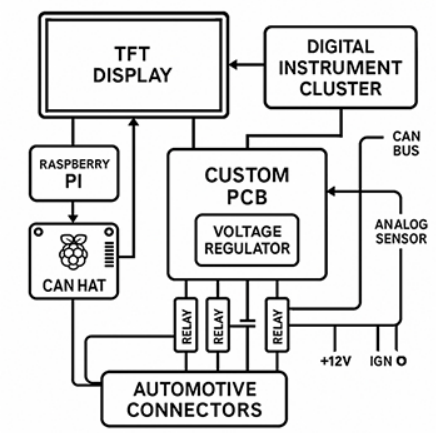
ENGINEERING SPECIFICATION DOCUMENT

Digital Instrument Cluster Development Program
Universal TFT-Based Cluster Platform



Prepared By:
Kevin Caldwell

Document Version: 1.0 Draft
Last Updated: November 18, 2025
Project Status: In Progress

Confidential – For Personal/Technical Use Only
Not for Distribution Without Permission

Contents

1 Digital Instrument Cluster Development Program

This engineering specification outlines the full life cycle of a universal digital instrument cluster platform: concept, system architecture, prototyping, software development, electrical integration, mechanical packaging, validation, and production planning.

Originally engineered around a BMW E90 335i donor car, the platform is currently being adapted for a Honda Civic EF track build. The core philosophy is to design **once** and then reuse the same electronics, software, and PCB across multiple vehicles, changing only wiring, CAN definitions, and enclosure geometry. In other words, the project targets a **universal** instrument cluster that can be dropped into almost any car with a repeatable process.

1.1 Design Goals

- Replace aging analog gauge clusters with a modern TFT display.
- Provide accurate, low-latency readings for RPM, speed, temperatures, pressures, and warnings.
- Maintain or exceed OEM reliability for a track environment (heat, vibration, electrical noise).
- Be serviceable and modular: one electronics platform, many car-specific harnesses and brackets.
- Support future features such as OBD-II diagnostics, code clearing, logging, and GPS lap timing.

2 System Overview

The cluster replaces the OEM analog gauge assembly with a high-brightness TFT display driven by a Raspberry Pi and a custom PCB. The PCB is responsible for:

- Interpreting CAN-BUS frames from the vehicle network.
- Converting analog sensor signals into digital values via ADC channels.
- Handling discrete inputs such as turn signals, high beam, and warning lamps.
- Providing clean, regulated power rails to the Raspberry Pi and display.

The Raspberry Pi runs a Python application that:

- Parses incoming CAN frames and ADC readings.
- Applies scaling, filtering, and sanity checks to signals.
- Renders the UI at 60 FPS using PyGame.
- Logs data and, in future revisions, performs OBD-II diagnostics and code clearing.

2.1 Subsystem Breakdown

- **Hardware Electronics:** Raspberry Pi, CAN HAT, TFT display, relays, ADC, custom PCB.
- **Software:** Python-based UI (PyGame), CAN decoding library, OBD-II communication stack.
- **UI/UX:** Photoshop-designed assets, gauge layouts, animations, warning messages.
- **Mechanical:** CAD-modeled enclosure printed to match OEM dimensions and ergonomics.
- **Electrical Integration:** Harness decoding, protection circuits, fusing, grounding strategy.

3 Hardware Subsystem

The hardware subsystem is the physical backbone of the cluster. It includes the compute module, display interface, power electronics, and all connectors that interface with the car.

