INTERFACING WITH THE INTERNET & RASPBERRY PI DEVELOPMENT BOARD

Alex McKenley Taylor

EET, Southern Utah University, 2013

Submitted to the College of Science and Engineering at

Southern Utah University in fulfillment of the

requirements for a degree in Engineering Technology with an Emphasis in

Electronics and Computing.

Table of Contents

Introduction

The amount of time we spend online is increasing every day. The more we become connected to our digital lives, the more time we spend online. As of now, our outlet to consume digital information is almost exclusively through a computer screen. This project came to fruition for this very reason. The goal of this project is to integrate social networks with real world electronics as a means to extend the reach of the internet beyond the computer screen using various hardware and software components in conjunction with the power of the recently released Raspberry Pi computer.

This document describes in detail the software, hardware, various technologies used, as well as step by step instructions to implement a gmail notification system as well as a twitter monitoring program that will flash a desk lamp whenever a predetermined phrase is mentioned on twitter.

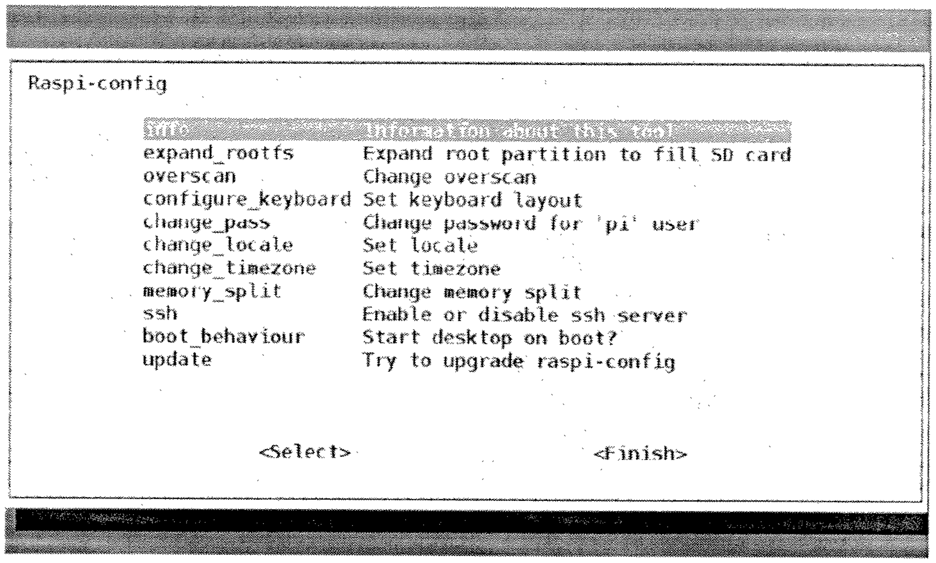
About the Raspberry Pi

This project utilizes the newly released Raspberry Pi computer as its central controller. For those who are unfamiliar with the Raspberry Pi, it is a single board computer about the size of a credit card with a plethora of IO capabilities such as HDMI, USB, Ethernet, Audio out, and a bank of GPIO pins. This small computer is powered by a 700MHz ARM CPU and 256MB of RAM, giving it enough power to run a full distribution of the Linux operating system. Despite its power, the Pi only uses 3.5 Watts of power, and costs only $35. Because of its low power consumption and cost, the Pi is a perfect candidate for this always-on system that will monitor our Gmail and Twitter accounts.

The two most important features of the Raspberry Pi utilized in this project are the Ethernet internet connection, and the onboard GPIO pins. The Ethernet port is an essential element for this project as other microcontrollers are unable to connect to the internet natively, or require expensive adapters to provide this capability. The GPIO pins allow us to control what will be our end devices, in our case, electromechanical relays switching 120VAC bulbs.

Connecting to the internet

The first step in this project is installing the Raspberry Pi's operating system, and connecting to this internet. To achieve this, first download the Debian OS disk image from http://raspberrypi.org and burn it to a blank SD card. For the scope of this tutorial I will not explain this process step by step, as it is well documented on the Raspberry Pi website. After you get the image installed, connect the Pi to a monitor via HDMI cable and a keyboard via USB to get the initial setup configured. Upon first boot, the Raspberry Pi configuration software will auto start, displaying various options as seen in Figure 1.



```
Raspi-config

          info                Information about this tool
          expand_rootfs       Expand root partition to fill SD card
          overscan            Change overscan
          configure_keyboard  Set keyboard layout
          change_pass         Change password for 'pi' user
          change_locale       Set locale
          change_timezone     Set timezone
          memory_split        Change memory split
          ssh                 Enable or disable ssh server
          boot_behaviour      Start desktop on boot?
          update              Try to upgrade raspi-config


                 <Select>                   <Finish>
```

Figure 1

For our purposes we need to configure the change_pass, memory_split, ssh, and boot_behaviour options. First, change_pass will simply have you enter a new root password. Then, under the memory_split option, set the video memory to the lowest setting, as we will not be utilizing the video after our initial configuration is complete. Next, enable SSH, and set the boot behavior to disable desktop on boot. These settings will allow us to connect to the raspberry pi via SSH, and also save system resources that would otherwise be wasted on video output. Once complete, you can disconnect the display and keyboard, and reboot the raspberry pi with only the network cable connected.

To verify your configuration is correct, attempt to connect to the Pi via SSH using another computer on the network. In my case I used the free software PuTTY to SSH into Pi from my windows laptop. On my network, the Pi's IP address was 10.0.0.2, however yours may be different, check your router's DHCP table in order to determine the Pi's address. Once connected, you can use the ping command to verify the Pi has internet access. For my test, I used the following command:

```
$ping raspberrypi.org
```

If you receive any replies from the ping command, you have successfully configured the Raspberry Pi, and are ready to move on.

Interfacing with the GPIO pins

Now that the Pi is connected to the internet, we need to take a look at how we will interface with the GPIO Pins. One disadvantage of the Raspberry Pi GPIO pins is that they are directly connected to the ARM CPU, meaning that if a user accidently shorts out a pin, or applies a voltage higher than the maximum rating of 3.3 Volts, the Pi could be irreversibly damaged or bricked. To account for this, I have designed a printed circuit board to protect the Pi by buffering the pins, and also provide additional functionality for future projects such as analog inputs and extra PWM outputs.

There are currently a few development boards available for purchase such as the Gertboard, developed by Gert van Loo, and the Pi Face breakout board, developed by the University of Manchester School of Computer Science. These boards are expensive to purchase (in some cases more expensive than the Pi itself) and also utilize complicated software libraries to communicate with the boards which can also limit their usefulness. My solution is cheap to build, easy to communicate with, and is accessible via a single python library that only needs to be imported with one line of code.

The next section will describe how the board was designed, and how the communication protocol functions between the board and the Pi. The following information will probably only be of use to someone trying to create their own design from scratch, or anyone who is simply interested in what is happening at a lower level.

Functionality of the Development Board

The development board was designed with four requirements in mind: It must protect the Raspberry Pi from accidental damage, provide analog inputs, provide PWM outputs, and finally provide a connection to an electromechanical relay board. It is also important to note that while this project in particular does not utilize all the functionality of the development board, these extra features will no doubt be required in future projects.

To protect the Raspberry Pi, I decided to utilize the 74HC244 Buffer IC. This chip takes an input signal with very low current requirements, and allows for an output current of up to 6mA on each pin (greater than the Pi's default current rating). In addition, this chip acts as a middleman between any accidents that may occur due to user error. This chip is very cheap and easy to find online, and can be powered by a variety of voltage levels, making it the ideal solution for my board. The datasheet for the 74HC244 chip can be found in Appendix B.

The task of providing analog inputs and PWM outputs was a little more challenging. After experimenting with various options, I concluded that the most effective way was to use a microcontroller communicating over a serial interface to the Raspberry Pi's UART GPIO pins. The microcontroller chosen was the Atmega328P (the same chip used in the popular Arduino boards) for ease of programming using the Arduino IDE as well as the wide range of IO provided by the chip. Relevant pages of the datasheet for this chip can be found in

Appendix C.

To get the communication with the microcontroller working, there were two pieces of code that needed to be written. The first being the code for the microcontroller, which must accept information on the serial port, perform the desired task, and return information to the Raspberry Pi. The second piece of code being the Raspberry Pi library, which must send a command to the microcontroller and accept a value back which the user can interpret.

An additional requirement for this serial communication protocol was that it needed to be fast. In order to be useful, the final product should be able to switch an LED output on the microcontroller faster than the human persistence of vision, or at least 48 Hz. To make this a reality, I came up with a protocol that only required a single byte of data to be sent over serial which could set pins as inputs and outputs, read values from analog channels, set digital pins on or off, and set a PWM value to an output pin. I succeeded in allowing control of all this functionality with only a single byte, save for one exception, setting a PWM pin, which requires an additional byte of data to carry the actual PWM value from 0-255. After selecting the fastest baud rate compatible or both devices (115200), I was able to achieve a solid 350 Hz frequency on an output led, surpassing my initial requirement. The code used in the microcontroller can be found in Appendix D.

The second piece of code used on the Pi was basically the opposite of the code used in the microcontroller. Written in Python, this code communicates with the microcontroller over serial, accepts a value back, and also addresses the available pins on the Raspberry Pi directly. This code serves as the only software needed to communicate with the board, and can be imported into any Python script using just a single line. This code can be found in Appendix E.

