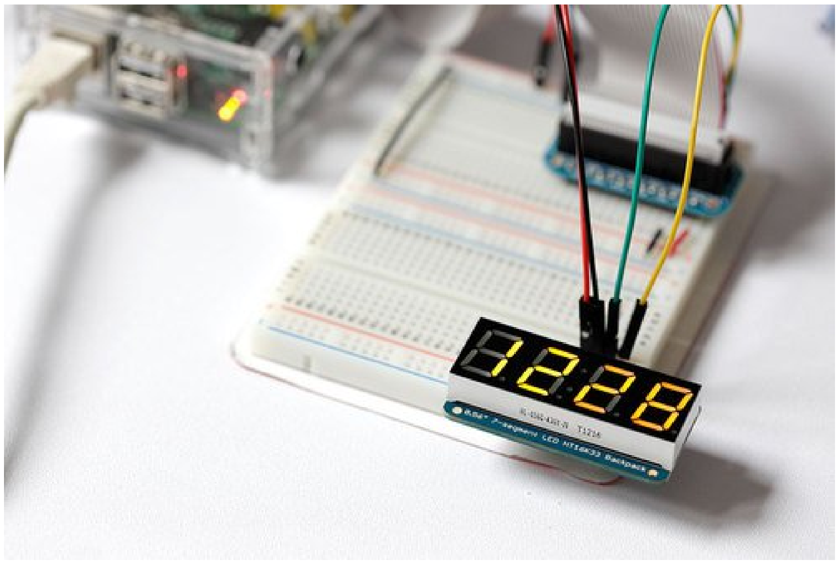
Matrix and 7-Segment LED Backpack with the Raspberry Pi

