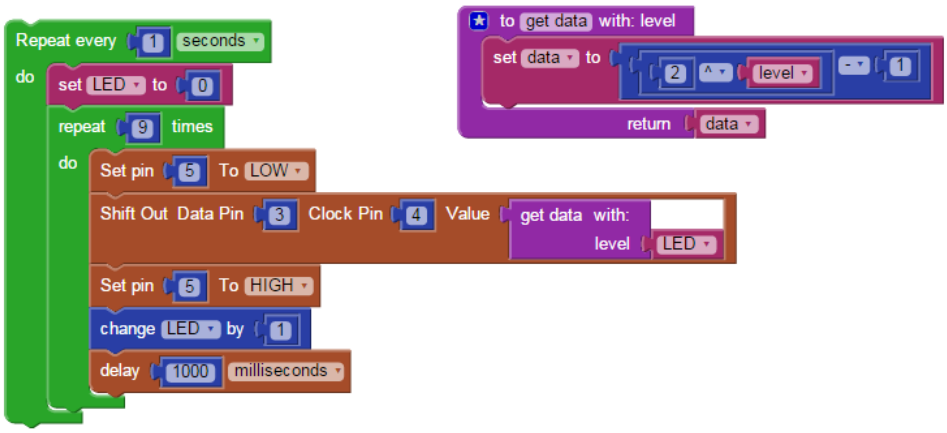

Figure 54: Shift out in Visual Programming

The code is very simple. For the 595 you must fill in the spaces with the Data pin, 3, and The Clock pin, 4.

As this is an analog sensor that means that you should receive a value from 0 to 1023, 0 when the object is not in range and 1023 when the object is immediately near the sensor.

You can use different colored LEDs like 5 Green, 1 Yellow and 2 Red. Basically you will have up to 8 LEDs lighting on depending on the distance.

Let's see how you can do it. First, you will read the value sent by the sensor and will map it because you have 8 LEDs. So as your object is closer, more LEDs will turn on one by one until all of them are turned on. First you need

a function that maps the value received from the sensor. You can suppose that the data received from the sensor will go up to 1023 but it might not be the case. If you want your parking sensor to be very exact you must perform some tests and see what is the maximum value you receive. After some tests we found that the max value is around 630(Figure 55).

Now let's add the buzzer. You can make it beep when there are more than 6 LEDs turned on. That means that you will turn it on, add a delay and turn it off.

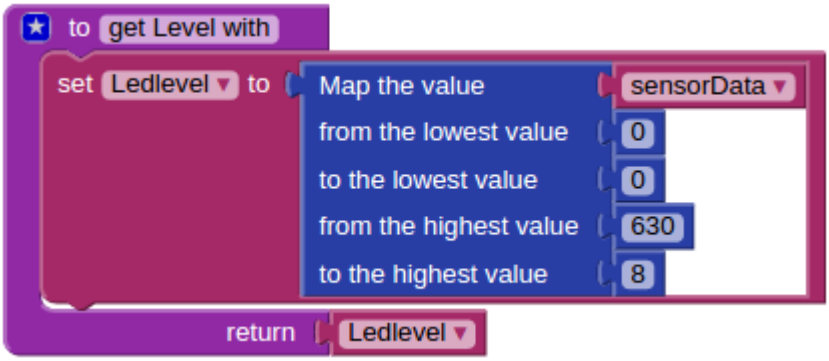

Figure 55: getLevel function

You also need a function that will light on the LEDs depending on this value (Figure 56).

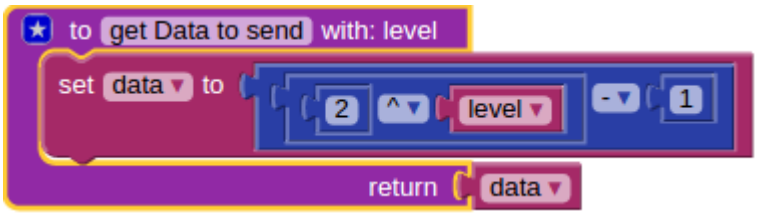

Figure 56: getData function

You can also send a signal with the value received from the sensor. From the Dasboard add a Spline Line, its maximum axes value will be 1023 and it will have fixed axes to measure its scale accurately. (Figure 57)

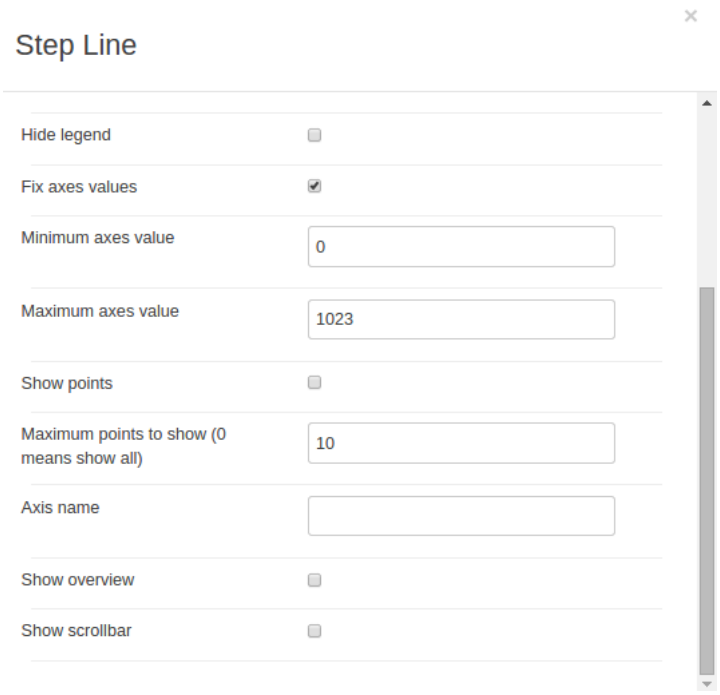

Figure 57: Spline Line

You can see in Figure 58 what your code should look like.

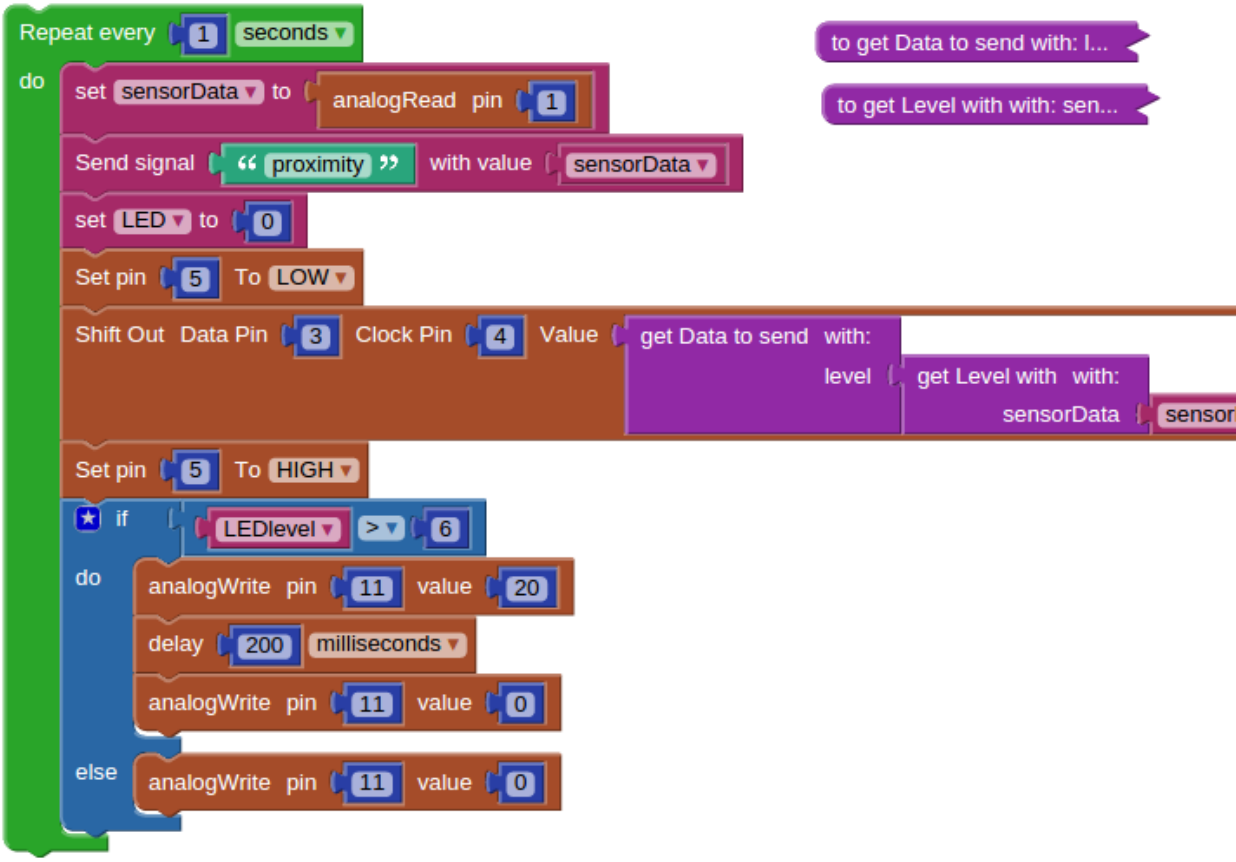

Figure 58: Parking Sensor in Visual Programming

You will see 2 unusual blocks: the 2 functions collapsed. Right click on a block and choose *collapse*. This way you can have a better image of your program as a whole.

Press Run and test it. Have fun! Also you can find the generated code below.

Python

```
1 from wyliodrin import *
2 from time import *
3 from threading import Timer
4 level = None
```

```
sensorData = None
data = None
LEDlevel = None
LED = None
pinMode (1, 0)
pinMode (5, 1)
pinMode (3, 1)
pinMode (4, 1)
pinMode (11, 1)
def get_Data_to_send(level):
  global data
  data = 2 ** level - 1
  return data
def get_Level_with(sensorData):
  global LEDlevel
  LEDlevel = map(sensorData, 0, 630, 0, 8)
  return LEDlevel
def loopCode():
  global sensorData, LED, LEDlevel
  sensorData = analogRead (1)
  sendSignal('proximity', sensorData)
  LED = 0
  digitalWrite (5, 0)
  shiftOut (3, 4, MSBFIRST, get_Data_to_send(get_Level_with(sensorData)))
  digitalWrite (5, 1)
  if LEDlevel > 6:
    analogWrite (11, 20)
    sleep ((200)/1000.0)
    analogWrite (11, 0)
  else:
    analogWrite (11, 0)
  Timer(1, loopCode).start()
loopCode()
```

Javascript

```
var level;
var sensorData;
var data;
var LEDlevel;
var LED;
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (1, 0);
wyliodrin.pinMode (5, 1);
wyliodrin.pinMode (3, 1);
wyliodrin.pinMode (4, 1);
wyliodrin.pinMode (11, 1);
function get_Data_to_send(level) {
  data = Math.pow(2, level) - 1;
  return data;
}
function get_Level_with(sensorData) {
  LEDlevel = (wyliodrin.map(sensorData, 0, 630, 0, 8));
  return LEDlevel;
}
function loopCode()
{
  sensorData = (wyliodrin.analogRead (1));
  wyliodrin.sendSignal('proximity', sensorData);
  LED = 0;
  wyliodrin.digitalWrite (5, 0);
  wyliodrin.shiftOut (3, 4, wyliodrin.MSBFIRST,
   get_Data_to_send(get_Level_with(sensorData)));
  wyliodrin.digitalWrite (5, 1);
  if (LEDlevel > 6) {
    wyliodrin.analogWrite (11, 20);
```

```
31      wyliodrin.delay (200);
32      wyliodrin.analogWrite (11, 0);
33    } else {
34      wyliodrin.analogWrite (11, 0);
35    }
36  }
37  setInterval(loopCode, 1000);
```

You can watch an online tutorial on how to create this project [17].

[17] https://www.youtube.com/watch?v=t4icd8GytL8&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr&index=7

Traffic Signaler

For this project you will build a traffic signaler consisting of a car traffic light, a pedestrian traffic light and a push button for the pedestrian requesting to cross the road.

What You Need

- Your Intel® Galileo board
- Five LEDs: 2 Red, 1 Yellow and 2 Green
- Six 150 Ω Resistors
- One breadboard
- Nine jumper wires
- One button TACT001

Building The System

Make sure your board is powered off. Please connect everything just like in Figure 59.

