



a silicon to systems solutions company

Tvastaa VP

(twa - us - thaw - V.P)

Agentic AI for Virtual Platform Development



D-Globalist Award
2025



Inspire Award 2025
Delivery Excellence in Tech Services

About Vayavya Labs

Corporate Snapshot

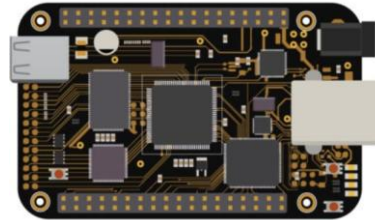
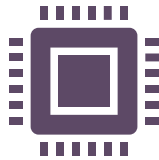
19 Years



India
US
Japan
Europe

Our Offerings

“Silicon to System” Engineering Solutions



Semiconductor/ESL Offerings from Vayavya

- Performance Modeling
- HSI Consulting

- Software Driven Verification
 - Bare metal/Linux
 - Portable Stimulus
- Emulation/FPGA

- S.W. bring-up on silicon
- Productizing of S.W. stack

Specification

Architecture
Exploration

RTL Design &
Verification

Physical Design

S.W. Bring up

- End-to-end Virtual platform development
- QEMU, SystemC, C/C++ models, ...

- BSP, device drivers, Protocol Stacks
- GPU, PCIe, CXL, UCle, ETH, NVMe, ...
- Linux, Android, ...

Virtual platform
Development

Software
Development

Agenda

- Introduction
- Design and Plan
- Stub Code Generation
- Code to model IP
- Impact
- Roadmap
- Case Studies

