Temperature Sensing Embedded System

EECS 3215: Embedded Systems

Instructor: Ebrahim Ghafar-Zadeh

Winter 2016

Ariel Laboriante

Nisha Sharma

Introduction

This project is part of the additional course work for EECS 3215 which improve the students' knowledge while working with different embedded systems. This project provides an opportunity to apply the knowledge gained in class in order to design systems with applications in different real world scenarios. Our project works towards exploring opportunities in the field of temperature sensing and is an initial step towards achieving the goal of wireless human temperature sensing. Our idea is to monitor the body temperature of living beings including humans and/or pets (and then store that information wirelessly for future applications and research) for clinical use. The scope of opportunities to use this information and expand on this implementation are endless. One application of this approach, when extended to wearable and wireless temperature sensing, could be for testing the effect of certain medicines on the body of individuals. Another application of this data could be to aid in behavioral studies to find the relationship between the change in temperature and a certain behavioural pattern. For the purpose of this short term project, given the limited time to work, we have designed and tested a simple embedded system to monitor temperature changes of surroundings and of an individual or pet. This report briefly documents our approach and implementation.

Hardware

The system uses BeagleBone Black (a microcontroller), a precision semiconductor temperature sensor LM35DZ (analog output), a potentiometer, the DM1623 LCD Display, a breadboard and wires. This section briefly describes the components and their connectivity to the board.

The LM35DZ temperature sensor has a better precision for the purpose of this project. Its features (as per the datasheet [1]) are listed here:
Typical accuracy: ±¼°C at room temperature; ±¾°C over a full−55°C to 150°C temperature range.
Operating Voltage: 4-30V (Works with BBB for body temperature sensing ~36-40°C)
Temperature Range: -55 to 155 °C

Figure 1 indicates the pin diagram of the sensor; below is the description of the connection with BeagleBone Black (BBB)

- Vout is the analog output signal connected to Pin P9_40 (AIN1) of BBB as analog input
- Vs is connected to Positive supply Pin P9_3 (VDD 3V3) of BBB
- Ground of sensor is connected to Pin P9_34 (GNDA_ADC) of BBB

Figure 1: Pin Diagram for LM35DZ

Figure 2 illustrates the connections to the BeagleBone Black. For the final implementation we extended the sensor wires by soldering as can be seen in actual hardware setup (Figure 4).

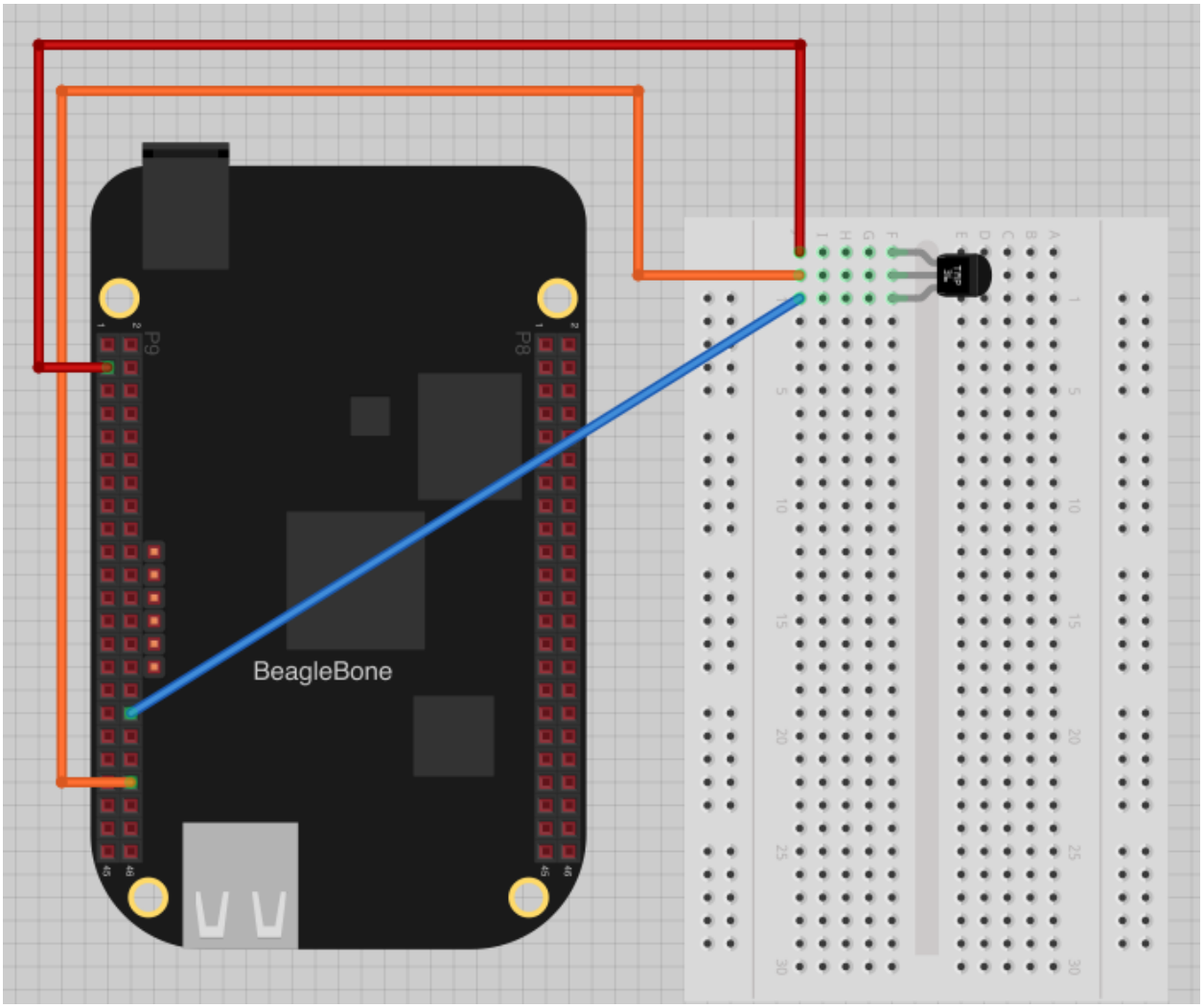


Figure 2: Connections between BeagleBone Black and LM35DZ using breadboard [2]

