Tell me about your self?

Thank you for the opportunity. My name is Manjunatha Reddy, and I have a little over five years of experience as a HIL Validation & Verification Engineer in the automotive domain. My core strengths include HIL testing, in-vehicle testing, ECU validation, automation, and strong hands-on knowledge of CAN, LIN, and CAN-FD diagnostics protocol.

I began my career in 2021 at Mobase Electronics, where I worked on validating automotive ECUs in both HIL and in-vehicle environments. My key project has been the Body Domain Controller (BDC) ECU for Hyundai and Kia, which integrates BCM and SMK , convenience, security, and safety-related features. My responsibilities include analysifunctionalities into a single controller.

In this role, I validate a wide range of comfortng customer specification documents, designing test cases, executing HIL test cases, developing CAPL automation scripts, I performing functional, fault-injection testing, and ensuring robust diagnostic validation.

I have strong hands-on experience with Vector tools such as VT System, vTESTstudio, and CANoe for log analysis. I also regularly use oscilloscope and multimeters for signal debugging. As part of my achievements, I received an On-the-Spot Award in 2023 and worked onsite in South Korea to support HIL bench setup and BDC feature validation.

Currently, I'm looking for an opportunity where I can apply my experience in HIL testing, automation, and ECU validation and contribute effectively to a strong engineering team.

Why do you want to join KPIT? / Why KPIT?

I want to join KPIT to contribute to next-generation mobility solutions while growing technically and professionally.
I believe KPIT's innovation-driven environment and global automotive projects are the perfect platform to apply my HIL and ECU validation expertise.

Why are you looking for a change?

I'm looking for a change to take on bigger challenges and grow in a more advanced automotive engineering environment. I want to work on system-level validation and contribute more effectively to complex automotive projects.

What is your biggest strength?

My biggest strength is my ability to quickly analyse and debug issues using CAN logs, signal behaviour, and system requirements, which helps in resolving problems efficiently and keeping validation on track."

What is your weakness?

"Earlier, I tried to complete multiple tasks simultaneously. Now I'm improving my prioritization by planning tasks better and focusing on the most critical items first."

What are your improvement areas?

I'm improving my automation skills, especially Python and CAPL, to increase efficiency in validation."

How do you handle conflicts or pressure?

I handle conflicts and pressure by staying calm, focusing on facts, and communicating clearly so that we can solve the issue quickly without affecting delivery.

Where do you see yourself in the next 3–5 years?

In the next 3–5 years, I see myself as a Lead Validation Engineer, taking ownership of system-level testing, contributing to automation, and guiding junior team members while delivering high-quality validation for complex automotive features.

Describe a challenging issue you solved in HIL. (Very common at KPIT)

Situation:
During BDC ECU HIL validation, one of the antenna signals was not getting detected by the ECU.

Task:
I needed to identify whether the issue was due to the HIL setup, wiring, configuration, BDC controller, or the ECU itself.

Action:

- Checked the I/O mapping in the HIL configuration
- Verified the CAN signals and cross-checked the DBC entries
- Reviewed panel connections and measured voltages to ensure proper signal levels
- Used the BDC controller to manually simulate the antenna input and confirm expected behavior
- Identified that the antenna signal was mapped incorrectly in the HIL I/O configuration file
- Corrected the mapping and reloaded the configuration

Result:
The antenna signal was detected correctly, testing resumed without delay, and we prevented an incorrect defect from being raised against the ECU software team.

Do you have experience interacting with clients or cross-functional teams?

Yes. I interact regularly with OEM validation teams, software developers, functional owners, and vehicle integration teams. I give daily status, defect discussions, and support software flashing activities."

Do you have any questions for us? (Very important) Ask any one of these:

1. What are the major validation activities for this project?
2. What is the automation expectation for this team?
3. How is performance measured in this project?
4. What growth or training opportunities exist at KPIT?”

1. **What are your salary expectations?**

Safe Answer:
"I'm open to a fair and competitive offer based on my experience and the company's compensation structure."

What are your salary expectations?

Safe Answer:
"I'm open to a fair and competitive offer based on my experience and the company's compensation structure."

---

3. Why should we hire you?

Answer:
"You should hire me because I bring strong hands-on experience in HIL, ECU validation, diagnostics, automation, and cross-team coordination.
I deliver reliably, take ownership, and can contribute to KPIT's validation activities from day one."