Created by Kevin Townsend

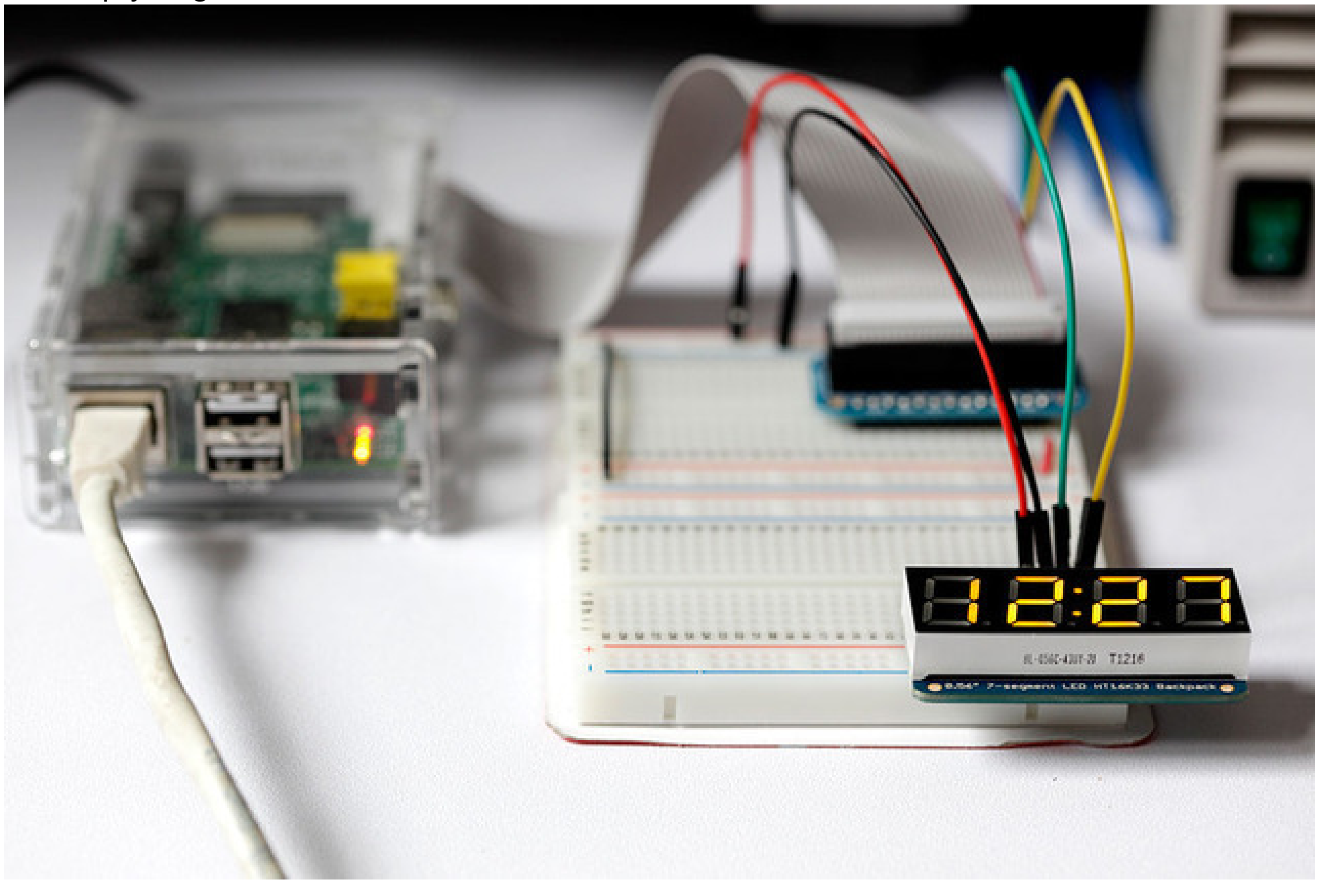


Last updated on 2016-11-03 10:11:42 AM UTC

Guide Contents

Overview

While we already have a great tutorial (http://adafru.it/aOV) and Arduino library (http://adafru.it/aLI) for our handy and easy to use LED backpacks, if you're wondering how you can make use of backpack-enabled LED displays on the Pi, this guide will help you get started!



What You'll Need

1. A Raspberry Pi
2. A Pi Cobbler (http://adafru.it/914) to make it easy connecting things up
3. One of our many 8x8 (http://adafru.it/aLG) or 4-Digit 7-Segment (http://adafru.it/aLH) backpack-enabled displays

Related Information

If you're looking for help on how to assemble one of our backpack-enabled LED displays, have a look at our earlier guide, Adafruit LED Backpacks (http://adafru.it/aOV). It contains a lot of complimentary information on these displays, including step by step soldering and assembly instructions.

Configuring your Pi for I2C

The Holtek HT16K33 (http://adafru.it/aMy) chip used in all of our backpacks communicates using the common I2C bus. Linux and the Pi both have native support for I2C, but you'll need to run through a couple quick steps from the console before you can use it in Python.

To learn more about how to setup I2C with Raspbian, take a minor diversion to this Adafruit Tutorial: http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

Then come back here to continue!

