Contents

So what's the go?

The Speeduino project aims to create a fully featured, totally open source (Hardware and firmware) Engine Management System (EMS) / Engine Control Unit (ECU) on top of the | Arduino Mega platform. In order of priority the specific aims are:

- Low barrier to entry (ie price and availability of hardware, clear, well documented code, easily accessible software development etc)
- Capabilities / Features
- Simplicity of development

In short, the main goal is to be in all places as simple as possible. No weird build environments, no knowledge of assembly needed, favour simplicity over performance where needed and make as low a barrier to entry as can possibly be achieved. Both the hardware and software/firmware sides of the system are covered with all being covered under open licenses.

Current Feature List

What you will get is flexible and configurable fuel and ignition management that will work for the majority of every day type setups. Currently supported features include:

- 16x16 3D fuel and ignition maps, with base of either TPS (Alpha-N) or MAP (Speed Density)
- Supports up to 8 cylinders fuel and ignition with 4 channels of fuel and 4 channels of ignition outputs
  - 1, 2 (Even fire only), 3 and 4 cylinder engines with full sequential fuel and ignition
  - 6 (even fire only) and 8 cylinder engines are supported with wasted spark and 2 squirts per cycle
- 6x6 3D individual cylinder trim on engines up to 4 cylinders
- After Start Enrichment
- Rev limiting (Spark based, hard and soft)
- Cranking specific enrichment, dwell timing and advance
- General logging through TunerStudio
- High speed tooth logging
- TPS calibration through TunerStudio
- Sensor calibration through TunerStudio (Coolant, IAT and O2)
- Warm Up Enrichment (WUE)
- TPS based acceleration enrichment

- Tacho output
- Fuel pump activation/deactivation (With priming)
- Over dwell and over duty protection
- Battery voltage compensation for dwell and injectors
- Modular wheel decoder support. Included decoders:
  - Missing tooth (Eg 36-1, 60-2 etc)
  - Dual wheel (Evenly spaced teeth on crank, single tooth on cam)
  - Basic distributor
  - GM 7X
  - GM 24X
  - 4g63 aka 4/2
  - “Jeep 2000”
  - Audi 135
  - Miata 99-05
  - Honda D17 (12+1)
  - Nissan 360
  - Subaru 6/7
  - Taking requests…
- Open and closed loop idle control (PWM and Stepper)
- Closed loop boost control
- Open loop VVT control
- Deceleration fuel cut off (DFCO)
- Launch control
- Flex fuel
- O2 based autotune (Registered version of TunerStudio required)

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)
- **Raspberry Pi** SpeedyLoader.AppImage
  - Linux / RaspberryPi versions require libusb libraries to be installed. EG if on Debian/Ubuntu: `sudo apt-get install libusb-1.0-0 libusb-0.1-4:i386`

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 http://platformio.org/platformio-ide
- 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: https://github.com/noisymime/speeduino/releases
2. If you want the latest and greatest (And occasionally flakiest) code, the git repository can be cloned and updated. See https://github.com/noisymime/speeduino

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 (square IC) - drivers included in Windows, MacOS and Linux:

or

WCH CH340G (Rectangular IC) - 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 (Looks like an arrow point to the right)

Assuming all goes well, you should see the IDE message that avrdude is done, similar to this:

