Course notes, module 6 week 10
I/O interfacing to sensors and actuators continued

Kjeld Jensen kjen@sdu.dk

Agenda

1. Recap of the exercises from previous module
2. Preparation for next module
3. Exercises
4. Optional exercises

1. Recap of previous module

In the previous module Henrik focused on the git version control system. We will briefly review the results and findings of this module in class, and the first exercise today is to start using git within your teams.

2. Preparation for next module

Upon completing this module we will leave I/O interfacing for now, so make sure that you have completed all exercises. No other preparations will be needed for the next module.

3. Exercises

These exercises will focus on the interfacing of microcontrollers to a linux embedded system. The programming of firmware for the microcontrollers is less relevant in this context, so you will be provided example code that is ready to use.

3.1 Create a git repository for your team

In module 5 you learned about the git version control system. Now it is time to start utilizing the benefit of git.

Create a git repository for your team and make sure that everybody can access it.

Install git on the RPi

For all future exercises make sure that your scripts etc. is added to the git repository. Make a directory structure for this course that suits your team, it could be something like:

```
/module_6/exercise_3.2_pico_blinking_led
/module_6/exercise_3.3_pico_internal_temperature
...
```

3.2 Pico blinking LED and serial "hello world"

In the following exercises you will be using the breadboard with the Raspberry Pico installed. You will find a datasheet and also a pdf book about the Pico on itslearning under the LEO1 kit plan.

We will begin by running an embedded version of the "hello world" app i.e. a blinking LED and serial text output to make sure that we can transfer scripts to the Pico and execute them.

Before moving on we need to add your current user on your linux desktop environment and on the RPi to the `dialout` group which provides permnission to serial communication without having to use the `sudo` command each time.

```
$ sudo usermod -a -G dialout your_user_name
```

After this command you may need to logout and login again for the change to take effect.

We then need to ensure that the Pico has a MicroPython bootloader installed. This will also clear the Pico of any previous code.

Connect the MicroUSB port to your linux desktop environment while holding down the push button on the Pico at the same time. After a few seconds you should see a flash drive named RPI-RP2

Copy a Pico MicroPython bootloader onto this drive. You will find a bootloader on itslearning next to this document. When the copying is finished, the RPI-RP2 drive will be removed, and your Pico is ready to run MicroPython scripts.

We will install the Thonny Python IDE onto your linux desktop environment to edit and transfer MicroPython scripts to the Pico. There are other means to do this including using the shell, but Thonny is simple and easy to use for now. There are various ways to install Thonny as described at https://thonny.org The following worked well under Ubuntu 20.04 and ensure that you have the newest available version.

```
 $ bash <(curl -s https://thonny.org/installer-for-linux)
```

Launch the Thonny IDE and then re-connect the Pico to your linux desktop environment. You should now be able to transfer and run scripts on the Pico. If not, then look at the bottom right of the Thonny IDE. It should say "MicroPython (Raspberry Pi Pico)". If not, then click directly on the text an select this using the drop-down menu that appears.

The example code below will flash the built-in LED and print "hello world" every 2 seconds. Try to run it on the Pico.

```
from machine import Pin
import utime

led = Pin(25, Pin.OUT)

while True:
      led.toggle()
      print ("hello world")
      utime.sleep(2.0)
```

Now connect an external LED from the LEO1 kit to the Pico GPIO pin 16 via the breadboard and modify the script to use this GPIO pin. Remember to put the LED in series with a 220 Ohm resistor and that the LED has a an anode (+) and a cathode (-), see LEO1 kit pdf for more information.

3.3 Pico internal temperature

The example script below will flash the LED and print the Pico internal temperature in degrees Celcius every 2 seconds. Run the script on the Pico.

```
import machine
import utime

led = machine.Pin(16, machine.Pin.OUT)

conversion_factor = 3.3 / (65535)
sensor_temp = machine.ADC(4)

while True:
      led.toggle()
      reading = sensor_temp.read_u16() * conversion_factor
      temperature = 27 - (reading - 0.706)/0.001721
      print (temperature)
      utime.sleep(2.0)
```

Configure this script to run on the Pico immediately after power on (connecting the MicroUSB cable). To do so it must be saved in a file name `main.py` on the Pico.

When you have observed in Thonny that the Pico sends the temperature via the serial port, then close down Thonny.

Now connect the Pico to a USB port of the RPi.

To determine the name of the serial device associated with the Pico, disconnect the MicroUSB and then reconnect it. Then run the command below to show the latest messages from the kernel:

```
$ dmesg
```

Look at the bottom for the serial device. It will likely be `/dev/ttyACM0`

Now write a shell script on the RPi that waits until next time the Pico sends a temperature string via the serial port, reads the temperature and echo's this to the terminal. Hint, this should be two simple lines, one for reading a line of text from the serial device into a variable and another line for echoing that variable to the terminal.

3.4 Sending temperature data from the Pico to an external IoT data platform via the RPi

Using the shell script from 3.3 we will create a script that each minute sends Pico temperature data to an external IoT data platform.

We will use the ThingSpeak.com platform by MathWorks. Please create a personal account at the website.

On ThingSpeak create a channel named `Pico`

Under this channel set the name of Field 1 to `PicoTemp`

Save the channel.

Using the linux command `curl` perform a POST request using the below test data. Remember to replace the API key with the one listed under your channel.

```
POST https://api.thingspeak.com/update
     api_key=8CH8WRDBSW5TA66W
     field1=20
```

When you have tested that you can post data to ThingSpeak using `curl` and have it presented as a graph (and much more) then make a shell script that is called from /etc/crontab each minute. The shell script should run the command from exercise 3.2 and pass on the temperature to ThingSpeak field1.