3.1 Core Components

- **Raspberry Pi 4 or 5 compute module:** Runs the UI, CAN decoding logic, data logging, and future OBD-II and GPS modules.
- **PiCAN2 or PiCAN3 CAN-BUS HAT:** Provides a robust CAN transceiver and isolates the Raspberry Pi from the automotive CAN network.
- **7"–10.1" IPS TFT display:** The main visual interface. Chosen for brightness, viewing angles, and resolution.
- **Custom PCB:** Integrates the CAN transceiver, analog-to-digital conversion, power conditioning, input protection, and connectors for the vehicle harness and display.
- **Buck converters and regulators:** Step 12 V battery power down to stable 5 V and 3.3 V rails for logic, while handling automotive voltage spikes.
- **Relays, MOSFETs, diodes, and fuses:** Provide switching, load control, and protection for both the cluster hardware and the vehicle harness.
- **Automotive-grade wiring and connectors:** Provide reliable connections to ignition, ground, CAN lines, and sensor feeds.

3.2 Power Architecture (Tapping 12 V)

The cluster taps into the vehicle's electrical system at three primary points:

1. **Permanent 12 V (Battery):** Drawn from a fused feed on the vehicle's main distribution panel. This line powers standby circuitry and allows for controlled shutdown.
2. **Switched 12 V (IGN):** Comes alive when the key is in the run position. This signal wakes the Raspberry Pi and display via a relay or MOSFET-controlled power path.
3. **Ground:** Multiple star-ground points are used to tie the PCB ground to the vehicle chassis while minimizing ground loops.

The 12 V feed is first passed through:

- An inline fuse (typically 3–5 A).
- A transient voltage suppressor (TVS diode) to clamp high-voltage spikes.
- A reverse-polarity protection device (diode or ideal MOSFET arrangement).

After protection, the power is sent into buck converters which create stable 5 V and 3.3 V rails for the Raspberry Pi, display, sensors, and logic ICs.

3.3 Signal Conditioning

Automotive environments are noisy. The custom PCB:

- Filters supply rails with bulk capacitors and high-frequency ceramic capacitors.
- Uses RC filters on analog inputs to reduce noise.
- Maintains separate analog and digital ground regions where possible, tied at a single star point.

3.4 Hardware Cost Summary

Component	DIY Cost (CAD)	Labour Cost (CAD)
Raspberry Pi 4/5	\$85–\$140	–
PiCAN2 / PiCAN3 HAT	\$60–\$120	–
TFT Display (7”–10.1”)	\$70–\$150	–
Voltage Regulators	\$15–\$35	–
Custom PCB Manufacture	\$25–\$80	–
PCB Components	\$20–\$40	–
Relays / MOSFETs / Diodes	\$10–\$40	–
Breadboard Materials	\$15–\$25	–
Automotive Connectors	\$10–\$30	–
Wiring & Fuses	\$10–\$40	–
Electronics Assembly Labour	–	\$150–\$300
Section subtotal	**\$300–\$650**	**\$150–\$300**

4 Software Architecture

The software stack is written primarily in Python. It is designed to be modular: the same core codebase runs on every car, with configuration files and themes swapped per platform.

4.1 Software Modules

- **CAN Input Manager:** Opens a CAN interface (via SocketCAN), listens to frames, and converts raw IDs and byte payloads into engineering units (for example, RPM, vehicle speed, coolant temperature).
- **Analog Sensor Manager:** Reads ADC channels from the PCB, applies scaling curves (such as fuel level or temperature sensors), and debounces signals.

- **UI Renderer:** Uses PyGame to draw gauges, numerical values, warning indicators, and menus at 60 FPS. The renderer is resolution-aware so it can support different screen sizes.
- **Warning Logic Engine:** Implements thresholds and hysteresis for oil pressure, coolant temperature, check-engine warnings, and user-defined alerts.
- **OBD-II / Diagnostics (Future):** Will issue OBD-II requests over CAN and interpret responses to display diagnostic trouble codes (DTCs) and allow clearing them directly from the cluster.
- **Data Logging Module:** Records selected channels (RPM, throttle, temperatures, speed, etc.) to files for later analysis, especially for track use.
- **Configuration System:** Loads per-car CAN maps, sensor calibration curves, and UI themes from JSON or YAML files without changing core code.

