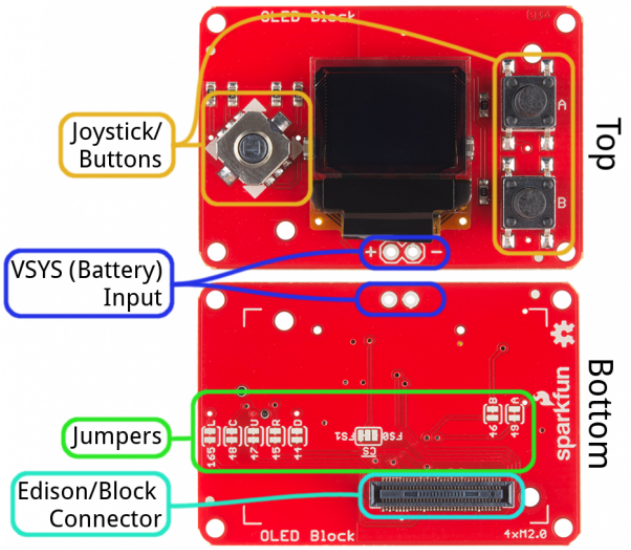
Using OLED Block for Intel® Edison

1 Board Overview

Before we jump into using the OLED Block, here's a quick overview of what features and components the board includes:



In total, the OLED block has seven button/joystick inputs, here's how they're mapped to the Edison's GPIO pins:

Button	Edison GPIO Pin
Up	47
Down	44
Left	165
Right	45
Select	48
A	49
B	46

Note 1: "Up" is towards the top of the screen (the side without the black ribbon connector).

Note 2: The "Select" button refers to a downward press on the joystick.

The bottom of the OLED block is filled with jumpers, which allow you to tailor the block to fit your project. Seven of the eight jumpers allow you to disconnect any of the joystick or button inputs from the Edison, in case you need those GPIO for another purpose. Keep in mind, if you cut any of the jumpers, that button will serve no purpose unless you wire it to another pin or re-solder the jumper.

The "CS" jumper allows you to flip the OLED's chip-select pin from "FS0" (default) to "FS1". If you switch this pin, you'll need to modify the code accordingly.

2 Using the OLED Block

To use the OLED Block, attach it to either an Edison or another SparkFun Block. This board, unlike most Edison blocks is only single-sided, so it must be at the top of your stack (you don't want to cover that beautiful display!).

The OLED block can supply power to the Edison, using the battery supply input, but we recommend using it in conjunction with a development block like the Console Block or Base Block.

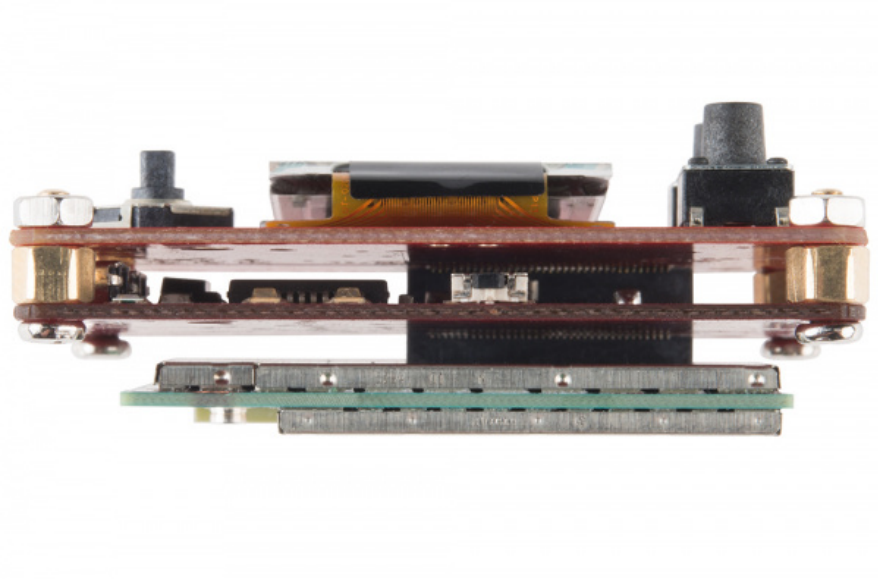


Figure 1: OLED Block in a stack with the Edison and a Console Block

Blocks can be stacked without hardware, but it leaves the expansion connectors unprotected from mechanical stress. We recommend adding a few screws from our Hardware Pack between the OLED and the next block.

