TWENTY-SEVENTH ANNUAL

TestConX™

Archive

DoubleTree by Hilton
Mesa, Arizona
March 1-4, 2026

© 2026 TestConX - Image: iStock-487117740

External Program Testing

Max Tenuta
Micro Control Company

Mesa, Arizona • March 1–4, 2026

Contents

- Overview
- Traditional Test Model (Comparison to Traditional Test Steps)
- External Program Concept
- Calling Scripts & Passing Parameters
- System Context Packet / JSON Input-Output Model
- Applications & Use-Cases
- Interfacing, Debugging & Hardware Control
- Summary

Traditional Burn-In Test Programs

Burn-in programs use a fixed, deterministic structure:

- Sequential test steps with defined power and temperature settings
- Each step runs pre-generated vector patterns on the burn-in board (BIB)
- Pass/fail determined by device output states at specific cycles
- Final disposition follows preassigned criteria defined before the run

Limitation:

This method is static — every device under test (DUT) is evaluated the same way, with no ability to adapt test behavior based on device response.

External Programs: A Dynamic Alternative

External Programs enable flexible, script-driven testing:

- Users can run custom Python scripts instead of fixed vectors
- Scripts communicate with the DUT via JTAG, SPI, I2C, etc.
- Hardware access provided through either a tester API or direct I/O control
- Scripts can be tailored to each board, using libraries or device-specific routines

External Programs Continued

Hybrid operation:

- External programs can run alongside traditional vector steps
- Scripts may condition hardware before deterministic tests
- Enables adaptive, board-aware, device-aware test flows

Benefit:

A more dynamic testing environment that responds to the DUT and supports complex, functional operations.