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):

```
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:

title: Hardware Requirements description: published: true date: 2020-01-14T07:31:48.212Z tags: wiring, hardware —

Hardware Requirements overview

This page presents the basic hardware requirements of a Speeduino system, as well as a number of options for different variations of these. It does not represent every supported combination of hardware, but provides an overview if you're starting out.

Arduino

Speeduino uses the Arduino Mega 2560 R3 as the controller. All official and most clone Arduino Mega 2560 boards will work fine, but it is recommended to use a board that has the 16u2 serial interface rather than the cheaper CH340. Which chip a board uses can usually be found on the information/specification listing from most retailers, but if in doubt, ask the seller you are looking to buy from.

Inputs

Crank sensor

This is arguably the most important sensor for Speeduino to function correctly. The signal going to the Arduino must be a 0v-5v square wave series of pulses (shown below) representing teeth on a wheel running at crank (or cam) speed. Many Hall and "opto" sensors meet this digital square-wave spec. If only a crankshaft trigger wheel is used (no cam signal), the crank wheel must have a "missing" tooth in order to provide position information as well as the engine RPM. Tested missing-tooth wheels currently are 4-1, 12-1, 36-1 and 60-2.

Alternatively (and necessary for full-sequential injection) an added cam signal with or without crank wheel missing teeth. These setups are indicated by the added "/x", such as 60-2/1, for a 60-tooth crank wheel, with 2 missing teeth, and a 1-tooth cam signal per cycle. Cam-speed missing-tooth wheels can also support semi and full-sequential.

VR (variable reluctance) sensors can also be used, however as the board does not contain any sort of signal conditioner to convert the sine wave (below) to the required square wave, an additional module

will be needed. An 8-pin DIP socket is located on v0.3.x and v0.4.x series official boards for this purpose as IC3. The MAX9926 chip has been tested to work with most types of input signals, and is available from the Speeduino Store, however any similar module that outputs a 0v-5v square wave (LM1815, LM358, SSC/DSC, many OEM modules, etc.) should also work fine with VR sensor signals.

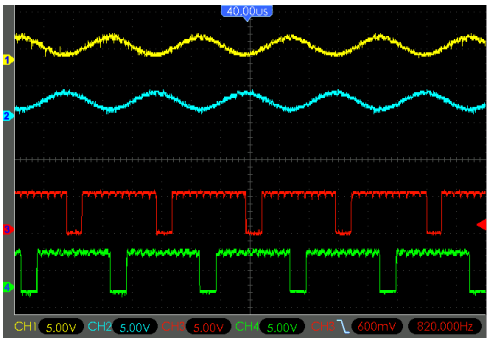

Figure 0.1: vr_wave.gif

TPS

TPS sensor must be of the 3 wire potentiometer type, rather than the 2 wire on/off switches found on some throttles. If your TPS is a 3 wire sensor then it will likely work, however you will need to confirm it is a potentiometer (variable) type sensor.

The TPS functions by sending an analog variable voltage signal to Speeduino in order to report the current position of the throttle. It is typically supplied with V+ of 5V and ground (GND, signal ground, or signal return), routing through an internal potentiometer to output a low voltage at low throttle opening, and a rising voltage with greater throttle opening.

If using a TPS with unknown connections; it is recommended to test the TPS with an ohm meter in order to determine the connection of each pin without risking damage by applying sensor power randomly. This can be accomplished on the bench or with the engine off and TPS disconnected:

- Assign a letter to each pin.
- Attach the ohm meter to two pins, and operate the throttle from closed (idle) to wide-open (WOT), recording the results.
- Find the pair of pins where the resistance does not change significantly from idle to WOT. These are your two power pins.
- The remaining pin is your **Signal** pin.
- In order to determine which power pin is **V+** and which is **GND**, test ohms between the **Signal** pin and one power pin.
- In idle position; if ohms are low that power pin is **GND**. If ohms are high that power pin is **V+**.

Most usable TPS sensors have 3 pins. If your TPS has a different number of pins, referring to the original engine wiring diagram may show the function, and whether it is usable or which pins to use for Speeduino. For TP sensors that work "backwards", and wiring cannot be changed, a simple code modification is available on the Forums to make use of this type of TPS.

MAP (Manifold Pressure)

Recommended MAP sensor is the MPX4250 from Freescale, however many MAP sensors are supported. If you want to use one that is not included in the list (Under Tools->Map Calibration in TunerStudio) then please make a new thread in the forum requesting this. Other sensors can and will work just fine, but you will need to calibrate these within TunerStudio against a different set of values.

Temperature Sensors (CLT and IAT)

Any standard 2-wire thermistor sensor can be used for these temperature functions. The sensors have 1 side connected to a ground (Preferably from the ECU) and the other running to the signal line. These sensors have no polarity, so the orientation of these wires does not matter.

For full details, please see the Sensor Calibration page

Exhaust Gas Oxygen Sensors (O2 and WBO2)

The type of O2 sensor (narrow or wide-band) must be selected in TunerStudio under *Tools > Calibrate AFR Table*.

Narrow-band

NBO2 sensor signals are read directly by Speeduino. TunerStudio applies the standard non-linear 0-volt to 1-volt values for all standard NBO2 sensors automatically under calibration. Once set in calibration, Speeduino will use the designated NBO2 to adjust fueling according to the entries you make in the AFR table (*Tuning > AFR Table*), and the sensor is selected for type and parameters (or disabled) under *Tuning > AFR/O2*. Note that narrow-band sensors were originally designed to target stoichiometric AFR (Lambda 1.0) for efficient catalytic emissions control, and are generally not sufficiently accurate or suitable for tuning efficient lean economy or rich power fueling. While not recommended; involved tuning methods are available to allow limited and approximate tuning for lean and rich AFRs using a NBO2 sensor.