Connecting with LCD display:

We used the DM1623 16-pin LCD display [3], for the purpose of this project to display sensed temperature in degrees Celsius and degrees Fahrenheit. The pin functions for the LCD is in table 1. The schematic of the connections is depicted in Figure 3.

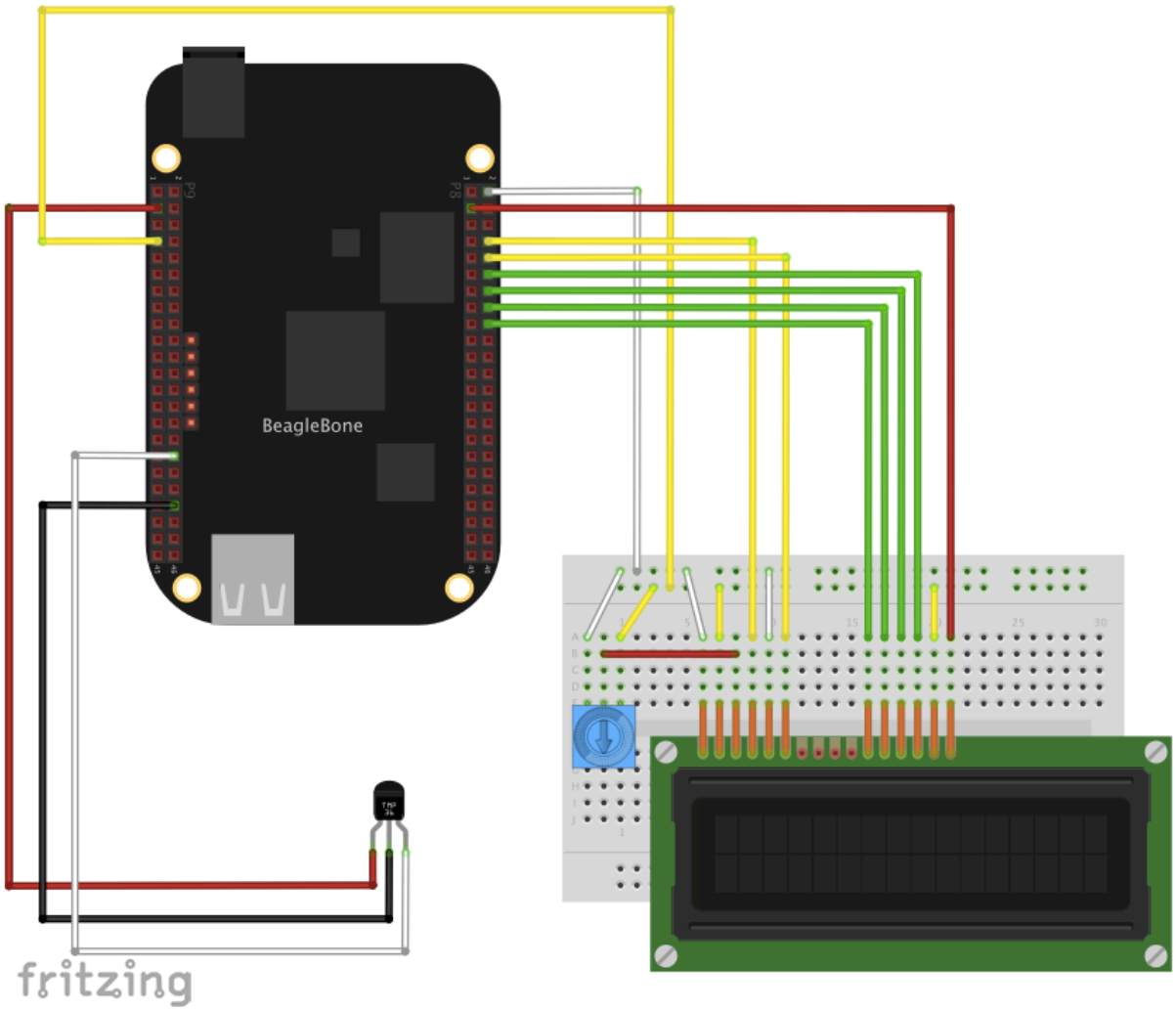


Figure 3: Schematic of the project connection, including the LCD display*

*The use of a 10k potentiometer was needed to adjust the contrast of the LCD display.
The use of a logic level converter is recommended for further implementation of LCD display in this project due to the different voltage range of the microcontroller (Vout = 3.3V) and the display (Vin = 5V).

