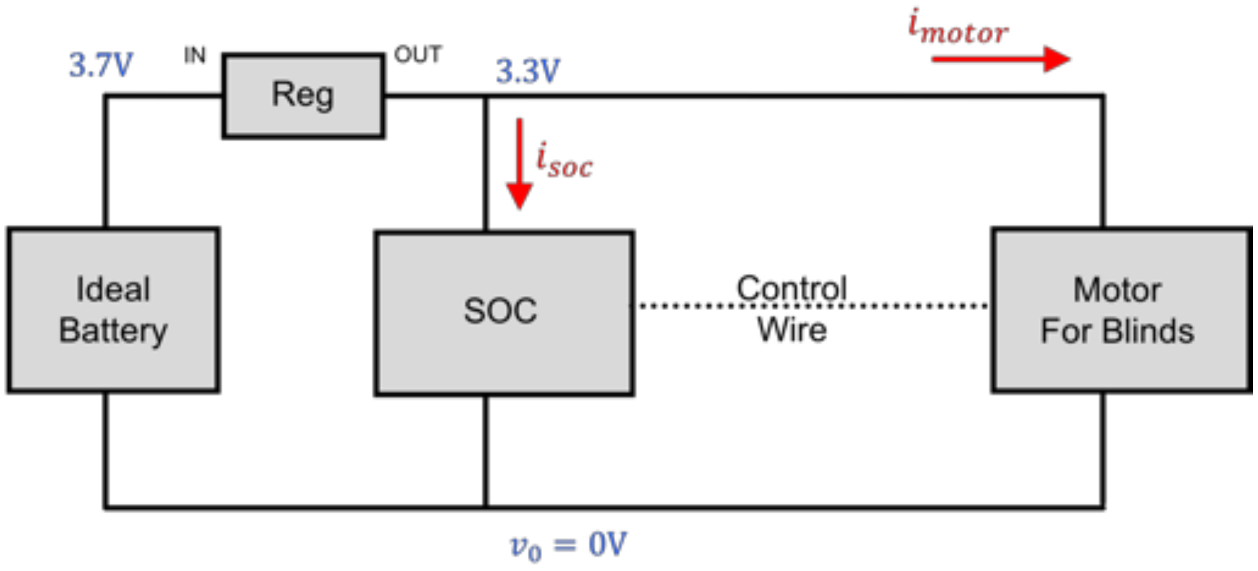
Cast-Off Exam Problems 2018 #1:

Assume we have a set battery-powered set of automatic blinds for a window to control outside illumination getting in, with the following schematic:



The system is intended to raise or lower the blinds based on the illumination detected outside (through a light detector).

The motor has two states:

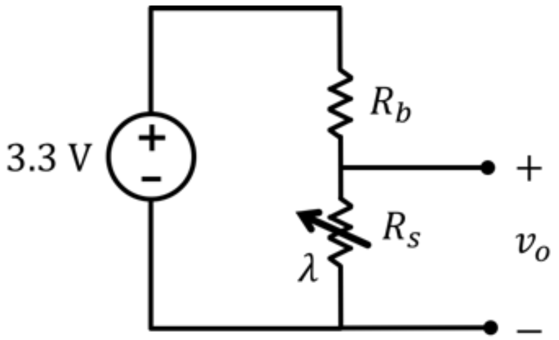
ON: The motor is running and either raising or lowering the blinds, i_{motor} = 300 mA
OFF: The motor is off and cannot raise or lower blinds, i_{motor} = 0 mA

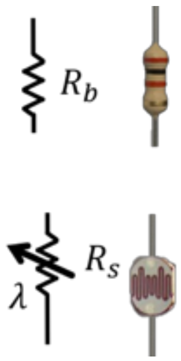
The SOC (System on a Chip), which contains both a microcontroller and a light sensor also has two states:

ON: The SOC can make measurements and control the motor: i_{soc} = 28 mA
SLEEP: The SOC cannot make measurements or control the motor, i_{soc} = 0.5 mA