Lastly, to provide the interface to the electromechanical relay module (to be discussed in the next section) I decided to implement a somewhat unconventional solution. Ease of use for this system was always on my mind, and in order to connect the development board to the relay module, I did not want a bundle of jumpers, which would take quite some time to connect and disconnect, and I also didn't want to deal with awkward ribbon cables and their specific connectors. The solution I came up with was to use an everyday Ethernet cable to connect the relay module to the development board. Since Category 5 Ethernet cable has 8 separate connections, and is widely available, this would suit my requirements perfectly. To make this work, I only needed to set up RJ45 sockets on each device and make sure my connection matched on each end. This solution proved to be very effective, and allows for the Raspberry Pi to be mounted a reasonable distance from the relay module, as well as easily assembled and disassembled.

The schematic and board diagrams for the completed development board can be found in Appendix A of this document.

Constructing the Relay Module

Next, we need a standard wall outlet to plug our desk lamp into. Also, we must be able to switch power to this outlet on and off programmatically. To achieve this, I bought a cheap plastic electrical box made for 4 standard outlets, and mounted a small electromechanical relay board inside of the housing. The schematic for the relay module can be found in Appendix F. Figure 2 shows the relay module mounted at the bottom of the electrical box, as well as the standard wall outlets before mounting.

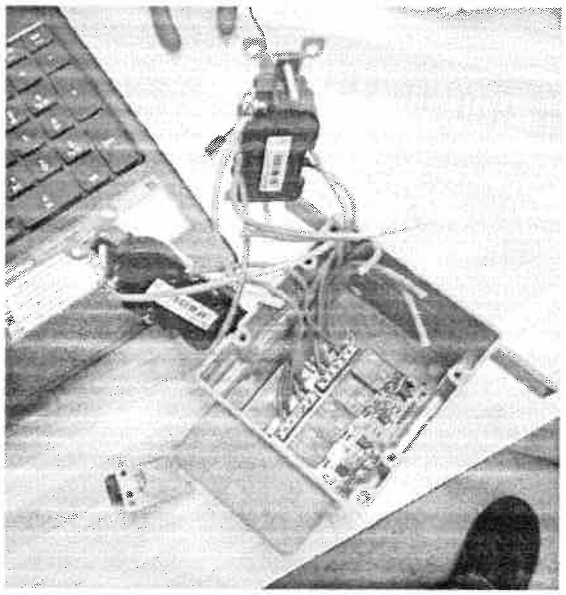


Figure 2.

The final step in construction of this control box is to provide an electrical connection to the development board. As stated in the previous section, I utilize a RJ45 connector for use with a standard CAT5 Ethernet cable. To make this connection, I designed a simple breakout board to connect the RJ45 connector to the pins on the relay board. The schematic for the breakout board can be found in Appendix G. All that remains is to securely mount the breakout board inside the electrical box, with an opening to allow the Ethernet cable to be connected. The end result with an Ethernet cable connected can be seen in Figure 3.

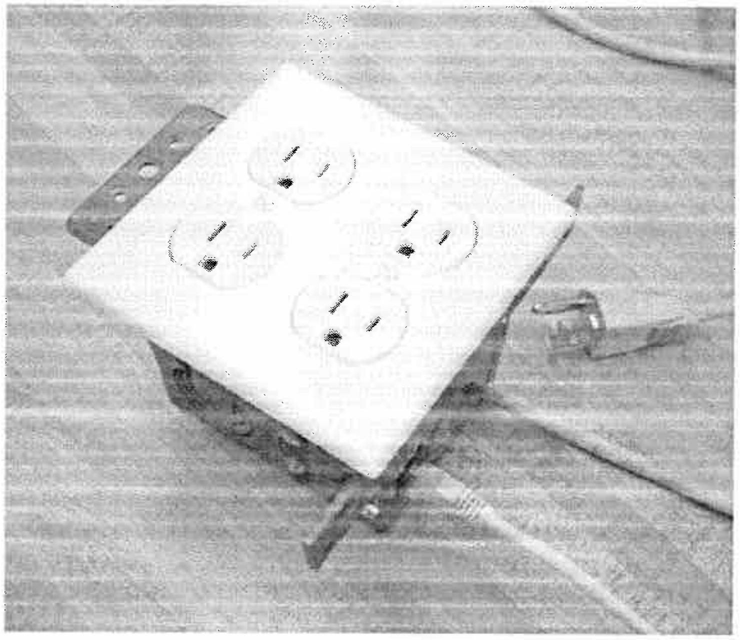


Figure 3.

Putting it all together

Now that we have the development board and the relay box functional, and the connection between them made, we can begin testing our setup. To test our functionality, first boot up the Raspberry Pi and connect the ribbon cable to the socket on the development board. Next, connect a standard Ethernet cable from the dev board to the relay box. Also, be sure to plug in a desk lamp (or equivalent) into outlet number 1.

Next, SSH into the Raspberry Pi, and open the Python interpreter. Once you are at the python prompt, import the development board library as follows:

```
python>> import dev_board as pi
```

Once you have imported the library, there are a number of ways to interact with our hardware. The available commands are as follows (more information about each function can be found in Appendix E):

```
setRelay(num,state)
setOutput(num,state)
pinMode(AorD, direction)
digitalWrite(num,state)
digitalRead(num)
analogRead(num)
analogWrite(num,value)
```

For testing purposes, we want to use the `setRelay()` command to turn on the light bulb.

```
python>> setRelay(0,1)
```

The first argument '0' of this statement tells the board to look at relay 0, and the second argument '1' tells the software to turn the relay on or 'high.' At this point you should hear the relay contacts click, and the lamp should illuminate. Now lets take a look at interfacing our system with some popular web apps.

Interfacing with the Internet

First, lets write a Gmail notifier program. When complete, this program will monitor your gmail account, and determine the number of unread email messages in your inbox. If this number is greater than zero, the lamp will illuminate, and upon opening the unread message, the light bulb will automatically turn off. We will use the feedparser python library to fetch the number of unread messages and the dev_board library to control the lamp. Below is some pseudo code that illustrates the function of the program. The actual code can be found in Appendix H.

```
//import the required libraries
//provide gmail login credentials
//loop forever
     //fetch number of unread messages using feedparser
     //compare number of new emails to a constant
     //if newmails > 0, turn lamp on
     //else, turn lamp off
```

```
          //time delay ~1 min or so
```

All that is needed is to change your username and password, then go ahead and run the script, or set the script to run automatically at boot time. At this point, you should have a gmail monitor that checks gmail every minute and notifies you of any unread messages. This setup could also be used with a neon sign or any custom light fixture for added wow factor.

For the next example, lets include support for the popular social networking site Twitter. Twitter is essentially a microblogging service that lets users send out publicly readable text messages of up to a maximum of 140 characters. For our project we want to monitor the entire twitter network for a specific word or phrase which, when used, will flash our lamp, informing us that our phrase has been mentioned by someone on twitter.

To make this happen, we need to use the publicly available Twitter API (Application Programming Interface) in order to communicate with the Twitter network. We will use the python-twitter library in order to make this easier. Once we have established the connection to twitter, all that remains is to perform a search query for a specific word or phrase, and the API will return a list of the latest tweets containing our string. We can then flash the lamp for each new tweet contained in the results list, and print the text of each tweet on the screen for us to read. You can see a simplified version of how this program works from the pseudo code below. The full code can be found in Appendix I of this document.

```
     //Import required libraries
     //Access the twitter API using the python-twitter library
     //loop forever
          //query twitter for a specific string such as
"SUUEET"
          //determine how many new results have been returned
          //flash the lamp for each result
```

```
        //Print each result on the screen
        //time delay ~1 min
```

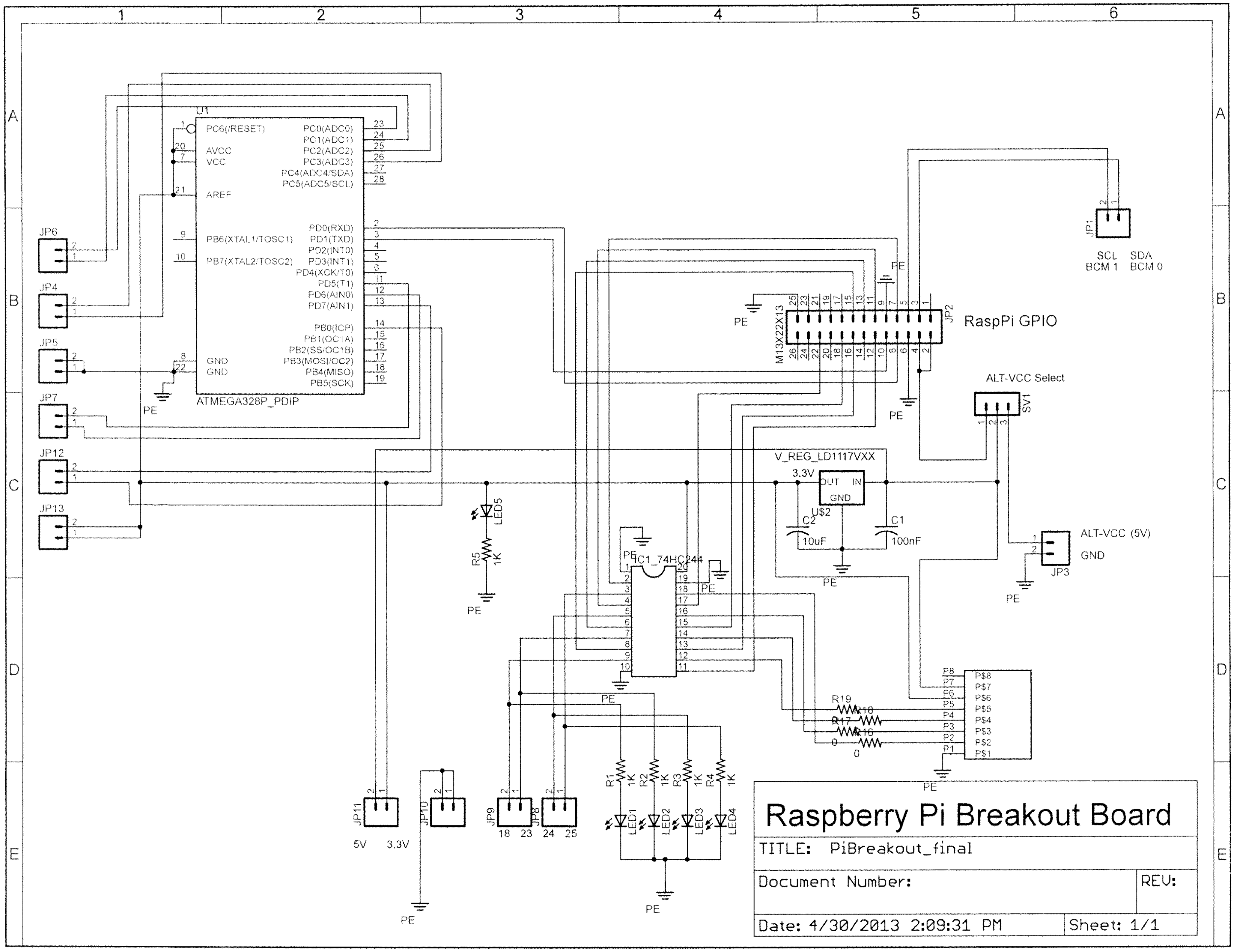
There you have it, at this point you should be able to run the python script and the lamp should flash every time your search query is mentioned on twitter by anyone in the world.

