


- Ford Mustang Instrument Cluster
- 
- +
- Q

Ford Mustang Instrument Cluster Project: Introduction



By Mr.RC-Cam,

January 1, 2019 in Ford Mustang Instrument Cluster

 Share

[Moderation Actions](#) ▼

[Start new topic](#)

[Reply to this topic](#)

1 2 **NEXT** »

Page 1 of 2 ▼



Mr.RC-Cam

Posted January 1, 2019 · IP ▼

The clock struck 2019 last night. So I'm starting our brand new year with another project. And of course that means I'm going to write another workbench adventure story. Hang on to your horse, I think it's going to be a fun ride.

Despite this blog's title, I don't currently own a Ford Mustang. But I had a '66 Mustang when I was a teenager. She was a tired pony with 130K miles when I drove off the used car lot. Nice looking, ran well, and had a sweet 8-track player. Dependable too, but I occasionally had to get my hands greasy. Like the time the transmission blew on the way to work (fixed for \$75 after a visit to the auto junkyard).

Back then I enjoyed the troubleshooting challenges while working on everyone's cars. I rebuilt engines and did just about any auto repair that came up. In those days working on cars was easy; A determined guy with wrenches and a floor jack could fix anything.

Being able to fix clunkers was an advantage to a cash-strapped kid. It allowed me to purchase a variety of used vehicles to play with. I even had some that were air cooled; That's VW Bug and Chevy Corvair territory. Besides a couple modified Beetles, I had a Greenbrier 6-door van and two Monzas (sedan & coupe).

Then there's that time I built a "kit" car. Not really a kit, but essentially a pile of fiberglass body parts that were assembled by guesswork and ingenuity. Running gear was decided by the builder. Mine ended up on a VW Squareback pan (Type 3 chassis) with a bored out 1500cc engine. The body's T-top styling looked good but the stock suspension gave it a stiff ride. See photo below.



My home built kit car. Late

1970's.

Then the 1980's arrived and working on American cars became a hideous chore. Horrible mechanical reliability, bad paint, fragile plastic parts, and ungodly smog systems that ruined engine performance. At this point I was busy working in the electronics industry and my interests morphed from motors to microprocessors. And higher wages meant I could finally buy better cars.



That's me on a typical

afternoon when grease was still my friend. Circa 1979.

Fast forward a few decades. Now I'm an occasional weekend mechanic that drives a 20+ year old Ford SUV. Bought it new (factory order) and over the years it has been a reliable friend. Other than DiY oil changes and some minor repairs, I really haven't needed to bust my knuckles. And I like that, because my love affair with grease ended long ago.

Why would I tell you all this? Well, it's the introduction to the story on how a Mustang instrument cluster became the centerpiece to my latest electronic project. At the heart of the story is a grown up kid that liked tinkering with cars, lost his passion, then gets reunited by happenstance. So if you are interested in where this is going then hang around the electronic campfire while the story unfolds. The ending is not written yet, it's a real-time adventure.

- Thomas



Mr.RC-Cam

Posted January 1, 2019 · IP ∨ (edited)

Hang in there, I'm nearly finished with the backstory to my Mustang Cluster project. We'll get to the technical stuff in a moment.

A couple weeks ago I noticed that my old SUV's instrument panel had a burned out light bulb; Half the speedometer was dark. In the old days I would reach under the dash and change the bad bulb. But easy access to anything in a car is a thing of the past.

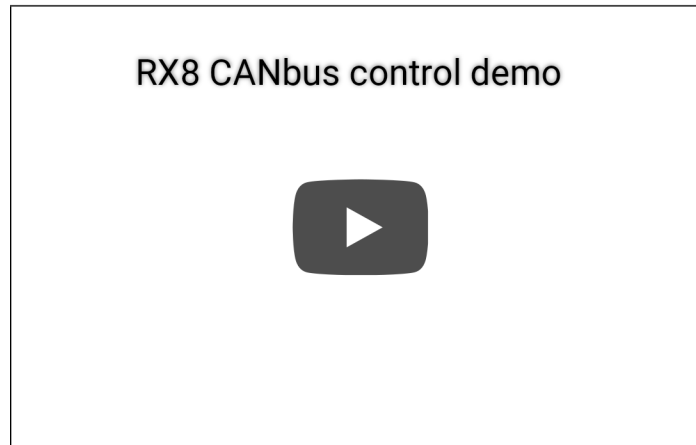
So I did the typical DiY guy thing and went to YouTube. I quickly learned that I have to pull out the instrument cluster. A few screws and some hidden fasteners. Nothing I can't handle.

But then I got trapped in a YouTube rabbit hole. For hours I binge watched instrument cluster repair videos. Blown FETs, failed needle driving stepper motors, bad LED upgrades, and so much more. Fixing these problems is interesting to watch, at least to me.

2nd chance at working again! Bu...



Then I tripped across some DiY hacker videos that show Arduino boards and secrete software controlling automobile instrument clusters. Instantly I was hooked and felt the sudden urge to know more about modern dashboard systems.



The video above shows an Arduino controlling a dashboard cluster. Too bad the project links are dead.

Keep in mind that I grew up with mechanical cable driven speedometers, hot-wire ammeters, and brainless hardwired warning lights. Of course I've known that modern technology has infiltrated the dashboard, but until now I just didn't care.

Intrigued, my adventure begins.

- Thomas



Mr.RC-Cam

Posted January 1, 2019 · IP ∨

Every project deserves a name. This instrument cluster hack has been officially badged the **CAN2Cluster Project**. The name includes a reference to the CAN-Bus interface, something we will soon discuss.

First on the to-do list is to research the communication signals used to control a modern vehicle's instrument cluster. Given the number of car brands out there I expect to find an assortment of proprietary protocols.

This is where you interrupt me to explain that there's an OBD port (On-Board Diagnostics, big connector under the dash) that is interfaced to the Engine Control Unit (ECU). And it will be able to provide every little bit of data that my project would need using a communication protocol that is well documented. You end your

lecture by saying that controlling the instrument cluster is going to be *Easy peasy lemon squeezy*.

Hmm. I like what you're saying but I don't expect it will be that easy. My project won't have an ECU, so the OBD interface is non-existent. Beyond that, it's true that some automakers provide access through the OBD connector to directly control the instrument cluster and vehicle actuators. But this special bi-directional functionality is not designated by the OBD standards (and not all cars support it). So any protocol information gleaned from the OBD documents is not going to be helpful to my project.



