


The TMS LCL HW Pack for Raspberry Pi open-source package

Introduction

At TMS software, we have recently published a first version of the free and open-source TMS LCL HW Pack for Raspberry Pi that you can download from:

<http://www.tmssoftware.com/site/tmslclhwpack.asp>. The goal of the TMS LCL HW Pack for Raspberry Pi is to make it very simple to do IO from Pascal applications. One of the things that makes the Raspberry Pi so versatile is its IO capabilities, especially the Raspberry Pi 2 with a 40 pins header that exposes generic GPIO pins, I2C, UART and SPI pins.

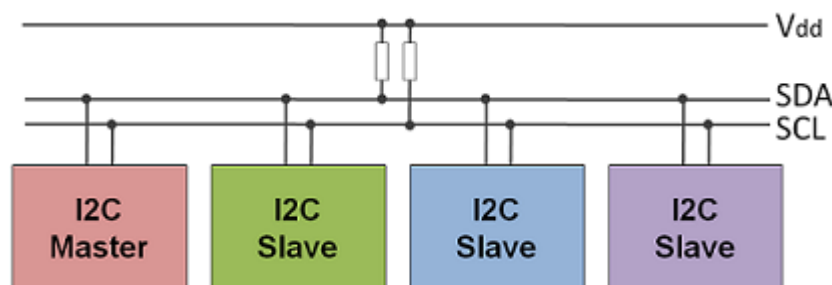


GPIO#	2nd func	Physical Pins		2nd func	GPIO#
		pin#	pin#		
N/A	+3V3	1	2	+5V	N/A
GPIO2	SDA1 (I2C)	3	4	+5V	N/A
GPIO3	SCL1 (I2C)	5	6	GND	N/A
GPIO4	GCLK	7	8	TXD0 (UART)	GPIO14
N/A	GND	9	10	RXD0 (UART)	GPIO15
GPIO17	GEN0	11	12	GEN1	GPIO18
GPIO27	GEN2	13	14	GND	N/A
GPIO22	GEN3	15	16	GEN4	GPIO23
N/A	+3V3	17	18	GEN5	GPIO24
GPIO10	MOSI (SPI)	19	20	GND	N/A
GPIO9	MISO (SPI)	21	22	GEN6	GPIO25
GPIO11	SCLK (SPI)	23	24	CE0_N (SPI)	GPIO8
N/A	GND	25	26	CE1_N (SPI)	GPIO7
EEPROM	ID_SD	27	28	ID_SC	EEPROM
GPIO5	N/A	29	30	GND	N/A
GPIO6	N/A	31	32	-	GPIO12
GPIO13	N/A	33	34	GND	N/A
GPIO19	N/A	35	36	N/A	GPIO16
GPIO26	N/A	37	38	N/A	GPIO20
N/A	GND	39	40	N/A	GPIO21

Raspberry Pi 2 40 pin header

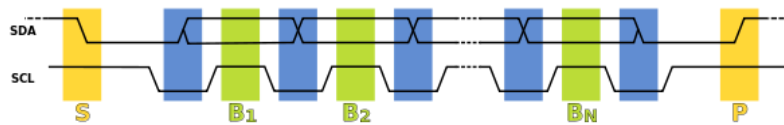
The generic GPIO pins are pins that can be programmatically configured as digital inputs and outputs. There are 6 such generic GPIO pins for a Raspberry Pi 2.

Under programmatic control, you can set a logic 1 or 0 for these pints that translates into a VCC or GND signal on the pin or you can read the voltage VCC or GND on the pin as either 1 or 0. This is useful for digital signals but sadly the Raspberry Pi doesn't offer built-in analog programmable pins like the Arduino does. Next is the i2c bus, which is a two-wire protocol to read & write data on a bus of i2c devices.



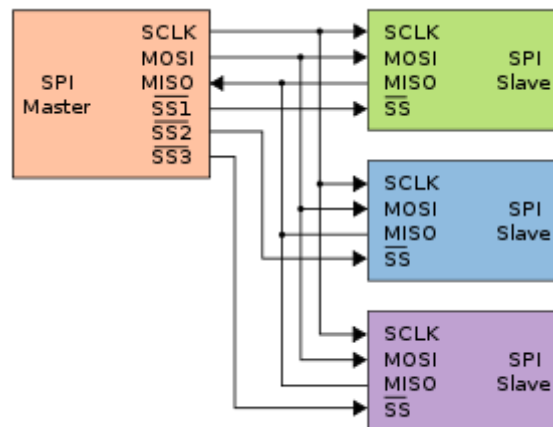
Typical connection between i2c master and slaves

The protocol is described here: <https://en.wikipedia.org/wiki/I%C2%B2C>. In a nutshell, data is digitally serialized and sent on a bus. By means of an address + read/write instruction, data can be read or written from a master who drives the clock. In this case, the master is the Raspberry Pi.