Table 1: Pin Functions for the LCD display*

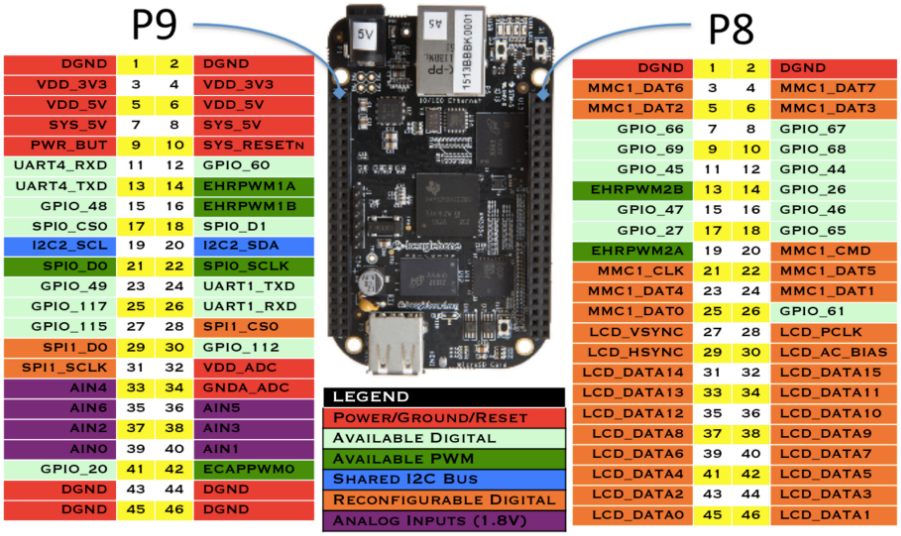
Pin No.	Symbol	Function
1	V_{SS}	0 V (GND)
2	V_{DD}	+5 V
3	V_O	LCD drive supply
4	RS	Register select pin 0: Instruction register (write) Busy flag and address counter (read) 1: Data register (read/write)
5	R/W	Read/write pin 0: Write; MPU → LCD module 1: Read; LCD module → MPU
6	E	Enable flag
7 to 10	DB0 to DB3	Data bus (tristate bidirectional pins) Used as the lower 4 bit pins when an 8-bit interface is used. Unused when a 4-bit interface is used.
11 to 14	DB4 to DB7	Data bus (tristate bidirectional pins) Used as the upper 4 bit pins when an 8-bit interface is used. Used as the 4 data bits when a 4-bit interface is used. DB7 is also be used as the busy flag.

*Pin 15 (LED+) was connected the VDD and Pin 16 (EL) was connected to P8_7 to enable the backlight

Table 2: Circuit Connections

LCD Pin	BBB Pin
1	DGND (Pin 8_2)
2	SYS_5V (Pin P9_7)
3	To output of potentiometer
4	GPIO_67(P 8_8)
5	DGND (Pin P8_2)
6	GPIO_68 (Pin 8_10)
7-10	Not used
11	GPIO_65 (Pin 8_18)
12	GPIO_46 (Pin 8_16)
13	GPIO_26 (Pin 8_14)
14	GPIO_44 (Pin 8_12)
15	SYS_5V (Pin P9_7)
16	MMC1_DAT6(Pin P8_3)

Table 3: Pinout diagram of the BBB [4]



