# General Information of Lab Instrument and Platform

Farhang Nemati, March 2017

# Introduction

In the lab we use Raspberry Pi B+ platform for lab assignments. Raspberry Pi is a cheap, small and general purpose computer mainly used for programming embedded applications. It was originally made for education purposes, however, it soon became a popular platform among hobbyists and those who wished to learn developing embedded applications. There is a big community for programming on Raspberry Pi and there are interesting projects and works available online. Usually, a distribution of Linux (Raspbian) is used as operating system for Raspberry Pi. This makes it quite useful since programs can be written in a wide range of languages, e.g., C, C++, Java, and Python.

The operating system we have installed on these platforms in the lab is Real-Time Linux (Raspbian compiled with Real-Time settings). The programming language to be used is C. We will use Visual Studio for programming and debugging environment. To be able to easily write programs, compile, run, and debug them using Visual Studio, we have installed VisualGDB which is a convenient add-on for Visual Studio. One of the usages of VisualGDB is programming and debugging C/C++ Linux applications for Raspberry Pi and other Linux platforms, Android apps etc.

# Raspberry Pi B+

The model B+ of Raspberry Pi (Figure 1) has 40 General Purpose Input/Output (GPIO) pins. GPIO pins are used for input/output purposes. However, not all of the 40 pins can be used for input/output; some of them are

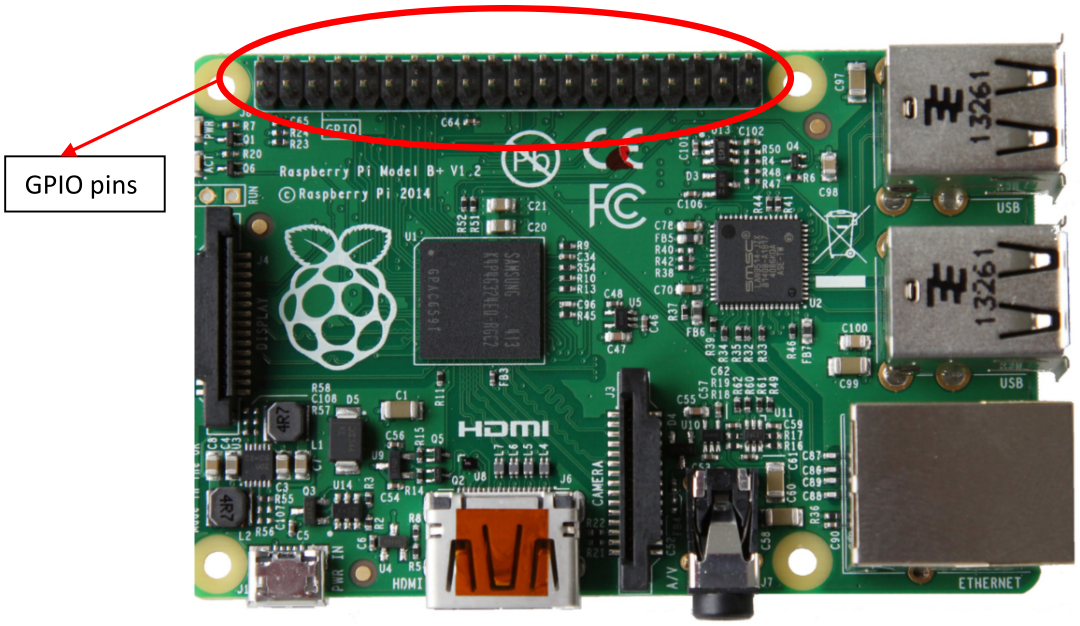

Figure 1: Raspberry Pi B+

for supplying power for devices connecting to Raspberry Pi; 8 pins for GND (ground), 2 pins to supply 3.3 Volt, and 2 pins to supply 5 Volt. As shown in Figure 1 it is difficult to recognize the pins, therefore we have connected the pins to a board where each pin is shown by a label next to it (Figure 2). Except the power supply pins the rest of GPIO pins can be used for either output or input purposes. This means that each of these pins can programmatically be set to either input or output. To ease working with input/output in the lab we have provided necessary functions written in C which can be downloaded from Blackboard. You have to be very careful when using GPIO; failure in connecting and using the pins might destroy the pins. Always ask the lab assistant for help to assist you connecting GPIOs to other devices. In the following section it is explained how to use the provided C code to work with GPIO.

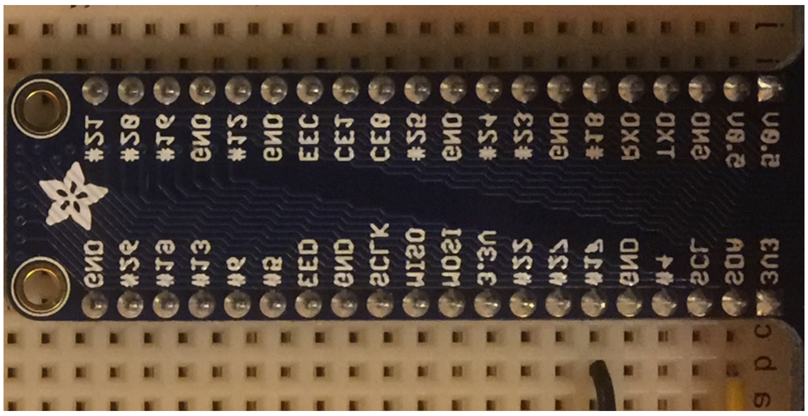

Figure 2: GPIOs with labels in the

There is a lot of online information and interesting projects done by Raspberry Pi. For more details about Raspberry Pi visit its official web site at www.raspberrypi.org.