Serialized data on the i2c 2-wire bus.

The next option is the SPI bus (https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus) which is a 4 wire bus protocol also based on master/slave principle where the Raspberry Pi is the master. Along with a device select signal and serial clock, there are two data wires, one for input and one for output.



Typical connection of an SPI master to SPI slaves

Finally, the last option on the 40 pin connector is a classic UART or serial port with TX, RX pins to do classic serial port communication. In this article, we'll focus on the i2c and SPI ports.

Getting devices to extend your Raspberry Pi

Lots of devices exist that can extend your Raspberry Pi. From off the shelf ICs with i2c or SPI interfaces that you use together with some control circuitry and connect to the Raspberry Pi header to ready-made little boards or extenders from well-known companies such as Adafruit (www.adafruit.com) or Sparkfun (www.sparkfun.com)

Using i2c

Each i2c device on the i2c bus has a 7bit address meaning that theoretically 128 devices can be connected. Data is serially transmitted and the clock rate can be selected. 400kbs is a realistic speed that can be used also with devices connected to the Raspberry Pi via a breadboard. Typically, along with the address, data is written or read in blocks of 8 bit. In order to make accessing i2c devices easily accessible from Pascal applications, we have created a base class: TTMSLCLRaspiI2C. This class has property I2CAddress to set the address of the i2c device you want to

access and I2CPort to select what I2C port of the Raspberry Pi to use. To make a connection to an i2c device, call the function TTMSLCLRaspil2C.Open: boolean. To close the connection again, there is the TTMSLCLRaspil2C.Close function and with TTMSLCLRaspil2C.Connected: boolean, you can check at all times the connection state. When there is a connection, you can write one byte of data to the device with TTMSLCLRaspil2C.SetByteRegister and read with TTMSLCLRaspil2C.GetByteRegister.

Interface of the base i2c class:

```
TTMSLCLRaspil2C = class(TComponent)
public
    constructor Create(AOwner: TComponent); override;
    function Open: boolean; virtual;
    function Close: boolean; virtual;
    function Connected: boolean;

    function SetByteRegister(RegNo: Integer; Val: Byte): Integer;
    function GetByteRegister(RegNo: Integer): byte;
published
    property I2CAddress: integer read FI2CAddress write FI2CAddress;
    property I2CPort: CInt read FI2CPort write FI2CPort;
end;
```

Encapsulating i2c device functionality

Typically, a useful communication with an i2c device comes down to read & write bytes in the proper and desired sequence to get the device to do something useful. We have therefore created a number of classes that descend from TTMSLCLRaspil2C and that encapsulate typical setup & communication with commonly used i2c devices:

- TTMSLCLAdaQuad7SegLed: component to control a quad 7 segment LED
- TTMSLCLAdaADC12B: component to read data from a quad analog digital convertor
- TTMSLCLAdaDAC12B: component to write data to a digital analog convertor
- TTMSLCLAdaDispl128x32: component to drive a 128x32 OLED display
- TTMSLCLAdaDispl16x2: component to drive a 16x2 LCD display and read its 5 key inputs
- TTMSLCLAdaBarTemp: component to retrieve temperature & barometric data from a sensor
- TTMSLCLAdaI2CIOExpander: component to control 16 i2c addressed extra GPIO pins
- TTMSLCLAda8x8MatrixLed: component to drive a 8x8 LED matrix



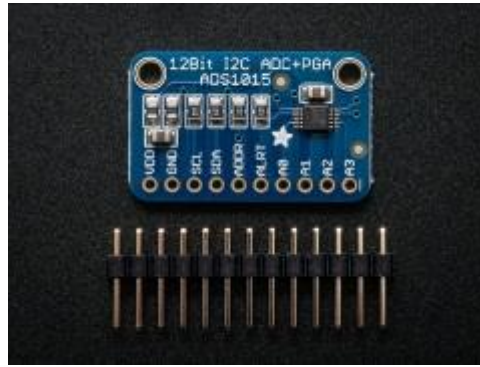
i2c quad 7 segment LED



i2c 128x32 OLED display



i2c digital analog convertor



i2c 4 channel analog digital convertor



i2c 16x2 LCD display + 5 key keypad



i2c temperature & barometric sensor



i2c 8x8 LED array



i2c 16 channel GPIO extender

Using SPI

The SPI protocol is similar to i2c and is encapsulated in the class `TTMSPILCRaspiSPI`. It has a `PortNum` property to select the Raspberry Pi which one of the two SPI ports to use.