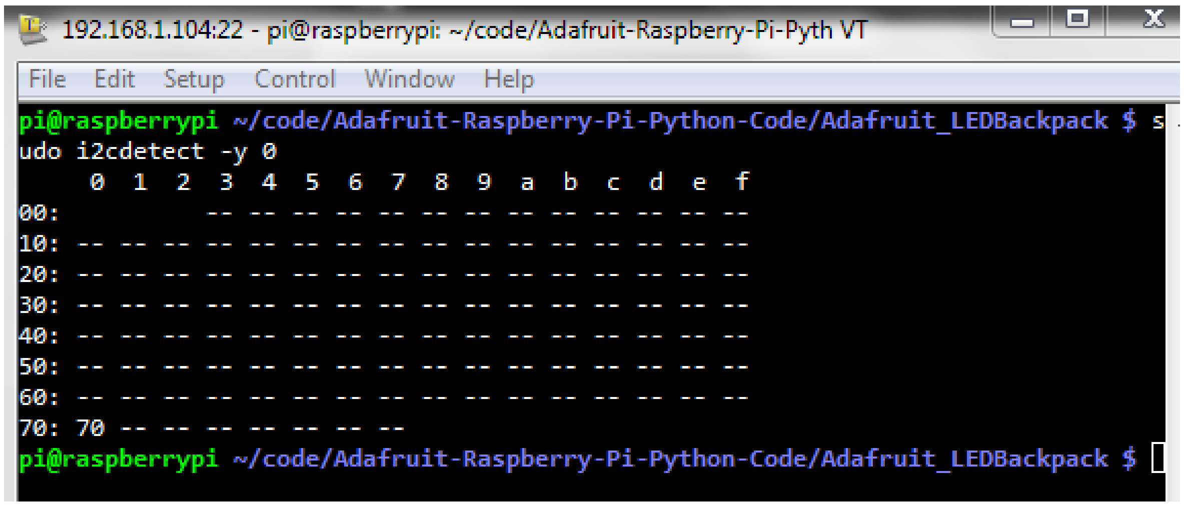
```
sudo apt-get install -y i2c-tools python-smbus
```

i2c-tools isn't strictly required, but it's a useful package since you can use it to scan for any I2C or SMBus devices connected to your board. If you know something is connected, but you don't know it's 7-bit I2C address, this library has a great little tool to help you find it:

```
sudo i2cdetect -y 1
```

This will search I2C for all address, and if a Holtek HT16K33 breakout is properly connected and it's set to it's default address it should show up as follows (the exact address will vary depending on whether or not you have any of the address solder jumpers set).

If you happen to have one of the original first batch of Raspberry Pis, you will need to change the 1 to a 0 in the command above.


192.168.1.104:22 - pi@raspberrypi: ~/code/Adafruit-Raspberry-Pi-Pyth VT
File Edit Setup Control Window Help

```
pi@raspberrypi ~/code/Adafruit-Raspberry-Pi-Python-Code/Adafruit_LEDBackpack $ s
udo i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
pi@raspberrypi ~/code/Adafruit-Raspberry-Pi-Python-Code/Adafruit_LEDBackpack $
```