Figure 1 – Simplified Test Program Flowchart

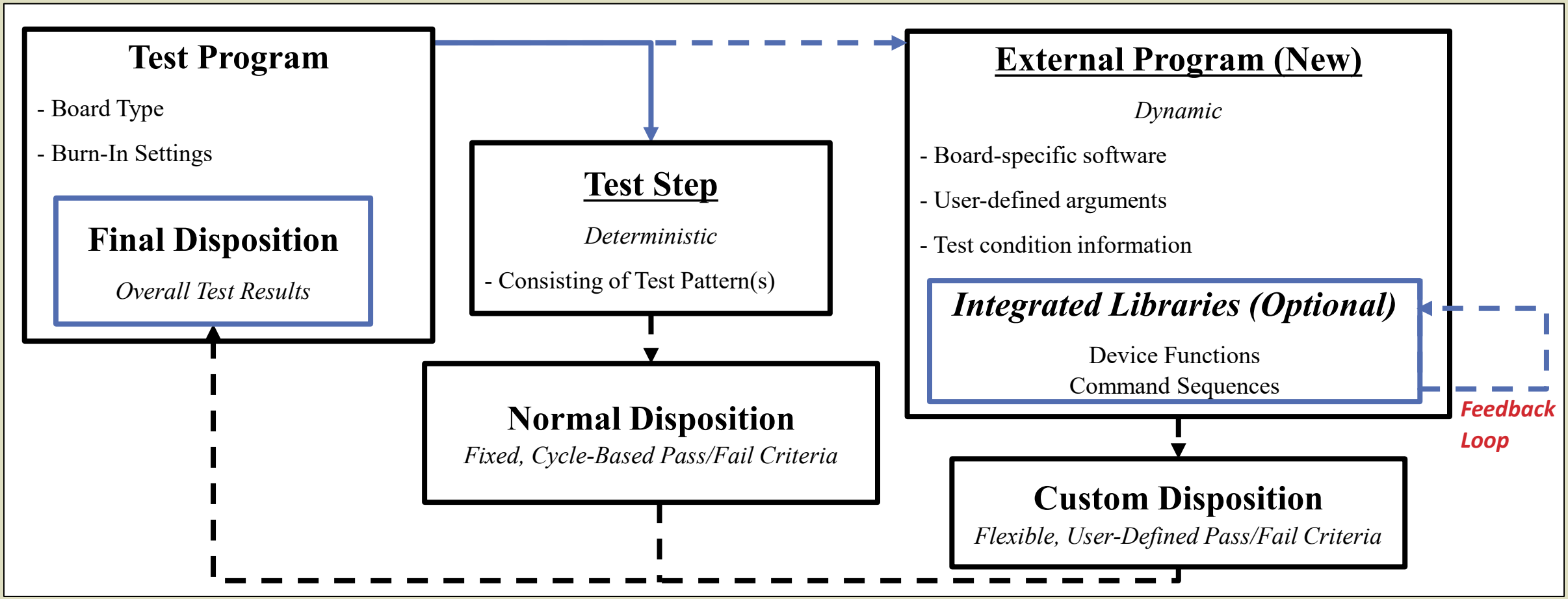


Figure 1 demonstrates the general flow of these two modes of operation, and where the two approaches diverge.

User Arguments & Script Configuration

External programs can be customized through user-defined arguments:

- Select which I/O group, chiplet, or interface to target
- Choose which functions or routines to run on those pins
- Enable debugging levels, logging formats, or verbose printouts
- Specify test modes (e.g., sequential vs. parallel DUT testing)
- Add additional flags to streamline or expand script behavior

Key Point:

Each script decides how many arguments to accept and what functionality they unlock.

System-Provided Context Packet

Alongside user arguments, the system supplies a structured data packet giving the script:

- Current run number and test step name
- The BIB's active DUT positions
- Number of total and populated sockets
- Metadata for logging and correlation with the run
- Output file location or return channel information

System-Provided Context Packet Continued

This allows scripts to:

- Test only populated sockets and tailor behavior per
- Coordinate interactions between DUT positions
- Log results accurately and consistently

Return value:

A similar packet where the script reports pass/fail per DUT, using criteria defined by the script designer.

End-to-End Data Exchange

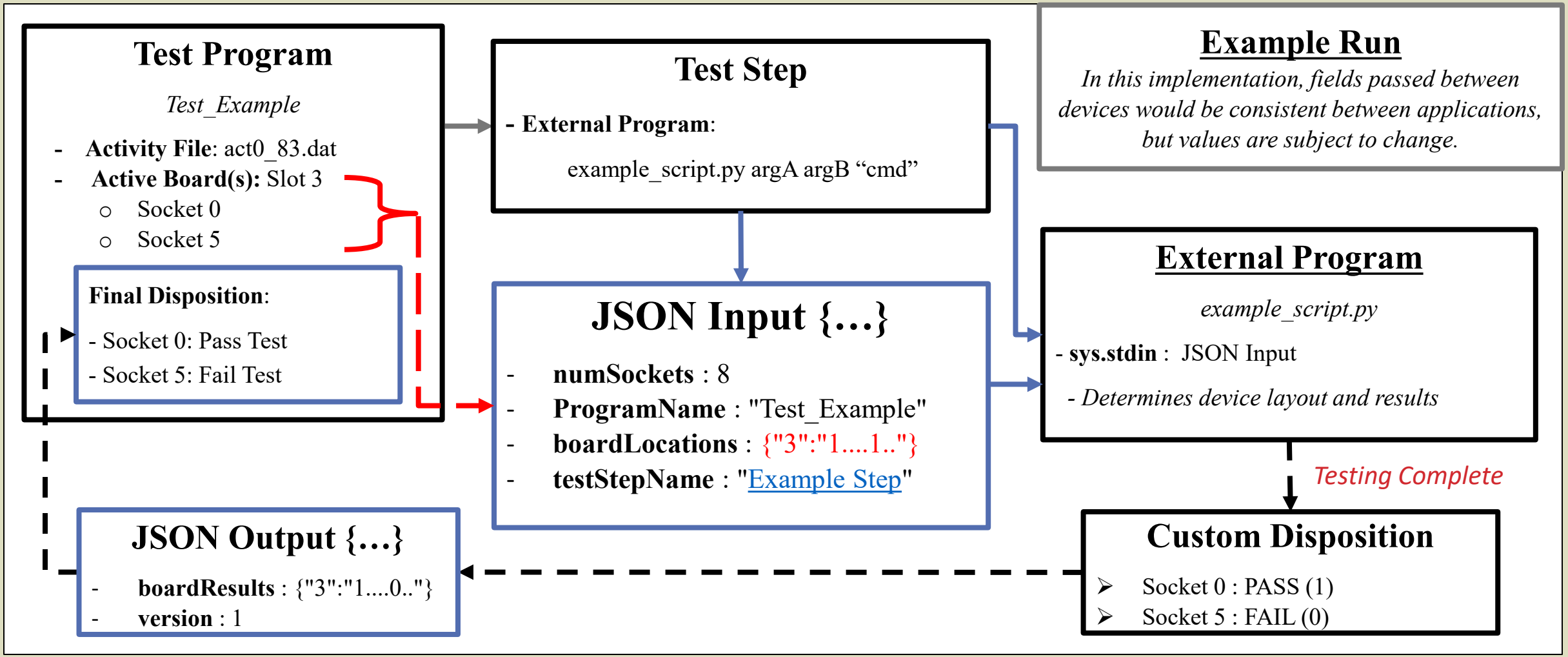
How the exchange works:

1. Burn-in software sends a JSON packet with test conditions
2. Python script receives the packet and parses system + user arguments
3. Script executes its defined test routines
4. Script returns results in a JSON output packet
5. Burn-in software reads the output and applies pass/fail disposition

Key Point:

This flow cleanly separates system context, user options, and script logic, enabling flexible, board-aware testing.

Figure 2 – Program Call Example Exchange



Functional & Dynamic Applications of External Programs

External Programs extend burn-in with SLT-like capabilities:

- Bring functional device testing into the high-stress burn-in environment
- Scripts can use existing device APIs, vendor DLLs, and ATE libraries
- Enable engineers to reuse familiar tools without needing burn-in expertise
- Allow interaction with the DUT's on-board functions, not just digital vectors
- Support flexible, board-specific routines for more meaningful device validation

Benefit:

Burn-in becomes capable of deeper, function-aware testing.

Expanded Interfacing & Debugging Capabilities

Adaptive & Device-Specific Test Flows

- Scripts react to results and automatically choose follow-up sequences
- Can run pattern variations to help diagnose failures
- Provide insight into why a device failed, not just whether it failed
- One script can support multiple device models on the same board
- Patterns can be tailored per DUT, including DUT-to-DUT interactions
- Scripts can adjust board hardware dynamically to optimize test conditions

Benefit:

Test programs become adaptive, individualized, and capable of coordinating across devices.

Example #1 – Adapting Libraries

Goal:

- *Customer A* wants to streamline the process of testing different IPs and protocols on their DUT, without generating patterns for each variation.

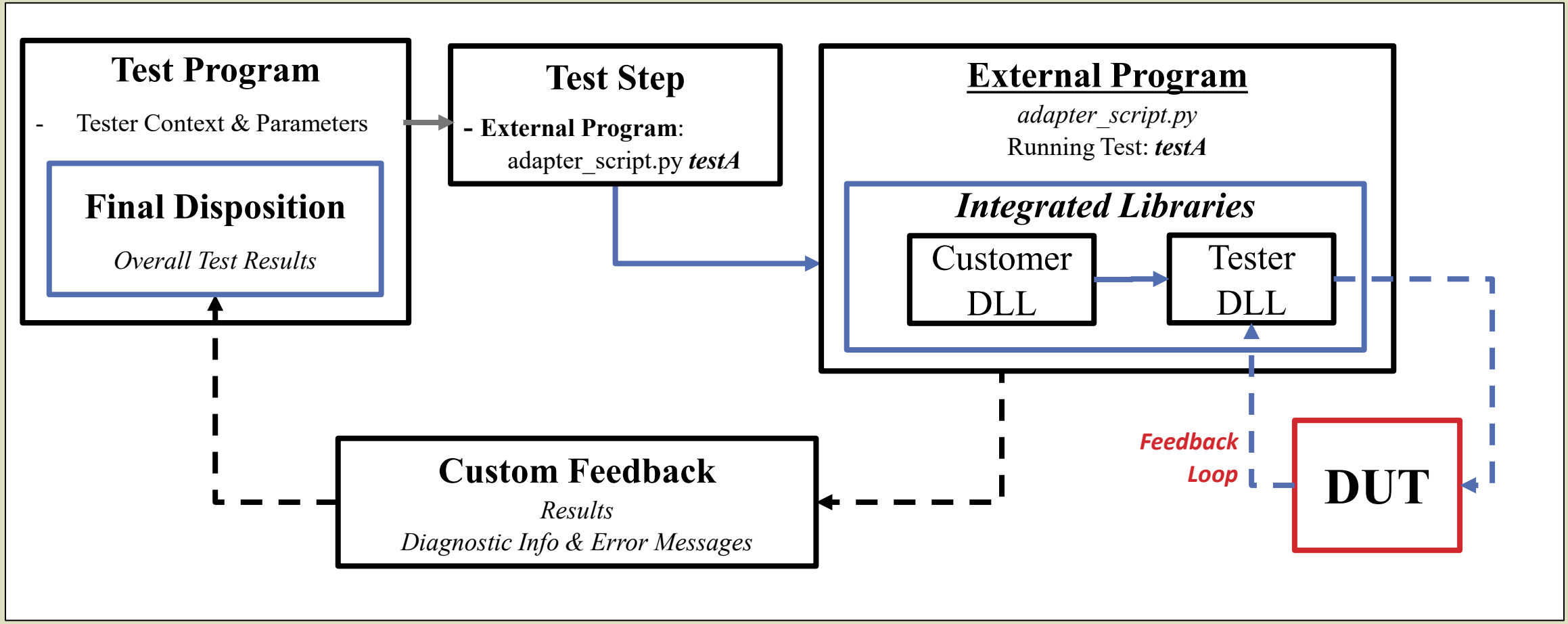
Prerequisites:

- *Customer A* has access to an existing dynamic library of commands from another system type, and wants to repurpose this in burn-in.

Solution:

- Create an additional library adapting customer's command structure to burn-in platform.
- External program script is written to mount both libraries, issue commands based on user arguments, and interface results between DUT and test program.
- Pass/Fail results, error codes, and diagnostic information are printed directly to user.

Figure 3 – Adapting Libraries Example



Interactive Device Access & Remote Debugging

Expanded Troubleshooting Capabilities

- Scripts can access on-chip debug features of the DUT
- Valuable in pre-production for refining test programs
- Useful in pre-production to refine test programs
- Engineers can observe device behavior in real time, not just through fixed patterns
- Provides deeper insight into performance, configuration, and failure causes

Core Idea:

Burn-in can support hands-on, exploratory debugging, not just automated sequences.

Hardware-Level Control & Tool Integration

External Programs can provide direct access to the DUT:

- Commands can be sent interactively using built-in or vendor libraries
- Scripts may expose remote access via SSH, Telnet, or similar protocols
- Users can issue native device commands and view immediate feedback
- Scripts without vendor libraries can still support call-and-response modes
- Real-time results help identify the best test modes, timing, and operating parameters

Benefit:

Interactive testing improves early-stage development and informs the final burn-in configuration.

Hardware-Level Control & External Tools

Hardware-Level Control & External Tools

- External Programs can integrate with external tools (debug pods, analyzers, meters)
- Scripts can monitor and drive buses like JTAG, SPI, I2C. They can directly control board hardware: switches, muxes, routing, clocks
- Example: dynamically rerouting JTAG pins for different device areas
- Supports feedback loops, timing adjustments, and custom board configurations

Benefit:

Enables highly tailored, device-specific test flows and deeper validation.

Example #2 – Dynamic Gating

Goal:

- *Customer B* wants to support a range of DUTs with varying characteristics on a single burn-in board
- Need to determining on mid-test if certain signals or supplies should be connected or disconnected