Conclusion

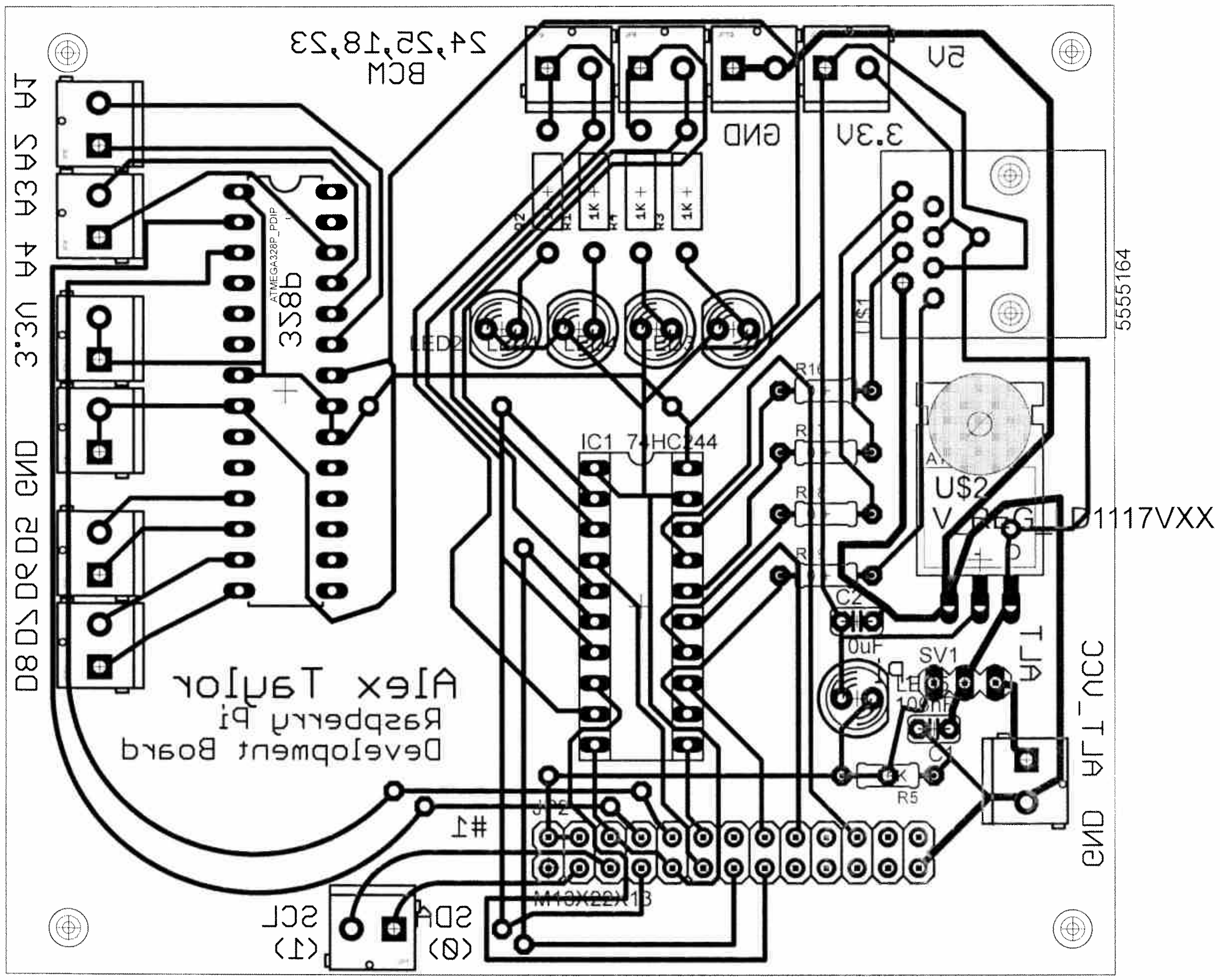
From this point on, the possibilities are endless. With only changes to code, we can accept analog inputs, set analog outputs, connect to any other social network via Python, create christmas light shows synced to music, or even a home automation system running on a web server hosted on the Raspberry Pi itself. Expanding the capabilities of the Raspberry Pi is becoming more important as its popularity increases, and I believe the need for simple development boards such as this will begin to rise. This project has been a great learning experience from the beginning, and I look forward to expanding its functionality and implementing this system in creative new ways in the future.

Appendix A

Development Board Schematic and Board Diagrams



4/30/2013 4:39:49 PM f=0.93 C:\Users\SUU\Downloads\Capstone\Schematics\PiBreakout_final.sch (Sheet: 1/1)


24,25,18,23
BCM
A1 A2 A3 A4 3.3V GND D5 D6 D7 D8
328b
ATMEGA328P_PDIP
GND
3.3V
5V
5555164
IC1 74HC244
U$2
V_REG_LD1117VXX
ALT_VCC
ALT
GND
SV1
C2
C1
R5
Alex Taylor
Raspberry Pi
Development Board
#1
JP2
M13X22X13
SDA (0)
SCL (1)

Appendix B

74HC244 Buffer IC Datasheet

FAIRCHILD
SEMICONDUCTOR™

September 1983
Revised May 2005

MM74HC244
Octal 3-STATE Buffer

General Description

The MM74HC244 is a non-inverting buffer and has two active low enables (1G and 2G); each enable independently controls 4 buffers. This device does not have Schmitt trigger inputs.

These 3-STATE buffers utilize advanced silicon-gate CMOS technology and are general purpose high speed non-inverting buffers. They possess high drive current outputs which enable high speed operation even when driving large bus capacitances. These circuits achieve speeds comparable to low power Schottky devices, while retaining the advantage of CMOS circuitry, i.e., high noise immunity, and low power consumption. All three devices have a fanout of 15 LS-TTL equivalent inputs.

All inputs are protected from damage due to static discharge by diodes to V_{CC} and ground.

Features

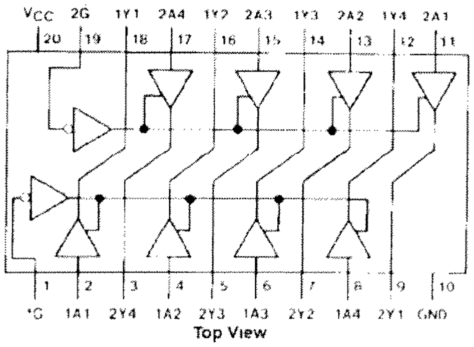
- Typical propagation delay: 14 ns
- 3-STATE outputs for connection to system buses
- Wide power supply range: 2–6V
- Low quiescent supply current: 80 µA
- Output current: 6 mA

Ordering Code:

Order Number	Package Number	Package Description
MM74HC244WM	M20B	20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
MM74HC244SJ	M20D	20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
MM74HC244MTC	MTC20	20-Lead Thin Shrink Small Outline Package (TSSOP), JEDEC MO-153, 4.4mm Wide
MM74HC244N	N20A	20-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Devices also available in Tape and Reel. Specify by appending the suffix letter "X" to the ordering code.