Introduction

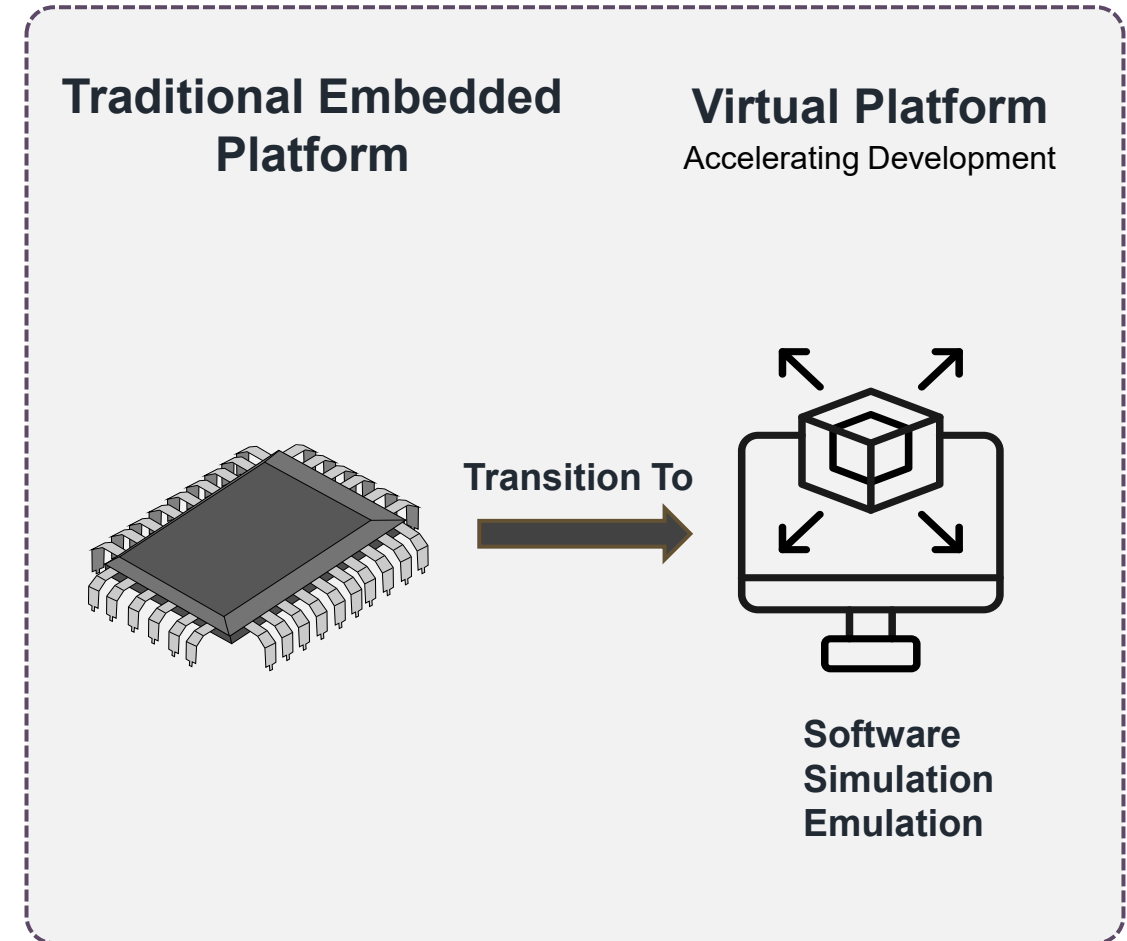
Agentic approach through Tvastaa VP

20min

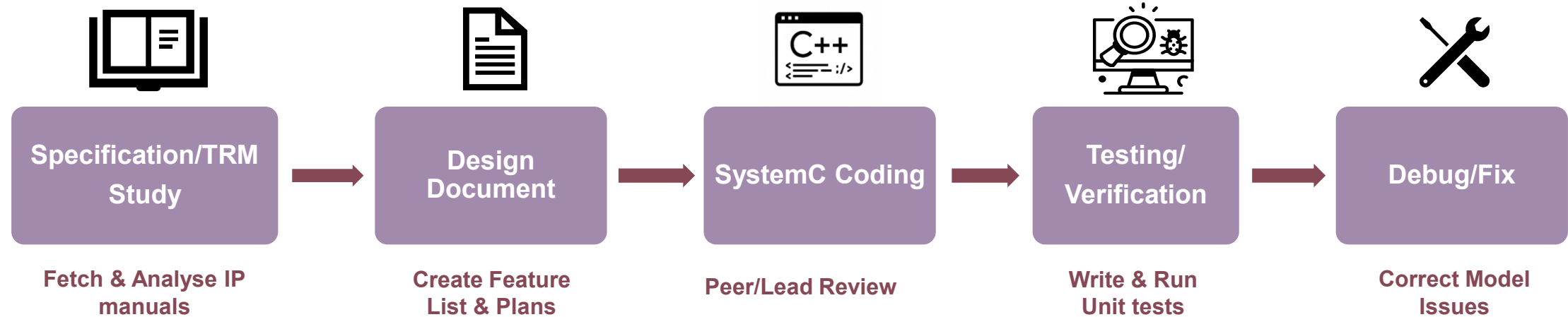


Virtual Platforms

- Virtual Platforms have moved from experimentation to production
- Development and “productizing” of Virtual Platforms
 - Time & Resource consuming – often acts as a barrier to adoption
 - Needs to hit the usage “sweet spot”
 - SystemC and C/C++ modelling
 - A niche skill
 - Small developer base i.e. lack of a wider talent pool



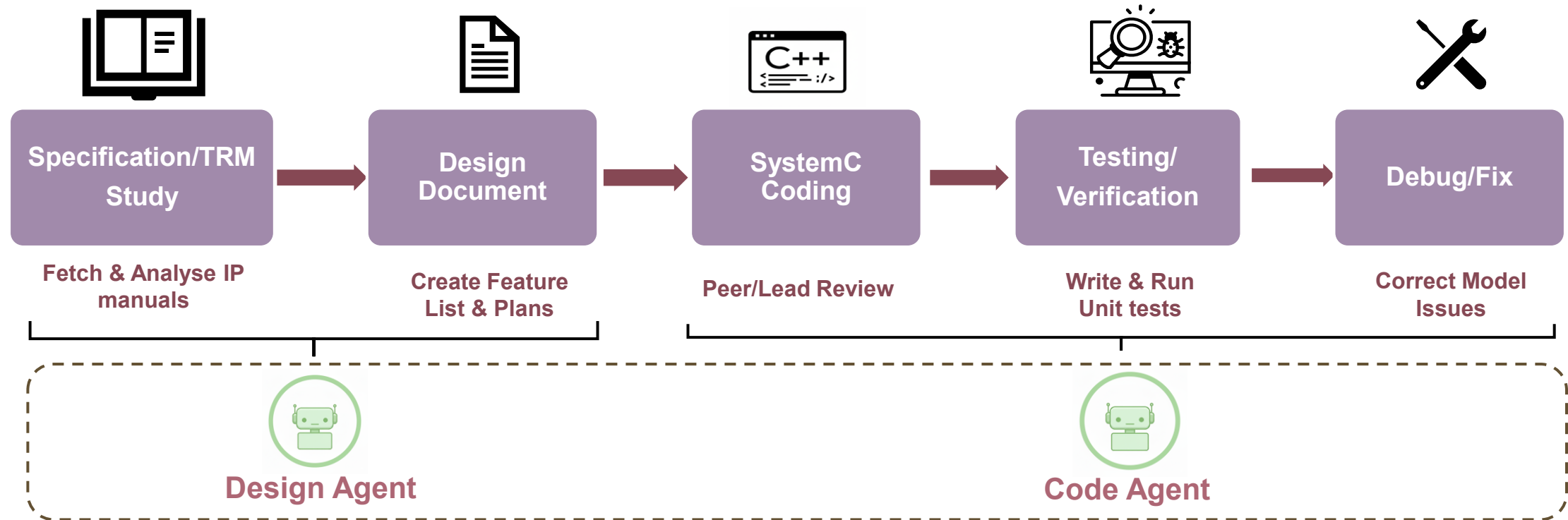
Traditional SystemC Workflow: Key Pain Points



- Complete Spec understanding is a slow process
- Requires deep IP knowledge and SystemC expertise
- Skilled SystemC engineers are scarce
- High manual effort - coding, testing and debugging
- Hard to maintain design intent and code consistency
- Long cycle from spec to usable virtual prototype

Agentic AI approach addresses these challenges comfortably

How Agentic AI Approach Helps...