4.2 Threading and Robustness

The software separates timing-critical tasks (CAN reading and rendering) from slower tasks (logging, OBD-II requests) via:

- Background threads or asynchronous loops for CAN reception and logging.
- A main loop dedicated to rendering and input handling.
- A watchdog timer that can trigger a safe restart if frames stop arriving or if the UI locks up.

4.3 Software Development Costs

Item	DIY Cost (CAD)	Labour Cost (CAD)
Python Libraries	Free	–
Development Tools	\$0–\$40	–
Programming Labour	–	\$300–\$900
Section subtotal	**\$0–\$40**	**\$300–\$900**

5 Universal Vehicle Compatibility

5.1 CAN and Sensor Abstraction

To make the cluster universal:

- Each supported vehicle has a JSON configuration file that lists CAN IDs, byte offsets, scaling factors, and endianness for key signals.
- Analog inputs (for example, fuel level or analog coolant sensors) are described by calibration curves, which can be linear or piecewise.
- A mapping layer converts generic channel names such as `rpm` or `coolant_temp` into whatever signal combination each car uses.

5.2 Mechanical and Harness Abstraction

- For each new platform, a custom harness adapter is designed that plugs into the OEM cluster connector and breaks out signals to the universal PCB pinout.
- The cluster enclosure is modeled in CAD to match the shape of the OEM cluster surround, mounting bosses, and viewing angles.

5.3 Customer Guide Pipeline

For each vehicle platform, the following deliverables are planned:

1. A wiring diagram showing where to tap 12 V, ignition, ground, CAN lines, and any analog feeds.
2. A set of CAD files (STL/STEP) for the mounting bracket and bezel.
3. A configuration file for CAN and analog signals.
4. A short installation guide describing mounting, wiring, and software flashing procedures.

6 Electrical Engineering

The electrical design covers everything between the car and the Raspberry Pi: harness integration, protection, analog measurement, and discrete input handling.

6.1 Harness Integration and 12 V Tapping

- **Power:** The cluster taps 12 V from an existing fused circuit (ideally the original cluster feed). If that is not available, a dedicated fused link is added at the fuse box.
- **Ignition Signal:** The IGN or ACC line triggers a solid-state relay or MOSFET circuit which powers up the Raspberry Pi and display only when the vehicle is on.
- **Ground:** A clean ground point near the original cluster is used to reduce loop area. Heavier gauge wire is used to minimize voltage drop.
- **CAN Lines:** CAN-H and CAN-L are taken from the original cluster connector or from a known CAN junction. The PiCAN HAT connects directly to these lines with appropriate termination.

6.2 Analog-to-Digital Conversion

Many signals are not on CAN and must be read as analog voltages:

- The PCB routes these signals into an external ADC (for example, 12-bit or 16-bit).
- Voltage dividers are used to scale sensors that output up to 12 V down into the ADC range (typically 0–3.3 V or 0–5 V).
- Low-pass RC filters (for example, 1 kΩ and 0.1 μF) smooth fast noise but preserve human-scale dynamics (fuel level, temperature).
- Certain inputs such as fuel level may require non-linear calibration tables because the tank shape is not perfectly linear.

6.3 Discrete Inputs

Signals such as turn indicators, high beams, brake warning, and check engine lamp may be simple switched 12 V lines. These are:

- Passed through resistor dividers and clamping diodes.
- Fed into optocouplers or transistor circuits to interface safely with 3.3 V logic.
- Debounced in software to avoid flicker.

6.4 OBD-II and Code Clearing Concept

To implement in-cluster OBD-II diagnostics and code clearing:

1. An additional OBD-II style connector is mounted behind the cluster or in an accessible location.
2. The Raspberry Pi connects to the vehicle OBD-II network either:
   - Directly over CAN (ISO 15765-4) using the existing PiCAN HAT, or
   - Via a dedicated OBD-II interface IC or module that handles protocol details.
3. The software sends service-mode requests (for example, OBD-II Mode 3 to read DTC codes, Mode 4 to clear them) over CAN.
4. Responses are parsed and shown on the display as readable code lists, possibly with short descriptions.
5. Clearing codes is gated behind a confirmation screen to avoid accidental operations.