Typical OBD port. It's located

under the dashboard on the driver side.

OK, I just returned from researching deeper into this topic with the assistance of Mr. Google. As expected, simple it's not. I've learned that there are several automobile network protocols that are brand dependent. Some are for diagnostics and some are for inter-module communication. Some do both. I don't know if it's a complete list, but the protocol wiki will help paint a clearer picture: [Click Me](#).

I'll ignore the network/protocol choices for now. My immediate goal is to find a late model instrument cluster with gauges and warning indicators that are controlled by the fewest number of wires. A dream solution is a cluster that can control **everything** from a dedicated serial bus. /* Somewhat confident that this probably exists, a typical thing a tech guy tells himself before reality strikes. */

So it's time to pick an instrument cluster for the project. All the YouTube hacker videos I watched showed tachometer and speedometer operation using Mazda and VW dashboards. Although I've owned cars from both of those brands, I prefer something different for this project.

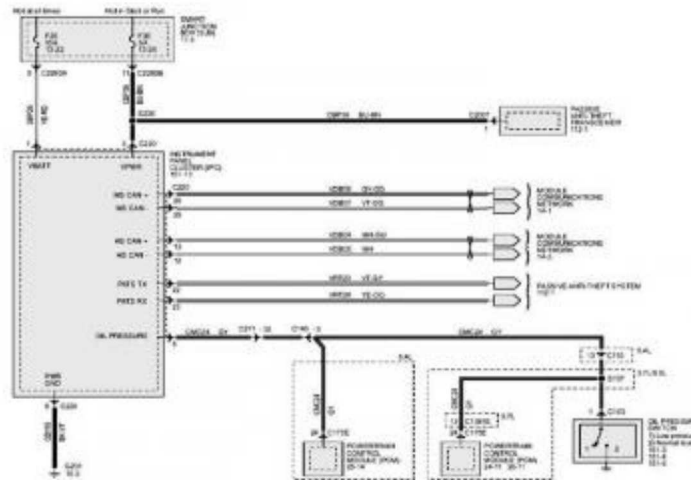
My heart is set on controlling the gauges from a Ford Mustang dashboard. Perhaps a working instrument cluster, displayed on my office bookcase, will comfort me with memories of the good times I had driving my old pony. And since auto junkyards are full of dead Mustangs, getting a donor instrument cluster will be the least of my problems.

Mr.RC-Cam

Posted January 1, 2019 · IP ▾

A quick search on eBay finds a nice looking 2002-2004 Mustang cluster. But this one is not a good choice for what I have in mind. It uses Ford's obsolete SCP (Standard Corporate Protocol) bus for communication. The fuel / oil pressure gauges are hard wired to their senders and many of the indicator lamps are also hard wired. And its two harness connectors have a couple dozen wires ([Click for Pin-out](#)). This means the Arduino will need a lot of digital I/O pins to control everything. All told, I'm not a fan of this choice.

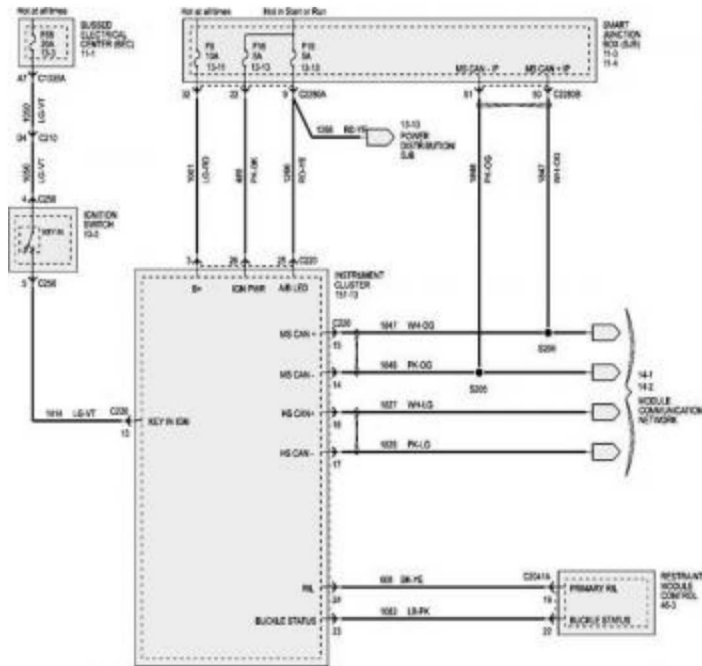
The next eBay prize to catch my attention is a 2012 Mustang cluster. This version uses CAN-Bus for communication with all indicator lamps and gauges appearing on its serial bus. But a key coded PATS Anti-theft module (not included) is directly connected to the cluster. I don't know if this missing module will disable some of the cluster's features. Keep looking.



2011-2014 Mustang Cluster

Wiring Diagram

But like Goldilocks, I come across a third choice that seems to be just right. It's a 2007-2009 series Mustang Cluster that uses CAN-Bus communication. But on this model year the PATS anti-theft module does not directly connect to the cluster. Maybe this is a good sign it will be less trouble to interface.



2005-2009 Mustang Cluster

Wiring Diagram

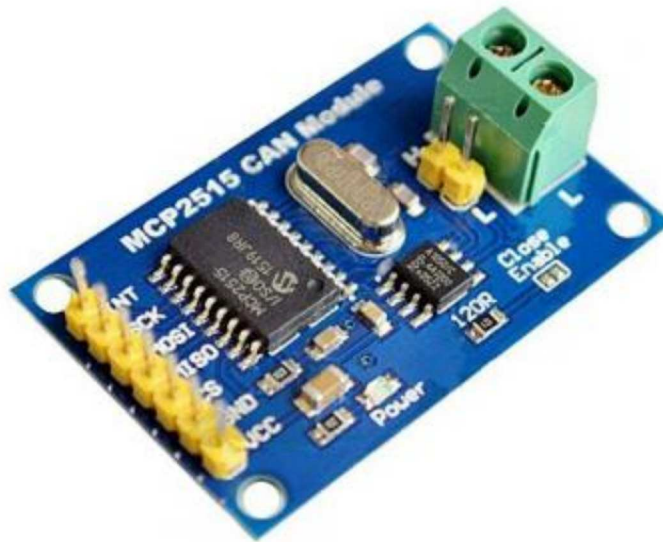
So a gut feeling tells me this is the one to hack.