Connection Diagram



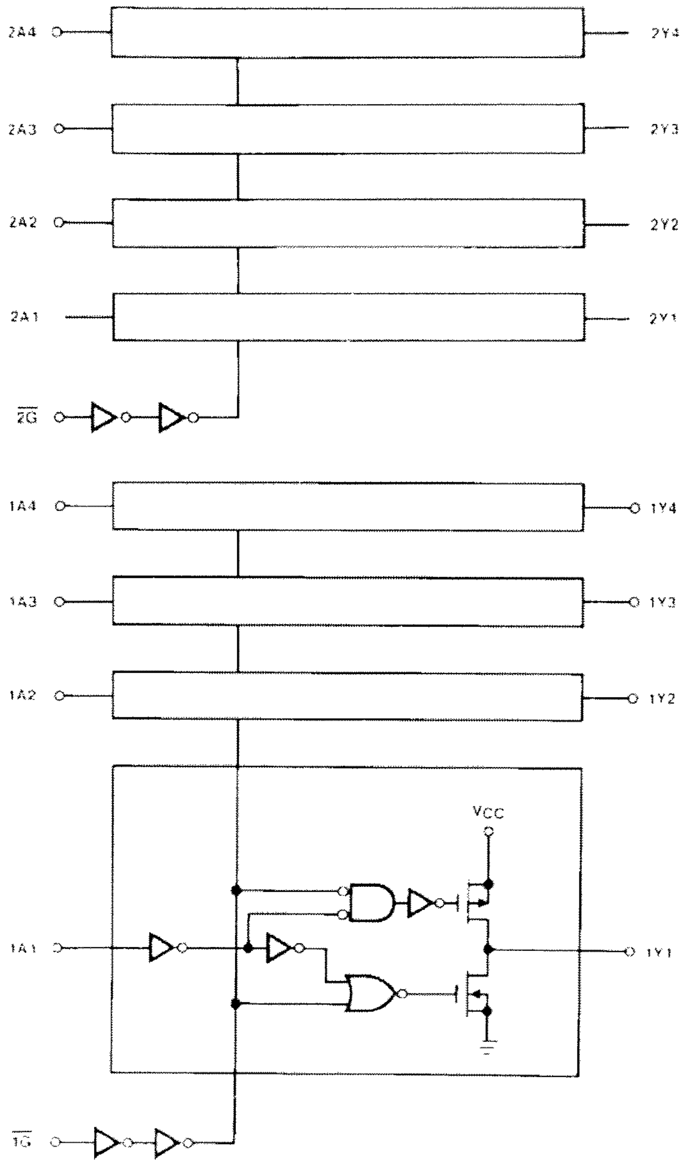
Top View

Truth Table

$\overline{1G}$	1A	1Y	$\overline{2G}$	2A	2Y
L	L	L	L	L	L
L	H	H	L	H	H
H	L	Z	H	L	Z
H	H	Z	H	H	Z

H = HIGH Level
L = LOW Level
Z = High Impedance

Logic Diagram



Absolute Maximum Ratings(Note 1)

(Note 2)

Supply Voltage (V_{CC})	0.5 to +7.0V
DC Input Voltage (V_{IN})	−1.5 to V_{CC} +1.5V
DC Output Voltage (V_{OUT})	0.5 to V_{CC} +0.5V
Clamp Diode Current (I_{IK}, I_{OK})	± 20 mA
DC Output Current, per pin (I_{OUT})	± 35 mA
DC V_{CC} or GND Current, per pin (I_{CC})	± 70 mA
Storage Temperature Range (T_{STG})	65°C to +150°C
Power Dissipation (P_D)	
(Note 3)	600 mW
S.O. Package only	500 mW
Lead Temperature (T_L)	
(Soldering 10 seconds)	260°C

Recommended Operating Conditions

	Min	Max	Units
Supply Voltage (V_{CC})	2	6	V
DC Input or Output Voltage			
(V_{IN}, V_{OUT})	0	V_{CC}	V
Operating Temperature Range (T_A)	-40	+85	°C
Input Rise or Fall Times			
(t_r, t_f) V_{CC} = 2.0V		1000	ns
V_{CC} = 4.5V		500	ns
V_{CC} = 6.0V		400	ns

Note 1: Absolute Maximum Ratings are those values beyond which damage to the device may occur

Note 2: Unless otherwise specified all voltages are referenced to ground

Note 3: Power Dissipation temperature derating — plastic "N" package: 12 mW/°C from 65°C to 85°C

DC Electrical Characteristics (Note 4)

Symbol	Parameter	Conditions	V_{CC}	T_A = 25°C		T_A = 40 to 85°C	T_A = 55 to 125°C	Units
				Typ	Guaranteed Limits			
V_H	Minimum HIGH Level		2.0V		1.5	1.5	1.5	V
	Input Voltage		4.5V		3.15	3.15	3.15	V
			6.0V		4.2	4.2	4.2	V
V_L	Maximum LOW Level		2.0V		0.5	0.5	0.5	V
	Input Voltage		4.5V		1.35	1.35	1.35	V
			6.0V		1.8	1.8	1.8	V
V_{OH}	Minimum HIGH Level	$V_N = V_{IH}$ or V_L						
	Output Voltage	$\lvert I_{OUT}\rvert \leq 20\ \mu A$	2.0V	2.0	1.9	1.9	1.9	V
			4.5V	4.5	4.4	4.4	4.4	V
			6.0V	6.0	5.9	5.9	5.9	V
		$V_N = V_{IH}$ or V_L						V
		$\lvert I_{OUT}\rvert \leq 6.0$ mA	4.5V	4.2	3.98	3.84	3.7	V
		$\lvert I_{OUT}\rvert \leq 7.8$ mA	6.0V	5.7	5.4	5.34	5.2	V
V_{OL}	Maximum LOW Level	$V_N = V_{IH}$ or V_L						
	Output Voltage	$\lvert I_{OUT}\rvert \leq 20\ \mu A$	2.0V	0	0.1	0.1	0.1	V
			4.5V	0	0.1	0.1	0.1	V
			6.0V	0	0.1	0.1	0.1	V
		$V_N = V_{IH}$ or V_L						
		$\lvert I_{OUT}\rvert \leq 6.0$ mA	4.5V	0.2	0.26	0.33	0.4	V
		$\lvert I_{OUT}\rvert \leq 7.8$ mA	6.0V	0.2	0.26	0.33	0.4	V
I_{IN}	Maximum Input	$V_N = V_{CC}$ or GND	6.0V		±0.1	±1.0	±1.0	µA
	Current							
I_{OZ}	Maximum 3-STATE	$V_N = V_{IH}$ or V_L	6.0V		±0.5	±5	±10	µA
	Output Leakage	$V_{OUT} = V_{CC}$ or GND						
	Current	$\bar{G} = V_{IH}$						
I_{CC}	Maximum Quiescent	$V_N = V_{CC}$ or GND	6.0V		8.0	80	160	µA
	Supply Current	$I_{OUT} = 0\ \mu A$						

Note 4: For a power supply of 5V ± 10% the worst case output voltages (V_{OH}, and V_{OL}) occur for HC at 4.5V. Thus the 4.5V values should be used when designing with this supply. Worst case V_H and V_L occur at V_{CC} = 5.5V and 4.5V respectively. (The V_H value at 5.5V is 3.85V.) The worst case leakage current (I_{IN}, I_{CC}, and I_{OZ}) occur for CMOS at the higher voltage and so the 6.0V values should be used

AC Electrical Characteristics

$V_{CC} = 5V$, $T_A = 25°C$, $t_r = t_f = 6$ ns

Symbol	Parameter	Conditions	Typ	Guaranteed Limit	Units
t_{PHL}, t_{PLH}	Maximum Propagation Delay	$C_L = 45$ pF	14	20	ns
t_{PZH}, t_{PZL}	Maximum Enable Delay to Active Output	$R_L = 1$ kΩ, $C_L = 45$ pF	17	28	ns
t_{PHZ}, t_{PLZ}	Maximum Disable Delay from Active Output	$R_L = 1$ kΩ, $C_L = 5$ pF	15	25	ns

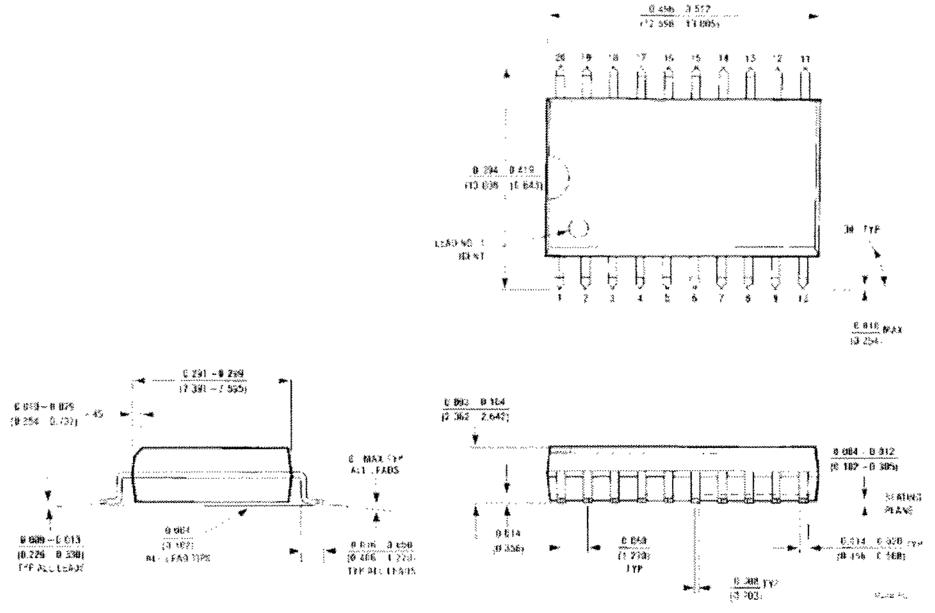
AC Electrical Characteristics

$V_{CC} = 2.0V–6.0V$, $C_L = 50$ pF, $t_r = t_f = 6$ ns (unless otherwise specified)