As the graph starts building itself for each minute of data try to gently place a fingertip on the microntroller (the biggest black chip) and notice via the ThingSpeak graph that the temperature rises.

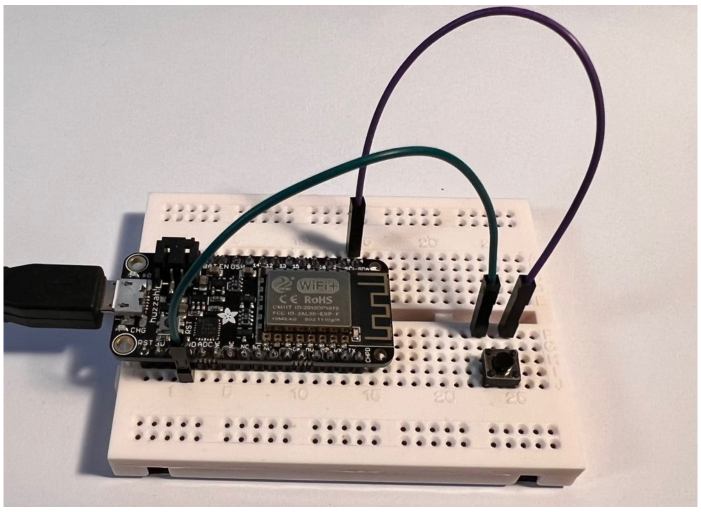
Take some time to explore ThingSpeak. It is quite impressive what you can do with IoT data in a user-friendly way using such a platform. Ultimately ThingSpeak facilitates that you can execute Matlab scripts to process incoming data.

3.5 Adafruit ESP8266 sending data to the RPi Apache2 webserver

In this exercise we will switch from the Pico to the Adafruit Feather HUZZAH ESP8266 microcontroller which has Wifi built-in. The purpose of the exercise is to have the ESP8266 count the number of times a button is pushed and send this to the RPi webserver as a POST request.

The ESP8266 only supports Wifi 2.4 GHz so if you have created a RPi wifi access point with only 5 GHz support, then you need to change that configuration. Hint: In module 3 exercise 3.2 you may have set `hw_mode=a` in `/etc/hostapd/hostapd.conf` You need to set `hw_mode=g` and `channel=1` (or 6 or 11) to switch to 2.4 GHz.

Add a push button to the ESP8266 breadboard between Pin 4 and GND.



Install the Arduino IDE on your linux desktop environment.

Install the ESP8266 Board Package in the Arduino IDE by adding the URL below to the Additional Boards Manager URLs under File - Preferences.

```
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

Under `Tools - Board - Boards Manager` search for `ESP8266` and install this Boards package.

Then under `Tools - Board - ESP8266 Boards` select `ESPino (ESP-12 Module)`

Open the Arduino `esp8266_post_count.ino` file available at itslearning

Change the WIFI_SSID and WIFI_PASSWORD to your current RPi wifi access point configuration.

Under `Sketch` choose `Upload` to upload the firmware to the ESP8266.

Copy the file `log.php` to `/var/www/html`

This php file will receive POST requests from the ESP8266 and store the data into a text file named `log.txt` in the same directory. To see the post requests run the following command:

```
$ tail -f /var/www/html/log.txt
```

4. Optional exercises

4.1 Adding another sensor to the Pico

Extend the exercises 3.3 and 3.4 by adding a the photoresistor to an analog input of the Pico and then send both the Pico temperature and ambient light measurement to ThingSpeack. This adds some complexity as you have to deal with two sensor values from the Pico MicroPython script to the curl command.

4.2 Controlling a servo using the Pico

Try connecting the servo data (orange) to the Pico pin 15, servo plus (red) to the Pico 5V and servo GND (brown) to Pico GND. Then run the script below to control the servo.

```
from machine import Pin, PWM
import utime

# defines
SERVO_NEUTRAL = 1500000
SERVO_MIN = 1000000
SERVO_MAX = 2000000

# pins
```

```
led = Pin(25, Pin.OUT)
pwm = PWM(Pin(15))

# setup pwm for servo
pwm.freq(50)
pwm.duty_ns(SERVO_NEUTRAL)

while True:
    led.toggle()
    pwm.duty_ns(SERVO_MIN)
    utime.sleep(0.5)

    led.toggle()
    pwm.duty_ns(SERVO_NEUTRAL)
    utime.sleep(0.5)

    led.toggle()
    pwm.duty_ns(SERVO_MAX)
    utime.sleep(0.5)

    led.toggle()
    pwm.duty_ns(SERVO_NEUTRAL)
    utime.sleep(0.5)
```

The script above moves the servo in a fixed pattern. Modify the script to take the servo position angle in degrees [0;180] from the serial port. Then write a shell script that takes an angle as parameter and sends this angle to the Pico.

4.3 Controlling a LED on the ESP8266 via the RPi Apache2 webserver

The current version of log.php responds `POST_OK` to all requests. Modify log.php to respond with either `LED_ON` or `LED_OFF` depending on the content of a file `led.ctrl` that you place in `/var/www/html`

If `led.ctrl` contains the number `0` then log.php should respond `LED_OFF`

If `led.ctrl` contains the number `1` then log.php should respond `LED_ON`

Now modify the `esp8266_post_count.ino` firmware to support reading this response from the log.php script (hint it already does that and prints the output to the serial port). Use the response to control a LED on the ESP8266, either the built-in or one that you connect to a GPIO pin via a 220 ohm resistor.