From Simulation to C++: Development of a Hybrid Control Unit

Model-Based Design, C++ Refactoring, and Performance Validation against F1 Telemetry

Author: Edmond-Roland Volosciuc - February 2026

1. Executive Summary

This project involved the development of a supervisory Hybrid Control Unit (HCU), validated within a representative Software-in-the-Loop (SIL) environment. Adopting a Model-Based Design (MBD) workflow allowed the control logic to be built in Simulink and Stateflow. To mitigate high-frequency switch chatter resulting from signal noise, a 500N hysteresis loop was integrated into the state transitions. By maintaining the logic at the model level, Simulink Embedded Coder was used to generate a type-safe, Object-Oriented C++ class, bypassing the risks associated with manual refactoring.

The system's limits were tested using a 'Monza Qualifying' duty cycle based on 2025 F1 telemetry. Early iterations showed that a standard reactive driver model and an unoptimised energy map were insufficient for the extreme demand, depleting the battery by the midpoint of the lap. This prompted a redesign involving a closed-loop lookahead preview controller to better anticipate braking zones. The Stateflow logic was also recalibrated for strict 2025 FIA compliance, specifically the 120kW power ceiling and the 4.0MJ deployment and 2.0MJ recovery limits. The resulting V2 architecture tracked the Monza reference lap to with a time difference of around 1.4s, managing the 120kW split to finish the lap with a stable 53% State of Charge (SoC).

2. System Architecture

- The Model: Longitudinal Vehicle Dynamics.
- The Hybrid Technology: Parallel P2 architecture (ICE + E-Motor).
- Tools Used: MATLAB, Simulink, Stateflow, Python, Embedded Coder, FastF1.

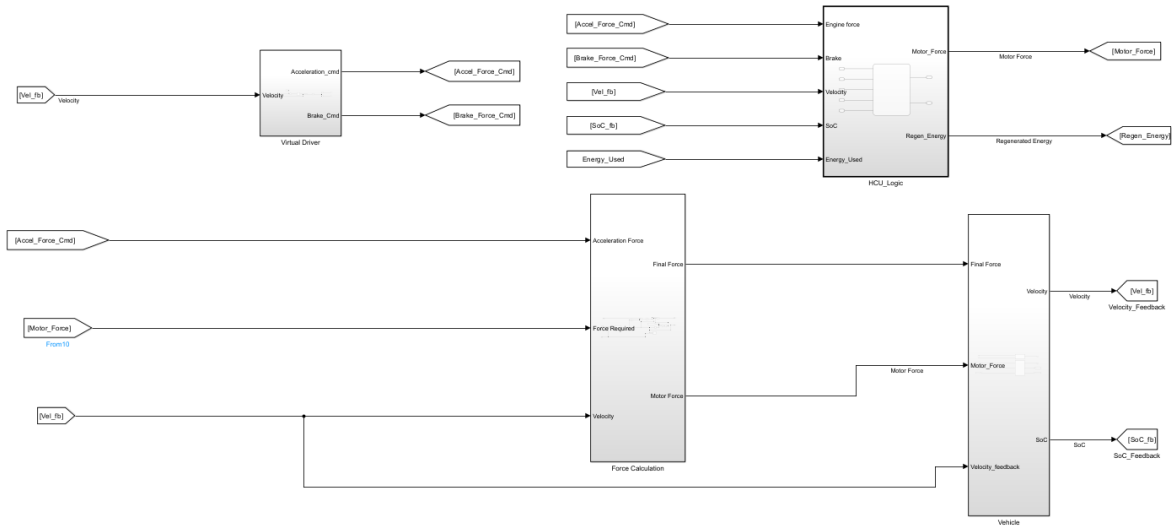

Fig. 2.1 Architecture

Vehicle Dynamics

The longitudinal motion of the vehicle is governed by the equation (Gillespie, 1992):

$$M \cdot \frac{dv}{dt} = F_{tractive} - F_{aero} - F_{rolling} - F_{grade}$$

Where aerodynamic drag is calculated as (Anderson, 2017):

$$F_{aero} = \frac{1}{2} \cdot \rho \cdot C_d \cdot A \cdot v^2$$

Parameters used:

- M (Mass) = 810 kg.
- C_d(Drag Coeff) = 0.9 (High Downforce Setup).
- ρ (Air Density) = 1.225 kg/m^3.