---

4. Why KPIT?

(Use your strong 2-line answer.)

---

5. Are you willing to relocate/work from client location?

Answer:
"Yes, I'm flexible and open to relocation or working from a client location if required."

---

6. Are you comfortable with shifts or weekend support if needed?

Answer:
"Yes, I am comfortable with shifts or weekend support during critical validation phases."

---

7. Do you have any questions for us? (Very Important)
"What are the automation expectations for this role?"

(This impresses them.)

STLC – Software Testing Life Cycle (Step-by-Step Explanation)

STLC is the process followed to ensure **quality** in software testing.
It includes **systematic phases**, and each phase has specific goals and deliverables.

🔶 **1. Requirement Analysis**

- Understand requirements (functional & non-functional)
- Identify testable and unclear requirements
- Interact with BA, developers, OEM teams for clarification

Output: Requirement understanding document, RTM start

🔶 **2. Test Planning**

- Define test strategy and scope
- Identify resources, timelines, and responsibilities
- Decide tools (HIL, CANoe, JIRA, automation etc.)
- Estimate effort

Output: Test Plan Document

🔶 **3. Test Case Design & Development**

- Write detailed test cases and scenarios
- Create test data
- Prepare automation scripts (Python/CAPL)
- Update RTM (Requirement Traceability Matrix)

Output: Test cases, RTM, Automation scripts

🔶 **4. Test Environment Setup**

- Prepare HIL setup or Test Bench
- Configure tools (CANoe, DBC, I/O mapping, models)
- Set up network, hardware, ECU or prototype vehicle
- Validate the setup

Output: HIL/test environment ready for execution

🔶 **5. Test Execution**

- Execute test cases (HIL, manual, automation, in-vehicle)
- Perform fault injection, regression, functional tests
- Record logs, measure signal behavior
- Compare actual vs expected results

Output: Test Results, Logs, Evidence

🔶 **6. Defect Reporting & Tracking**

- Raise defects in JIRA/Bugzilla
- Provide logs, screenshots, videos, CAN traces
- Retest defect after fix
- Participate in defect triage meetings

Output: Defect reports, retest status

🔶 **7. Test Closure**

- Ensure all tests are executed
- Confirm all critical defects are closed or agreed
- Prepare test summary report
- Evaluate quality, lessons learned

Output: Test Closure Report, Metrics, Lessons Learned

Left Side (Verification)

1. **Requirement Analysis**
   ↳ Corresponds to **User Acceptance Testing (UAT)**
   - Requirements are gathered and analyzed.
   - UAT later validates whether the final product meets these needs.
2. **System Design**
   ↳ Corresponds to **System Testing**
   - Overall architecture and system-level design are created.
   - System testing later validates complete system behavior.
3. **High-Level Design (HLD)**
   ↳ Corresponds to **Integration Testing**
   - Modules and their interactions are defined.
   - Integration testing validates communication between them.
4. **Low-Level Design (LLD)**
   ↳ Corresponds to **Unit Testing**
   - Individual components/functions are designed.
   - Unit testing validates each function/module.
5. **Coding Phase**
   - Implementation happens here.
   - Once coding is complete, testing on the right side begins.

Right Side(Validation)

- **Unit Testing** validates **Low-Level Design**
  - **Integration Testing** validates **High-Level Design**
  - **System Testing** validates **System Design**
  - **User Acceptance Testing** validates **Requirements**

CAN vs LIN Protocol – Comparison Table

Parameter	CAN (Controller Area Network)	LIN (Local Interconnect Network)
Speed	Up to **1 Mbps** (CAN-FD up to 8 Mbps)	Up to **20 kbps** (much slower)
Architecture	**Multi-master** (any node can send)	**Single-master, multiple-slave**
Use Case	Critical ECUs (Engine, ABS, BCM, BDC, ADAS)	Low-cost body functions (mirrors, windows, seats)
Cost	Higher (complex transceivers)	Very low-cost (simple hardware)
Data Size	8 bytes (CAN-FD supports 64 bytes)	8 bytes max
Communication Type	Event-triggered, arbitration-based	Time-triggered, schedule-based
Error Handling	Strong error detection & fault confinement	Basic error detection, limited fault handling
Determinism	High, based on priority	Very high, based on schedule
Cable Length	Longer (up to ~40m in vehicles)	Short (typically < 20m)
Typical ECUs	Powertrain, Chassis, ADAS, BDC, BCM	Door modules, HVAC flaps, seat motors, switches