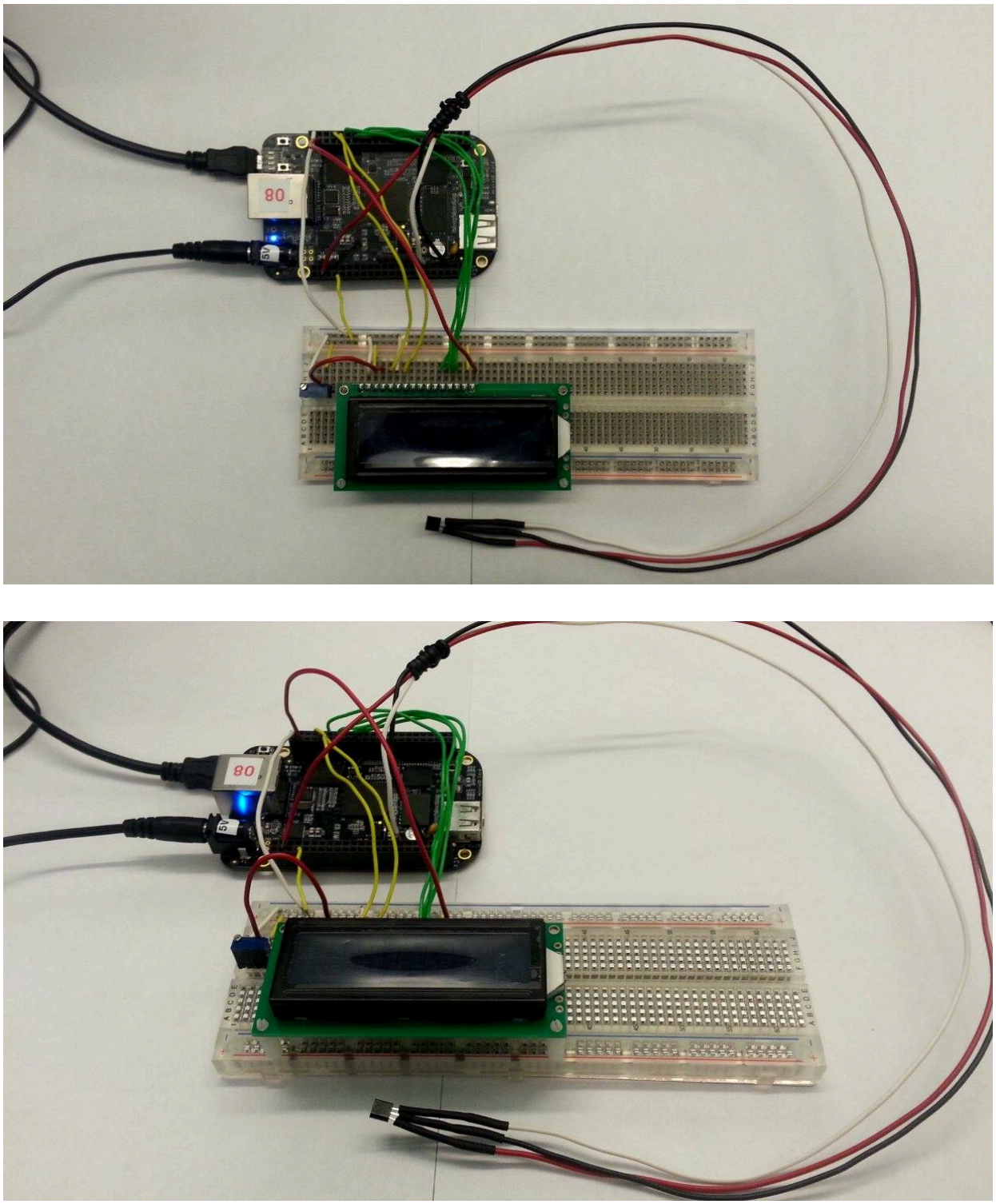


Figure 4a, 4b: Actual hardware setup

Software

For the purpose of this project we utilized Python, with the intention of learning a different language. Our program enables BBB to receive the analog input from the sensor and convert it to the digital value of the temperature (which is readable on the monitor) and then transferred the information to the LCD display.
For the implementation we utilized the source code already available from Adafruit Learning Systems (July 2013) and extended the program to display it on the LCD utilizing python library `Adafruit_CharLCD.py` [5] shown in Appendix B. Our code `temperature.py` is documented in Appendix A.