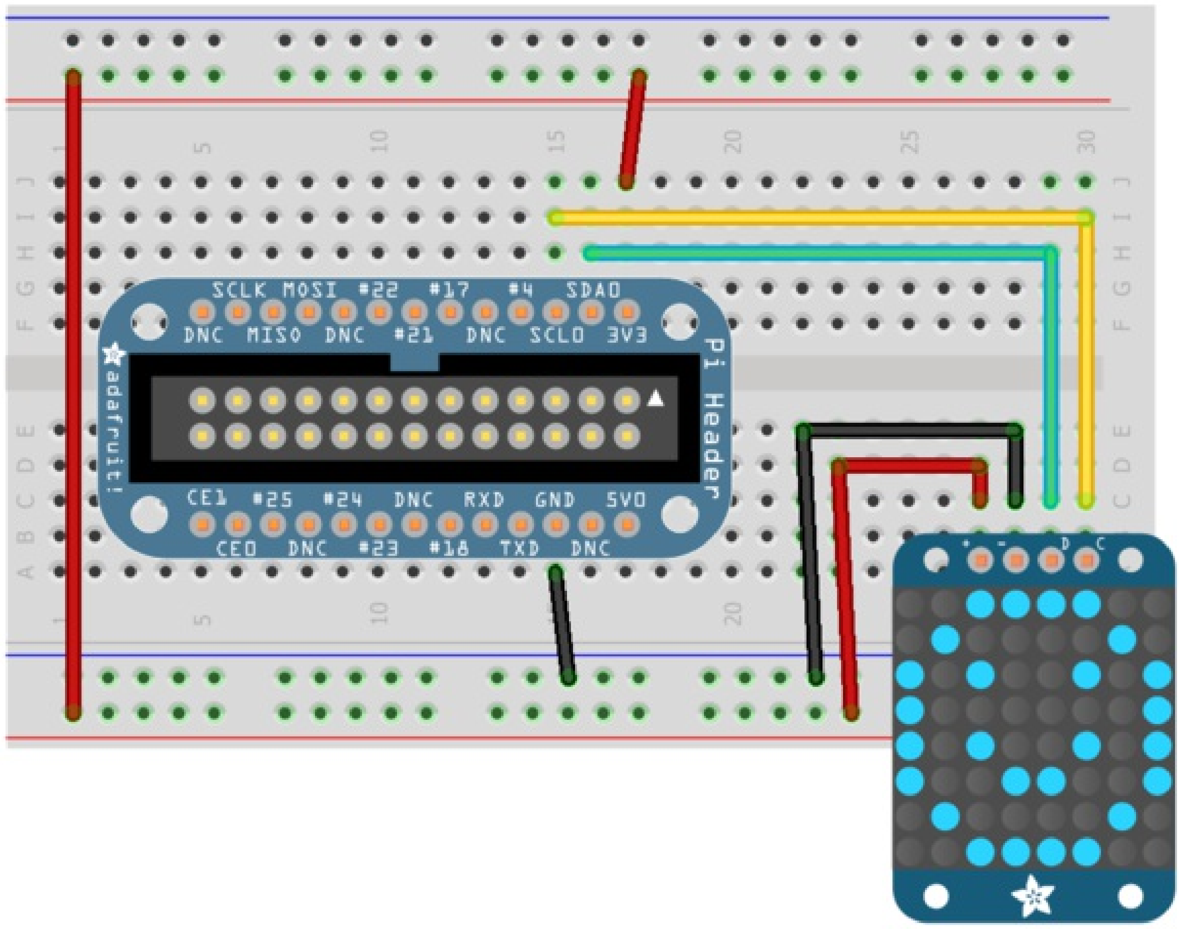
Once both of these packages have been installed, you have everything you need to get started accessing I2C and SMBus devices in Python.

Hooking Everything Up

The LED backpack displays are wonderfully easy to connect to the Pi. Both the 8x8 pixel and 4-character 7-segment displays are connected the same way, with two digital pins for I2C (SDA and SCL) and two power pins (VCC and GND), as follows.

If you're using a red, yellow or green backpack, you can power the backpack from 3.3V which will keep the I2C levels at 3.3V. If you have a blue or white backpack, the LEDs will be dim if powered from 3.3V.

If you want them a little brighter, connect the VCC pin of the backpack to 5V. There are 10K pullups on the backpack to 5V. **As long as the backpack is the only i2c device on the i2c bus with pullups to 5V this is perfectly safe for the Pi. If you have 2 or more 5V i2c devices, the 5V pullups may 'overpower' the Pi's strong 3.3v pullups, in this case you'll want to use a proper level shifter:**
https://www.adafruit.com/products/757 (http://adafru.it/757)


SCLK MOSI #22 #17 #4 SDA0
DNC MISO DNC #21 DNC SCL0 3V3
Pi Header
adafruit!
CE1 #25 #24 DNC RXD GND 5V0
CE0 DNC #23 #18 TXD DNC
+
-
D
C

Using the Adafruit Library

The Python code to work with Adafruit's LED Backpacks on the Pi is available on Github at https://github.com/adafruit/Adafruit_Python_LED_Backpack (http://adafru.it/dEL)
This code should be a good starting point to understanding how you can access SMBus/I2C devices with your Pi, and getting your blinky on.

If you're running Raspbian, you will need to set up I2C first.