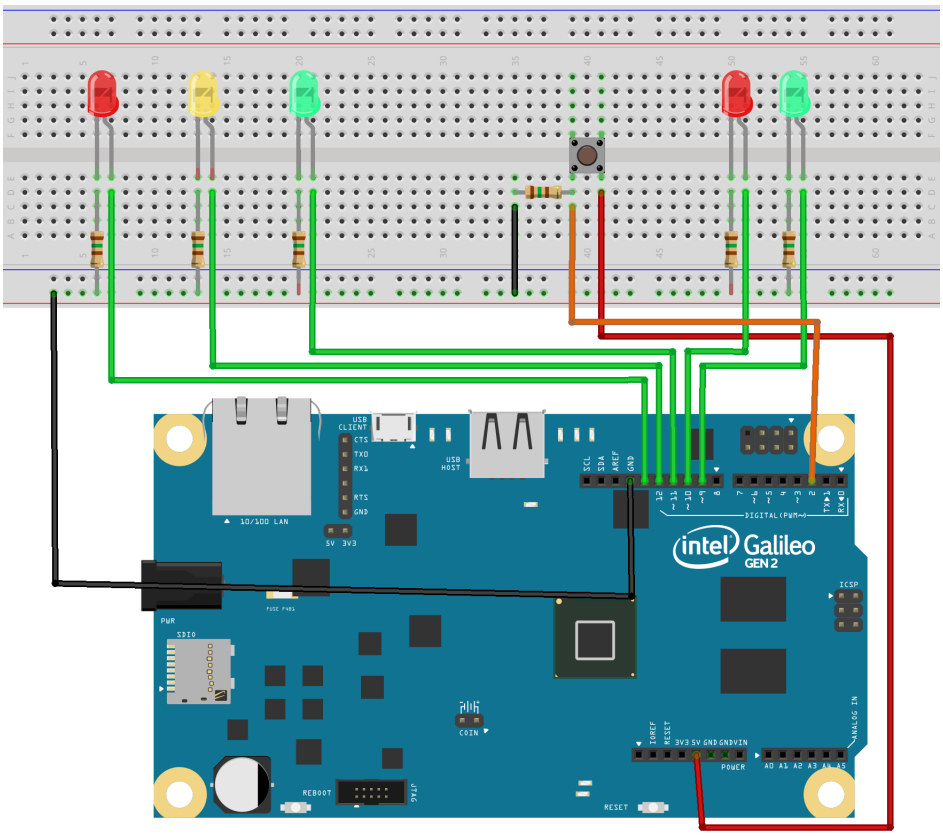

Figure 59: Building the system

In the end you got a car traffic signaler, a pedestrian traffic light and a pedestrian push button that will modify the state of the two traffic lights.

Now let's recall how the button works. If you connected it like in the schematics, you will read the value 1 when you press it. When you release it, you will read the value 0.

The button has four legs. The legs on the same side (bent in the same direction) make contact between them when the button is pressed. Make sure to connect them on the same side of the breadboard. The other two legs are the same (Figure 60).

Figure 60: Button [18]

Buttons connect two points in a circuit when you press them. When the button is not pressed, there is no connection between the two legs, so the pin is connected to ground and you read a LOW value. When the button is pressed, it makes a connection between its two legs, connecting the pin to 5 volts, so that you will read a HIGH value.

You should be accustomed with the rest of the setup from the prevoius chapters.

Power on the board.

The Code

How do you check if the button is pressed? To read the button's state you must use the *digitalWrite* block. Afterwards you must see if the state is either LOW or HIGH, then you will treat each case separately.

The board will react when the button is pressed by changing the state of the lights to make the cars stop and allow the pedestrian to cross.

The Car Traffic Light

[18] `http://www.pinoyexchange.com/forums/showthread.php?t=557755`

Here, the three LEDs must turn on one after the other. The first LED that must light on is the Red one followed by the Yellow one and the last is the Green LED.

The Pedestrian Traffic Light

As long as the Yellow and the Green LEDs of the car traffic light are lit on, the Green LED of the pedestrian traffic light must be lit off. When the Red LED of the car traffic light will turn on, the Green pedestrian LED will turn on. Whenever the button is pressed, the system will make the necessary changes to let the pedestrians pass.

First of all, it has to turn the car traffic light to yellow followed by red. Afterwards it will led the pedestrians pass for a while, then flash their green light so that they know the lights are about to change - as the pedestrians do not have a yellow light available. The first and last states of the system are the same - the lights are red for the pedestrians and green for the cars.

You have the code in Figure 61.

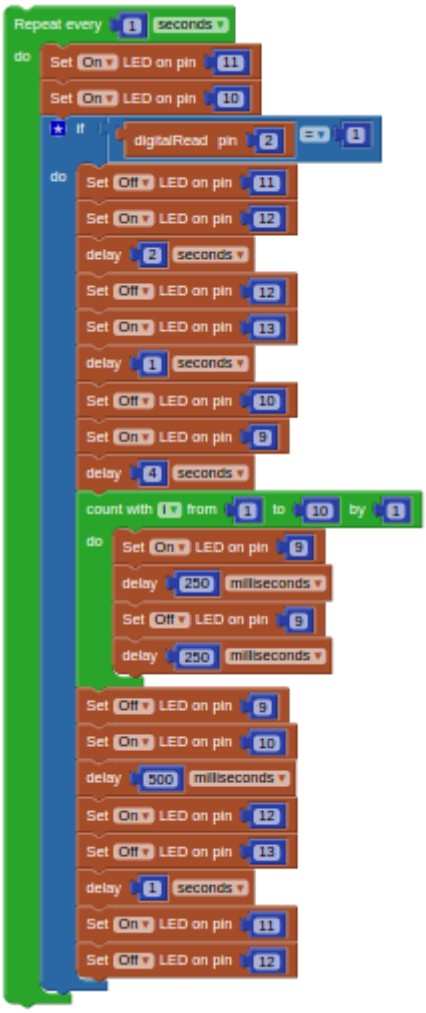

Figure 61: Traffic signaler in Visual Programming

If you prefer something more than the visual programming, here it is:

Python

```
from wyliodrin import *
from time import *
from threading import Timer
i = None
pinMode (11, 1)
pinMode (10, 1)
```

```
7  pinMode (2, 0)
8  pinMode (12, 1)
9  pinMode (13, 1)
10 pinMode (9, 1)
11 def loopCode():
12   global i
13   digitalWrite (11, 1)
14   digitalWrite (10, 1)
15   if (digitalRead (2)) == 1:
16     digitalWrite (11, 0)
17     digitalWrite (12, 1)
18     sleep (2)
19     digitalWrite (12, 0)
20     digitalWrite (13, 1)
21     sleep (1)
22     digitalWrite (10, 0)
23     digitalWrite (9, 1)
24     sleep (4)
25     for i in range(1, 11):
26       digitalWrite (9, 1)
27       sleep ((250)/1000.0)
28       digitalWrite (9, 0)
29       sleep ((250)/1000.0)
30     digitalWrite (9, 0)
31     digitalWrite (10, 1)
32     sleep ((500)/1000.0)
33     digitalWrite (12, 1)
34     digitalWrite (13, 0)
35     sleep (1)
36     digitalWrite (11, 1)
37     digitalWrite (12, 0)
38   Timer(1, loopCode).start()
39 loopCode()
```

Javascript

```
var i;
var wyliodrin = require("wyliodrin");
wyliodrin.pinMode (11, 1);
wyliodrin.pinMode (10, 1);
wyliodrin.pinMode (2, 0);
wyliodrin.pinMode (12, 1);
wyliodrin.pinMode (13, 1);
wyliodrin.pinMode (9, 1);
function loopCode()
{
  wyliodrin.digitalWrite (11, 1);
  wyliodrin.digitalWrite (10, 1);
  if ((wyliodrin.digitalRead (2)) == 1) {
    wyliodrin.digitalWrite (11, 0);
    wyliodrin.digitalWrite (12, 1);
    wyliodrin.delay ((2)*1000);
    wyliodrin.digitalWrite (12, 0);
    wyliodrin.digitalWrite (13, 1);
    wyliodrin.delay ((1)*1000);
    wyliodrin.digitalWrite (10, 0);
    wyliodrin.digitalWrite (9, 1);
    wyliodrin.delay ((4)*1000);
    for (i = 1; i <= 10; i++) {
      wyliodrin.digitalWrite (9, 1);
      wyliodrin.delay (250);
      wyliodrin.digitalWrite (9, 0);
      wyliodrin.delay (250);
    }
    wyliodrin.digitalWrite (9, 0);
    wyliodrin.digitalWrite (10, 1);
```

```
31      wyliodrin.delay (500);
32      wyliodrin.digitalWrite (12, 1);
33      wyliodrin.digitalWrite (13, 0);
34      wyliodrin.delay ((1)*1000);
35      wyliodrin.digitalWrite (11, 1);
36      wyliodrin.digitalWrite (12, 0);
37    }
38  }
39  setInterval(loopCode, 1000);
```

You learned how to use a range sensor in the previous project. Now, it would be interesting to replace the button for the pedestrians with a range sensor. Earlier you used the button to change the state of the traffic lights in the sense that if the button was pressed then the state changed, otherwise it worked normally. Now, instead of changing the state when the button is pressed, you will change it when the value read from the sensor is higher than 400.

Of course you can change this value depending on your preferences and the sensor you are working with. We advise you to take some tests and see what are values you receive from your sensor. Based on this, the system will work as you desire.

Basically, you take the first code and you change just the button part. Firstly, instead of a *digitalRead* on the pin where the button was connected, you should *analogRead* the analog pin the sensor is connected to. Secondly, instead of verifying whether the state of the button is 0 or 1, you verify if the value received is bigger or smaller than 400.

Run the project.

You can watch an online tutorial on how to create this project [19].

[19] https://www.youtube.com/watch?v=AJr0L4Kz8f0&index=8&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

Online Thermometer

In this chapter you will create an online thermometer that allows you to easily see the temperature in other cities.

What You Need

- Your Intel ® Galileo.

The Code

You just need to use a few blocks. You will get the temperature from a city using an Internet service, not a sensor. You can choose whatever city you want. Figure 62 depicts the code that prints on the screen the temperature in Paris. Basically, you need to use the *Get temperature from []* block. In this example a list is created with some text and the read temperature, then it list is printed on screen. You can create a text instead of a list by using the *Program/Text/create text with* block.

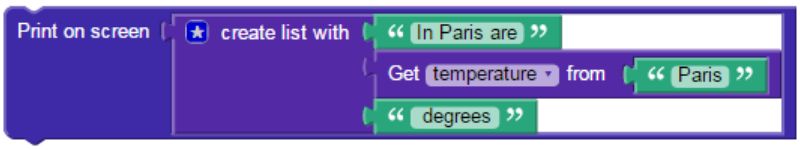

Figure 62: Getting the temperature from a city in Visual Programming

You can see the generated code here:

Python

```
import urllib
import json
def getWeather(city, param):
   response = urllib.urlopen\
   ('http://api.openweathermap.org/data/2.5/weather?q='+city)
   responseJson = json.loads (response.read())
   if param =="humidity" or param =="pressure":
     return float(responseJson['main'][param])
   if param == "temp":
     celsius = float(responseJson['main'][param])-273.15
     return celsius
   if param == "speed":
     return float(responseJson['wind'][param])
print(['In Paris are', \
getWeather('Paris', 'temp'), ' degrees'])
```

If you want something more complex you can print the temperature on two 7-segment displays and light up a LED when the temperature is negative.

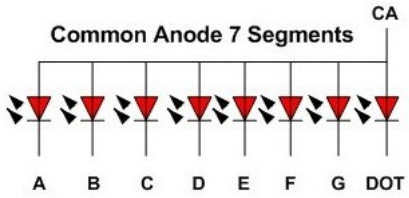

Figure 63: 7-segment display [20]

The 7-segment display consists of 8 LEDs. By lighting up only some of the LEDs you can display a letter or a number. Each LED acts like the ones you used so far. Each is connected to a digital pin, so you should know the exact pin for each LED. This is necessary because you need to know which pin to light up or off in order to display the correct number. All 8 LEDs have a

[20] http://www.ermicro.com/blog/wp-content/uploads/2009/03/pictemp_04.jpg

common anode.(Figure 63)

What You Need

1. Your Intel® Galileo
2. One LED
3. Two 7-segment display
4. Three 220 Ω Resistor.
5. One breadboard.
6. Seventeen jumper wires.

Building The System

See the connections in Figure 64.

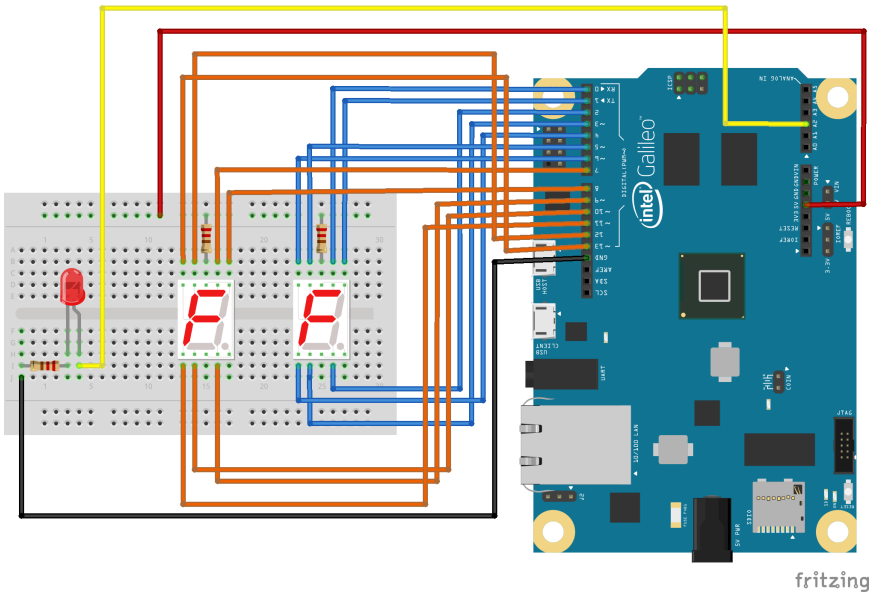

Figure 64: Connecting the 7-segment displays

Every line is a different LED and for each LED you have a corresponding pin. In order to know what every pin is, see the figure 65.

