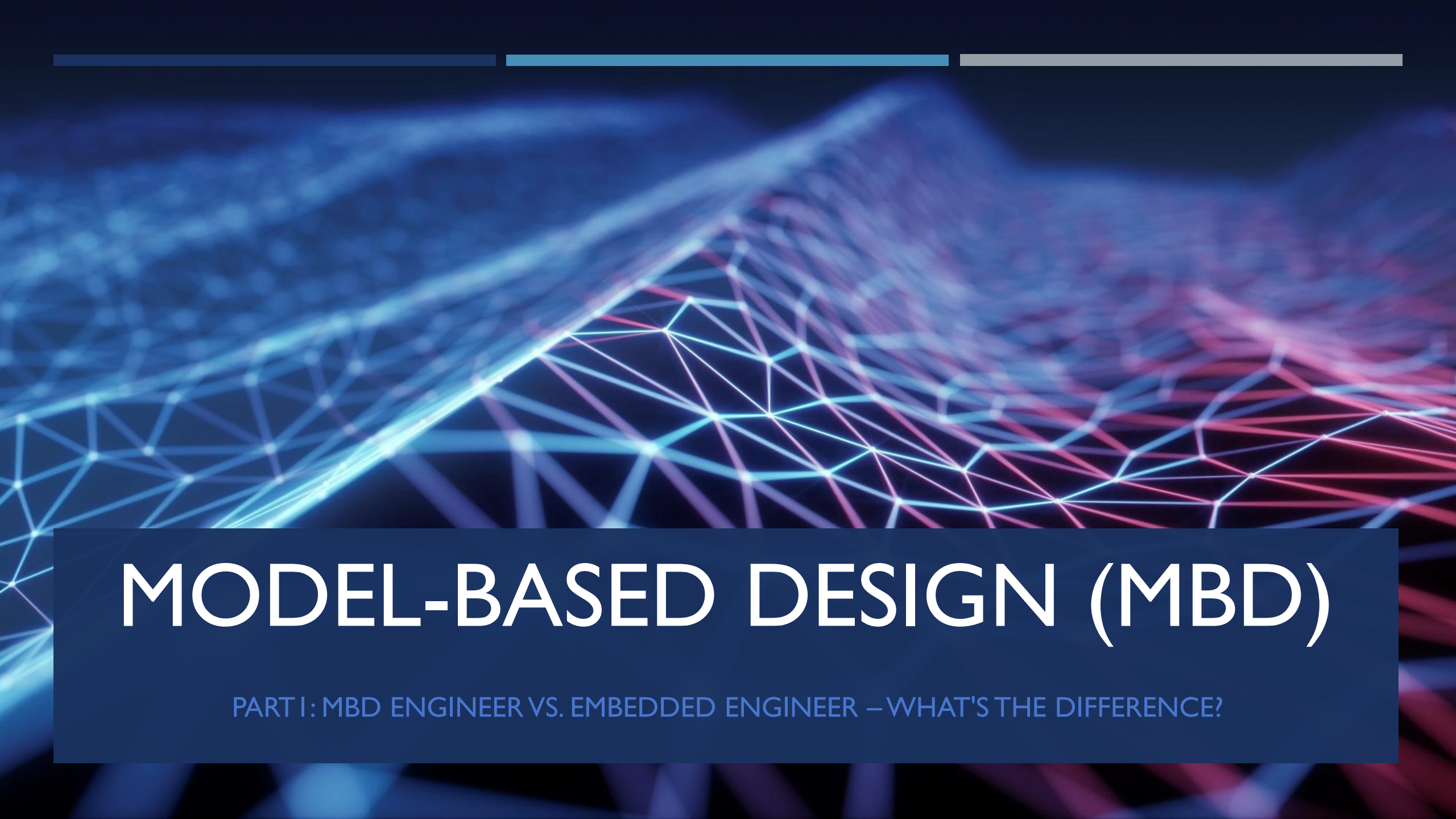
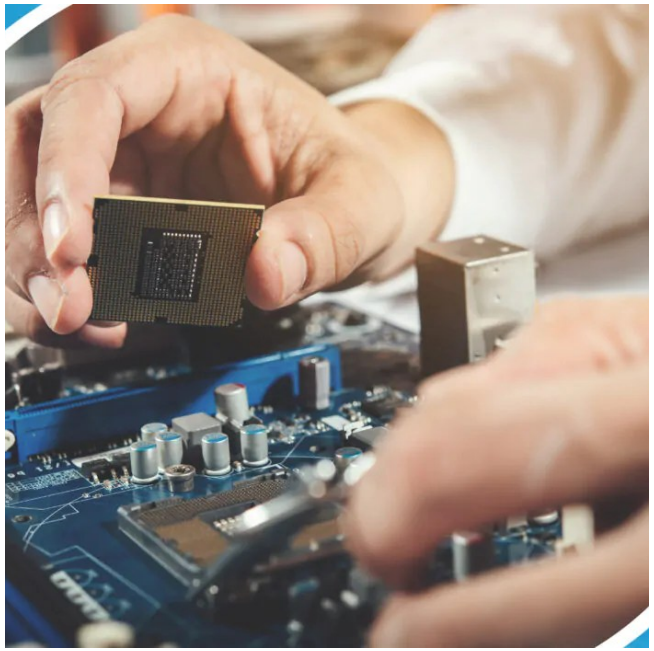


MODEL-BASED DESIGN (MBD)

PART I: MBD ENGINEER VS. EMBEDDED ENGINEER – WHAT'S THE DIFFERENCE?

