Speeduino Manual

02/05/21

Contents

Introduction

This manual covers the hardware (sensors, wiring etc), software configuration and tuning elements related to running a Speeduino unit. When beginning with Speeduino, particularly if it is your first time installing and configuring an engine management system, this manual will assist in understanding Speeduino's capabilities and how it should be installed, both in terms of hardware and software/-firmware.

Whilst this document will assist in providing information related to Speeduino's configuration, it does not cover advanced engine tuning, fuel / ignition strategies etc. As with any changes to engine management, the possibility of damage to hardware is very real should a system be configured incorrectly.

Getting Started

In terms of starting out with Speeduino, it can help to understand the various components that make up the system:

1. **A Speeduino board** - This is the muscle of the Speeduino ECU and contains all the drivers and IO circuits. This maybe one of the generic boards (Such as the v0.4) or a PNP board for a specific model car
2. **An Arduino** - This is the brains of Speeduino and contains the processor, memory and storage. It plugs into the Speeduino board in order to interface with the vehicle wiring. Usually a Arduino Mega 2560.
3. **Firmware** - This is the system software that runs on the Arduino board and powers its operation. New firmware is released regularly with updates, performance improvements and bug fixes.

As a starting point, it is generally recommended to first upload the firmware to your Arduino and get it connecting to the tuning software (Tuner Studio) before moving on to hardware assembly or wiring etc. Software setup and configuration on Speeduino can be completed without the need for any additional hardware to be present (Beyond the arduino itself) and this allows exploration of the software and options available before either an outlay of significant funds or a significant investment of time.

More details on hardware requirments and verson specific features can be found on the Getting Started page

About this manual

As an open source project, this documentation is growing continually and this means that you may come across gaps in the documentation where little information is currently provided. Please do not hesitate to post on the forum if there is something missing that you need critically (or even not so critically).

Additionally, if you would like to contribute to the Speeduino documentation, we would love to hear from you! The preferred method to request wiki access is via Slack

Loading the Speeduino firmware

Overview

The Speeduino firmware is the code that powers the hardware and must be installed onto your board prior to using the ECU. New firmware releases are made regularly (Approximately every 2 months) that bring new features, bug fixes and performance improvements so staying up to date is highly recommended.

With the goal of maximum simplicity in mind, the process of compiling and installing the firmware is reasonably straightforward. Most users will use the SpeedyLoader method for installing the firmware

Installation - SpeedyLoader

The simplest (and recommended) method of installing the Speeduino firmware onto a standard Arduino Mega 2560 is with the SpeedyLoader utility. SpeedyLoader takes care of downloading the firmware and installing it onto an Arduino without the need to manually compile any of the code yourself. You can choose the newest firmware that has been released, or select from one of the older ones if preferred. SpeedyLoader will also download the INI file and optionally a base tune for the firmware you choose so it can be loaded into your TunerStudio project.

- **Windows:** 32-bit / 64-bit
- **Mac:** SpeedyLoader.dmg
- **Linux:** SpeedyLoader.AppImage (Will need to be made executable after downloading)
  - Linux requires libusb libraries to be installed. EG if on Debian/Ubuntu: `sudo apt-get install libusb-1.0-0 libusb-0.1-4:i386`
- **Raspberry Pi** SpeedyLoader.AppImage

  - Raspberry Pi / Raspbian users can install the required libraries with: `sudo apt-get install libusb-1.0-0 libusb-0.1-4`

Once the firmware is installed on the board, see Connecting to TunerStudio for more details on how to configure TunerStudio

Installation - Manually Compiling

> Note that manually compiling the firmware is **NOT** required to install Speeduino, the easiest (and recommended for most users) method is using SpeedyLoader as described above. {.is-warning}

If you want to compile the firmware yourself, or make any code changes, then the source of both the releases and the current development version is freely available.

Requirements

- A Windows, Mac or linux PC
- One of the following:
  - The Arduino IDE. Current minimum version required is 1.6.7, although a newer version is recommended.
  - PlatformIO. Can be downloaded from here
- A copy of the latest Speeduino codebase. See below.
- A copy of TunerStudio to test that the firmware has uploaded successfully

Downloading the firmware

There are two methods for obtaining the Speeduino firmware:

1. Regular, stable code drops are produced and made as releases on Github. These can be found at: Releases
2. If you want the latest and greatest (And occasionally flakiest) code, the git repository can be cloned and updated. See here

Compiling the firmware

- Start the IDE, select *File* > *Open*, navigate to the location you downloaded Speeduino to and open the **speeduino.ino** file.

- Set the board type: *Tools > Board > Arduino Mega 2560* or Mega ADK (This is the only board currently supported)
- Click the **Verify** icon in the top left corner (Looks like a tick)

At this point you should have a compiled firmware! If you experienced a problem during the compile, see the Troubleshooting section below.

This video walks through the whole process of installing the firmware on your Arduino from scratch:

Optional (But recommended) There is an option available for changing the compiler optimization level, which can improve . By default, the IDE uses the -Os compile option, which focuses on producing small binaries. As the size of the Speeduino code is not an issue but speed is a consideration, changing this to -O3 produces better results (Approximately 20% faster, with a 40% larger sketch size) To do this, you need to edit the platform.txt file:

- Make sure the Arduino IDE isn't running
- Open the platform.txt file which is in the following locations:
  - On Windows: c:\Program Files\Arduino\hardware\arduino\avr
  - On Mac: /Applications/Arduino/Contents/Resources/Java/hardware/arduino/avr/
  - On Linux:
- On the following 3 entries, change the Os to be O3:
  - compiler.c.flags
  - compiler.c.elf.flags
  - compiler.cpp.flags
- Save the file and restart the Arduino IDE

Note: This is NOT required if using PlatformIO, the above optimisation is applied automatically there

Installing

Once you've successfully compiled the firmware, installation on the board is trivial.

- Plug in your Mega 2560 to a free USB port
- If you're running an older version of **Windows** and this is the first time you've used an Arduino, you may need to install drivers for the Arduino serial chip (USB-UART or "USB adapter chip").

Most official boards and many non-official versions use the ATMega16U2 or 8U2, whereas many of the Mega2560 clone boards utilize the CH340G IC. Both types work well. The serial chips can generally be identified by appearance:

- **ATMega16U** - This has a square IC near the USB connector - drivers are included in Windows 7+, MacOS and Linux.
- **WCH CH340G** - This has a rectangular IC near the USB connector- uses "CH341" drivers from WCH for Windows
  - WCH-original CH340/CH341 drivers for other systems (Mac, Linux, Android, etc) may be found here.
- In Arduino IDE; select the Mega2560: *Tools* > *Board*
- Select your system's serial port to upload: *Tools* > *Serial Port*
- Hit the *Upload* button from the top left corner

Older firmware releases

If required, older firmware releases and details can be found at Firmware_History

Verifying Firmware

The firmware is now loaded onto your board and you are now able to move onto Connecting to TunerStudio.

Optionally, you may perform a manual verification of the firmware by using the Arduino IDE's Serial Monitor. This can be started by selecting 'Serial Monitor' from the Tools menu.

In the window that appears, enter a capital "S" (no quotes) and press *Enter*. The Mega should respond with the year and month of the code version installed (xxxx.xx):

```
1 Speeduino 2017.03
```

NOTE: Ensure the baud rate is set to 115200

You can also enter "?" for a list of queries from your Mega.

Troubleshooting

Incorrect Arduino board selected If you see the following (or similar) errors when trying to compile the firmware and the solutions:

```
scheduler.ino:317:7: error: ''OCR4A was not declared in this scope
scheduler.ino:323:8: error: ''TIMSK5 was not declared in this scope
scheduler.ino:323:25: error: ''OCIE4A was not declared in this scope
```

You may have the wrong kind of Arduino board selected. Set the board type by selecting *Tools* > *Board* > *Arduino Mega 2560* or Mega ADK

Entire Speeduino project is not opened The following can occur if you have only opened the speeduino.ino file rather than the whole project.

```
speeduino.ino:27:21: fatal error: globals.h: No such file or directory
```

Make sure all the files are contained within the same directory, then select File->Open and find the speeduino.ino file. If you have opened the project correctly, you should have multiple tabs along the top:

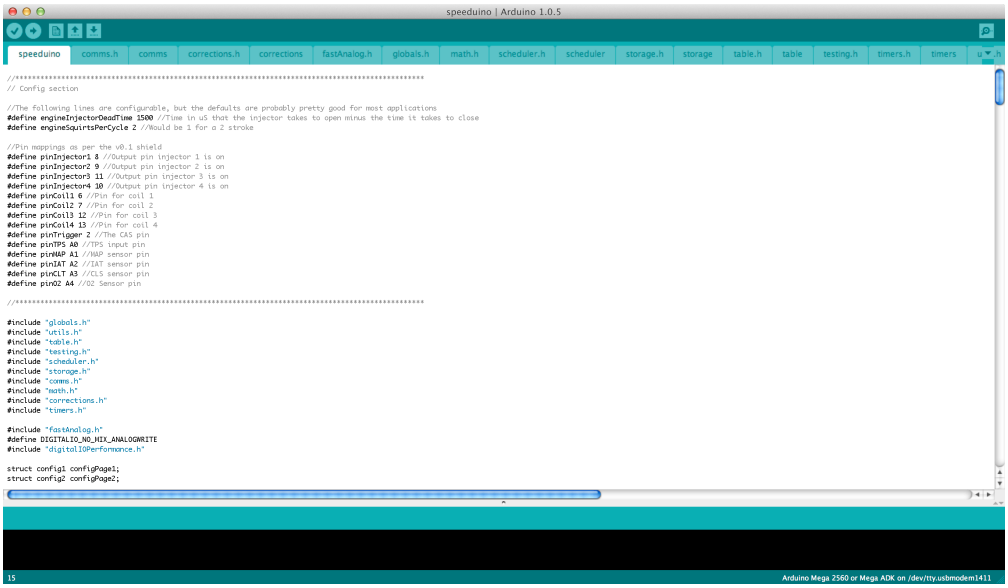


Figure 1: speeduinoIDE.png

If you see only a single file or a small number of files then you haven't opened the entire project.

Connecting to Tuner Studio

Tuner Studio is the tuning interface software used by Speeduino. It runs on Windows, Mac and linux and provides configuration, tuning and logging capabilities.

Once you have the firmware compiled and uploaded to your Arduino, you're ready to setup Tuner Studio in order to configure and monitor it. If you haven't yet compiled and uploaded the firmware, refer to the Installing Firmware page.

Downloading Tuner Studio

If you haven't already, grab a copy of Tuner Studio from EFI Analytics Tuner Studio is available for Windows, Mac and linux and will run on most PCs as it's system requirements are fairly low.

The current minimum version of TunerStudio required is 3.0.7, but the latest version is usually recommended.

If you find Tuner Studio to be useful, please consider paying for a license. This is a fantastic program from a single developer that rivals the best tuning software in the world, it's worth the money.

Setting up your project

Create new project

When you first start TunerStudio, you'll need to setup a new project which contains the settings, tune, logs etc. On the start up screen, select 'Create new project'

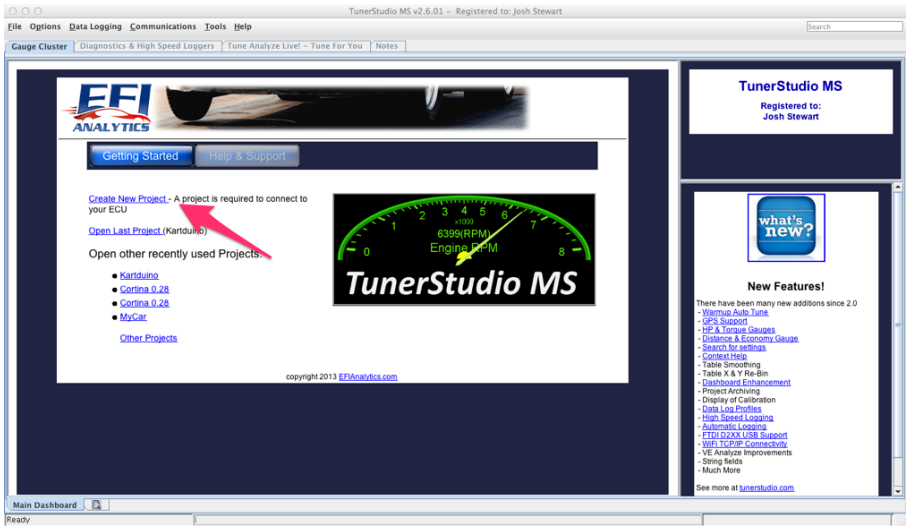


Figure 2: TS_1.png

Give you project a name and select the directory you want the project to be stored in. Tuner Studio then requires a firmware definition file in order to communicate with the arduino. Tick the 'Other / Browse' button.

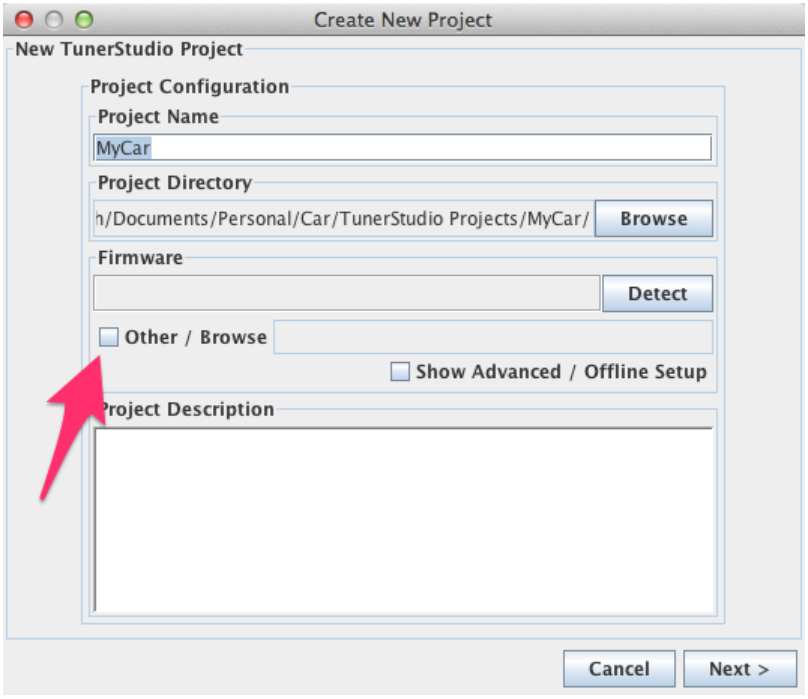


Figure 3: TS_2.png

Then browse to the Speeduino source directory, enter the reference subfolder and select speeduino.ini file

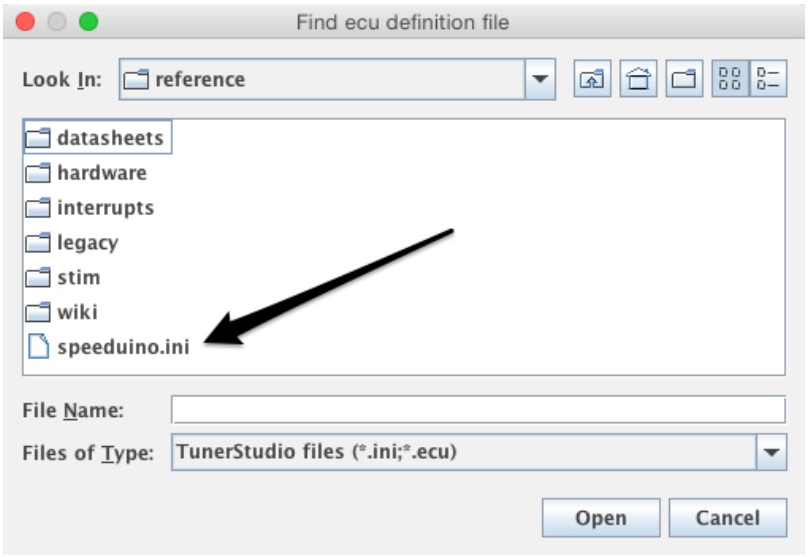


Figure 4: TS_3.png

Configuration options

Refer to the Configuring TunerStudio project options page for this

Comms settings

Select your comms options. The exact port name will depend on which operating system you are running and this will be the same as in the Arduino IDE. Baud rate should be 115200.

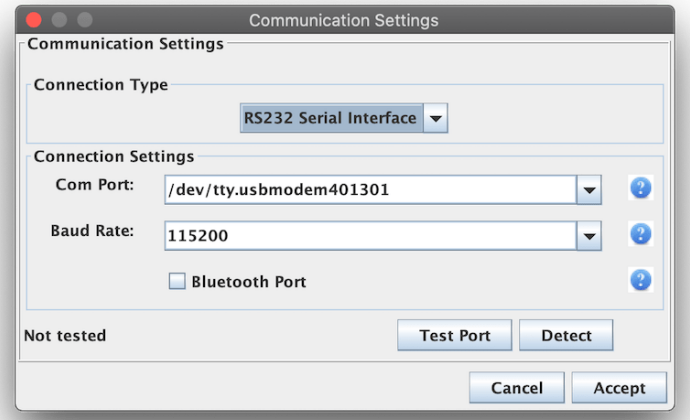


Figure 5: Comms Settings

> **Note:** The `Detect` and `Test port` options require Tuner Studio version 3.0.60 or above to work correctly {.is-info}

Load base tune

Once the project is created, you'll need to load in a base tune to ensure that all values are at least somewhat sane. Failure to do this can lead to very strange issues and values in your tune.

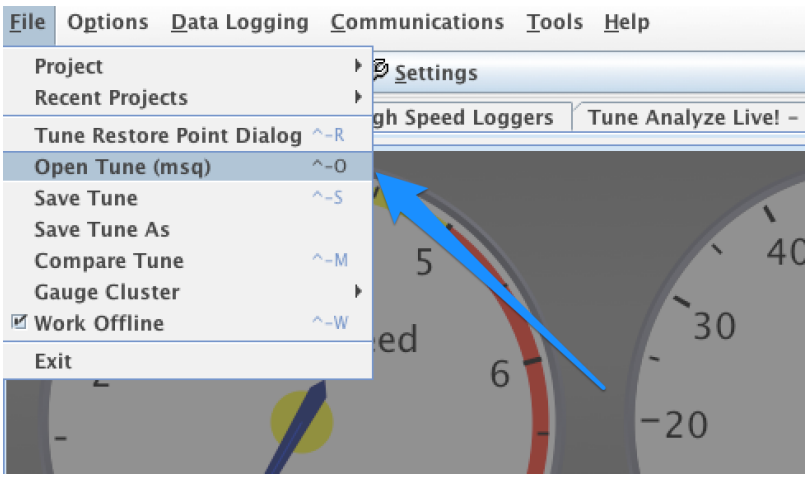


Figure 6: TS_6.png

In the Speeduino reference directory, you will find the base tune file to be opened:

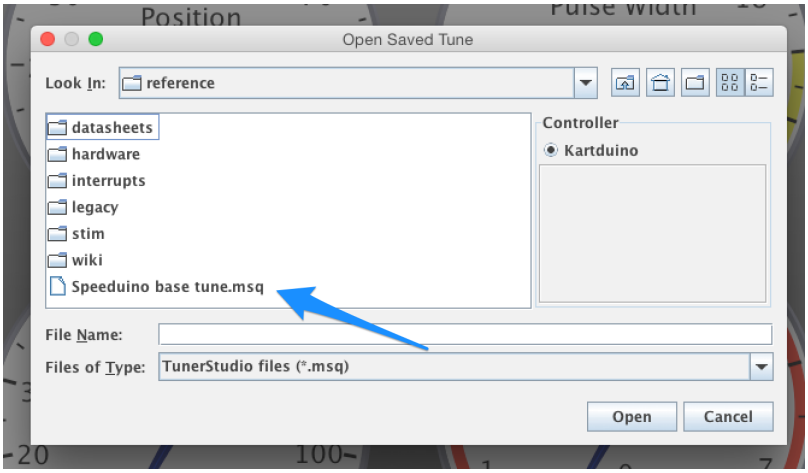


Figure 7: TS_7.png

And that's it! Tuner Studio should now attempt to connect to the Arduino and show a realtime display of the ECU.

Configuring TunerStudio Project Properties

The menu option for the project properties page can be found here

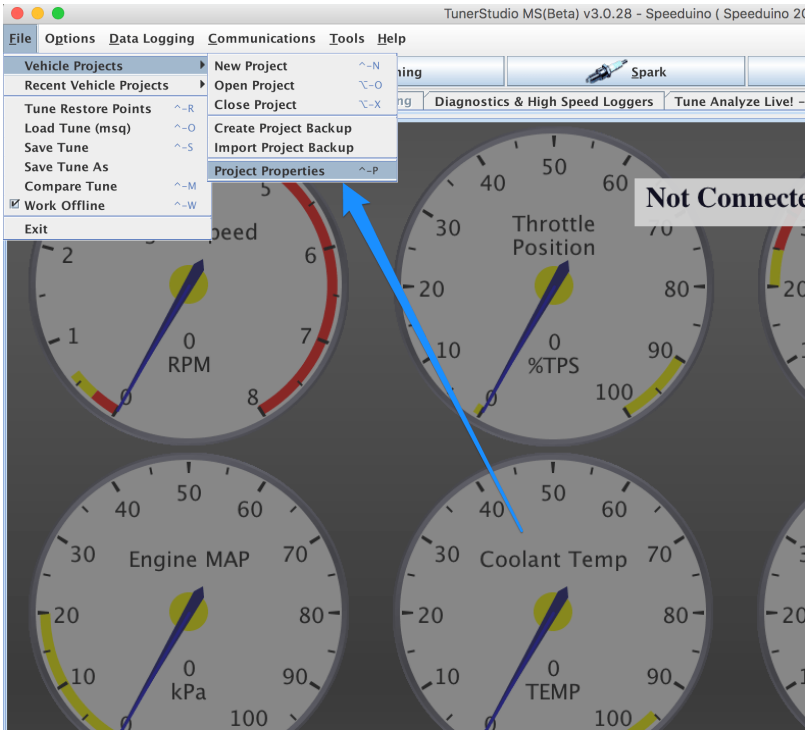


Figure 8: TS_9.png

Once opened this page will be seen.

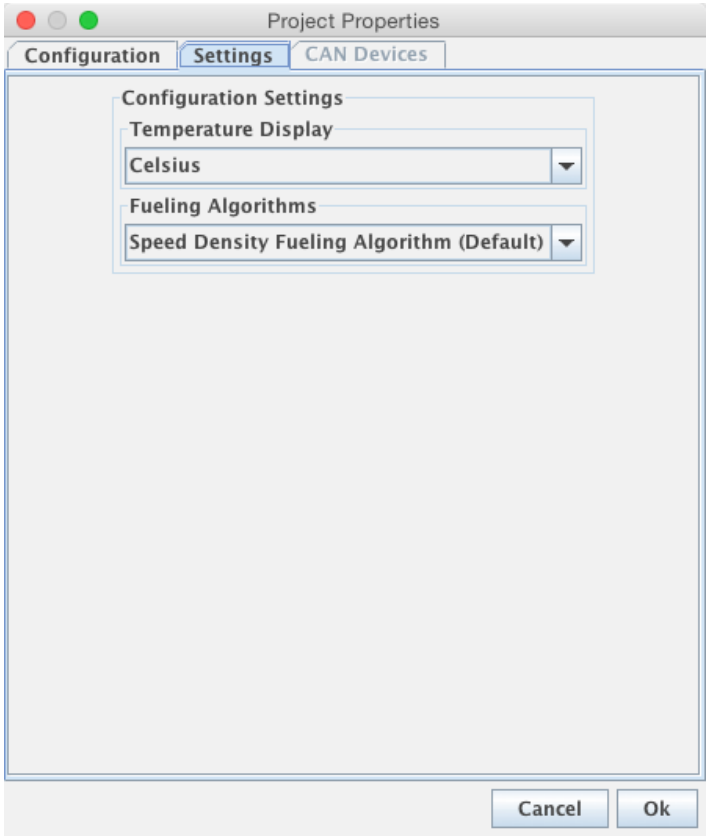


Figure 9: TS_4.png

Settings Tab

The Settings tab does not affect the tune directly, but does change the way some things are displayed within Tuner Studio. Some menus are hidden by default, either for safety reasons or because they are still under development, and they can be enabled here.

Lambda Display

This changes whether the oxygen sensor reasons are shown in AFR (default) or Lambda.

Temperature Display

The temperature selection changes all degrees values within TunerStudio.

- Fahrenheit(Default)

- Celsius

> Changing this value does not alter the values in tune at all, only which scale the values are displayed in {.is-info}

Enable Hardware Test

The hardware testing dialog allows you to manually turn the ignition and injection outputs on and off in order to test that the circuits are working. This can be dangerous if the outputs are connected to hardware however and so this dialog must be explicitely enabled.

> Please **ONLY** turn this on when the ECU is not connected to a vehicle {.is-danger}

If Enabled, an additional Tab will appear on the tuning page

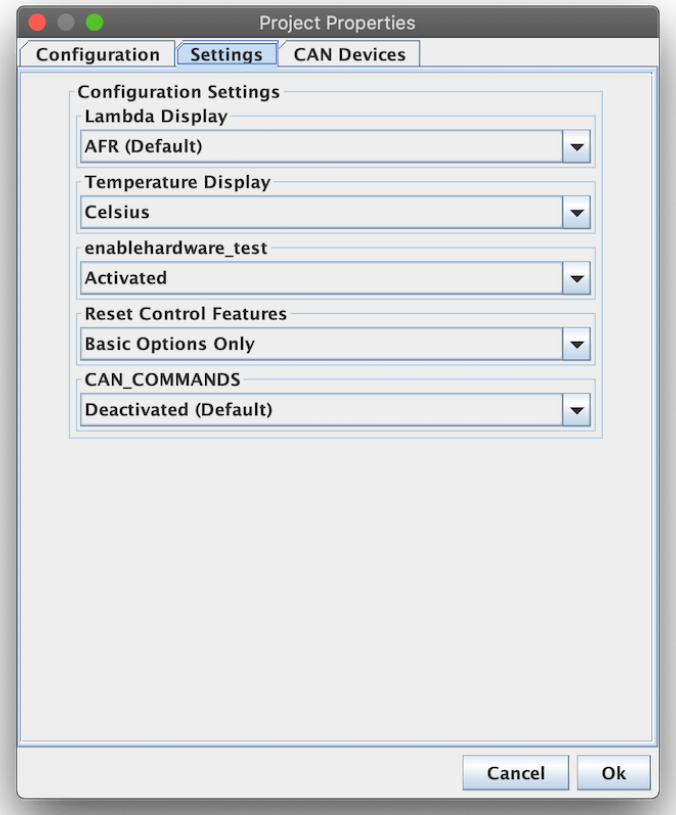


Figure 10: Project Settings

Reset control features

An optional Speeduino specific boot loader is available that has different methods of controlling the automatic reset. The vast majority of users should leave this on the default 'Basic options only'

CAN Devices Tab

CAN options are currently under development, but settings are available on this tab for testing if you have supported hardware.

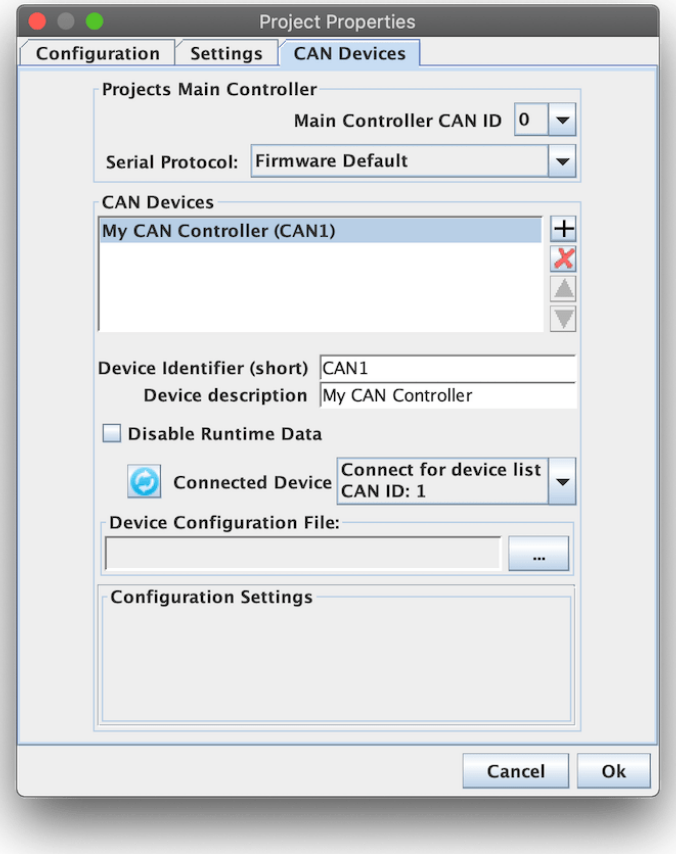


Figure 11: CAN Devices

Configuration of CAN devices is currently unsupported

High level wiring guide

Speeduino can be configured in many ways depending on the engine, sensors, ignition and fuel hardware being used. For this reason it is impossible to provide 1 single diagram that will cover all scenarios, however the below is provided as a high level guide that can be used as a starting point.

See the Hardware Requirements page for specific requirments and exceptions to the image below.