Follow this tutorial to fully enable i2c(http://adafru.it/aTI)

Install required software

```
    sudo apt-get update
sudo apt-get install -y git build-essential python-dev python-smbus python-imaging python-pip python-pil
```

Downloading the Code from Github

The easiest way to get the code onto your Pi is to hook up an Ethernet cable, and clone it directly using 'git'. Simply run the following commands from an appropriate location (ex. "/home/pi"):

```
git clone https://github.com/adafruit/Adafruit_Python_LED_Backpack.git
cd Adafruit_Python_LED_Backpack
sudo python setup.py install
```

Testing the Library

Once the code has be downloaded to an appropriate folder, and you have your LED Backpack board properly connected, you can test it out with the following commands (the driver includes a few simple demo programs)

For **8x8 displays**, you can run a simple test with:

```
cd examples
sudo python matrix8x8_test.py
```

Which should result in something like the following:
For **4-character, 7-segment displays**, you can run a clock demo with:

```
sudo python sevensegment_test.py
```

Which should result in the following:

Class Summary

If you're interested in writing your own code, the easy way is to start with the examples in the code repository (the .py files starting with 'ex_'), but the following information may be helpful as well.

Adafruit_LEDBackpack

All of the low-level IO and I2C access is passed through **Adafruit_LEDBackpack.py**, which is used by all of the other classes and examples in the library. This file implements a class named **LEDBackpack**, which has the following functions:

- **setBrightness(brightness)**
  Sets the display brightness with a value between 0 and 15

- **setBlinkRate(blinkRate)**
  Sets the optional blink rate for the display, with one of the following values for blinkRate

  0 = No Blinking
  1 = Blink at 2Hz
  2 = Blink at 1Hz
  3 = Blink at 1/2 Hz

- **setBufferRow(row, value)**
  Updates a single row of data. The HT16K33 contains an internal memory of 8 rows, with each row having 16 bits of data. Each row generally corresponds to one character on a 7-segment display or one row of pixels on an 8x8 display.

- **getBuffer()**
  Returns a list of 8 16-bit values representing the current state of the internal buffer, which can be used to update the display without affecting the rest of the content.

- **clear()**
  Clears the contents of the entire buffer

Adafruit_7Segment

The 7-Segment library is encapsulated in the file**Adafruit_7Segment.py**, and implements a class named **SevenSegment** (since class names can't start with a digit). It instantiates and instance of Adafruit_LEDBackpack internally, and you normally won't need to use the lower level class directly. It implements the following functions:

- **writeDigitRaw(charNumber, value)**
  This will raw a raw 16-bit value to the specified charNumber, which corresponds to a single row in the HT16K33, and one character in the 4x7-segment displays. (Note: row 2 is the colon on most 4 character displays). You can use this to display custom characters that aren't supported natively by the current library.

- **writeDigit(charNumber, value, dot=False)**
  Writes a single decimal (0..9) or hexadecimal (0..9 and A..F) character to the display, with charNumber being the character to update, and value being the digit. You can set a third optional argument, **dot**, to True to set the decimal place after each character.

- **setColon(state)**
  You can pass either True of False to this function to enable or disable the colon that is usually connected to row 2 of the HT16K33.

Adafruit_8x8.py

The **Adafruit_8x8.py** file implements a class names**EightByEight** that can be used to drive 8x8 pixel square LED blocks. It contains the following functions:

- **writeRowRaw(charNumber, value)**
  This will update an entire row of data with the specified 16-bit value (though normally only the last 8-bits are used on one-color 8x8 displays. This is faster than setting individual pixels.

- **setPixel(x, y, color)**
  This function will update a single pixel within the relevant X/Y space of the display. Please keep in mind that lists in Python are zero-based, meaning to sets pixels 1..8 in each direction you actually use the values 0..7, so the following will enable the pixel

3 over and 5 down:

```
grid = EightByEight(address=0x70)
grid.setPixel(2, 4)
```

Examples

If you need a bit of help, simply looking at the provided example python code:

1. **ex_7segment_clock.py**: Displays the current time on a 4*7-segment display, changing the state of the colon every second
2. **ex_8x8_pixels.py**: Constantly updates every pixel on an 8x8 display, one pixel at a time.

You can run these examples with the following code:

```
sudo python ex_8x8_pixels.py
```