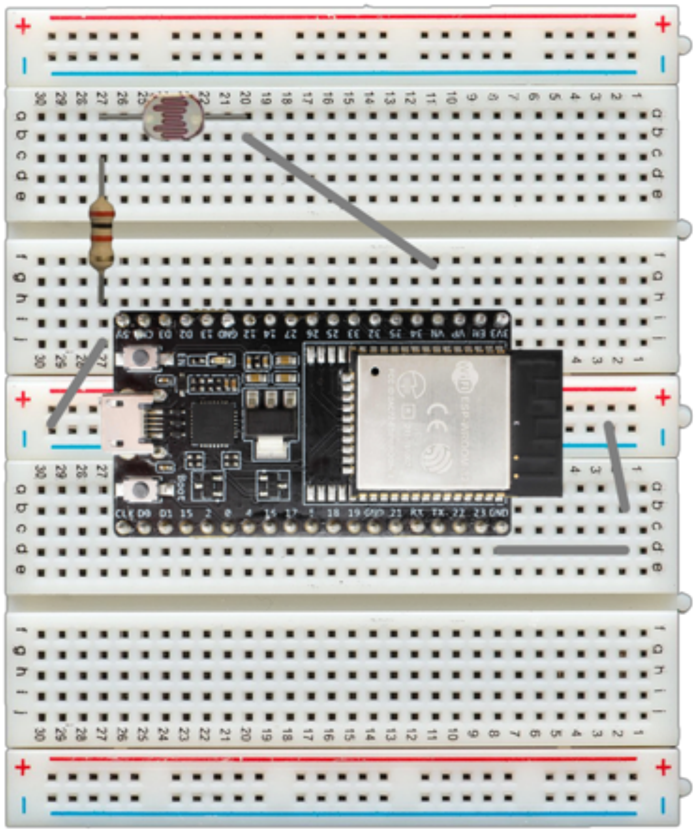
Note the microcontroller needs to be actively on in order to control the motor.

A)
The way this system operates is the following: With a fixed periodicity, the SOC makes a light measurement via the following circuit:





Where R_s is a photoresistor (a resistor whose resistance varies with light), and R_b is a fixed "bias" resistor of value 2000 Ohms (depictions of which are shown on the right above). The voltage v_o is measured by the ADC on Analog Pin A3 of the ESP32 using 11 bits of resolution. Your friend builds this circuit using your ESP32, but they aren't confident in their implementation. Is there anything wrong with this circuit? If so, make corrections by ***clearly*** crossing out bad wires and adding new wires. (note there are no hidden wires underneath the ESP32 in this particular circuit!) (also note the battery and motor connections are not shown)!



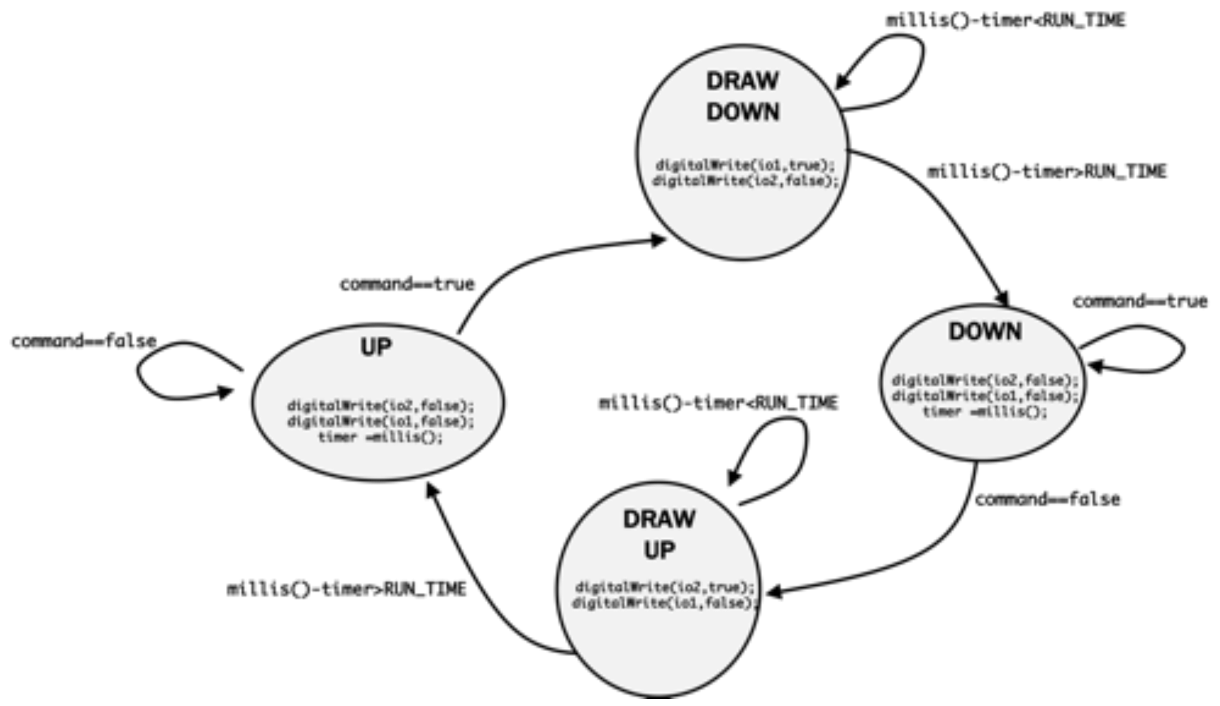
B.) Returning to the circuit, the photoresistor's behavior based on the illumination L (in lumens) it is exposed to has the following relationship:

