**Objective:** Integrate real-time PCR cycling routine into Raspberry Pi python command line.

* Communicate and record output from Arduino controlling thermocycling, fans, LEDs
* Synchronize camera capture with Arduino commands indicating PCR annealing

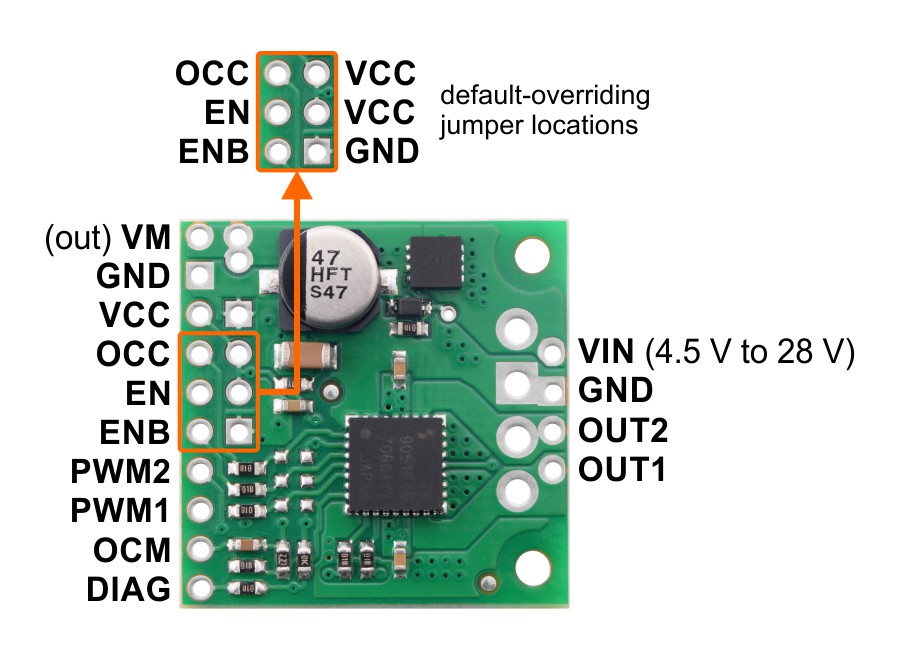
Workflow Draft

1. User enters “cycle” into command line for command.py script run on python
   1. runs temp.cycle()
      1. GUI will run all commands in a thread – running commands in command.py is blocking
2. Pi makes a new folder named by the date/time 🡪 puts it in home directory in a “PromptDx” folder 🡪 sets as current working directory
   1. Makes PromptDx folder if it doesn’t exist
3. Pi opens log files (log.csv, temp.csv, fluo.csv)
4. Pi sends Arduino cycling parameters
   1. Reverse transcription Temp
   2. Reverse transcription Time
   3. Hotstart Temp
   4. Hotstart Time
   5. Anneal Temp
   6. Anneal Time
   7. Denature Temp
   8. Denature Time
   9. Number of Cycles
5. Pi sends “<cycle>” to nano
6. Nano prints…
   1. Time and temperature data, comma-separated with “T” prefix and ended with CRLF
      1. Example: T,10.0,50.0
   2. “PB” or “PR” to indicate the blue or red LED are on and the Pi should take a picture
      1. Pi returns “P” when picture is taken
   3. Any additional info sent with “L” prefix to store in log file
   4. “E” to indicate End of run

**Heater Control**

TB9051FTG Single Brushed DC Motor Driver Carrier

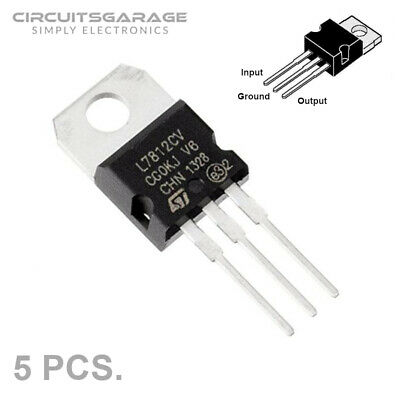
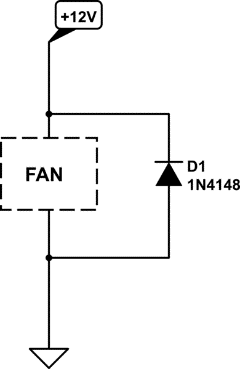
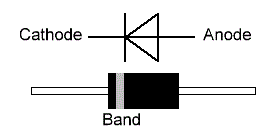
<https://www.pololu.com/product/2997>

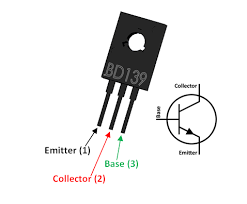


TB9051FTG simplified truth table (PWM1 + PWM2)