This 2007-2009 Mustang instrument cluster

is on its way to me, thanks to an eBay auto wrecker.

Next I ordered the CAN-Bus board for an Arduino MEGA2560 R3 that's been sitting on my workbench. See image below.



I also ordered a [Microchip APGDT002 CAN-Bus Analyzer Tool](#) (see image below). This USB connected dongle should provide a convenient way to experiment with automobile CAN-Bus protocols. I could have bought a similar CAN-Bus tool from a eBay seller at a fraction of the cost. But I've lost confidence in the functionality of cheap China sourced tools. So I spent a bit more to avoid unexpected surprises.



Keep in mind that I can't use a common OBD diagnostic scanner for the things I want to do. That tool is a beast with another purpose. That is to say, I'm not going to read [OBD PIDs](#) to troubleshoot a Check Engine light. Instead I need to get close and personal with the cluster's back panel CAN-Bus signals and read/write some mystery data.

While all this hardware makes its way to me I'll continue my search for the instrument cluster's CAN-Bus Arbitration ID (ArbID) list and payload parameters. I understand that these Ford secrets are not officially released to the public. But I suspect I will find useful details with my friend Mr. Google.

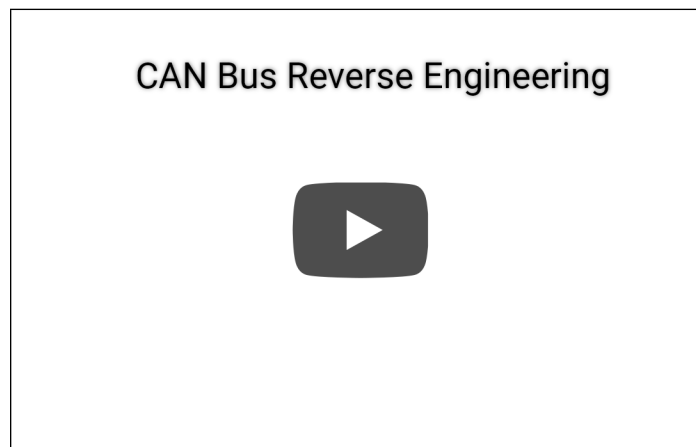
Mr.RC-Cam

Posted January 2, 2019 · IP ▾

After a long night of web mining I'm now confident I'll be able to control the Mustang's Tachometer (RPM) and Speedometer. But everything else is still a mystery.

What I unearthed is courtesy of like-minded hackers that used reverse engineering tricks. That is to say, they connected CAN-Bus sniffers to their cars and drove around while logging the real-time data. Back at home, they reviewed the timestamped activity while looking for clues on which data packets involved the dashboard.

The reverse engineering task is best described as finding a needle in a haystack. But choosing the right tools eliminates a lot of drudgery.



I'd like to use the sniff method too, but I don't own a late model Ford with CAN-Bus. The car in my garage was built years before this networking protocol was a Ford staple.

Fortunately I found the CAN-Bus ArbIDs to the Ford Tach / Speedometer. See this sourceforge wiki:

<https://sourceforge.net/p/ecu/wiki/canbus/>

But the search for controlling Ford's indicator lights and gauges has hit a dead end. However, during the online investigations I noticed that Mazda partially shares some of Ford's CAN-Bus arbitration codes (these two car makers have been exchanging technology). I expanded my search and found OpenGarage's list of Mazda CAN-Bus ArbID descriptors and I plan to try them:

http://opengarages.org/index.php/Mazda_CAN_ID

And to make things a bit more interesting, the Mustang Instrument Cluster I

selected has two separate CAN-Bus ports. The HS-CAN (High Speed CAN, 500kbps) port is for the engine and drivetrain data. It provides the RPM, speed, coolant temperature, and similar data. The MS-CAN (Medium Speed CAN, 125kbps) is for ancillary information, such as fuel level, oil pressure, seat belts, parking brake, doors, etc.

To fully control the cluster I'll probably need to send data to both ports. But I'm hoping that MS-CAN data is allowed to tunnel into the HS-CAN bus so that only one Arduino CAN-Bus shield is needed. /* I'll order a second shield in case my wishful thinking is cruelly trampled by reality. */

- Thomas



Mr.RC-Cam

Posted January 4, 2019 · IP ▾

The Microchip CAN-Bus Analyzer dongle arrived. It's always a pleasure to be visited by the big brown truck.

I ordered the CAN-Bus tool directly from Microchip's official online store to ensure it was the latest release. Wishful thinking at best; In the box was a CD containing old V2.0 software (2011 release). So the CAN-Bus dongle hardware is probably aged like a fine wine too.



I read that V2.0 wasn't compatible with Windows 10. So I downloaded the V2.3 Win10 upgrade from the Microchip web site and installed it. The app launched and I was able to setup the CAN-Bus tool. I currently don't have any Can-Bus devices to test out, but that doesn't stop me from investigating the tool's GUI.

Wait a minute, the GUI's features look like someone half-hardheartedly phoned it in. It has basic functionality, but it's missing important features mentioned in the manual. For example, I can't save my tool configuration settings and there's no Trace Filter.

And it's buggy too. For example, moving a trace window creates graphic ghosts that require a program exit to clear up.

I found a possible explanation to these problems. The help screen reports that the PC Software is a BETA V2.2 released in 2014. That's not a good sign because the installed version is supposed to be V2.3 released in 2016. Dooh! */* Guy shakes head in frustration and mumbles to himself. */*



But there's another possible reason for the missing functions. The manual states when upgrading the PC software it is necessary to re-flash the tool's two PIC18F microprocessor chips using the provided hex files.

2.1 INTRODUCTION

The following chapter describes the procedures for installing the CAN Analyzer hardware and software.

The chapter contains the following information:

- Installing CAN BUS Analyzer Software
- Installing CAN BUS Analyzer Hardware

2.2 SOFTWARE INSTALLATION

2.2.1 Installing the GUI

Install .NET framework version 3.5 before installing the CAN BUS Analyzer.

1. Run "CANAnalyzer_verXYZ.exe", where "XYZ" is the version number of the software. By default this will install the files to: C:\Program Files\Microchip Technology Inc\CANAnalyzer_verXYZ
2. Run the setup.exe from folder: C:\Program Files\Microchip Technology Inc\CANAnalyzer_verXYZ\GUI
3. The setup will create a shortcut in the Programs Menu under "Microchip Technology Inc" as Microchip CAN Tool ver XYZ.
4. If the CAN BUS Analyzer PC software is being upgraded to a newer version, the firmware should be updated to match the revision level of the PC software. When updating the firmware, ensure that the hex files are programmed into their respective PIC18F microcontrollers on the CAN BUS Analyzer hardware.