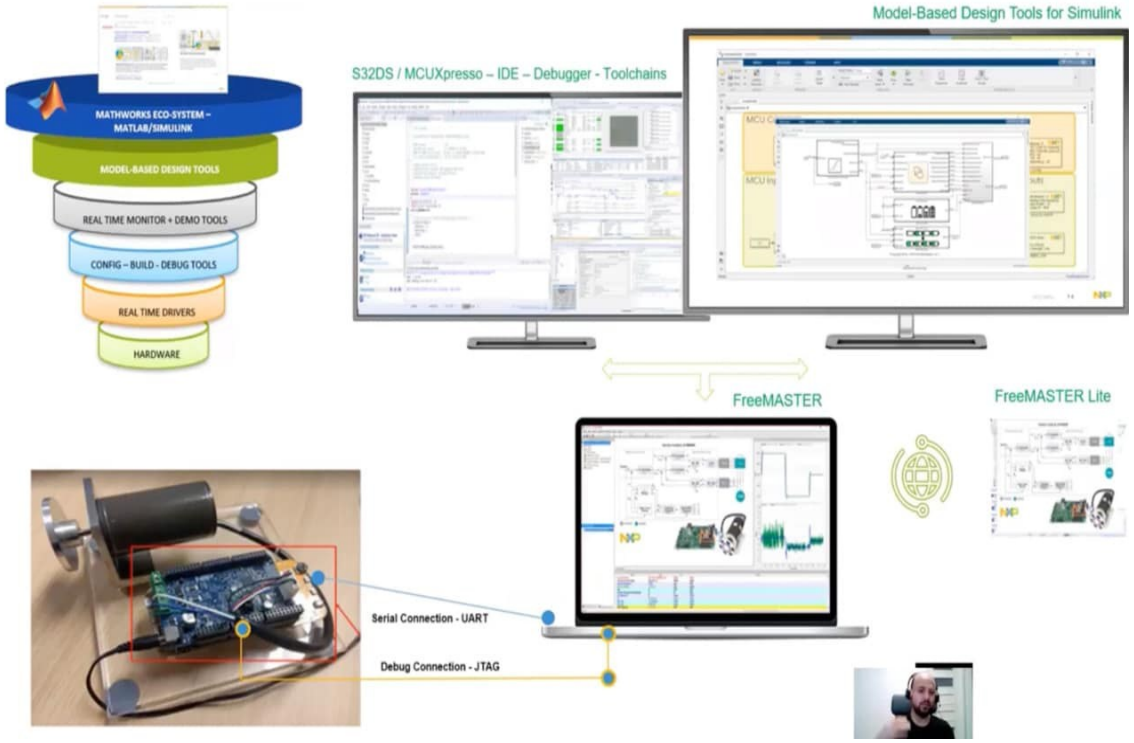
MBD



Embedded

MBD ENGINEER VS. EMBEDDED ENGINEER: WHICH PATH SUITS YOU? 🤔

MBD ENGINEER:



- Works at a **higher level of abstraction**, focusing on designing, simulating, and verifying systems using tools like MATLAB/Simulink.
- Uses **block diagrams and state machines** to model control algorithms before implementation.
- Relies on **automatic code generation** (via tools like Embedded Coder) to translate models into embedded C code.
- Plays a key role in **rapid prototyping, Hardware-in-the-Loop (HIL), and Software-in-the-Loop (SIL) testing**.

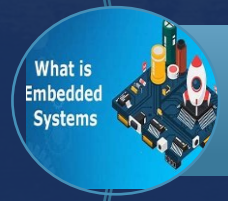
EMBEDDED ENGINEER:

- Works **closer to hardware**, writing optimized C/C++ code for microcontrollers and embedded systems.
- Focuses on **low-level programming, firmware development, and hardware interfacing**.
- Deals with **real-time constraints, memory management, and power optimization**.
- Implements and debugs software directly on hardware using **IDE, debuggers, and logic analyzers**.

SO, WHICH ROLE IS RIGHT FOR YOU?

Love math, simulations, and high-level design?

Go for MBD!



Enjoy low-level coding, hardware, and real-time systems?

Embedded Engineering is for you!

👉 **Which path are you more interested in? Let's discuss in the comments!** 💬

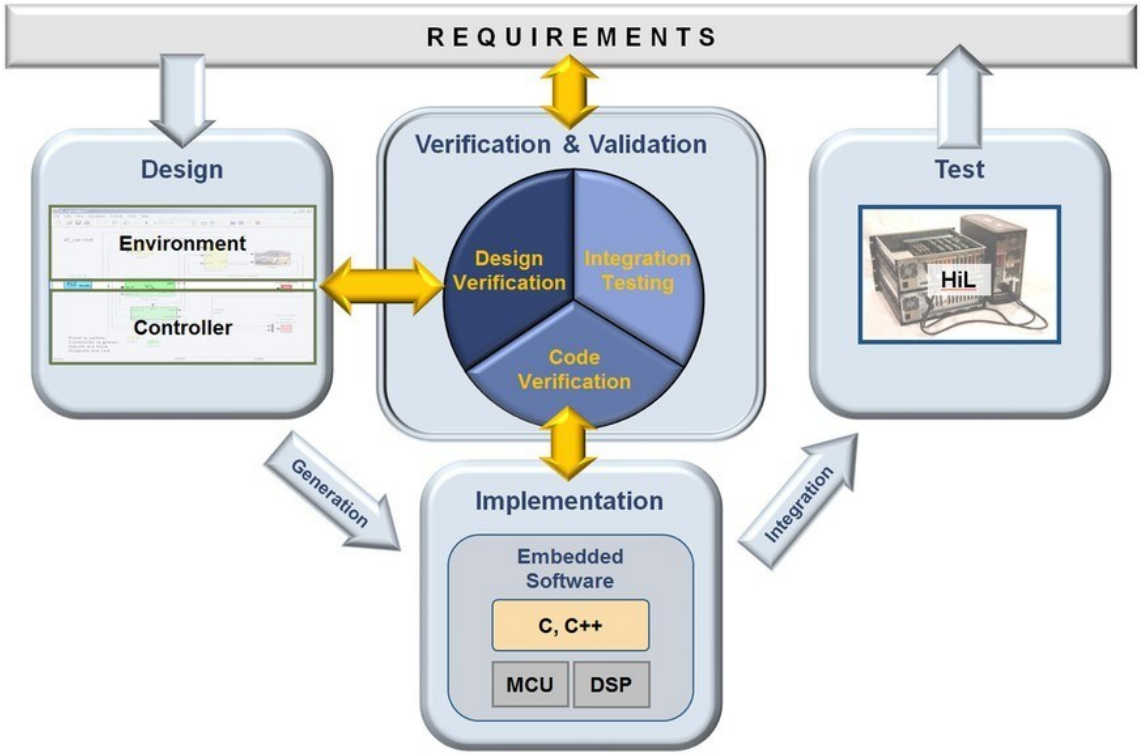
MODEL-BASED DESIGN (MBD)