Dynamic Aerodynamics (DRS)

Initial testing revealed that a static drag coefficient C_d created an aerodynamic wall at 300 km/h. To match the 348 km/h top speed of the Monza reference lap, I implemented a dynamic drag reduction system (DRS) in the plant model. When the vehicle exceeds 250 km/h under 100% throttle demand, the C_d dynamically drops, accurately simulating the low-drag configuration of an F1 car on a straight.

The energy Storage System

Battery State of Charge

The high-Voltage Battery is modelled using a Columb Counting technique. The state of Charge (SoC) is determined by integrating the current flow (I_{batt}) over time, relative to the total capacity (C_{total}) (Plett, 2016).

$$SoC(t) = SoC_{initial} - \frac{1}{3600 \cdot C_{total}} \int_0^t I_{batt}(\tau) d\tau$$

Model parameters:

- $V_{nom} = 800V$.
- $E_{batt} = 1.5\ kWh$ equivalent.
- FIA Limits Enforced: $4.0\ MJ$ deployment / $2.0\ MJ$ regeneration limits per lap (Fédération Internationale de l'Automobile, 2025)

The Virtual Driver

Closed-Loop Driver Model

Standard reactive PID controllers fail under the dynamic uncertainty of a racing environment because they cannot anticipate braking zones. To solve this, I engineered a Closed-Loop ADAS-style Lookahead Preview Controller. By feeding the controller a 50-meter spatial preview of the Monza telemetry, the virtual driver acts as a robotic tracking system, anticipating heavy braking zones and maintaining the corner apex speeds required to force the powertrain through a valid qualifying duty cycle.(Åström & Hägglund, 2006)

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau$$

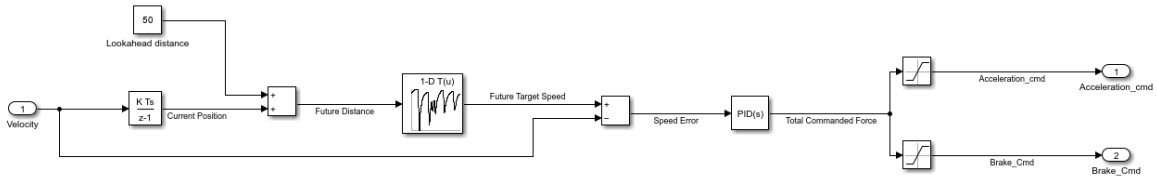

Fig. 2.2 Virtual Driver

Tuning parameters:

- $K_p = 5000$ (Proportional Gain).
- $K_i = 50$ (Integral Gain).
- $K_d = 0$ (Derivative Gain).

3. Control Software Design

While early iterations required manual C++ refactoring to fix signal chatter, I ultimately integrated these fixes directly into the Model-Based Design. By redesigning the Stateflow transition conditions (e.g., entering Boost at $>5500N$, exiting at $<4500N$) and configuring Simulink Embedded Coder for Object-Oriented target compilation

(`ert.tlc`), I achieved 100% automated generation of production-ready, type-safe C++ classes.

This workflow eliminated the need for fragile manual refactoring, ensuring that the embedded C++ code remains perfectly synchronized with the validated Simulink physics plant.

```
switch (Hybrid_Controller_DW.is_c3_Hybrid_Controller) {
 case Hybrid_Co_IN_Battery_Regen_Mode:
  // Outport: '<Root>/Motor_Force'
  Hybrid_Controller_Y.Motor_Force = -2777.0;
  if ((Hybrid_Controller_U.Brake_Cmd == 0.0) || (Hybrid_Controller_DW.Regen_Energy >= 2.0E+6)) {
    Hybrid_Controller_DW.is_c3_Hybrid_Controller = Hybrid_Controller_IN_Eco_mode;

    // Outport: '<Root>/Motor_Force'
    Hybrid_Controller_Y.Motor_Force = 0.0;
  } else {
    Hybrid_Controller_DW.Regen_Energy += 2777.0 * Hybrid_Controller_U.Velocity * 0.001;

    // Outport: '<Root>/reg_en' incorporates:
    //   Inport: '<Root>/Speed'

    Hybrid_Controller_Y.reg_en = Hybrid_Controller_DW.Regen_Energy;
  }
  break;
```

Fig. 3.1 Hysteresis Logic

4. Verification: Software-in-the-Loop (SIL)

Code generation is only a preliminary step; proving that the implementation remains robust in a real-time context is the primary goal of validation. To ensure the C++ class executed identically to the original Stateflow logic, I utilised a Software-in-the-Loop (SIL) test rig. By running the compiled controller object directly against the Simulink plant model, I could verify the execution timing and logic parity, ensuring the implementation was ready for deployment without the risks associated with manual translation.