2.2.2 Upgrading the Firmware

If upgrading the firmware in the CAN BUS Analyzer, the user will need to import the HEX files into MBLAB[®] IDE and program the PIC[®] MCUs. When programming the PIC18F2680, the user may power the CAN BUS Analyzer by external power supply or by the mini USB cable. When programming the PIC18F550, the user needs to power the CAN BUS Analyzer by external power supply. Additionally, when programming HEX files into PIC MCUs, it is recommended to check the firmware version from the GUI. This can be done by clicking on the "Help > About" menu option.

The GUI's reported firmware version numbers match the two hex file names found in the downloaded software. It appears the dongle's chips are already the latest version. But I don't trust this Microchip tool, so out comes my [MPLAB ICD3 ISP Debugger/programmer](#). With a 12V supply connected to the CAN-Bus tool I successfully flashed both PIC chips with the provided hex files.

But the firmware flashing was a wasted effort. The GUI's reported firmware version information is the same as before and the missing features are still missing. Pardon me while I take a break and try to sort this out.

There's closure to this mess. I found the Release Notes file in the upgrade distribution. It confirms I have the latest PC GUI software (V2.2). It also mentions that those missing features I wanted were dropped years ago in the V2.0 release. So some functionality is now vaporware and any bugs that appear are here to stay. I've concluded that this tool is an abandoned Microchip orphan. */* More mumbling and head banging. */*

If this were a product review it would be generous to give Microchip's tool a one star rating. Hopefully my opinion gets upgraded after I've done some basic CAN-Bus experiments with the Mustang instrument cluster.

The Mustang Instrument Cluster has arrived. It came from a wrecked car (eBay based recycler). There's some scuff marks but overall it looks good.



Every part from a wrecking yard has a story. The cluster was tagged with the VIN and an online search provides a wealth of information. It was a 2009 Mustang 4.0L V6 coupe with 60K miles that had gotten sandwiched in a roadway skirmish. If by chance you were the owner of this pony when it took its last breath then please accept my condolences. And I hope everyone involved in the accident is OK.



The 2009 Instrument cluster

came from a pretty pony that has been put out to pasture.

One of the reasons this particular cluster was chosen was because the seller's photos showed it included the harness connector plug (chopped off, with several inches of wire). I even email them before the purchase and they confirmed the connector would be included. I'm happy with the condition of the cluster I received, but the connector is missing. */* More head banging followed by silent screams. */*

Without the connector plug I'm at a disadvantage. Besides the obvious reasons for wanting a factory made plug, the colored wires on it would have helped me determine the pin numbering (connector orientation).

It's best to roll with the punches, so after a few minutes with an ohmmeter I have what I need. Connector orientation was determined by identifying the ground pins and matching them to a [wiring list](#) found online.



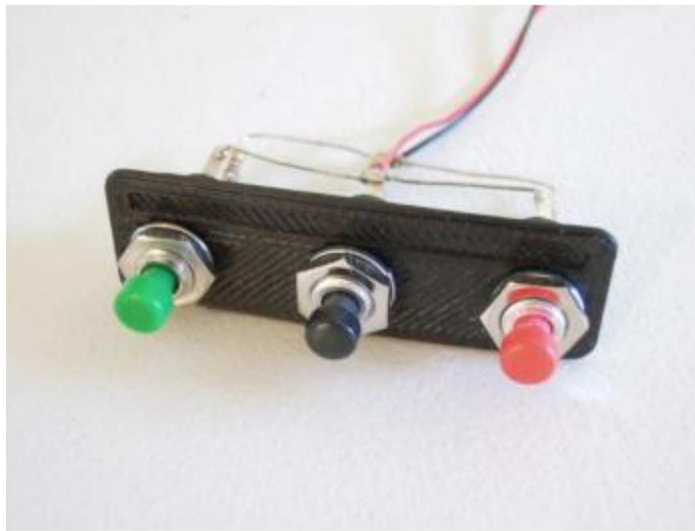
Rear panel photo: Connector

Pin numbering (plug orientation).

Some clip-on jumper wires and a variable power supply provides good news. The cluster powers up and the six gauges successfully perform their needle calibration dance. Its voltmeter gauge is centered with the supply voltage set to 12.0V, but the other gauges are zero (as expected). All the indicators light up and backlighting is working too. The warning buzzer chirps as well. The donor organ is alive.

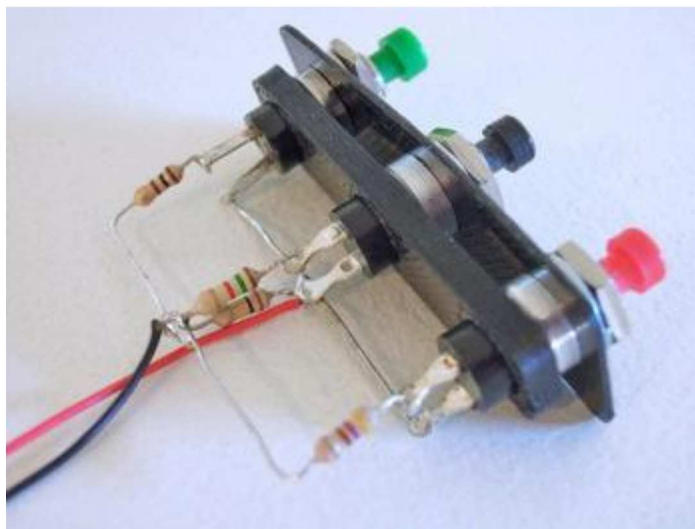
There's a pair of wires (Pins 6 & 7) that need a 3-button Message Center switch. The switch is expensive so it's DiY time. A couple hours of experimentation determines that the switch is a resistor ladder (voltage divider) for the cluster's analog input on pin 6. The resistor values are as follows:

