
Badge Documentation

Release 1

Radomir Dopieralski

Aug 19, 2018

Contents:

CHAPTER 1

Hardware Overview

The badge consists of several main components:

1.1 Microcontroller

The microcontroller controlling everything is an Atmel SAMD21, an ARM Cortex-M0 microcontroller running at 48MHz. It has native USB support, which lets us expose the filesystem on the device as a USB disk and the REPL console as a serial device. It's running a custom version of CircuitPython, an implementation of the Python language for microcontrollers based on MicroPython.

All of the GPIO pins are exposed in the `board` module, which lets us use them from CircuitPython.

1.2 Buttons

The device has eight buttons, arranged into two crosses, that can be used to control the code running on it. The buttons are connected to GPIO pins on one end, and to GND on the other, and they are scanned in the background by the `gamepad` module. The state of the buttons can be accessed with the `badge.keys()` function.

1.3 Display

There is a display of 14×11 red pixels on the device, controlled by a IS31FL3733 chip connected over an I^2C bus to the `board.SDA0` and `board.SCL0` pins. It can be accessed with the `badge.show()` function and its brightness can be adjusted with the `badge.brightness()` function. It's capable of displaying 256 shades of red, however, since most of those shades are very similar, the library selects only 16 distinct shades that you can use.

1.4 Accelerometer

There is a LIS3DH accelerometer in the middle of the board, under the display, which can be used to read the position and movement of the device. It is also connected over an I^2C protocol to the `board.SDA0` and `board.SCL0` pins. It can be accessed with the `badge.accel()` function.

1.5 Radio

There is an NRF24L01+ radio module on board, connected to the `board.RADIO*` pins. The connections are as follows:

- `RADIO0` — `CE`
- `RADIO1` — `CSN`
- `RADIO3` — `SCK`
- `RADIO4` — `MOSI`
- `RADIO5` — `MISO`

There are currently no built-in functions or libraries for handling the communication with this module. And external library needs to be used.

1.6 Power

The device is powered with three 1.5V AAA alkaline batteries or the USB port. The source of power is switched with the power switch. In the "ON" position, the power comes from the batteries, while in the "OFF" position, it's powered from the USB port (if it's connected).

CHAPTER 2

Library Reference

`badge.`**`init`**`()`
Initialize the module.

This function switches the display on and performs some basic setup.

`badge.`**`brightness`**`(`*level*`)`
Set the brightness of the display, from 0 to 15.

`badge.`**`show`**`(`*pix*`)`
Show the provided image on the display, starting at the top left corner. You will want to call this once for every frame.

`badge.`**`keys`**`()`
Return a number telling which keys have been pressed since the last check. The number can then be filtered with the `&` operator and the `K_X`, `K_DOWN`, `K_LEFT`, `K_RIGHT`, `K_UP`, `K_O`, `K_S`, and `K_Z` constants to see whether any of the keys was pressed.

`badge.`**`tick`**`(`*delay*`)`
Wait until `delay` seconds have passed since the last call to this function. You can call it every frame to ensure a constant frame rate.

`class` `badge.`**`Pix`**`(`*width=14*, *height=11*, *buffer=None*`)`
Pix represents a drawing surface, `width` pixels wide and `height` pixels high.

If no `buffer` is specified for storing the data, a suitable one will be automatically created.

`classmethod from_string``(`*cls*, *string*`)`
Creates a new Pix and initialzes its contents from a provided string. Each line corresponds to one line of the image, and each character corresponds to a pixel on that line. The brightness of a pixel is encoded as a single hexadecimal digit, *0* being black and *F* being white. Characters that are not hex digits will cause an error.

`classmethod from_text``(`*cls*, *text*, *color=None*, *background=0*, *colors=None*`)`
Creates a new Pix and renders the specified text on it. It is exactly the size needed to fit the specified text. Newlines and other control characters are rendered as spaces.

If `color` is not specified, it will use bright text on dark background by default. Otherwise it will use the specified color, with `background` color as the background.

Alternatively, `colors` may be specified as a 4-tuple of colors, and then the `color` and `background` arguments are ignored, and the four specified colors are used for rendering the text.

`pixel`(*self*, *x*, *y*, *color=None*)

If `color` is specified, sets the pixel at location `x`, `y` to that color. If not, returns the color of the pixel at that location.

If the location is out of bounds of the drawing surface, returns 0.

`box`(*self*, *color*, *x=0*, *y=0*, *width=self.width*, *height=self.height*)

Draws a filled box with the specified `color` with its top left corner at the specified location and of the specified size. If no location and size are specified, fills the whole drawing surface.

`blit`(*self*, *source*, *dx=0*, *dy=0*, *x=0*, *y=0*, *width=None*, *height=None*, *key=None*)

Copied the `source` drawing surface onto this surface at location specified with `dx` and `dy`.

If `x`, `y`, `widht` and `height` are specified, only copies that fragment of the `source` image, otherwise copies it whole.

If `key` color is specified, that color is considered transparent on the source image, and is not copied onto this drawing surface.

`badge.`**`accel`**`()`

Return a triple `(x, y, z)` of acceleration values reported by the on-board accelerometer.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index