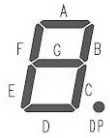

Figure 65: 7-segment display [21]

So, for the first 7-segment display you have:

A - Pin 0

B - Pin 1

C - Pin 2

D - Pin 3

E - Pin 4

F - Pin 5

G - Pin 6

And for the second:

A - Pin 7

B - Pin 8

C - Pin 9

D - Pin 10

E - Pin 11

[20] http://3.bp.blogspot.com/-5nbaUFCYLSA/UV6fRg7qduI/AAAAAAAADMc/7pv-sLjk0iY/s1600/Screen+Shot+2013-04-05+at+3.23.32+PM.png

F - Pin 12

G - Pin 13

Basically, the whole system consists of connecting 15 LEDs. You don not have to connect the floating point LED as you are not going to use it.

The Code

Firstly, you have to initialize the two 7-segment displays. Make sure to update the option for a common cathode/anode connection. For a better view of the program collapse them.(Figure 66)

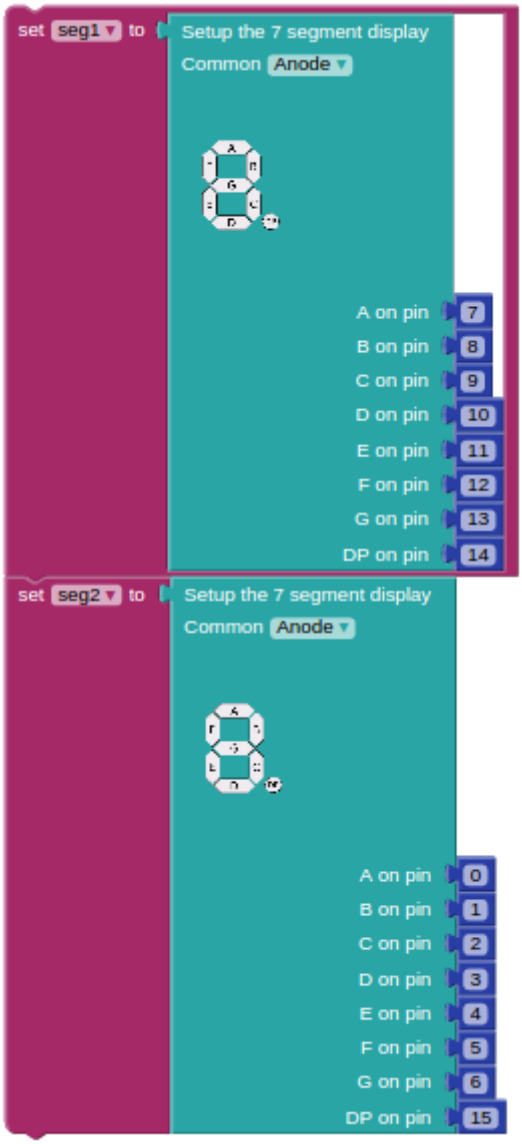

Figure 66: 7-segment display initialization in Visual Programming

Although you did not connect the point LED to any pin, you have to set a pin for it, otherwise the code will not work. This is why we suggested pins 14 and 15 (they are not connected).

You first need a function that will take the temperature from a city and then print it on the screen so you can check it everything work fine. After

that, it will check whether the temperature is negative. If the temperature is negative the LED will turn on and you will change the sign of the temperature variable. This way, if the LED is on, you will know that the temperature you see on the 7-segment is negative. You also need another condition: if the temperature is between 0 and 10 (a single digit number), display 0 on the first 7-segment display and then the actual temperature on the other one. Otherwise, you will display the first digit of your number (temperature/10) on the first 7-segment and the last digit on the second one (Figure 67).

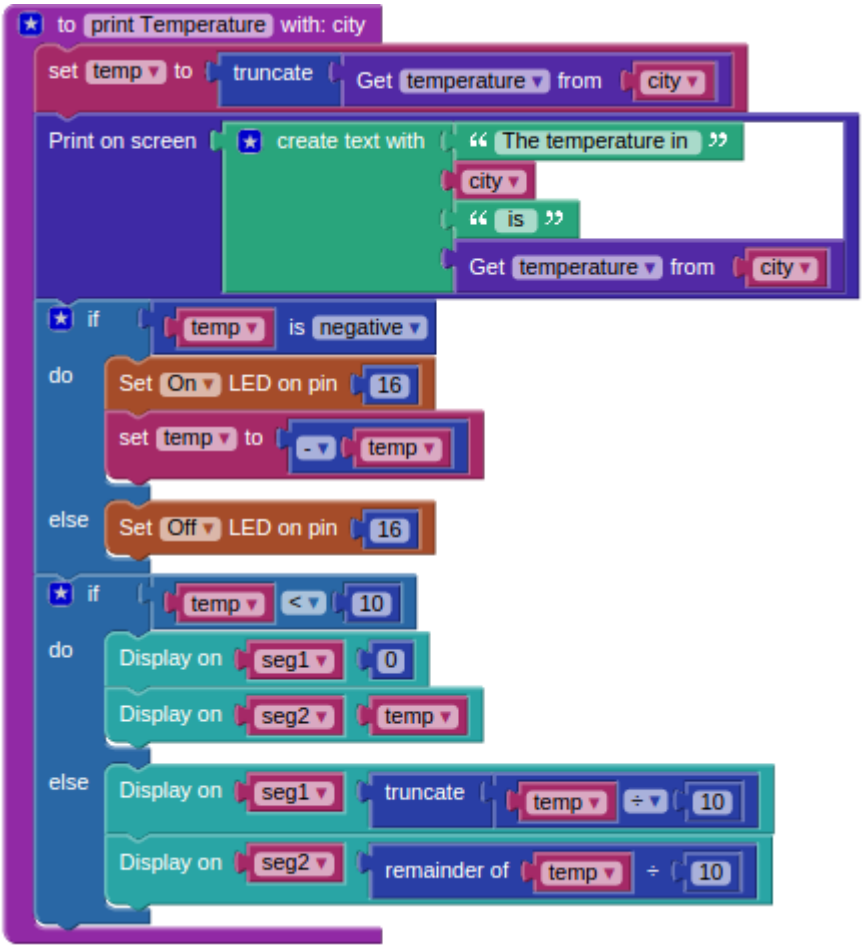

Figure 67: printTemperature function

Your program should look like the one in Figure 68. You will also display the temperature of each city in a graph. You can use a thermometer like you did in the *Temperature Sensor Application* chapter, but this time you will have 3 plots, each one for a city and a different signal (Figure 69). To achieve this, you have to use the *Signals/Send signal [] with value* block, where the signal name is the city you read the temperature from and the value is the temperature. The result is that 3 different signals will be sent to 3 different graphs.