## Specifications

- 700MHz BCM2835 CPU
- 512 MB SDRAM
- 10/100 Ethernet port
- Full-size HDMI port
- 4 USB ports
- Micro SD slot
- Combined 3.5mm audio jack and composite video output
- Camera interface (CSI)
- Display interface (DSI)
- 40 pins GPIO header

# Starting up Shutting down Raspberry Pi in the Lab

## Starting up

Connect the electronic devices that you are supposed to work with before turning the Pi on (connecting it to electricity). Make sure that you have connected the Pi to network using the network cable. When ready connect the Pi to electricity to turn it on. RTLinux will boot and after several seconds the Pi will be ready to use.

### Shutting down

When you are finished with the Pi shut down it properly. Never turn it off by unplugging it from the electricity before shutting it down. To shutdown send the following terminal command (for example using PuTTY) to the Pi:

```
sudo shutdown -h now
```

## Using GPIO

To use GPIOs always use the C functions in the provided C files (*piodirect.h* and *piodirect.c*). For each lab when you create a project in Visual Studio first download *piodirect.h* and *piodirect.c* from Blackboard ("Kursmaterial/Labs/Helper Code") and add them to the project. The details on how to create a project for Raspberry Pi in Visual Studio will be explained in the instructions for the first lab.

### Setting up GPIO

Always call `gpioSetup()` at the beginning of function main() in any project:

```
int main(int argc, char *argv[])
{
	gpioSetup ();
	...
}
```

### Setting a GPIO pin as input or output

For setting a pin as an input or output call the following function:

```
GPIO create(enum Pin pin, int direction)
```

Put the returned GPIO type in a variable of type GPIO. From that point you will use this variable to work with the pin.

Parameter *pin* is the name of pin (written on the label next to the pin) followed by "_". For example GPIO pin #17 is entered as _17

The direction sets the pin to either input or output; enter IN_PIN if the pin is to be an input, and enter OUT_PIN if the pin is going to be an output pin.

**Examples:**

```
GPIO gpio4 = create(_4, OUT_PIN); /*sets pin #4 (see Figure 2) as output pin*/
GPIO gpioScl = create(_SCL, OUT_IN); /*sets pin SCL (see Figure 2) as input pin*/
```

### Writing a value to an output GPIO pin

The value of an output pin can be set to high (1) or low (0). When it is high the voltage of the pin is around 3.3 Volt and when it is low its voltage is around 0 Volts (GND). To put an output pin to high call the following function:

```
int onOff(GPIO gpio_, int onOff);
```

Parameter onOff has to be ON to put the pin on high and OFF to put the pin on low.

**Example:**

```
GPIO gpio17 = create(_17, OUT_PIN);
onOff(gpio17, ON); /*sets pin #17 to high*/
onOff(gpio17, OFF); /*sets pin #17 to low*/
```

## Reading the value of an input GPIO pin

To read a value from a GPIO pin that is set as input, call the following function:

```
int readIn(GPIO gpio_);
```

If the return value is 1 it means the input is high, if the return value is 0 the value of input pin is low.

**Example:**

```
int value;

GPIO gpio4 = create(_4, IN_PIN);
value = readIn(gpio4);
```

## Using a GPIO pin for buttons and switches

When using a button with Raspberry Pi, one pin of the button has to be connected to a GND and its other pin has to be connected to an input GPIO pin. First of all the GPIO pin has to be created as input pin and then function `pullUpDown(GPIO gpio_, int pud)` has to be called on that pin. The value of parameter *pud* can be either PUD_UP or PUD_DOWN. If PUD_UP (pull up) is given as the value of *pud* the GPIO pin by default is high (1) and when the button is pushed the input value becomes low (0). On the other side if PUD_DOWN (pull down) is given, the default value of the GPIO pin by default is low and when the button is pushed the input value of the GPIO pin becomes high.

**Example 1:**

```
GPIO gpio17 = create(_17, IN_PIN);

pullUpDown(gpio17, PUD_UP);

while (1)
{
    if (readIn(gpio17) == 0)
    {
        printf("Button pushed!\n");
        break;
    }
    else
    {
        printf("Button released!\n");
    }
    usleep(20000);
}
destroy(gpio17);
```

There is a function that does the work of both create() and pullUpDown() together. It is preferred to use this function for buttons:

```
GPIO createWithPullUpDown(enum Pin pin, int up);
```

**Example 2:**

```
GPIO gpio17 = createWithPullUpDown(_17, PUD_UP);
while (1)
{
    if (!readIn(gpio17))
    {
        printf("Button pushed!\n");
        break;
    }
    else
    {
        printf("Button released!\n");
    }
    usleep(20000);
}
destroy(gpio17);
```

**Example 3:**

```
GPIO gpio17 = createWithPullUpDown(_17, PUD_DOWN); /*Notice that it is Pull Down!*/
while (1)
{
    if (readIn(gpio17))
    {
        printf("Button pushed!\n");
        break;
    }
    else
    {
        printf("Button released!\n");
    }
    usleep(20000);
}
destroy(gpio17);
```

## Unset a GPIO pin

When you are finished working with a pin and you don't need it anymore call the following function (always do it!):

```
void destroy(GPIO gpio_);
```

**Example:**

```
destroy(gpio17);
```