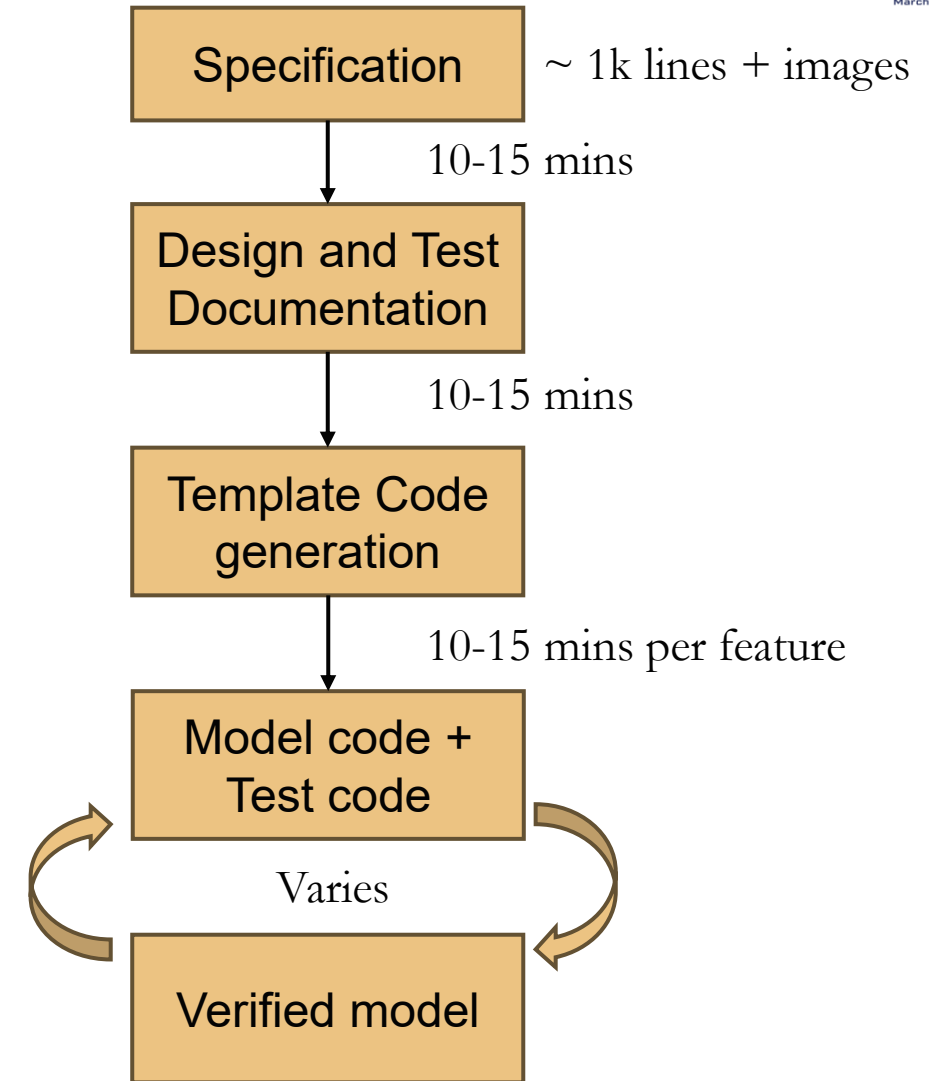


- ✓ Faster spec-to-model turnaround
- ✓ Reduced dependence on SystemC human experts
- ✓ Reduced design and documentation efforts
- ✓ Consistent code quality across engineers
- ✓ Automated test generation and debugging
- ✓ Faster iteration and prototype availability

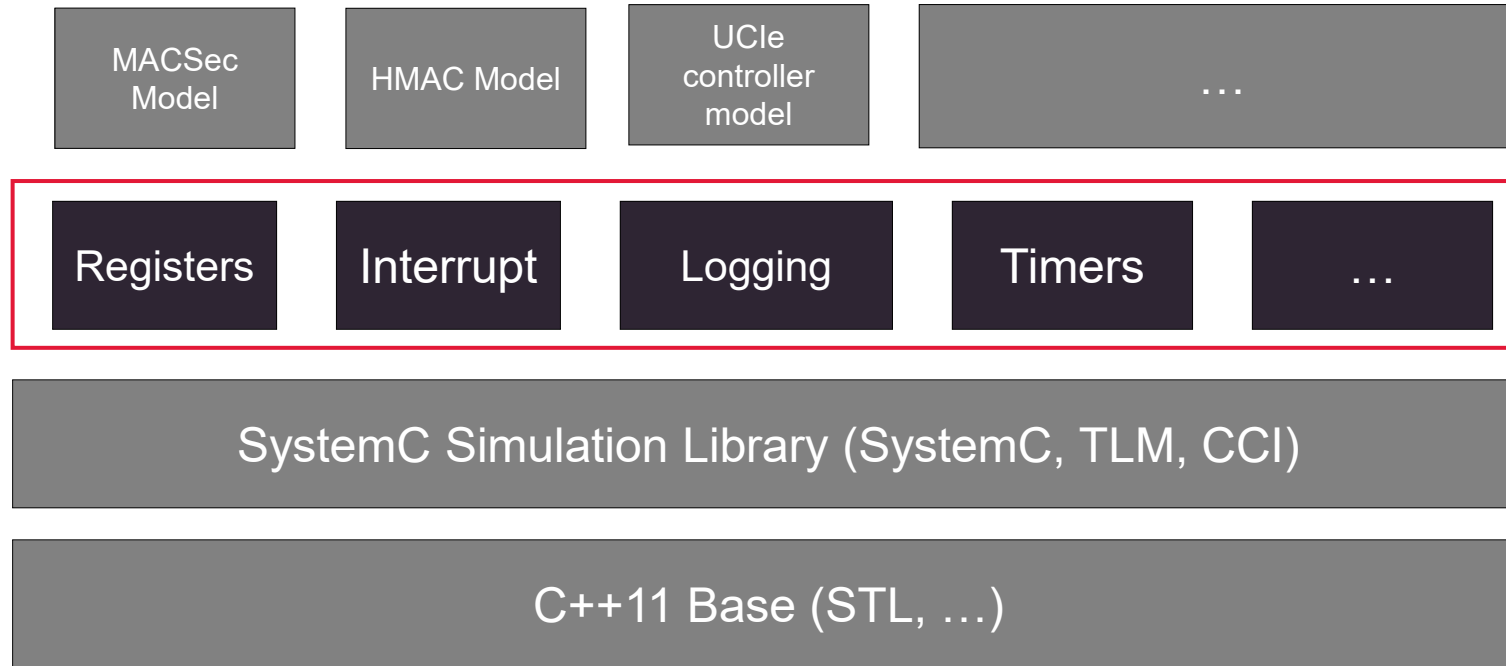
	L0 No Automation	L1 Driver Assistance	L2 Partial Automation	L3 Conditional Automation	L4 High Automation	L5 Full Automation
DRIVER	 In charge of all the driving	 Vibe Coding	 Agentic AI, human-in-the-loop	 Conditional Autonomous Engineering unable to continue	 Can be a passenger who, with notice, can take over driving when the self-driving systems are unable to continue	 No human driver required—steering wheel optional—everyone can be a passenger in an L5 vehicle
VEHICLE	Responds only to inputs from the driver, but can provide warnings about the environment 	Can provide basic help, such as automatic emergency braking or lane keep support 	Can automatically steer, accelerate, and brake in limited situations 	Can take full control over steering, acceleration, and braking under certain conditions 	Can assume all driving tasks under nearly all conditions without any driver attention 	In charge of all the driving and can operate in all environments without need for human intervention 