PART II: MBD PROCESS & V-MODEL – THE SECRET TO ROBUST SYSTEM DESIGN

🚀 EVER WONDERED HOW COMPLEX SYSTEMS ARE DESIGNED WITH MINIMAL ERRORS?

- 🔎 The **Model-Based Design (MBD) Process** follows a structured approach to **develop, simulate, test, and deploy** control systems, ensuring efficiency and reliability.

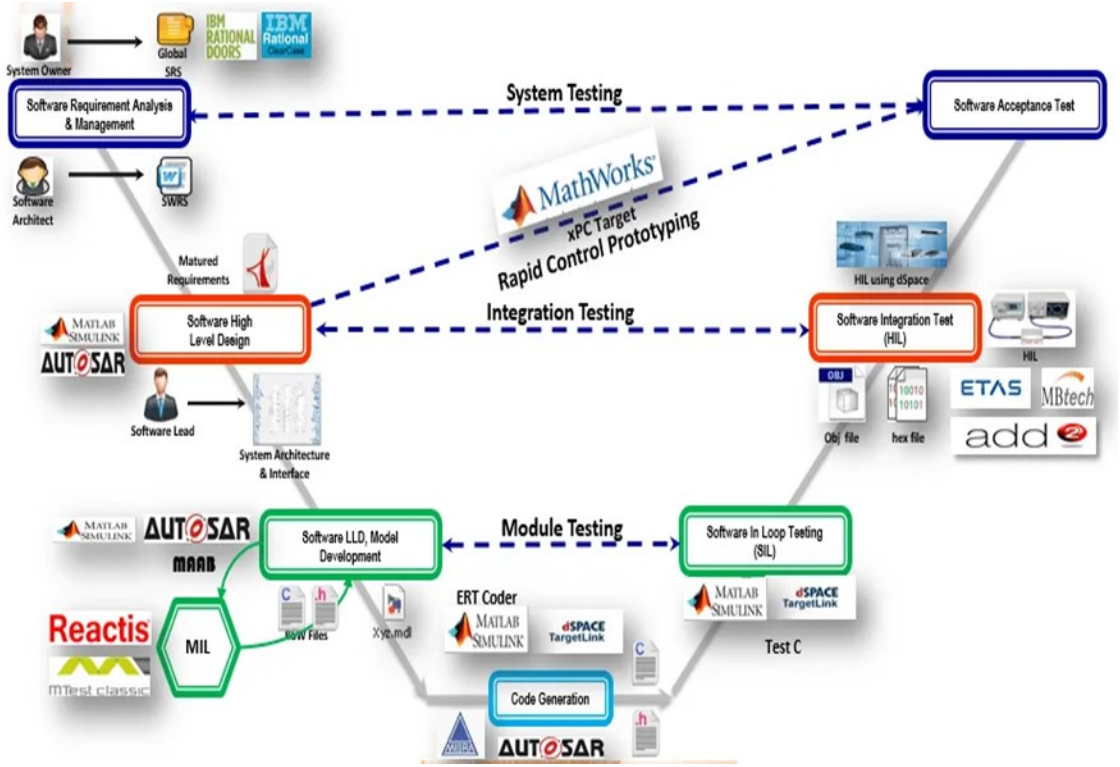
- *One of the most powerful methodologies in MBD is the **V-Model**. But why is it so important?* 🤔