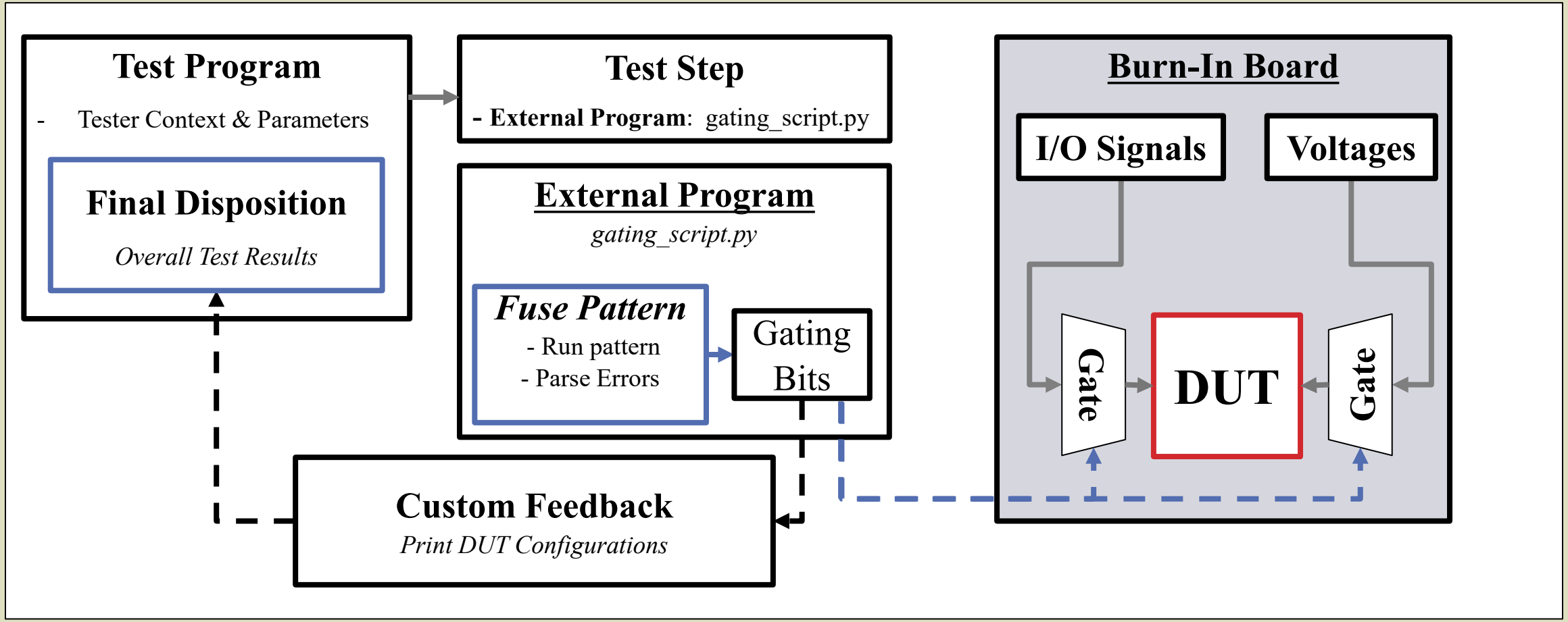
Prerequisites:

- Burn-in board must be designed to allow manual gating on a per-DUT basis
- Need a test pattern exists that can characterize the device, with specific cycles identifying which signals or supplies should be present

Solution:

Create an external program that executes the characterizing pattern, parses out which bits are toggled for which DUTs, and conditions the boards' gating circuits accordingly.

Figure 4 – Dynamic Gating Example



Summary

- External Programs introduce dynamic, script-driven burn-in testing
- Test behavior can adapt to device feedback in real time
- JSON-based context packets enable board-aware, DUT-specific operation
- Scripts can integrate device libraries, external tools, or on-chip debug features
- Supports advanced use-cases, multi-device boards, and hardware-level control

Key Takeaway:

External Programs bring flexibility, intelligence, and deeper visibility to burn-in test flows.

COPYRIGHT NOTICE

The presentation(s) / poster(s) in this publication comprise the Proceedings of the TestConX 2026 workshop. The content reflects the opinion of the authors and their respective companies. They are reproduced here as they were presented at the TestConX 2026 workshop. This version of the presentation or poster may differ from the version that was distributed at or prior to the TestConX 2026 workshop.

The inclusion of the presentations/posters in this publication does not constitute an endorsement by TestConX or the workshop's sponsors. There is NO copyright protection claimed on the presentation/poster content by TestConX. However, each presentation / poster is the work of the authors and their respective companies: as such, it is strongly encouraged that any use reflect proper acknowledgement to the appropriate source. Any questions regarding the use of any materials presented should be directed to the author(s) or their companies.

"TestConX", the TestConX logo, the TestConX China logo, and the TestConX Korea logo are trademarks of TestConX. All rights reserved.

www.testconx.org