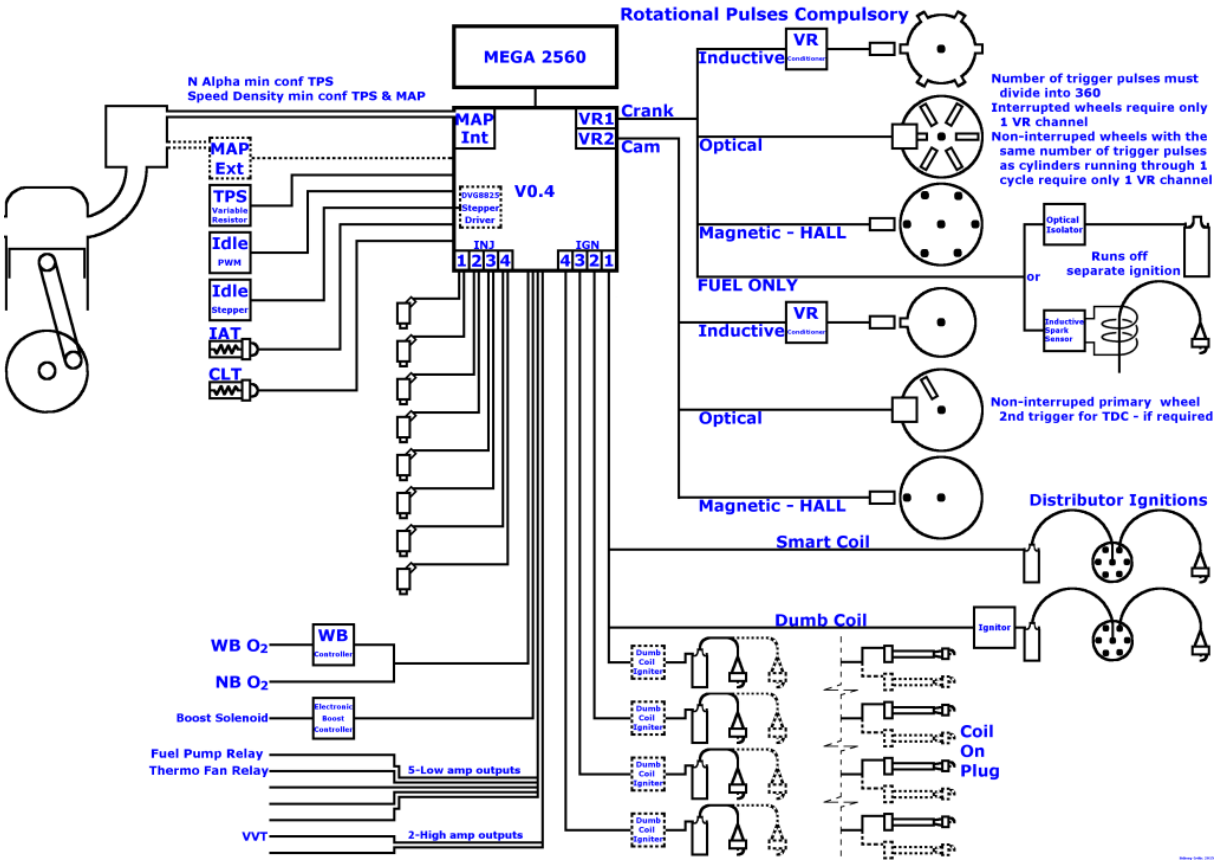


Figure 12: wiring_overview.png

Injector wiring

Overview

Speeduino contains 4 injector control circuits and is capable of supporting up to 8 injectors (and cylinders) with these.

Supported Injectors

Speeduino supports High-Z (aka 'high-impedance' or 'saturated') injectors natively. Low-Z injectors are supported with the addition of resistors wired in series with the signal wires. High-Z injectors are typically those with a resistance greater than 8 Ohms.

If "Low-Z" ("peak and hold" or PWM-controlled) injectors that are lower impedance are used, the wiring will require series resistors on each injector to avoid damaging the board with excessive current. The resistor ohms and watt rating can be calculated by Ohm's Law, or use an Internet calculator page such as the Speeduino Injector Resistor Calculator.

Layouts

There are a number of ways that the injectors can be wired depending on your configuration and preference.

1, 2 and 3 injectors

For these configurations, each injector is wired into it's own output from the Speeduino board.

4 injectors

For 4 cylinders/injectors, there are 2 ways that these can be connected to Speeduino:

Method 1 (Paired) The standard method is the same as that used for 6 or 8 cylinder setups, where 2 injectors are connected to each injector channel. In this configuration, only 2 injector channels will be used. The injectors paired together must have their Top Dead Centres (TDC) 360 crank degrees apart.

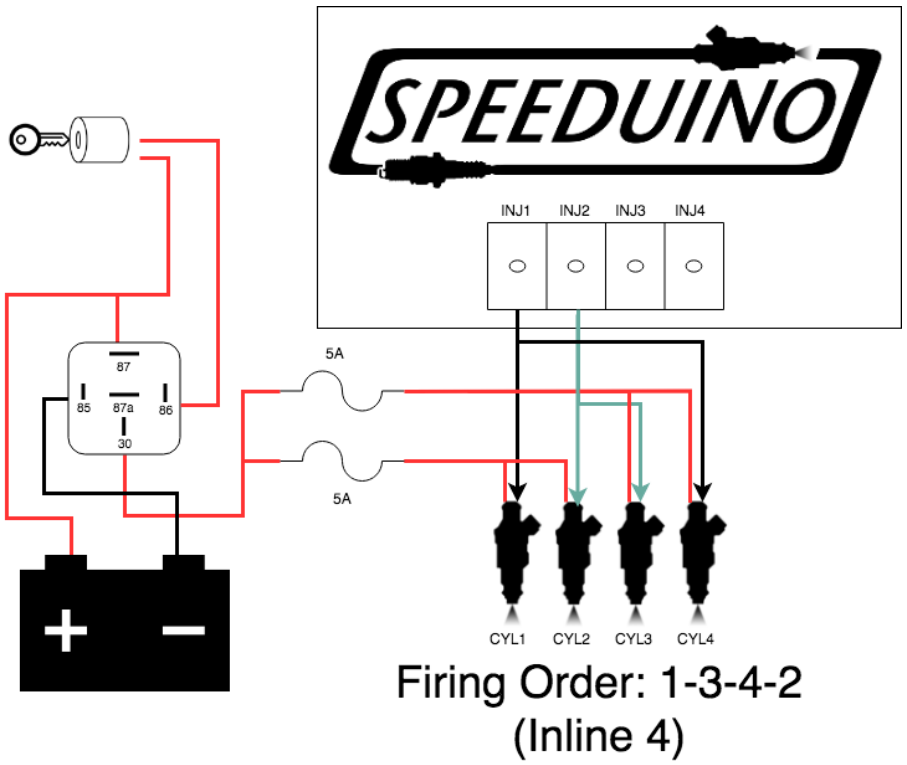


Figure 13: inj_4Cyl_semi-seq.png

Method 2 (Full sequential) This method is only available on 4 cylinder / 4 injector applications and allows you to wire 1 injector per channel. The injector channels always fire in numerical order (ie 1, 2, 3, 4) so your injectors should be wired to take your firing order into account. Within Tuner Studio, this option can be enabled by selecting:

`Settings -> Engine Constants -> Injector Timing -> Sequential`

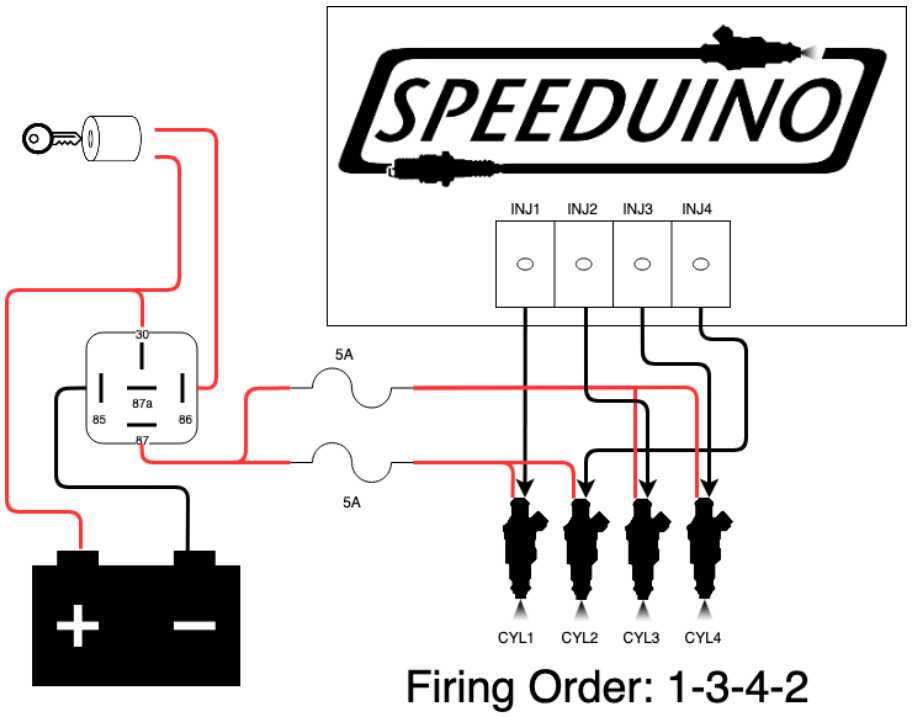


Figure 14: inj_4Cyl_seq.png

> **Note:** Using sequential fueling requires a compatible Cam signal to be used in addition to the Crank. If no cam signal is provided when the sequential option is selected, the system will not sync {.is-warning}

5 injectors

5 cylinder setups should be wired to use all 4 injector outputs with 2 injectors sharing output #3. For the typical inline 5 cylinder firing order (1-2-4-5-3), injectors 4 and 3 would be joined together on injector 3 output.

More than 5 injectors

For setups with more than 4 injectors, the number of outputs used will be equal to half the number of injectors.

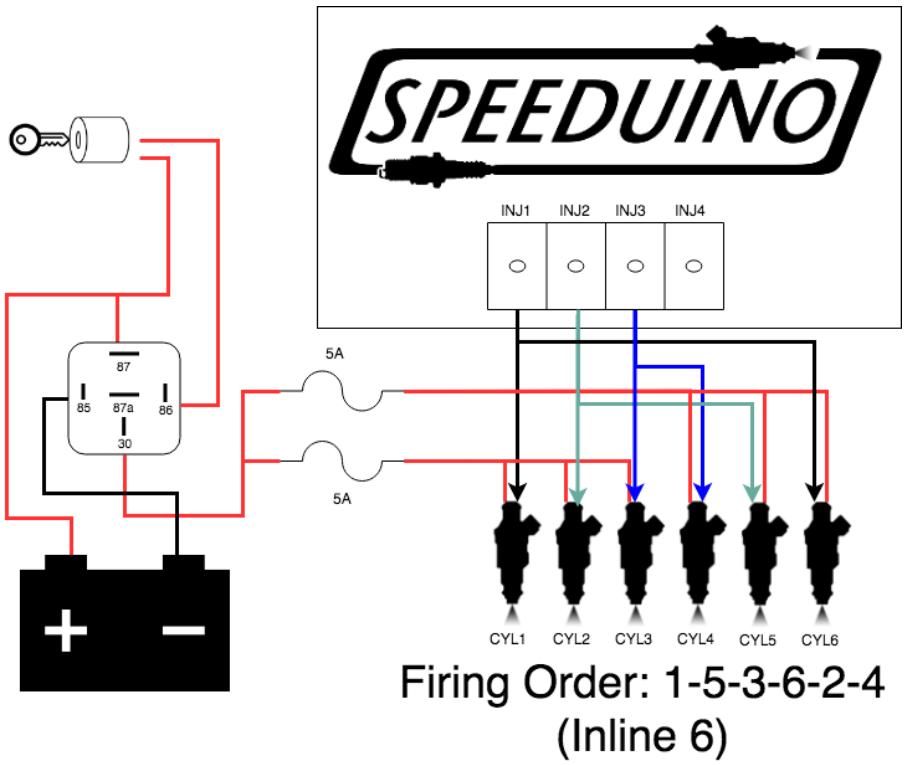


Figure 15: inj_6Cyl_semi-seq.png

6 Cylinder For a V6 with a firing order of (1,4,2,5,3,6) the injectors will be wired in 3 groups of (1,5) and (4,3), and (2,6) as these cylenders are 360 crank degrees apart.

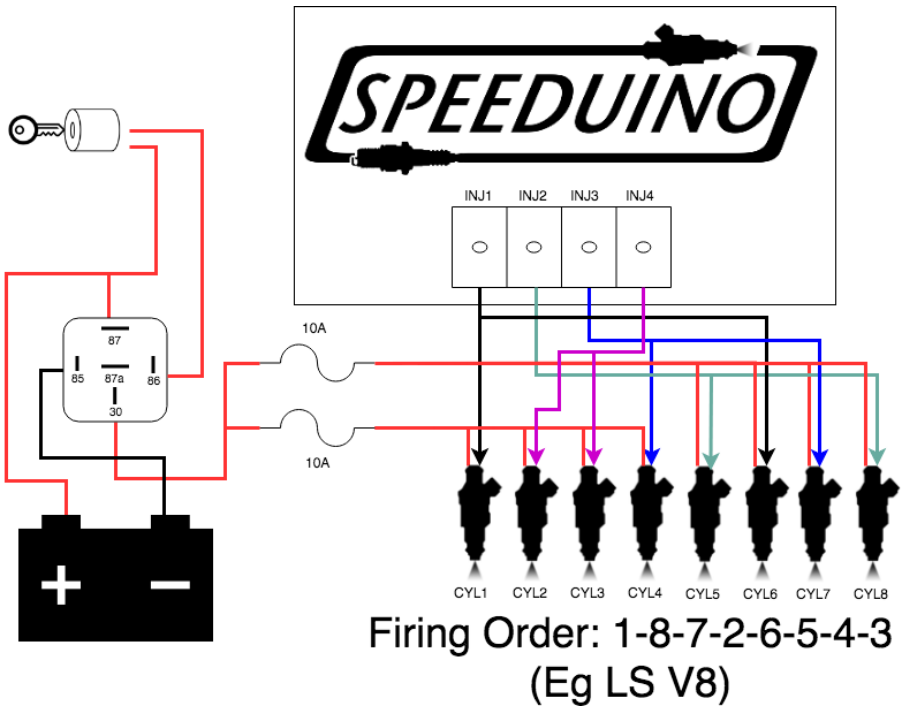


Figure 16: inj_8Cyl_semi-seq.png

8 Cylinder Inline with the above, this configuration requires each injector output to be connected to 2 injectors. The injectors should be grouped in opposing pairs, that is, cylinders whose Top Dead Centres are 360 degrees apart.

Ignition Wiring

Overview

Ignition output configuration can be one of the most difficult areas of ECU wiring and one that often causes the most confusion. A large part of this complexity comes from the huge number of different ignition types that are available, with there being significant changes in the hardware used in the late 80s and throughout the 90s compared to newer designs.

Whilst this guide does not cover all ignition styles and hardware, it does cover the most common scenarios. Generally, it is recommended (Where possible) to use newer styles of ignition hardware (Typically 'smart' Coil-on-Plug or Coil-Near-Plug) rather than utilising separate igntion modules.

Wasted Spark

Wasted spark is a common means of controlling spark that requires only half the number of ignition outputs as there are cylinders, with 2 cylinder being attached to each output. EG: * 4 cylinder engine requires 2 ignition outputs * 6 cylinder engine requires 3 ignition outputs * 8 cylinder engine requires 4 ignition outputs

Wasted Spark has the advantage of not requiring any cam signal or input as it does not need to know the engine phase. This is possible by firing the ignition outputs once per revolution and pairing that output to 2 cylinders that are both at TDC (With one cylinder on compression stroke and the other on exhaust)

When using wasted spark, it is critical the correct pairs coils and/or spark plugs are joined together.

There are many dual pole, wasted spark coil packs available both with and without built in igniters. Either are suitable for use with Speeduino, but use of coils with built-in igniters is recommended

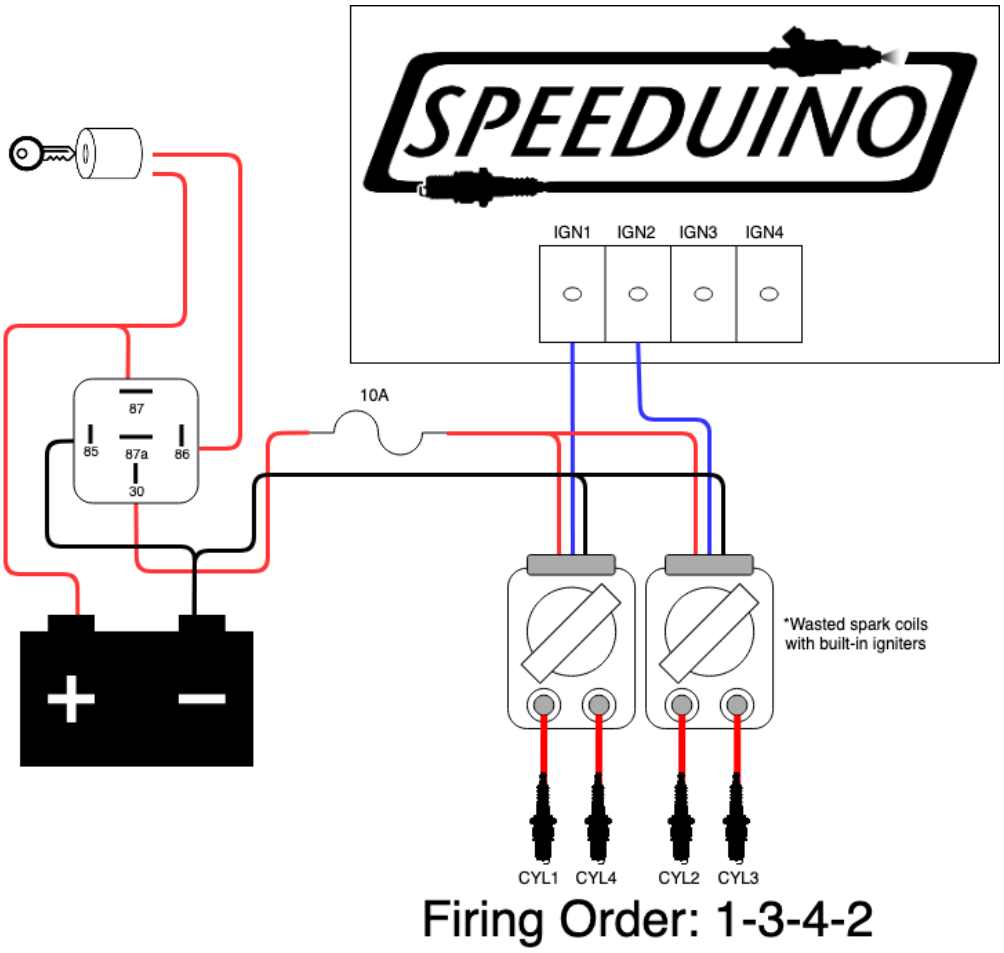


Figure 17: ign_4Cyl_COP_wasted-spark.png

> Note: The above example uses 'smart' coils with built in igniters. Do NOT attach high current (dumb) coils without adding an igniter {.is-warning}

Coil on Plug

As an alternative to a dual pole wasted spark coil, individual coil on plug units can be used in a wasted spark configuration.

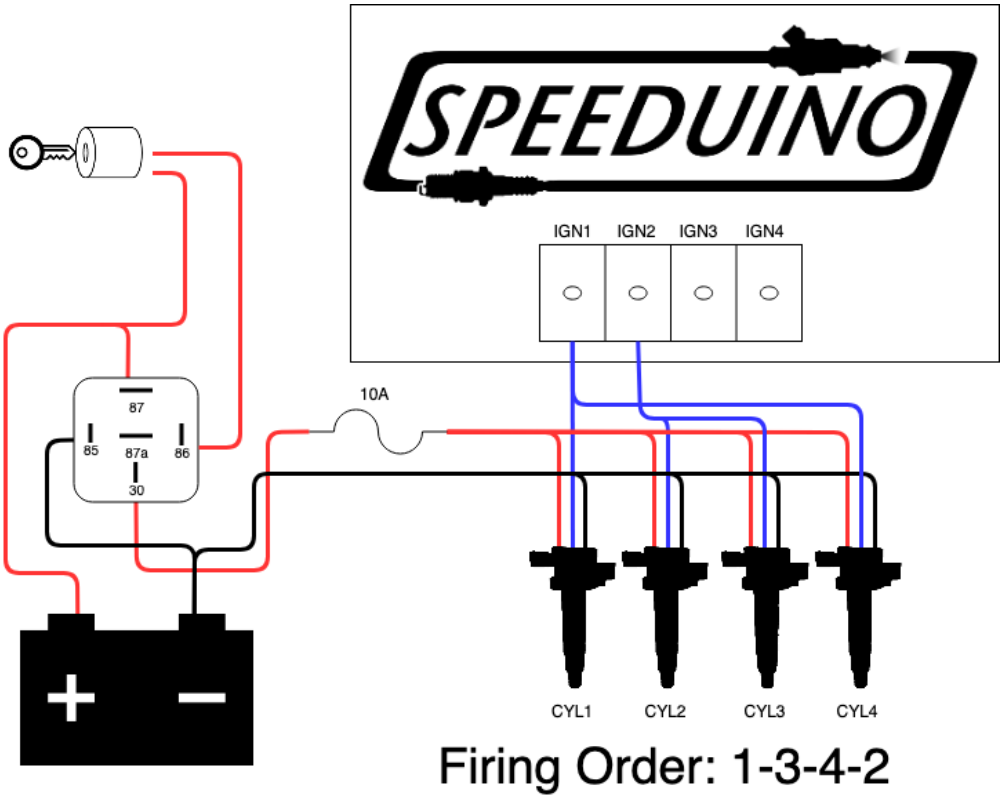


Figure 18: ign_4Cyl_COP_wasted-spark.png

> Note: The above examples use 'smart' coils with built in igniters. Do NOT attach dumb COPs (2 pins) without adding an igniter {.is-warning}

Sequential (COPs)

Sequential ignition control using Coil-on=Plugs coils dramtically simplifies the ignition wiring. With this configuration, each coil (and subsequently each cylinder) connects to a single ignition outpt, wired in the firing order.

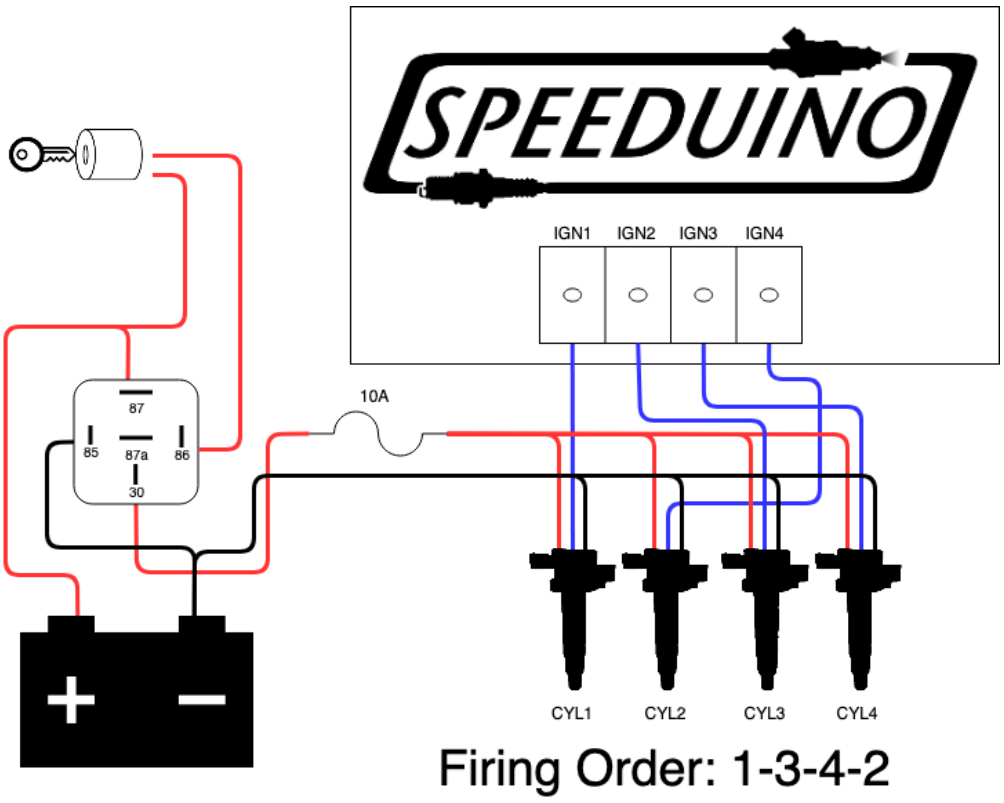
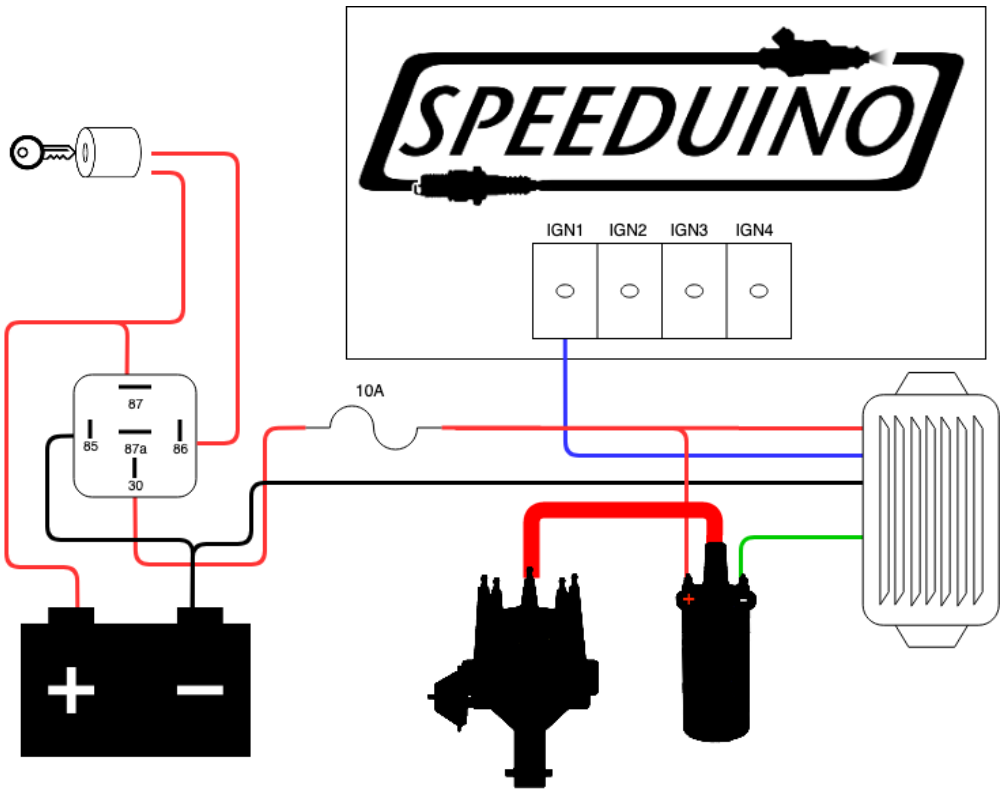


Figure 19: ign_4Cyl_COP_seq.png

> Note: The above example uses 'smart' coils with built in igniters. Do NOT attach dumb COPs (2 pins) without adding an igniter {.is-warning}

Distributor

If a distributor remains in use, only a single output is required from the ECU. This should be fed into a single channel ignition module (Such as the common Bosch 124) which can then drive the coil.



Analog Sensor Wiring

Analog sensors provide data such as temperatures, throttle position and O2 readings to the ECU. The diagram below shows the typical wiring for these sensors.