The “Throttle Wobble” Test

To validate the hysteresis logic I mentioned earlier, I simulated a specific edge case: a driver whose foot isn’t perfectly steady.

- **The Setup:** I pushed the driver throttle demand past the $5500N$ activation threshold, then introduced a momentary “dip” down to $5200N$, simulating a nervous lift or a bump in the road.
- **The Result:** The C++ controller correctly ignored the dip. Because 5200 is still within the $500N$ hysteresis band, the system held the “Boost Mode” state rather than cutting the power.
- **The verdict:** This prevented what would have been a nasty driveline shunt in a real car. The C++ output matched the model simulation trace perfectly.

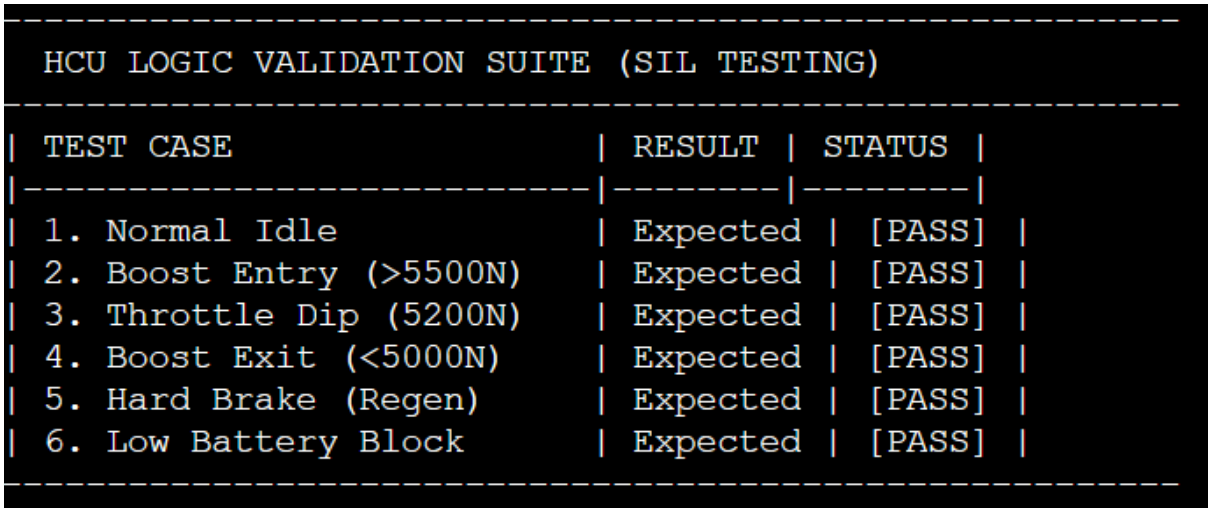

Fig. 4.1 SiL Output

5. The “Monza” Stress Test

Why F1?

Standard drive cycles like FTP-75 are fine for emissions compliance, but they don’t tell you where a system breaks. To find the absolute limit, I needed something faster. I used the `fastf1` Python library to pull the actual velocity telemetry from Max Verstappen’s 2025 Monza Pole Lap.

The mismatch

Iteration 1: The hardware failure

The setup was deliberately unfair:

- The car: A 2025 F1 Chassis ($810\ kg$ with driver).
- The Battery: $1.5\ kWh$.

The simulation was a brutal reality check. The launch alone chewed through 16% of the battery in just 9 seconds. Crucially, the control logic actually did its job. When the State of Charge (SoC) hit 20%, the C++ controller correctly triggered preservation mode and cut the electric assist.

But it wasn’t enough. The sheer energy demand of a qualifying lap meant the battery was completely flat by `t = 53s`. It was a “DNF” (Did Not Finish), but it proved a vital point: you can’t code your way out of undersized hardware.