Info: 100 Ohms
Setup: 1.5K Ohms
Reset: 470 Ohms
Default: 100K Ohms



DiY Message Center Switch,

3D Printed Bezel



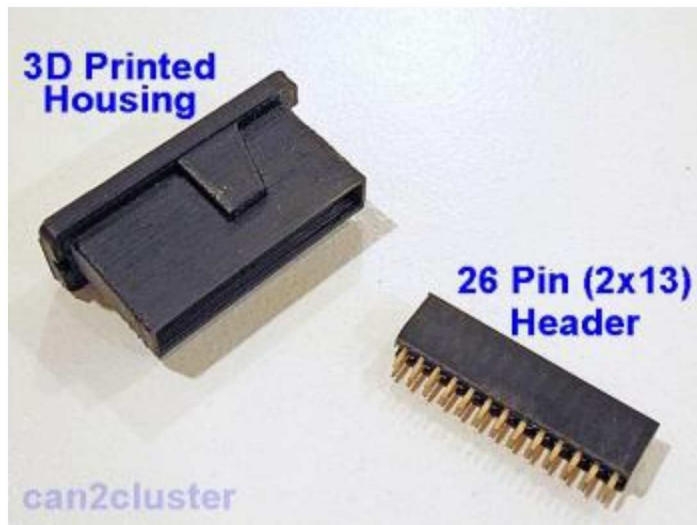
Green = Info, Black = Setup,

Red = Reset

The DiY Message Center switch is wired to the cluster and working great. I have to admit I'm not thrilled by the push-buttons; I'll swap them out with higher quality switches later on.

The CAN-Bus is the next item to test. But rather than stuff more jumper wires on the naked connector I've decided to step back and solve the missing plug dilemma. Instead of complaining to the seller about the forgotten plug I decided to build my own.

The cluster's connector is a 26-Pin (2 × 13) 2.54mm header. I have a female header that is 40-Pins and a close encounter with a sharp saw turns it into a suitable doppelganger. A test fit determines that the plug is difficult to install/remove in the cluster's deep opening. So I designed a 3D printed housing shell to make it an easier effort. The housing is keyed to prevent backwards plug installation.



{A few hours later} All wires are soldered and safely potted in the 3D printed housing with hot melt adhesive. The DIY Cluster plug is complete.



Power/Signal plug is a 26-pin header modified to fit. See text.

There's two cables terminated to the plug. One has twisted pairs for the CAN-Bus (HS CAN and MS CAN), plus some miscellaneous signals. The other has power and all the remaining signals. I won't use all the wired signals, but just in case they're all accessible.

It's time to connect the Microchip CAN-Bus analyzer.

- Thomas



Mr.RC-Cam

Posted January 8, 2019 · IP ▾

It's been a good day. CAN-Bus is working and I can control the tachometer and speedometer.

There's something satisfying about driving 80mph while sitting at my desk. And I have photos to prove I'm exceeding the office speed limit.



80 mph @ 4400 rpm. Oops,

left the parking brake on.

Despite some detours the project is moving along nicely. But there's a long ways to go. It's time to shift into overdrive and find out how to control everything else in the dashboard cluster.

After I conquer all those mystery bits I can get to building what I have in mind. This Mustang cluster is going to become a working bookcase display with a key start ignition.

- Thomas



Mr.RC-Cam

Posted January 9, 2019 · IP ▾

After several hours of experimental CAN-Bus data injection I have identified a few more ArbIDs. Besides rpm and speed, I can now actuate the temperature gauge.

Several indicator lights are under my control too, as follows: Overdrive Off (orange), Overheat (red), Check Engine (orange), Charge System Fault (red), Powertrain Fault (orange), Cruise Control Enabled (green), Security Enabled (red).

These items are handled by two HS CAN-Bus ArbIDs (0x201 & 0x420). As follows:

ArbID 0x201, with 8 byte payload

The packet takes the form: [RR, rr, 00, 00, SS, ss, 00, 00]

Where RRrr is the tachometer rpm and SSss is the Speed mph.

The following formulas are used:

$$\text{rpm} = 0.25 * (\text{RRrr}) - 24$$

$$\text{Speed (mph)} = 0.0065 * (\text{SSss}) - 67$$

Byte 0 & 1 = Tachometer rpm. See formula above.

Byte 2 & 3 = Unknown. Set to zero.

Byte 4 & 5 = Speed. See formula above.

Byte 6 & 7 = Unknown. Set to zero.

ArbID 0x420, with 8 byte payload

Byte 0, Temperature Gauge:

0x55 = LOWEST Temp, 0 line

0x7F = Middle Temp

0xA0 = High Temp (top mark)

0xA1 = Max Temp (red line) with Red warning symbol (Below Tach)

Byte 1, Unknown.

Byte 2, Unknown.

Byte 3, Unknown.

Byte 4, Indicators and Temp Gauge Override:

Bit 0, Unknown.

Bit 1, Unknown.

Bit 2, 1 = Orange O/D OFF Indicator (Below Tachometer).

Bit 3, 1 = Orange O/D OFF Indicator Blinking (Bit D2 must be 1)

Bit 4, 1 = Force Max Temperature (red-line gauge), no warn:

Bit 5, 1 = Force Max Temperature (red-line gauge), no warn:

Bit 6, 1 = Orange Check Engine (Below Tach).

Bit 7, 1 = Orange Check Engine Blinking (Bit D6 must be ze:

Byte 5, Indicators:

Bit 0, Unknown.

Bit 1, Unknown.

Bit 2, Unknown.

Bit 3, 1= Red Charge System Fault Indicator (Below Tach)

Bit 4, Unknown.

Bit 5, Unknown.

Bit 6, Unknown.

Bit 7, 1= Orange Power Train Fault Indicator (Near Tach's r

Byte 6, Indicators:

Bit 0, Unknown.

Bit 1, Unknown.

Bit 2, Unknown.

```
Bit 3, 1= Green Cruise Control Indicator (below Temp Gauge).  
Bit 4, 1= Red Security Indicator, (Below Tach). Bit 5 must  
Bit 5, 1= Flashing Security Indicator (Below Tach). Bit 4 r  
Bit 6, Unknown.  
Bit 7, Unknown.  
Byte 7, Unknown.
```

I haven't been able to find the magic bits for the fuel and oil pressure gauges. Or for an assortment of indicators such as the hand brake, turn signals, and fluid levels. To control them I suspect I'll need a second CAN-Bus interface connected to the cluster's MS CAN-bus port. But I haven't given up on finding a way to do it through the HS CAN-Bus.

- Thomas



Mr.RC-Cam

Posted January 11, 2019 · IP ∨

After two late night sessions there has been more progress. The ArbIDs to the Fuel and Oil gauges have been identified. Plus those that control the door status and tire pressure monitor. I should mention that these functions are not accessible from the HS CAN-Bus. Instead, they use the companion MS CAN-Bus for communication.