Wide-band

Wide-band oxygen (WBO2) sensors can detect and report a wider range of lambda (⍺) or AFRs than narrow-band, and with greater accuracy, from approximately 10:1 to 20:1 (about 0.7 to 1.3 lambda), depending on specific sensor version and controller. Speeduino cannot use WBO2 sensors directly, requiring an external controller to process the signal and to apply sensor heating control. Enter the controller brand and model from the list displayed. If the controller signal is generic linear or custom, select and enter the required information, or an option to install a custom INC file is available in the menu list.

Once set in *Tools* > *Calibrate AFR Sensor*, Speeduino can use the designated WBO2 to report lambda/AFR to TunerStudio for gauge display. After the sensor is selected for type and parameters under *Tuning* > *AFR/O2* it can adjust corrective fueling on-the-fly according to the entries you make in the AFR table (*Tuning* > *AFR Table*), and for auto-tuning in TunerStudio, or MegaLogViewer in real-time or from logs. Settings also include the option to disable. Although Speeduino can use the WBO2 information to correct fueling; it is strongly suggested it not be used to compensate for poor tuning.

Application-Specific Inputs

Circuits and techniques Speeduino users have found useful for adapting or implementing certain inputs or functions.

Flex Fuel Sensor

See the Flex Fuel section for details on hardware and configuration of flex fuel setups.

12V Input Signal

Some position sensors output a 12v signal. To correct this, and avoid damaging the Arduino, a circuit like the one in the diagram can be constructed. The resistor R1 is not always required, but will make sure that any output that is not high is pulled low. Along with this circuit use the pull-up jumper on the Speeduino. This will effectively change a 0v/12v into a 0v/5v signal.

Many thanks to PSIG for the info and diagram.

GM 7 / 8 pin Distributor Module

The GM 7 /8 pin modules have been used in a wide variety of GM engines from 4 cylinder to V8s (small and big block). The 8 pin distributor was also widely used in marine applications by Indmar, Mer-

cruiser, and others.

GM 7-Pin Module

GM 8-Pin Ignition Module

The 7 and 8 pin modules are functionally equivalent and largely share the same wiring. The 7 pin is used in the large coil-in-cap distributors while the 8 pin is used in the small cap distributors with remote mounted coils. The 8 pin has one additional terminal that provides a sensor ground. Both modules provide coil ground via the metal grommets used to secure them to the distributor.

These modules provide an simple means for computer controlled timing while retaining the distributor. They were designed to be used with throttle body injection and port injection motors and provide automatic coil current limiting (7.5 amps was the GM specification) and automatic dwell control. The can be adapted to other distributor applications that use either variable reluctor or hall type sensors.

Pin Descriptions and connections • "**+**" : Battery voltage from a switched ignition source. Provides the power to operate the module.

• "**C-**" : Ignition coil negative connection.

• "**P & N**" : Positive and Negative of the distributor reluctor. Polarity is important. GM distributor connectors can only be connected one way. For use with other distributors, verify polarity of the reluctor leads.

• "**B**" : Ignition bypass. When cranking, grounding this line bypasses computer control of timing. The timing iscontrolled by the module only. This can be done using the Speeduino Cranking Bypass pin function (see below).

• "**R**" : Reference or tach signal. This outputs a 5 volt square wave that serves as the RPM1 input for the Speeduino. To use this, connect it to RPM1 and set JP2 to Hall and JP4 connected (ie, 5 volt pull up).

• "**E**" : Timing control signal. When pin B has 5 volts on it, the module allows Speeduino to control the timing using this pin. The output of Ign1 should be connected to this pin.

• "**G**": *(8 pin only)* Signal ground. Should be connected to the Speeduino sensor ground. (Module ground is provided through the metal mounting grommets)

A timing bypass circuit must be constructed to utilize the Speeduino ignition timing control. The small circuit below should be built in the proto area.

In addition, the cranking bypass should be turned on and the bypass pin should be set to pin 3 in the Cranking Settings dialog (under Starting / Idle in TunerStudio):

Trigger settings (under Starting / Trigger Setup in TunerStudio) should be as shown below. You will need to adjust the trigger angle to get correct timing. Instructions for this are in the wiki.

Timing control is set in the Spark / Spark Settings dialog:

The module works well, however, some applications result in a noisy trigger signal. This shows as an unstable RPM (either at idle, or more frequently, at higher speeds). Trigger filtering may help, but a modification may be necessary to clean up the signal. The circuit below effectively cleans/filters the signal, allowing use with no trigger filtering by the Speeduino. It has been tested with single and 8-pole reluctors and modules from GM, Transpo and a no-name generic.

Many thanks to apollard for this outline information.

Outputs

Injectors

Speeduino injector drivers use on/off (not PWM) control and are designed to work with “High-Z” injectors. This type of injectors are also known as “saturated” or “high-impedance” that use full battery voltage to control the injector open cycle, and generally the impedance is greater than about 8 Ohms. If you are running “Low-Z” (“peak and hold” or PWM-controlled) injectors that are lower impedance, you will need to install series resistors on these 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.