In the simplest implementation, the cluster can plug into the car's existing OBD-II port via a small internal cable. In more advanced installs, the cluster can tee into the port so that external tools still work.

6.5 Electrical Cost Summary

Component	DIY Cost (CAD)	Labour Cost (CAD)
Wiring Loom Materials	\$20–\$40	–
Automotive Connectors	\$10–\$30	–
Protection Components	\$10–\$25	–
Harness Fabrication Labour	–	\$100–\$250
Section subtotal	**\$40–\$95**	**\$100–\$250**

7 Mechanical Engineering and CAD Prototyping

7.1 Measurement and Modeling

For each car:

- The OEM cluster is removed and measured: overall width and height, depth, mounting points, viewing angle, and steering column clearances.

- Critical surfaces are modeled first in CAD (for example, in Fusion 360 or SolidWorks), focusing on the interface with the dashboard and mounting screws.
- The display outline, PCB outline, and connector access zones are imported into the CAD model to ensure they physically fit.

7.2 Enclosure Design

The enclosure is designed to:

- Angle the screen toward the driver for visibility.
- Include internal ribs and standoffs to mount the PCB and hold the TFT panel.
- Provide small airflow paths where needed so the Raspberry Pi does not overheat.
- Allow for a removable front lens or bezel so the screen can be serviced.

7.3 3D Printing and Iteration

- Early prototypes are printed in PLA at low infill to quickly check fit in the dashboard opening.
- Iterations refine mounting tab positions, screw hole diameters, and clearances around the steering wheel and stalks.
- Once geometry is validated, a final enclosure is printed in ABS or PETG for better heat resistance, or in Nylon/SLS if higher strength is desired.

7.4 Prototyping Cost Summary

Item	DIY Cost (CAD)	Labour Cost (CAD)
PLA Prototype Prints	\$10–\$20	–
ABS/PETG Final Prints	\$20–\$40	–
Nylon/SLS Prints	\$40–\$120	–
CAD Modelling Labour	–	\$150–\$300
Section subtotal	**\$30–\$180**	**\$150–\$300**

8 PCB Engineering and Manufacturing

8.1 PCB Capabilities

The custom PCB consolidates:

- A CAN transceiver and physical layer interface.
- An external ADC (or multiple ADC channels) for analog sensor inputs.
- 5 V and 3.3 V generation from a 12 V input, with appropriate filtering.
- Surge and reverse-polarity protection circuitry.
- Dedicated connectors to the vehicle harness, TFT display, and Raspberry Pi.

8.2 Layout Considerations

- Power and ground traces are sized for expected currents and use short, wide paths.
- Sensitive analog lines are routed away from high-current switching nodes.
- Decoupling capacitors are placed close to IC supply pins.
- Test pads or headers are added for debugging (for example, CAN, power rails, and key ADC inputs).

8.3 PCB Cost Summary

Item	DIY Cost (CAD)	Labour Cost (CAD)
PCB Manufacturing	\$25–\$80	–
Component Kit	\$20–\$40	–
Assembly (optional)	\$30–\$90	–
Professional Assembly	–	\$100–\$250
Section subtotal	**\$45–\$210**	**\$100–\$250**

9 UI/UX Development

9.1 Interface Design

UI is built from layered Photoshop assets and then implemented in PyGame. Major elements:

- A large tachometer arc with smooth sweep for RPM.
- A digital speed readout for easy track visibility.
- Coolant and oil temperature bars, ideally coloured by range.
- Shift lights that respond to RPM and gear, with adjustable thresholds.
- Warning pop-ups for low oil, high coolant temperature, battery voltage, and generic faults.
- A menu system accessible via steering wheel buttons or side buttons on the cluster.