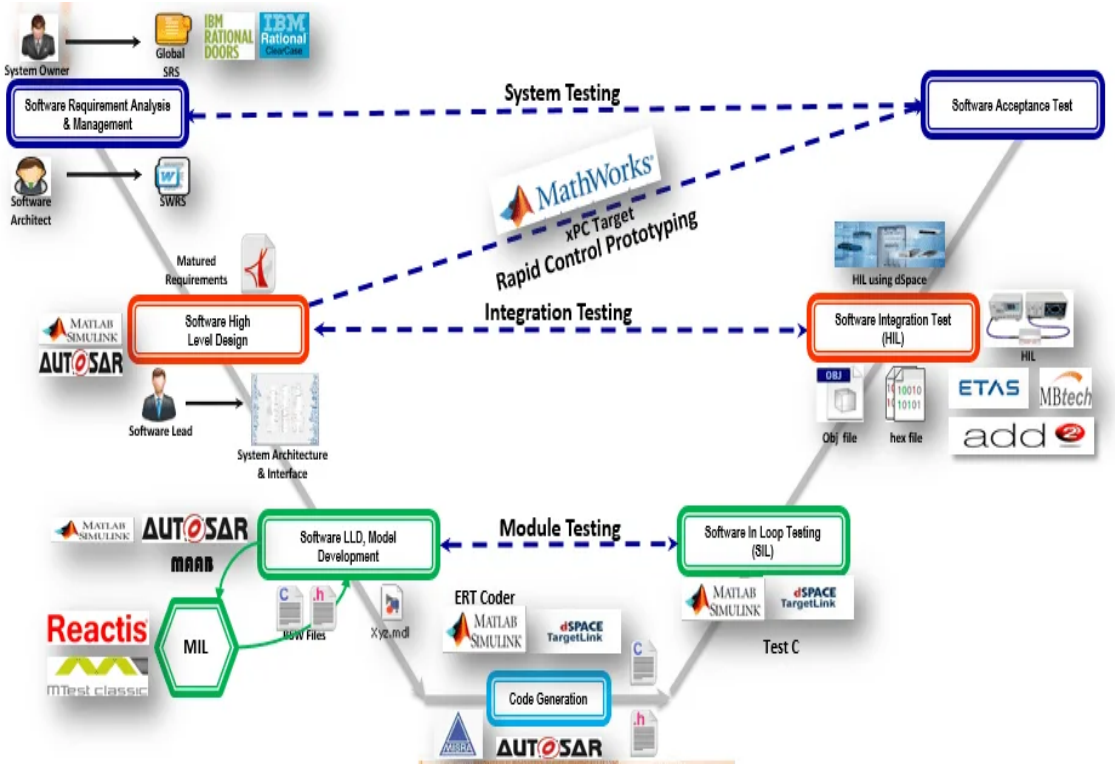
THE V-MODEL: A BLUEPRINT FOR SUCCESS

- **Left Side – System Design & Decomposition**

1. **Requirements Definition** (System-level specs & constraints)
2. **System Design** (Defining high-level architecture)
3. **Modeling & Simulation** (Building Simulink models)
4. **Code Generation & Integration** (Automatic Embedded C code from models)



THE V-MODEL: A BLUEPRINT FOR SUCCESS



- **Right Side – Testing & Validation**

5 **Unit Testing** (Verifying small components)

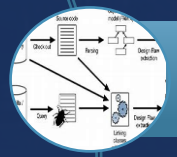
6 **Integration Testing** (Ensuring modules work together)

7 **Hardware-in-the-Loop (HIL) Testing** (Validating on real hardware)

8 **System Validation** (Final validation before deployment)

WHY DOES MBD + V-MODEL MATTER?

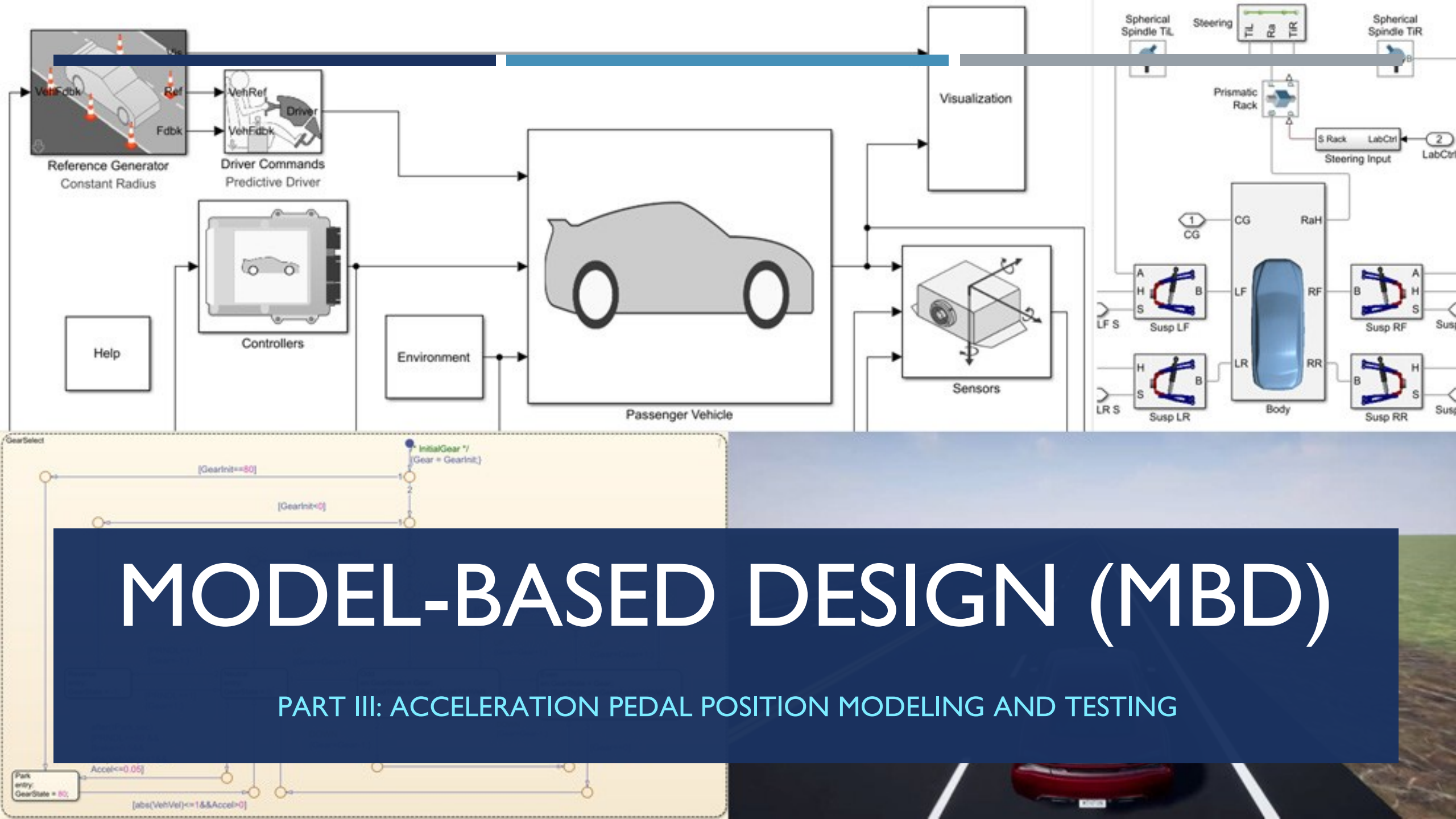
Reduces development time and cost



Catches design flaws early through simulation

Ensures smooth integration between software & hardware

Enhances system reliability & safety



MODEL-BASED DESIGN (MBD)