Our Demo today: HMAC

- SHA-2 hash-based authentication code generator
- Two modes: SHA-2 | HMAC based on SHA-2
- Multiple digest sizes supported (for both modes): SHA-2 256/384/512 hashing algorithm
- Configurable key length up to 1024-bit secret key for HMAC mode
- Support for context switching (via saving and restoring) across multiple message streams
- 32 x 32-bit message FIFO buffer



CSML: A Permissively licensed Modeling Library



- Fixes the missing features in SystemC for real-life peripheral models
- Vendor neutral
 - Supports Accellera Reference simulator
 - Supports commercial SystemC simulators
- Permissively licensed

“Tvastaa VP” – Our Agentic AI Tool

Before we see what Tvastaa VP is...



Vayavya Labs's Journey with LLMs

Genesis: Why We Looked at LLMs

- Despite good resource pool, we faced:
 - Growing customer demand across multiple SoCs/IPs
 - Long spec-to-model cycles
- Scaling talent linearly was not sustainable
- Started exploring how LLM could be leveraged for modeling workflows

From Exploration to Structured Adoption

- Began applying LLMs to SystemC modelling when they gained traction
- Two years of internal experimentation
- Presented work at DVCon U.S. 2024

DVCon U.S. 2024: Proof of Execution

- LLM-assisted SystemC Modeling with Qux constructs
- Prompt-driven, single-agent workflow
- ~20% productivity improvement

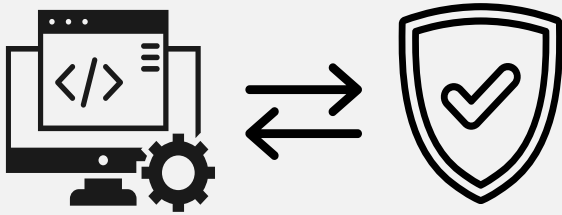
DVCon U.S. 2024 - Program Grid 7 March 2024

TIME (PST)	Cascade	Donner	Siskiyou
9:00 – 10:30	Data-Driven Design, Verification, Validation, and Signoff Case Studies of RISC-V SoCs SIEMENS	Emulation Moves Into 4-State Logic and Real Number Modeling cā dence	SystemC Model Code Generation using Large Language Models Vayavya Labs
10:30 – 11:00	Coffee Break (Bayshore Foyer)		
11:00 – 12:30	Data-Driven Design, Verification, Validation, and Signoff Case Studies of RISC-V SoCs (continued) SIEMENS	FPGA Prototyping for Large Multi-Die / Multi-Core Designs SYNOPSYS	Advanced UCle-based Chiplets verification from IP to SoC cā dence
12:30 – 13:30	Lunch Sponsored by: SYNOPSYS (Sierra)		
13:30 – 15:00	Expanding role of Static Signoff in Verification Coverage REAL INTENT	RISC-V Core Verification: A New Normal in Verification Techniques BREKER	Automating the Integration Workflow with IP Centric Design PERFORCE

What changed in 2025: The Agentic Leap

- Shift from manual modelling to Agent-driven tasks
- Domain specific Agentic workflows for Virtual Platforms (Spec ingestion, model planning, design docs, code and test generation)
- Covers the complete modelling lifecycle:
HW IP Specifications → Verified SystemC TLM Model
- ~80% automation across lifecycle
- Human role moves from coding to supervision and approval
- Zero manual SystemC authoring

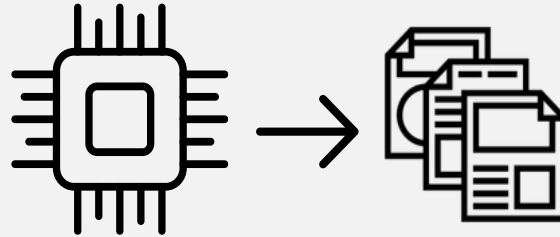
Tvastaa Agentic Platform Family



Tvastaa Janus

Code & System Compliance Automation

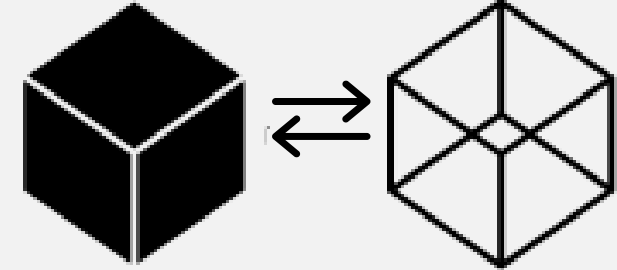
- Automates safety and security compliance workflows
- Identifies, triages, and tracks violations
- Aligns with real development and review cycles
- Reduces manual compliance effort