There is no device address involved with SPI as a device on SPI is accessed by driving the device selector pin low for the device. This limits the number of devices that can be directly connected to the Raspberry Pi. The Raspberry Pi SPI clock speed is configurable and maximum speed is 125MHz. The `TTMSPILCRaspiSPI` class also has `Open` & `Close` functions and a `Connected` property to read the state. To transfer data via SPI to a device, there are also the functions `SetByteRegister` and `GetByteRegister`. In addition, there are functions: `ReadTransfer` and `WriteTransfer`.

```
function ReadTransfer(buf: pointer; wsize, rsize: integer): boolean;
function WriteTransfer(buf: pointer; wsize: integer): boolean;
```

Read transfer will send wsize number of bytes from the buffer pointed to by buf and read back in the buffer rsize number of bytes. The WriteTransfer function will write wsize bytes from the buffer pointed to by buf.

At this time, there is one component descending from TTMSLCLRaspiSPI that controls the TTMSLCLAdaFram8KSPI device, an 8K FRAM device.



SPI 8k FRAM

Reading and writing in the FRAM (ferro-electric memory that can keep its memory also when not powered) is simple with read & write commands per byte:

```
function WriteByte(Adr: word; val: byte): boolean;
function ReadByte(Adr: word; var val: byte): boolean;
```