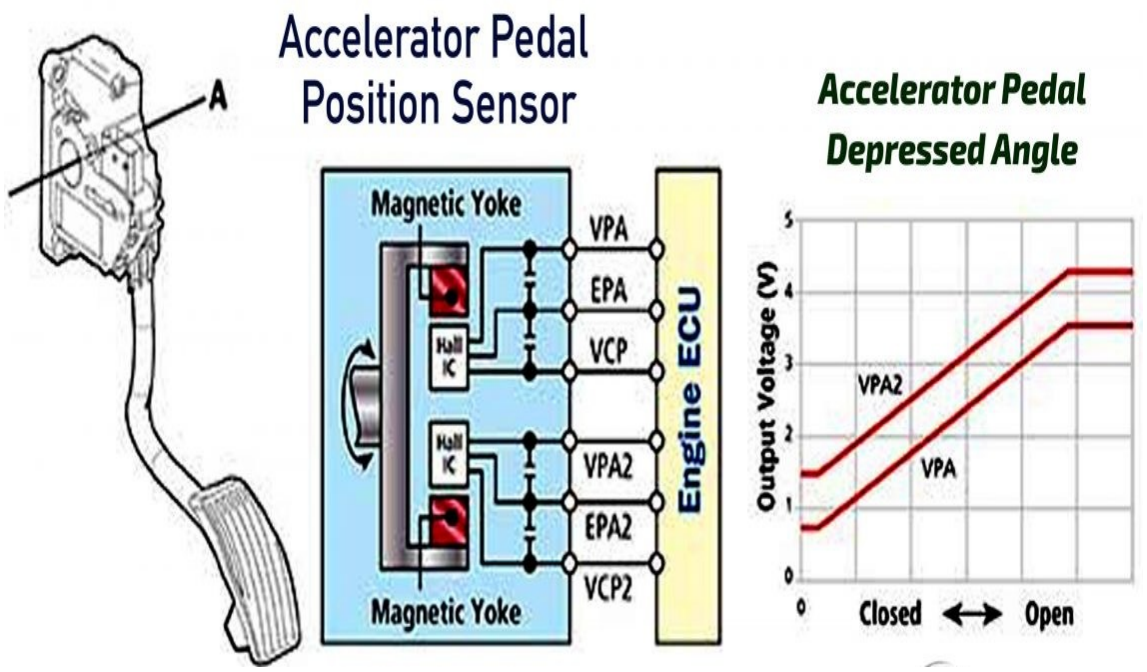
PART III: ACCELERATION PEDAL POSITION MODELING AND TESTING

🔧 ACCELERATION PEDAL POSITION MODELING

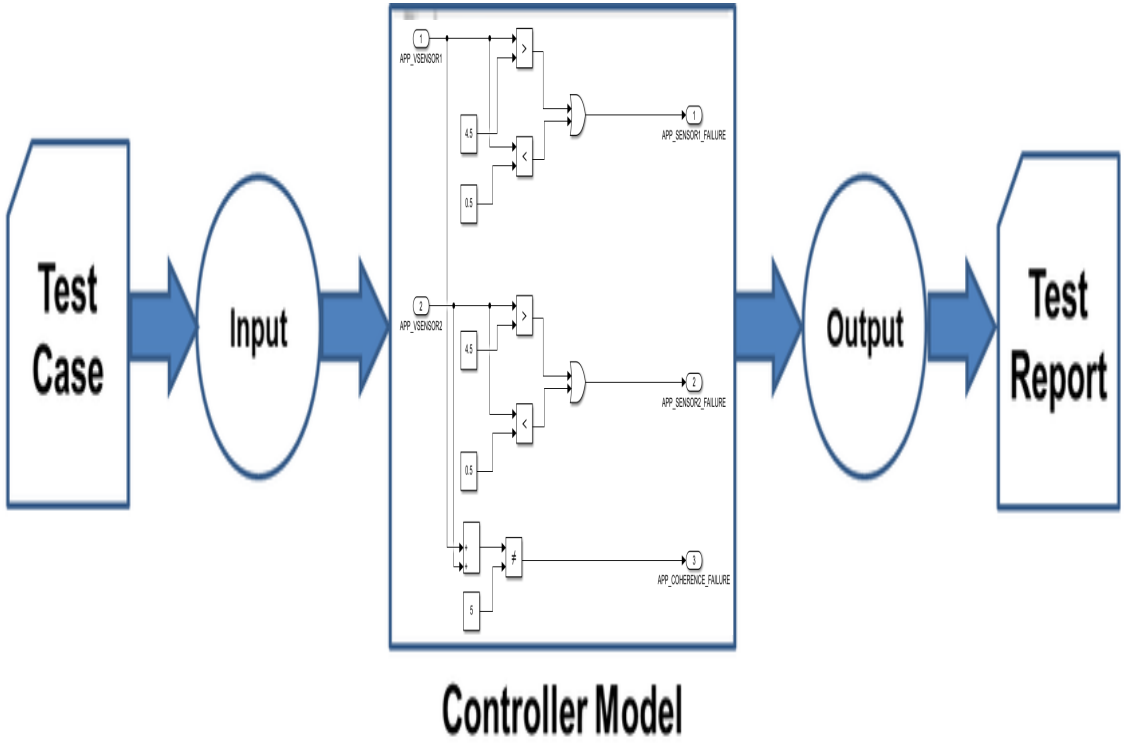
THE ACCELERATION PEDAL POSITION IS USUALLY MEASURED USING **TWO REDUNDANT SENSORS** TO ENHANCE SAFETY AND RELIABILITY. THE SENSORS PROVIDE INDEPENDENT SIGNALS TO THE **ELECTRONIC CONTROL UNIT (ECU)** TO DETECT ANY DISCREPANCIES AND ENSURE FAIL-SAFE OPERATION.