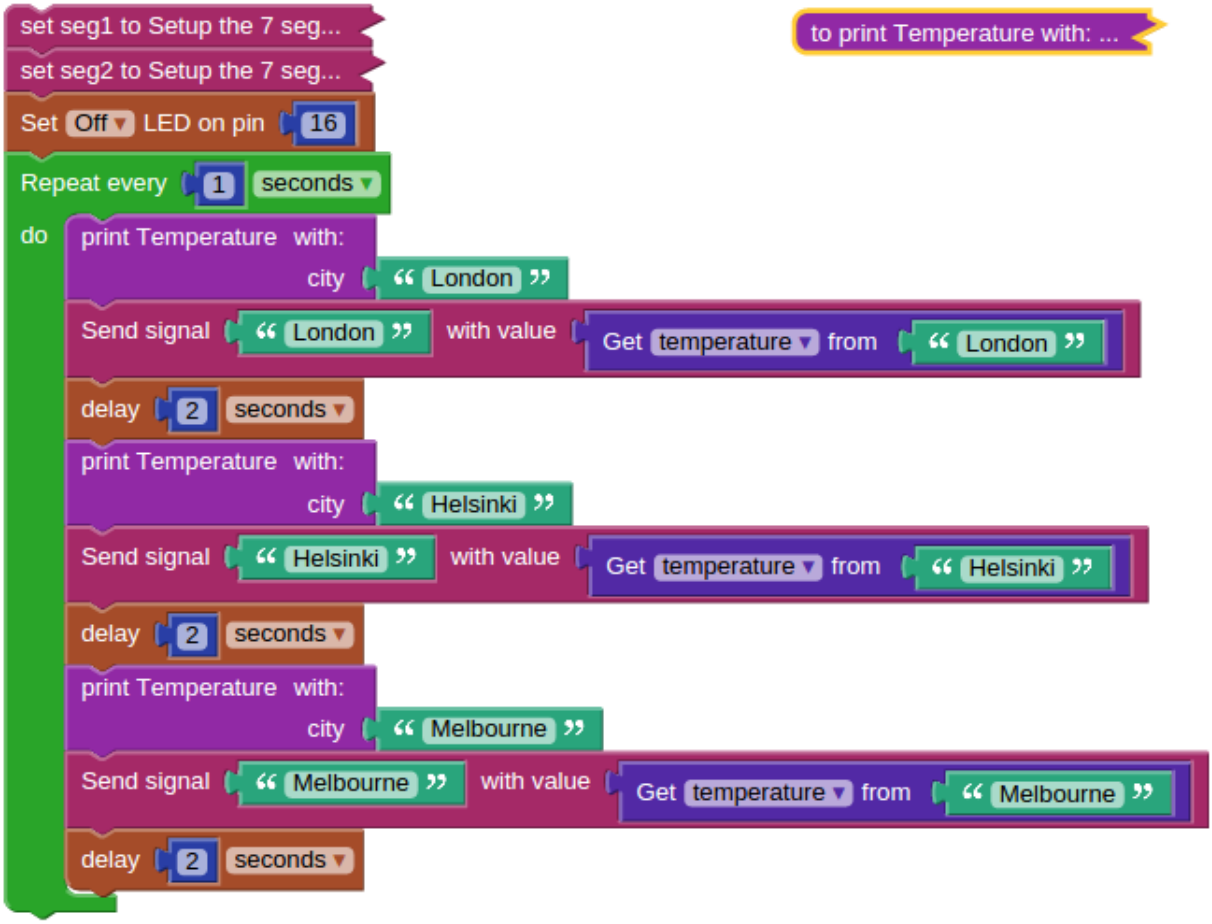

Figure 68: Online Thermometer in Visual Programming

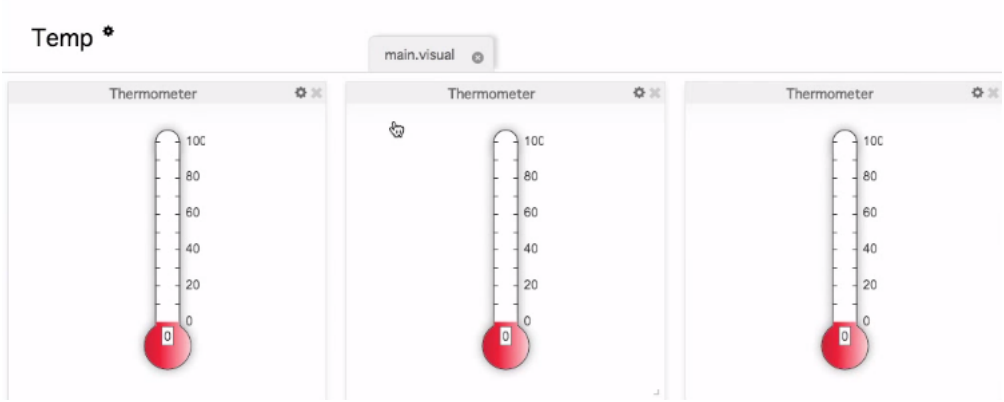

Figure 69: Thermometers

Python

```python
from wyliodrin import *
import urllib
import json
from time import *
from threading import Timer

seg1 = None
city = None
seg2 = None
temp = None

pinMode (7, 1)
pinMode (8, 1)
pinMode (9, 1)
pinMode (10, 1)
pinMode (11, 1)
pinMode (12, 1)
pinMode (13, 1)
pinMode (14, 1)
pinMode (0, 1)
pinMode (1, 1)
```

```
22 pinMode (2, 1)
23 pinMode (3, 1)
24 pinMode (4, 1)
25 pinMode (5, 1)
26 pinMode (6, 1)
27 pinMode (15, 1)
28 pinMode (16, 1)
29
30 def getWeather(city, param):
31    response = \
32    urllib.urlopen('http://api.openweathermap.org/data/2.5/weather?q= '+city)
33    responseJson = json.loads (response.read())
34    if param =="humidity" or param =="pressure":
35      return float(responseJson['main'][param])
36    if param == "temp":
37      celsius = float(responseJson['main'][param])-273.15
38      return celsius
39    if param == "speed":
40      return float(responseJson['wind'][param])
41
42 def displaySegment(sevenSegment, a, b, c, d, e, f, g, dp):
43   digitalWrite(sevenSegment[1], abs(sevenSegment[0]-a))
44   digitalWrite(sevenSegment[2], abs(sevenSegment[0]-b))
45   digitalWrite(sevenSegment[3], abs(sevenSegment[0]-c))
46   digitalWrite(sevenSegment[4], abs(sevenSegment[0]-d))
47   digitalWrite(sevenSegment[5], abs(sevenSegment[0]-e))
48   digitalWrite(sevenSegment[6], abs(sevenSegment[0]-f))
49   digitalWrite(sevenSegment[7], abs(sevenSegment[0]-g))
50   if len(sevenSegment)>=9: digitalWrite(sevenSegment[8], \
51   abs(sevenSegment[0]-dp))
52
53 def display(sevenSegment, value):
54   if value == "0":
```

```
55      displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 0, 1)
56    if value == "1":
57      displaySegment(sevenSegment, 0, 1, 1, 0, 0, 0, 0, 1)
58    if value == "2":
59      displaySegment(sevenSegment, 1, 1, 0, 1, 1, 0, 1, 1)
60    if value == "3":
61      displaySegment(sevenSegment, 1, 1, 1, 1, 0, 0, 1, 1)
62    if value == "4":
63      displaySegment(sevenSegment, 0, 1, 1, 0, 0, 1, 1, 1)
64    if value == "5":
65      displaySegment(sevenSegment, 1, 0, 1, 1, 0, 1, 1, 1)
66    if value == "6":
67      displaySegment(sevenSegment, 1, 0, 1, 1, 1, 1, 1, 1)
68    if value == "7":
69      displaySegment(sevenSegment, 1, 1, 1, 0, 0, 0, 0, 1)
70    if value == "8":
71      displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 1, 1)
72    if value == "9":
73      displaySegment(sevenSegment, 1, 1, 1, 1, 0, 1, 1, 1)
74    if value == 'A':
75      displaySegment(sevenSegment, 1, 1, 1, 0, 1, 1, 1, 1)
76    if value == "B":
77      displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 1, 1)
78    if value == "C":
79      displaySegment(sevenSegment, 1, 0, 0, 1, 1, 1, 0, 1)
80    if value == "D":
81      displaySegment(sevenSegment, 1, 1, 1, 1, 1, 1, 0, 1)
82    if value == "E":
83      displaySegment(sevenSegment, 1, 0, 0, 1, 1, 1, 1, 1)
84    if value == "F":
85      displaySegment(sevenSegment, 1, 0, 0, 0, 1, 1, 1, 1)
86    if value == ".":
87      displaySegment(sevenSegment, 0, 0, 0, 0, 0, 0, 0, 1)
```

```
88
89  def print_Temperature(city):
90    global temp, seg1, seg2
91    temp = int(getWeather(city, 'temp'))
92    print(''.join([str(temp_value) \
93    for temp_value in ['The temperature in ', city, \
94    ' is ', getWeather(city, 'temp')]]))
95    if temp < 0:
96      digitalWrite (16, 1)
97      temp = -temp
98    else:
99      digitalWrite (16, 0)
100   if temp < 10:
101     display(seg1, str(0))
102     display(seg2, str(temp))
103   else:
104     display(seg1, str(int(temp / 10)))
105     display(seg2, str(temp % 10))
106
107 seg1 = [1, 7, 8, 9, 10, 11, 12, 13, 14]
108 seg2 = [1, 0, 1, 2, 3, 4, 5, 6, 15]
109 digitalWrite (16, 0)
110 def loopCode():
111   print_Temperature('London')
112   sendSignal('London', getWeather('London', 'temp'))
113   sleep (2)
114   print_Temperature('Helsinki')
115   sendSignal('Helsinki', getWeather('Helsinki', 'temp'))
116   sleep (2)
117   print_Temperature('Melbourne')
118   sendSignal('Melbourne', getWeather('Melbourne', 'temp'))
119   sleep (2)
120   Timer(1, loopCode).start()
```

```
121 loopCode()
```

You can watch an online tutorial on how to create this project [22].

[22] https://www.youtube.com/watch?v=5YEEKUE8-Y0&index=9&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

Social Lamp

Now that you learned how to create small projects, let's think bigger. How about a social lamp that can be controlled by your friends?

This chapter will be a little different because you will need to use many new notions. Let's just take it step by step.

First Step

Light Up a Multicolor LED

What You Need

- Your Intel® Galileo
- One BreadBoard
- One RGB LED
- Three 330 Ω Resistors
- Four jumper wires

Building The System

First you must know what an RGB LED is. RGB stands for Red-Green-Blue, as this LED is actually composed of three LEDs. The RGB LED can have a multicolored light combining the three colors.

The LED has four legs: red, ground, green, blue. The longer leg is the ground so you can use this to position the LED correctly. What you have to do is to connect each of the three LED like you would do with a regular one. However, each of the LEDs has to be connected to an analog pin. You will use an analog value to light up each of the three LEDs. This is how you can create any color you wish. Basically, eash LED lights up with a certain intensity and you see the color obtained by combining the three different intensities.

Connect the RGB LED to the board following the schematics in Figure 70.

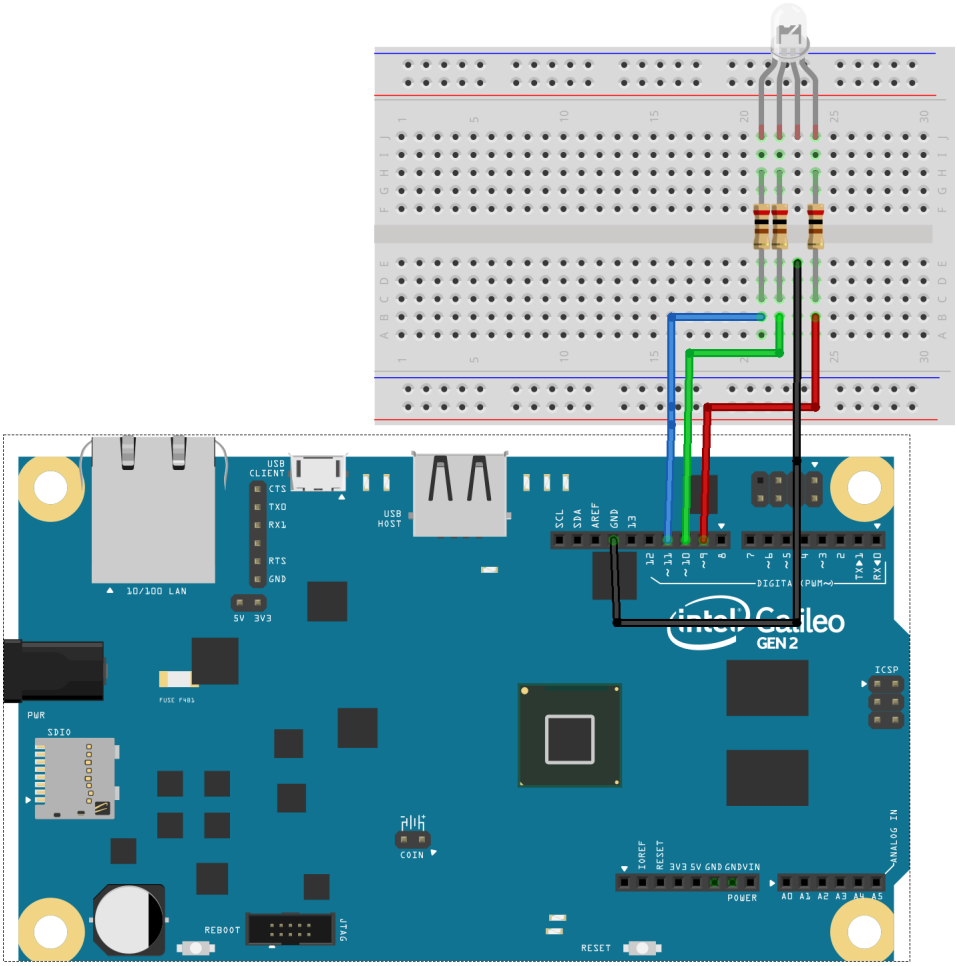

Figure 70: Connecting an RGB LED

The Code

Figure 71 displays the block you should use to control the RGB LED. To use the block you have to select a color and write the pins that you connected the LEDs to. They are 9 for Red, 10 for Green and 11 for Blue. What the block will do, is to light up with a different intensity each of the LED in order to obtain the desired color.

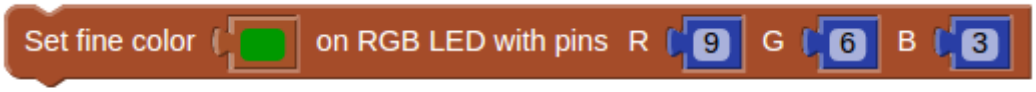

Figure 71: Setting colors for an RGB LED in Visual Programming

Python

```python
from wyliodrin import *
import struct
def colorToRGB (color):
  return struct.unpack ('BBB', color[1:].decode('hex'))
def basic_color (color):
  value = 0
  if color>=128:
    value = 1
  else:
    value = 0
  return value
pinMode (9, 1)
pinMode (6, 1)
pinMode (3, 1)
color = colorToRGB ('#009900')
analogWrite (9, color[0])
analogWrite (6, color[1])
analogWrite (3, color[2])
```

Javascript

```javascript
var wyliodrin = require("wyliodrin");
function colorToRGB(color)
{
  var result = /^#?([a-f\d]{2})([a-f\d]{2})([a-f\d]{2})$/i.exec(color);
  return result ? {
    r: parseInt(result[1], 16),
    g: parseInt(result[2], 16),
```

```
    b: parseInt(result[3], 16)
  } : null;
}

wyliodrin.pinMode (9, 1);
wyliodrin.pinMode (6, 1);
wyliodrin.pinMode (3, 1);
color = colorToRGB ('#009900')
wyliodrin.analogWrite (9, color.r)
wyliodrin.analogWrite (6, color.g)
wyliodrin.analogWrite (3, color.b)
```

Second Step

Boards Communication

If you have friends that have also Wyliodrin configured boards you can use them to communicate and it is actually very easy.

You can send text messages and print them on the screen or on the LCD and you can control other boards' pins as well.

The only thing is that you need *the approval*. So each of you has to be connected to Wyliodrin and each will use specific blocks. Here is an example of what that may look like:

The sender:

The sender and the receiver will have different codes. Look at Figure 72 for the sender code.

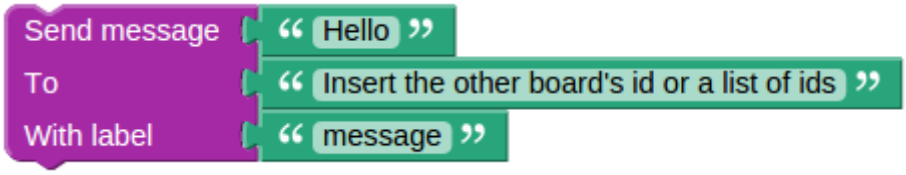

Figure 72: Sender - Board Communication in Visual Programming

Complete the *To* label with a list comprising the IDs of the boards you want to send the message to.

You can get the board's ID from the main page, just click on the board's settings button (Figure 73).

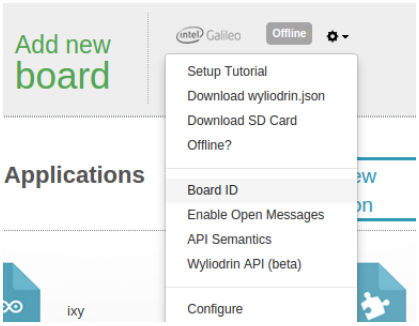

Figure 73: Board ID

Complete the *Label* field with whatever identifier you want to give to the communication channel you open with the other teams. The one receiving the message has to be aware of the label so that they can subscribe to that channel. Think of these like ports used in networking, if it makes it easier.

The receiver:

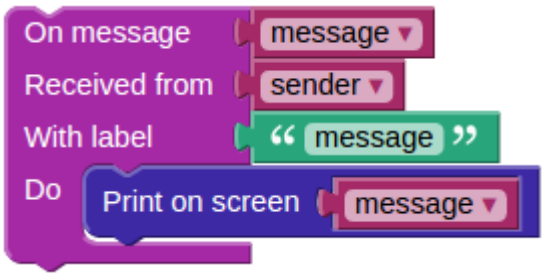

Figure 74: Receiver - Board Communication in Visual Programming

You can even receive messages from several other members of a team by using the *On message [] Received from[] With label[] Do* block (Figure 74).

The message you receive will be stored in the message variable. The sender will be stored in the sender variable.

The label has to be the same with the one used by the colleague sending the message.

All of you must have the projects running at the same time in order to have the boards to be able to communicate.

Social Lamp

Now that you learned the basis, let's build the Social Lamp.

The setup consists of an RGB LED, 3 buttons and the board. In the end you will control the light of another friend's *Social Lamp*.

What You Need

1. Your Intel® Galileo board
2. One RGB LED
3. Five 330 Ω Resistors

4. One breadboard

5. Fifteen jumper wires

6. Three buttons TACT001

Building The System

You have 3 buttons and each button controls a color of the RGB LED. One button for Red, one for Green and the last for Blue. The sender must complete the *To* field with the Board ID of the board they want to control. There are 3 variables (red, green and blue) that control the amount of each color that is being used to light up the RGB LED.