Symbol	Parameter	Conditions	V_{CC}	$T_A = 25°C$		$T_A = -40$ to 85°C	$T_A = -55$ to 125°C	Units
				Typ	Guaranteed Limits			
t_{PHL}, t_{PLH}	Maximum Propagation Delay	$C_L = 50$ pF	2.0V	58	115	145	171	ns
		$C_L = 150$ pF	2.0V	83	165	208	246	ns
		$C_L = 50$ pF	4.5V	14	23	29	34	ns
		$C_L = 150$ pF	4.5V	17	33	42	49	ns
		$C_L = 50$ pF	6.0V	10	20	25	29	ns
		$C_L = 150$ pF	6.0V	14	28	35	42	ns
t_{PZH}, t_{PZL}	Maximum Output Enable Time	$R_L = 1$ kΩ						
		$C_L = 50$ pF	2.0V	75	150	189	224	ns
		$C_L = 150$ pF	2.0V	100	200	252	298	ns
		$C_L = 50$ pF	4.5V	15	30	38	45	ns
		$C_L = 150$ pF	4.5V	30	40	50	60	ns
		$C_L = 50$ pF	6.0V	13	26	32	38	ns
		$C_L = 150$ pF	6.0V	17	34	43	51	ns
t_{PHZ}, t_{PLZ}	Maximum Output Disable Time	$R_L = 1$ kΩ, $C_L = 50$ pF	2.0V	75	150	189	224	ns
			4.5V	15	30	38	45	ns
			6.0V	13	26	32	38	ns
t_{TLH}, t_{THL}	Maximum Output Rise and Fall Time		2.0V		60	75	90	ns
			4.5V		12	15	18	ns
			6.0V		10	13	15	ns
C_{PD}	Power Dissipation Capacitance (Note 5)	(per buffer)						
		$\overline{G} = V_{IH}$		12				pF
		$\overline{G} = V_{IL}$		50				pF
C_{IN}	Maximum Input Capacitance			5	10	10	10	pF
C_{OUT}	Maximum Output Capacitance			10	20	20	20	pF

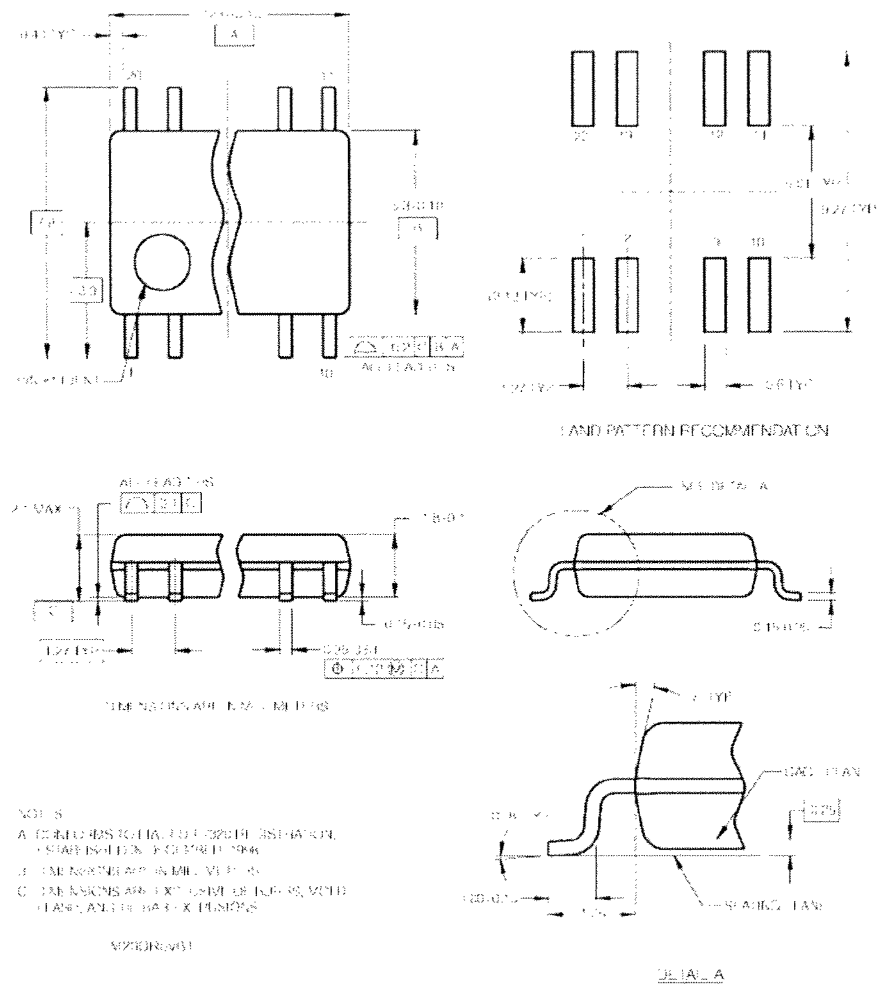
Note 5: C_{PD} determines the no load dynamic power consumption, $P_D = C_{PD} V_{CC}^2 f + I_{CC} V_{CC}$, and the no load dynamic current consumption, $I_S = C_{PD} V_{CC} f + I_{CC}$.

Physical Dimensions inches (millimeters) unless otherwise noted



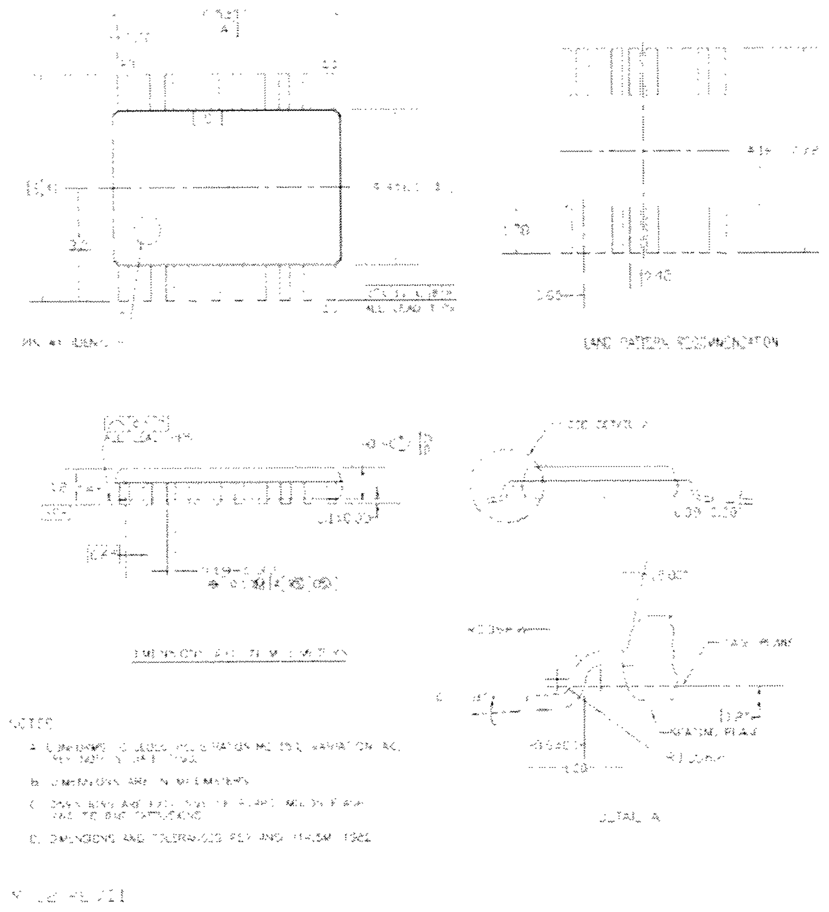
20-Lead Small Outline Integrated Circuit (SOIC), JEDEC MS-013, 0.300" Wide
Package Number M20B

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



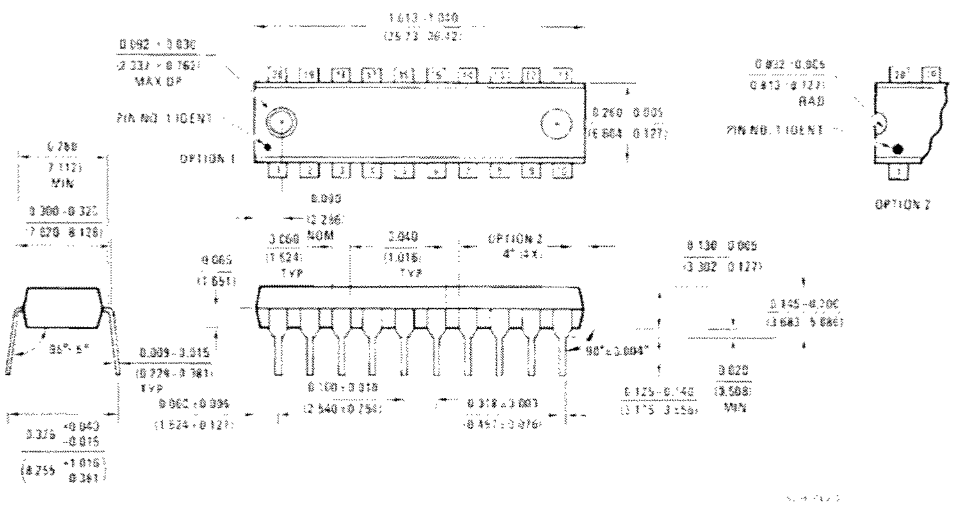
20-Lead Small Outline Package (SOP), EIAJ TYPE II, 5.3mm Wide
Package Number M20D

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



20-Lead Thin Shrink Small Outline Package (TSSOP), JEDEC MO-153, 4.4mm Wide
Package Number MTC20

Physical Dimensions inches (millimeters) unless otherwise noted (Continued)



20-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide
Package Number N20A

Appendix C

Relevant Pages of Atmega328P Microcontroller Datasheet

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32K Bytes of In-System Self-Programmable Flash progam memory (ATmega48PA/88PA/168PA/328P)
  - 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
  - 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C[1]
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^{2}C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 20 MHz @ 1.8 - 5.5V
- Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
  - Active Mode: 0.2 mA
  - Power-down Mode: 0.1 µA
  - Power-save Mode: 0.75 µA (Including 32 kHz RTC)