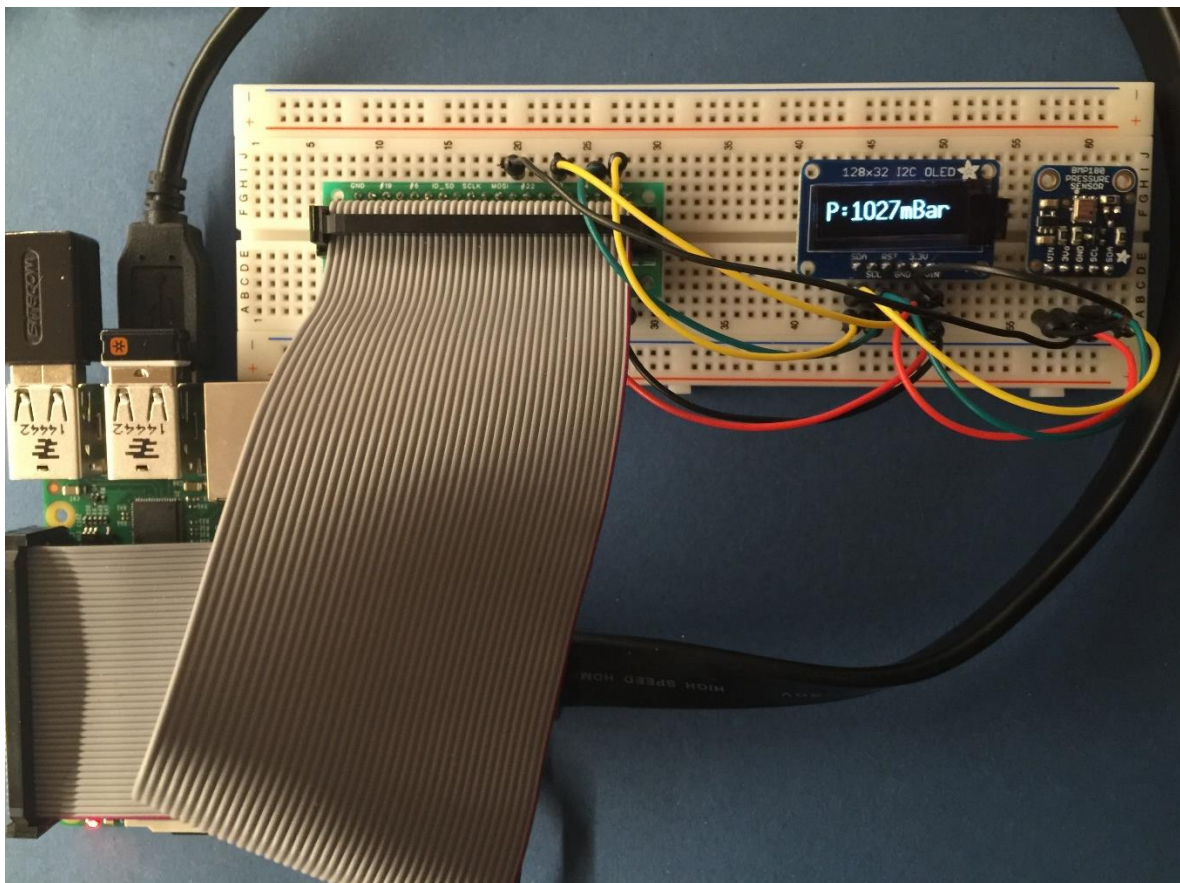
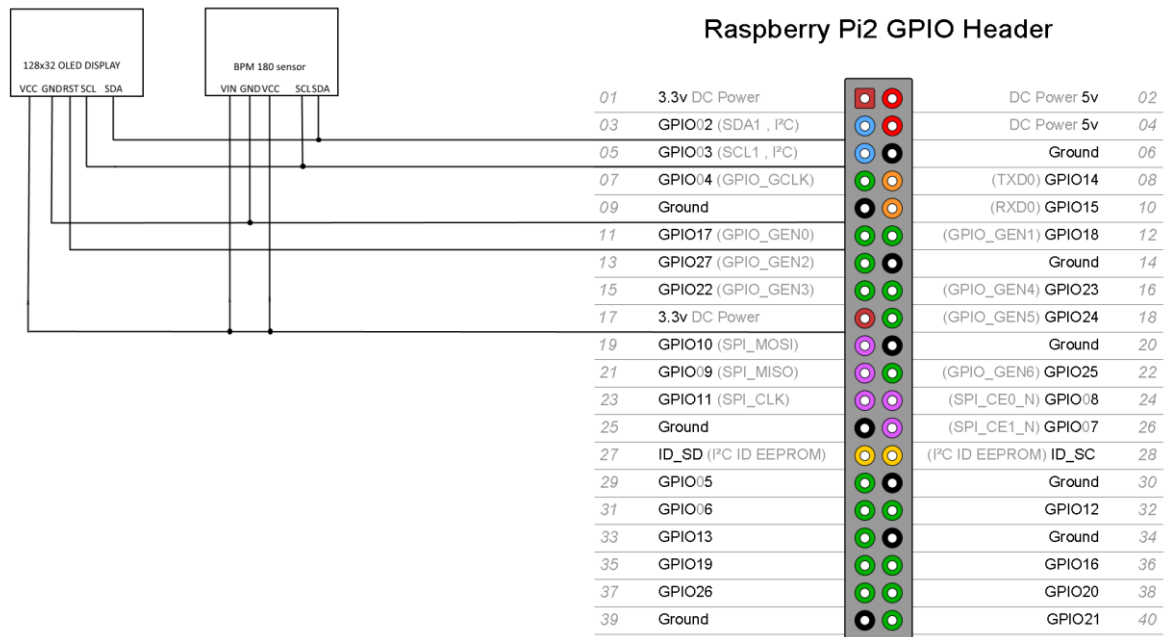
Putting it together in a fun project

To put the pieces together, we create a small project. We'll use the TTMSLCLAdaBarTemp component to read out temperature and air pressure from this BMP180 sensor. The values read will be displayed alternatingly on a 128x32 pixel OLED screen connected to the Raspberry Pi. While both devices are i2C devices, there is one thing particular about the 128x32 OLED screen and that is that it needs to get a reset pulse to get initialized. So, the TTMSLCLAdaDispl128x32 assumes the Raspberry Pi GPIO pin 17 is connected to the RST pin of the 128x32 OLED screen as its method TTMSLCLAdaDispl128x32.InitGPIO will configure and pull this RST pin low to reset the device.

Assembling the hardware

To quickly put the hardware together, you can use a breadboard and a 40 pin flatcable with breadboard connector. Wire up the Raspberry Pi connector signals GPIO 2 (SDA) to the SDA pin on both the Adafruit BMP180 (<https://www.adafruit.com/products/1603>) and 128x32 OLED (See: <https://www.adafruit.com/products/931>). Do the same for GPIO 3 (SCL) to the SCL pin. Connect a connector GND and VCC pin also the BMP180 and 128x32 respective GND and VCC pins. Finally, the temperature and barometric sensor requires a 3V3 pin, so connect this to a Raspberry Pi header 3.3V pin and for the reset pulse of the OLED display, connect GPIO 17 to the OLED128x32 pin RST.