Tvastaa Driver

Device Driver & HSI Intelligence

- Intelligent device driver generation
- Translates HW specs into OS-ready software artifacts
- Automates HSI workflows
- Reduces integration surprises

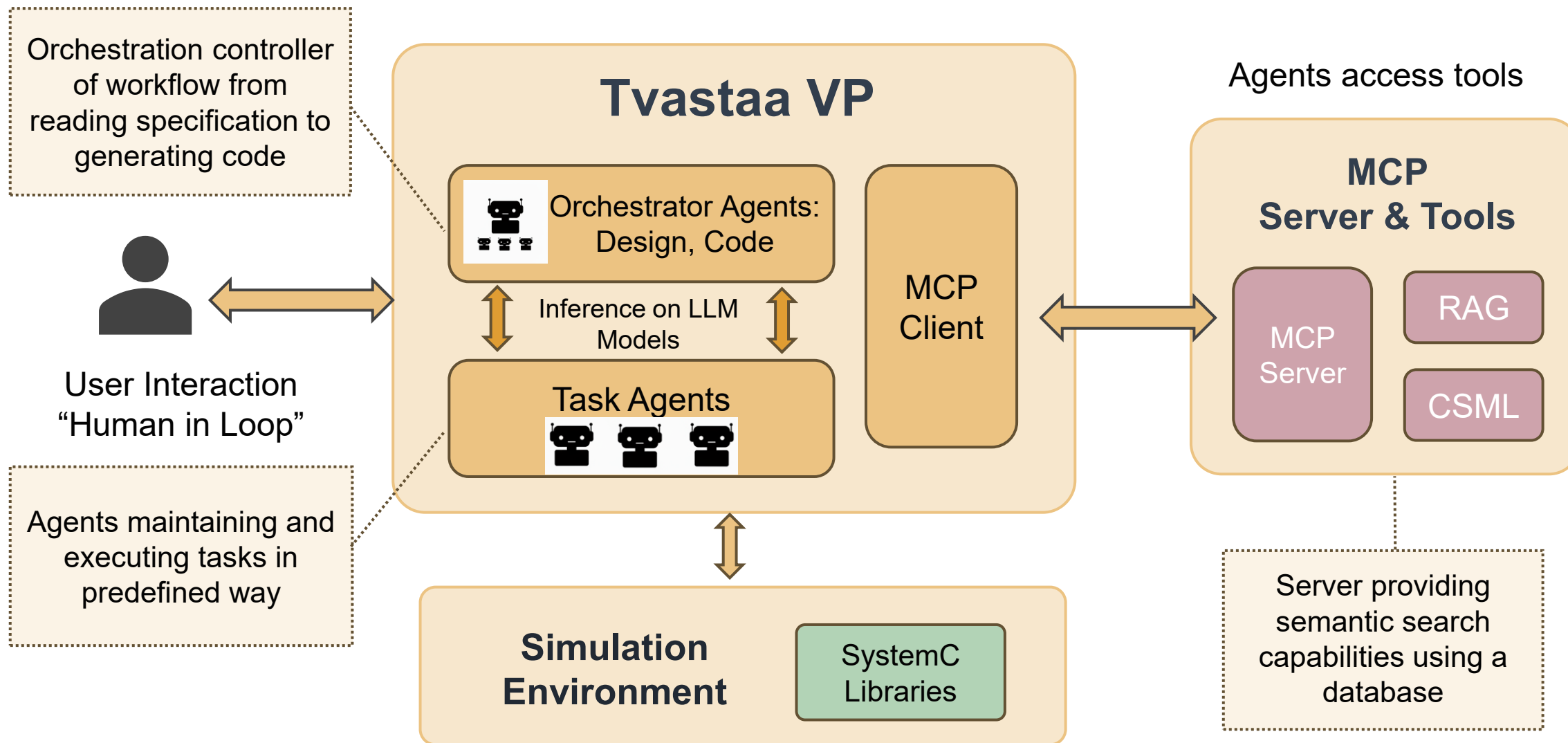


Tvastaa VP

Digital Twins for IPs & SoCs

- Generates digital twins for IPs and SoCs
- Enables early software development
- Accelerates pre-silicon validation
- Reduces dependency on physical hardware