$$\log_{10}(R_s) = -1.8\log_{10}(L) + 7.1$$

The control scheme is to calculate the amount of light shining onto the window (as picked up by the photoresistor and its surrounding circuit), and compare to a user-defined threshold. If the level is above the threshold (it is sufficiently light outside), the blinds will go down, accomplished by calling the `run_the_blinds` function with true as its input argument. Otherwise (it isn't bright enough out) the blinds will come up by calling the `run_the_blinds` function with a false argument. Finish the code below so that this control scheme holds true (you may assume math.h is included, along with the pow function.

```
void loop(){
  int reading = analogRead(A3);
  float illumination; //in lux










  run_the_blinds(illumination>threshold);

  while (millis()-timer<loop_time);
  timer = millis();
}
```

C) The `run_the_blinds` function is a **blocking** function call and operates by controlling two digital output pins which control (through some intermediate electronics we don't need to worry about), two different sides of a motor. In order to drive the motor upwards digital pin `io2` needs to be set High and `io1` needs to be set low. For movement in the opposite direction (drawing the blinds down), `io1` needs to be High and `io2` needs to be Low. For no movement, both `io1` and `io2` need to be low.



Using ONE global variable for your state, implement a version of run_the_blinds that adheres to the state machine above.

```
void run_the_blinds(bool command){











}
```

C Part 2) Bonus side question. You get sad seeing that your implementation above is a blocking function. Also the whole nature of that primary loop isn't great either. Instead, if we rewrite the overall loop structure like so, we now could rewrite run_the_blinds to be non-blocking

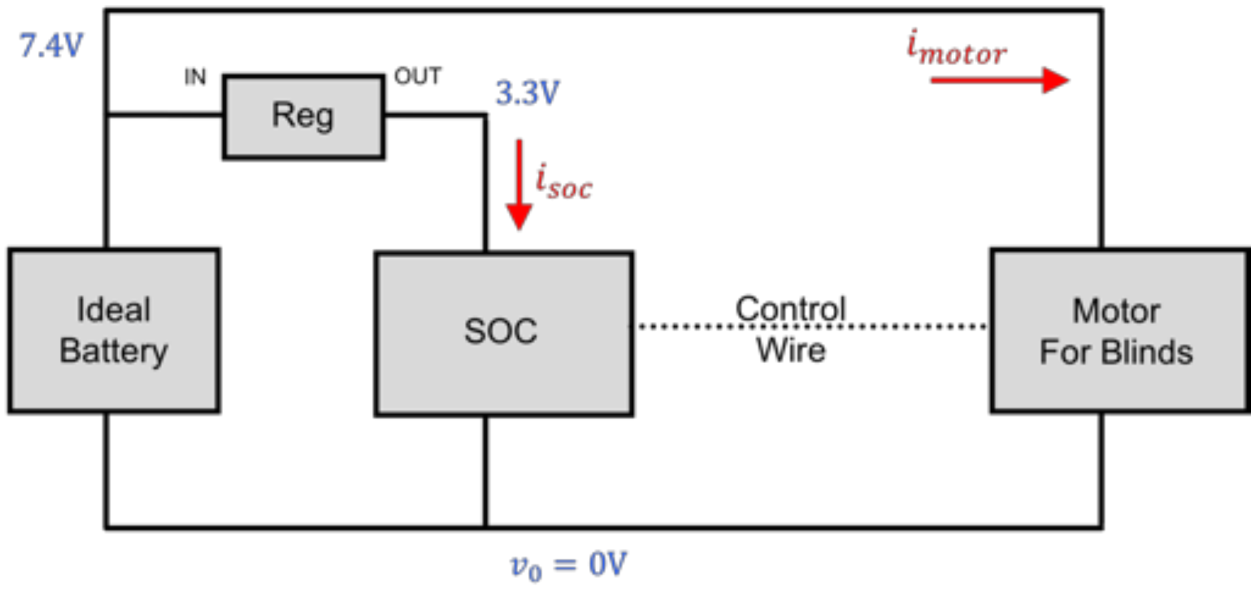
```
int reading; //global now
float illumination; //global now
//assume things are setup correctly

void loop(){
  if (millis() - timer >loop_time){
    timer = millis();
    reading = analogRead(A3);
    float illumination; //in lux
    //your code from above to determine illumination
  }

  run_the_blinds(illumination>threshold);
}
```

