CMPE-240 Laboratory Exercise 1

Blinky

Christian Greaves
Performed: Feb 9th, 2017
Submitted : Feb 20th, 2017

Lecture Section: 01
Professor: Alessandro Sarra
TA: David Desrochers, Michael Baumgarten

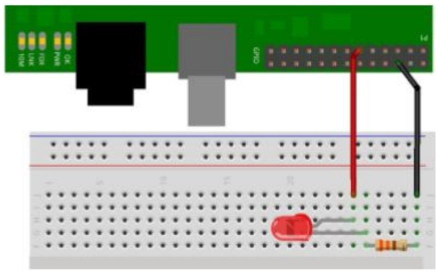
Abstract

Engineering Fundamentals of Computers Systems: Lab 1 consists of three major parts and is designed as a quick introduction to how the Raspberry Pi 3 functions on a basic level, and prepare for future, more complex Lab assignments. The first part entails proving that the Raspberry Pi 3 can boot an SD card with the Raspberry Pi Debian variant. The second part of the lab is to make the LED circuit on a breadboard, connect it to the Raspberry Pi and make the LED blink with 1 second intervals. The third and final portion of the lab is to set up a Visual Studio project to prove an ability in using breakpoint debugging in an IDE. The results of the lab were successful, with a successfully blinking LED circuit being created while connected to the Raspberry Pi's GPIO3 pin, and a Visual Studio project being successfully debugged.

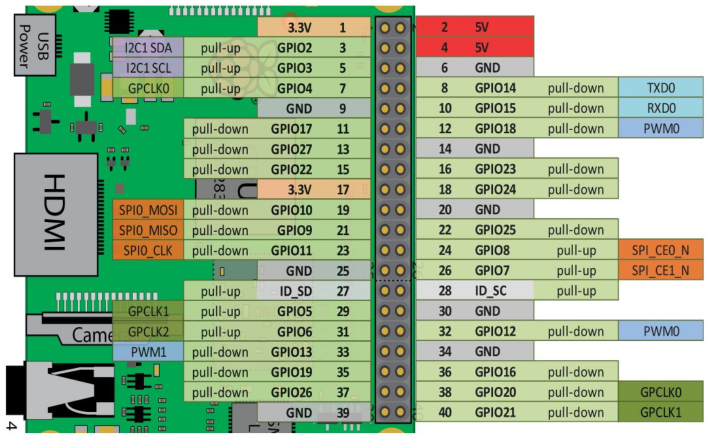
Design Methodology

Lab 1 does not have much of a design methodology behind it due to the nature of the lab being so short, structured and introductory.

Installing the Raspberry Pi Debian is done by flashing the downloaded image file onto an SD Card using a program such as Win32 Disk Imager, and then going through the process of logging in via the provided USB serial adapter and PuTTY, or connecting an HDMI connected monitor to the port on the Pi and proving that it can boot.

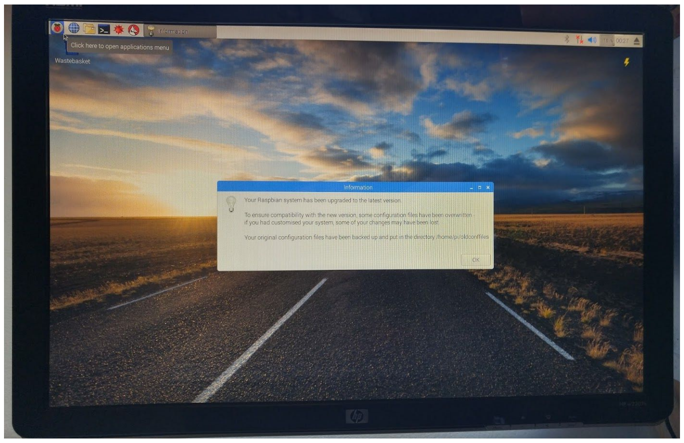


The LED circuit used is detailed with the provided figure to the right. The GPIO that the circuit connects to is not important, but for the purposes of this exercise, GPIO3 was used and coded for, and is therefore required. The GPIO Pin map used to identify the pin types and labels is provided to the left. The code portion of the lab is simply a toggle on the GPIO pin to turn on the LED, a delay for the appropriate time based on the CPU clock speed, another toggle on the GPIO pin to turn the LED off, and a second delay of the same time as the first, all contained within an infinite loop.