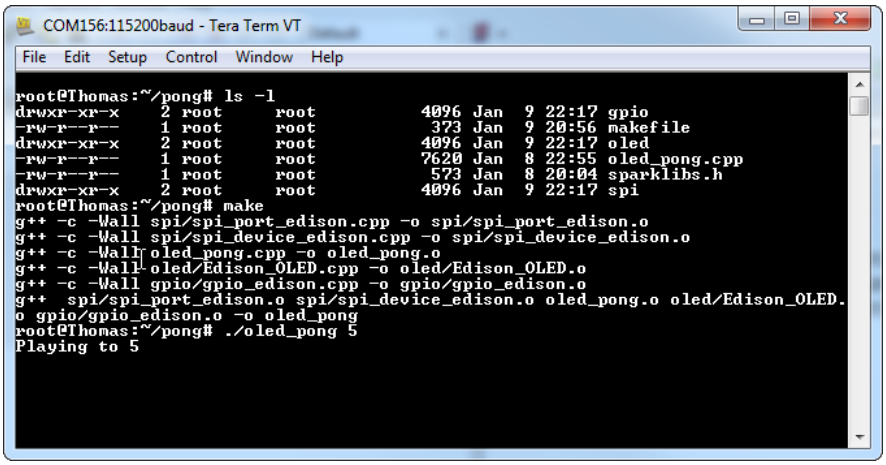
3 Programming for the OLED Block

It'll take a bit of coding to get your display up-and-running. The Edison has all sorts of programming-language support, but we've decided to stick with C++ to control the display.

We've written a simple library to help get you started. You can download the latest version from our GitHub repository [1].

The example code includes a simple makefile that should compile directly on the Edison. But first, you need to get these files onto the Edison. There are a few ways to do that. For example, you could get the Edison connected to a WiFi network and SSH it over – check out the SSH how-to section of our Getting Started with Edison tutorial for help with that. Even better than that, you could follow along with our Programming the Intel® Edison: Beyond the Arduino IDE and set up Eclipse to remotely upload the code – even remotely compile it on your development computer.

Once you've loaded the code, navigate to the "pong" folder and type **make**. The dependencies (spi, gpio, and oled libraries) will build, then the main example code ("oled_pong.cpp") will build to create an "oled_pong" executable.



[1] https://github.com/sparkfun/Edison_OLED_Block/tree/master/Firmware/pong

After the code has successfully been built, type ./oled_pong to play. This should trigger the OLED to light up and start a game of pong. Have fun!

