Raspberry PI (RPI) 3 Model B

EEE3096S RPi LAB MANUAL

Author: Do Yeou Ku

University of Cape Town

Revised for EEE3096S by S. Winberg and O. Konan, June 2018

Practical 4: RPI-3B SPI and Interrupt (RPI-Python Serial Com. and RPI-Python Interrupt)

The objective of practical 4 is to develop code for a simple environment monitor system. The system includes a temperature sensor (MCP9700A), a LDR (1K), a pot (1K), a MCP3008 IC and three switches.

The monitor works as follow:

- By default, the system continuously monitors the sensors every 500ms using this format:

Time	Timer	Pot	Temp	Light
10:17:15	00:00:00	0.5 V	25 C	10%
10:17:20	00:00:05	1.5 V	30 C	68%

- The *reset* switch resets the timer and clean the console
- The *frequency* switch changes the frequency of the monitoring. The possible frequencies are *500ms, 1s, 2s*. The frequency must loop between those values per event occurrence.
- The *stop* switch stops or starts the monitoring of the sensors. *The timer is not affected by this functionality.*
- The *display* switch displays the first five reading since the *stop* switch was pressed.

1. Answer the following questions (15):
    a. Explain the SPI communication protocol with a timing diagram. (2)
    b. Define interrupt and threaded call-back in the context of an embedded system. (2)
    c. Write a function that converts a 10-bit ADC reading from the potemtiomenter to a 3V3 limited voltage output. (2)
    d. Write a function that converts a 10-bit ADC reading from the temperature sensor to a reading in degree Celsius (*Have a look at the datasheet*). (3)
    e. Write a function that converts a 10-bit ADC reading from the LDR to a percentage representing the amount of light received by the LDR. (2)
    The flashlight from a smartphone could be used as the maximum amount of light received by the LDR.
    f. Draw a flowchart of the system. (4)
2. Demonstration. (15)
    a. Default (3)
    b. Reset (2)

    c. Frequency (3)
    d. Stop (3)
    e. Display (4)
3. Git (only for negative marking)
    a. You must have a git repository containing meaningful commit messages. (3 per group member) (-1)
    b. Each member must work on different branches, different from the master branch. (-1)
    c. Your final submission must be on your master branch. (-1)
    d. Each member must merge their branch to the master branch. (-1)
    e. Your repository must contain any files used for the assignment.

At the end of practical 4, you are to submit one .zip file containing all files used (datasheets, python code, answers to the question in pdf format) on Vula.

Make sure that your .git file is included in your zip file.

The zip file must have the following format
prac_<prac_number>_<std_number1>_<std_number2>.zip.

Only one submission per group.

Contents

I. RPI-Python Serial Com.

Imagine you have 6 analog sensors that you must read to monitor an environment. Now imagine that the microcontroller you are supposed to use can only take up to 2 ADC channels. Without the use of an external device (which can handle 6 ADC channels), your microcontroller will not be able to the task, you will need more microcontrollers otherwise. Let's talk about the external device mentioned before. The device needs to communicate with your microcontroller and share certain information such as the reading from the analog sensor. How does it do it? By using defined communication protocol such as SPI, I2C, UART, etc.

In this practical we will focus on SPI using the Raspberry Pi. The RPi has libraries that remove the complexity of implementing the communication protocol.

For example, Figure 2 shows an example code for setting up SPI on an STM microcontroller using the Standard Peripheral Library (used in the previous embedded system course)

```
void mySPI_Init(void){

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);

    SPI_InitTypeDef SPI_InitTypeDefStruct;


    SPI_InitTypeDefStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitTypeDefStruct.SPI_Mode = SPI_Mode_Master;
    SPI_InitTypeDefStruct.SPI_DataSize = SPI_DataSize_8b;
    SPI_InitTypeDefStruct.SPI_CPOL = SPI_CPOL_High;
    SPI_InitTypeDefStruct.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_InitTypeDefStruct.SPI_NSS = SPI_NSS_Soft;
    SPI_InitTypeDefStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
    SPI_InitTypeDefStruct.SPI_FirstBit = SPI_FirstBit_MSB;

    SPI_Init(SPI1, &SPI_InitTypeDefStruct);


    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE ,
    ENABLE);

    GPIO_InitTypeDef GPIO_InitTypeDefStruct;

    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7 | GPIO_Pin_6;
    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOA, &GPIO_InitTypeDefStruct);

    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_Init(GPIOE, &GPIO_InitTypeDefStruct);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);

    GPIO_SetBits(GPIOE, GPIO_Pin_3);


    SPI_Cmd(SPI1, ENABLE);

}
```

Figure 2: SPI initialisation on an STM microcontroller using the standard peripheral library

While this is all the work you have to do using hardware SPI with the spidev library (recommended) on the Raspberry Pi.

```
# Open SPI bus

spi = spidev.SpiDev()                    # create spi object

spi.open(0,0)                            # initialise SPI
```

A. SPI on RPI

To establish SPI communication protocol on RPI, SPI interface needs to be enabled. Use the command below to enter configuration menu.

```
sudo raspi-config
```

Go to advanced options > SPI> Yes, to enable SPI on Raspberry PI.