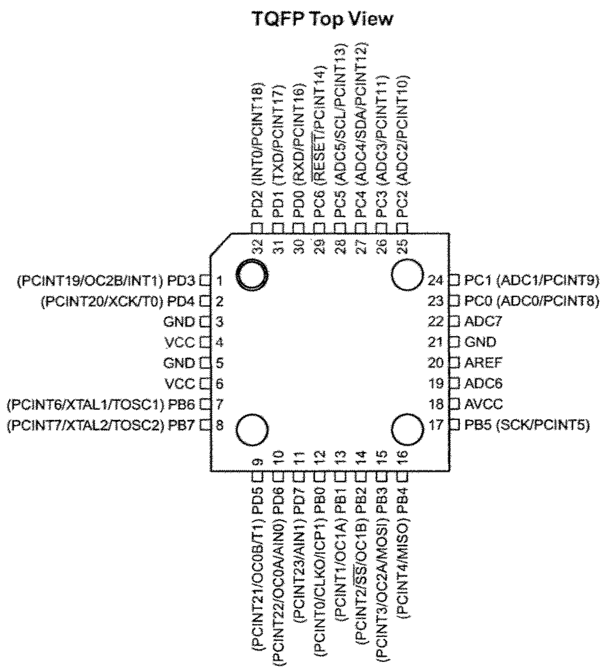
8-bit AVR® Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash

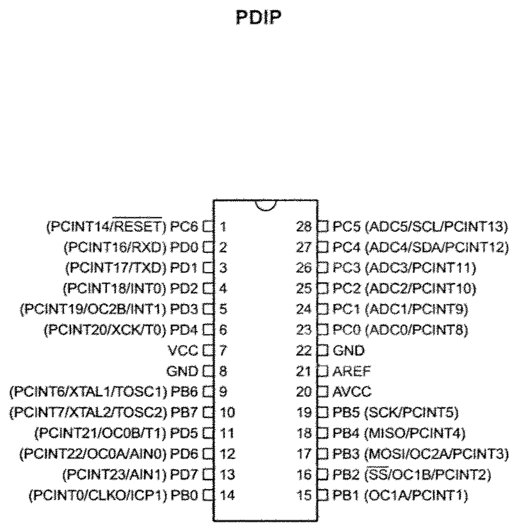
ATmega48PA
ATmega88PA
ATmega168PA
ATmega328P

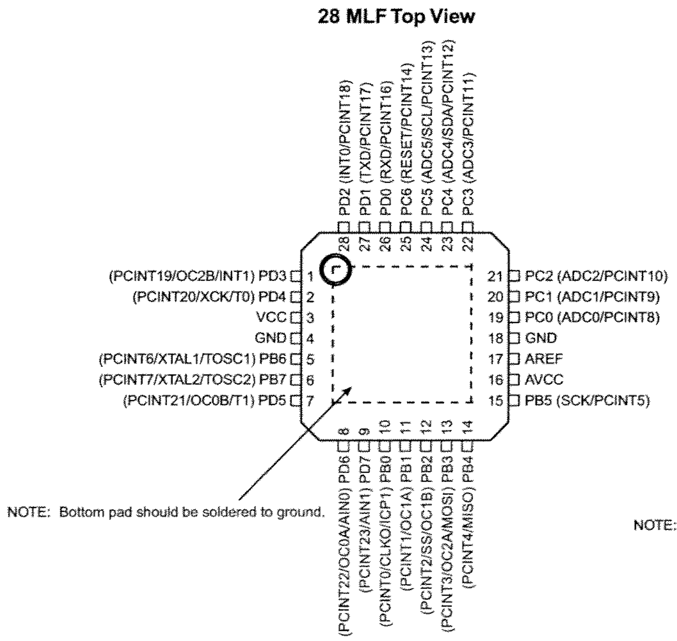
Rev. 8161D–AVR–10/09

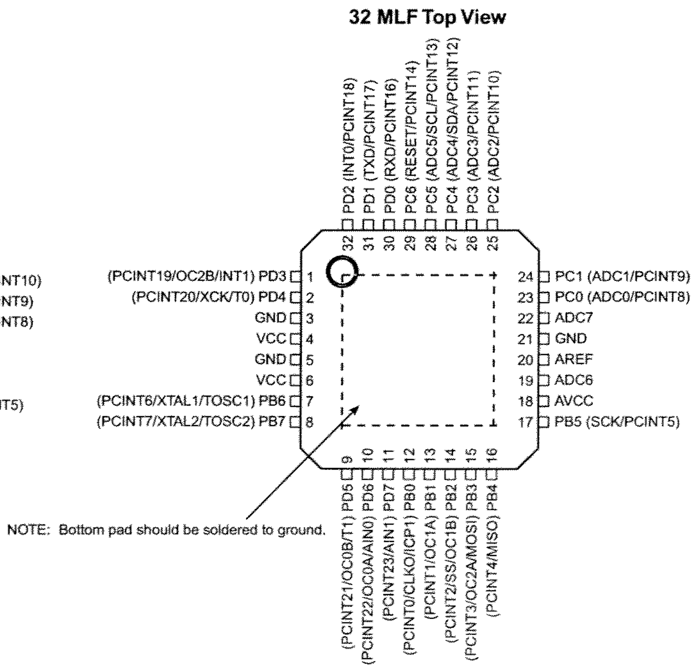
1. Pin Configurations

Figure 1-1. Pinout ATmega48PA/88PA/168PA/328P









28. Electrical Characteristics

28.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{RESET}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{RESET}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

28.2 DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, except XTAL1 and $\overline{RESET}$ pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH}	Input High Voltage, except XTAL1 and $\overline{RESET}$ pins	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL1}	Input Low Voltage, XTAL1 pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH1}	Input High Voltage, XTAL1 pin	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.8V_{CC}$(2) $0.7V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{IL2}	Input Low Voltage, $\overline{RESET}$ pin	V_{CC} = 1.8V - 5.5V	-0.5		$0.1V_{CC}$(1)	V
V_{IH2}	Input High Voltage, $\overline{RESET}$ pin	V_{CC} = 1.8V - 5.5V	$0.9V_{CC}$(2)		V_{CC} + 0.5	V
V_{IL3}	Input Low Voltage, $\overline{RESET}$ pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	-0.5 -0.5		$0.2V_{CC}$(1) $0.3V_{CC}$(1)	V
V_{IH3}	Input High Voltage, $\overline{RESET}$ pin as I/O	V_{CC} = 1.8V - 2.4V V_{CC} = 2.4V - 5.5V	$0.7V_{CC}$(2) $0.6V_{CC}$(2)		V_{CC} + 0.5 V_{CC} + 0.5	V
V_{OL}	Output Low Voltage(3) except RESET pin	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.9 0.6	V
V_{OH}	Output High Voltage(4) except Reset pin	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.2 2.3			V
I_{IL}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	V_{CC} = 5.5V, pin high (absolute value)			1	µA

28.2.4 ATmega328P DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

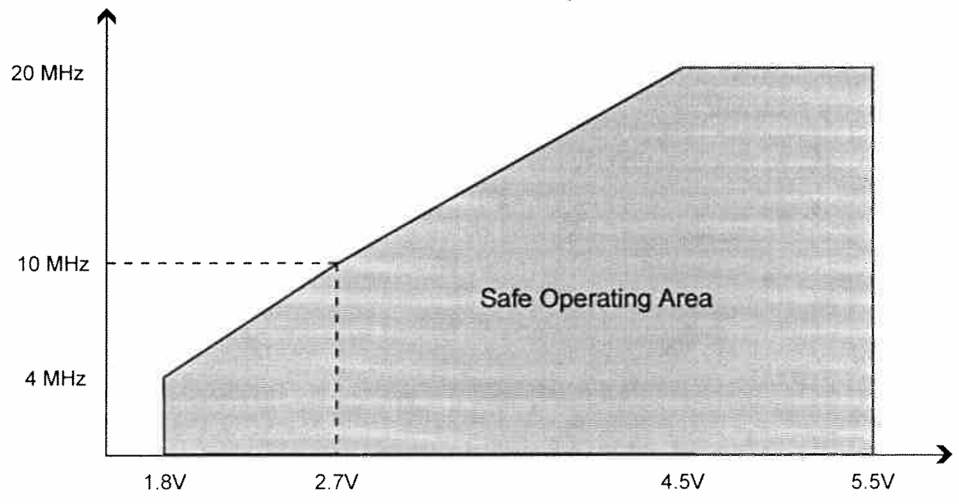
Symbol	Parameter	Condition	Min.	Typ.[2]	Max.	Units
I_{CC}	Power Supply Current[1]	Active 1 MHz, V_{CC} = 2V		0.3	0.5	mA
		Active 4 MHz, V_{CC} = 3V		1.7	2.5	mA
		Active 8 MHz, V_{CC} = 5V		5.2	9	mA
		Idle 1 MHz, V_{CC} = 2V		0.04	0.15	mA
		Idle 4 MHz, V_{CC} = 3V		0.3	0.7	mA
		Idle 8 MHz, V_{CC} = 5V		1.2	2.7	mA
	Power-save mode[3][4]	32 kHz TOSC enabled, V_{CC} = 1.8V		0.8	1.6	μA
		32 kHz TOSC enabled, V_{CC} = 3V		0.9	2.6	μA
	Power-down mode[3]	WDT enabled, V_{CC} = 3V		4.2	8	μA
		WDT disabled, V_{CC} = 3V		0.1	2	μA

Notes: 1. Values with "Minimizing Power Consumption" enabled (0xFF).
2. Typical values at 25°C. Maximum values are test limits in production.
3. The current consumption values include input leakage current.
4. Maximum values are characterized values and not test limits in production.