- 🛠️ **Modeling Approach:**

1. **Sensor Modeling:** Develop mathematical models for both sensors that simulate real-world behavior.
2. **Signal Processing:** Implement filtering to minimize noise and enhance accuracy.
3. **Fault Detection Logic:** Create control logic that:
    1. Compares signals from both sensors.
    2. Detects inconsistencies and flags potential faults.
    3. Identifies which sensor is malfunctioning based on signal deviation and plausibility checks.



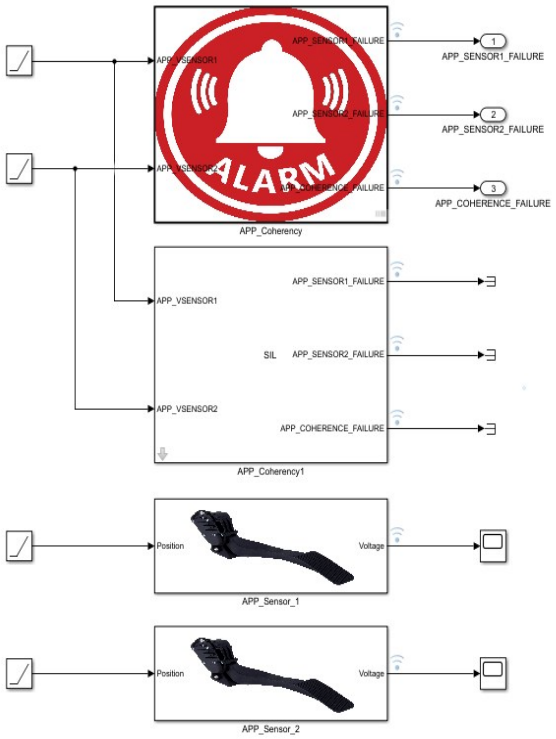
🧪 MODEL-IN-THE-LOOP (MIL) TESTING:

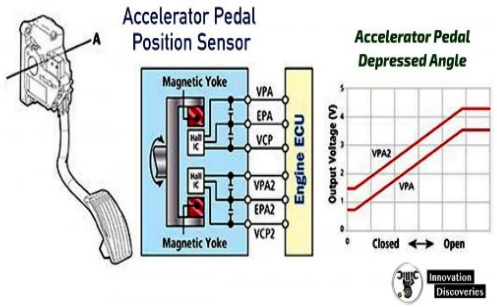


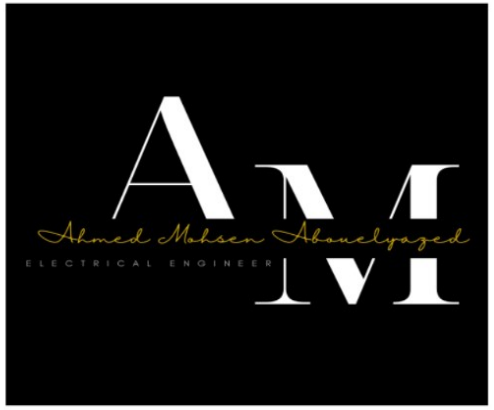
- Model-based testing is crucial for verifying the accuracy and robustness of the control logic.
- ✅ **MIL Test Steps:**

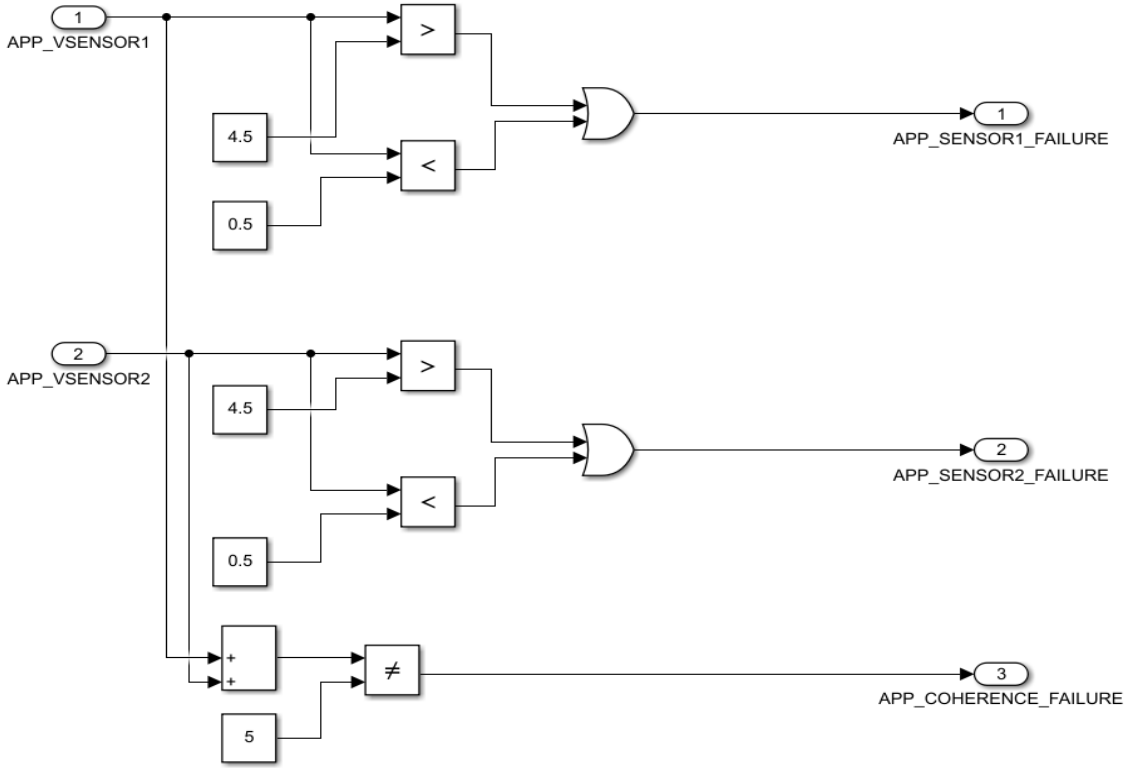
1. **Simulate Normal Operation:** Ensure both sensors deliver consistent signals.
2. **Inject Faults:** Introduce discrepancies like stuck-at values or noisy signals to simulate sensor failure.
3. **Verify Fault Detection Logic:** Check that the system accurately identifies faulty sensors and initiates safety measures.