Unfortunately none of the online posted Ford or Mazda CAN-Bus hacker information helped find them. It was a painful exercise of manual keyboard entry and patiently watching the cluster's reaction (if any). There's over 2000 possible ArbIDs and their data payloads have some bit field dependencies that enable/disable other functions. The proverbial Needle in a Haystack.

The Fuel gauge was particularly fussy to identify since the control data has weird split scaling and an unexpected control field. To make things more cruel, it would secretly go dormant for 35 seconds on some written data values. I pulled out a lot of hair while deciphering the magic data that it wanted. But I won the battle and can now fill up the tank.

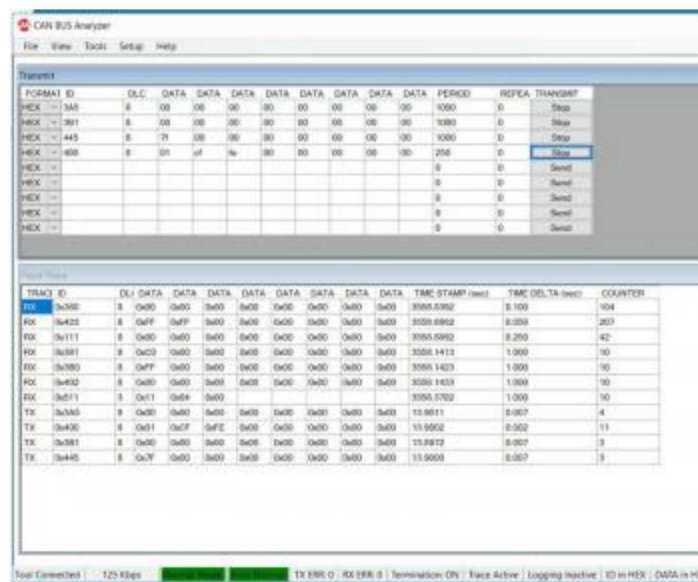
The Oil Pressure gauge is interesting too. It's an idiot light in the disguise of a linear display. A single bit controls it, so there's only one active position. This makes sense since the engine's oil pressure sensor is just a brainless on/off switch.

Here's the ArbIDs to the newly discovered MS CAN-Bus (MS-CAN) items:
0x3A5: Tire Pressure Monitor (TPM)

- 0x3B1: Door Status
- 0x400: Fuel Gauge
- 0x445: Oil Pressure idiot Gauge

Without a real Ford Mustang's data to sniff, my technique was brute force. I used the Microchip CAN-Bus analyzer and manually entered experimental ArbIDs and 8 byte data payloads. The data fields would be populated with test bytes (0x00, 0xFF, 0xAA, or 0x55). Then I would watch the cluster for several seconds to see if anything changed. This went on for about 15 hours, with occasional bathroom breaks.

Below is a screenshot of the Microchip tool's GUI. The four ArbIDs have been configured for 1/2 tank of gas, valid oil pressure, all doors closed, and good tire pressure.



There's a handful of remaining ArbID's to find. Such as:

- Dashboard Backlight Intensity
- Turn Signals (Indicator)
- Handbrake (Indicator)
- Parking Brake On warning (message center)
- Fuel Level Low warning (message center)
- DTE Data ODO Data Error (message center)
- Brake Fluid Level Low warning (message center)
- Check Charging System (message center)

I'm a bit worn out from the tedious ArbID mining. Going forward I think it's best to wait for the Arduino CAN-Bus module to arrive. I can use it with some custom code that will assist me with the search for the remaining ArbID's. But the board is coming from China and I don't expect to see it for a couple weeks.

In the meantime I'll build a nice looking stand for the Mustang instrument cluster. Woodworking tools won't get me greasy, so I'm good.

- Thomas



Mr.RC-Cam

Posted January 14, 2019 · IP ▾

Rather than sit idle waiting for the Arduino CAN-Bus shields to arrive, I shifted over to woodworking tools.

The instrument cluster will be a functioning bookcase display piece. It'll have a hollow oak base to hide the electronic parts. The instrument cluster will get a plywood "dashboard" that will either be wrapped in vinyl carbon fiber film or painted black.



Just getting started. Oak

base frame and plywood dashboard bezel.



Oak base is assembled (test

fit). 1/2" threaded rod risers with 3D printed brackets.



Vintage '66 Mustang emblem

badges will be installed on the base.



Wood finish is Medium Oak

Oil Stain and Polyurethane.



There's plenty of room inside

for the electronic goodies.

The base is finished and looks nice. I put it aside for now since there's some painting to do on the 3D printed plastic parts.

BTW, the late 1960's through 1970's was an era of faux wood grain car trim. I owned a '73 Ranchero that was factory wrapped in that stuff. The cluster's oak base is a playful tribute to Detroit's past attempts at wood grain fakery.

- Thomas



Mr.RC-Cam

Posted January 17, 2019 · IP ∨

There will be a pair of 4-inch car speakers under the instrument cluster. An Arduino MP3 player with 30W audio power amp will fill the room with the lovely sounds of a revving V8. Yes, it's become obvious that my sanity level doesn't have all four wheels on the ground.

The speaker kit includes nice looking grills. But they need enclosures too. No problem, I'll DiY my own. At times like this it's nice to have a 3D printer for making custom plastic parts. I use 123D Design; It's an Autodesk CAD program that has sadly gone extinct (but I'm still a fan).



DiY 3D printed speaker

enclosure. There will be two speakers for the V8 motor.

The 3D printed ABS plastic parts need sanding, priming, and painting. The final color will be satin black.



{A bit later, after eating some dust and inhaling my quota of paint fumes ... }



The rattle can painted parts

look fantastic.

A lot of sanding and filler primer has upgraded the rough 3D printed surfaces to a near factory made appearance.



The plywood dashboard also

looks good in satin black.

All the fabricated parts for the display stand are ready for assembly. But none of that matters at the moment. What I really need is to get busy writing the Arduino code to control the CAN-Bus hardware that is coming from the Far East. Patience, young grasshopper.