CAN vs LIN vs CAN-FD – Comparison Table

Parameter	LIN (Local Interconnect Network)	CAN (Classical CAN)	CAN-FD (Flexible Data Rate)
Speed	Up to **20 kbps**	Up to **1 Mbps**	Up to **8 Mbps** (data phase)
Architecture	**Single-master, multi-slave**	**Multi-master**	**Multi-master**
Use Case	Low-cost body functions (mirrors, windows, seats, switches)	Medium/critical ECUs (BCM, BDC, ABS, airbags, powertrain)	High-bandwidth ECUs (ADAS, BMS, powertrain analytics)
Data Size	8 bytes max	8 bytes max	**Up to 64 bytes**
Cost	Very low	Medium	Higher (due to advanced transceivers)
Communication Type	**Time-triggered (scheduled)**	Event-triggered (priority-based arbitration)	Event-triggered + high-speed data phase
Error Handling	Basic	Strong	**Stronger with enhanced CRC**
Determinism	Very high (schedule-based)	Medium to high (priority-based)	High (fast and deterministic in data phase)
Typical ECUs	Door modules, HVAC flaps, switches, seat motors	BCM, BDC, ECM, TCM, airbags, steering, clusters	ADAS ECUs, BMS, gateways, domain controllers

Parameter	LIN (Local Interconnect Network)	CAN (Classical CAN)	CAN-FD (Flexible Data Rate)
Frame Length	Fixed	Fixed (8 bytes)	Flexible (up to 64 bytes)
Backward Compatible	No	—	**Yes, coexists with classical CAN**
Real-Time Capability	Low	Medium	High

1. What is Functional Testing?

Answer:
"Functional testing verifies that the ECU or feature works according to the requirements. It checks whether each function performs as expected under normal conditions. In HIL, this includes validating feature behavior, signal responses, timing, and state changes."

⭐ **2. What is Smoke Testing?**

Answer:
"Smoke testing is a basic health check performed after flashing new software. It ensures the ECU boots normally, communicates on CAN/LIN, and all essential functions are stable before starting detailed testing."

⭐ 3. What is Sanity Testing?

Answer:
“Sanity testing is a quick check to verify that specific functions affected by a recent change or bug fix are working properly. It confirms that the build is stable enough to proceed with further testing.”

⭐ 4. What is Regression Testing?

Answer:
“Regression testing is performed after software changes or bug fixes to ensure that previously working features have not been impacted. It validates that new updates do not introduce new defects.”

⭐ 5. What is Fault Injection Testing?

Answer:
“Fault injection testing involves intentionally creating electrical or communication faults to verify ECU robustness and diagnostic behavior. This includes short-to-battery, short-to-ground, open-load, over/under voltage, and stuck-switch conditions.”

Top Process-Related Questions (KPIT Manager Round)

🔷 1. Explain your testing process end-to-end.

Answer:
“My testing process includes requirement analysis, test case design, HIL setup/commissioning, test execution, defect logging, regression testing, report preparation, and closure. I maintain traceability from requirements to test cases and logs.”

🔷 2. Explain STLC (Software Testing Life Cycle).

Answer:
“STLC includes requirement analysis, test planning, test case design, test environment setup, test execution, defect tracking, and test closure.”

🔷 3. Explain the V-Model.

Answer:
“In the V-Model, every development phase has a corresponding testing phase — requirements with UAT, system design with system testing, HLD with integration testing, and LLD with unit testing.”

🔷 4. What is your defect life cycle? (Bug life cycle)

Answer:
“My defect cycle includes: New → Assigned → In Progress → Fixed → Retest → Verified → Closed. If not fixed, it may move to Reopen, Deferred, or Rejected.”

🔷 5. What is your test case creation process?

Answer:
“I analyze requirements, identify test conditions, write detailed steps and expected results, include positive/negative/fault cases, review them, and map them to requirements in the RTM.”

🔷 6. How do you ensure test coverage?

Answer:
“I ensure coverage through requirement review, RTM mapping, peer review, covering edge cases, and validating all functional and diagnostic requirements.”