USB Power
HDMI
Came
3.3V 1 | 2 5V
I2C1 SDA pull-up GPIO2 3 | 4 5V
I2C1 SCL pull-up GPIO3 5 | 6 GND
GPCLK0 pull-up GPIO4 7 | 8 GPIO14 pull-down TXD0
GND 9 | 10 GPIO15 pull-down RXD0
pull-down GPIO17 11 | 12 GPIO18 pull-down PWM0
pull-down GPIO27 13 | 14 GND
pull-down GPIO22 15 | 16 GPIO23 pull-down
3.3V 17 | 18 GPIO24 pull-down
SPI0_MOSI pull-down GPIO10 19 | 20 GND
SPI0_MISO pull-down GPIO9 21 | 22 GPIO25 pull-down
SPI0_CLK pull-down GPIO11 23 | 24 GPIO8 pull-up SPI_CE0_N
GND 25 | 26 GPIO7 pull-up SPI_CE1_N
pull-up ID_SD 27 | 28 ID_SC pull-up
GPCLK1 pull-up GPIO5 29 | 30 GND
GPCLK2 pull-up GPIO6 31 | 32 GPIO12 pull-down PWM0
PWM1 pull-down GPIO13 33 | 34 GND
pull-down GPIO19 35 | 36 GPIO16 pull-down
pull-down GPIO26 37 | 38 GPIO20 pull-down GPCLK0
GND 39 | 40 GPIO21 pull-down GPCLK1

The final part of the lab, proving the ability to debug C code in an advanced IDE like Visual Studio, is done by creating a new project in Visual Studio, writing some basic "Hello World!" printing code, and the breaking on a point in the main function, so that when run with the debugger, will allow stepping through the code sequence to see how the code is performing.

Results and Analysis



Information

Your Raspbian system has been upgraded to the latest version.

To ensure compatibility with the new version, some configuration files have been overwritten - if you had customised your system, some of your changes may have been lost.

Your original configuration files have been backed up and put in the directory /home/pi/oldconffiles

Installing the Debian operating system was simple, and was completed successfully with no problems occurring. The successful LogIn screen is shown in the image to the right. The successfully installed system gives the ability to add new users/change passwords, connect to a network, among the many other things one can do with a desktop operating system.

Creating the LED circuit on a breadboard, given the circuit diagram/picture was also simple and was completed with no problems. The code for the blinking LED was done by activating the correct GPIO pin, setting it to the correct mode (in this case, Output 001), clearing the register by bit-shifting 1 into the correct bit position for the pin chosen (for GPIO3 it is 3 bits in), waiting on a delay by simply counting up from 0 to a given number that is dependent on the clockspeed of the CPU being used (in this case 900000 is just about 1 second on the current setup of the RPi3), setting the same bit for GPIO3 to 0, and then delaying again before looping back through the code again, starting at clearing the GPIO pin register. Everything in the coding the circuit building was completed successfully with everything going according to the design.

The final section of debugging a Visual Studio project was completed in about 5 minutes and, while part of the lab instructions, was simply added to verify an aptitude in using advanced IDE Debugger technology, and thus does not require analysis. That said, the code produced a terminal for output and the breakpoint made the code execution halt and was able to be stepped through.

Questions

No questions were asked in this lab.

Conclusions

The Lab experiment went smoothly and successfully, with only a few early moments causing complications. Originally when creating the circuit, GPIO17 was desired for use, however after some struggle, it became clear that GPIO17 was not going to work for some unknown reason. Refactoring the code for GPIO3 worked immediately, and the GPIO pin being used is unimportant. Coding also went smoothly with the only problems being a small misinterpretation of the RPi3 GPIO API. The confusion originated in how the GPSET and GPCLR functions were mapped to the pins, and how the bits were aligned for the pins. Now, going forward with experience, this shouldn't be a problem in the future. The Visual Studio Debug breakpoint portion was trivial and caused no problems. All information gathered together shows that the entire lab experiment was a success, and shows at least a basic understanding of the C language and how the Raspberry Pi 3 interaction via GPIO pins works.

Code

```
void delay(volatile uint32_t count)
{
   volatile uint32_t x = 0;    //Volatile flag moves the uint32_t to RAM
                              //to disable compiler optimization.
    while (x != count)         //Loop 'count' number of times.
   {
      x += 1;  //Simple implementation. Count up to
             //the number provided in the function call to waste time.
   }
}

int main()
{
   volatile uint32_t countValue = 900000; //Arbitrary number that, after testing,
                                        //turns out to be almost perfect for a 1 second
                                        //delay on a RPi3 proc @ stock clock speeds.

          // Select output mode and which pin to Drive
   gpio[GPFSEL0] |= (001 << 9);    //Output mode 001 put on pin 3
                                 //which is 9 bits over from the start
                                 //of the GPIO set 0.
   while (1)
   {
                    //toggle clear register for the chosen pin
                   // GPIO3 bits 11-9  bit 001 = output
      gpio[GPCLR0] |= (1 << 3);
                    //apply a delay
      delay(countValue);
                    //toggle set register for the chosen pin
      gpio[GPSET0] |= (0 << 3);
                    //apply a delay
      delay(countValue);
   }
   return 0;
}
```