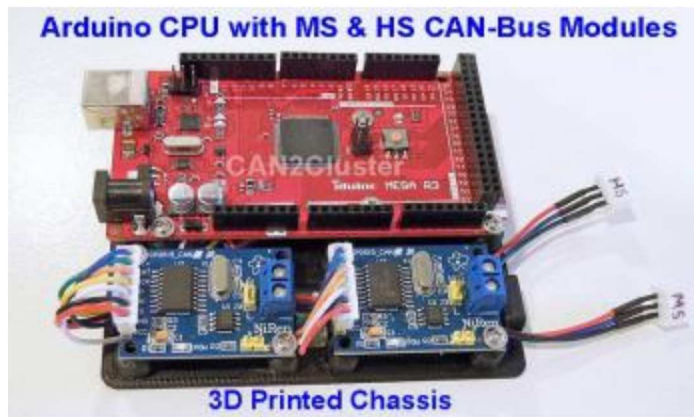
- Thomas



Mr.RC-Cam

Posted January 21, 2019 · IP ▾

The CAN-Bus modules showed up in the daily mail. They were quickly wired to the [Arduino MEGA 2560 R3](#). All three boards are mounted on a DiY 3D printed plastic chassis.



Their arrival was perfect timing: It was a rainy weekend so I stayed inside and wrote Arduino software that helped me find for the missing ArbIDs. The coding effort was a success.

Here's the ArbIDs to the newly discovered MS CAN-Bus (MS-CAN) items:

0x10A: Headlight Control & Backlight Intensity.

0x383: Turn Signal Indicators.

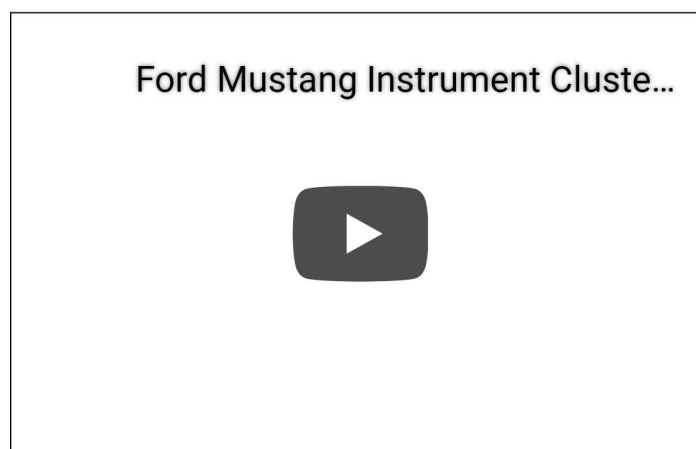
0x3B3: Warning Beeper.

0x3B8: High Beam Indicator.

0x3C1: Parking Brake Indicator, Low Brake Fluid Warning.

Having all the ArbIDs was my E-ticket to building an interactive Instrument Cluster control program. Now I can enter short commands and instantly control any gauge or indicator light. For example, to set the Tachometer's RPM to 2500 I simply type **TR, 2500** into a serial terminal window running on my PC. The serial monitor available in the Arduino IDE is convenient, but any serial terminal program can be used.

Here's a short video that shows the test program in action. As you can see, my wild pony has been tamed.



Now it's time to start mounting the hardware in the oak base and write the animation code for the project's bookcase display. But some important parts

(ignition key switch, Arduino MP3 player, IR Remote, etc.) are on a slow boat from the Far East. I need these things to finish the project. It seems waiting for parts is a right of passage for every one of my projects.

- Thomas



Mr.RC-Cam

Posted January 21, 2019 · IP ▾

Despite all the good news, there's one nagging issue. The instrument cluster has a periodic warning beep and "ODOMETER DATA ERROR" message in the Message Center Display. And instead of mileage it displays "Error mi Error" (bad odometer and trip meter data).



ODOMETER DATA ERROR

appears every 10 minutes.



Odometer and Trip Meter

show **Error** instead of mileage.

Google provides some useful information. The Instrument Cluster has EEPROM storage for the odometer data. The mileage should appear on power-up but I only see the odometer error. I wondered if it needed something from the CAN-Bus so I

spent several futile hours in CAN-Bus purgatory trying to find a solution. No luck, despite threats and begging.

But more digging with Google leads to some bad news. The cluster has a secret self-test feature (button press sequence) and it detects the culprit: **DTC A143** error. That's an *Odometer NVM Memory Failure*. The error is caused by a corrupted EEPROM chip or a circuitry problem related to reading the data.

I disassembled the cluster and inspected the PCB. Everything looks clean and wholesome. I did not see an EEPROM Chip so I suspect the odometer data is stored inside the main microcontroller. Fixing this problem will probably involve a miracle or a deal with the devil.

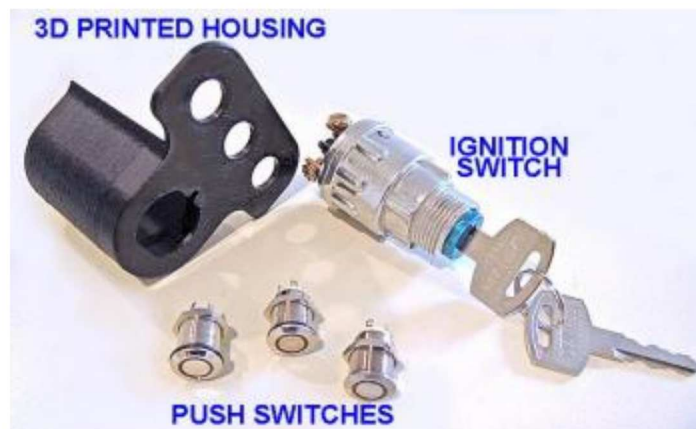
- Thomas



Mr.RC-Cam

Posted January 28, 2019 · IP ▾

The ignition key switch is here. A DiY 3D printed plastic housing combines it with some nice looking push-switches for upgrading the Message Center Switch console.



All the new switches are

ready for the 3D printed ABS plastic housing.



The housing needs

sanding/paint before it's mounted on the dashboard.