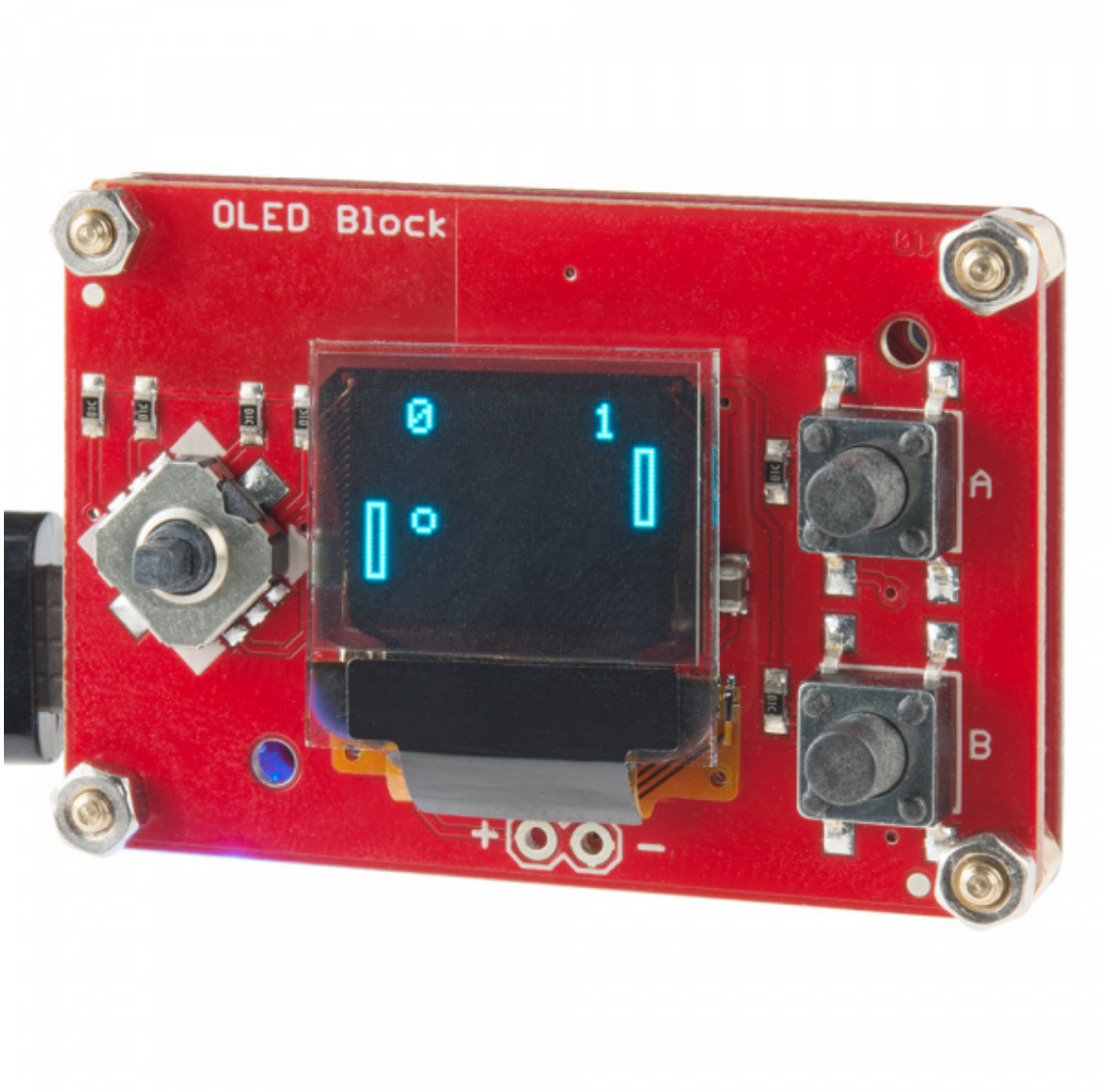


Figure 2: Edison Pong in action

4 Using OLED Library

The OLED library, included in the download above, allows you to draw anything from pixels and lines to shapes and text on the little display.

The "pong" example code should serve as an excellent teaching tool. To begin, create an instance of the **edOLED** class, then initialize the display like this:

```
edOLED oled;
...
oled.begin()
oled.clear(ALL);
oled.display();
```

Then, continue to use the **oled** object to draw pixels, lines, and other shapes like this:

```
// Draw a pixel at x,y
oled.pixel(x, y);
// Draw a line from x0,y0 to x1,y1
oled.line(x0, y0, x1, y1);
// Draw a rectangle, begnning at x,y with a set width and height
oled.rect(x, y, width, height);
// Draw a circle, centered at x,y with a set radius
oled.circle(x, y, radius);
```

Or you can draw text on the screen with function calls like:

```
// Set the text cursor to x,y
oled.setCursor(x, y);
// Set the font to one of four types.
oled.setFontType([0:3]);
// Draw a character
oled.print(char);
// Draw an array of characters (string)
oled.write(char *);
// Draw an integer value
oled.write(int);
```

Don't forget: the OLED's display will not update until you call **oled.display()**.

5 Using the GPIO Library

The "gpio" library, also included with the example code, can be used to listen for button presses. Again, the pong example should serve as a good place to start learning how to use the library. First, you'll need to initialize the pins:

```
// UP button is tied to GPIO 47
gpio BUTTON_UP(47, INPUT);
// Down is tied to GPIO 44
gpio BUTTON_DOWN(44, INPUT);
// Left is tied to GPIO 165
gpio BUTTON_LEFT(165, INPUT);
// Right is tied to GPIO 45
gpio BUTTON_RIGHT(45, INPUT);
// Select (pushing the joystick down) is GPIO 48
gpio BUTTON_SELECT(48, INPUT);
// Button A is GPIO 49
gpio BUTTON_A(49, INPUT);
// Button B is GPiO 46
gpio BUTTON_B(46, INPUT);
```

Then you can use the **readPin()** function of the **gpio** class to see if they're HIGH or LOW. The buttons are all pulled high, so they'll read as LOW if they're pressed down. The following example might help you.

```
if (BUTTON_UP.pinRead() == LOW)
{
    printf("You're pressing down");
}
if (BUTTON_A.pinRead() == LOW)
{
    printf("You're pressing the A button");
}
```

6 Exercise

Using OLED and GPIO Library for Intel Edison to design and implement the snack game. You might also propose another application/demonstration using OLED Block.