🔷 7. How do you track requirements?

Answer:
“Using an RTM. Each requirement is mapped to one or more test cases. This ensures no requirement is missed.”

🔷 8. How do you do regression testing?

Answer:
“After a new software release or fix, I re-run previously passed test cases to ensure older features are not impacted.”

🔷 9. What is your process for defect reporting?

Answer:
“I capture logs, screenshots, or CAN traces, raise the defect in JIRA with clear steps, expected/actual results, severity, and attach supporting evidence. Then I retest after the fix.”

🔷 10. How do you decide test priority?

Answer:
“I prioritize based on criticality, safety impact, dependency with other features, and deadlines for delivery.”

🔷 11. How do you plan your daily work?

Answer:
“I track tasks using JIRA or project sheets, plan based on priority, update status in daily stand-ups, and buffer time for debugging.”

🔷 12. What is your process for handling urgent builds?

Answer:
“I perform smoke testing first, validate impacted test areas, run necessary regression, and communicate status to software and management teams.”

🔷 13. Explain HIL test execution workflow.

Answer:
“Load SW → Smoke test → Execute HIL test cases → Log results → Capture CAN logs → Raise defects → Retest after fixes → Final report.”

🔷 14. How do you maintain logs, reports, and evidence?

Answer:
“I store logs in structured folders: test cases, results, defects, CAN traces, screenshots. All are uploaded to JIRA/Confluence/shared drives for traceability.”

🔷 15. What is your approach to test closure?

Answer:
“After completing tests, ensuring all defects are closed or accepted, generating a summary report, updating metrics, and getting customer sign-off.”

Bug Life Cycle (Defect Life Cycle) – Simple Explanation

The **bug cycle** describes the **steps a defect goes through** from the moment it is found until it is closed.

⭐ Bug Life Cycle – Steps

1. **New**
    - Tester finds a bug and logs it in the tool (JIRA, Bugzilla, etc.)
2. **Assigned**
    - The defect is assigned to a developer/software engineer.
3. **In Progress / Open**
    - Developer starts analyzing and working on the fix.
4. **Fixed / Resolved**
    - Developer fixes the issue and marks it as “Fixed.”
5. **Retest**
    - Tester re-tests the issue using logs, test cases, and expected behavior.
6. **Verified**
    - If fixed correctly, tester marks it “Verified.”
7. **Closed**
    - After verification, the defect is marked “Closed.”

HIL Commissioning – Best Explanation

HIL commissioning is the process of setting up, configuring, and validating the Hardware-in-the-Loop test bench so that the ECU can communicate correctly with the HIL system and all signals behave as expected before starting actual testing.

It ensures the **HIL bench is stable, mapped correctly, and ready for validation**.

🔷 HIL Commissioning – Step-by-Step

1. Hardware Setup

- Connect ECU, load box, VT modules, harness, sensors, actuators
- Ensure correct power supply, grounding, and safety connections

2. I/O Mapping

- Map ECU pins to HIL hardware channels (analog, digital, PWM, CAN, LIN, etc.)
- Verify correct mapping inside VT System or dSPACE/ETAS configuration

3. Model Integration

- Load plant models / simulation models into HIL (Simulink models, LDF, DBC integration)
- Check model execution rate and real-time synchronization

4. Network Configuration

- Load CAN/LIN DBC, LDF files
- Verify message transmission and reception
- Check signal scaling, data types, and byte order

5. Panel / UI Setup

- Configure CANoe panels, VT GUI, or custom dashboards
- Add switches, sliders, indicators to stimulate inputs or observe outputs

6. Signal Flow Verification

- Stimulate inputs (switches, sensors) and verify ECU response
- Send outputs and check feedback loops
- Validate communication on CANoe logs
- Check analog/digital signals using oscilloscope or multimeter

7. Fault Injection Validation

- Test STB, STG, Open load, Over/Under voltage
- Verify diagnostic and fault behavior

8. Final Bench Validation

- Perform smoke test
- Run sample test cases
- Confirm that the ECU and HIL bench behave correctly
- Bench is now “commissioned” and ready for real testing

Final Notice Period Answer (Perfect for HR & Manager Round)

“My official notice period is 90 days, but my organization allows buyout. I am required to serve a minimum of 45 days, and after that I can be released earlier through buyout.”