DEPARTMENT OF MECHATRONICS ENGINEERING

Distributed Embedded Architecture

Manufacturing Testing in Embedded Systems

Manufacturing testing refers to the process of verifying that a physical device, typically produced in large quantities, functions correctly and meets the specifications before being delivered to the customer. Manufacturing testing ensures that each unit coming off the production line is free of defects and ready for deployment.

A. Key Goals of Manufacturing Testing

- **Functional Verification**: Ensure that all components of the embedded system, including sensors, processors, communication modules, and power supplies, function correctly.
- **Compliance Testing**: Verify that the product complies with industry standards and regulations (e.g., electromagnetic compatibility (EMC), safety standards).
- **Defect Detection**: Identify any manufacturing defects or issues such as soldering faults, improper component placement, or defective chips.
- **Stress and Environmental Testing**: Ensure the system operates correctly under various stress conditions like temperature, humidity, and vibrations, often encountered in real-world use.

B. Common Manufacturing Tests for Embedded Systems

1. **Board-Level Testing**
    - **In-Circuit Testing (ICT)**: This test is performed on the printed circuit board (PCB) after components are assembled. ICT checks for proper connections, soldering, and the functionality of basic components like resistors, capacitors, and integrated circuits. This is done using test probes that access various test points on the board.
    - **Boundary Scan (JTAG)**: JTAG (Joint Test Action Group) is used to test embedded systems' interconnections without physical probes. It’s useful for testing connections between chips and verifying that digital circuits behave correctly.

- **Flying Probe Testing**: This test is used to check for electrical faults on a PCB without requiring a dedicated test fixture. Flying probes move across the PCB to make contact with different test points, identifying shorts, opens, or missing components.

2. **Functional Testing**
    - **Power-On Self-Test (POST)**: After assembly, the embedded system undergoes a power-on self-test, checking that all hardware components initialize and function correctly.
    - **Hardware/Software Integration Test**: This test checks whether the embedded software can successfully interface with the hardware (e.g., sensors, actuators). It verifies the functionality of each module, such as communication protocols (I2C, SPI, CAN), peripheral devices, and processors.
    - **Burn-In Testing**: This test involves running the system at elevated temperatures and voltages for an extended period to ensure the reliability of components over time.
3. **Environmental and Stress Testing**
    - **Thermal Testing**: The system is subjected to various temperature conditions (extreme heat or cold) to ensure it operates within specified limits.
    - **Vibration and Shock Testing**: Embedded systems used in transportation or industrial environments must withstand physical stress, so they are tested with vibration and shock simulators to ensure mechanical integrity.
    - **EMC and EMI Testing**: Electromagnetic compatibility (EMC) and electromagnetic interference (EMI) tests ensure that the embedded system can operate without causing or being affected by electromagnetic noise.
4. **End-of-Line Testing**
    - Once the product has passed all earlier tests, it undergoes end-of-line testing to verify its final functionality. This typically includes a full system test where the system is powered on, and all features are exercised to simulate real-world use.

---

2. Program Validation and Testing

Program validation and testing ensure that the software running on the embedded system functions as expected and meets the desired performance, safety, and reliability requirements. This is particularly

critical for embedded systems since they often perform real-time, safety-critical tasks.

A. Key Goals of Program Validation

- **Correctness**: Ensure that the software behaves according to the design specification, without errors.
- **Performance**: Validate that the software meets real-time deadlines and operates efficiently within hardware constraints (e.g., memory, CPU usage).
- **Reliability and Robustness**: Test that the software handles unexpected inputs or conditions gracefully and maintains stable operation.
- **Safety and Security**: For systems like automotive control units or medical devices, safety-critical software must be validated rigorously to ensure it does not cause harm under any condition. Security tests ensure the software is resilient against unauthorized access or tampering.

B. Common Validation and Testing Techniques for Embedded Software

1. **Unit Testing**
    - **Definition**: Unit testing involves testing individual components or functions of the embedded software in isolation to verify their correctness.
    - **Automation**: Tools like **Ceedling** or **Google Test** are used to automate unit testing in embedded systems. These tools simulate inputs and check outputs for correctness without involving the entire system.
    - **Code Coverage**: Unit tests aim for high coverage, ensuring that most of the code paths are exercised during testing.
2. **Integration Testing**
    - **Definition**: Integration testing checks how different software modules interact with each other and the hardware. It ensures that the modules communicate correctly, even under boundary conditions.
    - **Hardware/Software Integration**: For embedded systems, integration testing often involves validating the software with real hardware or using **Hardware-in-the-Loop (HIL)** simulations.
    - **Driver Testing**: Testing low-level drivers (e.g., for sensors or communication buses) is crucial since these drivers act as the interface between the hardware and the rest of the

software stack.

3. **System Testing**
    - **Definition**: System testing validates the embedded software as a whole, ensuring that it meets all the functional requirements. This is usually performed on the final hardware platform.
    - **Real-Time Testing**: For real-time systems, system testing includes checking that the software meets timing constraints, such as responding to sensor data within specified deadlines.
4. **Regression Testing**
    - **Definition**: Regression testing ensures that changes made to the software (e.g., bug fixes or updates) do not introduce new errors. Every time a change is made, a suite of tests is run to ensure that all previous functionality still works as expected.
    - **Automation**: Regression tests are often automated and run frequently during the development cycle to catch any bugs early in the process.
5. **Code Verification**
    - **Static Analysis**: Tools like **Polyspace**, **Coccinelle**, and **Coverity** are used to perform static analysis, checking the code for potential runtime errors (e.g., memory leaks, buffer overflows) without executing it. This is important for embedded systems where manual debugging may be difficult.
    - **Dynamic Analysis**: Dynamic analysis tools like **Valgrind** check for errors such as memory leaks or incorrect memory access during program execution.
6. **Stress and Load Testing**
    - **Definition**: Stress testing pushes the embedded software beyond its normal operational limits (e.g., high data rates or excessive inputs) to ensure that it can recover gracefully from overloads.
    - **Timing Constraints**: Real-time embedded systems are also subjected to load tests to ensure that they meet deadlines under heavy workloads or stress conditions.
7. **Formal Verification (for Safety-Critical Systems)**
    - **Definition**: Formal verification uses mathematical methods to prove that the software

meets its specification, ensuring no corner cases are missed.

- **Use in Safety-Critical Systems**: Formal verification is used in industries like automotive (ISO 26262), avionics (DO-178C), and medical devices to provide the highest assurance that the software is free of critical bugs.

8. **Field Testing**
    - **Definition**: Once the software passes lab testing, it is deployed in real-world conditions for field testing. The system operates in its intended environment (e.g., a car, factory, or medical device) to observe how it performs under actual use.
    - **Bug Reporting**: Any bugs or issues that arise during field testing are recorded for further debugging and validation before the final release.