Each press of a button will increase the amount of that particular color up to 255. The steps with which this increase takes place are controlled by a variable called *step*.

The fun thing about this is that the sender sends a color code and then the receiver simply adds the message they are receiving as a direct input for the fine color of the LED - the system takes care of everything else.

Look at Figure 75 for the sender wiring.

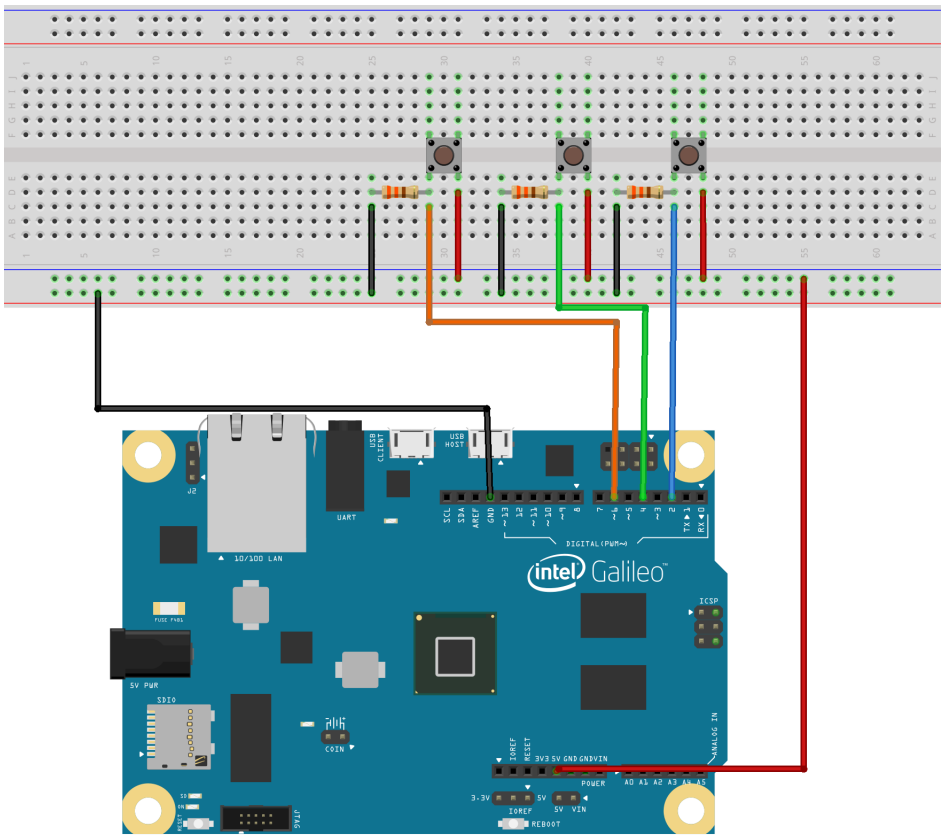

Figure 75: Social Lamp - sender

The recipient of the message has to connect the RGB LED correctly, then check for the board ID to assure that they receive the message from the correct board and make sure they are using the same label as the sender.

The Code

The sender: - Figure 76

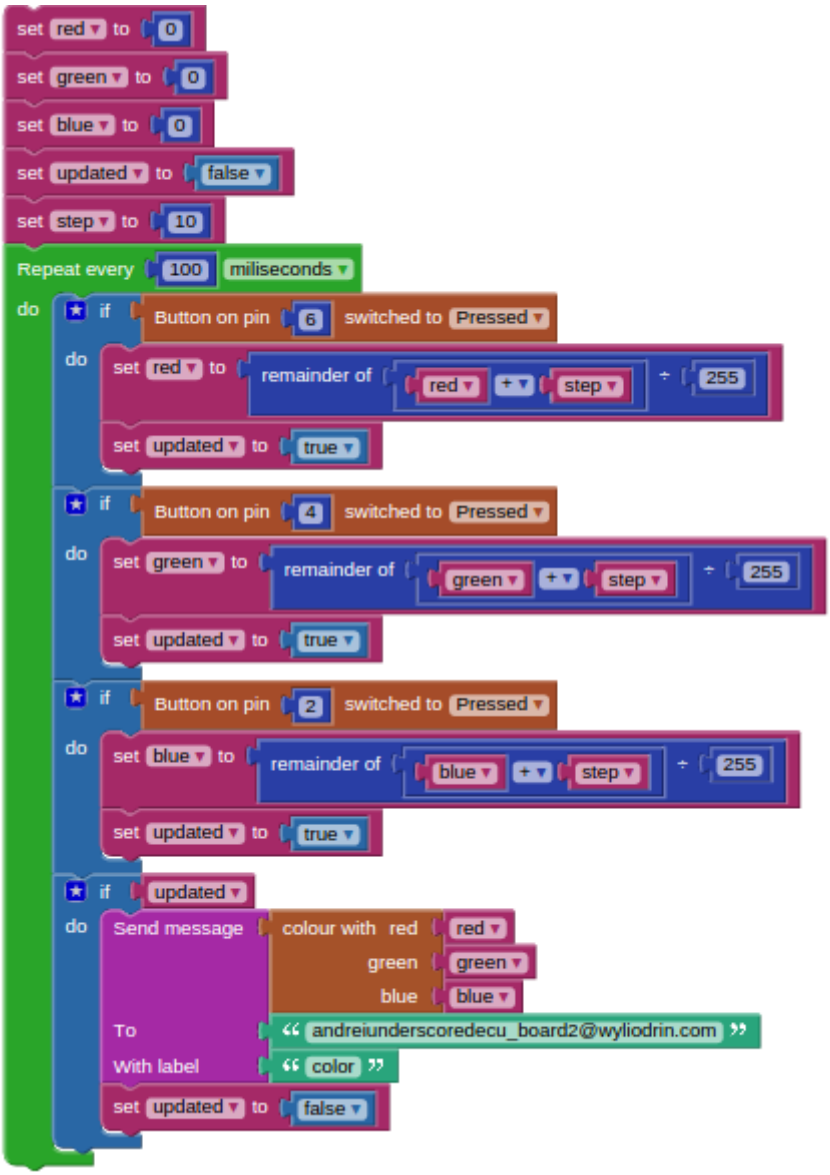

Figure 76: Social Lamp - sender in Visual Programming

The values that you have to send go from 0 to 255, so you are going to increase in steps of ten but at one point you will get to 255, so you need to pay attention to that. What you are going to do is to divide the value by 255 so whenever it gets above, it will move down and the variable will always have values between 0 and 255.

Python

```python
from wyliodrin import *
import json
from threading import Timer
red = None
green = None
blue = None
updated = None
step = None
pinMode (6, 0)
buttons = {}
def buttonSwitched(button, expectedValue):
  value = digitalRead (button)
  stable = True
  for i in range (100):
    valuenext = digitalRead (button)
    if value != valuenext:
      stable = False
  if stable:
    if button in buttons and value != buttons[button]:
      buttons[button] = value
      return value == expectedValue
    elif not button in buttons:
      buttons[button] = value
      return False
    else:
      return False
  return False
buttons[6] = digitalRead (6)
pinMode (4, 0)
buttons[4] = digitalRead (4)
pinMode (2, 0)
buttons[2] = digitalRead (2)
```

```
initCommunication()
def colour_rgb(r, g, b):
  r = round(min(100, max(0, r)) * 2.55)
  g = round(min(100, max(0, g)) * 2.55)
  b = round(min(100, max(0, b)) * 2.55)
  return '#%02x%02x%02x' % (r, g, b)
red = 0
green = 0
blue = 0
updated = False
step = 10
def loopCode():
  global red, step, updated, green, blue
  if buttonSwitched (6, 1):
    red = (red + step) % 255
    updated = True
  if buttonSwitched (4, 1):
    green = (green + step) % 255
    updated = True
  if buttonSwitched (2, 1):
    blue = (blue + step) % 255
    updated = True
  if updated:
    sendMessage('Board ID', 'color', \
    json.dumps(colour_rgb(red, green, blue)))
    updated = False
  Timer(0.1, loopCode).start()
loopCode()
```

The recipient wants the LED to be controlled only by a board that they choose. For that you must check that the ID of the board sending the messages is the right one (you will have to get in touch with your friends to set this up). The *If* block will run the program only if the messages are being

sent by the board with the right **ID** and by using the correct **label**.

The receiver: - Figure 77

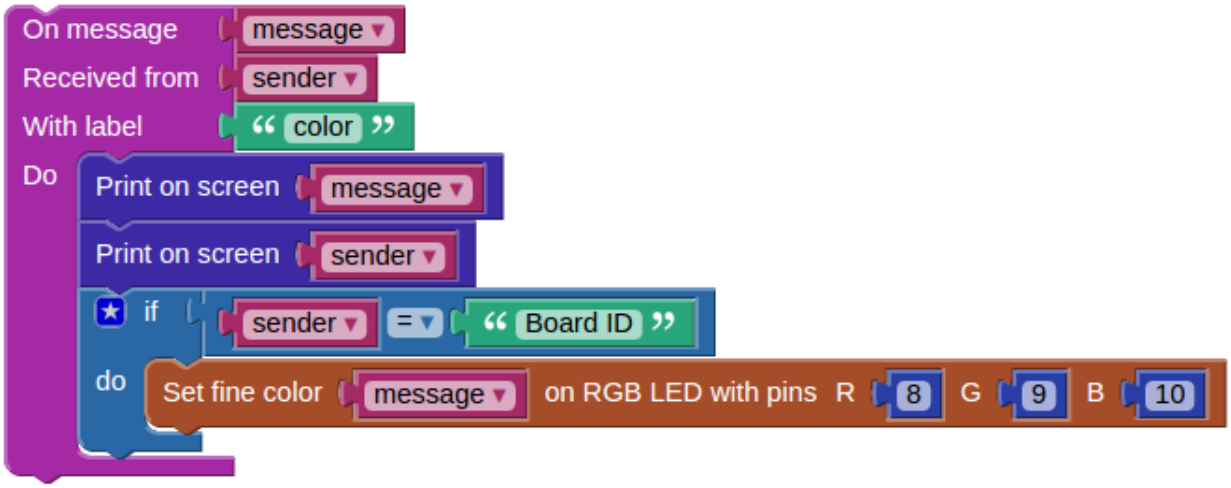

Figure 77: Social Lamp - receiver in Visual Programming

Python

```
from wyliodrin import *
import json
import struct
message = None
sender = None
def colorToRGB (color):
  return struct.unpack ('BBB', color[1:].decode('hex'))
def basic_color (color):
  value = 0
  if color>=128:
    value = 1
  else:
    value = 0
  return value
pinMode (9, 1)
pinMode (6, 1)
```

```
pinMode (3, 1)
def myFunction(__sender, __channel, __error, __message):
  global message, sender
  sender = __sender
  message = json.loads(__message)
  print(message)
  print(sender)
  if sender == 'BoardID':
    color = colorToRGB (message)
    analogWrite (9, color[0])
    analogWrite (6, color[1])
    analogWrite (3, color[2])
openConnection('color', myFunction)
```

You can watch an online tutorial on how to create this project [23].

[23] https://www.youtube.com/watch?v=49ScSHx0ZWA&index=10&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

Social Alarm Clock

You've just built a Social Lamp, so how about creating a Social Alarm Clock?

Like in the previous chapter, you will take small steps at first and afterwards you will find it easy to actually build the *device*.

First of all, you shall learn how to play sound using your Intel® Galileo, Wyliodrin and a USB sound card. Secondly, you will learn how to use LED displays. Then you will learn how to make calls using Twilio. Lastly, you will be able to bring all of this information together to create a social alarm clock that enables you and another person to get a wake up call with the same tune. Cool, right?

First Step

Play a Radio Station

What You Need

- Your Intel® Galileo
- One USB Audio Sound Card

- One Speaker

Building The System

Connect the audio sound card to the board. The board has a USB port that you can use. Then simply connect the speaker to the audio sound card.

The Code

Everytime you start a new application you must select the language. This time select the Music - Visual Programming one (Figure 78). This will create an application starting from an example for playing music.