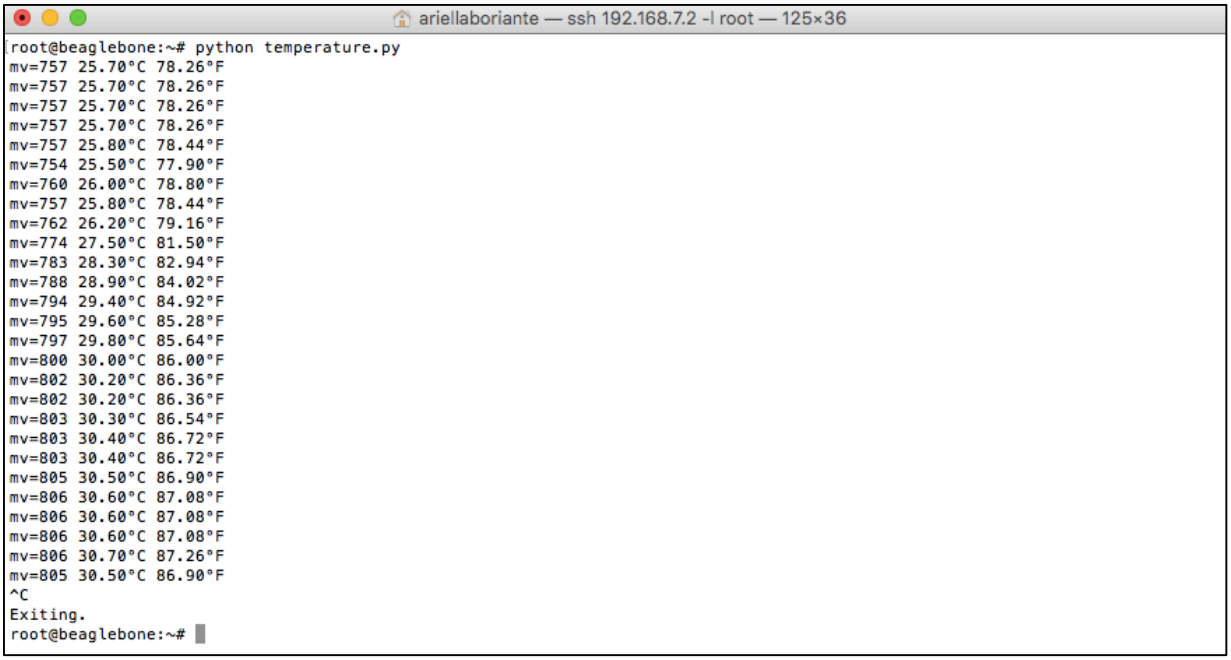


Figure 5: Screenshot of the output on the terminal

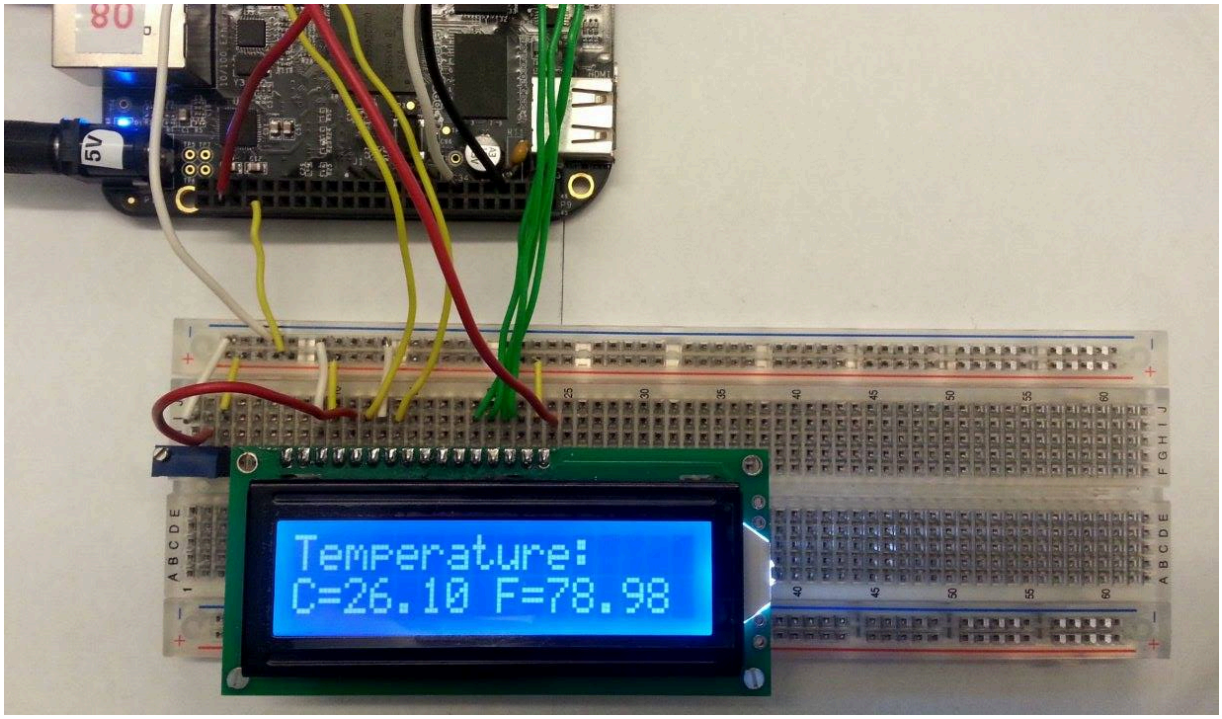


Figure 6: Output on LCD display

Conclusion

The prototype built and programmed in the duration of this project is able to successfully measure precise temperature for various purposes including but not limited to room temperature and body temperature. Further work and research is required to extend the prototype's functionality to a portable and wireless temperature sensing system. In order to make these implementations feasible we recommend using a different microcontroller that allows the final device to be smaller and compact.

References

[1] T. Instruments, "LM35 Precision Centigrade Temperature Sensors," Aug 1999 [Revised Jan 2016].
[2] S. Monk, "Measuring Temperature with a BeagleBone Black," Adafruit Learning Systems, Jul 2013.
[3] SANYO, "16 Characters X 2 Lines Liquid Crystal Dot Matrix Display Module," DM1623 datasheet
[4]"BeagleBoard.org - bone101", Beagleboard.org, 2016. [Online]. Available: http://beagleboard.org/support/bone101. [Accessed: 15- Apr- 2016]
[5] T. DiCola, "Adafruit_CharLCD.py," Adafruit Industries, 15 Jul 2014. [Online]. Available: https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/blob/master/Adafruit_CharLCD/Adafruit_CharLCD.py. [Accessed 15 Apr 2016].

Appendix A – temperature.py

```
#!/usr/bin/python
import Adafruit_BBIO.ADC as ADC
import Adafruit_CharLCD as LCD
import time

#BeagleBone Black configuration:
lcd_rs        = 'P8_8'
lcd_en        = 'P8_10'
lcd_d4        = 'P8_18'
lcd_d5        = 'P8_16'
lcd_d6        = 'P8_14'
lcd_d7        = 'P8_12'
lcd_backlight = 'P8_7'

# Define LCD column and row size for 16x2 LCD.
lcd_columns = 16
lcd_rows    = 2

# Initialize the LCD using the pins above.
lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7,
                           lcd_columns, lcd_rows, lcd_backlight)

# Initialize heat sensor
sensor_pin = 'P9_40'
ADC.setup()

while True:
    try:
        reading = ADC.read(sensor_pin)
        millivolts = reading * 1800  # 1.8V reference = 1800 mV
        temp_c = (millivolts - 500) / 10
        temp_f = (temp_c * 9/5) + 32
        # print('mv=%d C=%d F=%d' % (millivolts, temp_c, temp_f))
        s = u"mv=%d %d\u00B0C %d\u00B0F" % (millivolts, temp_c, temp_f)
        print u'{0}'.format(s).encode('utf-8')
        lcd.clear()
        lcd.message('Temperature:\nmv=%d C=%d F=%d' % (millivolts, temp_c, temp_f))
        time.sleep(1.0)
    except KeyboardInterrupt:
        sys.stdout.write('\nExiting.\n')
        sys.exit()
```