Using <u>**TWO**</u> global variables, implement a version of `run_the_blinds` that adheres to the state machine on the other page and which is non-blocking!

```
void run_the_blinds(bool command){









}
```

D) Returning the power consumption of the device discussed at the beginning of this problem, we'd now like to make sure our entire system can last as long as possible given certain constraints. For all modes of operation, what is the efficiency of the system when on battery power?

E) After some initial tests we realize we can't get the battery to motor to generate sufficient torque when run on 3.3V or 3.7V. The solution is to string two 3.7V ideal batteries together to get a 7.4V battery. Further, we rewired some things so that the motor is now powered directrly from 7.4V rather than 3.3V.



Ei) In this new circuit, if the motor is OFF and the SOC is on, what is the efficiency of the system overall?

Eii) In this new circuit, if the motor is OFF and the SOC is in SLEEP, what is the efficiency of the system overall?

Eiii) In this new circuit, if the motor is ON and the SOC is ON, what is the efficiency of the system overall?

F) If we choose a 3Ah battery and mess up our code so that the SOC and the motor are constantly in an ON/RUNNING state, how long will the system last overall (in hours?)

G) If we reconfigure our operational code so that the SOC sleeps for 59.95 seconds, wakes up to take a 50 ms-long measurement and then either:

- Changes the state of the window-blinds , which takes 4.0 seconds of running the motor, before going back to sleep for 55.95 seconds
- Goes immediately back to sleep for 59.95 seconds

How long will the system last if, on average, the blinds only need to have their state changed every 10 minutes?

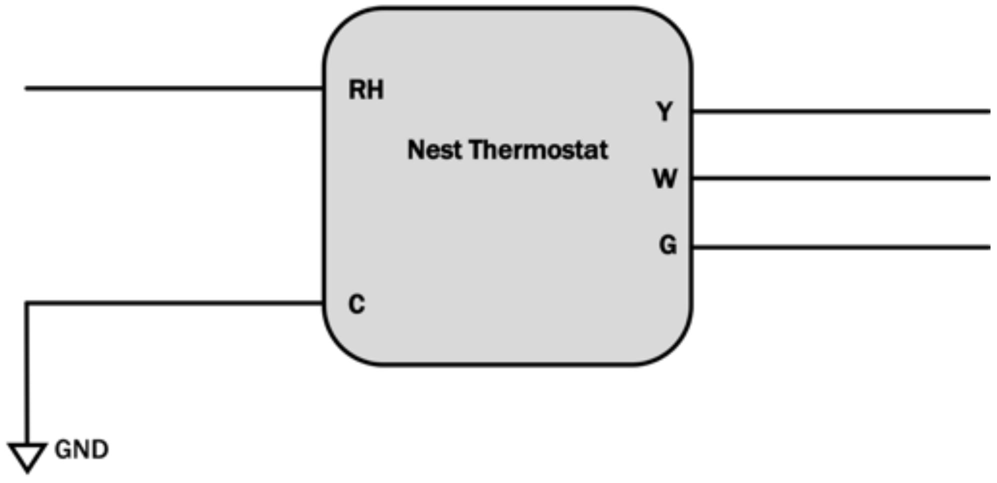
H) If we then reconfigure our operational code so that the SOC sleeps for 9.95 seconds, wakes up to take a 50 ms-long measurement and then either:

- Changes the state of the window-blinds , which takes 4.0 seconds of running the motor, before going back to sleep for 5.95 seconds
- Goes immediately back to sleep for 9.95 seconds

How long will the system last if, on average, the blinds only need to have their state changed every 20 minutes?