Speeduino can drive up to 2 High-Z injectors per output channel.

Coils

Current versions of the Speeduino use low-power output signals, designed to work with external small-signal ignition coil drivers, whether a separate type (module or ICM, igniter, IGBT, etc.), or built into the coil assembly (“smart” coils). This method permits Speeduino to have great flexibility to control most types of ignition systems. Attaching the Speeduino outputs directly to a traditional high current passive (“dumb” or 2-pin) ignition coil without an ignition coil driver WILL cause damage to your Arduino.

How Speeduino controls ignition circuit power In prior history, the coil driver was a set of mechanical contact points (“points”), simply replaced today by an electronic version. The added coil driver can be anywhere from inside the Speeduino to inside the coil assembly; though near or in the coil reduces electrical noise.

In the example animated image below, the Speeduino ignition signal is normally low (near ground or 0V) while Speeduino calculates the time to the next coil firing. At the proper time, Speeduino switches the ignition output to high (near 5V or 12V selectable) in order to switch the coil driver (example IGBT) on, allowing current to flow through the coil to ground. This is called the "dwell" period. During dwell an increasing energy field is generated around the ignition coil core and wire windings.

At the end of the dwell period and therefore at the proper time for spark; Speeduino switches the coil driver off, stopping current flow, which collapses the energy field to create high voltage and the resulting spark:

In TunerStudio, the setting for this example would be to fire the spark "*going low*". The dwell setting is highly variable depending on coil type, voltage levels, etc. Too little dwell will give weak or no spark and excess dwell will rapidly increase heat, possibly damaging any of the ignition components, but usually the ignition coil or coil driver.

The wasted-spark version is below to show how it is identical in operation, but with the high-voltage spark returning through the second spark plug to complete the circuit:

A good run down of "smart" coil types can be found at: <http://www.megamanual.com/seq/coils.htm>. There are many ignition modules available that Speeduino can use to control standard coils, or for smart coils you can generally use 4 or 5-pin types as these will always be logic level, although some 3-pin coils are also of this variety. GM LS1/2 coils are an example of powerful smart coils that are commonly used and can usually be obtained easily and cheaply.

(Note: In the past, some ignition control modules with current limiting or dwell control features (e.g., 1970s GM HEI, Bosch "024" types, and Ford DS1) were referred-to as "smart" modules. While still true, common terminology of individual ignition coils with at least a driver integrated, or newer technology with greater controls (e.g., controlled spark duration or multi-spark) are all considered "smart" coils. You must know the control requirements of the specific drivers, control modules, or coils you intend to use in order to operate them properly with Speeduino.)

Aux Outputs

Low Current

Most Speeduino versions have low-current (LC) signal outputs directly from the MEGA processor to (usually) the prototyping (proto) area of the board. These outputs are generally not suited to control power devices directly in this form, and need suitable output circuits built on the proto area to enable their use. Alternatively, the output functions such as Fuel Pump or Fan are re-assignable to other pins and components, such as the existing HC medium-current output MOSFETs.

Some Speeduino versions include an 8-channel ULN2803A Darlington transistor array IC that is capable of switching up to 1/2 amp per channel. These auxiliary outputs are sufficient to switch small devices directly, or to switch power-handling devices, such as power MOSFETs or automotive relays. Configuration and settings of these outputs is described in the Configuration section. Additional information for using a ULN2803A on v0.4-series boards may be found here. Similar output options and pin assignments may be used on other board designs.

Medium Current

v0.3x and later boards include medium-power MOSFET auxiliary outputs to switch up to 3 amps directly. These are labeled "HC" in schematics and some references. These outputs are commonly used to operate idle valves, boost-control valves, VVT solenoids, etc., or to control relays for handling much larger loads, such as electric radiator fans. Configuration and settings of these outputs is described in the Configuration section.

Auxilary IO

On Arduino Mega2560 based Speeduino boards (official or just running current firmware) git commit 13f80e7 support is available for the external connection of 8 16bit analog inputs via Serial3

CanBus

As the Arduino mega2560 has no CanBus interface a seperate "co-processor" interface has been designed. More information about this unit can be found here https://github.com/Autohome2/Speeduino-can-interface. This uses the functionality provided by the Serial3 port and connects via that port.

On the upcoming Teensy3.5 variant of Speeduino the CanBus code will be incorporated into the main system firmware as the Teensy3.5 has integrated CanBus and only requires a transceiver module added.

Third Party Addon's

In This section you will find information about third party designed hardware designed to be used in conjunction with the Speeduino ECU

GPIO for Speeduino

There are several variants of the GPIO , The firmware can be downloaded here https://github.com/Autohome2/Speeduino-GPIO

More information GPIO_for_Speeduino