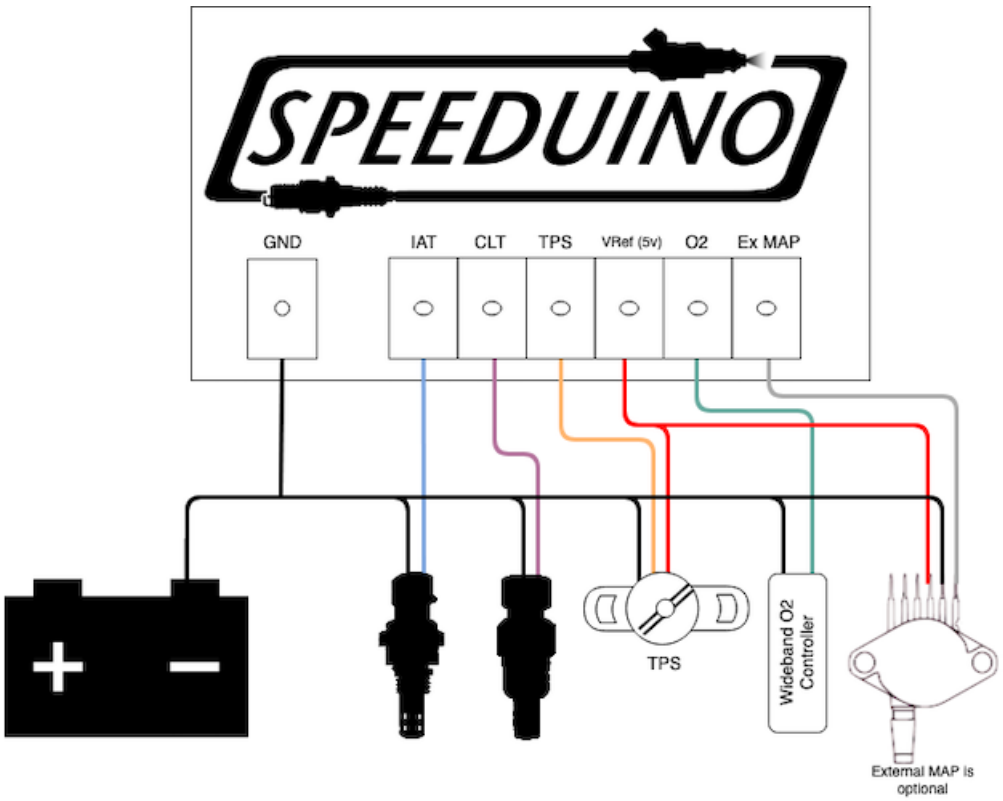


Figure 20: analog_sensors.png

Notes

- Use of 2 wire temperature sensors is **highly** recommended. Whilst 1 wire sensors will work, they are almost always considerably less accurate. Running a dedicated ground wire back to the ECU from the sensor is also recomended.
- The external MAP sensor in the above diagram is optional and may be omitted if the onboard MAP is used. Alternatively an external Baro sensor may be added in the same was as an external MAP
- A 3 wire variable TPS is required. On/Off type throttle switches are not suitable

Engine Constants

Overview

From the Settings menu, select Constants

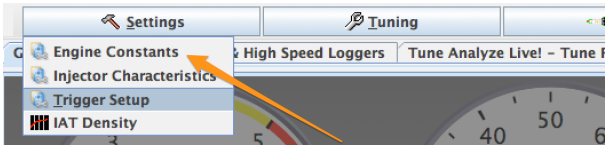


Figure 21: TS_8.png

Here you need to setup the engine constants. Fill out the fields in the bottom section before calculating the Required Fuel.

Configuration

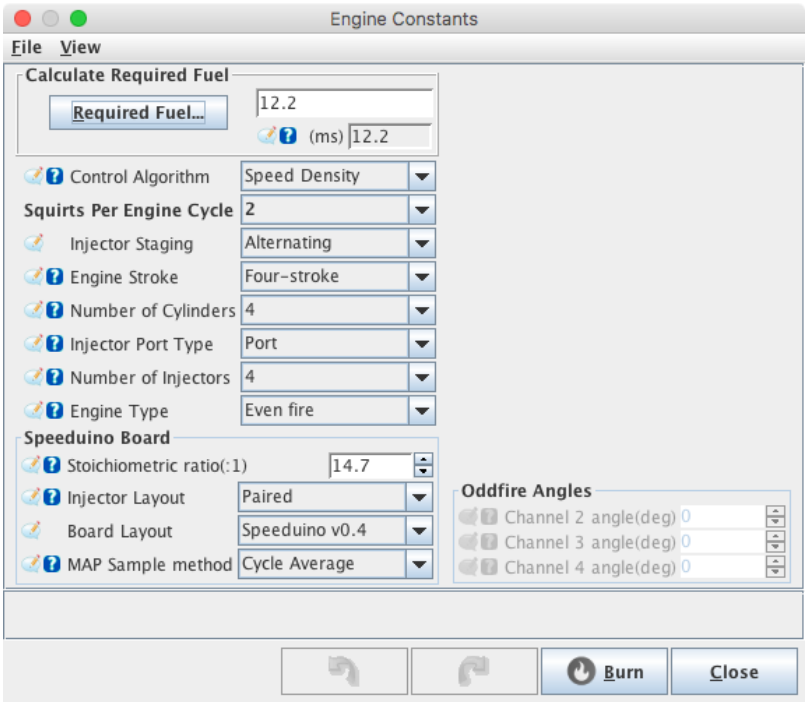


Figure 22: engine.png

Required Fuel Calculator

The required fuel calculator determines the theoretical fuel injection time that would be required at 100% VE. This is determined by knowing the engine capacity, the size and number of the fuel injectors and the number of squirts that will be performed in each cycle. Increasing this figure will lead to an overall increase in the amount of fuel that is injected **at all points** of the VE map (And vice versa).

> You should set all the values in the `Settings` section below before performing the `Required Fuel` calculation {.is-info}

Settings

- **Control Algorithm:** The load source that will be used for the fuel table
- **Squirts per Engine Cycle:** How many squirts will be performed over the duration of the engine cycle (Eg 720 degrees for a 4 stroke). most engines will not require values greater than 4. For sequential installations, this should be set to 2 with the injector staging set to 'Alternating'(Internally Speeduino will adjust the squirts to 1)
  - Note that for 3 and 5 squirts, you must have a cam signal in addition to the crank.
- **Injector Staging:** This configures the timing strategy used for the injectors
  - **Alternating** (Recommended for most installs) - Injectors are timed around each cylinders TDC. The exact closing angle can be specific in the Injector Characteristics dialog.
  - **Simultaneous** - All injectors are fired together, based on the TDC of cylinder 1.
- **Engine stroke:** Whether the engine is 2 stroke or 4 stroke
- **Number of cylinders:** Number of cylinders in the engine. For rotary engines, select 4.
- **Injector Port Type:** Option isn't used by firmware. Selection currently does not matter
- **Number of injectors:** Usually the same as number of cylinders (For port injection)
- **Engine Type:** Whether the crank angle between firings is the same for all cylinders. If using an Odd fire engine (Eg Some V-Twins and Buick V6s), the angle for each output channel must be specific.
- **Injector Layout:** Specifies how the injectors are wired in
  - **Paired:** 2 injectors are wired to each channel. The number of channels used is therefore equal to half the number of cylinders.
  - **Semi-Sequential:** Semi-sequential: Same as paired except that injector channels are mirrored (1&4, 2&3) meaning the number of outputs used are equal to the number of cylinders. Only valid for 4 cylinders or less.
  - **Sequential**: 1 injector per output and outputs used equals the number of cylinders. Injection is timed over full cycle. Only available for engines with 4 or fewer cylinders.
- **Board Layout:** Specifies the input/output pin layout based on which Speeduino board you're using. For specific details of these pin mappings, see the utils.ino file
- **MAP Sample Method:** How the MAP sensor readings will be processed:
  - **Instantaneous:** Every reading is used as it is taken. Makes for a highly fluctuating signal, but can be useful for testing

- **Cycle Average:** The average sensor reading across 720 crank degrees is used. This is of Event average are the recommended options for 4 of more cylinders
- **Cycle Minimum:** The lowest value detected across 720 degrees is used. This is the recommended method for less than 4 cylinders or ITBs
- **Event Average:** Similar to Cycle Average, however performs the averaging once per ignition event rather than once per cycle. Generally offers faster response with a similar level of accuracy.

The Oddfire angles should only be used on oddfire engines (Primarily some specifica V6s)

Injector Characteristics

Overview

Fuel injectors have unique hardware properties that must be accounted for within your tune. Ideally these will be provided as part of the specifications for your injectors, however in some cases the data may not be avaialble or be difficult to find. Typical values are given below as starting points for these cases.

Settings

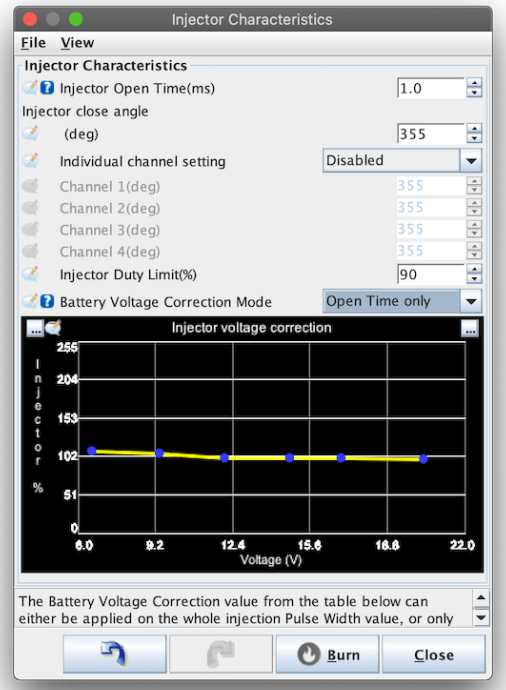


Figure 23: Injector Characteristics

Variable	Typical value	Comment
Injector Open Time	0.9 - 1.5	The time the injector takes to open completely once triggered, plus the time necessary to close. This is specific to each injector type and version.
Injector Close Angles	355	This represents the angle, relative to each cylinders TDC, that the injector squirt will end. This can be varied per channel (Including for semi-sequential wiring), but the default value of 355 is suitable for most applications.

Variable	Typical value	Comment
Injector Duty Limit	85%	The injector opens and closes once per crank revolution so, taking into account the open time of the injector, the duty cycle is limited to avoid this exceeding the revolution time. A value of 85% is recommended, but a higher value can be used for faster opening injectors. Note that once this duty cycle limit is reached, it will not be exceeded as the fuel injector cannot close and reopen fast enough to supply more fuel. This may potentially cause lean conditions at high RPM. If hitting this limit, strongly consider whether larger injectors are required.
Injector Voltage Correction	100%	The percentage the the injector pulse width is varied with changes in supply voltage. A value of 100% means no change to the pulse width.
Voltage Correction Mode	Open time only	Whether the voltage correction applies to just the opening time or the whole pulse width.

Trigger Setup

Overview

One of the most critical components of an EFI setup is the Crank Angle Sensor (CAS) and how it is used by the ECU. The Trigger settings dialog is where the trigger configuration is defined and it is vitally important to have this correct before trying to start your engine.

With incorrect settings, you may have issues getting sync or see erratic RPM readings.

Note that many of the settings on this dialog are dependant on your configuration and it is therefore normal that some options maybe greyed out.

Trigger Settings

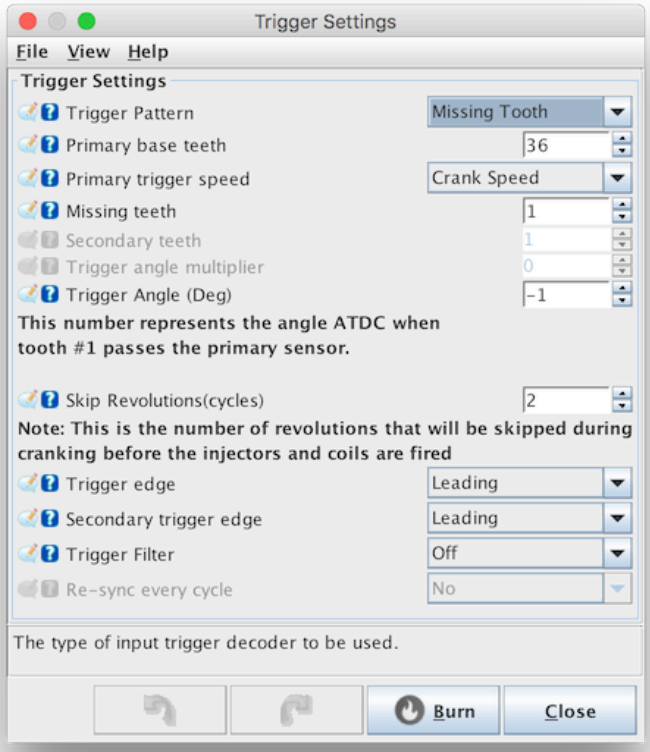


Figure 24: Trigger settings dialog

- **Trigger Pattern** - The pattern used by the crank/cam sensor setup on your engine. For a full list of the supported patterns, see the Decoders page
- **Primary Base teeth** - For patterns where the number of teeth are variable (missing tooth, dual wheel etc), this number represents the number of teeth on the primary wheel. For missing tooth type wheels, this number should be the count as if there were no teeth missing.
- **Primary trigger speed** - The speed at which the primary input spins. It is closely related to the Primary Base teeth setting and indicates whether that number of teeth passes the sensor once every crank revolution or every cam revolution.
- **Missing teeth** - If using the missing tooth pattern, this is the size of the gap, given in 'missing teeth'. Eg 36-1 has 1 missing tooth. 60-2 has 2 missing teeth etc. The missing teeth **MUST** be all located in a single block, there cannot be multiple missing tooth gaps around the wheel.
- **Secondary teeth** - As above, but for the secondary input. This input is always assumed to run

at cam speed.
- **Trigger angle multiplier** - This option is used only on the Non-360 pattern.
- **Trigger angle** - The angle of the crank, **After Top Dead Centre (ATDC)**, when tooth #1 passes the sensor on the primary (crank) input. This setting is critical for Speeduino to accurately know the current crank angle. See section below ('Finding tooth #1 and trigger angle') for further information on how to determine this value. You should be using a timing light to confirm angle is correct once calculated. Without doing this your angle may be incorrect.

Trigger options

- **Skip revolutions** - The number of revolutions the engine should perform before the Sync flag is set. This can help prevent false sync events when cranking. Typical values are from 0 to 2
- **Trigger edge** - Whether the primary signal triggers on the Rising or Falling edge. VR Conditioners require specific setting depending on model used. See hardware requirements
- **Secondary trigger edge** -Whether the secondary signal triggers on the Rising or Falling edge
- **Trigger filter** - A time based software filter that will ignore crank/cam inputs if they arrive sooner than expected based on the current RPM. The more aggressive the filter, the closer to the expected time the filter will operate. Higher levels of filtering may cause true pulses to be filtered out however, so it is recommended to use the lowest setting possible
- **Re-sync every cycle** - If set to yes, the system will look for the sync conditions every cycle rather than just counting the expected number of teeth. It is recommended that this option should be turned on, however if you have a noisy crank/cam signal you may need to turn it off as it can cause sync to drop out occassionally. Once Speeduino has full sync it will continue to run in full sequential mode unless sync loss on crack trigger occurs.

Finding tooth #1 and trigger angle

Please refer to the Trigger Patterns and Decoders for the trigger that you are using

IAT Density

Overview

The IAT density curve represents the change in oxygen density of the inlet charge as temperature rises.

Example Curve

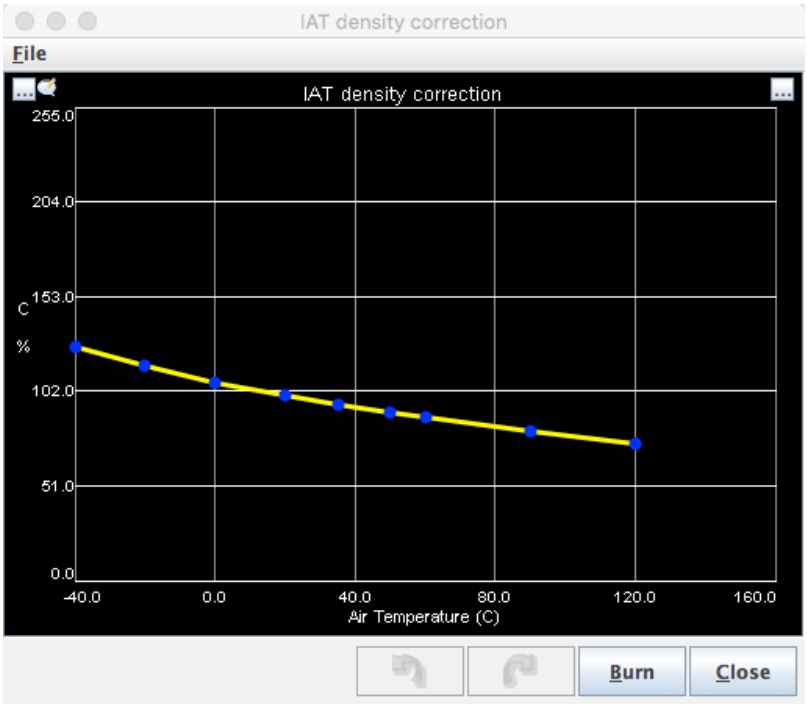


Figure 25: iatDensity.png

This default curve approximately follows the ideal gas law and is suitable for most installations, however if you are seeing very high inlet temperatures (Either due to heat soak in the engine bay or from turbocharging) the you may need to adjust the hot end of this curve.

Fuel (VE) table

The fuel or VE table is the primary method of controlling the amount of fuel that will be injected at each speed/load point.

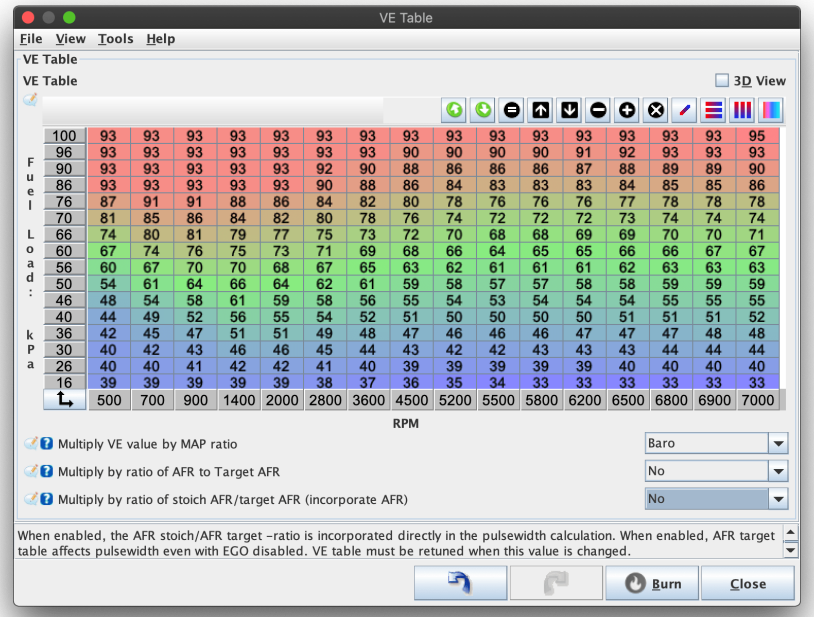


Figure 26: VE Table 1

Configuration

The fuel map is a 3D, interpolated table that uses RPM and fuel load to lookup the desired VE value. The fuel load axis is determined by whether you are using Speed Density (MAP kPa) or Alpha-N (TPS) for your fuel load (See Engine_Constants)

The values in this table represent a percentage of the `Required Fuel` amount that will be injected when the engine is at a given speed/load point.

Options

- **Multiply VE value by MAP ratio:** Enabling this option 'flattens' the fuel table by multiplying the value in the current speed/load point by the MAP value divided by either the Baro value (in kPa) or a fixed 100% (For compatibility with tunes from other systems).
  - You can tune with or without this option enabled, but it is generally recommended to be turned on as it will allow for simpler and more predictable tuning results.
  - For new tunes it is recommended to use the `Baro` option

> **Warning:** Changing this value will require retuning of the fuel map!{.is-warning}

- **Multiply by ration of AFR to Target AFR:** This option is normally set to No for most setups. It allows basic close loop feedback by adjust the base fuel amount according to how far away from the target AFR the engine is currently running.
- **Multiply by ratio of stoich AFR to target AFR ('Incorporate AFR')**: This option is similar to the one above, but instead of using the current AFR value, the fixed stoich AFR is used.

Secondary Fuel table

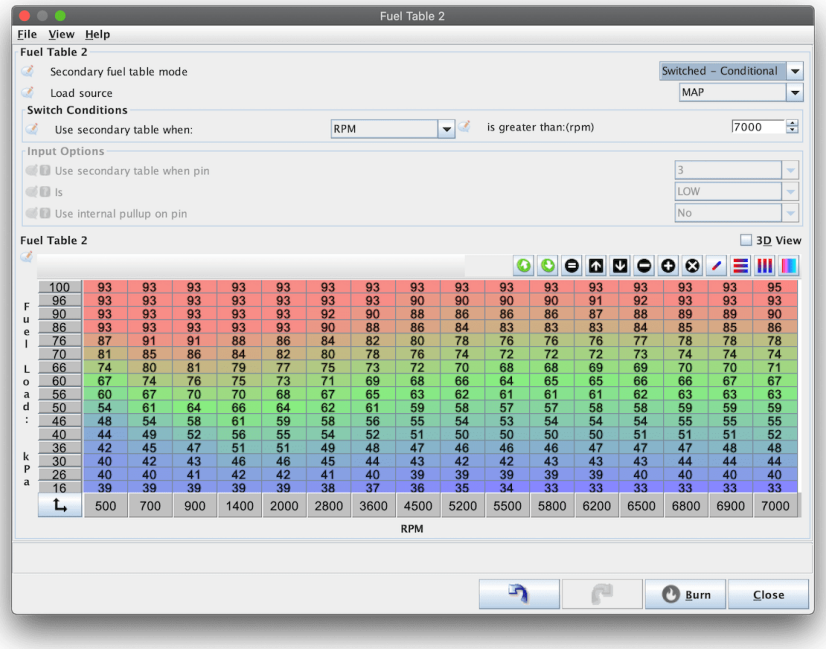


Figure 27: Secondary Fuel table

Speeduino also has the ability to use a secondary fuel table which allows for blended and switched mode fueling. There are 2 blended modes and 2 switched modes available.

Blended fuel modes work in conjunction with the primary fuel table to come up with a single, combined VE. Switched fuel modes are where either the primary or secondary fuel table is used, but not both at the same time. Which table is being used at any given time can be configured based on either an external input (Eg dash switch) or set via certain conditions.

Multiplied %

This is a blended fuel mode (ie it uses both the primary and secondary fuel tables together) that allows for different load and RPM axis to be combined. Commonly this is used for having primary and secondary fuel tables with different load sources (**Eg:** Primary map using TPS and secondary map using manifold pressure).

This mode is often used on engines with Individual Throttle Bodies (ITBs) to allow TPS and MAP based tables to be combined.

The final fuel value is derived from treating both values (Primary and Secondary) as percentages and multiplying them together.

Example 1

- **Primary Fuel table value:** 75
- **Secondary fuel table value:** 100
- **Final value:** 75

Example 2

- **Primary Fuel table value:** 80
- **Secondary fuel table value:** 150
- **Final value:** 120

Example 3

- **Primary Fuel table value:** 90
- **Secondary fuel table value:** 80
- **Final value:** 72

Added

This is a blended fuel mode that is very similar to the above `Multipled %` mode. The only difference between the two is that instead of multiplying the values from the primary and secondary tables, the 2 are added together.

This is a less commonly used mode, but is an alternative in the same setups that you would use `Multiplied %`

Switched - Conditional

Conditional switched mode will allow use of the 2nd fuel table when a certain value goes above a defined level. The available switching values are:

- RPM
- Ethanol content
- MAP
- TPS

Dpending on the desired outcome, this can be used to expand the resolution of the main fuel table, automatically handle alternate fuels or as an alternative ITB mode (Particularly if running boosted ITBs).

Switched - Input based

Input based switch mode let's you change the fuel table that is in use via an external input to the ECU. The options required are:

- The (Arduino) pin that the input is connected to
- The polarity of this input (IE Is the secondary fuel table used with the signal is high or low). For a standard ground switching input, this should be `LOW`
- Whether to use the internal pullup on this input. For a standard ground switching input, this should be `Yes`

Acceleration Enrichment (AE)

Acceleration Enrichment (AE) is used to add extra fuel during the short transient period following a rapid increase in throttle. It performs much the same function as an accelerator pump on a carbureted engine, increasing the amount of fuel delivered until the manifold pressure reading adjusts based on the new load.

> To operate TPS based AE correctly, you must have a variable TPS installed and calibrated. {.is-info}

Theory

Tuning of acceleration enrichment is based on the rate of change of the throttle position, a variable known as TPSdot (TPS delta over time). This is measured in %/second, with higher values represent-

ing faster presses of the throttle and values in the range 50%/s to 1000%/s are normal. Eg:

- 100%/s = pressing the throttle from 0% to 100% in 1 second
- 1000%/s = pressing the throttle from 0% to 100% in 0.1s

TPSdot forms the X axis of the acceleration curve, with the Y axis value representing the % increase in fuel.

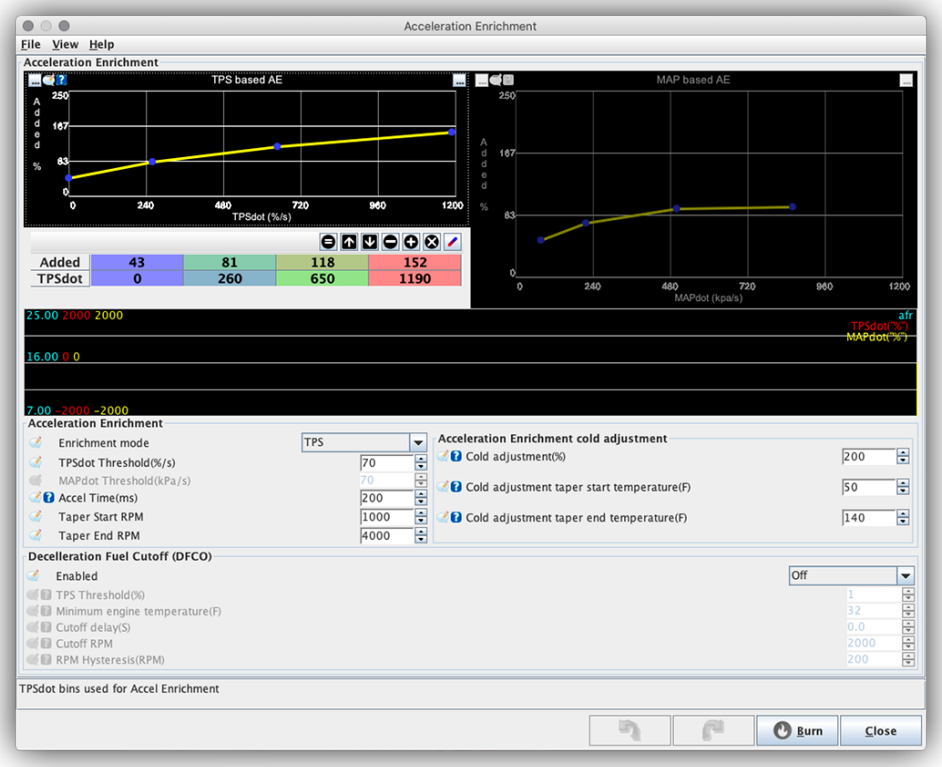


Figure 28: Acceleration Enrichment curves

Tuning

The enrichment curve included with the base Speeduino tune is a good starting point for most engines, but some adjustment is normal depending on injector size, throttle diameter etc.

In most cases, tuning of the AE curve can be performed in a stationary environment, though dyno or road tuning is also possible. Fast and slow blips of the throttle should be performed and the affect on the AFRs monitored using the live line graph on the AE dialog. This graph shows both TPSdot and AFR values in sync with each other, making adjustments to the correct part of the AE curve simpler to identify.

If you find that the AFR is initially good, but then goes briefly lean, you should increase the 'Accel Time' setting, with increments of 10-20ms recommended.

False triggering

In cases where the TPS signal is noisy, spikes in its reading may incorrectly trigger the acceleration enrichment. This can be seen in a log file or on a live dash in TunerStudio by the activation of the 'TPS Accel' indicator when there is no (or little) throttle movement occurring.

Should this occur (and assuming that the TPS wiring cannot be corrected to reduce noise) then the false triggers can be prevented from triggering AE by increasing the "TPSdot Threshold" value. This should be increased in increments of ~5%/s, pausing between each increase to observe whether AE is still being incorrectly activated.

Fields

- **Enrichment Mode** Chose whether to use Throttle Position Sensor or Manifold Absolute Pressure for acceleration enrichment.
- **TPSdot Threshold** Percentage of throttle position change per second required to trigger acceleration enrichment. For example, if set to 70, the throttle position must change at a rate of 70% per second for acceleration enrichment to become active.
- **MAPdot Threshold** Same as TPSdot Threshold, but applies when using MAP enrichment mode.
- **Accel Time** Duration of acceleration enrichment. Once enrichment is triggered, it will last this many milliseconds.
- **Taper Start RPM, Taper End RPM** Scales the enrichment taper at different RPMs. If RPM is less than or equal to Start RPM, enrichment will be 100% of the calculated enrichment value, based on the TPSdot(or MAPdot) value seen. If RPM is greater than or equal to End RPM, enrichment will be 0%. As RPM increases, the total amount of required enrichment decreases. Enrichment is scaled linearly in between these values.
- **Cold Adjustment** Scales the acceleration enrichment percentage linearly based on coolant temperature. At Start Temperature, adjustment will be equal to the Cold Adjustment field (%). At End Temperature, adjustment will be 0%.
- **Deceleration Fuel Cutoff** Stops injecting fuel when: *RPM is above **Cutoff RPM** TPS is below **TPS Threshhold** Engine temperature is above **Minimum engine temperature** The above conditions are met for **Cutoff delay** seconds* ** RPM Hysteresis can be adjusted to account for fluctuating RPM conditions to prevent accidental DFCO.