Appendix B – Adafruit_CharLCD.py

```
# Copyright (c) 2014 Adafruit Industries
# Author: Tony DiCola
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
# THE SOFTWARE.
import time

import Adafruit_GPIO as GPIO
import Adafruit_GPIO.I2C as I2C
import Adafruit_GPIO.MCP230xx as MCP
import Adafruit_GPIO.PWM as PWM


# Commands
LCD_CLEARDISPLAY        = 0x01
LCD_RETURNHOME          = 0x02
LCD_ENTRYMODESET        = 0x04
LCD_DISPLAYCONTROL      = 0x08
LCD_CURSORSHIFT         = 0x10
LCD_FUNCTIONSET         = 0x20
LCD_SETCGRAMADDR        = 0x40
LCD_SETDDRAMADDR        = 0x80

# Entry flags
LCD_ENTRYRIGHT          = 0x00
LCD_ENTRYLEFT           = 0x02
LCD_ENTRYSHIFTINCREMENT = 0x01
LCD_ENTRYSHIFTDECREMENT = 0x00

# Control flags
LCD_DISPLAYON           = 0x04
LCD_DISPLAYOFF          = 0x00
LCD_CURSORON            = 0x02
LCD_CURSOROFF           = 0x00
LCD_BLINKON             = 0x01
LCD_BLINKOFF            = 0x00

# Move flags
LCD_DISPLAYMOVE         = 0x08
LCD_CURSORMOVE          = 0x00
LCD_MOVERIGHT           = 0x04
LCD_MOVELEFT            = 0x00

# Function set flags
LCD_8BITMODE            = 0x10
```

```
LCD_4BITMODE            = 0x00
LCD_2LINE               = 0x08
LCD_1LINE               = 0x00
LCD_5x10DOTS            = 0x04
LCD_5x8DOTS             = 0x00

# Offset for up to 4 rows.
LCD_ROW_OFFSETS         = (0x00, 0x40, 0x14, 0x54)

# Char LCD plate GPIO numbers.
LCD_PLATE_RS            = 15
LCD_PLATE_RW            = 14
LCD_PLATE_EN            = 13
LCD_PLATE_D4            = 12
LCD_PLATE_D5            = 11
LCD_PLATE_D6            = 10
LCD_PLATE_D7            = 9
LCD_PLATE_RED           = 6
LCD_PLATE_GREEN         = 7
LCD_PLATE_BLUE          = 8

# Char LCD plate button names.
SELECT                  = 0
RIGHT                   = 1
DOWN                    = 2
UP                      = 3
LEFT                    = 4

class Adafruit_CharLCD(object):
    """Class to represent and interact with an HD44780 character LCD display."""

    def __init__(self, rs, en, d4, d5, d6, d7, cols, lines, backlight=None,
                 invert_polarity=True,
                 enable_pwm=False,
                 gpio=GPIO.get_platform_gpio(),
                 pwm=PWM.get_platform_pwm(),
                 initial_backlight=1.0):
        """Initialize the LCD.  RS, EN, and D4...D7 parameters should be the pins
        connected to the LCD RS, clock enable, and data line 4 through 7 connections.
        The LCD will be used in its 4-bit mode so these 6 lines are the only ones
        required to use the LCD.  You must also pass in the number of columns and
        lines on the LCD.

        If you would like to control the backlight, pass in the pin connected to
        the backlight with the backlight parameter.  The invert_polarity boolean
        controls if the backlight is one with a LOW signal or HIGH signal.  The
        default invert_polarity value is True, i.e. the backlight is on with a
        LOW signal.

        You can enable PWM of the backlight pin to have finer control on the
        brightness.  To enable PWM make sure your hardware supports PWM on the
        provided backlight pin and set enable_pwm to True (the default is False).
        The appropriate PWM library will be used depending on the platform, but
        you can provide an explicit one with the pwm parameter.

        The initial state of the backlight is ON, but you can set it to an
        explicit initial state with the initial_backlight parameter (0 is off,
        1 is on/full bright).

        You can optionally pass in an explicit GPIO class,
        for example if you want to use an MCP230xx GPIO extender.  If you don't
        pass in an GPIO instance, the default GPIO for the running platform will
        be used.
```

```
        """
        # Save column and line state.
        self._cols = cols
        self._lines = lines
        # Save GPIO state and pin numbers.
        self._gpio = gpio
        self._rs = rs
        self._en = en
        self._d4 = d4
        self._d5 = d5
        self._d6 = d6
        self._d7 = d7
        # Save backlight state.
        self._backlight = backlight
        self._pwm_enabled = enable_pwm
        self._pwm = pwm
        self._blpol = not invert_polarity
        # Setup all pins as outputs.
        for pin in (rs, en, d4, d5, d6, d7):
            gpio.setup(pin, GPIO.OUT)
        # Setup backlight.
        if backlight is not None:
            if enable_pwm:
                pwm.start(backlight, self._pwm_duty_cycle(initial_backlight))
            else:
                gpio.setup(backlight, GPIO.OUT)
                gpio.output(backlight, self._blpol if initial_backlight else not
self._blpol)
        # Initialize the display.
        self.write8(0x33)
        self.write8(0x32)
        # Initialize display control, function, and mode registers.
        self.displaycontrol = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF
        self.displayfunction = LCD_4BITMODE | LCD_1LINE | LCD_2LINE | LCD_5x8DOTS
        self.displaymode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT
        # Write registers.
        self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)
        self.write8(LCD_FUNCTIONSET | self.displayfunction)
        self.write8(LCD_ENTRYMODESET | self.displaymode)  # set the entry mode
        self.clear()

    def home(self):
        """Move the cursor back to its home (first line and first column)."""
        self.write8(LCD_RETURNHOME)  # set cursor position to zero
        self._delay_microseconds(3000)  # this command takes a long time!

    def clear(self):
        """Clear the LCD."""
        self.write8(LCD_CLEARDISPLAY)  # command to clear display
        self._delay_microseconds(3000)  # 3000 microsecond sleep, clearing the display
takes a long time

    def set_cursor(self, col, row):
        """Move the cursor to an explicit column and row position."""
        # Clamp row to the last row of the display.
        if row > self._lines:
            row = self._lines - 1
        # Set location.
        self.write8(LCD_SETDDRAMADDR | (col + LCD_ROW_OFFSETS[row]))

    def enable_display(self, enable):
        """Enable or disable the display.  Set enable to True to enable."""
        if enable:
```

```