Tvastaa VP High-Level



Pre-Processing

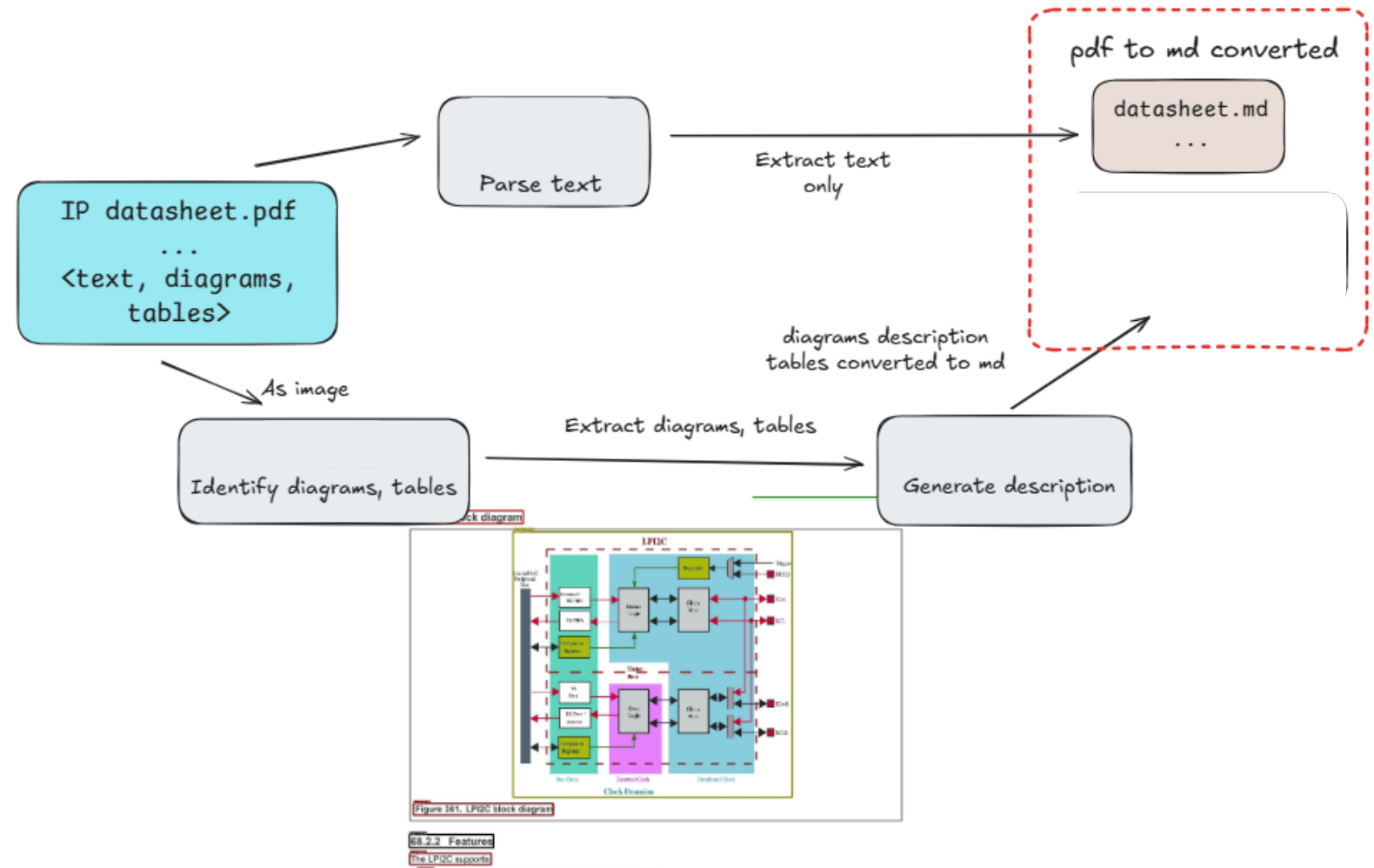
Prepare Input Data

40min



Preprocessing – “X” to Markdown

- LLMs work with text (tokens), all relevant information has to be converted to text
- IP Information is typically present as
 - PDF Datasheets, Technical Reference manuals
 - IP-XACT XML
 - XLS files
 - Text and diagrammatic representation
- Pre-processing ingests all data and generates markdown output