AFR/O2 (Closed loop fuel)

AFR/O2 (for **A**ir:**F**uel **R**atio), dialog controls the closed loop fuel control, used for adjusting injector load based on input from an exhaust oxygen sensor (O2 sensor). In conjunction with the AFR Table, the closed loop AFR system will compare the actual O2 reading with the current target fuel ratio and make adjustments accordingly.

Use of a wideband sensor and controller is **strongly** recommended, however basic functionality is possible with a narrowband sensor if this is not available.

Note that closed loop fuel control is not a replacement for a poor tune. Many good configurations do not used closed loop control at all or only allow it very small adjustment authority.

Settings

Speeduino supports 2 closed loop algorithms, each intended for different configurations:

1. **Simple** - A time based 'target chasing' algorithm where the amount of fuel adjustment is dependant on how long the reading has been lean or rich compared to the current target. This algorithm is best suited to narrowband sensors where only basic rich/lean information is available. In particular, this algorithm performs poorly if you have a fuel map that is not close to complete. If you have this enabled and are seeing oscillations in the pulse width and/or AFRs, even when cruising, then you should disabled closed loop control until the base fuel MAP is better tuned.
2. **PID** - This is the preferred closed loop algorithm and will provide better results when combined with a wideband sensor and tuned correctly.

Common variables

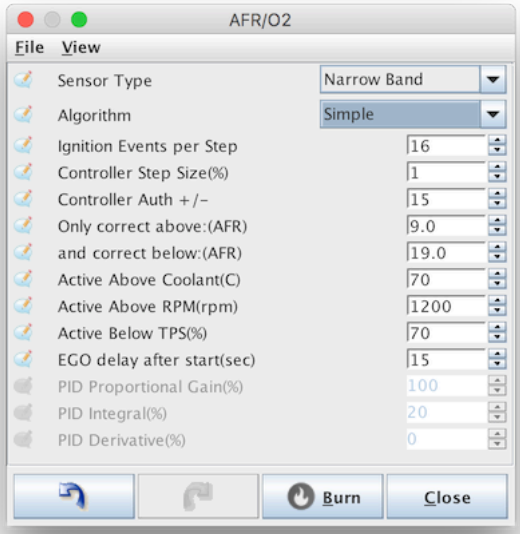


Figure 29: o2_simple.png

- **Sensor type** - Narrowband or wideband, depending on hardware configuration. Narrowband sensor should be of the 0-1v type, wideband sensors should have a 0-5v signal. Wideband sensors need to be calibrated in the Tools->Calibrate AFR Table dialog
- **Algorithm** - See above for description of each algorithm available
- **Ignition events per step** - The AFR adjustment calculation will be performed every this many ignition cycles. Changes to closed loop adjustment typically have some lag before their impact is registered by the O2 sensor and increasing this value can take this lag into account. Typical values are 2-5.
- **Controller step size** -
- **Controller Auth** - The maximum % that the pulse width can be changed through this closed loop adjustment. Recommended value is no more than 20%.
- **Correct above/below AFR** - The AFR range that closed loop adjustments will be applied within. This range is typically limited by the sensor and controller in use.
- **Active above Coolant** - Closed loop should only operate once engine is up to operating temperature. This value should be set to match the engines standard operating temp.
- **Active above RPM** - Closed loop adjustments should genrally not be made at idle. Use this value to specify when adjustment should begin being made.
- **Active below TPS** - Above this TPS value, closed loops adjustments will be disabled

- **EGO delay after start** - All O2 sensors require a warmup period before their readings are valid. This varies based on the sensor in use, but 15s is a safe value in most cases.

PID only variables

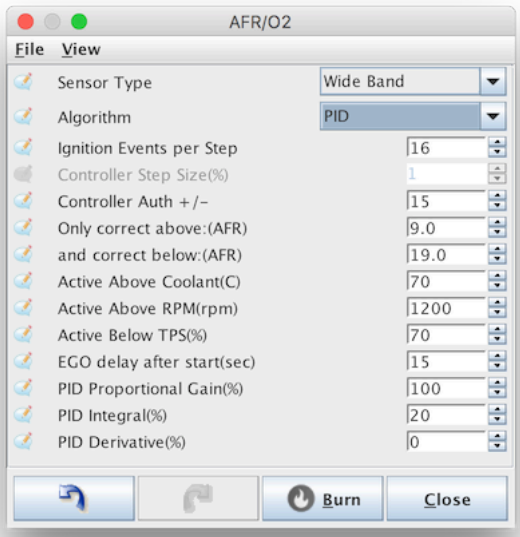


Figure 30: o2_pid.png

- **P/I/D** - PID Proportional Gain, Integral and Derivative percentages.

These options are in addition to the Simple conditions and specify the parameters of the closed loop operation

Limiters

Speeduino includes a spark based rev limited with both hard and soft cuts.

The soft cut limiter will lock timing at an absolute value to slow further acceleration. If RPMs continue to climb and reach the hard cut limit, ignition events will cease until the RPM drop below this threshold.

> **Note** As this is spark based limiting, fuel only installs cannot use the rev limiter functionality {.is-info}

Settings

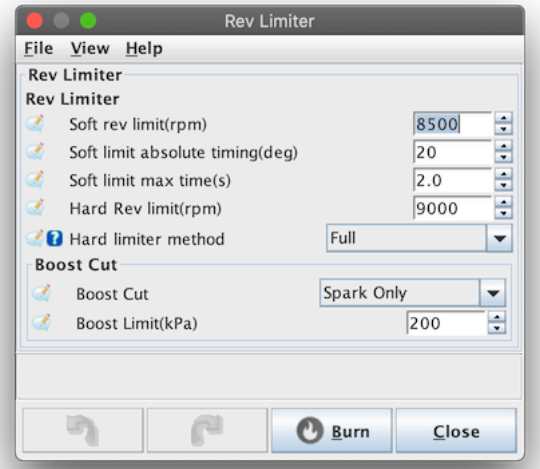


Figure 31: Rev limiter settings

- **Soft rev limit:** The RPM at which the soft cut ignition timing will be applied over.
- **Soft limit absolute timing:** Whilst the engine is over the soft limit RPM, the ignition advance will be held at this value. Lower values here will have a greater soft cut affect.
- **Soft limit max time:** The maximum number of seconds that the soft limiter will operate for. If the engine remains in the soft cut RPM region longer than this, the hard cut will be applied.
- **Hard rev limiter:** Above this RPM, all ignition events will cease.

Flex Fuel

Overview

Speeduino has the ability to modify fuel and ignition settings based on the ethanol content of the fuel being used, a practice typically known as flex fuelling. A flex fuel sensor is installed in the feed or return fuel lines and a signal wire is used as an input on the Speeduino board.

As ethanol is less energy dense, but also has a higher equivalent octane rating, adjustments to the fuel load and ignition timing are required.

Hardware

Speeduino uses any of the standard GM/Continental Flex fuel sensors that are widely available and were used across a wide range of vehicles. These were available in 3 different units, all of which are functionally identical, with the main difference being only the physical size and connector. The part numbers for these are:

- Small - #13577429
- Mid-size - #13577379
- Wide - #13577394 (Same as the mid-size one, but with longer pipes)

All 3 use a variant of the Delphi GT150 series connector. You can use a generic GT150 connector, but you will have to clip off 2 tabs from the side of the sensor.

Part numbers :

- Housing (#13519047)
- Pins (#15326427)
- Seal (#15366021)

Alternatively, there is a GM part for a harness connector, part number 13352241: http://www.gmpartsdirect.com/oe-gm/13352241

Wiring

All units are wired identically and have markings on the housing indicating what each pin is for (12v, ground and signal) Speeduino boards v0.3.5+ and v0.4.3+ have an input location on their proto areas that the signal wire can be directly connected to.

On boards earlier to these, you will need to add a pullup resistor of between 2k and 3.5k Ohm. Recommended value is 3.3k, however any resistor in this range will work. Note that this is a relatively strict range, more generic values such as 1k or 10k DO NOT WORK with these sensors.

Tuning

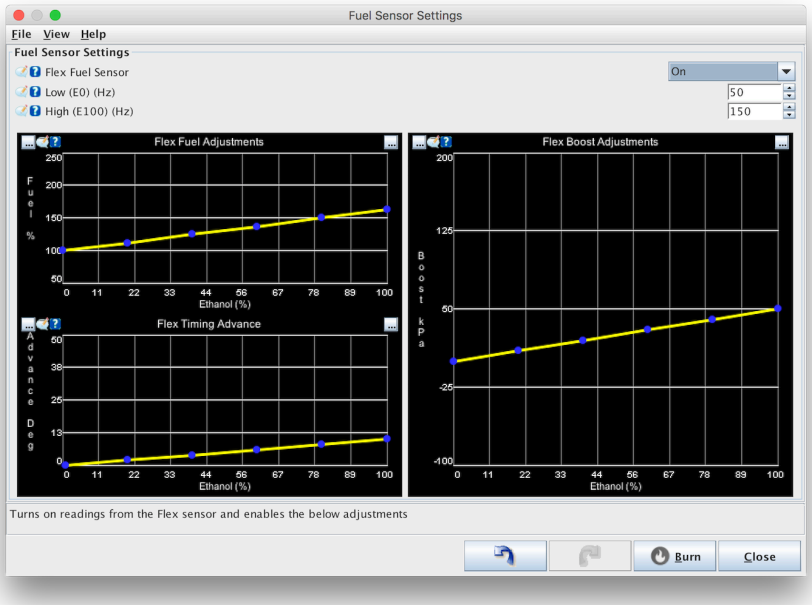


Figure 32: flex_settings.png

- **Sensor frequency -** The minimum and maximum frequency of the sensor that represent 0% and 100% ethanol respectively. For standard GM/Continental flex sensors, these values are 50 and 150
- **Fuel multiplier% -** This is the additional fuel that should be added as ethanol content increases. The Low value on the left represents the adjustment to the fuel map at 0% ethanol and will typically be 100% if the base tune was performed with E0 fuel. If the base tune was made with E10 or E15 however, this value can be adjusted below 100%. The high value represents the fuel multiplier at 100% ethanol (E100) and the default value of 163% is based on the theoretical difference in energy density between E0 and E100. Tuning of this value may be required
- **Additional advance -** The additional degrees of advance that will be applied as ethanol content increases. This amount increases linearly between the low and high values and is added after all other ignition modifiers have been applied.

Staged Injection

Overview

Speeduino has the ability to control a secondary fuel stage for engines that have 2 sets of injectors, typically of different capacities. Whilst there are few stock engines that come with secondary injectors (the notable exception being many Mazda rotaries) secondary staged injection is a common modification, in particular used whenever large injectors are required, but where it is desirable to keep smaller injectors for smoother low RPM performance.

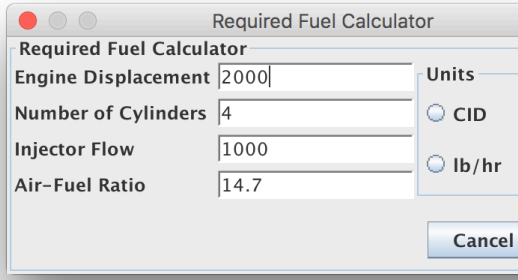
Configuration

No matter which control strategy is chosen, you must enter the sizing of the primary and secondary injectors in order to allow Speeduino to know the split in the overall fuelling.

> **CRITICAL** - The req-Fuel value in the Engine Constants MUST be updated when staged injection is turned on. **When staging is in use, the value entered in the req_fuel calculator MUST be equal to the sum of both the primary and secondary injector sizes** Failure to set these values correctly will result in excessive rich or lean conditions. {.is-danger}

Eg:

- **Primary Injectors :** 300cc
- **Secondary Injectors :** 700cc
- **Value entered into the req_fuel calculator :** 1000cc



Control methods

Speeduino provides 2 staging control modes, each with their own strengths and weaknesses. In most cases it is recommended to start with the Automatic mode, which only requires tuning of the standard VE table, and reviewing to see if you get the desired outcome. Only if this can't be tuned to give a satisfactory fuel split would changing to the manual table tuning be recommended.

Automatic staging When using the automatic staging method Speeduino takes into account the full capacity of the injectors (ie the sum of the 2 injector stages) and will perform a split of these itself. With this method, the user can simply tune the VE table in the same manner as if only a single set of injectors were used and the system takes care of the rest.

In this mode, Speeduino will attempt to use the primary injectors up to their 'Injector Duty Limit' (As configured in the Injector Characteristics dialog. When staging is being used, it is recommend that this limit should be no higher than 85%. Once the primary injectors reach this duty limit, Speeduino

will begin to perform any further fueling from the secondary injectors. In this way, the VE table is all that is required for tuning as the system will take care of allocating the current fuel load to the best injectors.

Table control Table control allows the use of a manual 8x8 map that indicates what percentage of the fuel load will be performed by the **secondary** injectors - 0% = Secondary injectors disabled - 100% = Primary injectors disabled

It is important to note that the values in this table do NOT correspond directly to the split of the duty cycle or pulse width. They represent the percentage of the total fuel load that the secondaries will be asked to perform. The affect this value has on the pulse width depends on the ratio of the primary and secondary injector capacities.

One disadvantage of the table tuning method is that it does not allow for the full fuel load of the primary and secondary injectors to be used simultaneously. As the table is a split of the total fuel load, as one set of injectors performs more, the other will perform less.

Note

The dead time of the 2 sets of injectors is currently assumed to be the same. This may be altered in future firmwares if required (Post a feature request if needed).

Spark Settings

Overview

The Spark settings dialog contains the options for how the ignition outputs will function, including which of the 4 IGN outputs are used and how. They are critical and incorrect values will result in an engine not starting and in some cases damage to hardware is possible. This dialog also contains a number of options for fixing the ignition timing for testing and diagnosis.

Please ensure you have reviewed these settings prior to attempting to start your engine.

To generate a base timing map that will give you better numbers than the default map from speedy loader there are several tools online like: http://www.useasydocs.com/theory/spktable.htm use them at your own risk and always listen for pre-detonation / knocking. It is best to tune the spark tables on a rolling road or dyno.

Settings

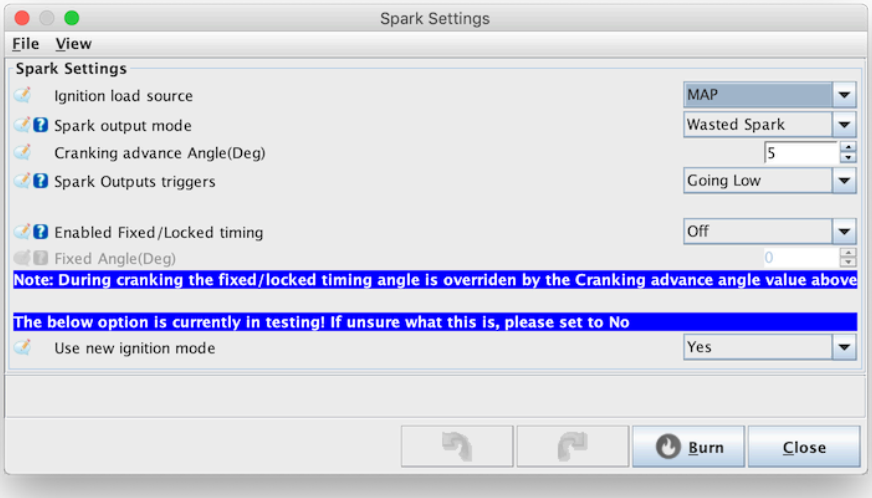


Figure 33: spark_settings.png

- **Spark Output mode** - Determines how the ignition pulses will be outputted and is very specific to your ignition wiring. **Note that no matter which option is selected here, ignition signals ALWAYS fire in numerical order (ie 1->2->3->4) up to the maximum number of outputs**. The firing order of the engine is accounted for in the wiring order.
  - **Wasted Spark** - Number of ignition outputs is equal to half the number of cylinders and each output will fire once every crank revolution. One spark will therefore take place during the compression stroke and the other on the exhaust stroke (aka the 'wasted' spark). This method is common on many 80s and 90s vehicles that came with specific wasted spark coils, but can also be used with individual coils that are wired in pairs. Wasted spark will function with only a crank angle reference (Eg a missing tooth crank wheel with no cam signal)
  - **Single Channel** - This mode sends all ignition pulses to IGN1 output and is used when the engine contains a distributor (Typically with a single coil). The number of output pulses per (crank) revolution is equal to half the number of cylinders.
  - **Wasted COP** - This is a convenience mode that uses the same timing as the 'Wasted Spark' option, however each pulse is sent to 2 ignition outputs rather than one. These are paired IGN1/IGN3 and IGN2/IGN4 (ie When IGN1 is high, IGN3 will also be high). As this is still a wasted spark timing mode, only crank position is required and there will be 1 pulse per pair, per crank revolution. This mode can be useful in cases where there are 4 individual

coils, but running full sequential is either not desired or not possible (Eg when no cam reference is available).
  - **Sequential** - This mode is only functional on engines with 4 or fewer cylinders.
  - **Rotary** - See below for full detail

- **Cranking advance** - The number of absolute degrees (BTDC) that the timing will be set to when cranking. This overrides all other timing advance modifiers during cranking.
- **Spark output triggers** - **THIS IS A CRITICAL SETTING!** Selecting the incorrect option here can cause damage to your igniters or coils. Specifies whether the coil will fire when the ignition output from Speeduino goes HIGH or goes LOW. The VAST majority of ignition setups will require this to be set **GOING LOW** (ie the coil charges/dwells when the signal is high and will **fire** when that signal goes low). Whilst GOING LOW is required for most ignition setups, there are some configurations that perform the dwell timing on the ignition module and fire the coil only when they receive a HIGH signal from the ECU.
- **Fixed Angle** - This is used to lock the ignition timing to a specific angle for testing. Setting this to any value other than 0 will result in that exact angle being used (ie overriding any other settings) at all RPMs/load points, except during cranking (Cranking always uses the above Cranking Advance setting). This setting should be set to 0 for normal operation.

Rotary modes

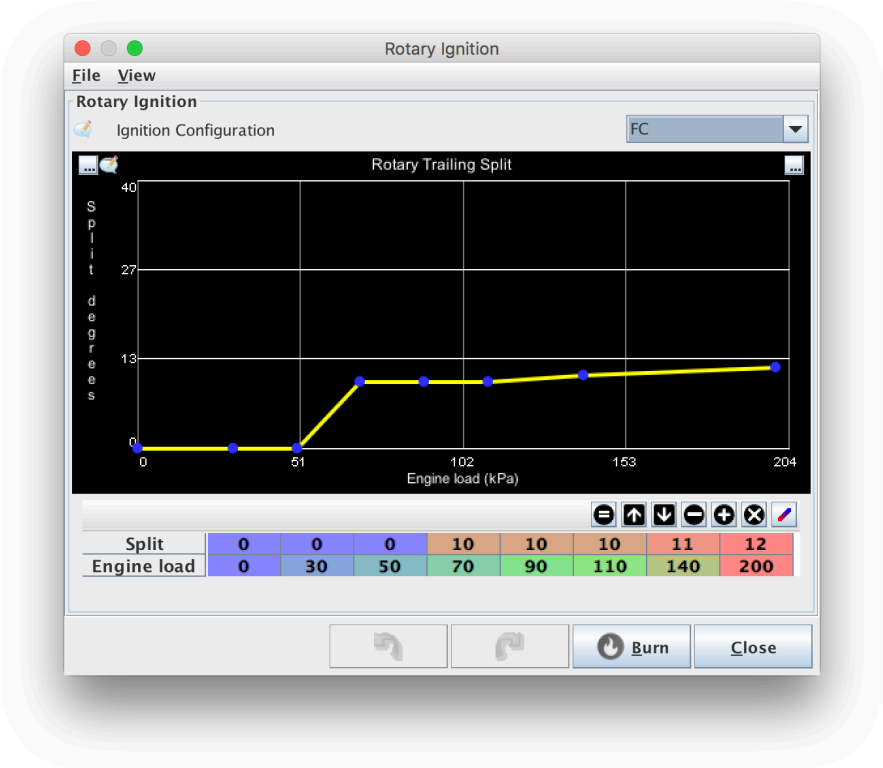


Figure 34: rotary_settings.png

Speeduino supports the ignition configurations found on FC/FD RX7 and RX8 engines and this option becomes avaialble when the Rotary ignition mode is selected above. The leading / trailing split angle can be set as a function of the current engine load.

- **FC** - Outputs are configured for the Leading/Trailing setup that was used on FC RX7s. Wiring is:
  - **IGN1** - Leading (wasted) sparks
  - **IGN2** - Trailing spark
  - **IGN3** - Trailing select
  - **IGN4** - Not used
- **FD** - Uses the same wasted spark signal for both leading sparks as FC, but individual signals for the trailing sparks. Wiring is:
  - **IGN1** - Leading (wasted) sparks
  - **IGN2** - Front rotor trailing
  - **IGN3** - Rear rotor trailing

  - **IGN4** - Not used

- **RX8** - Individual outputs are used for each spark signal. Wiring is:
  - **IGN1** - Front rotor leading
  - **IGN2** - Rear rotor leading
  - **IGN3** - Front rotor trailing
  - **IGN4** - Rear rotor trailing

Dwell Control

Overview

The dwell control dialog alters the coil charging time (dwell) for Speeduino's ignition outputs. Care should be taken with these settings as igniters and coils can be permanently damaged if dwelled for excessive periods of time.

From the April 2017 firmware onwards, dwell will automatically reduce when the configured duration is longer than the available time at the current RPM. This is common in single channel ignition configurations (Eg 1 coil with a distributor) and in particular on higher cylinder count engines.

Settings

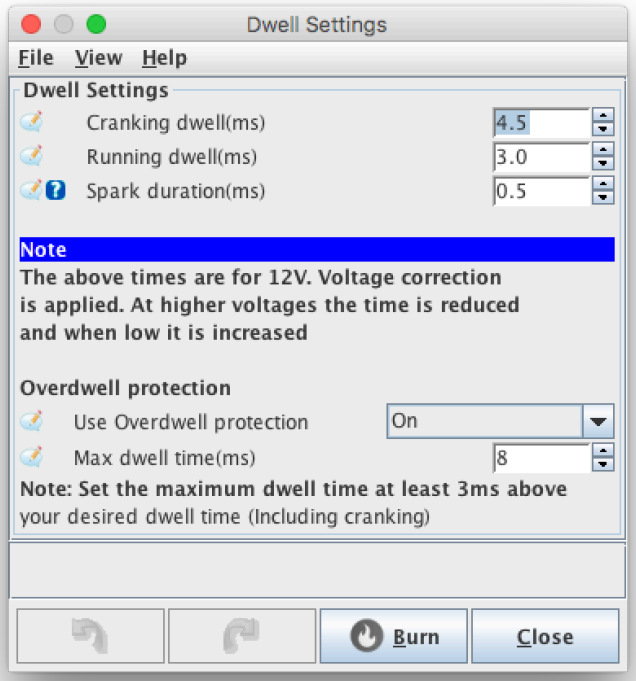


Figure 35: dwell.png

Note: Both the running and cranking dwell times are nominal values, assumed to be at a constant voltage (Usually 12v). Actual dwell time used will depend on the current system voltage with higher voltages having lower dwell times and vice versa. See section below on voltage correction

- Cranking dwell - The nominal dwell time that will be used during cranking. Cranking is defined as being whenever the RPM is above 0, but below the 'Cranking RPM' values in the Cranking dialog
- Running dwell - The nominal dwell that will be used when the engine is running normally.
- Spark duration - The approximate time the coil takes to fully discharge. This time is used in calculating a reduced dwell when in time limited conditions, such as mentioned above on single coil, high cylinder count engines. The limited dwell time is calculated by taking the maximum revolution time at the given RPM, dividing by the number of spark outputs required per revolution and subtracting the spark duration. Outside of those conditions, this setting is not used.
- Over dwell protection - The over dwell protection system runs independently of the standard

ignition schedules and monitors the time that each ignition output has been active. If the active time exceeds this amount, the output will be ended to prevent damage to coils. This value should typically be at least 3ms higher than the nominal dwell times configured above in order to allow overhead for voltage correction.

Voltage correction

As the system voltage rises and falls, the dwell time needs to reduce and increase respectively. This allows for a consistent spark strength without damaging the coil/s during high system voltage conditions. It is recommended that 12v be used as the 'nominal' voltage, meaning that the Dwell % figure at 12v should be 100%.

The correction curve in the base tune file should be suitable for most coils / igniters, but can be altered if required.

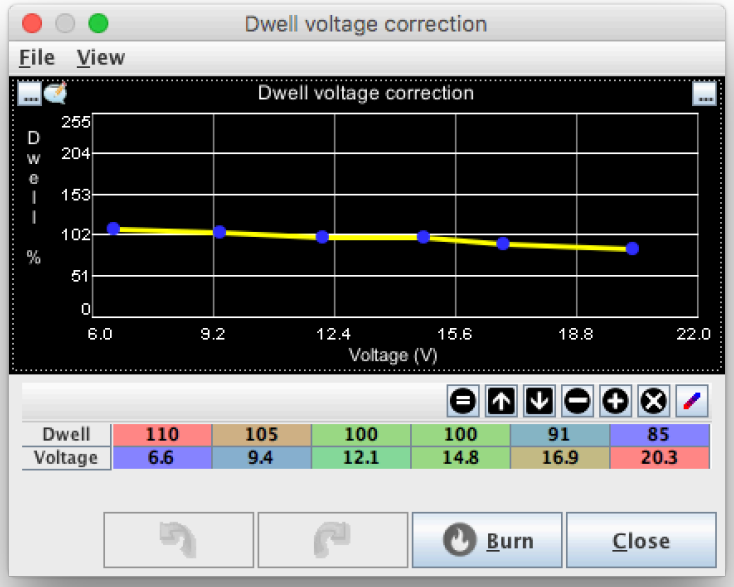


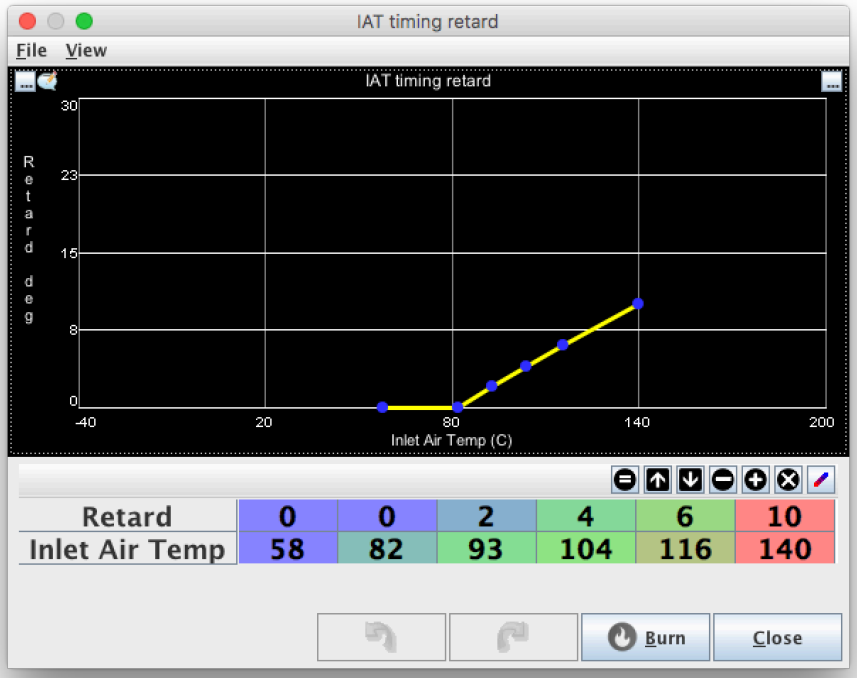
Figure 36: dwell_correction.png

Temperature based timing changes

Changes in Inlet Air Temperature (IAT), in particular significant increases whilst under boost, can require ignition timing to be pulled. The IAT retard settings allow for this timing adjustment

Example

Exact settings will be engine dependant, but pulling of ignition timing beyond 100*C is a common



scenario.

Overview

Cranking conditions during starting typically require multiple adjustments to both fuel and ignition control in order to provide smooth and fast starts. The settings on this dialog dictate when Speeduino will consider the engine to be in a cranking/starting condition and what adjustments should be applied during this time.