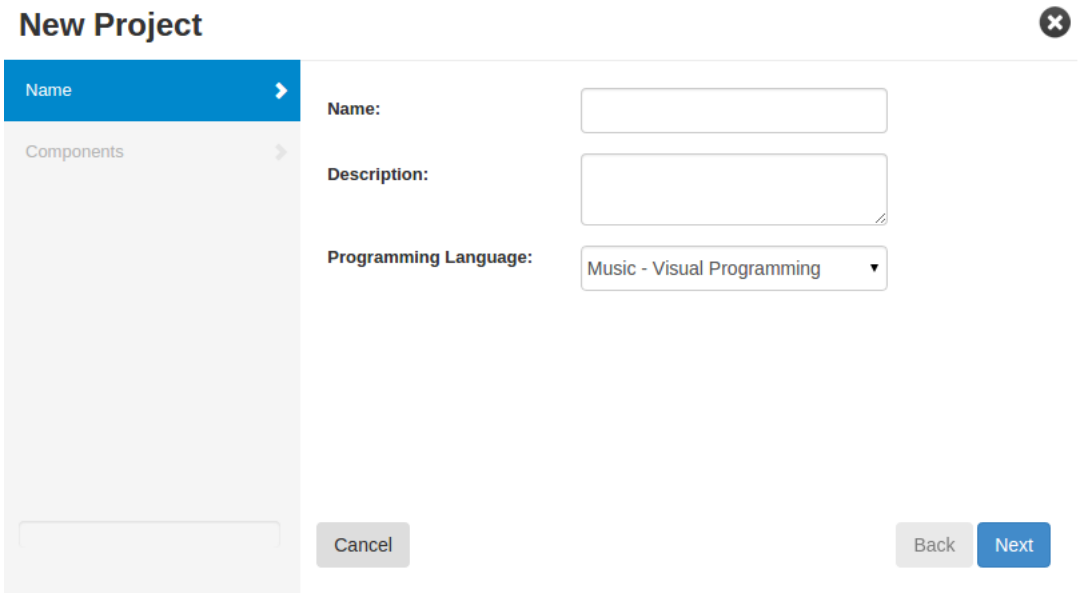

Figure 78: Creating a music application

Python

```
from pybass import *
from wyliodrin import *
from time import *
music = None
BASS_Init (-1, 44800, 0, 0, 0);
```

```
6  BASS_PluginLoad ('/usr/local/lib/libbassflac.so', 0)
7  BASS_PluginLoad ('/usr/local/lib/libbass_aac.so', 0)
8  music = BASS_StreamCreateURL ('http://80.86.106.35:9000/', \
9  0, 0, DOWNLOADPROC(0), 0)
10 BASS_ChannelPlay (music, False)
11 while BASS_ChannelIsActive (music) == BASS_ACTIVE_PLAYING:
12   sleep ((1000)/1000.0)
```

By default there is a selected radio station. You are free to change it.

Run the project.

Since we're all about social, Wyliodrin has a feature that allows you to share the projects.

Use the buttons on the left side to share the project. Click on any of them and Wyliodrin will ask if you allow the project to be made public. Answer yes. Press the buttons again and share the project with your friends. You can either send it via Facebook as a private message. Press again the button or simply copy the link from the browser and send it by email.

You can watch an online tutorial on how to create this project [24].

Second Step

Write On The LCD

What You Need

1. Your Intel® Galileo.

[24] https://www.youtube.com/watch?v=KU6ePaqpMxA&index=11&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

2. One BreadBoard.

3. One 16×2 LCD - GDM1602K.

4. One BreadBoard.

5. Sixteen Jumper wires.

6. One Potentiometer (Rotary angle sensor) - RM065.

Building The System

The LCD has a parallel interface, meaning that the board has to manage several interface pins at once to control the display.

The first pin is the *RS*(Register Select) that controls where exactly in the LCD's memory you are writing data to.

There is an *Enable* pin that enables writing to the registers.

The third one is the *R/W*(Read/Write) pin that selects from reading or writing mode.

The LCD also has *8 data* pins (D0 -D7). The states of these pins (high or low) are the bits that you're reading to a register when you read, or the values you are writing when you write.

There are also power supply pins (5V and GND) for powering the LCD, LED Backlight (Bklt- and Bklt+) pins that you can use to turn off and on the LED backlight and a display contrast pin (Vo) to control the display contrast.

What does 16×2 LCD mean? It means that the LCD has 16 columns and 2 rows.

Place the LCD on the breadboard and connect it to the Galileo as shown in Figure 79.

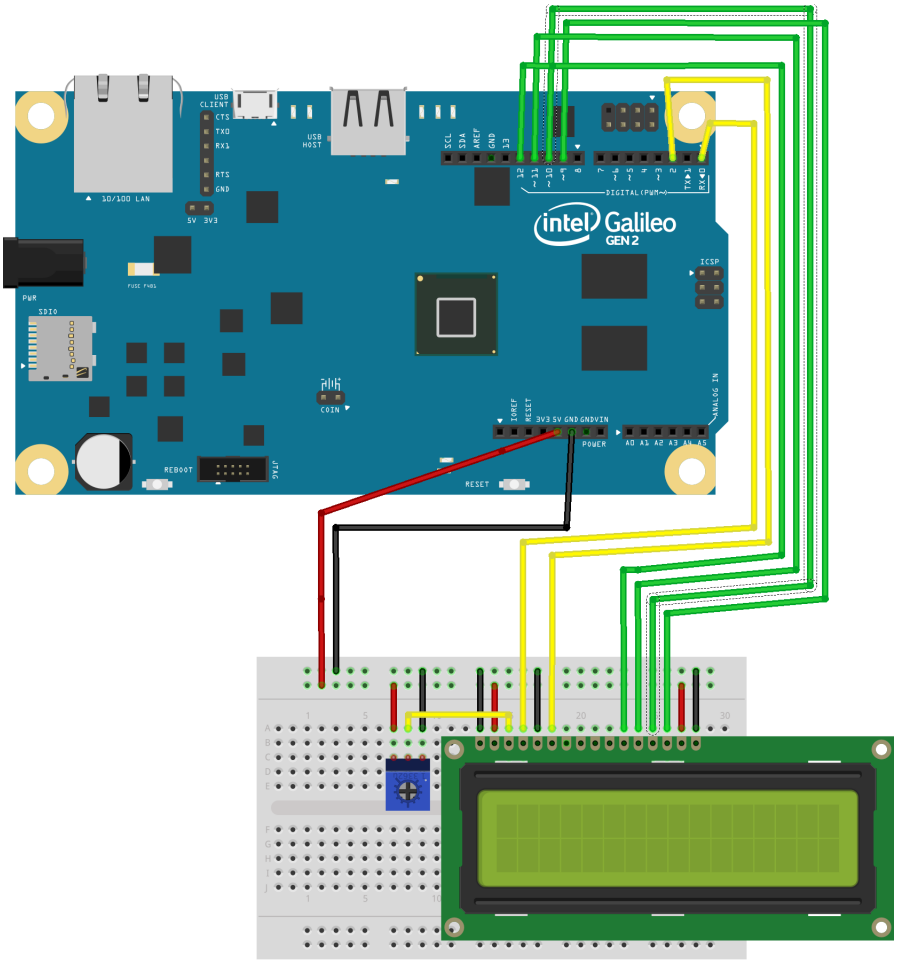

Figure 79: Connecting a LCD

You will use a potentiometer for adjusting the display contrast.

The Code

You need to init the LCD. To do this, select the *Peripherals/LCD/Init LCD* block. You need to complete the numbers of the pins for the LCD. They are the following:

- RS - pin 0;

- Enable - pin 2;
- Data 1 - pin 12;
- Data 2 - pin 11;
- Data 3 - pin 10;
- Data 4 - pin 9;

To write something, select the *Peripherals/LCD/Print on LCD []* block. Place it below the *Init* block and write something in it.

Start the project.

Once the project runs, you should see what you wrote on the LCD, however, the contrast might be to high and the text difficult to read. In order to adjust the contrast, just roll the potentiometer.

Your code should look like the one in Figure 80.

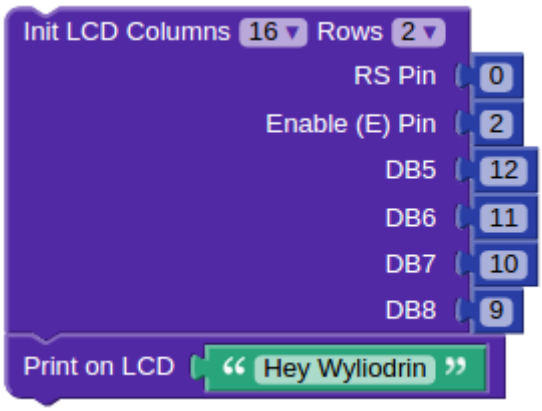

Figure 80: Print on LCD in Visual Programming

You can watch an online tutorial on how to create this project [25].

[25] https://www.youtube.com/watch?v=LZX7u9Mhftk&index=12&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr

Third Step

Make a Phone Call

What You Need

1. the setup from the previous step

Building The System

How can you do this?

Twilio is a cloud communications company that allows software developers to programmatically make and receive phone calls and send and receive text messages.

Attention! If you want to send a message to yourself, that will be free, but if you want to send a message or to call another person you must pay.

For more information please go to `https://www.twilio.com/pricing`.

The Code

The first thing that you have to do is to sign up on Twilio. Go to . Follow the steps.

Before using any other Twilio block, you need the *Setup* block. Without it, you cannot run your program. This block has two fields: an account and a token. You must fill in the AccountSID and AuthToken. For the last one, you have to click on the lock to see the token (Figure 81).

Figure 81: Auth Token - Twilio

You can find this information on the dashboard, after you log in. (Figure 82)

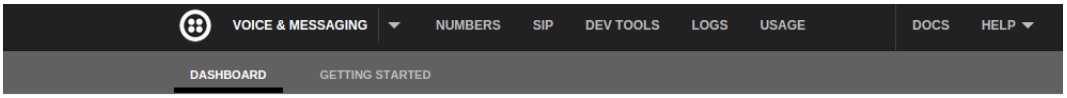

Figure 82: Dashboard - Twilio

You will need the *Make a call* block. For that, you have to know your Twilio phone number. You will find this on the same page under API Explorer. Make sure you use your country code (Figure 83).

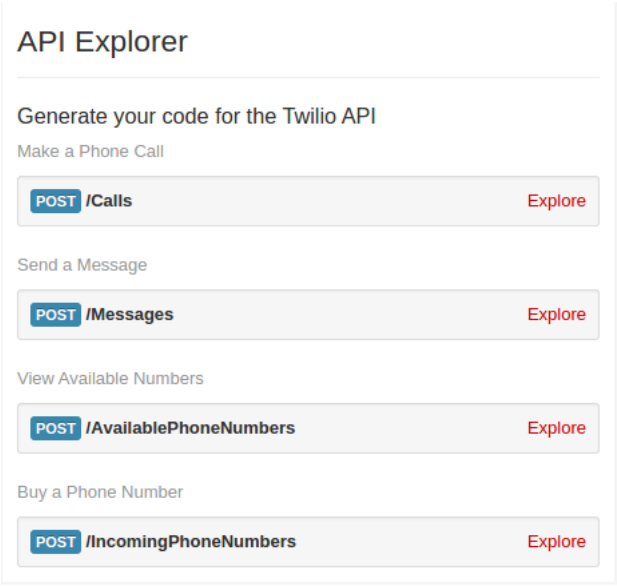

Figure 83: API Explorer - Twilio

Press the Explore button for the Calls. This is where you will find your Twilio phone number.

Use the two blocks you see in Figure 84.

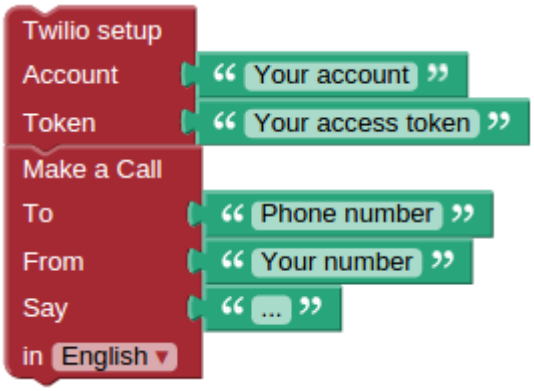

Figure 84: Make a call with Twilio in Visual Programming

Start the project. When the alarm starts, the number in the first box will receive a phone call from Twillio saying the text you entered. Once again, you can test this for free.

You can watch an online tutorial on how to create this project [26].

Social Alarm

Let's piece together everything now. You want a Social Alarm with an LCD that plays a radio stream. When your alarm starts, Twilio will call another person and they will hear the same song that you hear as alarm.

You can use Dropbox to play the song. Simply upload the song you want on your Dropbox account and click the share button from the online interface. Then replace the *www* in the link with *dl* and now you can stream it (Figure 85).

[26] `https://www.youtube.com/watch?v=oJZQ6-IJlaw&index=13&list=PLHih6DnKQaoaMTyPOzn6-MaNppf4VoaRr`

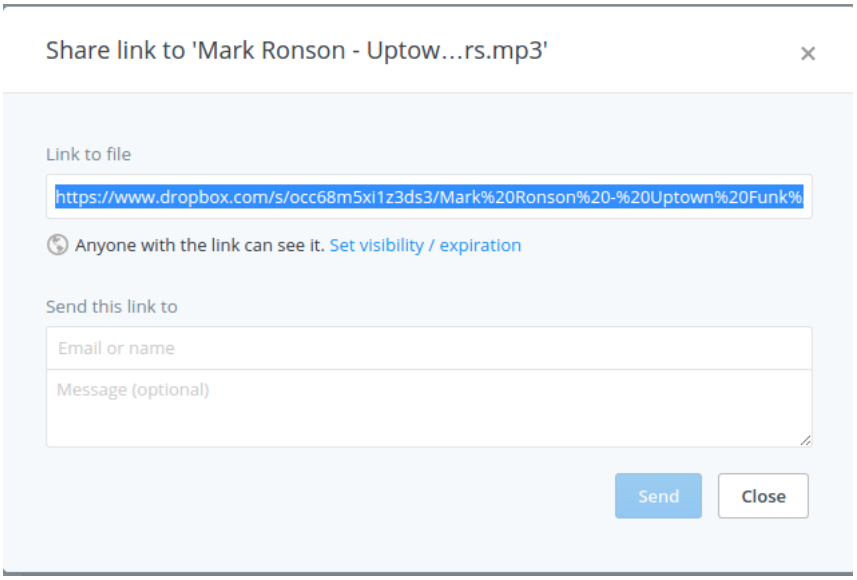

Figure 85: Songs - Dropbox

You will see an URL. Copy it. (Figure 86)

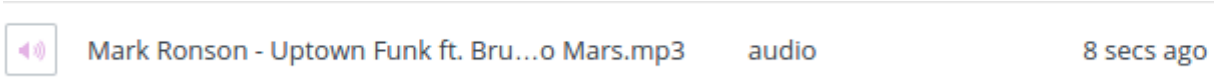

Figure 86: Song URL - Dropbox

You need to print the current time on the LCD, play a radio stream and set the alarm. Check out the code in Figure 87.