28.3 Speed Grades

Maximum frequency is dependent on V_{CC}. As shown in Figure 28-1, the Maximum Frequency vs. V_{CC} curve is linear between 1.8V < V_{CC} < 2.7V and between 2.7V < V_{CC} < 4.5V.

Figure 28-1. Maximum Frequency vs. V_{CC}



Appendix D

Microcontroller Serial IO Expander C Code

```
/*
Alex Taylor
Southern Utah University, 2013

Raspberry Pi to Arduino
Serial IO Expander
*/

//set constants
const byte pins[] = {8,7,6,5};
const byte pinsAnalog[] = {17,16,15,14};

byte first = 0;
byte value = 0;
byte analogAsDigital = 0;

byte x;


void setup(){
  Serial.begin(115200);

  //Set aref to external value
  analogReference(EXTERNAL);

  //Set all digital pins to input by default
  for(x = 0; x<4; x++)
    pinMode(pins[x], INPUT);

}


void loop(){
  //wait until there is some serial data recieved
  if (Serial.available() > 0){
    //read in the first byte
    first = Serial.read();

    //if setting pwm or digital pin, read the second bit for the value
    if ((bitRead(first, 7) == 0) && (bitRead(first,6) == 1)){
      while(Serial.available() < 1);
      value = Serial.read();

    }


    //if the program flag is set, run the setMode function
    if (first & (1<<7))
      setMode();
    else{
      //If bit 6 == 1 then SET if 0, then GET
      if(first & (1<<6))
```

```
        set();
      else
        get();
    }
  }

}

void setMode(){
  //If bit 5 is 1 then set DIGITAL pin, if 0, set ANALOG pin);
  //Return 0xFF when complete

  if(first & (1<<5)){
    for( x = 0; x < 4; x++){
        pinMode(pins[x], bitRead(first,x));
    }
  }
  else{
    for( x = 0; x < 4; x++){
      pinMode(pinsAnalog[x], bitRead(first,x));
      if (bitRead(first,x) == 0)
        bitClear(analogAsDigital, x);
      else
        bitSet(analogAsDigital, x);
    }
  }

  Serial.write(255);
}

void set(){
  //Set the specified pin, and return 0xFF.
  //Check first for PWM flag
  if(first & (1<<4)){
    for (x = 2; x < 4; x++){
      if(bitRead(first,x))
        analogWrite(pins[x], value);
    }

  }
  //If PWM is not set, continue on
  else{
    if(first & (1<<5)){
      //Run through the last four bits of first
      for( x = 0; x < 4; x++){
        //If one of the bits is 1, write value to that pin.
        if(bitRead(first,x))
          digitalWrite(pins[x], value);
      }
    }
    //this code sets the analog pin to a digital value (the pin must be set to output for
    this to work)
    else{
      for( x = 0; x < 4; x++){
        if(bitRead(first,x) && bitRead(analogAsDigital, x))
          digitalWrite(pinsAnalog[x], value);
      }
    }
  }
  Serial.write(255);
}

void get(){
  //Get the specified pin, and return the value.
  if(first & (1<<5)){
    for( x = 0; x < 4; x++){
      if(bitRead(first,x)){
        Serial.write(digitalRead(pins[x]));
        return;
      }
    }
  }
  else{
    for( x = 0; x < 4; x++){
      if(bitRead(first,x) && (bitRead(analogAsDigital, x)==0)){
        //Serial.write(pinsAnalog[x]);
        Serial.write(analogRead(pinsAnalog[x])/4);
      }
    }
  }

}
```

Appendix E

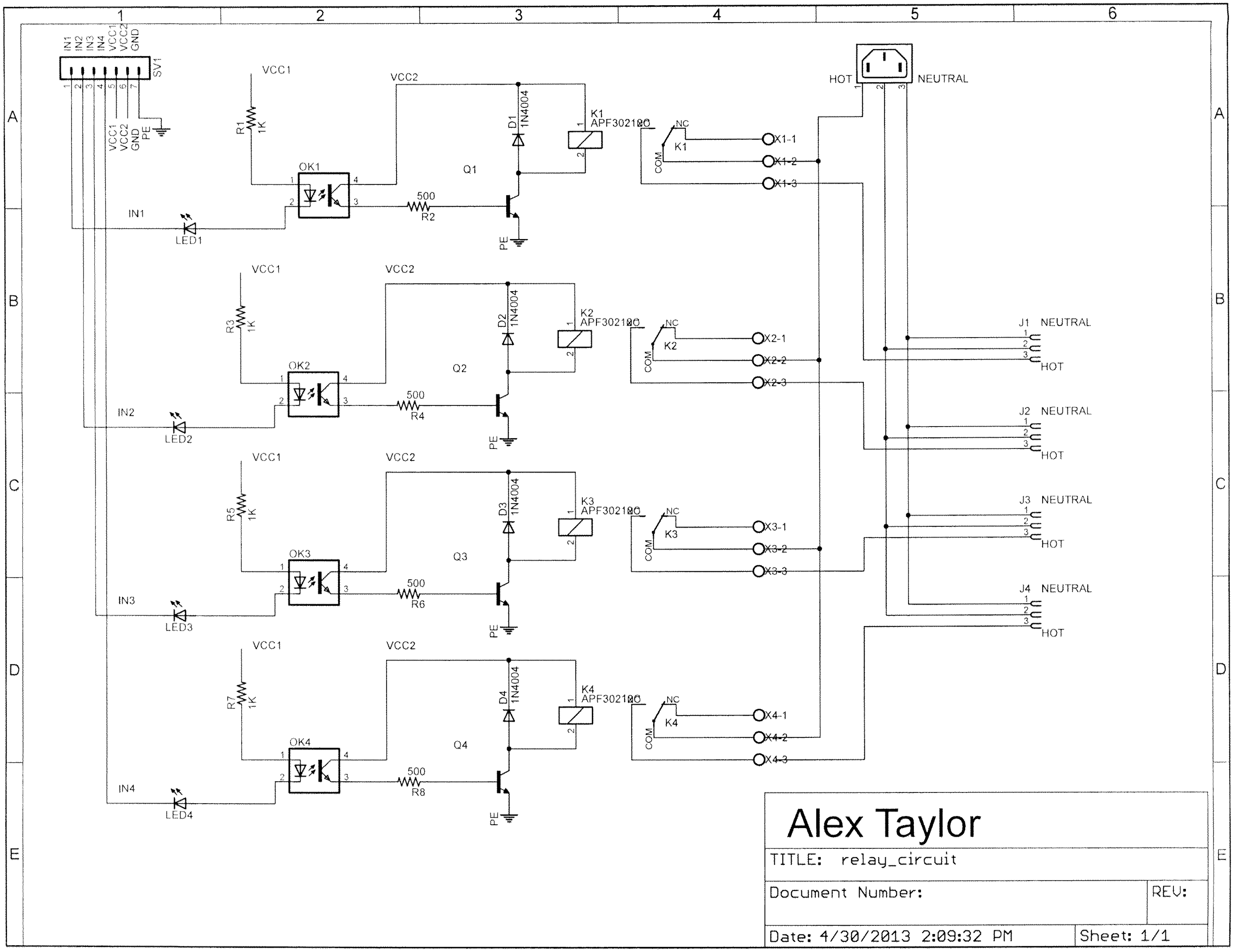
Raspberry Pi Development Board Python Library

```
#Alex Taylor
#Raspberry Pi Development Board
#Southern Utah University, 2013


import RPi.GPIO as GPIO
import time, serial, struct


#Initalize serial port
port = serial.Serial("/dev/ttyAMA0", baudrate=115200, timeout=0)

#Declare constants
OUTPUT = 1
INPUT = 0
relayPins = [4, 17, 21 ,22]
outputs = [23, 18, 25, 24]

#initalize GPIO output pins
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
for x in range(4):
  GPIO.setup(relayPins[x], GPIO.OUT)
  GPIO.setup(outputs[x], GPIO.OUT)
  GPIO.output(relayPins[x], 1)
  GPIO.output(outputs[x], 0)




#Turn the specified relay coil to the provided state
def setRelay(relayNum, state):
  GPIO.output(relayPins[relayNum], not state) # invert state because the relay is active
  low.



#Sets the specified extrenal output pin to the provided state
def setOutput(outputNum, state):
  GPIO.output(outputs[outputNum], state)



#Set the microcontroller pins to input or output

#args:
#ad = single character 'A' or 'D'
#pins = string with 4 1's or 0's

#Example: setMode('D', "1111") -- Sets all digital to outputs.
def pinMode(ad, pins):
  #Set the program flag true, and insert second bit(its irrevelant for setMode)
  temp = "11"
  #Determine if A or D channel
  if (ad == 'A'):
    #set a/d channel false if analog
    temp += "01"
    for x in range(4):
        temp += pins[x]
  elif (ad == 'D'):
    #set a/d channel true if digital
    temp += "11"
    for x in range(4):
        temp += pins[x]
  else:
    print "Error, Please choose A or D"
    return

  #convert the string of 1's and 0's to an integer
  temp = int(temp, 2)
  #Debugging - Uncomment to verify what integer value is being sent
  #print temp
  temp = struct.pack('B', temp)  # here you are packing the number 1 as a binary.
  port.write(temp)
  #wait until there is a response
  while (port.inWaiting() < 1):
    pass
  response = port.read(1)
  response = ord(response)
  if (response != 255):
    print "ERROR, didnt recieve a confirmation byte or its not equal to 255"



#set output pin over serial
def digitalWrite(pin, state):
  temp = "01"
  #set digital/analog bit depending on the value of pin
  if (pin < 4):
    temp += "10"
  else:
    temp += "00"
    pin -= 4

  for x in range(4):
    if (x == pin):
      temp += "1"
    else:
      temp += "0"

  #convert the string of 1's and 0's to an integer
  temp = int(temp, 2)
  #DEBUG
  #print(temp)
  temp = struct.pack('B', temp)  # pack the string to binary
  port.write(temp)
  #Send second byte containing the value
  port.write(struct.pack('B', state))

  #wait until there is a response
  while (port.inWaiting() < 1):
    pass
  response = port.read(1)
  response = ord(response)
  if (response != 255):
    print "ERROR, didnt recieve a confirmation byte or its not equal to 255", response



#Read a digital input pin over serial
def digitalRead(pin):
  temp = "0011"
  for x in range(4):
    if (x == pin):
      temp += "1"
    else:
      temp += "0"

  return read2(temp)


def analogRead(pin):
  temp = "0000"
  for x in range(4):
    if (x == pin):
      temp += "1"
    else:
      temp += "0"

  return read2(temp)

def read2(temp):
  #Clear out serial buffer
  port.read()
  #convert the string of 1's and 0's to an integer
  temp = int(temp, 2)
  #DEBUG
  #print(temp)
  temp = struct.pack('B', temp)  # pack the string to binary
  port.write(temp)
  while (port.inWaiting() < 1):
    pass
  response = port.read(1)
  return ord(response)
```

```
def analogWrite(pin, value):
  temp = "01"
  #set digital/analog bit depending on the value of pin
  temp += "11"

  for x in range(4):
    if (x == pin):
      temp += "1"
    else:
      temp += "0"

  #convert the string of 1's and 0's to an integer
  temp = int(temp, 2)
  #DEBUG
  #print(temp)
  temp = struct.pack('B', temp)  # pack the string to binary
  port.write(temp)
  #Send second byte containing the value
  port.write(struct.pack('B', value))

  #wait until there is a response
  while (port.inWaiting() < 1):
    pass
  response = port.read(1)
  response = ord(response)
  if (response != 255):
    print "ERROR, didnt recieve a confirmation byte or its not equal to 255", response



def cleanup():
  for x in range(4):
    GPIO.output(relayPins[x], 1)
    GPIO.output(outputs[x], 0)

  pinMode('D', "0000");
  pinMode('A', "0000");
```