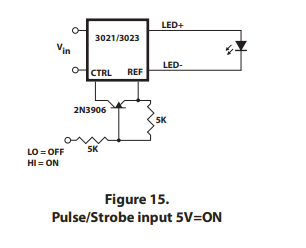
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TB9051FTG simplified truth table (PWM1 + PWM2)** | | | | | | |
| **Inputs** | | | | **Outputs** | | **Operation** |
| **EN** | **ENB** | **PWM1** | **PWM2** | **OUT1** | **OUT2** |
| 1 | 0 | PWM | 0 | PWM (H/L) | L | forward/brake at speed *PWM %* |
| 0 | PWM | L | PWM (H/L) | reverse/brake at speed *PWM %* |
| 0 | 0 | L | L | brake low (outputs shorted to ground) |
| 1 | 1 | L | L |
| 0 | X | X | X | Z | Z | coast (outputs floating/disconnected) |
| X | 1 | X | X | Z | Z |

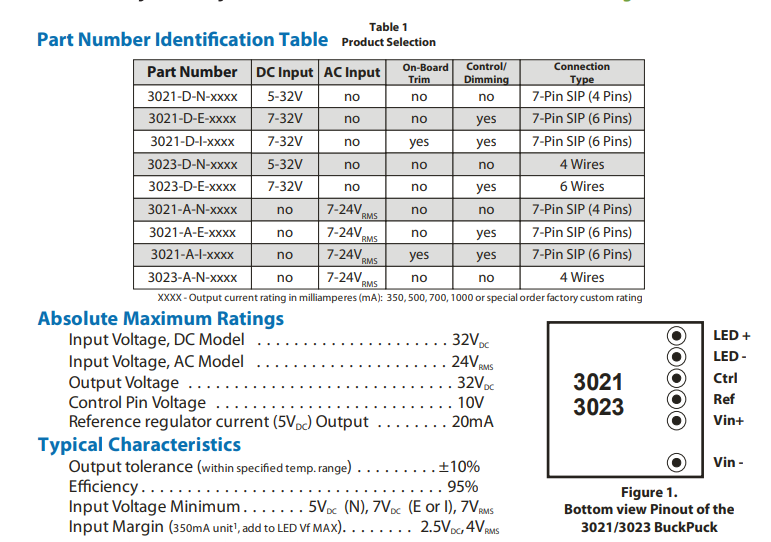
Fan control 12V regulation with L7812

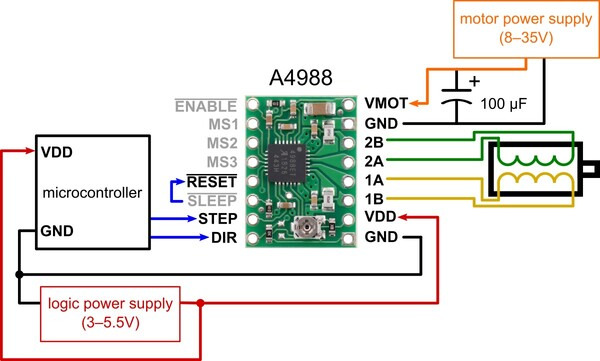






Buckpuck LED control



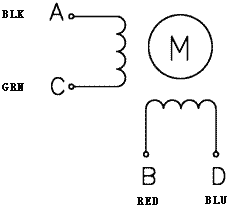
<https://www.pololu.com/product/1182>

**USE “SLEEP” AS THE ENABLE PIN – this is by default in the BasicStepperDriver library**

**Enable**. This input turns on or off all of the FET outputs. When set to a logic high, the outputs are disabled. When set to a logic low, the internal control enables the outputs as required. The translator inputs STEP, DIR, and MSx, as well as the internal sequencing logic, all remain active, independent of the ENABLE input state

**Sleep**. To minimize power consumption when the motor is not in use, this input disables much of the internal circuitry including the output FETs, current regulator, and charge pump. A logic low on the SLEEP pin puts the A4988 into Sleep mode. A logic high allows normal operation, as well as start-up (at which time the A4988 drives the motor to the Home microstep position). When emerging from Sleep mode, in order to allow the charge pump to stabilize, provide a delay of 1 ms before issuing a Step command.

**Reset**. The RESET input sets the translator to a predefined Home state (shown in Figures 9 through 13), and turns off all of the FET outputs. All STEP inputs are ignored until the RESET input is set to high.

<https://www.pololu.com/product/1209>

The above diagram shows a standard bipolar stepper motor. To control this with the A4988, connect stepper leads **A** and **C** to board outputs **1A** and **1B**, respectively, and stepper leads **B** and **D** to board outputs **2A** and **2B**, respectively. Note that if you happen to swap which way the wires are connected for any coil, the stepper motor will turn in the opposite direction, and if you happen to pair up wires from different coils, the motor should be noticeably erratic when you try to step it, if it even moves at all.