Cast-Off Exam Problems 2018 #2:
Nest thermostat. A Nest Thermostat works using a five wire system. *(In real life the entire system works on 24 Volts of Alternating Current since that is standard and goes back to the early part of the twentieth century…anywho…). For our system we'll assume the following. There are five connections:

- RH: A constant 24 V supply. When used in conjunction with the C wire, this provides power to the Nest Thermostat as well as a common signal reference for all devices in the system.
- C: The "common" wire, used for power supply return and signal reference. Essentially the system's ground.
- Y: Y, for "yellow" is the fan signal line. Low when OFF, High when ON
- W: W, for "white" is the heater signal line. Low when OFF, High when ON
- G for "green" is the air-conditioning signal line: Low when OFF, High when ON



The Nest thermostat has a standard internal "3.7" battery similar of similar charging characteristics to the battery used in 6.08 except it has a capacity of 300 mAH.

A) If our Nest thermostat uses a linear regulator to convert the incoming 24V down for battery charging, what is the efficiency of our charging system when the battery is at a voltage of 3.6V and the incoming current is 200 mA?

B) What if instead our Nest thermostat used a non-linear regulator with 95% efficiency to convert between 24V and the 3.7V for the battery? If the input current to this non-linear regulator was 50 mA, what is the output current from this regulator?

C) Our Nest thermostat has an onboard temperature sensor which expresses the temperature ranging from 0 degrees Celsius up to 65 degrees Celsius using 14 bits. The measurement can be obtained via the following member function call: `nest.get_temp();` What is the "bin" size of temperature value from this sensor? Put another way, what is the smallest incremental temperature change the sensor can report?

D) Write a non-blocking implementation of a state machine for one operation mode for the therostate controller, which we'll call `void system_update(int temperature)`

The function should be expected to be used like the following:

```
void loop(){
  int temp = nest.get_temp(); //measured temperature
  system_update(temp);
}
```

A global variable `set_temperature`, expresses the desired temperature in degrees Celsius.

- If the actual temperature is at or below `set_temperature`, the system should heat until the measured temperature is at or above `set_temperature`.
- If the actual temperature is greater than 2 degrees warmer than the `set_temperature`, the system should run the air conditioning until the measured temperature is at or below 2 degrees above `set_temperature.`
- The system should never run the heater or air conditioner, at the same time.
- We can ignore the fan in this problem.

The Nest can turn on and off pins Y, W, and G using `digitalWrite(Y, value)`, `digitalWrite(W,value)`, and `digitalWrite(G, value)`, respectively, where `value` is either 1 or 0.

*We don't intend to make any programming question **this** open-ended on the exam, but since this is for studying, we're leaving it phrased as is.*

```
//declare globals here and note initial values if needed









void system_update(int temperature){

























}
```

E) Part of the great thing about a Nest Thermostat is that it can go off of a schedule (heat for certain hours and cool for others). In order to do this, it must periodically make queries to a server containing a database to collect instructions. It does this using a POST request of the following structure where the body contains the Nest ID and password. For example, here's the structure of one POST.

```
POST /thermostat/commands HTTP/1.1
Host: nest.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

nest_id=30127&password=evita
```

If Nest had a `request_handler` function on their server that looked like the following:

```
def request_handler(request):
    return str(request)
```

The following response would be returned:

```
{'form': {'nest_id': '30127', 'password': 'evita'}, 'values': {},
'is_json': False, 'args': [], 'method': 'POST'}
```