            self.displaycontrol |= LCD_DISPLAYON
        else:
            self.displaycontrol &= ~LCD_DISPLAYON
        self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

    def show_cursor(self, show):
        """Show or hide the cursor.  Cursor is shown if show is True."""
        if show:
            self.displaycontrol |= LCD_CURSORON
        else:
            self.displaycontrol &= ~LCD_CURSORON
        self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

    def blink(self, blink):
        """Turn on or off cursor blinking.  Set blink to True to enable blinking."""
        if blink:
            self.displaycontrol |= LCD_BLINKON
        else:
            self.displaycontrol &= ~LCD_BLINKON
        self.write8(LCD_DISPLAYCONTROL | self.displaycontrol)

    def move_left(self):
        """Move display left one position."""
        self.write8(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT)

    def move_right(self):
        """Move display right one position."""
        self.write8(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT)

    def set_left_to_right(self):
        """Set text direction left to right."""
        self.displaymode |= LCD_ENTRYLEFT
        self.write8(LCD_ENTRYMODESET | self.displaymode)

    def set_right_to_left(self):
        """Set text direction right to left."""
        self.displaymode &= ~LCD_ENTRYLEFT
        self.write8(LCD_ENTRYMODESET | self.displaymode)

    def autoscroll(self, autoscroll):
        """Autoscroll will 'right justify' text from the cursor if set True,
        otherwise it will 'left justify' the text.
        """
        if autoscroll:
            self.displaymode |= LCD_ENTRYSHIFTINCREMENT
        else:
            self.displaymode &= ~LCD_ENTRYSHIFTINCREMENT
        self.write8(LCD_ENTRYMODESET | self.displaymode)

    def message(self, text):
        """Write text to display.  Note that text can include newlines."""
        line = 0
        # Iterate through each character.
        for char in text:
            # Advance to next line if character is a new line.
            if char == '\n':
                line += 1
                # Move to left or right side depending on text direction.
                col = 0 if self.displaymode & LCD_ENTRYLEFT > 0 else self._cols-1
                self.set_cursor(col, line)
            # Write the character to the display.
            else:
                self.write8(ord(char), True)
```

```
    def set_backlight(self, backlight):
        """Enable or disable the backlight.  If PWM is not enabled (default), a
        non-zero backlight value will turn on the backlight and a zero value will
        turn it off.  If PWM is enabled, backlight can be any value from 0.0 to
        1.0, with 1.0 being full intensity backlight.
        """
        if self._backlight is not None:
            if self._pwm_enabled:
                self._pwm.set_duty_cycle(self._backlight,
self._pwm_duty_cycle(backlight))
            else:
                self._gpio.output(self._backlight, self._blpol if backlight else not
self._blpol)

    def write8(self, value, char_mode=False):
        """Write 8-bit value in character or data mode.  Value should be an int
        value from 0-255, and char_mode is True if character data or False if
        non-character data (default).
        """
        # One millisecond delay to prevent writing too quickly.
        self._delay_microseconds(1000)
        # Set character / data bit.
        self._gpio.output(self._rs, char_mode)
        # Write upper 4 bits.
        self._gpio.output_pins({ self._d4: ((value >> 4) & 1) > 0,
                                 self._d5: ((value >> 5) & 1) > 0,
                                 self._d6: ((value >> 6) & 1) > 0,
                                 self._d7: ((value >> 7) & 1) > 0 })
        self._pulse_enable()
        # Write lower 4 bits.
        self._gpio.output_pins({ self._d4: (value        & 1) > 0,
                                 self._d5: ((value >> 1) & 1) > 0,
                                 self._d6: ((value >> 2) & 1) > 0,
                                 self._d7: ((value >> 3) & 1) > 0 })
        self._pulse_enable()

    def create_char(self, location, pattern):
        """Fill one of the first 8 CGRAM locations with custom characters.
        The location parameter should be between 0 and 7 and pattern should
        provide an array of 8 bytes containing the pattern. E.g. you can easyly
        design your custom character at http://www.quinapalus.com/hd44780udg.html
        To show your custom character use eg. lcd.message('\x01')
        """
        # only position 0..7 are allowed
        location &= 0x7
        self.write8(LCD_SETCGRAMADDR | (location << 3))
        for i in range(8):
            self.write8(pattern[i], char_mode=True)

    def _delay_microseconds(self, microseconds):
        # Busy wait in loop because delays are generally very short (few
microseconds).
        end = time.time() + (microseconds/1000000.0)
        while time.time() < end:
            pass

    def _pulse_enable(self):
        # Pulse the clock enable line off, on, off to send command.
        self._gpio.output(self._en, False)
        self._delay_microseconds(1)       # 1 microsecond pause - enable pulse must be
> 450ns
        self._gpio.output(self._en, True)
```

```
        self._delay_microseconds(1)       # 1 microsecond pause - enable pulse must be
> 450ns
        self._gpio.output(self._en, False)
        self._delay_microseconds(1)       # commands need > 37us to settle

    def _pwm_duty_cycle(self, intensity):
        # Convert intensity value of 0.0 to 1.0 to a duty cycle of 0.0 to 100.0
        intensity = 100.0*intensity
        # Invert polarity if required.
        if not self._blpol:
            intensity = 100.0-intensity
        return intensity


class Adafruit_RGBCharLCD(Adafruit_CharLCD):
    """Class to represent and interact with an HD44780 character LCD display with
    an RGB backlight."""

    def __init__(self, rs, en, d4, d5, d6, d7, cols, lines, red, green, blue,
                 gpio=GPIO.get_platform_gpio(),
                 invert_polarity=True,
                 enable_pwm=False,
                 pwm=PWM.get_platform_pwm(),