Settings

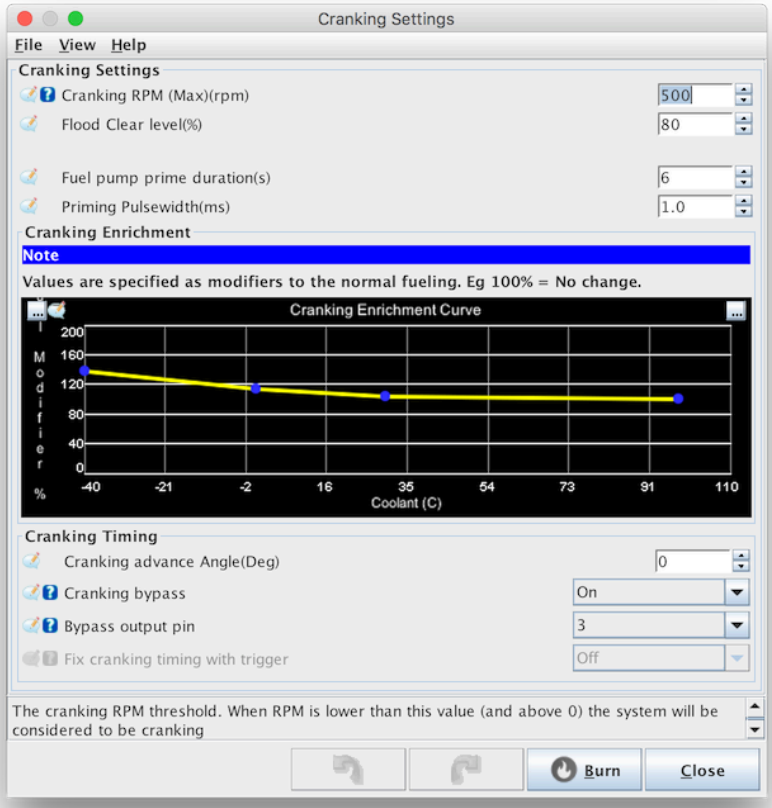


Figure 37: cranking.png

- **Cranking RPM** - This sets the threshold for whether Speeduino will set its status to be cranking or running. Any RPM above 0 and below this value will be considered cranking and all cranking related adjustments will be applied. It's generally best to set this to be around 100rpm higher than your typical cranking speed to account for spikes and to provide a smoother transition to normal idle
- **Flood Clear level** - Flood clear is used to assist in removing excess fuel that has entered the cylinder/s. Whilst flood clear is active, all fuel and ignition events will be stopped and the engine can be cranked for a few seconds without risk of starting or further flooding. To trigger flood clear, the RPM must be **below** the above Cranking RPM setting and the TPS must be **above** the threshold of this setting.
- **Fuel pump prime duration** - When Speeduino is first powered on, the fuel pump output will be engaged for this many seconds in oder to pressurise the fuel system. If the engine is started

in this time, the pump will simply keep running, otherwise it will be turned off after this period of time. Note that fuel pump priming only occurs at system power on time. If you have USB connected, Speeduino remains powered on even without a 12v signal.

- **Priming Pulsewidth** - Upon power up, Speeduino will fire all injectors for this period of time. This pulse is NOT intended as a starting fuel load, but is instead for clearing out air that may have entered the fuel lines. It should be kept short to avoid engine flooding.
- **Cranking enrichment** - Whilst cranking is active (See Cranking RPM above), the fuel load will be increased by this amount. Note that as a standard correction value, this cranking enrichment **is in addition** to any other adjustments that are currently active. This includes the warmup enrichment etc.
- **Cranking Bypass** - This option is specifically for ignition systems that have a hardware cranking ignition option. These systems were used throughout the 80s and early 90s and allowed ignition timing to be fixed and controlled by the ignition system itself when active (Via an input wire). With this option you can specify an output pin that will be set HIGH when the system is cranking. The pin number specified is the ARDUINO pin number.
- **Fix cranking timing with trigger** - Some (usually low resolution) trigger patterns are designed to align one of their pulses with the desired cranking advance. This is typically 5 or 10 degrees BTDC. When enabled, Speeduino will wait for this timed input pulse before firing the relevant ignition output (A dwell safety factor is still applied incase this pulse is not detected). This option is only made available when a trigger pattern that supports this function is selected (See Trigger Setup)

Warmup Enrichment

Overview

The Warm Up Enrichment (WUE) dialog contains settings related to the period after start (ie not cranking) but before the engine has reached normal operating temperature. It allows for modifications to fueling during this time to

Settings

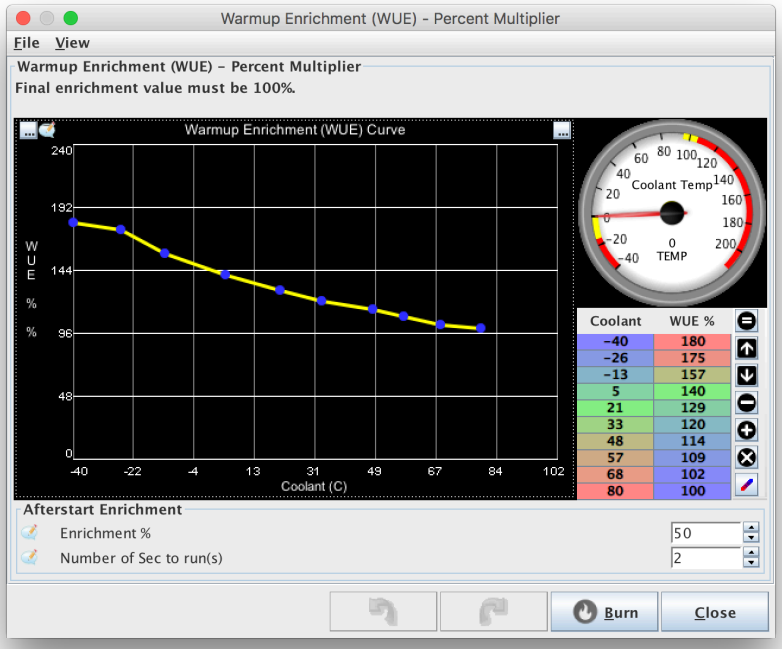


Figure 38: warmup.PNG

Warmup curve

This curve represents the additional fuel amount to be added whilst the engine is coming up to temperature (Based on the coolant sensor). The final value in this curve should represent the normal running temperature of the engine and have a value of 100% (Representing no modification of the fuel from that point onwards).

Afterstart Enrichment

Afterstart Enrichment (ASE) is a seperate fuel modifier that operates over and above the WUE for a fixed period of time after the engine first starts. Typically this is a 3 - 10 second period where a small enrichment can help the engine transition smoothly from cranking to idling.

Idle Control

Overview

The idle control outputs are used to alter the state of an idle control valve to increase the amount of air entering the engine at idle. These come in multiple types (Described below) and each is configured and tuned differently.

Open and closed loop idle control is available for both PWM and Stepper based idle valves.

Compatible Idle Valve Types

There are currently 3 modes of idle control available, using on/off, PWM duty cycle, or a stepper step count, enabled below a set coolant temperature. These modes cover the most common types of idle mechanisms in use.

On/Off (aka Fast Idle)

This is a simple digital on/off "switch" output by Speeduino that triggers at a selected temperature. It is intended to control an on/off fast idle valve as found in many older OEM setups, or an open/closed solenoid-type valve that is chosen for the purpose. In addition to OEM idle valves, examples of valves popular for re-purposing as on/off idle valves are larger vacuum, breather, or purge valves, and even fuel valves. Idle speed adjustment is generally set only once, with an in-line adjustable or fixed restrictor, pinch clamp, or other simple flow-control method.

Note: On/Off valves can be used in many ways to increase or decrease air flow for various idle purposes in-addition to warm-up. Examples are use as dashpot valves to reduce deceleration stalling, idle speed recovery for maintaining engine speed with accessory loads such as air conditioning, or air addition for specific purposes such as turbo anti-lag air control. See Generic Outputs for control information.

PWM

While similar in construction to many solenoid on/off valves; PWM idle valves are designed to vary the opening, and therefore flow through the valve, by PWM valve positioning.

These valves are opened and closed by varying the duty cycle of signal sent to them.

> **Note:** As a fail-safe, some PWM idle valves default to a partially-open state when they are disconnected or are receiving 0% duty cycle. They will close then re-open with increasing PWM DC%,

> so be sure to research or test your valve type for proper operation. {.is-info}

PWM Settings Settings in TunerStudio include selecting PWM idle control, temperature and DC settings for warmup, and PWM DC during cranking under the following selections:

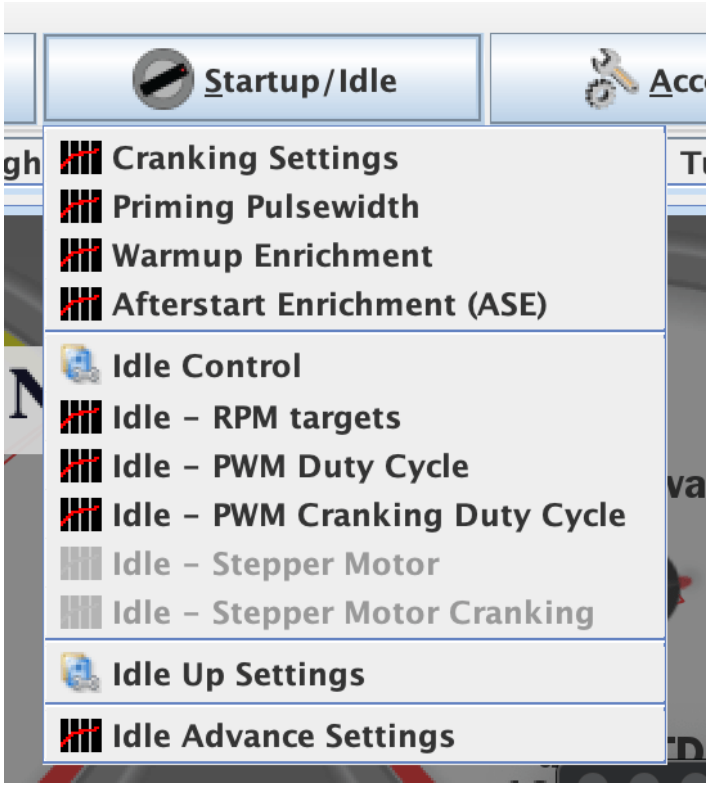


Figure 39: Example PWM idle settings

The '*Idle - PWM Duty Cycle*' and '*Idle - PWM Cranking Duty Cycle*' options will only be available when '*PWM Open Loop*' is selected in the Idle Control options

Under Idle control type, PWM is selected:

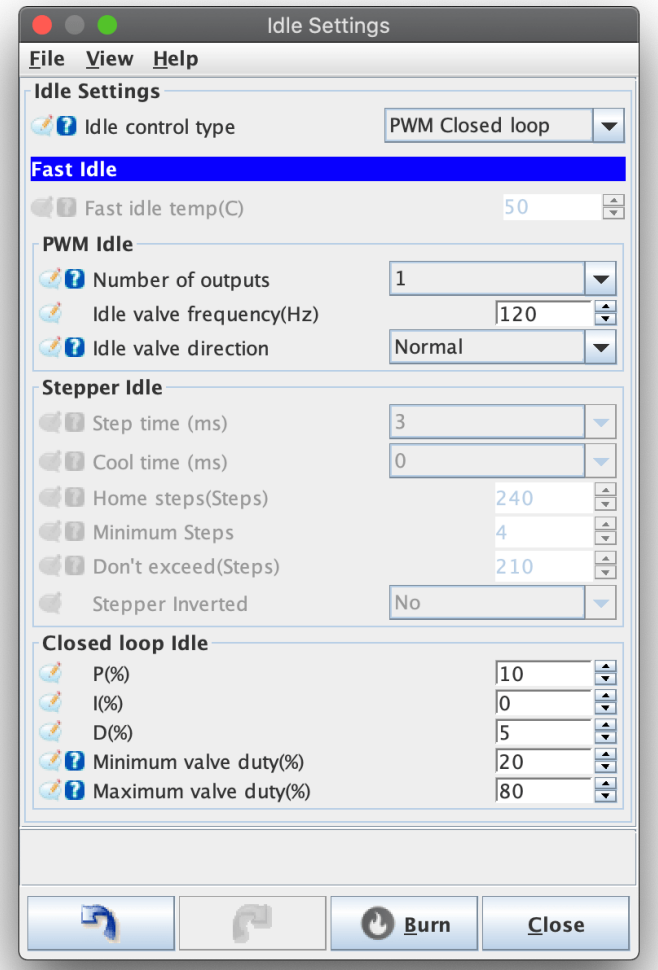


Figure 40: Example PWM idle settings

The temperature-versus-DC is selected under the Idle - PWM Duty Cycle selection. Note the relationship between temperature and PWM DC can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:

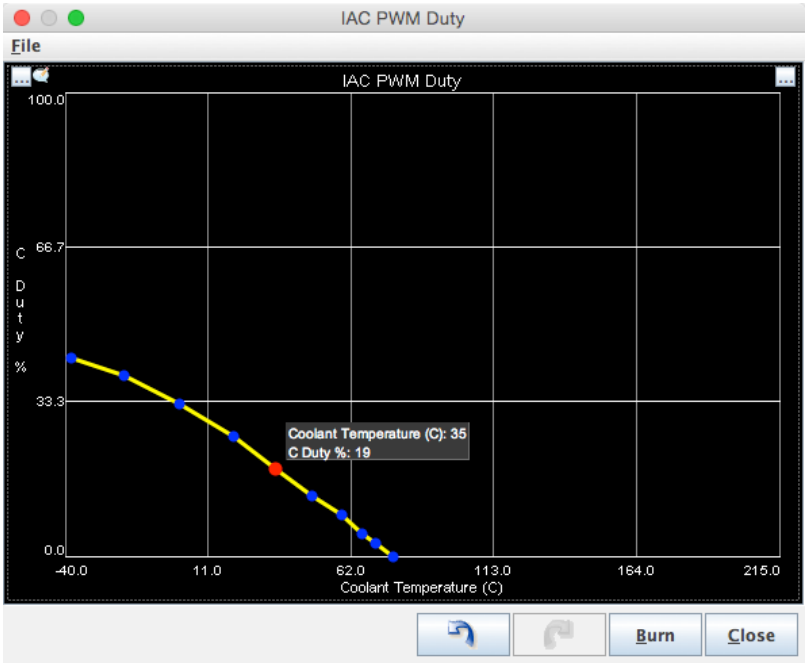


Figure 41: Example PWM idle curve

Some engines prefer additional airflow during cranking for a reliable start. This air can be automatically added only during cranking by using the Idle - PWM Cranking Duty Cycle settings. Once the engine starts and rpm rise above the set maximum cranking rpm, the idle control switches to the previous warmup settings. Note the relationship between coolant temperature during cranking and PWM DC can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:

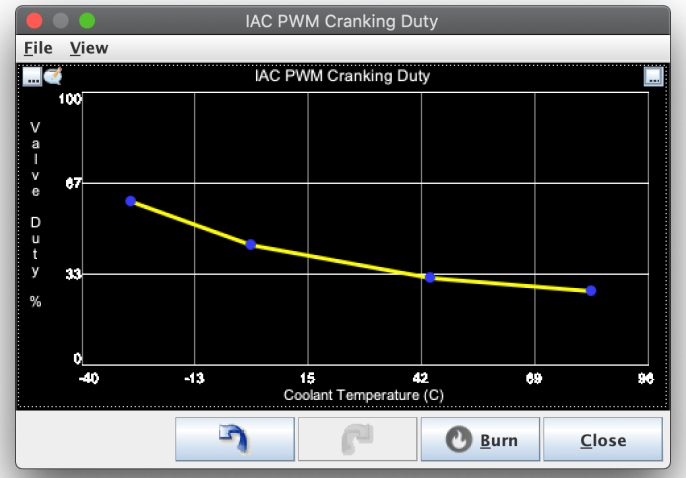


Figure 42: Example PWM cranking curve

> ***NOTE:*** *Every engine, valve type and tune is different. Suitable settings must be determined by the tuner. Do not infer any tuning settings from the images in this wiki. They are only examples.* {.is-warning}

2 wire vs 3 wire valves Both 2 and 3 wire PWM idle controllers are supported. In general, the 3 wire models will provide a smoother response than the 2 wire ones, but the difference is not always significant. For 3 wire valves, 2 of the Aux outputs will be required.

Stepper Motors

Stepper motor idle controls are very common on GM and other OEM setups. These motors typically have 4 wires (bi-polar). They must be driven through power transistors or a driver module, such as the DRV8825 stepper motor driver optional to the v0.4 board. These driver modules can be purchased inexpensively from a variety of vendors on sites such as eBay, Amazon, etc.

Most stepper idle valves function by turning a threaded rod in and out of the valve body in a series of partial-turn steps, increasing or decreasing airflow around the plunger (on end of valve below), and into the engine. The idle airflow bypasses the primary throttle body:

Example of a generic DRV8825 driver module on a v0.4 board:

Note the board is mounted at a standoff for air circulation and cooling:

The DRV8825 motor outputs are labeled as A2-A1-B1-B2, and the wiring connection examples are to this labeling. Check your schematics for the output connections that route to these DRV8825 outputs:

Examples of wiring to the DRV8825 driver:

The GM "screw-in" style used 1982 to 2003 on many models:

Stepper Driver Current Adjustment The DRV8825 stepper driver module includes a potentiometer (adjustable resistor) indicated by the yellow arrow in the image below. The potentiometer is used for setting the driver's maximum current output limit. Because Speeduino uses full-step operation, the current limit is not critical to protect the module, but should be adjusted to the module's maximum value for best operation of most automotive stepper IACs.

You will need a multi-meter or volt-meter to make the adjustment as outlined here. In order to set the potentiometer to maximum current before first use, ensure power to the module is OFF, then gently turn the potentiometer dial clockwise to the internal limit. **Do not force the adjustment beyond the internal stop.** Power-up Speeduino with 12V, and use the meter to test the voltage between the center of the potentiometer and any 12V ground point. Note the voltage reading. Power-down and repeat the test, this time turning the potentiometer counter/anti-clockwise gently to the internal limit. The test direction that resulted in higher voltage is the correct setting for the module.

Note: Original Pololu modules are typically adjusted clockwise for maximum voltage. However, clone modules may be either clockwise or counter-clockwise, which makes this testing necessary.

The module's rated *continuous* current is up to 1.5A. While the module can supply a peak of 2.2A of current; in full-step mode and with the potentiometer adjusted to this position, the driver is limited to approximately 70% of full current, or approximately 1.5A.

Stepper Settings Settings in TunerStudio include selecting stepper idle control, temperature and step settings for warmup, and open steps during cranking under the following selections:

Under Idle control type, stepper is selected. The basic stepper operational settings are also located in this window:

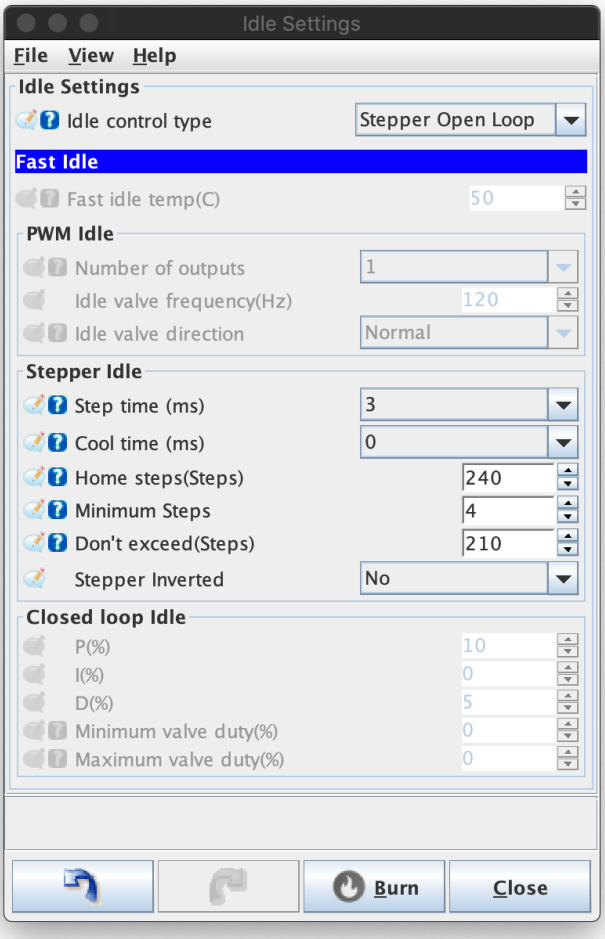


Figure 43: Stepper idle settings

- **Step time:** This is how long (in ms) that the motor requires to complete each step. If this is set too low, the idle motor will not have completed the step before the ECU tries to make the next one, which leads to the motor 'twitching' and not functioning correctly. If this is set longer than needed the system will take longer to make each adjustment and the overall idle response will be slower. Typical values are usually 2ms - 4ms. The common GM stepper motor requires 3ms.
- **Cool time:** Some motors require a slight pause in between steps in order to function correctly. This is know as the 'cooling' time. Typically this value will be less than 4ms at the most, with many motors operating normally with no cooling period (0ms)
- **Home Steps:** Stepper motors must be 'homed' before they can be used so the that ECU knows their current position. You should set this to the maximum number of steps that the motor can move.
- **Minimum steps:** In order to allow a smooth idle that isn't continually fluctuating, the ECU will

only move the motor if at least this many steps are required. Typical values are in the 2-6 range, however if you have a noisy coolant signal line, this value may need to be increased.

- **Don't exceed:** In order to prevent the stepper motor attempting to move beyond it's maximum range, this is a limit placed on the total number of steps that will be made. The value in this field must always be lower than the number of homing steps

The temperature-versus-steps is selected under the Idle - Stepper Motor selection. Note the relationship between temperature and motor steps can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:

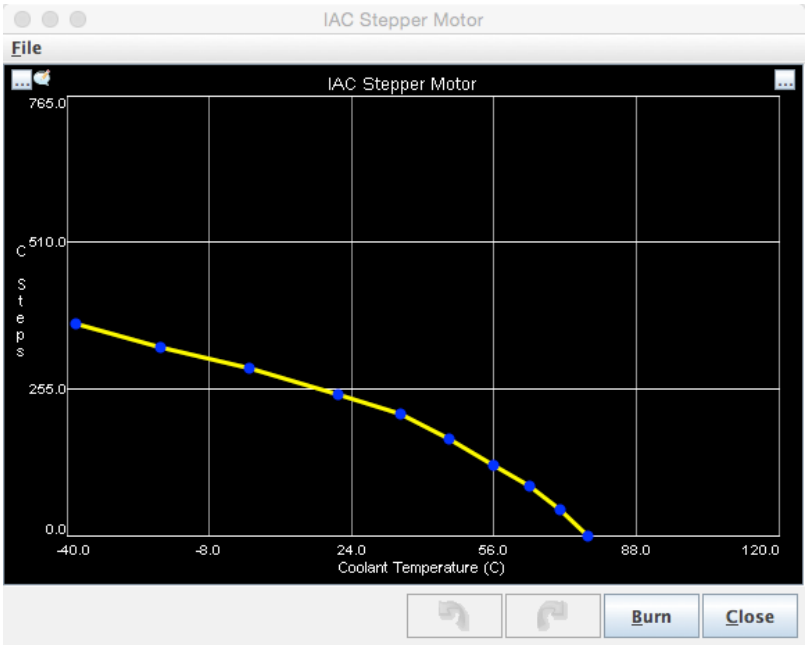


Figure 44: Example stepper idle curve

Some engines prefer additional airflow during cranking for a reliable start. This air can be automatically added only during cranking by using the Idle - Stepper Motor Cranking settings. Once the engine starts and rpm rise above the set maximum idle rpm, the idle control switches to the previous warmup settings. Note the relationship between coolant temperature during cranking and motor steps can be altered by simply moving the blue dots in the curve, or by selecting the table for manual entry as shown here:

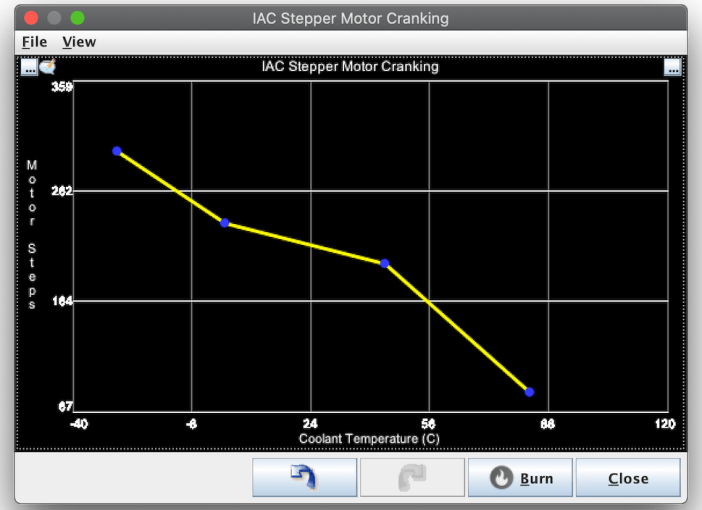


Figure 45: Example cranking stepper curve

> ***NOTE:*** *Every engine, valve type and tune is different. Suitable settings must be determined by the tuner. Do not infer any tuning settings from the images in this wiki. They are only random examples.* {.is-info}

NOTE: *Refer to the Pololu video for instructions to set the DRV8825 driver current level to maximum for most automotive full-step stepper motors.*

Examples

Motor	Step time	Home steps
GM 4-wire	3	250
DSM 4-wire	4	270-320

NOTE: *While normal DSM stepper function is seen at room temperatures at 3ms, step skipping occurs just under that speed. Very cold temperatures may cause skipping, thus the recommendation of 4ms. Test for the most suitable speeds for your setup.*

Stand-Alone (Non-Electronic)

While not an idle control mode, Speeduino is compatible with stand-alone idle valves that are self-controlling. Examples of this are thermal wax or bi-metal spring idle or auxiliary air valves like the one below. Internally expanding and contracting material opens and closes air valves, providing increased air flow and engine rpm when cold for warmup. Speeduino functions to enrich the cold engine and adjust for the additional air, in the same way it would if you opened the throttle slightly.

Other examples of stand-alone valves are simple On/Off valves as shown in the next section, controlled by inexpensive thermal switches like these:

Closed Loop Control

Closed loop idle control operates by setting RPM targets rather than configuring the duty cycle or steps directly. A PID algorithm is used and can be tuned to match the valve/motor that you are using.

Closed loop settings

Closed loop targets

Idle advance control

Idle speed can be controlled without the use of an idle valve (IACV) by adjustiming timing. This feature references the same idle RPM target curve that is used by the closed loop idle control and will then adjust the advance based on the error between current and target RPM.

Settings

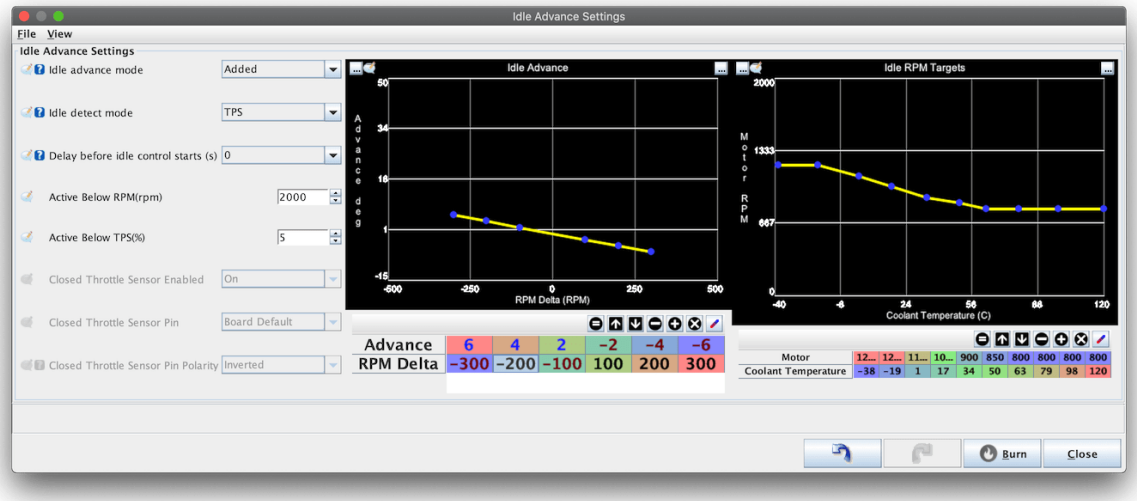


Figure 46: idle_advance.png

- **Idle advance mode**
  - **Added** - This is the most common mode and will alter the regular advance amount by adding (or subtracting) a certain number of degrees based on the amount of RPM delta (Between target and actual RPMs)
  - **Switched** - The ignition advance will switch to the values in the idle advance curve rather than adjusting the normal advance values
- **Idle detect mode** - This setting specifies how the ECU determines whether it is at idle or not. Most commonly this is based on a variable TPS and a specific TPS%, but if a closed throttle switch (CTPS) is available, this may be used instead
- **Delay before idle control** - This allows the idle RPM to settle during decelleration before the ignition advance is changed.
- **Active below** - Maximum RPM that the idle advance control will be active under
- **Active Below** - If the idle detect mode is set to TPS, this is the throttle position that the control will be active below
- The following 3 settings are only used if idle detection uses a CTPS input
  - **CTPS enabled** - Whether to use a CTPS input
  - **CTPS Pin** - The Arduino pin that the CTPS is connected to
  - **CTPS Polatiry** - Whether idle is indicated by the input being pulled to ground (Normal) or indicated by the input being pulled to 5v (Inverted). In Normal mode, the internal pullup

will be enabled.

Idle Advance curve

This curve specifies the amount of timing adjustment (Added mode) or the absoolute advance amount (Switched mode) that will be used based on the delta (error) from target RPM.

Generally timing will be added (positive values) in order to try and increase RPM and timing will be removed (Negative values) to redue RPM.

Idle RPM target curve

This curve specifies what the desired idle RPM is based on the current coolant temperature. This table is shared with the idle air control if that is being used in conjuction with idle advance control.

Thermo fan

Control of a cooling (thermo) fan is available through the Thermo fan dialog.