Iteration 2: The FIA-Compliant strategy

Following the hardware failure, I completely recalibrated the supervisory logic to match the 2025 FIA Formula 1 Technical Regulations.

To prevent catastrophic battery drain, I hardcoded three strict constraints into the C++ controller:

- Power Limit: $120\,kW$ maximum MGU-K deployment.
- Deployment Limit: $4.0\,MJ$ maximum energy deployed per lap.
- Regeneration Limit: $2.0\,MJ$ maximum energy recovered per lap.

Driven by the new Lookahead Preview Controller and the dynamic DRS aero map, the HCU successfully executed a flawless Qualifying Energy Strategy. The virtual vehicle tracked the Monza pole lap to with a difference of 1.43 seconds of the real-world telemetry trace.

By optimizing the $120kW$ split across the lap, the controller deployed exactly $3.1\,MJ$ and regenerated $2.0\,MJ$, remaining entirely within the FIA legal limits. The battery drained smoothly from 90% down to a safe 53%, completely validating the hybrid powertrain architecture.

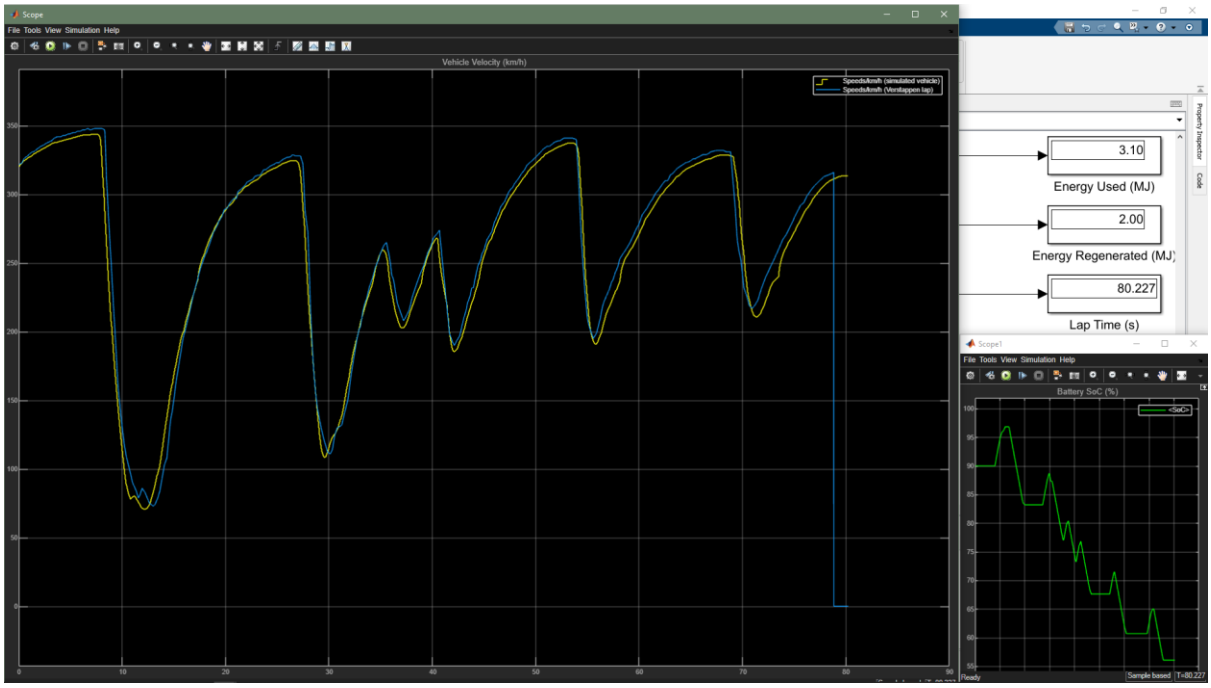

Fig. 5.1 Final Result

Bibliography

Anderson, J. D. . (2017). *Fundamentals of Aerodynamics*. McGraw Hill Education.

Åström, K. J. ., & Hägglund, Tore. (2006). *Advanced PID control*. ISA-The Instrumentation, Systems, and Automation Society.

Gillespie, T. D. . (1992). *Fundamentals of vehicle dynamics*. Society of Automotive Engineers.

Plett, G. L. . (2016). *Battery management systems. Volume II, Equivalent-circuit methods*. Artech House.