Firstly, you setup your twillio account and initialise the LCD. You should use the *Clear LCD* block to make sure that nothig is written on it. Further on, you need to load the audio stream for the alarm. Then you need two boolean variables, first stores whether you should wake up, to be more precise, whether it is time to wake up or not. The other one knows if you are up, that means if the alarm started to play or not.

Each second, the LCD gets updated with the current time. Then, you check if the current time is the same with the one you set the alarm to. In this case

the alarm is set for 3:11. If this is the time you set the *WeShouldWakeUp* variable to true. Now, in case the alarm is not playing, you start to play the audio stream and the other person gets called. The alarm will stop playing when the whole stream has been played.

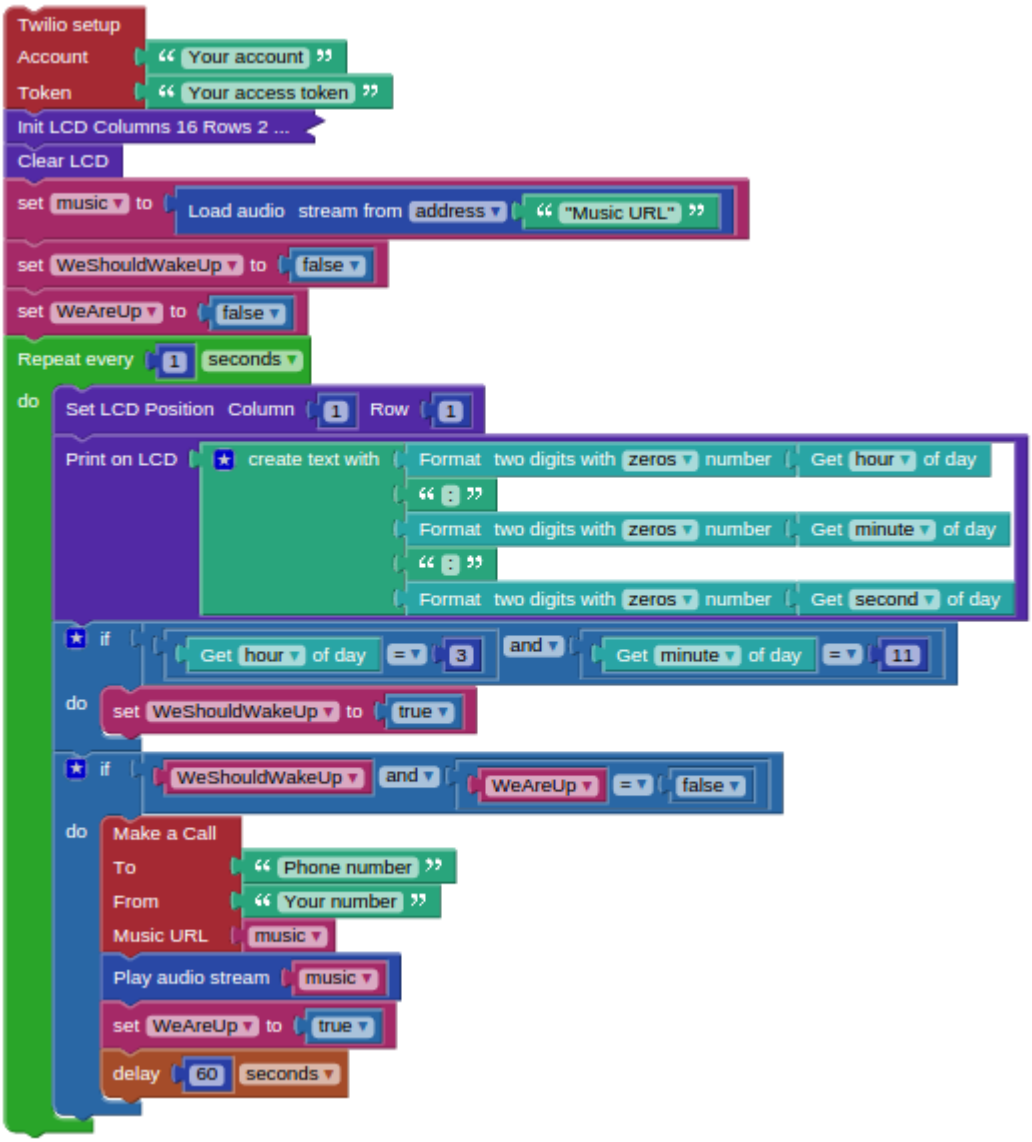

Figure 87: Social Alarm in Visual Programming

Python

```
1 import sys
```

```
try:
  from twilio.rest import TwilioRestClient
except:
  print("Please open the Shell and run 'social_install' script")
  sys.exit(1)
from wyliodrin import *
from pybass import *
from datetime import *
from urllib import *
from time import *
from threading import Timer

music = None
WeShouldWakeUp = None
WeAreUp = None
twilio_account = 'Your account'
twilio_token = 'Your access token'
lcd = lcdInit (2, 16, 4, 0, 2, 12, 11, 10, 9, 0, 0, 0, 0)
lcdHome (lcd)
BASS_Init (-1, 44800, 0, 0, 0);
BASS_PluginLoad ('/usr/local/lib/libbassflac.so', 0)
BASS_PluginLoad ('/usr/local/lib/libbass_aac.so', 0)

twilio_client = TwilioRestClient(twilio_account, twilio_token)
lcdClear (lcd)
music = BASS_StreamCreateURL ('"Music URL"', 0, 0, DOWNLOADPROC(0), 0)
WeShouldWakeUp = False
WeAreUp = False
def loopCode():
  global WeShouldWakeUp, WeAreUp, music
  lcdPosition (lcd, 1-1, 1-1)
  lcdPuts (lcd, str(''.join([str(temp_value) for temp_value in \
  [str(datetime.now().hour).zfill(2), ':', \
```

```
35    str(datetime.now().minute).zfill(2), ':', \
36    str(datetime.now().second).zfill(2)]])))
37    if (datetime.now().hour) == 3 and (datetime.now().minute) == 11:
38      WeShouldWakeUp = True
39    if WeShouldWakeUp and WeAreUp == False:
40      call =twilio_client.calls.create(to='Phone number',\
41      from_='Your number', \
42      url='http://twimlets.com/message?'+urlencode({'Message[0]':music}))
43      BASS_ChannelPlay (music, False)
44      WeAreUp = True
45      sleep (60)
46    Timer(1, loopCode).start()
47  loopCode()
```

Don't forget you can use a friend's number (if you are no longer a free user) and remember to set the hour that you wish for the alarm. Run the project!

Resistor Color Code

Every resistor has colored bands. Those bands and colors are not random at all, they help to identify the specifications of the resistor.

Here is how you can see the value of a resistor depending on its colored bands.

Every color represents a different number.

Here is the table of all the colors and numbers:

Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Grey	8
White	9

Counting from right to left, the second color is the multiplier. Digits of the first colors must be multiplied with the number of this color.

Black	1
Brown	10
Red	100
Orange	1000
Yellow	10000
Green	100000
Blue	1000000
Gold	0.1
Silver	0.01

And the last color is the tolerance. Tolerance is the precision of the resistor and it is a percentage.

Brown	1
Red	2
Gold	5
Silver	10
Nothing	20

There are a lot of programs that can calculate the value of a resistor but if you do not have access to the Internet and you need to know a certain value you can use the table from Figure 88.

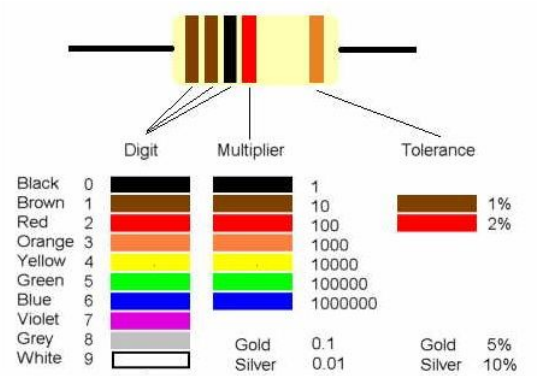

Figure 88: Resistor Color Code

Examples:

1.Yellow and green represent 4 and respectively 5. They represent the first and second digits, so you will have 45.

The third band is orange. As a multiplier, it is x 1000, so you will calculate 45 x 1000 thus, the resistance is 45,000 Ω.

The forth band, silver, represents the tolerance, so the final expression of the resistance is 45,000 +/- 10Ω (Figure 89)

Figure 89: Resistor Color Code

2. A resistor colored Orange-Orange-Black-Brown-Violet would be 3.3 kΩ with a tolerance of +/- 0.1(Figure 90)

Figure 90: Example 2

Visual Programming

A visual program language is a programming language that lets users create programs by using graphic elements.

This chapter presents the specific blocks that were used in this book.

Program

Functions

Functions

Creates a function.

You can have functions that return something (Figure 91) of functions that do not return anything (Figure 92).

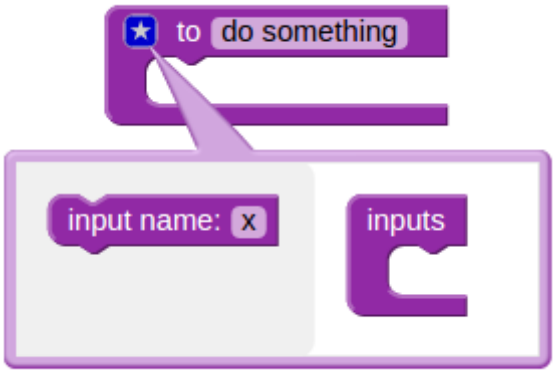

Figure 91: Functions that do not return Block

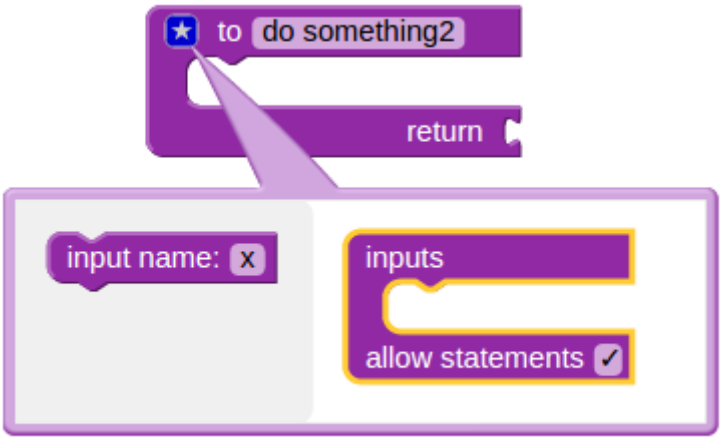

Figure 92: Functions that return Block

Loops

Repeat while [](Figure 93):

Figure 93: Repeat While Block

This block will repeat your block as long as your condition is true.

Count with (Figure 94):

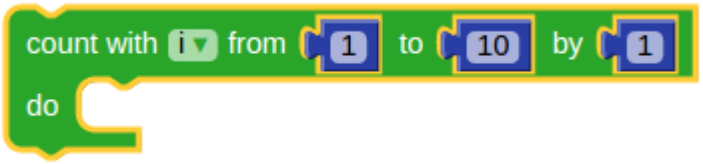

Figure 94: Count Block

Advances a variable from the first value to the second value by the third value.

Repeat every [] (Figure 95):

Figure 95: Repeat every [] Block

It will repeat the statements every set time interval.

Repeat [] times (Figure 96):

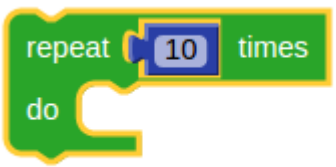

Figure 96: Repeat [] times Block

It runs the code in its body the specified number of times.

Logic

True (Figure 97):

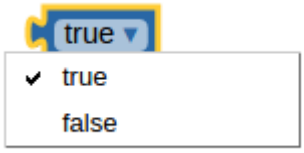

Figure 97: Boolean Block

Can be used as a boolean value: true or false.

Comparisons (Figure 98):

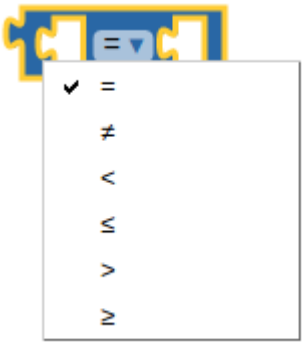

Figure 98: Comparison Block

Takes two inputs and returns true or false depending on how the inputs compare with each other.