> Note that only On/Off fan control is currently possible (Not fan speed control using PWM). {.is-info}

Settings

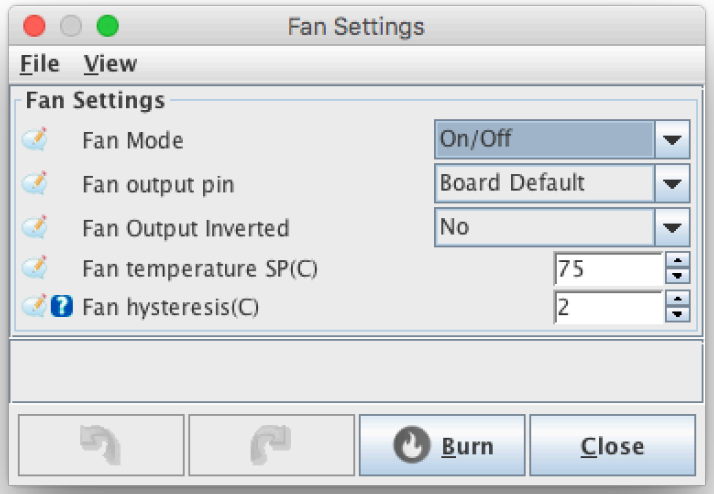


Figure 47: fan.png

- **Fan mode** - Currently only `On`/`Off` mode is supported
- **Output pin** - The arduino pin that the fan control will use. In most cases this should be left as `Board Default`
- **Output inverted** - Most setups will use `No` for this setting, but if you have a fan circuit that flips the output, the On/Off polarity can be reversed with this setting
- **Fan temperature** - The temperature above which the fan will be turned on.
- **Fan hysteresis** - The number of degrees below the fan set point that fan will be turned off. This is used to avoid oscillation around the set point resulting in the fan turning on and off rapidly.

Launch Control & Flat Shift

Speeduino features a 2-step launch control combined with a flat shift feature. These are each dependant on a clutch switch (Usually a ground switching type) being wired in.

Setup

Both the 2-step and flatshift modes have hard and soft cut states. When under soft cut, the ignition timing will be altered to reduce the RPM acceleration, though this is generally not sufficient to stop or limit RPM rising. Under hard cut, the ignition signal is stopped completely until the RPMs drop

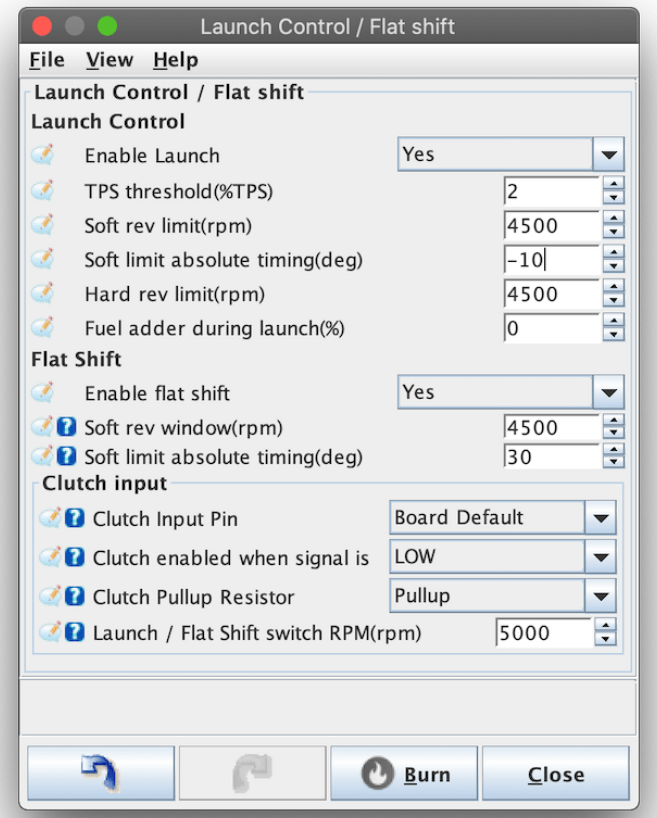


Figure 48: Launch and flat shift settings

Launch

- **TPS Threshold** - A minimum value for the launch engagement. The limiter will only be engaged above this RPM. Typical values are 1%-3% TPS, depending on how much noise is on your signal
- **Soft rev limit** - The RPM at which the timing will be adjusted to slow RPM increase
- **Soft limit absolute timing** - The **absolute** timing that will be used once the soft RPM limit is reached. This overrides all other timing adjustments at this time

- **Hard rev limit** - The RPM at which the ignition signal will be cut entirely.
- **Fuel adder during launch** - A percentage modifier to the current pulse width to add extra fuel when launch (soft or hard) is active. This can aid in building boost on turbo setups at launch time

Flat shift

- **Soft rev window** - This is an RPM window below the `Launch / Flat shift switch RPM` point during which an alternative timing will be applied. Typical values are 100 to 1000rpm.
- **Soft limit absolute timing** - The abosloute timing that will be used when in the flat shift soft RPM window

Clutch settings

Both launch and flat shift require a clutch input in order to activate. This is generally a ground active type switch attached behind the clutch pedal.

- **Clutch input pin** - The Arduino pin that the switch is wired to. Most setups should leave this as the Board Default
- **Clutch enabled when switch is** - The polarity of the clutch input. Typically this should be set to `LOW` for a switch that connects to ground when activated
- **Clutch pullup resistor** - Whether the internal pullup will be enabled on this input. Typicall this should be set to `Pullup` if you have selected `LOW` for the above setting
- **Launch / Flat shift switch RPM** - The ECU will use the RPM point the clutch is engaged at to determine whether it is in launch mode or flatshift. If the clutch is pressed above this RPM value, it will be assumed to be a flat shift, below it will be considered a launch

> The engagement point of the clutch switch can make a significant difference in the applcation of launch control. The switch should trigger as close to the clutches take up point as possible for the fastest response. {.is-success}

Fuel pump

Fuel pump control is a simple but important function performed by the ECU. Currently Speeduino does not perform variable (PWM) pump control. Can only be connected to a relay. DO NOT CONNECT DIRECTLY TO FUEL PUMP.

Settings

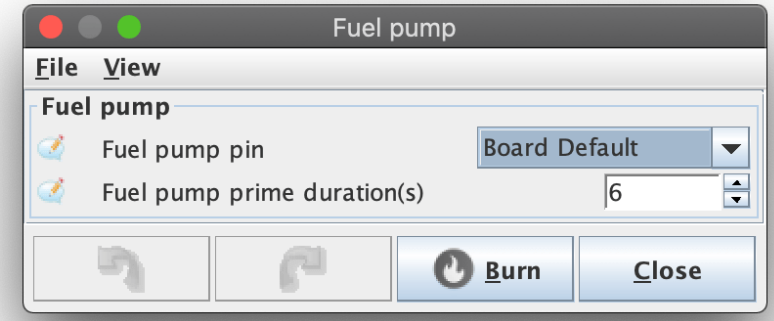


Figure 49: fuel_pump.png

- **Fuel pump pin** - The Arduino pin that the fuel pump output is on. In most cases this should be left to `Board Default` unless you have a specific reason to change this.
- **Prime duration** - How long (In seconds) the fuel pump should run when the system is first powered up. Note that this is triggered **when the ECU is powered on**, which will not always be the same as when the ignition is turned out. If you have a USB cable connected then the ECU is already powered up.

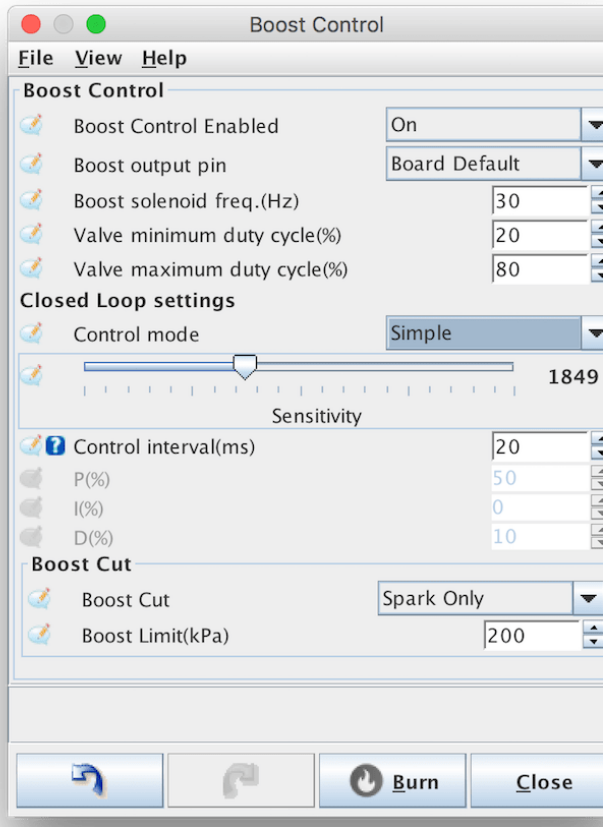
Boost Control

Speeduino has an on board closed loop boost controller than can be used to regulate standard single turbo setups.

Most 3 or 4 port boost solenoids can be used, with frequencies between 15Hz and 500Hz supported. Any of the on board high current outputs can be directly connected to the solenoid and is controlled via a boost target table and PID tuning. Over boost limiting is also available.

Settings

Speeduino's boost control uses a PID algorithm with 2 modes of operation, Simple and Full. Each has



their own advantages and disadvantages, as outlined below
In Simple mode, the PID values themselves are controlled by the ECU itself and a sensitivity slider is used to adjust how aggressive the output duty cycle will be set. The simple mode can be easy and fast to setup, however has the downside that to avoid overboost, the sensitivity may need to be set low, which can increase lag.

Target table

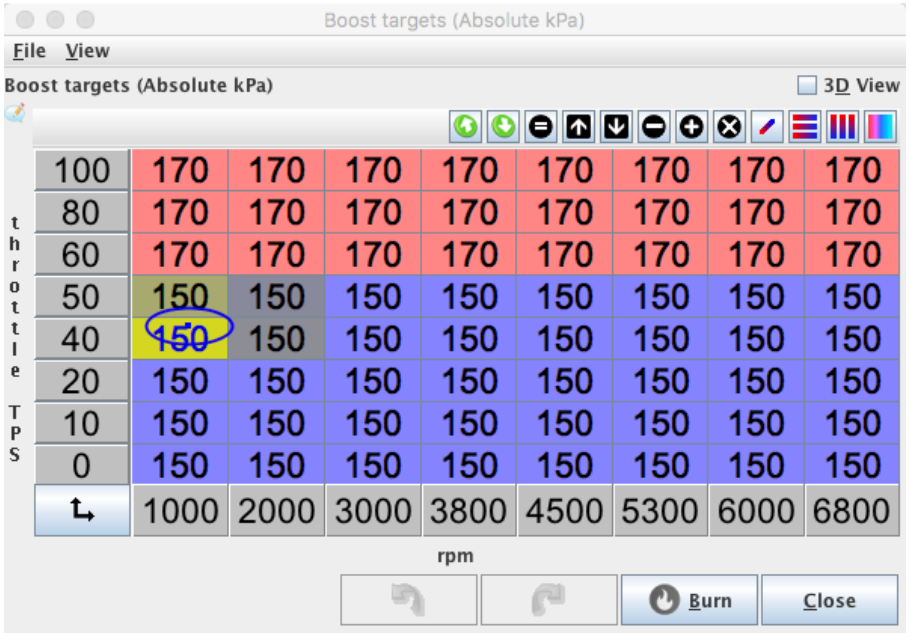


Figure 50: Example boost map

The boost map function varies depending on whether open or closed loop boost control has been selected.

- In closed loop mode, this map serves as a target table. The values in the map are the desired boost pressures (in kPa). In closed loop mode, these target values can optionally be modified by a flex fuel value if available.
- In open loop mode, the map values are the duty cycle percents that will be used

Nitrous Control

Speeduino contains a 2 step nitrous control system for controlling valves and making fueling adjustments for dry setups. The 2 stages operate independantly and can overlap (ie both run at the same time) if needed.

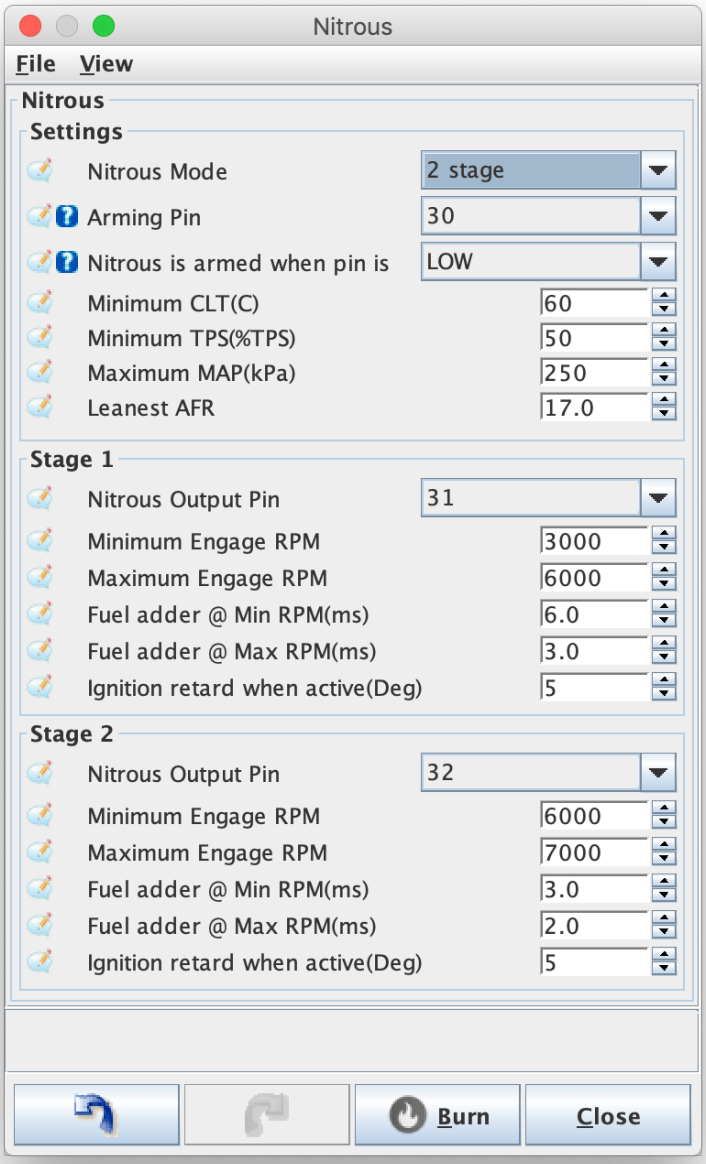


Figure 51: nitrous_settings.png

Activation Settings

- **Nitrous Mode:** Whether 1 or 2 stages will be used

- **Arming Pin:** The Arduino Pin to be used for arming the nitrous control.
- **Arming pin polarity:** What pin state is considerd to be armed. Generally this will be LOW for a ground switching input
- **Minimum CLT:** The minimum coolant temperature that the stages will activate at
- **Minimum TPS:** The minimum TPS that the stage will activate at
- **Maximum MAP:** A protection to ensure that the nitrous will not activate above a certain level of boost
- **Leanest AFR:** Nitrous will only activate if the AFR is (And remains) below this value

Stage Settings

The settings for each stage are identical and allow for the 2 stages to run individually or joinly overlapping during a given RPM window.

- **Nitrous output Pin:** The (Arduino) pin that will be driven high when the stage is active.
- **Minimum Engage RPM:** The RPM at which the stage will begin
- **Maximum Engage RPM:** The RPM at which the stage ends
- **Fuel adder @ Min/Max RPM:** The amount of fuel to be added at the minimum and maximum RPM points.
  - The fuel adder amount will be scaled between these 2 values as the RPMs rise
  - A calculator for these fuel adder values can be found at: https://bit.ly/3a0e9WU
- **Ignition retard when active:** An ignition modifier to be used to retard timing when the stage is active
  - Note that the retard values are cumulative. If both stages are active then the total retard amount will be the sum of both stages.

Sensor Calibration

Before your Speeduino can correctly interpret the signals from the sensors, it must know which sensors you are using. Inputting this information into TunerStudio (TS) writes the correct calibration to your Speeduino. It is necessary to perform this step before you can effectively check your Speeduino build. Note that this is not tuning your system, but only telling it how to understand the signals from the sensors.

This should be completed after completing the Settings for your engine. Your computer must be connected to your Speeduino through TS to perform the calibrations.

MAP Sensor

Open the **Tools** menu: tools_menu

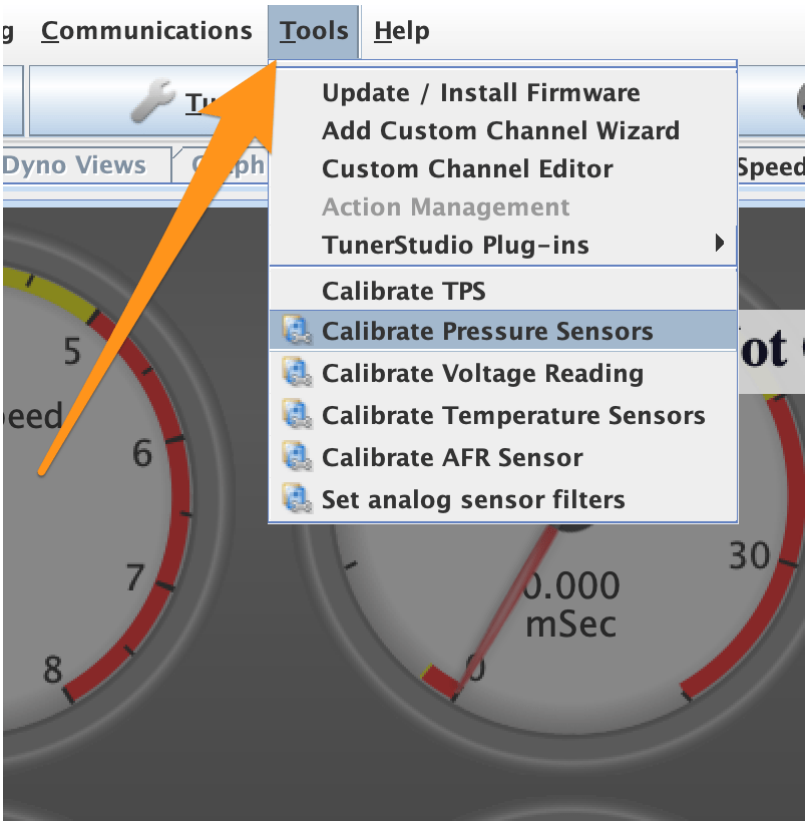


Figure 52: tools_menu.png

Select **Calibrate Pressure Sensors**, the window below will open:

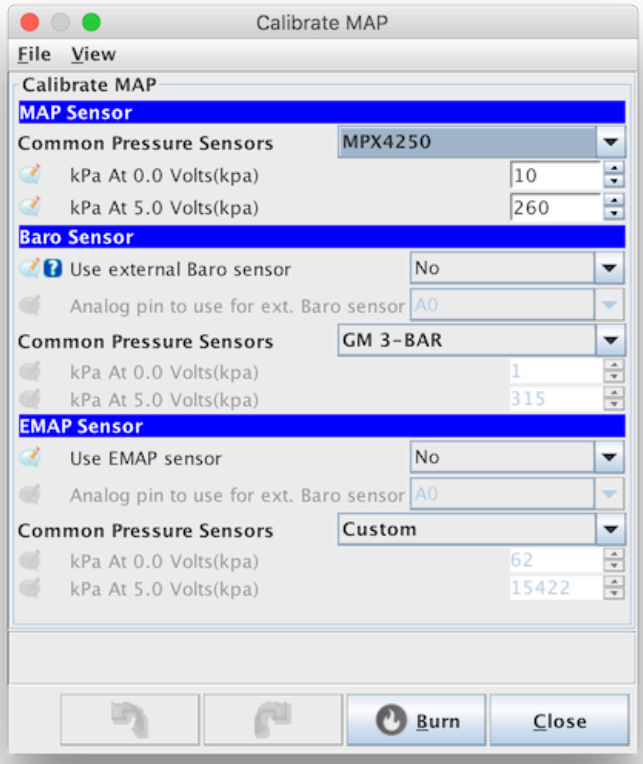


Figure 53: MAP_calibration.png

Select your MAP Sensor from the drop down list. If you used the MAP sensor in the bill of materials, this will be the MPX4250A. If you are using another MAP or one from the engine manufacturer, select it from the list. Click **Burn** to send the information to your Speeduino.

If used, the external Baro and EMAP (exhaust pressure) sensors can be calibrated in the same manner.

Coolant and Intake Temperature Sensors

Open the **Tools** menu and select **Calibrate Thermistor Tables**:

The sensor selected will be the **Coolant Temperature Sensor**. Select your sensor from the **Common Sensor Values** drop-down list. This will place the correct values into the temperature and resistance charts and the Bias resistor value. If your sensor is not listed, see Entering Custom Values below.

Note that the standard Speeduino build is to have a 2490 ohm bias resistor, which is standard for sensors used by most manufacturers. If your sensor requires another value, you may need to change resistor R3 to the correct value for your sensor. You can try overriding the Bias Resistor Value with 2490 ohms, but check to be sure your sensor reads correctly in TS.

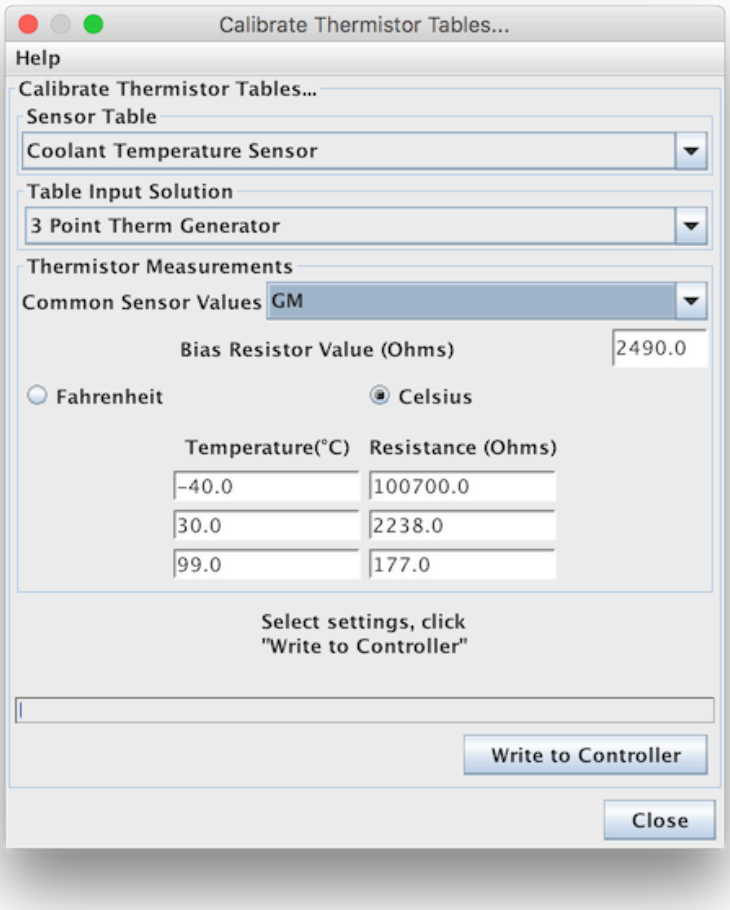


Figure 54: CLT_calibration.png

The same calibration can then be performed for the Inlet Air Temperature (IAT) sensor by changing the **Sensor Table** to **Air Temperature Sensor**:

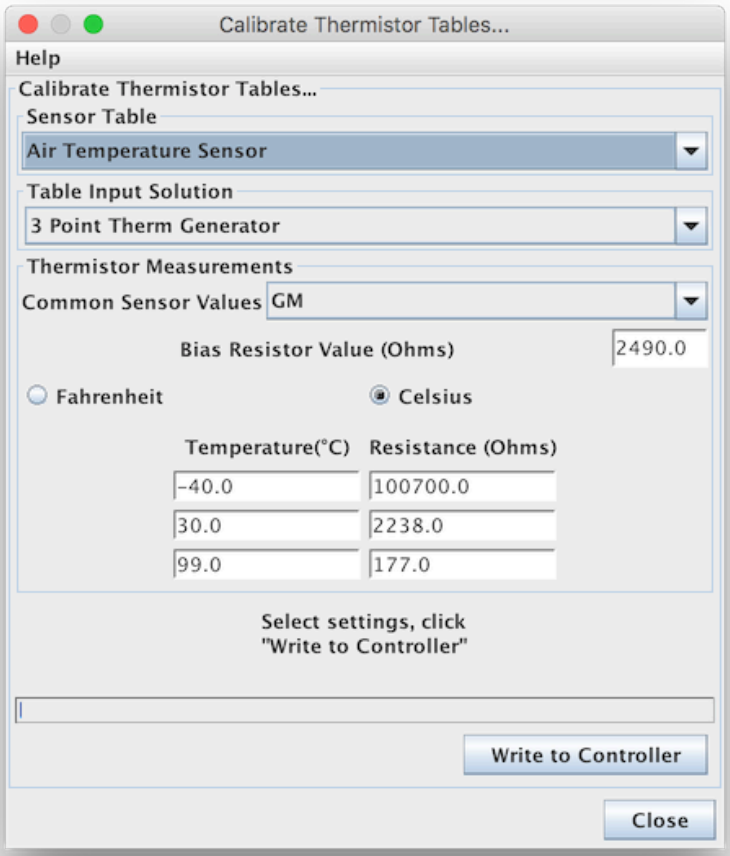


Figure 55: IAT_calibration.png

Select your sensor from the **Common Sensor Values** drop-down list. This will place the correct values into the temperature and resistance charts and the Bias resistor value. Click **Write to Controller** to send this information to your Speeduino. If your sensor is not listed, see Entering Custom Values below.

Note that the standard Speeduino build is to have a 2490 ohm bias resistor, which is standard for sensors used by most manufacturers. If your sensor requires another value, you may need to change resistor R3 to the correct value for your sensor. You can try overriding the Bias Resistor Value with 2490 ohms, but check to be sure your sensor reads correctly in TS.

Entering Custom Values

Some sensors are not listed in the tables for the common sensors. If yours is not listed, you will need to enter the values into the fields your self. You will need two bits of information: 1. The value of your bias resistor (2490 if you used the standard values in the Bill of Materials, or you have a pre-made Speeduino), and 2. The resistance of your sensor at three different temperatures.

The sensor resistance can be generated by measuring the resistance of the sensor in ambient air, putting it in a freezer and then in boiling water. You will need a good multimeter and an accurate thermometer that measures -10C to 100C (14°F to 212°F). It is best to use jumper wires to allow the resistance of the sensor to be read without holding it in your hand (some sensors react quickly to temperature changes). Some sensors react slowly to temperature changes, so allow the sensor at least 10 minutes to reach a stable temperature, and then record the temperature and resistance observed.

In the **Calibrate Thermistor Tables** screen, first ensure the correct temperature unit is selected (**F** or **C**). Then record the bias resistor value and the temperature / resistance values in the fields. Click **Write to Controller** to send this information to your Speeduino.

Note that his procedure can also be used to enter the values of resistance on simulators for testing and troubleshooting. Two points should be remembered if you use simulator values – first, never enter zero for resistance. Although your simulator may go to zero, enter some small value above zero, say 10 ohms. Entering zero leads to false values in the firmware. Second – remember to enter the correct sensor values before installing your Speeduino!

Oxygen Sensor

Open the **Tools** menu again and select **Calibrate AFR Table**:

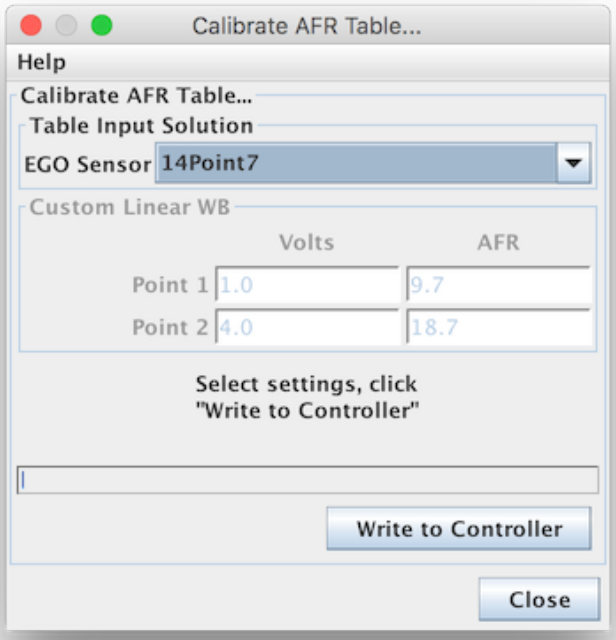


Figure 56: O2 calibration

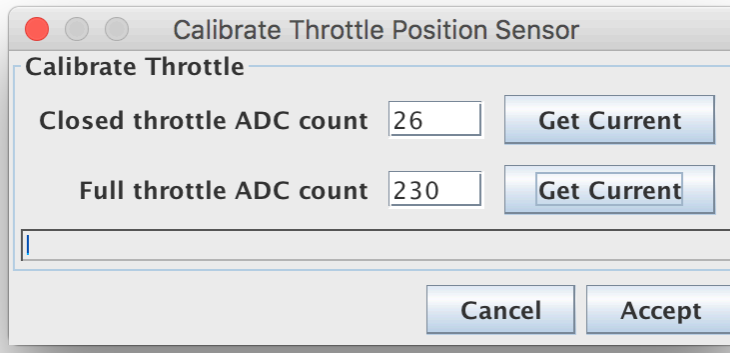
Select your **Oxygen Sensor sensor** from the **Common Sensor Values** drop-down list. If you are using a custom Oxygen Sensor controller, select **Custom Linear WB** and then you can enter the values for **Volts** and **AFR** at two points (should be published in the controller manual).