- Thomas



Mr.RC-Cam

Posted January 31, 2019 · IP ▾

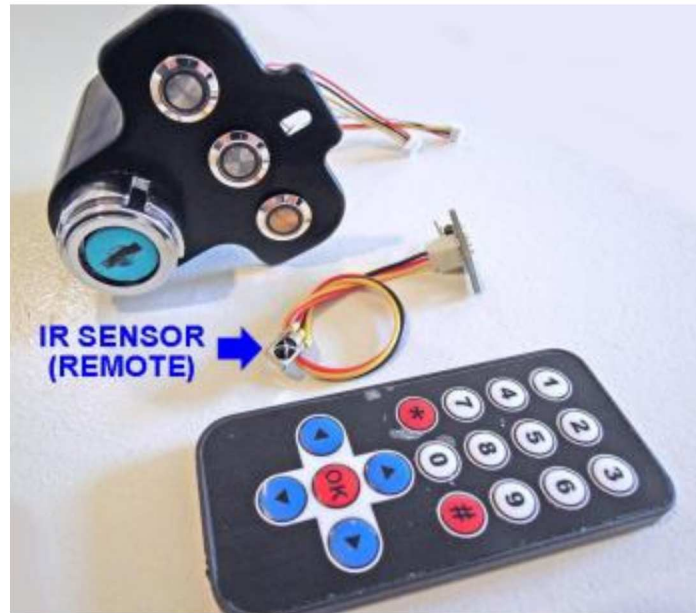
More parts have arrived. Not everything, but enough to put me back to work for a couple days.

The IR Remote was wired to the Arduino and sprinkled with some sweet software. Now I can operate several functions from the handheld controller. As mentioned before, the instrument cluster will be a bookcase exhibit (nerd art); The remote provides another way to turn it on, rev the engine, and do other amusing activities.

Besides the Ignition Key, the handheld remote can start the imaginary motor and animate the cluster gauges. The animations are table based and have 10Hz updates. So a one minute "drive" uses a table with 600 data sets. The advantage of this method is that adding new animations doesn't require rewriting code. Only the table data (gauge values stored in an array) needs to be updated.

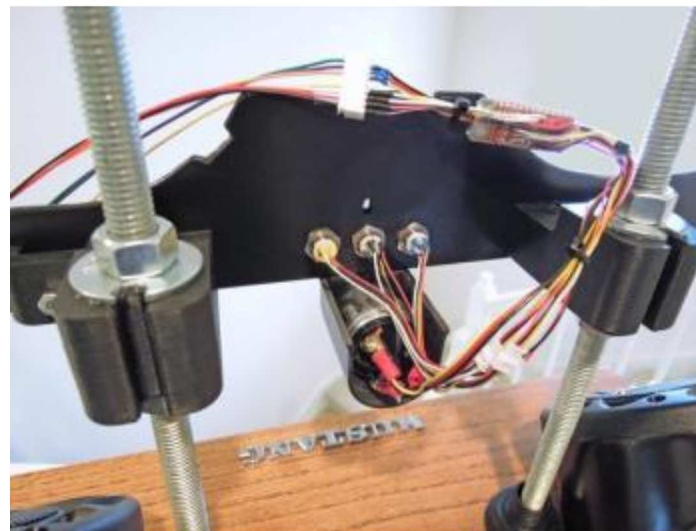
The IR Remote Receiver's PCB was removed from the sensor and replaced with a

directly soldered 3-wire cable. This allows the sensor to fit at the top of the Ignition Switch / Message Center console. So the 3D printed Switch Housing was updated with a window opening for the IR sensor.



Revised housing and

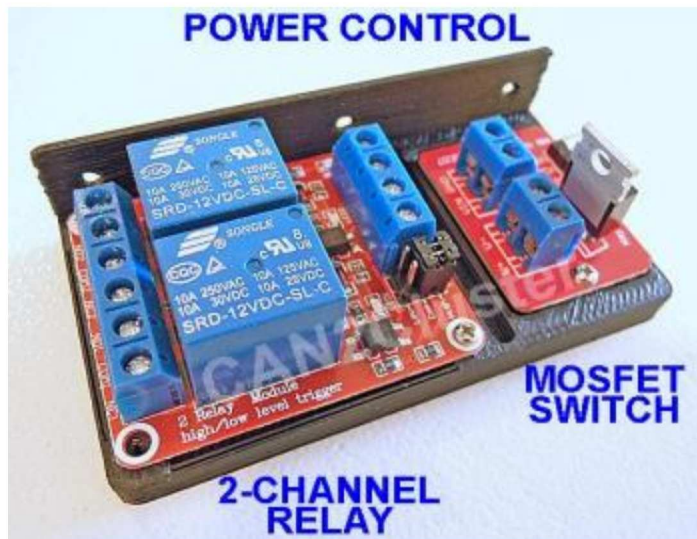
modified IR Sensor (extended from PCB).



Dashboard rear view, Ignition

/ Message Center Switch with preliminary wiring.

Some Arduino compatible Relay and a MOSFET modules are used for programmable Power Control. Instrument Cluster and Amplifier power uses a 2-Channel relay module. The MOSFET controls the light intensity in the Message Center's push button switches. The two modules are mounted on a DiY 3D printed chassis.



- Thomas



2 weeks later...

Mr.RC-Cam

Posted February 8, 2019 · IP ▾

The Arduino compatible MP3 Audio Player I ordered five weeks ago (from China) is still not here. So I gave up waiting for it and ordered the same item from Amazon. That was on Wednesday and it arrived the next day. So it wasn't long before I was hearing delightful audio from the clever little module.

It's the DFPlayer Mini, a 16 pin DIP module that uses micro SD cards. It was soldered onto a small protoboard and complemented with some power filter caps. After a bit of experimentation I found that transformer coupled audio greatly improved the sound quality. So a miniature 600Ω:600Ω audio Xfmr was installed too. It sounds very good with the 2×15 Watt stereo amplifier module driving a pair of 4-inch car speakers.



The DFPlayer MP3 Module is

mounted on protoboard.

There's an Arduino library for it, so incorporating audio capability was painless. Besides adding the software functions, several 16-bit audio files were produced in Audacity that give this bookcase exhibit a bit of character. There's attention getting V8 engine revs and fast driving audio clips. Plus an ambitious car horn for making sure pedestrians get out of the way when I recklessly "drive" around the office.

The MP3 player module was the last item I needed to finish all the wiring. I'm happy to report that all the hardware assembly is complete.



The bottom view. All modules

are wired and working.

There's a bit more software to write. But this Mustang cluster is getting close to the finish line.

- Thomas



It was a productive weekend; The software is done and everything works. Except for that frustrating odometer data error problem I reported last month. It has me stumped.

Time to reflect. Six weeks ago I was a wet behind the ears CAN-Bus wanna-bee developer. I still have a lot to learn but now I know something about the inner workings of a modern instrument cluster.

And I'm having a blast driving my new faux ride in the office. Zero to sixty in six seconds, all from my office chair. And you can tell by the deep exhaust sound that this imaginary Mustang had its factory V6 engine swapped with a V8. Engine swaps are easy in the virtual world.

Here's some photos of the finished display.