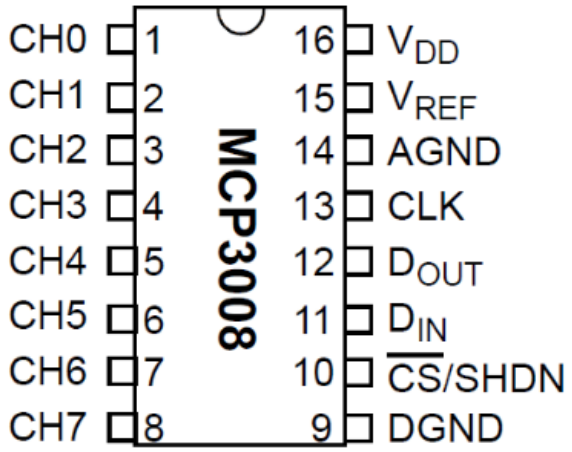

Figure 3 MCP3008 pinout

It is crucial that you connect the chip correctly. I hope that you have a skill to read datasheet.

This document will include two ways of communicating with MCP3008, one using *adafruit* library another using *spidev* library.

To install *adafruit* MCP3008 library, run the following commands: (Already done if you are using the image on Vula)

```
$ sudo apt-get update

$ sudo apt-get install build-essential python-dev python-smbus git

$ cd ~
```

```
$ git clone https://github.com/adafruit/Adafruit_Python_MCP3008.git

$ cd Adafruit_Python_MCP3008

$ sudo python setup.py install
```

If you prefer installing from the PyPI, execute the following commands.

```
$ sudo apt-get update

$ sudo apt-get install build-essential python-dev python-smbus python-pip

$ sudo pip install adafruit-mcp3008
```

```
#!/usr/bin/python


import RPi.GPIO as GPIO

import Adafruit_MCP3008

import time

import os


GPIO.setmode(GPIO.BCM)

# pin definition

SPICLK = 11

SPIMISO = 9

SPIMOSI = 10

SPICS = 8


GPIO.setup(SPIMOSI, GPIO.OUT)

GPIO.setup(SPIMISO, GPIO.IN)

GPIO.setup(SPICLK, GPIO.OUT)

GPIO.setup(SPICS, GPIO.OUT)
```

```
mcp    =    Adafruit_MCP3008.MCP3008(clk=SPICLK,    cs=SPICS,    mosi=SPIMOSI,
miso=SPIMISO)

# global variable

values = [0]*8


while True:

       for i in range(8):

            values[i] = mcp.read_adc(i)

       # delay for a half second

       time.sleep(0.5)

      print values
```

To install *spidev* library, run the following commands: (Already done if you are using the image on vula)

```
$ sudo apt-get update

$ sudo apt-get install build-essential python-dev python-smbus git

$ cd ~

$ git clone https://github.com/doceme/py-spidev.git

$ cd py-spidev

$ sudo python setup.py install
```

```
#!/usr/bin/python


import spidev

import time

import os

import sys


# Open SPI bus
```

```
spi = spidev.SpiDev()                    # create spi object
spi.open(0,0)
# RPI has one bus (#0) and two devices (#0 & #1)

# function to read ADC data from a channel
def GetData(channel):                    # channel must be an integer 0-7
  adc = spi.xfer2([1,(8+channel)<<4,0])   # sending 3 bytes
  data = ((adc[1]&3) << 8) + adc[2]
  return data

# function to convert data to voltage level,
# places: number of decimal places needed
def ConvertVolts(data,places):
  volts = (data * 3.3) / float(1023)
  volts = round(volts,places)
  return volts

# Define sensor channels
channel = 0
# Define delay between readings
delay = .5

try:
      while True:
        # Read the data
        sensr_data = GetData (channel)
        sensor_volt = ConvertVolts(sensor_data,2)
```

```
        # Wait before repeating loop
        time.sleep(delay)
except KeyboardInterrupt:
        spi.close()
```

II. RPI-Python Interrupt

A. External Interrupt

Threaded call-back is simply another term for an interrupt handler. Following code explains multiple threaded call-back system with interrupt triggered by input switches.

```
#!/usr/bin/python
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
switch_1 = switch_1_pin_number
switch_2 = switch_2_pin_number
switch_3 = switch_3_pin_number

# switch 1 & switch 2: input – pull-up
GPIO.setup(switch_1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(switch_2, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# swtich 3: input – pull-up
GPIO.setup(switch_3, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# function definition: threaded callback
def callback1(channel):
      # put code here
def callback2(channel):
      # put code here
```

```
# Under a falling-edge detection, regardless of current execution
# callback function will be called
GPIO.add_event_detect(switch_1,       GPIO.FALLING,       callback=callback1, bouncetime=200)


GPIO.add_event_detect(switch_2,       GPIO.FALLING,       callback=callback2, bouncetime=200)
# 'bouncetime=200' includes the bounce control
# ‘bouncetime=200’ sets 200 milliseconds during which second button press will be ignored.
# to remove: GPIO.remove_event_detect(port_number)
try:
      GPIO.wait_for_edge(switch_3, GPIO.RISING)
except KeyboardInterrupt:
      GPIO.cleanup() # clean up GPIO on CTRL+C exit


GPIO.cleanup() # clean up GPIO on normal exit
```

For more information go to https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/