9.2 Day/Night Modes and Themes

- Bright daytime theme prioritizing contrast against sunlight.
- Dim nighttime mode that automatically engages when headlights are on or by user preference.
- Vehicle-specific themes (for example, BMW-style vs. Honda-style graphics) driven by configuration files.

9.3 UI Cost Summary

Component	DIY Cost (CAD)	Labour Cost (CAD)
Photoshop Assets	Free–$20	–
UI Coding (PyGame)	Free	–
Professional UI Labour	–	$200–$600
Section subtotal	**$0–$20**	**$200–$600**

10 Testing and Validation

10.1 Bench Testing

Before going into the car, the cluster is tested on the bench:

- A bench power supply simulates the 12 V system and allows controlled brown-out tests.
- A CAN simulator (or another Raspberry Pi) sends fake RPM, speed, and temperature data so the UI can be exercised.
- Heat tests are done by running the cluster in a warm environment for extended periods.
- Power-loss tests verify that the Raspberry Pi shuts down cleanly and that the file system does not corrupt.

10.2 In-Vehicle Testing

Once bench tests are passed:

- The cluster is temporarily mounted and wired into the car using a non-destructive harness adapter.
- Real CAN traffic is captured and recorded to refine scaling and ID mappings.
- Track-style vibration and braking are used to check for display flicker, connector loosening, or mounting issues.
- Logs are reviewed for missed frames or CPU usage spikes.

10.3 Testing Cost Summary

Item	DIY Cost (CAD)	Labour Cost (CAD)
Bench Power Supply	$30–$60	–
Multimeter	$20–$50	–
CAN Simulator Tools	$0–$80	–
Testing Labour	–	$150–$400
Section subtotal	**$50–$190**	**$150–$400**

11 Business and Sales Notes

11.1 Scalability Factors

- The same electronics platform can be reused across many cars, lowering per-unit cost.

- Car-specific parts (harness adapters and bezels) are low-cost 3D prints or small harness assemblies.
- Software is modular enough that new cars mostly require configuration rather than new features.

11.2 Estimated Customer Pricing

- Hardware: $300–$500.
- Labour: $150–$300 (wiring, CAD tweaks, configuration, installation).
- Retail Range: $600–$900, depending on options and level of customization.

12 Total DIY Cost Estimate

Category	Cost Range (CAD)
Display + Pi	$150–$250
CAN HAT	$60–$120
PCB + Components	$45–$120
Wiring / Relays	$20–$50
3D Printing	$10–$40
Development Tools	$30–$80
OBD/CAN Tools	$20–$40
Total Estimated DIY Cost	**$335–$700**

13 Summary

The instrument cluster is a fully engineered, modular, universal platform capable of replacing OEM analog clusters across multiple vehicles, including the Honda Civic EF track build. The architecture separates car-specific details (harness, CAN mapping, enclosure) from universal elements (electronics, software, UI logic), making it practical to support many platforms.

14 Conclusion

The digital instrument cluster development program integrates mechanical engineering, PCB design, UI/UX, software development, and CAN-BUS decoding into a single universal platform.

Originally built around a BMW E90 335i, the system is deliberately structured so that only wiring and configuration need to change to support a new vehicle such as the Honda Civic EF. The combination of:

- A Raspberry Pi compute core,
- A high-brightness TFT display,
- A custom PCB with CAN, ADC, and power management,
- CAD-modeled, 3D-printed enclosures,

- And a modular Python software stack

creates a flexible foundation for future expansion.

Future revisions can add in-cluster OBD-II diagnostics and code clearing, predictive shift lights, GPS-based lap timing, wireless updates, and more advanced data logging. The platform not only modernizes vintage vehicles, but also positions itself as a repeatable, semi-universal aftermarket product for custom builds and track cars.

Overall, the project elevates cockpit functionality, demonstrates a complete multi-disciplinary engineering workflow, and lays the groundwork for a small-scale product that is scalable, reliable, and fully customizable.