                 initial_color=(1.0, 1.0, 1.0)):
        """Initialize the LCD with RGB backlight.  RS, EN, and D4...D7 parameters
        should be the pins connected to the LCD RS, clock enable, and data line
        4 through 7 connections. The LCD will be used in its 4-bit mode so these
        6 lines are the only ones required to use the LCD.  You must also pass in
        the number of columns and lines on the LCD.

        The red, green, and blue parameters define the pins which are connected
        to the appropriate backlight LEDs.  The invert_polarity parameter is a
        boolean that controls if the LEDs are on with a LOW or HIGH signal.  By
        default invert_polarity is True, i.e. the backlight LEDs are on with a
        low signal.  If you want to enable PWM on the backlight LEDs (for finer
        control of colors) and the hardware supports PWM on the provided pins,
        set enable_pwm to True.  Finally you can set an explicit initial backlight
        color with the initial_color parameter.  The default initial color is
        white (all LEDs lit).

        You can optionally pass in an explicit GPIO class,
        for example if you want to use an MCP230xx GPIO extender.  If you don't
        pass in an GPIO instance, the default GPIO for the running platform will
        be used.
        """
        super(Adafruit_RGBCharLCD, self).__init__(rs, en, d4, d5, d6, d7,
                                                  cols,
                                                  lines,
                                                  enable_pwm=enable_pwm,
                                                  backlight=None,
                                                  invert_polarity=invert_polarity,
                                                  gpio=gpio,
                                                  pwm=pwm)
        self._red = red
        self._green = green
        self._blue = blue
        # Setup backlight pins.
        if enable_pwm:
            # Determine initial backlight duty cycles.
            rdc, gdc, bdc = self._rgb_to_duty_cycle(initial_color)
            pwm.start(red, rdc)
            pwm.start(green, gdc)
            pwm.start(blue, bdc)
```

```
        else:
            gpio.setup(red, GPIO.OUT)
            gpio.setup(green, GPIO.OUT)
            gpio.setup(blue, GPIO.OUT)
            self._gpio.output_pins(self._rgb_to_pins(initial_color))

    def _rgb_to_duty_cycle(self, rgb):
        # Convert tuple of RGB 0-1 values to tuple of duty cycles (0-100).
        red, green, blue = rgb
        # Clamp colors between 0.0 and 1.0
        red = max(0.0, min(1.0, red))
        green = max(0.0, min(1.0, green))
        blue = max(0.0, min(1.0, blue))
        return (self._pwm_duty_cycle(red),
                self._pwm_duty_cycle(green),
                self._pwm_duty_cycle(blue))

    def _rgb_to_pins(self, rgb):
        # Convert tuple of RGB 0-1 values to dict of pin values.
        red, green, blue = rgb
        return { self._red:   self._blpol if red else not self._blpol,
                 self._green: self._blpol if green else not self._blpol,
                 self._blue:  self._blpol if blue else not self._blpol }

    def set_color(self, red, green, blue):
        """Set backlight color to provided red, green, and blue values.  If PWM
        is enabled then color components can be values from 0.0 to 1.0, otherwise
        components should be zero for off and non-zero for on.
        """
        if self._pwm_enabled:
            # Set duty cycle of PWM pins.
            rdc, gdc, bdc = self._rgb_to_duty_cycle((red, green, blue))
            self._pwm.set_duty_cycle(self._red, rdc)
            self._pwm.set_duty_cycle(self._green, gdc)
            self._pwm.set_duty_cycle(self._blue, bdc)
        else:
            # Set appropriate backlight pins based on polarity and enabled colors.
            self._gpio.output_pins({self._red:   self._blpol if red else not
self._blpol,
                                    self._green: self._blpol if green else not
self._blpol,
                                    self._blue:  self._blpol if blue else not
self._blpol })

    def set_backlight(self, backlight):
        """Enable or disable the backlight.  If PWM is not enabled (default), a
        non-zero backlight value will turn on the backlight and a zero value will
        turn it off.  If PWM is enabled, backlight can be any value from 0.0 to
        1.0, with 1.0 being full intensity backlight.  On an RGB display this
        function will set the backlight to all white.
        """
        self.set_color(backlight, backlight, backlight)



class Adafruit_CharLCDPlate(Adafruit_RGBCharLCD):
    """Class to represent and interact with an Adafruit Raspberry Pi character
    LCD plate."""

    def __init__(self, address=0x20, busnum=I2C.get_default_bus(), cols=16, lines=2):
        """Initialize the character LCD plate.  Can optionally specify a separate
        I2C address or bus number, but the defaults should suffice for most needs.
        Can also optionally specify the number of columns and lines on the LCD
```

```
        (default is 16x2).
        """
        # Configure MCP23017 device.
        self._mcp = MCP.MCP23017(address=address, busnum=busnum)
        # Set LCD R/W pin to low for writing only.
        self._mcp.setup(LCD_PLATE_RW, GPIO.OUT)
        self._mcp.output(LCD_PLATE_RW, GPIO.LOW)
        # Set buttons as inputs with pull-ups enabled.
        for button in (SELECT, RIGHT, DOWN, UP, LEFT):
            self._mcp.setup(button, GPIO.IN)
            self._mcp.pullup(button, True)
        # Initialize LCD (with no PWM support).
        super(Adafruit_CharLCDPlate, self).__init__(LCD_PLATE_RS, LCD_PLATE_EN,
            LCD_PLATE_D4, LCD_PLATE_D5, LCD_PLATE_D6, LCD_PLATE_D7, cols, lines,
            LCD_PLATE_RED, LCD_PLATE_GREEN, LCD_PLATE_BLUE, enable_pwm=False,
            gpio=self._mcp)

    def is_pressed(self, button):
        """Return True if the provided button is pressed, False otherwise."""
        if button not in set((SELECT, RIGHT, DOWN, UP, LEFT)):
            raise ValueError('Unknown button, must be SELECT, RIGHT, DOWN, UP, or
LEFT.')
        return self._mcp.input(button) == GPIO.LOW
```