ACCELERATION PEDAL POSITION MODELING





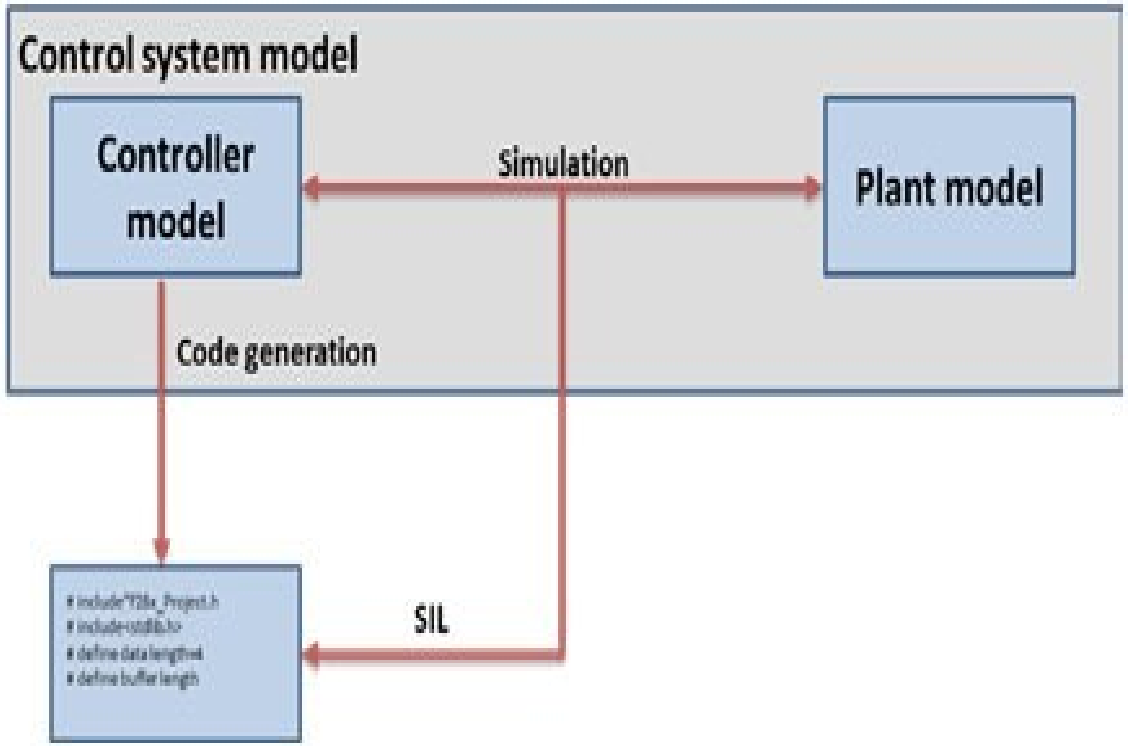




💻 SOFTWARE-IN-THE-LOOP (SIL) TESTING:

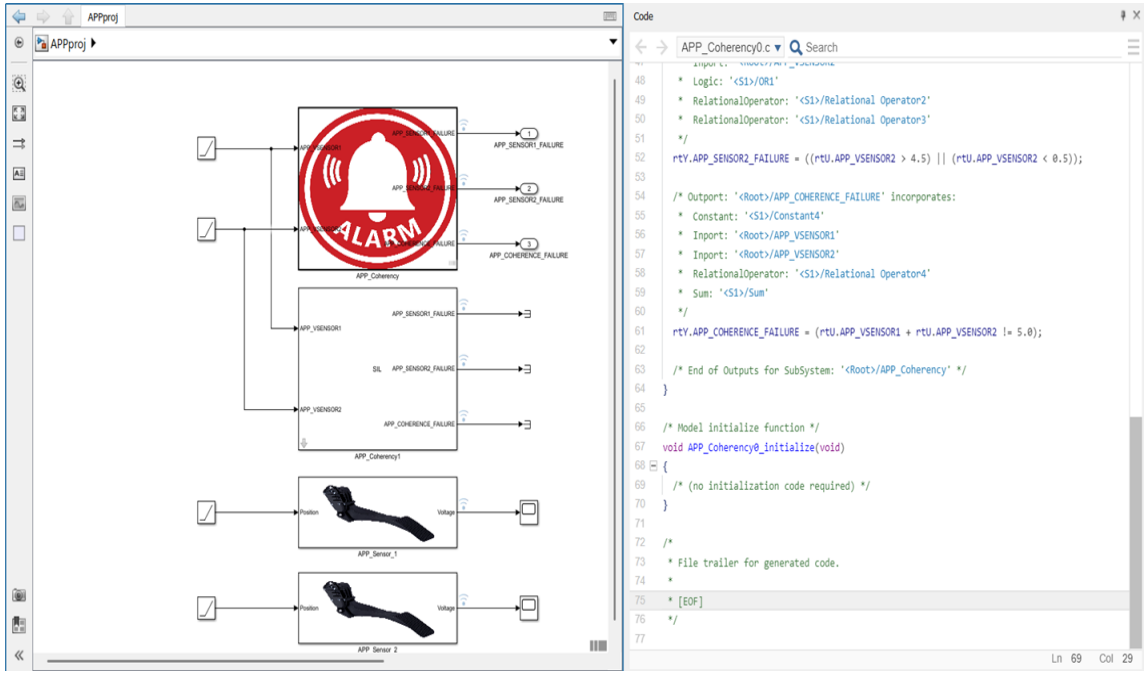
- After validating the model, it's time to test the generated code.
- ✅ **SIL Test Steps:**