Click **Write to Controller** to send this information to your Speeduino.

This will set up your Speeduino so that you can also run simulations to check your build before installation.

Throttle Position Sensor

Before Speeduino can work correctly with your engine, you will also need to Calibrate the Throttle Position Sensor. This must be done using the throttle body and TPS used on the engine. It is best to do this while the throttle body is installed on the engine.



Open the **Tools** menu and select **Calibrate TPS**:

With the throttle closed, click the **Get Current** button beside the Closed Throttle ADC count field. Then move the throttle to full open and hold it there. Then click the **Get Current** button beside the Full Throttle ADC count field.

Click **Accept** to save the information to Speeduino.

Auxillary IO Configuration

Speeduino also supports the reading of up to 16 additional input channels. These inputs can be either Analog or Digital Pins on the Mega2560(or other mcu in use) or from a remote data aquistion device (such as The DxControl GPIO for instance) via the secondary serial port or the Canbus interface(if available).

A data channel configured here will have the raw data avilable in TunerStudio as a Gauge and will also be loggable too.

How to Use

The configuration is mainly split into two categories,

- **Local MCU Pin** - How to Configure to use a Local MCU pin
- **External Data Source** - How to Configure to use a External Data Source

How to Configure to use a Local MCU pin

The configuration page is accessed from the Accessories drop down within TunerStudio ,select the “local Auxillary input channel configuration” option

This window will be opened.

In the above image the first two channels have been configured as an local analog and a local digital input respectively.

- **Input Alias** - This is a user defined Alias name (up to 20 characters) for the input channel
- **Aux Input Channel Enable** - This Enables/Disables the input channel
- **PIN** - Is the pin selected(only for local options)

Input Alias The input alias can be any ascii character name the user wishes up to 20 characters long. This can also be left as the default .

Aux input Channel Enable Options here are

- **OFF** - Channel is disabled
- **EXT/CAN** - The channel is assigned to an external data source

(this option is only visible if CAN_COMMANDS is enabled in project properties.See here for further information

- **Local_analog** - Select a local analog mcu pin
- **Local_Digital** - Select a local digital mcu Pin

PIN This setting is only available for local mcu pin selections. It is the actual mcu pin name.

How to Configure to use a External Data Source

To use the Auxillary input channels for external data the Secondary IO must be enabled. See here for further information on how to do this.

The configuration page is accessed from the Accessories drop down within TunerStudio ,select the “External Auxillary input channel configuration” option

This window will be opened.

For External data inputs to be active the “Enable External Data Input” option must be enabled.

In the above image the first three channels have been configured as an local analog and a local digital input and a External input respectively.

- **Input Alias** - This is a user defined Alias name (up to 20 characters) for the input channel
- **External Aux Input Channel Enable** - This Enables/Disables the input channel
- **Source CAN Address** - Is Real Can address of the source device
- **Source Data Start Byte** - Is the first byte (of the 8bytes in a canbus command) where the data can be found
- **Input Parameter Number of Bytes** - Is the number of bytes the data is stored in(lsb first)

Input Alias The input alias can be any ascii character name the user wishes up to 20 characters long. This can also be left as the default .

External Aux input Channel Enable Options here are

- **OFF** - Channel is disabled
- **EXT/CAN** - The channel is assigned to an external data source
- **Local_analog** - Select a local analog mcu pin
- **Local_Digital** - Select a local digital mcu Pin

Source CAN Address This is the Hex address of the remote Device

Source Data Start Byte A can data command has up to 8 bytes. This value sets the first data byte the data value begins at.

Input Parameter Number of Bytes The data byte can be made from a single byte or two (word or 16bit value)

Supported trigger patterns

Missing Tooth Pattern

Overview

A missing tooth crank trigger is used as standard equipment by a number of OEMs, most notably Ford, but is also very popular as an aftermarket fitment.

It is comprised of crank wheel with a given number of evenly spaced teeth, and one or more 'missing' teeth. Common values are typically 60-2, 36-1, 24-1, 12-1 and 4-1 where the first number represents the total number of teeth the wheel would have if there were none missing. The second number after a dash "-" indicates the number of teeth missing.

> **Note**: If there is a third number (e.g., 36-1-1), the missing teeth are not sequential, and this decoder does not apply. Do not confuse counts with slashes "/", as numbers following slashes represent cam teeth—not missing teeth. Wheels with "+" indicate added teeth rather than missing, and again this decoder does not apply. {.is-warning}

Applications

Missing tooth crank wheels can be used on virtually any engine and is one of the more popular aftermarket options. It provides very good resolution in the higher tooth count versions (Eg 36-1 and 60-2) without being CPU intensive to decode.

Timing Requirements

The missing tooth crank and cam decoders require that the wheel is spinning at roughly the same speed throughout the rotation. For single missing tooth decoders: If the next tooth does not come within 1.5 * The Delta Time of the last 2 teeth it is assumed we just observed the missing tooth. For more than one missing tooth decoder there is a bit more leeway if the next tooth does not come within 2 * The Delta Time of the last 2 teeth it is assumed we just observed the missing teeth.

Usually this can be fixed by ensuring that the starter motor has enough current available to power through any harder spots through the rotation / opening closing cams / engine accessories.

If the starter motor is good and getting the right voltage ensure the mechanical components of the engine are correct.

Tuner Studio Configuration

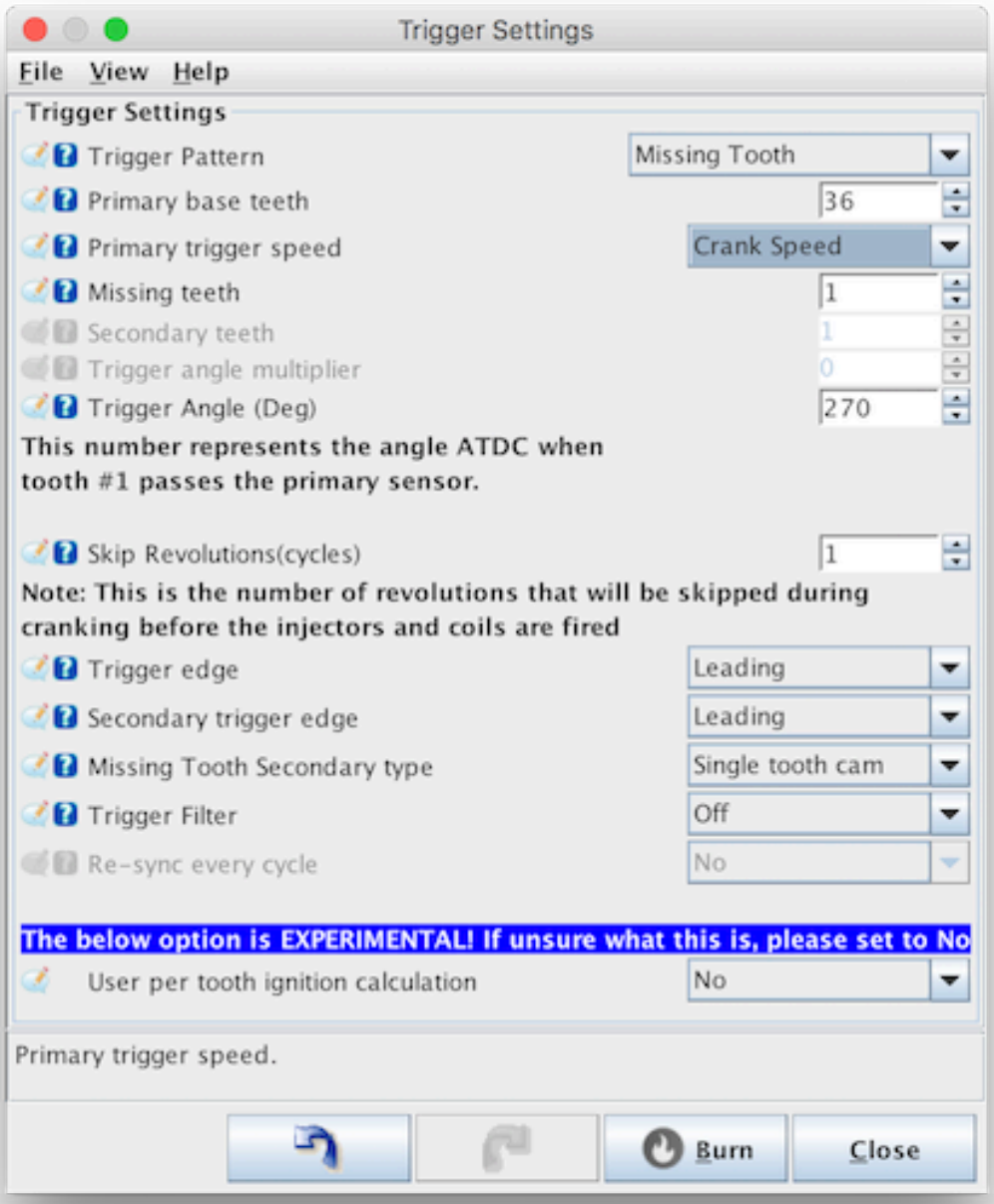


Figure 57: missingtooth_triggerconfig.png

Fields:

- Primary base teeth: This is the number of teeth the wheel would have if there were none missing. Eg a 36-1 wheel has only 35 actual teeth, but you would enter 36 into this field.
- Missing Teeth: The size of the 'gap' in the number of teeth. These missing teeth must be situated in a single block (ie there's only a single gap in the teeth)
- Trigger Angle: This is the angle in crank degrees **AFTER** TDC (ATDC) of the first tooth following the gap

Timing Setting

The trigger angle can be found by placing the engine at TDC, then calculating how far it must be rotated until the first tooth after the gap reaches the sensor.

Sequential operation

The missing tooth decoder supports sequential operation if an additional cam input is present. If Sequential mode is selected for either the fuel timing or spark timing, the system will expect to see a cam signal and will not sync correctly without this. Note that this is ONLY the case if sequential is selected for one or both of fuel and spark timing.

This cam signal should take the form of a single pulse every complete cycle. This can be a short tooth or a half moon type arrangement, provided that electrically there is only a single rising (or falling) pulse per cycle.

Trigger Diagram

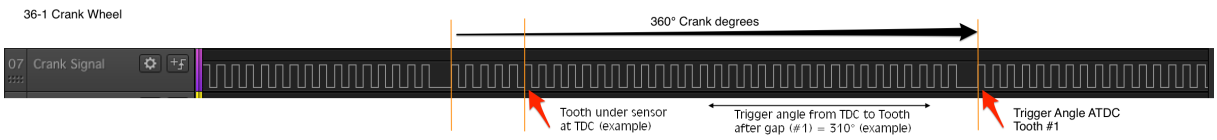


Figure 58: missingtooth_trace1.png

Missing tooth (Cam speed)

The missing tooth cam-speed trigger is a Speeduino innovation, that permits function similar to a dual-wheel setup, thereby allowing sequential or wasted spark operation from cam-mounted or distributor wheels. The operation is based on both Missing Tooth and Dual Wheel. It is suggested to read those

sections first for familiarization as this section will only highlight the fundamental differences to those common decoders.

This decoder is comprised of a single cam-speed wheel in the same configuration as a crank-mounted missing-tooth wheel. The number of teeth ***must*** be evenly divisible into 720°. As it rotates at half crank speed, the sensor reads half the wheel teeth on each 360° crank revolution, and the remaining teeth on the next crank rotation. A single missing tooth will appear on only one of the two crank rotations, and is then used as a phase indicator, much as the dual-wheel system uses the cam signal.

Applications

Missing tooth cam or distributor wheels can be used with cam or distributor wheel modification or fabrication as no OEM systems use it originally. The wheel ***must*** have at least as many teeth as cylinders, ***not*** including the missing tooth. This generally requires double the number of teeth as cylinders or more. As many teeth, slots, or other readable features (sensor targets) as possible in the limited space is recommended in order to satisfy this requirement, and to maximize resolution. The sensor must be capable of reliably reading smaller or closely-spaced teeth.

Due to typically limited teeth, only half the teeth being read on each revolution, and the potential for reduced accuracy due to timing drive wear; the timing accuracy may be reduced in comparison to crank wheel systems. A figure of error cannot be predicted here as the wear or 'slop' of a given engine will be unique. However, it should be reasonable to assume the timing error will not exceed the accuracy of an OEM-equivalent cam-driven system such as typical distributor systems, or possibly better due to more sensor targets.

Timing Requirements

The missing tooth crank and cam decoders require that the wheel is spinning at roughly the same speed throughout the rotation. For single missing tooth decoders: If the next tooth does not come within 1.5 * The Delta Time of the last 2 teeth it is assumed we just observed the missing tooth. For more than one missing tooth decoder there is a bit more leeway if the next tooth does not come within 2 * The Delta Time of the last 2 teeth it is assumed we just observed the missing teeth.

Usually this can be fixed by ensuring that the starter motor has enough current available to power through any harder spots through the rotation / opening closing cams / engine accessories.

If the starter motor is good and getting the right voltage ensure the mechanical components of the engine are correct.

Tuner Studio Configuration

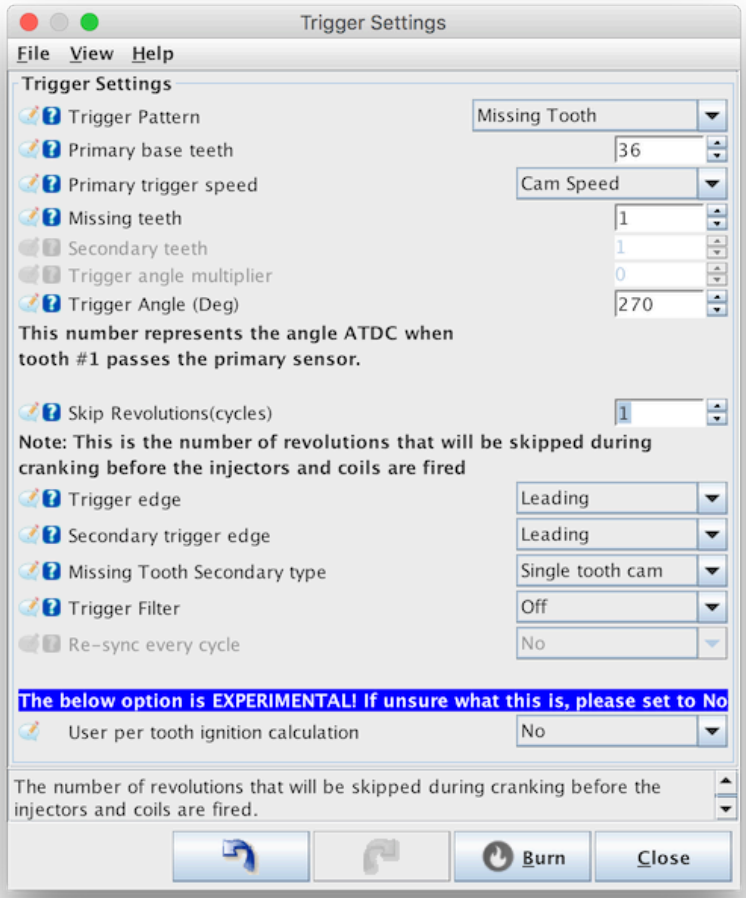


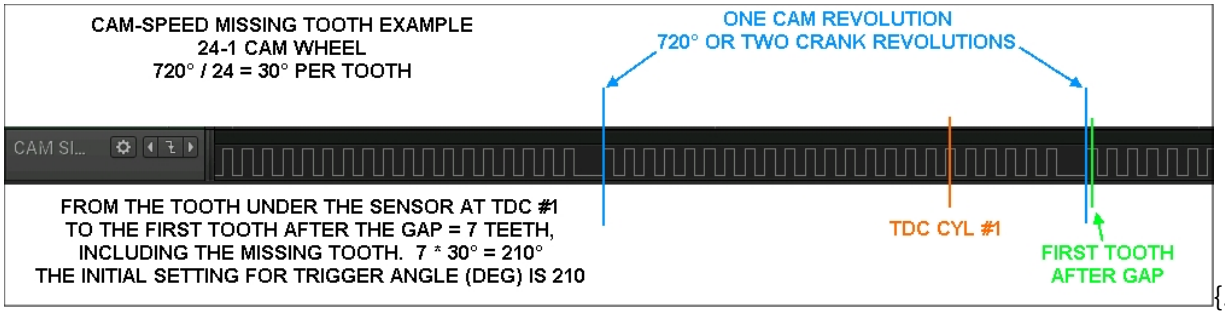
Figure 59: missingtooth_cam_triggerconfig.png

Fields: - **Primary base teeth**: This is the number of teeth the wheel would have if there were none missing, e.g. a 36-1 wheel has only 35 actual teeth, but you would enter 36 into this field. - **Missing Teeth:** The size of the 'gap' in the number of teeth. These missing teeth must be situated in a single block (ie there's only a single gap in the teeth). One missing tooth is recommended. - **Trigger Angle**: This is the angle in **crank degrees AFTER** TDC (ATDC) of the first tooth following the gap. This number ranges from -360° to +360°. - **Cam Speed**: Ensure this box is checked for this cam-speed system.

Timing Setting

The trigger angle is set in CRANK degrees, not cam.

Trigger Pattern



{.align-center width75%}

Dual Wheel

A dual wheel trigger is one where there is a primary multi-tooth wheel combined with a secondary single pulse to provide location information. The primary input should contain no missing teeth. Both pulses can run at either cam or crank speed, but sequential operations requires that the secondary pulse is located on the cam. The design of the secondary trigger can vary (Eg a single short tooth, half-moon wheel etc), provided it only provides a single pulse per revolution.

As with other arbitrary tooth count wheels, the number of teeth must evenly divide into 360 (or 720 if running at cam speed).

Tooth #1 is defined to be the first tooth on the primary wheel AFTER the pulse on the secondary wheel.

Applications

Dual wheel triggers are standard fitment on a number of Euro make cars, particularly those from VW and Audi. They are also a popular aftermarket fitment due to their simplicity and ease of fitment.

Tuner Studio Configuration

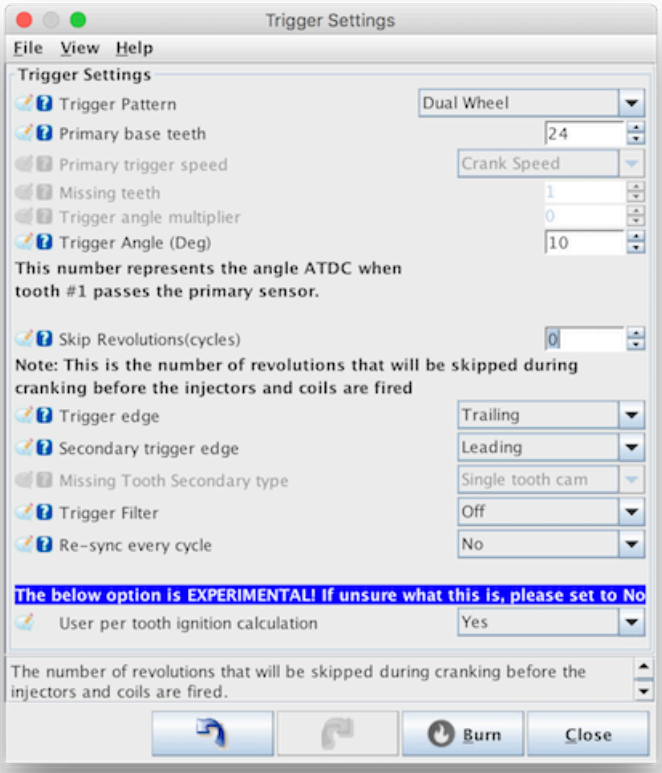


Figure 60: dualwheel_triggerconfig.png

Fields:

- **Primary base teeth**: This is the number of teeth on the primary input wheel. If the primary wheel is located on the cam (or is otherwise running at cam speed), divide it's teeth by two for this setting
- **Trigger Angle**: This is the angle in crank degrees **AFTER** TDC (ATDC) of the first tooth on the primary input, following the single pulse on the secondary input.
- **Trigger edge**: Whether the trigger will be taken from the leading (rising) or trailing (falling) edge of the primary input
- **Secondary trigger edge**: As above, but for the secondary input
- **Re-sync every cycle**: Whether the system will reset the sync level every time the secondary input is seen. This can be useful for noisy crank triggers that otherwise my lose sync permanently

and not recover until a restart.

Timing Setting

The trigger angle can be found by placing the engine at TDC, then calculating how far it must be rotated until the first primary tooth after the secondary pulse.

Sequential operation The missing tooth decoder supports sequential operation if the secondary input is running at cam speed. If Sequential mode is selected for either the fuel timing or spark timing, the system will expect that the secondary input is running at cam speed and will only provide half the output pulses if this is not the case.

This cam signal should take the form of a single pulse every complete cycle. This can be a short tooth or a half moon type arrangement, provided that electrically there is only a single rising (or falling) pulse per cycle.

Basic Distributor

The Basic Distributor trigger is a very simplistic decoder that expects input like what a traditional distributor outputs. That is, 1 pulse per cylinder per cycle.

For this reason the signal lacks any cylinder position signal and so without a missing/added tooth or camshaft signal reference Speeduino cannot calculate crankshaft angle, phase of cycle, or cylinder assignment. **A distributor must be used to route the resulting sparks to the correct cylinders** (With the exception of single cylinder engines).

The signal can be as simple as the breaker points from an old pre-electronic distributor, to a crankshaft wheel without any abnormal, extra, or missing slots, provided it is conditioned appropriately to 0v-5v. Most who have installed aftermarket tachometers are familiar with the simplicity of the signal with the only variation being the number of pulses in each crankshaft rotation.

A note on resolution

By its nature the resolution of a simple distributor wheel is quite low. The exact resolution depends on the number of cylinders (The more cylinders, the higher the resolution), but even with an 8 cylinder engine there are only 4 pulses per revolution. This can impact timing accuracy if running ignition control from Speeduino and in most cases upgrading to a higher resolution trigger wheel is strongly recommended.

Trigger Signal

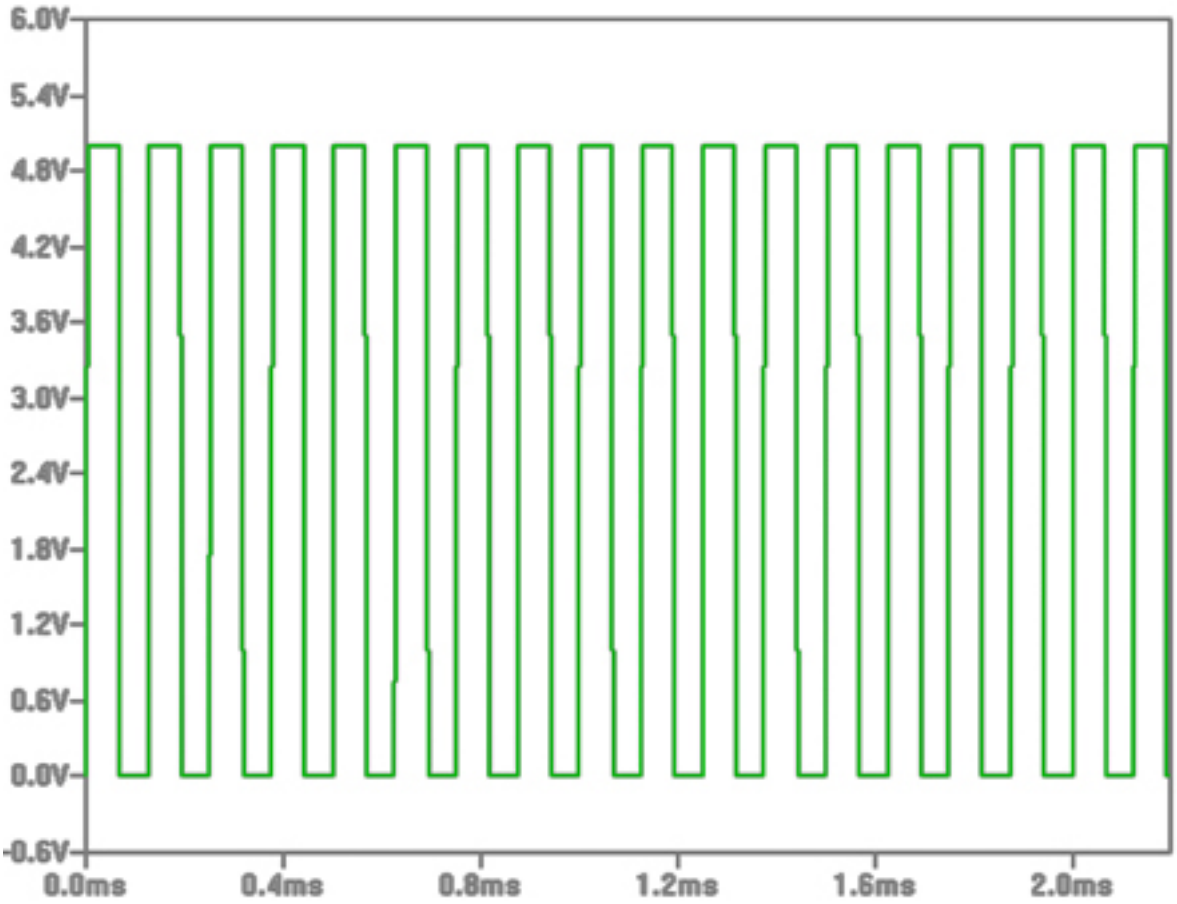


Figure 61: Basic_sistributor case.png

GM 7X

This decoder uses a GM trigger wheel with six notches spaced evenly apart and one uneven notch. The uneven notch is counted as #3 with a total of seven notches.

4G63 Pattern

The 4g63 trigger is used across a large number of both Mitsubishi and Mazda 4 cylinder engines. See below for applications.

It is comprised of crank and cam signals that are provided by either a hall sensor or an optical sensor. The signal is electrically the same in both cases.

Applications

- Mitsubishi Lancer
- NA Miata / MX-5 (Up to 1997)

Tuner Studio Configuration

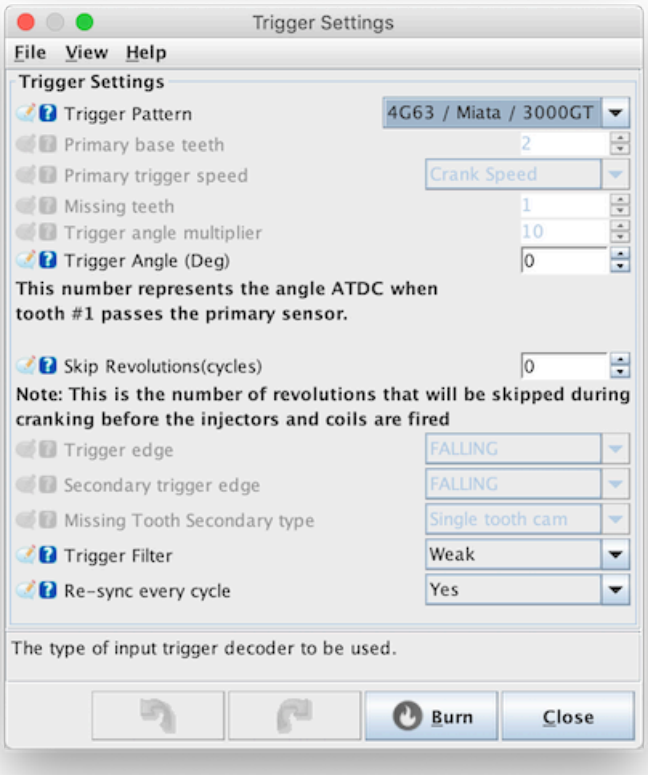


Figure 62: 4g63_triggerconfig.png

> **NOTE** Within the `Cranking options` dialog, ensure that the `Fix cranking timing with trigger` option is turned **ON** {.is-warning}

Timing adjustment

In most cases altering the trigger angle should not be required, however there is some small variation between the OEM versions of thIs trigger so some minor adjustment may be needed. Once you have

the engine started, set a fixed ignition angle and check the timing with a timing light. If this is a few degrees out (<20°), adjust the trigger angle here. If this is more than 20° out, there may be a larger problem.