Appendix F

Relay Module Schematic Diagram



Alex Taylor

TITLE: relay_circuit

Document Number:

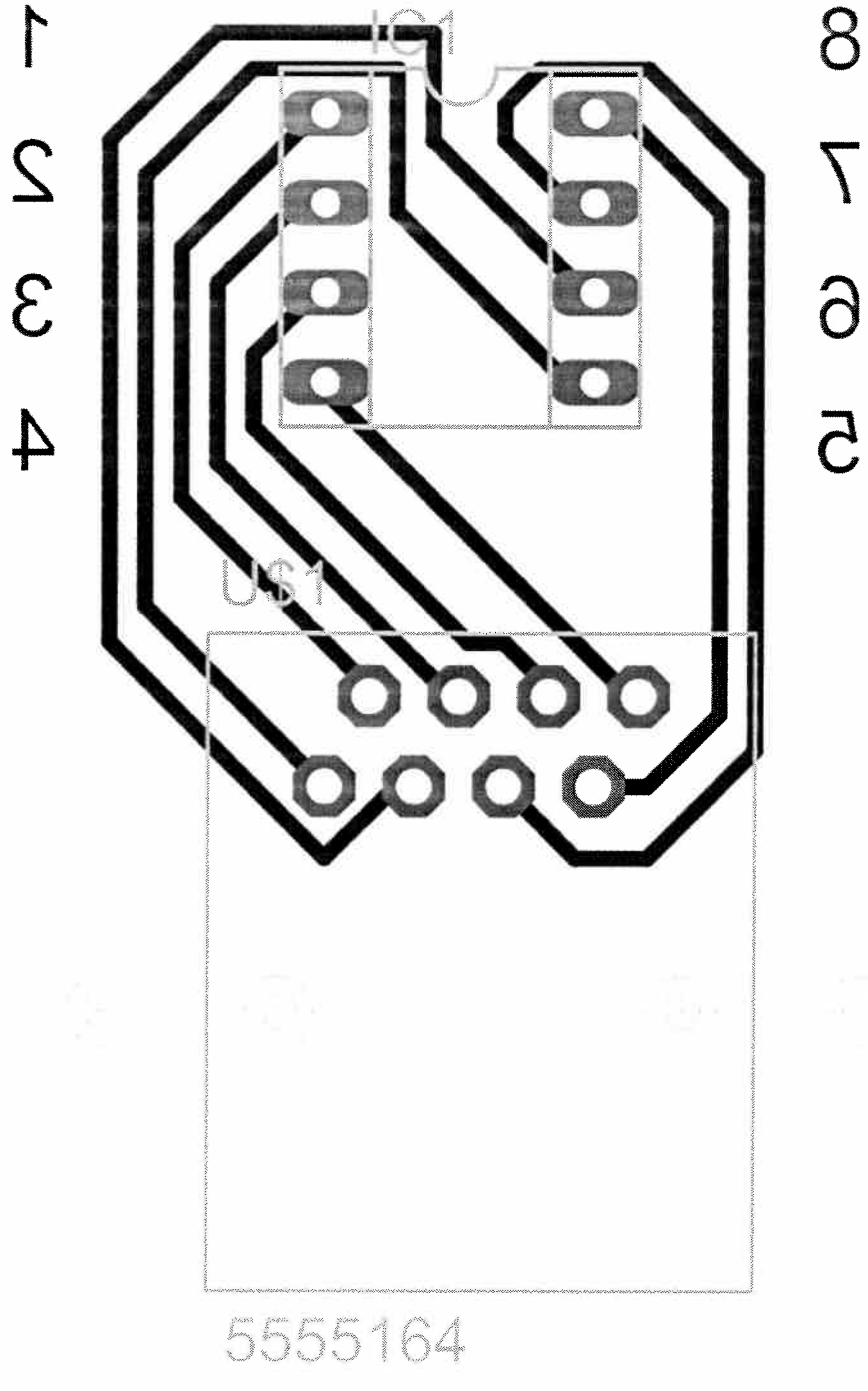
REV:

Date: 4/30/2013 2:09:32 PM

Sheet: 1/1

Appendix G

RJ45 Breakout Board


1
2
3
4
8
7
6
5
IC1
US1
5555164

Appendix H

Gmail Notifier Python Program

```
#Alex Taylor
#Python Gmail Notifier
#Southern Utah University, 2013


#import required libraries
import dev_board as pi, feedparser, time

#provide gmail login credentials
USERNAME = "username_here"     # just the part before the @ sign
PASSWORD = "password_here"

NEWMAIL_OFFSET = 0          # how many messages to keep the light off
MAIL_CHECK_FREQ = 60        # check mail every 60 seconds


RELAY_NUM = 0


try:
  #loop forever
  while True:

    #fetch number of unread messages using feedparser
    newmails = int(feedparser.parse("https://" + USERNAME + ":" + PASSWORD +
    "@mail.google.com/gmail/feed/atom")["feed"]["fullcount"])
    print newmails

    #compare number of new emails to the offset
    #if newmails is greater than offset, turn lamp on, else turn lamp off
    if newmails > NEWMAIL_OFFSET:
      pi.setRelay(RELAY_NUM, 1)
    else:
      pi.setRelay(RELAY_NUM, 0)

    #notify the user every time the program checks gmail
    print "Just Updated..."

    #time delay to check Gmail
    time.sleep(MAIL_CHECK_FREQ)

except KeyboardInterrupt:
  pi.cleanup();
```

Appendix I

Twitter Monitor Python Program

```
#Alex Taylor
#Twitter Notifier
#Southern Utah University, 2013

#import required libraries
import dev_board as pi, time, twitter

#access the twitter API using my oauth token
def main():
  api = twitter.Api(consumer_key='EGJDu5LfotURQkzkduAleA',consumer_secret=
  'PaAnVZZmTu0sHDebUSHm0smJU14Ue9t17ZN7fEHSgus',
  access_token_key='1392430634-BitdX4NV4WAtCSib1c5zAzV0Gp3CpzzSmHLve3T',
  access_token_secret='VXMDx24kJxH3Ek11A92aiAhUHf7xoLIgaBPB5nvCk')



  try:
    #flash lamp to signal program start
    flash(1)

    #check twitter initially to determine most current id number
    query = api.GetSearch(term='#SUUEET')
    idNum = 0
    for s in query:
      if idNum < s.id:
        idNum = s.id

    #loop forever
    while 1:
      count = 0
      #query twitter using specific string
      query = api.GetSearch(term='#SUUEET', since_id=idNum)
      print "Just Checked"
      #determine how mnay new results have been returned
      for s in query:
        #print each result on the screen
        print s.text
        idNum = s.id
        count += 1
        #flash the lamp for each result
        flash(count)
      #limit the rate we poll twitter
      time.sleep(60)

  except KeyboardInterrupt:
    pi.cleanup()

#function to flash the lamp 'count' number of times
def flash(count):
  for x in range(count):
    for y in range(3):
      pi.setRelay(0,1)
      time.sleep(.25)
      pi.setRelay(0,0)
      time.sleep(.25)
    time.sleep(1)
  return

if __name__ == '__main__':
  main()
```