Schema:

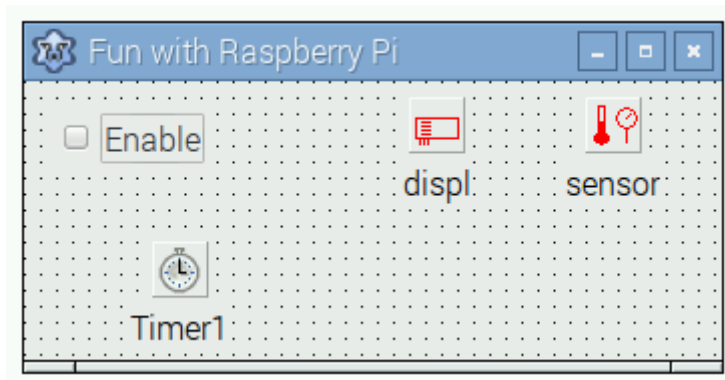


Hardware assembled on a breadboard

Putting the software together

To access the BPM180 sensor, drop the component TTMSLCLAdaBarTemp on the form and set its name to 'sensor'. For the OLED display, drop the TTMSLCLAdaDispl128x32 on the form and set its

name to 'displ'. Drop a TCheckBox and TTimer on the form and set the TTimer interval to 2000ms and Enabled to false.



In the form's OnShow event, we'll open the connection the BPM180 and OLED display by calling the Open method:

```
procedure TForm1.FormShow(Sender: TObject);
begin
    // this initializes GPIO 17 and pulse
    // the reset pin on the display low to initialize
    displ.InitGPIO;
    // open the display and show a startup text
    if displ.Open then
    begin
        displ.Clear;
        displ.DrawTextLarge(0,4,'Ready....');
        displ.Display;
    end;
    // open the sensor
    sensor.Open;
end;
```

Next, to start displaying the temperature and air pressure alternatingly, check the checkbox to start the timer:

```
procedure TForm1.CheckBox1Change(Sender: TObject);
begin
    timer1.Enabled := Checkbox1.Checked;
end;
```

Then, from the TTimer.OnTimer event, alternatingly we read the temperature and air pressure from the sensor and show it on the display:

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
    f: single;
    s: string;
begin
    if sensor.Connected then
    begin
        if ShowTemp then
        begin
```

```

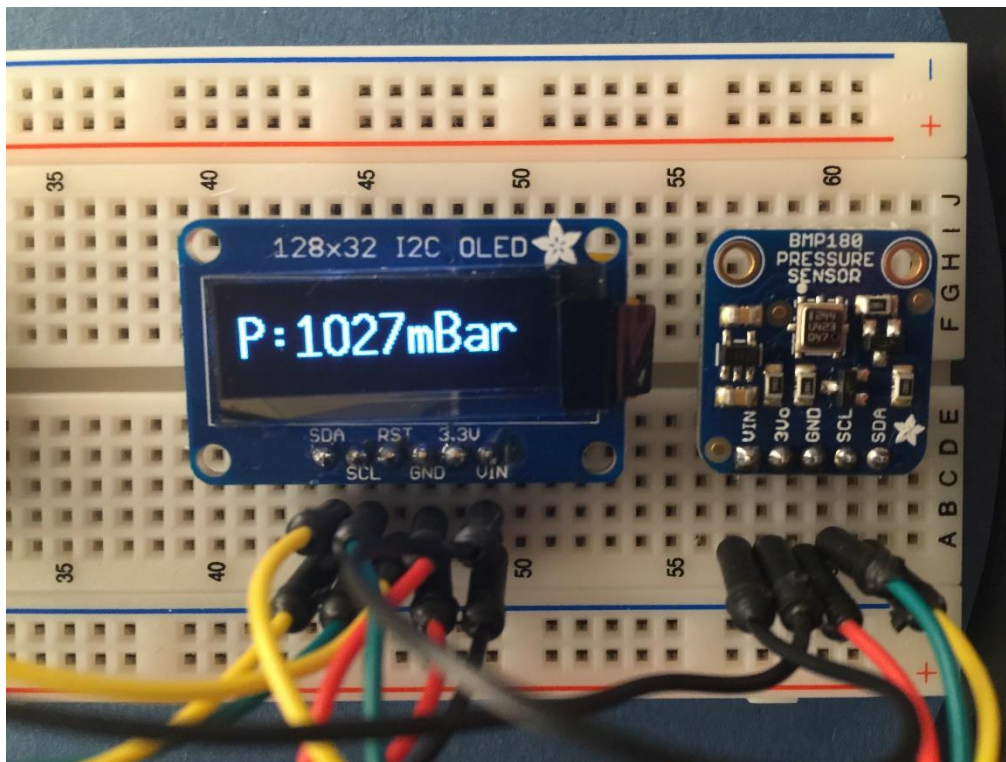
    f := sensor.GetTemperature;
    s := 'T:' + Format('%.2f',[f]) + ' C';
end
else
begin
    f := sensor.GetPressure;
    s := 'P:' + Format('%.0f',[f]) + 'mBar';
end;