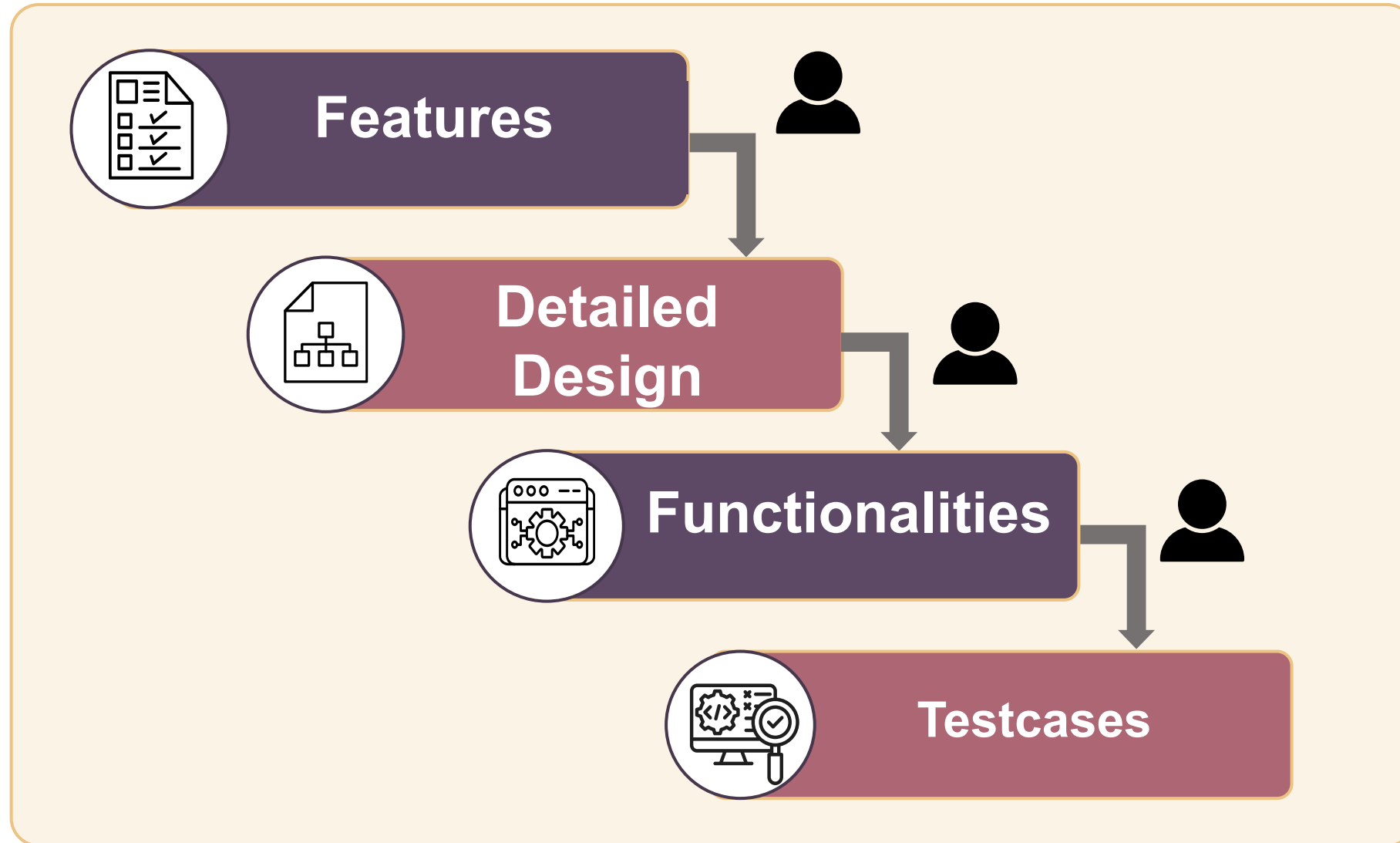
Design

Plan for IP functionality implementation

40min



Tvastaa VP – Plan and Design



Demo – Hash-based Message Authentication Code (HMAC)



Design phase – Snapshots (1/2)

- **Tasks** • Design-Orchestrator
 - 1 in progress • 12 pending
 - Creating directory structure and empty CLAUDE.md
 - o Extract features for HMAC IP
 - o Extract configuration parameters
 - o Extract port interfaces
 - o Extract memory-mapped registers
 - o Extract register callbacks
 - o Extract modeling assumptions
 - o Generate high-level design document
 - o Generate detailed design document
 - o Generate hardware behaviour documentation
 - o Generate test plan
 - o Generate functionality list
 - o Generate test case list

```
Subagent needs your input!

Context: I have analyzed the HMAC IP documentation from the knowledge base and classified all features according to TLM modeling requirements. The IS category contains features that affect software behavior and system-level functionality, while IS NOT contains implementation details and non-software-visible characteristics.

? Question: I have completed the feature extraction and classification for the HMAC IP. Here is the summary:

IS Features (MUST Model in TLM): 9 categories, 31 total features

• Operational Modes (2 features)
• Cryptographic Key Configuration (2 features)
• Message Processing (4 features)
• Interrupt Generation (3 features)
• Error Conditions (5 features)
• Context Switching Support (3 features)
• FIFO Management (3 features)
• Security Features (2 features)
• Alerts (1 feature)

IS NOT Features (EXCLUDE from TLM): 6 categories, 11 total features

• Implementation-Specific Performance (3 features)
• RTL-Level Implementation Details (5 features)
• Side-Channel and Security Hardening (2 features)
• Integration and Future Features (3 features)
• Padding Logic Implementation (2 features)
• Data Path Timing (2 features)

The classification follows the SystemC TLM modeling guidelines, focusing on software-visible features, interrupt generation, error handling, and functional behavior while excluding RTL-level timing details, implementation specifics, and non-functional attributes.

Do you approve this classification, or would you like me to make any adjustments?

You: For model cryptographic operations, use 3rd party libraries, for example OpenSSL. Rest all classifications are approved.
```

Design phase – Snapshots (2/2)

```
Tasks · design-config-params
3 completed · 1 in progress · 9 pending
✓ Create directory structure and empty CLAUDE.md
✓ Extract features for HMAC IP
✓ Extract configuration parameters
● Extracting port interfaces
  ○ Extract memory-mapped registers
  ○ Extract register callbacks
  ○ Extract modeling assumptions
  ○ Generate high-level design document
  ○ Generate detailed design document
  ○ Generate hardware behaviour documentation
  ○ Generate test plan
  ○ Generate functionality list
  ○ Generate test case list
```

Subagent needs your input!

Context: I've completed the analysis of HMAC documentation from the knowledge base and extracted all port interface information based on the "Is" features list. The port interfaces follow SystemC TLM-2.0 conventions suitable for virtual platform modeling.

? Question: I have analyzed the HMAC IP documentation and generated the SystemC TLM port interfaces. Here is the summary:

Port Interfaces Identified:

Register Bus (1 port):

- bus (tlm_target_socket<32>) - TL-UL register interface for all configuration and control

Interrupts (4 ports):

- intr_hmac_done (sc_out) - Digest computation complete
- intr_fifo_empty (sc_out) - Message FIFO empty
- intr_hmac_err (sc_out) - Error condition detected
- alert_fatal_fault (sc_out) - TL-UL bus integrity fault alert

Clock & Reset (2 ports):

- clk_i (sc_in) - Functional clock frequency
- rst_ni (sc_in) - Active-low reset

Key Observations:

- 1 HMAC is a cryptographic accelerator IP with no external communication protocol interfaces
- 2 No DMA request signals (documentation mentions DMA compatibility but no dedicated DMA request outputs)
- 3 No bus master capability (all operations are register-controlled)
- 4 All features are software-visible through the register interface

Total: 7 ports organized in the standard categories (Register Bus, Interrupt, Clock, Reset)

Does this summary look correct? Should I proceed to save the port interfaces document?