Trigger Pattern

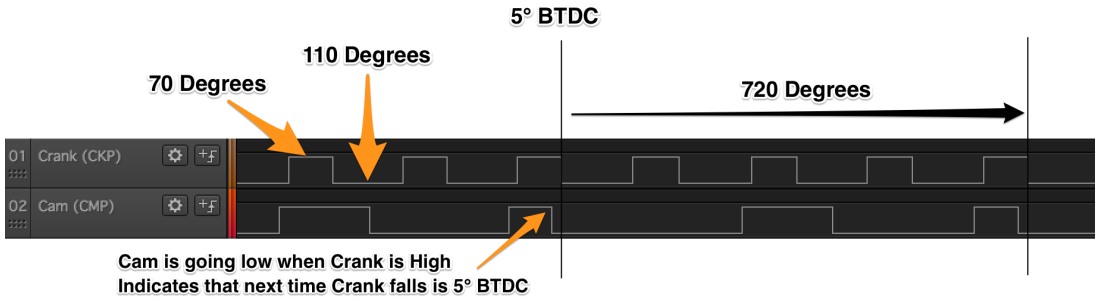


Figure 63: 4g63_trace.png

Overview

This a 24 tooth wheel with 12 wide teeth and 12 narrow teeth. The narrow provides 3 degrees of pulse while the wide provides 12. All of the falling edges are 15 degrees apart. This decoder uses the falling edges, requiring the cam signal to determine crank angle.

Trigger Signal

File:24x.png

Overview

There are two signals one from the crank wheel and the other from the cam. The crank wheel puts out a series of four pulses every 120 degrees. Each of the four pulses is 20 degrees apart and lasting only 2 degrees. The cam wheel pluses once every 360 degrees or 720 crank degrees. The pulse last for 180 degrees or 360 crank degrees.

Trigger Signal

File:Syncsignal.jpg

Harley Evo

The Harley EVO pattern is used on V-Twin engines from ’86 through to ’99.

This pattern will work on all injected EVO engines.

Overview

The Honda D17 decoder applies to the Honda engine family using a 12+1 crankshaft wheel. The 4+1 camshaft signal is not currently used with Speeduino. Without the cam signal, all standard fuel and ignition modes up to semi-sequential and wasted-spark are supported.

Applications

- TBA

Tuner Studio Configuration

Timing adjustment

In most cases altering the trigger angle should not be required, however there is some small variation between the OEM versions of thIs trigger so some minor adjustment may be needed. Once you have the engine started, set a fixed ignition angle and check the timing with a timing light. If this is a few degrees out (<20°), adjust the trigger angle here. If this is more than 20° out, there may be a larger problem.

Trigger Pattern

The crank trigger wheel consists of 12 evenly spaced teeth plus 1 additional 13th tooth which provides position information. The first tooth after this 13th one is considered Tooth #1

Figure 64: honda_D17.png

Miata 99-05

From MY99 onwards, Miatas moved to a new trigger patter that, whilst similar to that used on the 4g63, is more tolerant to noise and does not rely on both edges of a tooth being tracked. Crucially it also permits movement of the cam signal relative to the crank signal which is required due to the addition of variable cam timing in these engines. Sync can be determined in the same way regardless of if the variable cam is at it's maximum or minimum movement.

The trigger consists of a 4 tooth wheel located on the crankshaft and a 3 tooth wheel on the cam. The teeth on both wheels are unevenly spaced.

Applications

NB Miatas from 1999 until 2005.

Tuner Studio Configuration

- The trigger angle should not need changing once this pattern has been selected (ie Make sure it is set to 0)
- Both trigger edges should be set to `RISING`
- For most installs, Trigger filtering set to Off or Weak is sufficient.
- In the `Starting`/`Idle` -> `Cranking Settings` dialog ensure the following options are turned on:
  - 'Fix cranking timing with trigger'
  - 'Use new ignition mode'

Trigger Pattern

The crank wheel contains 4 teeth, separated by an alternating 70 and 110 degrees.

Sync is determined by counting the number of secondary (cam) pulses that occur between the primary (crank) pulses and can be confirmed at 2 points in the cycle. The first crank pulse after 2 cam pulses is tooth #6 and the first crank pulse after a single cam pulse is tooth #2. Tooth #1 is located at 10 degrees BTDC and cannot be identified directly, only relative to teeth #2 and #6. As the camshaft timing is moved as part of the VVT, the secondary pulses remain within the same 'window' relative to the primary pulses. Sync can therefore before confirmed at all loads and speeds, no matter what VVT value is being currently used.

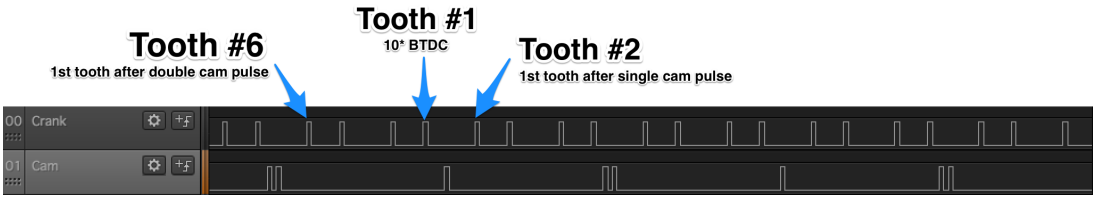


Figure 65: miata9905.png

Non-360 Decoder

This is a variation of the dual-wheel decoder that can be used with tooth counts that do not divide evenly into 360° This decoder system is usually unique to a particular brand or engine series, and therefore has previously been assigned a name to identify the type, such as the Audi 135 decoder. While this “uneven divisor” decoder can be used with a variety of tooth counts, not all tooth counts can be used with this system.

Overview

The Daihatsu +1 triggers are used across a number of 3 and 4 cylinder engines from Daihatsu. See below for applications.

It is comprised of a single cam signal provided by either a hall sensor. This should be fed into the RPM1 input on Speeduino

Applications

- TBA (3 cylinder)
- TBA (4 cylinder)

Tuner Studio Configuration

Simply select the Daihatsu +1 trigger option.

Timing adjustment

In most cases altering the trigger angle should not be required, however there is some small variation between the OEM versions of thIs trigger so some minor adjustment may be needed. Once you have

the engine started, set a fixed ignition angle and check the timing with a timing light. If this is a few degrees out (<20°), adjust the trigger angle here. If this is more than 20° out, there may be a larger problem.

Trigger Pattern

In 3 cylinder engines, there are 3 evenly spaced teeth at 0, 240 and 480 crank degrees. There is an additional (+1) tooth located at 30 crank degrees to provide position info

The 4 cylinder is the same, except with 4 evenly spaced teeth. The 5 teeth are therefore located at 0, 30, 180, 360 & 540 (Crank degrees, ATDC)

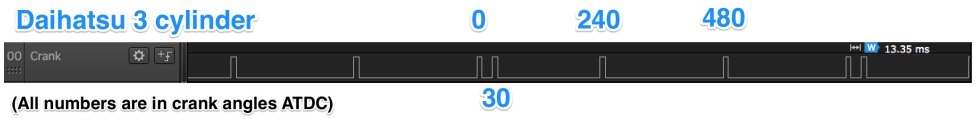


Figure 66: daihatsu_3_1.png

Subaru 36-2-2-2

The 36-2-2-2 wheel is common on many 4 cylinder Subaru engines from approx. 2000 onwards. It utilises a crank trigger wheel containing a nominal 36 teeth, spaced 10 crank degrees apart, and 3 groups of 2 missing teeth. These missing tooth groupings allow for sync to be determined within at most 1/2 a crank turn.

Early wheels were VR triggered however after the switch to variable valve timing, Subaru switched to Hall sensors. Most configurations are paired with one or two 4-1 cam sensors, however these are not required for sync on Speeduino.

> Note that there are 2 variations of the 36-2-2-2 pattern, the H4 and the H6. Whilst visually very similar, the patterns have different groupings of teeth and are not compatible. **This trigger is for the H4 variant** {.is-info}

Configuration

Trigger Angle: 0 **Trigger Edge:** FALLING **Secondary Trigger edge:** N/A **Skip Revolutions:** 1 **Trigger Filter:** Weak (Depending on install)

Trigger Pattern

The 3 sets of 2 missing teeth are located such that one group is on its own and the other two are located adjacent to one another, with a single tooth in between. Sync can be determined by detecting the missing 2 teeth, then seeing if there is another set of missing teeth immediately after it.

Cylinder 1 TDC compression happens on the fourth tooth after the single gap. Speeduino watches for any missing tooth period, then waits to confirm whether it is followed by another. Sync can therefore be determined in this manner at 2 points in a single crank revolution.

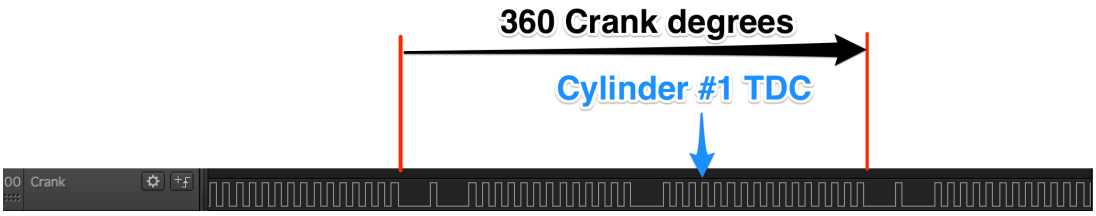


Figure 67: 36-2-2-2.png

Note: Many diagrams and trigger wheel images available online show the wheel from the backside, making it show as rotating counter clockwise. For the correct orientation, when looking at the front of the engine, the wheel spins clockwise.

V0.4 Board

Overview

The v0.4 board is a testing board that was developed with the goals of reproducing the existing v0.3 boards capabilities, but with the following improvements:

- Lower cost (Primarily due to reduced size, but also some component changes)
- More compatible with off the shelf cases/enclosures
- Stepper-style IAC driver option
- Has a single 40 pin connector for all IO (Excluding 12v power)

> **Note:** The v0.4 is **NOT** intended as a replacement for the v0.3 line of boards! The 2 are designed with different goals in mind. The v0.4 is intended to be integrated more closely into existing wiring, with the aim being that interface boards can be used to easily connect through the IDC40

> connector. Unless you understand the interface on the v0.4 board and believe it is the best option for your install, the v0.3 may well be a better option for you. {.is-warning}

Board Features

The v0.4 boards includes the following features:

- 4 injector channels
- 4 Ignition outputs
- Fully protected input channels for CLT, IAT, TPS and O2
- Optional VR conditioner mount on crank and cam inputs
- MAP sensor mount location
- DRV8825 stepper module mount location
- 4 medium-current spare outputs (e.g., fuel pump, thermo fan, boost control, VVT, etc)
- 5 unpopulated/configured optional low-current spare outputs in “proto” section, including tachometer-out
- A single 40-pin IDC connector includes all pins required for the board with the exception of the 12v input

Physical Layout

Note that there are some differences between the various versions of the board, however the pinouts on the main IDC40 connector remain the same.

Note: Injector Pins have 1/2 and 2/2 markings this is to more easily and clearly route injectors for semi sequential and batch modes. If the application requires less than 4 injectors simply use either pin 1/2 or 2/2. If the application requires 5 or more injectors it is recommended to use both 1/2 and 2/2 when available to more evenly distribute the current from the injector coils triggerting. See Injector Wiring for more specific details.

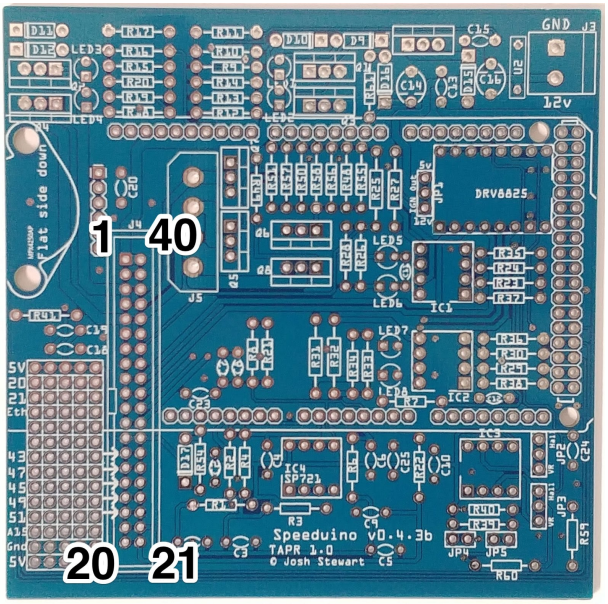


Figure 68: v0_4_board_annotated_1.jpg

Pin #	Function
1	Injector 1 - Pin 1/2
2	Injector 2 - Pin 1/2
3	Injector 3 - Pin 1/2
4	Injector 3 - Pin 2/2
5	Injector 4 - Pin 1/2
6	Injector 4 - Pin 2/2
7	Ignition 1
8	Ignition 4
9	Ground
10	Ground
11	MAP Sensor (0v-5v)
12	Ground
13	5v
14	Proto Area 1 (0.4.4b+ Flex Sensor)
15	Proto Area 2 (0.4.4b+ Fan)

Pin #	Function
16	Proto Area 3 (0.4.4b+ Fuel Pump)
17	Proto Area 4 (0.4.4b+ Tachometer)
18	Proto Area 5 (0.4.4b+ Clutch)
19	Coolant (CLT)
20	Inlet Air Temp (IAT)
21	O2 Sensor
22	TPS input
23	Ground
24	Cam Input / VR2+
25	Crank Input / VR1+
26	VR2- (Not used for hall sensor)
27	VR1- (Not used for hall sensor)
28	5v
29	Idle Stepper 2B
30	Idle Stepper 2A
31	Idle Stepper 1A
32	Idle Stepper 1B
33	Ignition 3
34	Ignition 2
35	Boost
36	Idle 2 (For use with 3 wire idle valves)
37	PWM Idle
38	VVT
39	Injector 2 - Pin 2/2
40	Injector 1 - Pin 2/2

Board Assembly

Assembly of a complete board is virtually identical to the v0.3 and remains relatively straightforward with all components being through hole and labelled on the board. Whilst it does not technically matter which order components are installed, the following is recommended for simplicity:

1. All resistors
2. All diodes (Including LEDS)
3. All capacitors > Take note that C14 and C16 are polarised capacitors, meaning that they must be put in the correct way around. The capacitors should be marked with a + sign on one side. On the PCB, the positive side is indicated by a line on the capacitor symbol. {.is-warning}

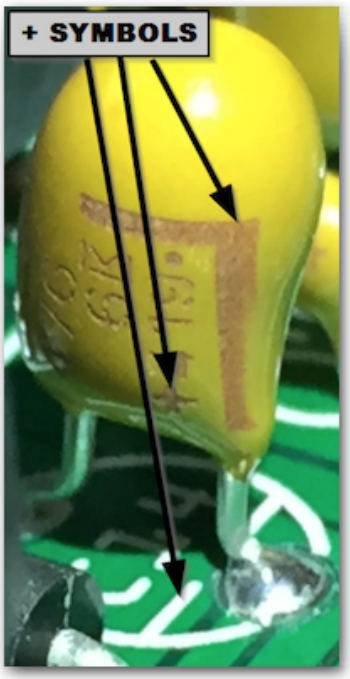


Figure 69: capacitor_orientation.png

4. All jumper headers (JP*)
5. Arduino pins:
    - **Suggested method**: Break header pins into required lengths and insert into an Arduino Mega. Place the board over the top of the pins and solder in place
    - Note that not all the pins on the end double row need to be populated (Though there's no harm in doing so). The odd numbered pins (Eg D23, D25 .. DD53) do not need pins on them.
6. IDC 40 connector

7. IC sockets
8. All screw terminals
9. All MOSFETs
10. Power regulator
11. MAP sensor (If used) > **NOTE:** ALL self assembly boards have the MAP sensor with the hole at the top. All assembled boards will typically have the hole on the bottom {.is-warning}

Assembly Instruction video

This video is for the v0.3 board, but it largely applies to v0.4 designs as well.

Board Configuration

The board can be configured in multiple ways depending on the hardware you use and way your setup is configured.

Board default outputs

Multiple functions within Speeduino have adjustable outputs or can be set to Board Default. The following are the Default pin outs for the v0.4, however all of these functions can be reassigned to other pins if required (Eg to use the onboard high current outputs)

Function	Board output	Arduino pin
Boost control	IDC Pin 35	7
VVT	IDC Pin 38	4
Idle 1	IDC Pin 37	5
Idle 2 (3 wire idle valves)	IDC Pin 36	6
Fuel pump	Proto area (45) (0.4.4b+ IDC 16)	45
Fan	Proto area (47) (0.4.4b+ IDC 15)	47
Tacho	Proto area (49) (0.4.4b+ IDC 17)	49
Launch / Clutch	Proto area (51) (0.4.4b+ IDC 18)	51

Optional Components

If using a VR crank sensor, the board will require the addition of a VR conditioner. The board has been designed to work with the dual VR conditioner from JBPerf (http://www.jbperf.com/dual_VR/index.html) which will plug directly in. These have been out of stock to purchase directly for some time but can be built from the parts list as the instructions are still available.

There is also an official VR board that can be purchased from the speeduino shop which also plugs in directly.

Most partner resellers have their own conditioners with other features such as LEDs for when the signal is triggering high / low. Other 3rd party VR conditioners will also likely work but obviously not all configurations can be validated.

SP721 Over-voltage Protection For users having difficulty obtaining the SP721 used in some versions, see info on the SP721 Diode Alternate page

Jumper Configs

Depending on the type of crank and cam sensors you have, there are a number of jumpers that will need to be set.

Some VR sensors can send high AC voltage into the arduino board. If you are unsure of your sensor type identify it before connecting it to the board. Using a VR sensor with the 'direct' pins closed (JP2) and or (JP3) may cause damage to the microprocessor. Danger!

The jumpers that need setting are:

- JP1 - This sets whether the Ignition outputs are 12v or 5v. Note that even if you set this to 12v you should **NOT** connect these directly to a high current coil. These outputs should only ever go to a logic level coil or an igniter
- JP2 - Whether or not the RPM1 (Crank) input should be routed via the (Optional) VR conditioner. This should be set to VR when using either a VR sensor or a hall sensor that switches between 0v-12v
- JP3 - Same as JP2, but for the RPM2 (Cam) input
- JP4 - 1k pullup resistor for RPM1 input. Should be jumpered ('On') when a sensor is used that switches between ground and floating (Which is most hall effect sensors)
- JP5 - Same as JP4, but for the RPM2 (Cam) input

To make this simpler, the most common sensor types and their required configurations are below:

Crank Sensor	Cam Sensor	JP2	JP3	JP4	JP5
Hall sensor	-	Hall	Off	On	Off
VR Sensor	-	VR	Off	Off	Off
Hall sensor	Hall sensor	Hall	Hall	On	On
VR Sensor	Hall sensor	VR	Hall	Off	On

40-pin connection

You can solder wires directly to the board or use IDC (Insulation Displacement Contact) connectors. The 40-pin IDC is the connector that was used on computer drive ribbon cables for years and old computer cables can be used. A heavier cable, called DuPont cable is recommend for long term use though. Later in the IDE/ATA interfaces life the speed was increased and this required a new fine 80-wire cable. These cables are **NOT** compatible. Some of the pins are connected together causing the magic blue smoke to be released.

Board revisions

Version	Changes	BOM
V0.4.4b	new ground up, all SMD, board design that includes additional onboard drivers and protection circuits. It is electrically and physically compatible with all other v0.4 versions.	Not Required
V0.4.4	Modified for easier automated assembly, including some SMD components and mounting the pressure sensor flat side up. Run/program switch added. Only sold officially as complete boards	Not Required
V0.4.3	Filter capacitors added to both primary and secondary RPM inputs. Voltage clamp added to secondary RPM input. Flex fuel input added to proto area	Download
V0.4.2	Considerable number of routing improvements. Neater proto area layout. Voltage clamp added to primary RPM input	Download

Version	Changes	BOM
V0.4.1	Added Proto area. Replaced diode array with SP721. Added optional high current aux output socket (J5). Diode relocated on power circuit to prevent USB back feeding 5v onto 12v rail when ignition off	Same as v0.4.2
V0.4	Initial release	Download

V0.3 Board

Overview

The v0.3 board was the first widely available Speeduino shield and is suitable for many typical 1-4 cylinder injection and ignition applications (Excluding direct injected engines). It uses screw terminals for all connections in order to make test wiring simple and quick for prototyping.

Board Features

The v0.3 boards includes the following features:

- 4 injector channels
- 4 Ignition outputs
- Fully protected input channels for CLT, IAT, TPS and O2
- Optional VR conditioner mount on crank and cam inputs
- MAP sensor mount location
- 4 medium current spare outputs (Eg Fuel pump, thermo fan etc)
- All I/O through screw terminals on the board
- Proto area with IO, SPI and power breakouts.

Physical Layout

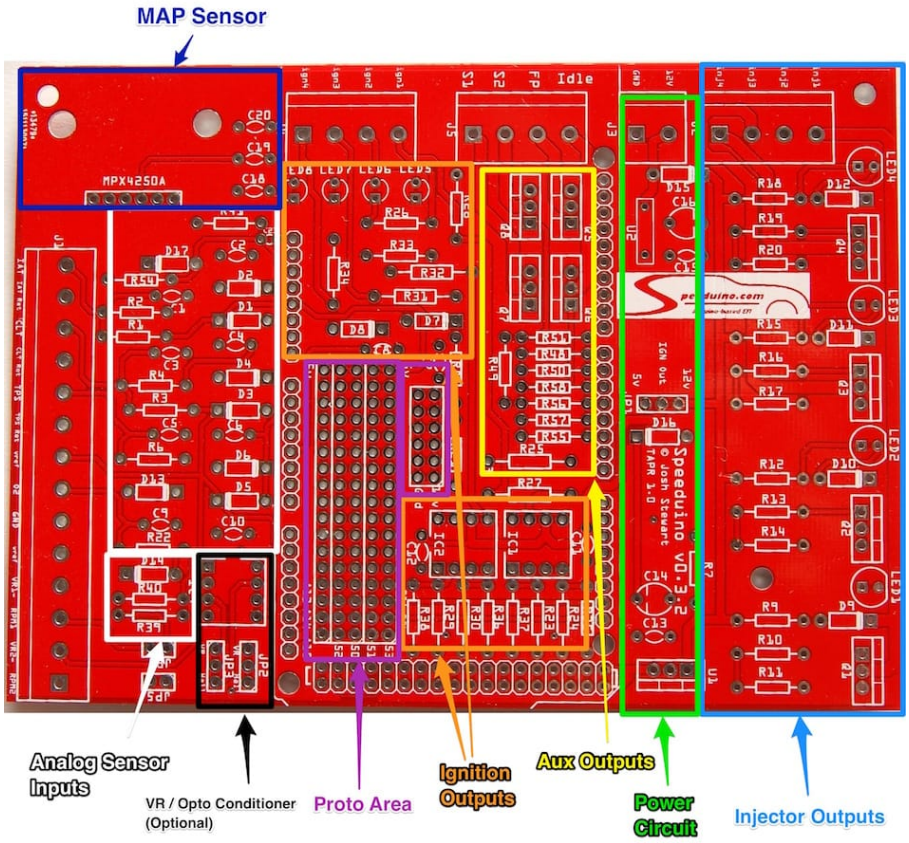


Figure 70: v0_3_2_board_annotated.jpg

Proto area

The proto area can be used for adding your own circuits on to Speeduino if required or simply as a convenient access point to various connections. The connections broken out to the proto board are:

- 5v and 12v
- Grounds
- SPI pins (MOSI, MISO, SCK and SS). Alternatively these can be used as generic digital IO (Arduino pins 50-53)
- 3 generic analog inputs (13-15)

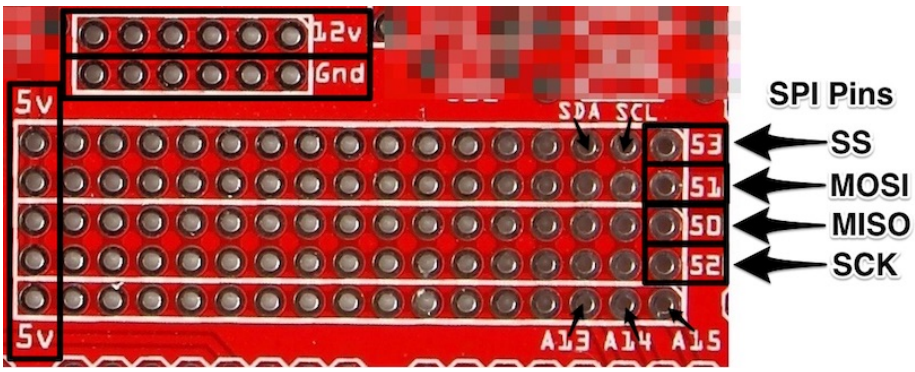


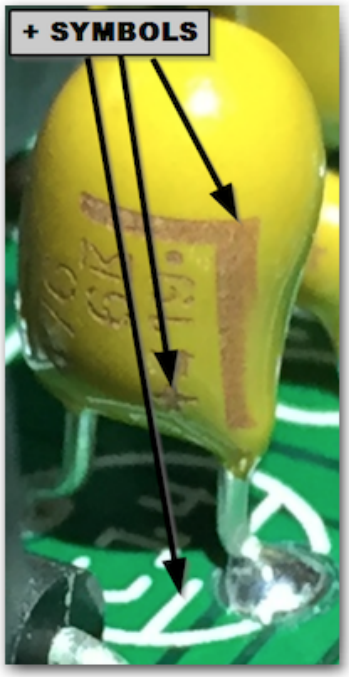
Figure 71: v0_3_2_proto_annotated.jpg

Board Assembly

Refer to the Board revisions for a link to the Bill of Materials (BOM) of your specific board.

Assembly of a complete board is relatively straightforward with all components being through hole and labelled on the board (See above mentioned BoM for parts list). Whilst it does not matter which order components are installed, the following is recommended for simplicity:

1. All resistors
2. All diodes (Including LEDS)
3. All capacitors
    - Take note that C14 and C16 are polarised capacitors, meaning that they must be put in the correct way around. The capacitors should be marked with a + sign on one side. On the PCB, the positive side is indicated by a line on the capacitor symbol.

- 

Correct capacitor orientation 4. All jumper headers (JP*) 5. Arduino pins: * Suggested method: Break header pins into required lengths and insert into an Arduino Mega. Place the board over the top of the pins and solder in place * Note that not all the pins on the end double row need to be populated (Though there's no harm in doing so). The odd numbered pins (Eg D23, D25 .. DD53) do not need pins on them. 6. IC sockets 7. MAP sensor (If used) * **NOTE:** ALL self assembly boards have the MAP sensor with the hole at the top. 8. All screw terminals 9. All MOSFETs 10. Power regulator

Assembly Instruction video

Board Configuration

The board can be configured in multiple ways depending on the hardware you use and way your setup is configured.

Board default outputs

Multiple functions within Speeduino have adjustable outputs or can be set to Board Default. The following are the Default pin outs for the v0.3

Note: *These defaults are applicable to the Jan 2017 firmware and newer*

Function	Board output	Arduino pin
Boost control	S2 Screw terminal	7
VVT	S1 Screw terminal	6
Idle 1	Idle Screw terminal	5
Idle 2 (3 wire idle valves)	Proto area (Labelled 53)	53
Fuel pump	FP Screw terminal	4
Launch/Clutch	Proto area (Labelled 51)	51

Optional Components

If using a VR crank sensor, the board will require the addition of a VR conditioner. The board has been designed to work with the dual VR conditioner from JBPerf (http://www.jbperf.com/dual_VR/index.html) which will plug directly in. Other VR conditioners will also likely work, but have not been tested. There is now also an official VR board that can be used, see link on the left.

SP721 Over-voltage Protection For users having difficulty obtaining the SP721 used in some versions, see info on the SP721 Diode Alternate page.

Jumper Configuration

Depending on the type of crank and cam sensors you have, there are a number of jumpers that will need to be set. The jumpers that need setting are:

- JP1 - This sets whether the Ignition outputs are 12v or 5v. Note that even if you set this to 12v you should **NOT** connect these directly to a high current coil. These outputs should only ever go to a logic level coil or an igniter
- JP2 - Whether or not the RPM1 (Crank) input should be routed via the (Optional) VR conditioner. This should be set to VR when using either a VR sensor or a hall sensor that switches between 0v-12v
- JP3 - Same as JP2, but for the RPM2 (Cam) input
- JP4 - 10k pullup resistor for RPM1 input. Should be jumpered ('On') when a sensor is used that switches between ground and floating (Which is most hall effect sensors)
- JP5 - Same as JP4, but for the RPM2 (Cam) input

To make this simpler, the most common sensor types and their required configurations are below:

Crank Sensor	Cam Sensor	JP2	JP3	JP4	JP5
Hall sensor	-	Hall	Off	On	Off
VR Sensor	-	VR	Off	Off	Off
0v-12v Hall Sensor (Requires VR Conditioner)	-	VR	Off	Off	Off
Hall sensor	Floating Hall sensor	Hall	Hall	On	On
VR Sensor	Floating Hall sensor	VR	Hall	Off	On

Board revisions

Version	Changes	BOM
V0.3.7	Added bluetooth header	Same as v0.3.6
V0.3.6	Added filter caps to both crank and cam inputs	Download
V0.3.5	Added flex fuel input to proto area. Many routing improvements.	Download
V0.3.4	Routing cleanup and more useful silkscreening	Same as v0.3.3
V0.3.3	Replaced diode array with SP721	Download
V0.3.2	Added Proto area. Removed spare IC socket (Had not been used)	Download
V0.3.1	Moved MAP sensor closer to edge of board. Beefier routing on the high current outputs (Including injectors)	Download
V0.3	Initial release	Download