Logical Operators (Figure 99):

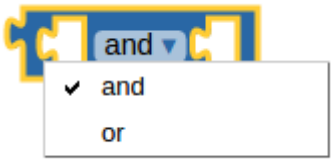

Figure 99: Logical Operators Block

The *and* block will return true only if both of its two inputs are also true.

The *or* block will return true if either of its two inputs are true.

If/else (Figure 100):

Figure 100: If/Else Block

If the expression or variable being evaluated is true, then do the first block of statements. If it's false do the second block of statements.

Variables

Set item to [] (Figure 101):

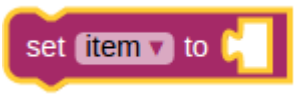

Figure 101: Set Block

Assigns a value to a variable, creating the variable if it doesn't exist.

Item (Figure 102):

Figure 102: Item Block

This block provides the value stored in a variable.

Lists

Create list (Figure 103):

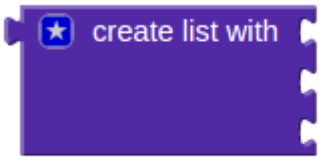

Figure 103: Create list Block

Creates a list of items.

Numbers And Maths

Truncate (Figure 104):

Figure 104: Truncate Block

Round a number to the nearest integer.

Map (Figure 105):

Figure 105: Map Block

It will map the value.

Number (Figure 106):

Figure 106: Number Block

Reminder of (Figure 107):

Figure 107: Reminder Block

The Remainder or Modulus Operator returns the remainder of a division between two numbers.

Change (Figure 108):

Figure 108: Change Block

Increments a particular variable by a set number.

Text

Text (Figure 109):

Figure 109: Text Block

Create text (Figure 110):

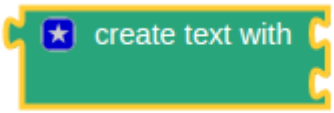

Figure 110: Create text Block

It creates a piece of text by joining together any number of items.

Screen And Keyboard

Print on screen (Figure 111):

Figure 111: Print on Screen Block

Print a text on the screen and put the cursor on a new line.

Timing

Delay [] (Figure 112) :

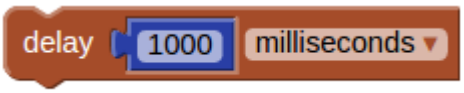

Figure 112: Delay Block

This will delay the program execution for a period of seconds, milliseconds or microseconds.

Date And Time

Get hour (Figure 113):

Figure 113: Get hour Block

Gives either the current hour/minute/second.

Social

Twilio

Twilio Setup (Figure 114):

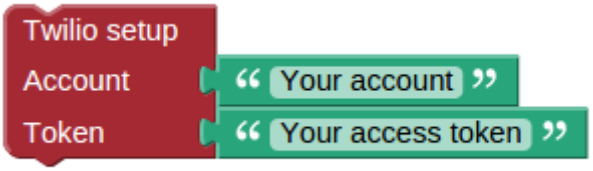

Figure 114: Twilio Setup Block

Make a call (Figure 115):

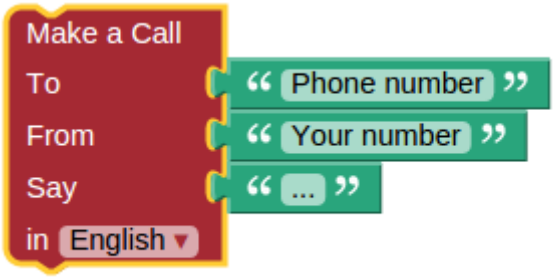

Figure 115: Make a Call Block

Make a call with music (Figure 116):

Figure 116: Make a Call with music Block

Board Communication

Send message (Figure 117):

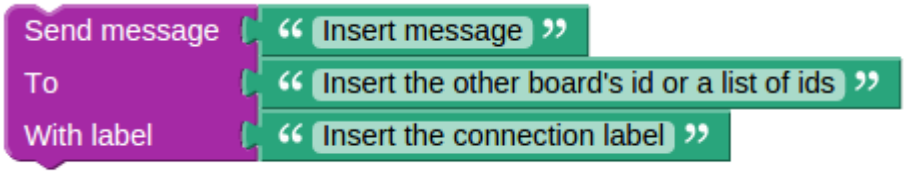

Figure 117: Send message Block

Sends a message to another board using a particular label.

On message (Figure 118):

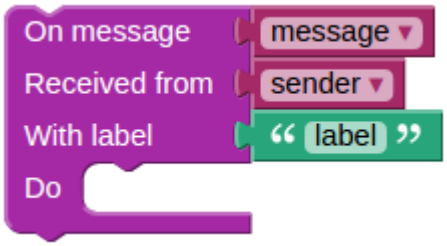

Figure 118: On message Block

Tests that a message variable is received from a particular other board using the correct label and enables further actions to be carried out.

Pin Access

analogWrite (Figure 119):

Figure 119: analogWrite Block

Sets a value between 0 and 255 on the pin.

analogRead (Figure 120):

Figure 120: analogRead Block

Reads a value between 0 and 1023 from the pin.

digitalRead (Figure 121):

Figure 121: digitalRead Block

Reads a value from one of the digital pins and returns 0 or 1.

digitalWrite (Figure 122):

Figure 122: digitalRead Block

Sets value 0 or 1 on the pin.

Peripherals

Pins

Set pin [] to LOW/HIGH (Figure 123):

Figure 123: Set pin Block

Set a digital value on a pin. You have to add the pin number and the value you want: *HIGH* means 1 and *LOW* means 0.

Shift out (Figure 124):

Figure 124: Shift Out Block

Writes a value serially on a pin. This will send out a value bit by bit to the data pin and generate a clock signal on the clock pin. It will start with the LSB (Least Significant Bit). Usually this function is used with shift registers.

LED

Set On/Off LED on pin [] (Figure 125):

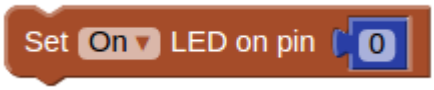

Figure 125: Set On/Off LED Block

You have to add the pin number and the value you want.

This block turns the LED connected to the pin number you added on or off .

Set fine color on RGB LCD (Figure 126):

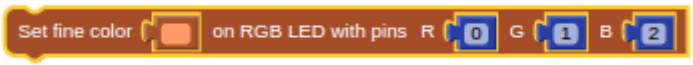

Figure 126: Set Fine Color Block

Sets the LED with the specified pins for RGB at the selected color. You can select any color. This block can be used only if the pins used are PWMs.

7 Segment Display

Set 7-seg (Figure 127):

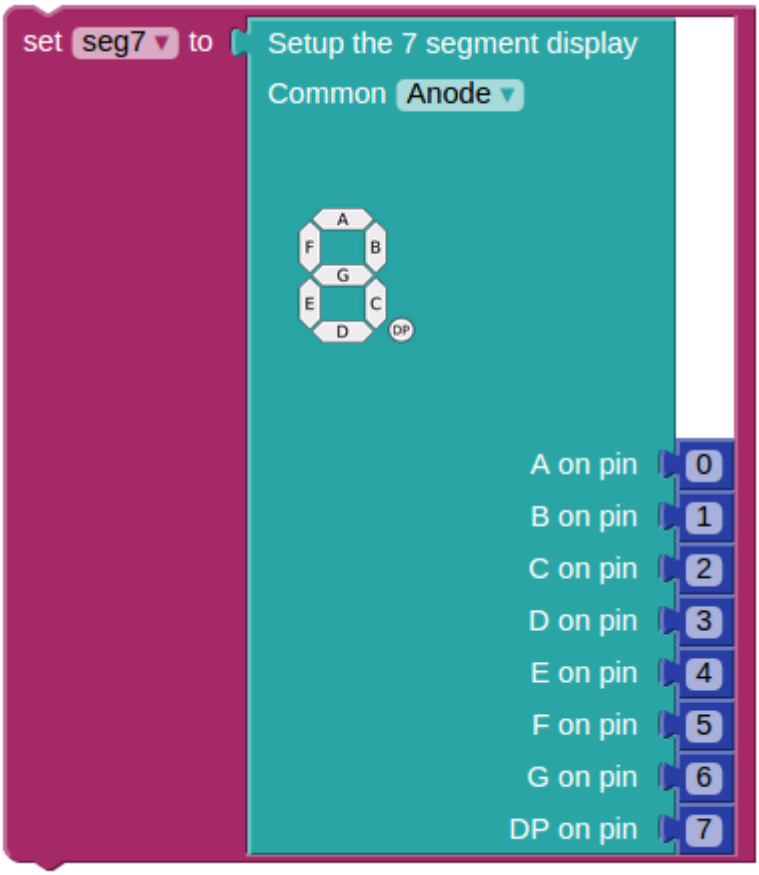

Figure 127: Set 7-segment Block

Initialize the 7-segment display.

Display on 7-seg (Figure 128):

Figure 128: Display on 7-seg Block

Displays a number on the 7-segment display.

LCD

Init LCD (Figure 129):

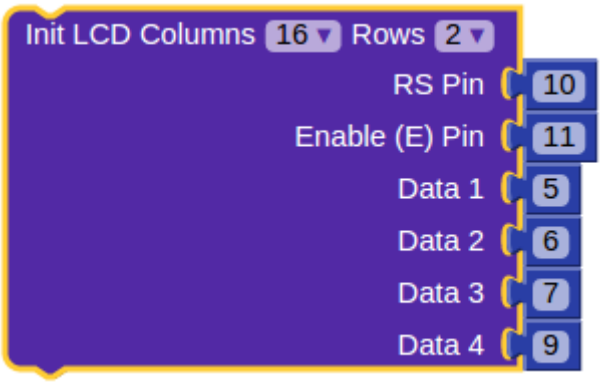

Figure 129: Init LCD Block

Initializes the LCD with 1, 2 or 4 rows and 16 or 20 columns.

Print on LCD (Figure 130):

Figure 130: Print on LCD Block

Prints the text you typed on the LCD.

Sensors

Buttons

Button on pin [] switched to (Figure 131):

Figure 131: Button Block

Shows the change of state of the respective button to Pressed/Released.

Internet

Sevices

Weather

Get temperature from [] (Figure 132):

Figure 132: Get Temperature Block

It returns the temperature from a city.

Signals

Send signal (Figure 133):

Figure 133: Send Signal Block

It will send a signal with a value to a graph on the dashboard.

Multimedia

Load audio stream (Figure 134):

Figure 134: Load audio Block

Returns an Audio Stream variable. Most of the other functions request this type of variable as a parameter. The Audio Stream can be created from a file, in this case the second parameter will be the file's name, or from a URL and the second parameter will be the address.

Play audio stream (Figure 135):

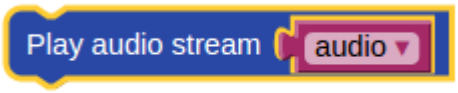

Figure 135: Play audio Block

Plays an Audio Stream.

Audio stream is playing (Figure 136):

Figure 136: Audio stream is playing Block

Verifies whether the player is playing.

Useful Functions

1. *pinMode()*: Configures the specified pin to behave as either an input or an output.

   - Syntax: pinMode(pin, mode)
   - Pin: the number of the pin whose mode you wish to set
   - Mode: *INPUT* or *OUTPUT*
   - Comment:

   It is used only for digital pins.

2. *digitalWrite()*: Writes a HIGH or a LOW value to a digital pin.

   - Syntax: digitalWrite(pin, mode).
   - Pin: the number of the pin whose mode you wish to set.
   - Mode: *HIGH* or *LOW*.
   - Comment:

   It is used only for digital pins.

3. *digitalRead()*: Reads the value from a specified digital pin, either HIGH or LOW.

- Syntax: digitalRead(pin).
- Pin: the number of the pin whose mode you wish to set.
- Comment:

It is used only for digital pins.

4. *analogWrite()*: Writes an analog value (*0-255*) to a pin.
   - Syntax: analogWrite(pin, mode).
   - Pin: the pin to write to.
   - Mode: between 0 (always off) and 255 (always on).
   - Comment:

   It is only used only for PWM pins (more details will be presented further on).

5. *analogRead()*: Reads a value from the specified analog pin.
   - Syntax: analogRead(pin).
   - Pin: the number of the analog pin to read from (you will receive a value from 0 to 1023).
   - Comment:

   It is only used for analog pins.

6. *map()*: This function will map the lowest/highest values to a new set of lowest/highest values.
   - Syntax: map(value, fromLow, fromHigh, toLow, toHigh).
   - Return: The mapped value.

7. *delay()*: It pauses the execution of the program for a period of time.
    - Syntax: delay(ms); ms = the number of milliseconds to pause.
8. *shiftOut()*: Shifts out a byte of data one bit at a time.
    - Syntax: shiftOut(dataPin, clockPin, bitOrder, value).
9. *shiftIn()*: Shifts in a byte of data one bit at a time.
    - Syntax: shiftIn(dataPin, clockPin, bitOrder).
    - Return: the value read.
10. *print()*: Prints text on the LCD
    - Syntax: lcd.print("text")