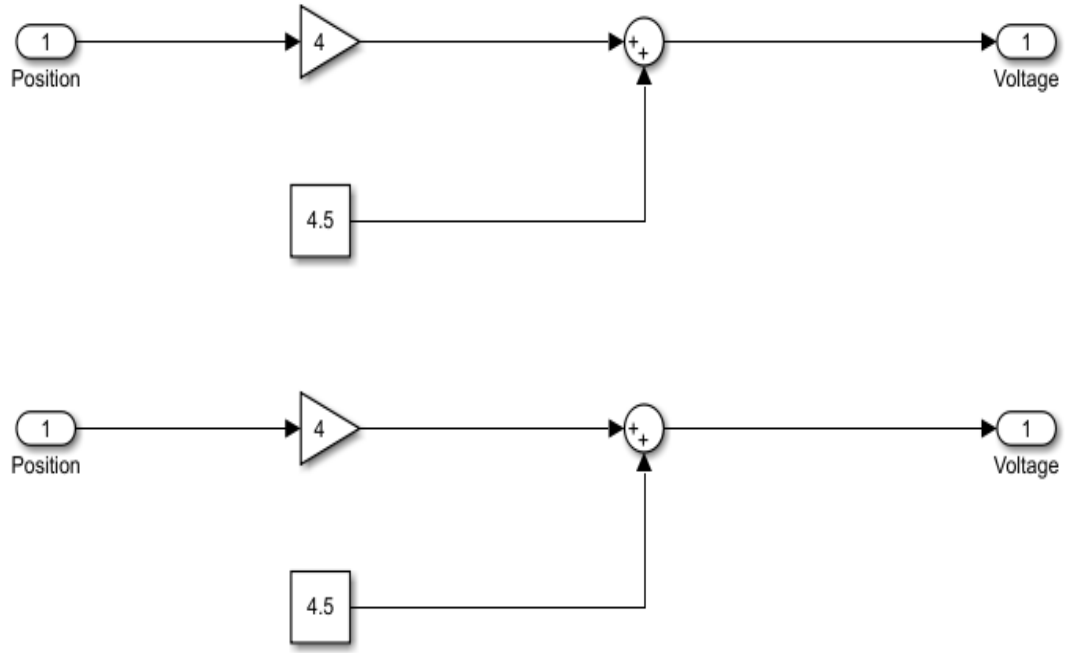
1. **Integrate the Auto-Generated Code** into a virtual ECU environment.
2. **Re-run Test Cases** from MIL to ensure the behavior is consistent.
3. **Monitor Real-Time Performance:** Check the system's ability to detect faults without compromising speed.



APP_Coherency0.c

ACCELERATION PEDAL POSITION MODELING





ACCELERATION PEDAL POSITION MODELING

```
/*
* File: APP_Coherency0.c
*
* Code generated for Simulink model 'APP_Coherency0'.
*
* Model version : 1.7
* Simulink Coder version : 9.5 (R2021a) 14-Nov-2020
* C/C++ source code generated on : Mon Mar 10 00:21:51 2025
*
* Target selection: ert.tlc
* Embedded hardware selection: Atmel->AVR
* Code generation objectives:
* 1. Execution efficiency
* 2. RAM efficiency
* Validation result: Not run
*/

#include "APP_Coherency0.h"


/* External inputs (root inport signals with default storage) */
ExtU rtU;


/* External outputs (root outports fed by signals with default storage) */
ExtY rtY;


/* Real-time model */
static RT_MODEL rtM_;
RT_MODEL *const rtM = &rtM_;


/* Model step function */
void APP_Coherency0_step(void)
{
/* Outputs for Atomic SubSystem: '<Root>/APP_Coherency' */
/* Outport: '<Root>/APP_SENSOR1_FAILURE' incorporates:
* Constant: '<S1>/Constant'
* Constant: '<S1>/Constant1'
* Inport: '<Root>/APP_VSENSOR1'
* Logic: '<S1>/OR'
* RelationalOperator: '<S1>/Relational Operator'
* RelationalOperator: '<S1>/Relational Operator1'
*/
rtY.APP_SENSOR1_FAILURE = ((rtU.APP_VSENSOR1 > 4.5) ||
(rtU.APP_VSENSOR1 < 0.5));

/* Outport: '<Root>/APP_SENSOR2_FAILURE' incorporates:
* Constant: '<S1>/Constant2'
* Constant: '<S1>/Constant3'
* Inport: '<Root>/APP_VSENSOR2'
* Logic: '<S1>/OR1'
* RelationalOperator: '<S1>/Relational Operator2'
* RelationalOperator: '<S1>/Relational Operator3'
*/
rtY.APP_SENSOR2_FAILURE = ((rtU.APP_VSENSOR2 > 4.5) ||
(rtU.APP_VSENSOR2 < 0.5));

/* Outport: '<Root>/APP_COHERENCE_FAILURE' incorporates:
* Constant: '<S1>/Constant4'
* Inport: '<Root>/APP_VSENSOR1'
* Inport: '<Root>/APP_VSENSOR2'
* RelationalOperator: '<S1>/Relational Operator4'
* Sum: '<S1>/Sum'
*/
rtY.APP_COHERENCE_FAILURE = (rtU.APP_VSENSOR1 +
rtU.APP_VSENSOR2 != 5.0);

/* End of Outputs for SubSystem: '<Root>/APP_Coherency' */
}


/* Model initialize function */
void APP_Coherency0_initialize(void)
{
/* (no initialization code required) */
}


/*
* File trailer for generated code.
*
* [EOF]
*/
```

THANK YOU 😇🤗🌹

PROJECT GITHUB REPO QR CODE