displ.Clear;
displ.DrawTextLarge(0,4,s);
displ.Display;

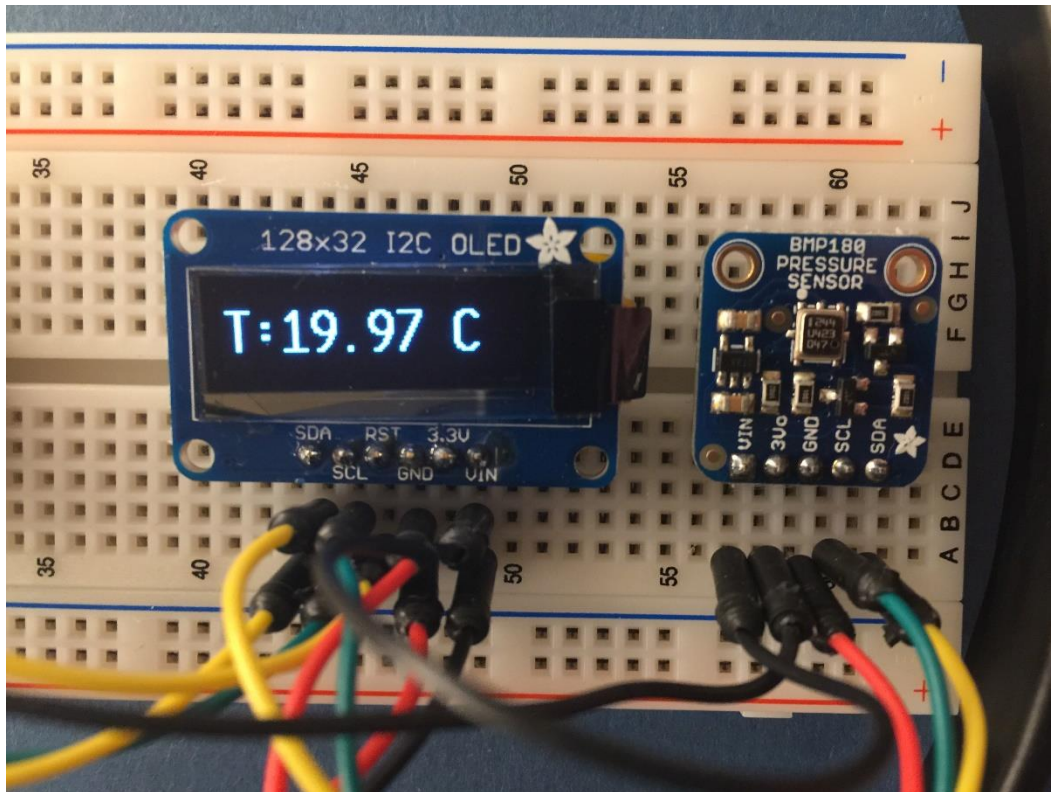
ShowTemp := not ShowTemp;
end;
end;

```

One important note: configuring the GPIO pins as input or output on Raspberry Pi requires admin privileges, so you might want to make sure the app runs with admin privileges.



Project in action 1: showing barometric pressure



Project in action 2: showing ambient temperature

Conclusion

With a few classes that enable you to directly access hardware extensions for the Raspberry Pi, accessing this hardware becomes dead-easy and you can fully focus on being creative hooking up all kinds of devices to the Raspberry Pi and automating things. At this time we have classes for 9 hardware extensions and we're busy developing more for other extensions. As the TMS LCL HW Pack is an open-source project, we also invite anyone to participate and share classes that you might have created that wraps other hardware devices. Yes, it's 2016 and programming in Pascal has never been more fun!