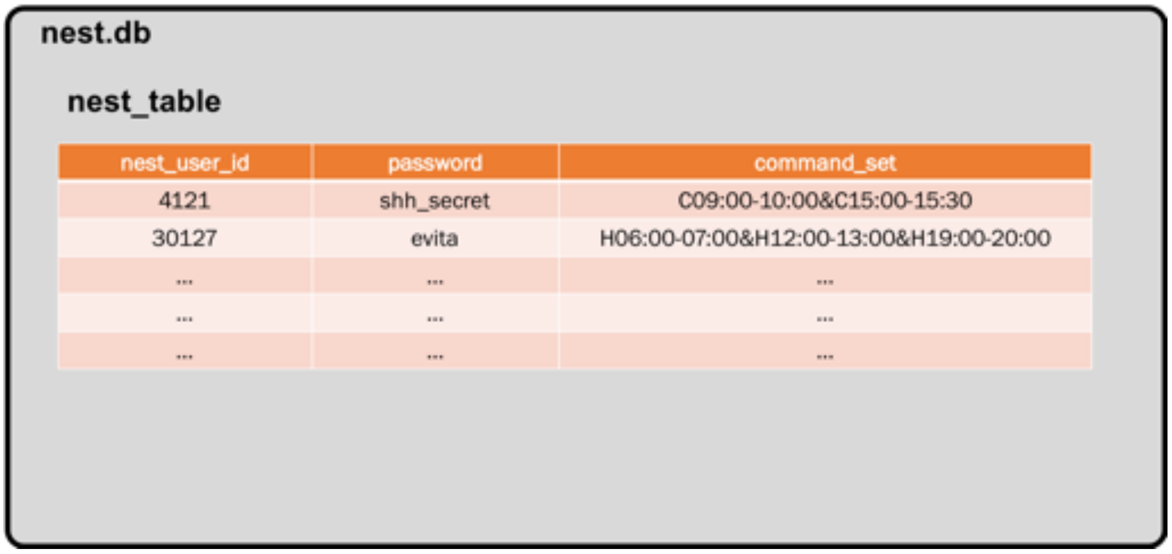
This is, of course, pretty useless. What we'd like instead is for our request_handler to accept in the Nest User ID and password, verify that the password is correct, and then based on the time of day, return either a:

- "N" for "no command"
- "H" for "run heater"
- "C" for "run air conditioner"
- "F" for "run fan"

Your implementation can get all needed information from the nest_table inside the nest.db file located on the server, a sqlite setup.

Our nest database, depicted on the following page, contains three columns:

- `nest_user_id`: the Nest thermostat user id. For example 30127
- `password`: The raw password for the user
- `command_set`: A string expressing when and what the particular Nest thermostat is supposed to do during any 24 hour period. The command string is split up into different commands via the & symbol and each command has either a "C", for Cool, "H" for Heat, or "F" for Fan followed by a time-range in 24-hour clock format. The command "C05:00-08:00"means run the air conditioning from 5AM to 8AM, for example.



nest.db

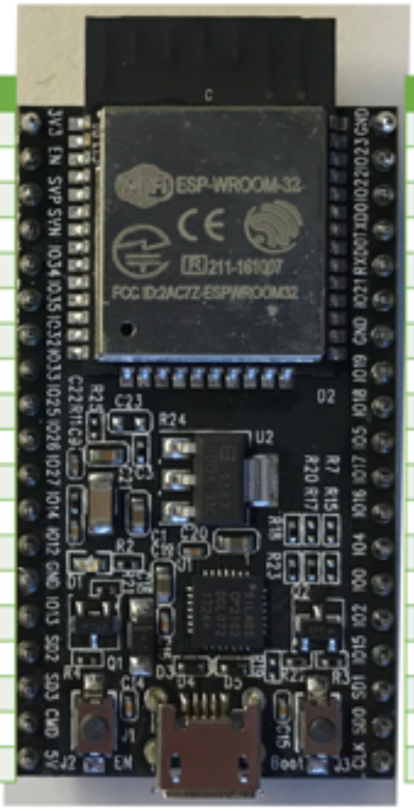
nest_table

nest_user_id	password	command_set
4121	shh_secret	C09:00-10:00&C15:00-15:30
30127	evita	H06:00-07:00&H12:00-13:00&H19:00-20:00
...	...	...
...	...	...
...	...	...

Create an implementation of `request_handler` that satisfies all the. Since this is a practice problem, you can assume we'd give you sufficient documentation here for working with this (any necessary SQLITE example blips, and/or `datetime` examples).

```
def request_handler(request):
```

Other	Analog	Digital
3.3V		
	A0	IO36
	A3	IO39
	A6	IO34
	A7	IO35
UART RX	A4	IO32
UART TX	A5	IO33
	A18	IO25
	A19	IO26
	A17	IO27
	A16	IO14
	A15	IO12
GND		
	A14	IO13
		IO9
		IO0
		IO11
5V		



Digital	Analog	Other
		GND
IO23		SPI MOSI
IO22		I2C SCL
IO1		
IO3		
IO21		I2C SDA
		GND
IO19		SPI MISO
IO18		SPI SCK
IO5		SPI SS
IO17		
IO16		
IO4	A10	
IO0	A11	
IO2	A12	
IO15	A13	
IO8		
IO7		
IO6		