You:

Stub/Template Code

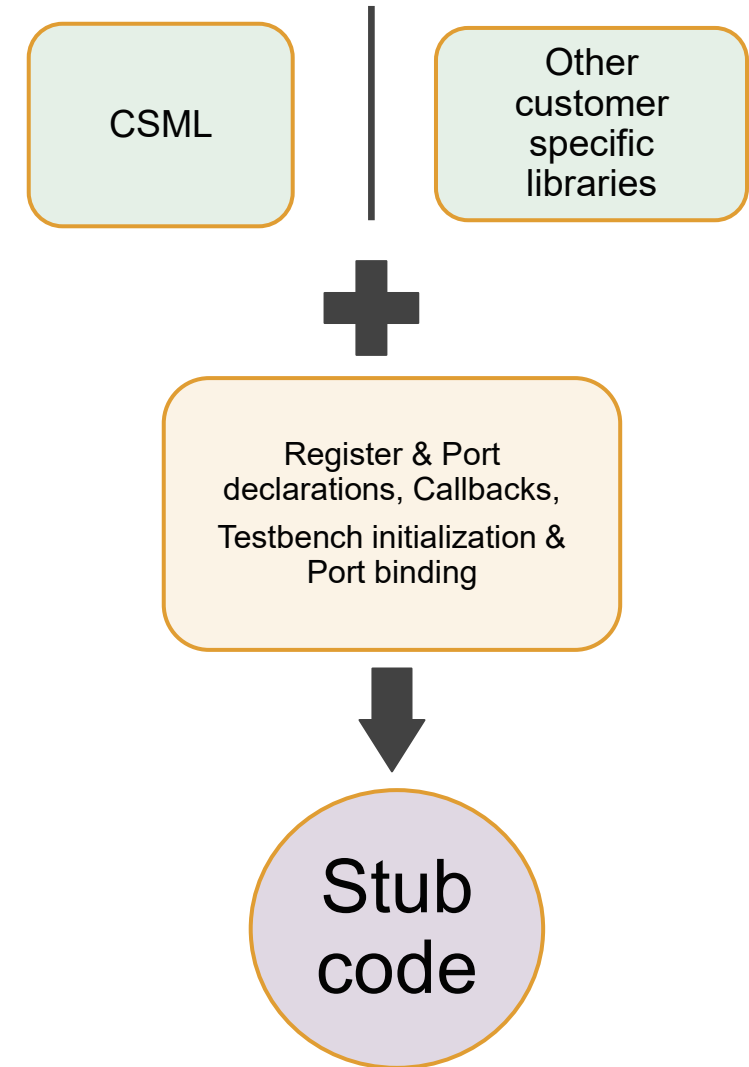
Initial code generation

15min



Tvastaa VP – Stub/Template Phase

- Uses tools like CSML or any other customer specific libraries to generate initial code
- Generate initial code for given IP
 - Register declaration
 - Register callbacks
 - Port declarations
 - Initialization of testbench and port bindings
- Executes CSML tool to generate initial code



Demo – Stub Phase



Stub phase – Snapshots (1/2)

Step 1: Implement CSML Code

I'm ready to invoke the csml-csv-generator subagent to:

- Query the knowledge base for HMAC IP
- Generate the register map CSV file at: `hmac/hmac-generated-docs/sections/hmac-register-map.csv`

Please confirm: Should I proceed with generating the CSML CSV register map for HMAC?

 You:

- Tasks · bind-ports
 - 1 in progress · 10 pending
 - Analyzing existing SystemC model structure and port interface requirements
 - Create hmac_interface.h with abstract interface class
 - Implement port declarations in hmac.h (model header)
 - Implement hmac.cpp with control and data logic
 - Implement hmac_register.cpp with register callbacks
 - Implement hmac_bus.cpp with TLM transport logic
 - Implement hmac_test.h and hmac_test.cpp with test interface
 - Create testbench.h and testbench.cpp with port binding
 - Create Makefile for compilation
 - Compile and fix compilation errors
 - Run tests and validate functionality



Stub phase – Snapshots (2/2)

```
Subagent needs your input!

Context: The CSV follows CSML format specifications with proper headers, register types, field definitions, and array structures. The register map is extracted from hmac-memory-map-registers.md and includes all functional requirements.

? Question: I've generated a comprehensive HMAC register map CSV file based on the documentation. Here's a summary:

CSV File Created: /home/ashwinp/basavarajh/demo/feb_19/tvastaavp_dvcon/hmac/hmac-generated-docs/sections/hmac-register-map.csv

Summary of Registers:

• INTR_STATE (0x0): Interrupt state with WIC clearing for hmac_done and hmac_err
• INTR_ENABLE (0x4): Interrupt enable control
• INTR_TEST (0x8): Interrupt test register
• ALERT_TEST (0xC): Fatal alert test
• CFG (0x10): Configuration register with digest_size, key_length, and swap controls
• CMD (0x14): Command register for hash operations
• STATUS (0x18): Read-only status with idle, FIFO states, and depth
• ERR_CODE (0x1C): Read-only error code
• WIPE_SECRET (0x20): Security wipe register
• KEY[0-31] (0x24-0xA0): RegisterArray with 32 registers, spacing 4 bytes
• DIGEST[0-15] (0xA4-0xE0): RegisterArray with 16 registers, spacing 4 bytes
• MSG_LENGTH_LOWER (0xE4): Message length lower 32 bits
• MSG_LENGTH_UPPER (0xE8): Message length upper 32 bits
• MSG_FIFO (0x1000): Message FIFO window

Key Features:

• All registers are 32-bit wide
• Proper field definitions with reserved bits
• RegisterArray type used for KEY and DIGEST arrays
• Correct reset values from documentation (CFG=0x4100, STATUS=0x3, others=0x0)
• Appropriate SW Access modes (RW, RO, WO)
• WIC clearing mode for interrupt bits

Next Steps: After your approval, I will automatically:

1 Generate C++ SystemC register model code using CSML generator
2 Create model/, test/, and csml/ directories in the hmac folder
3 Validate the generated code for correctness
4 Apply any necessary fixes to ensure reset values and templates are correct

Do you approve this CSV structure? Should I proceed with C++ code generation?

You: 
```



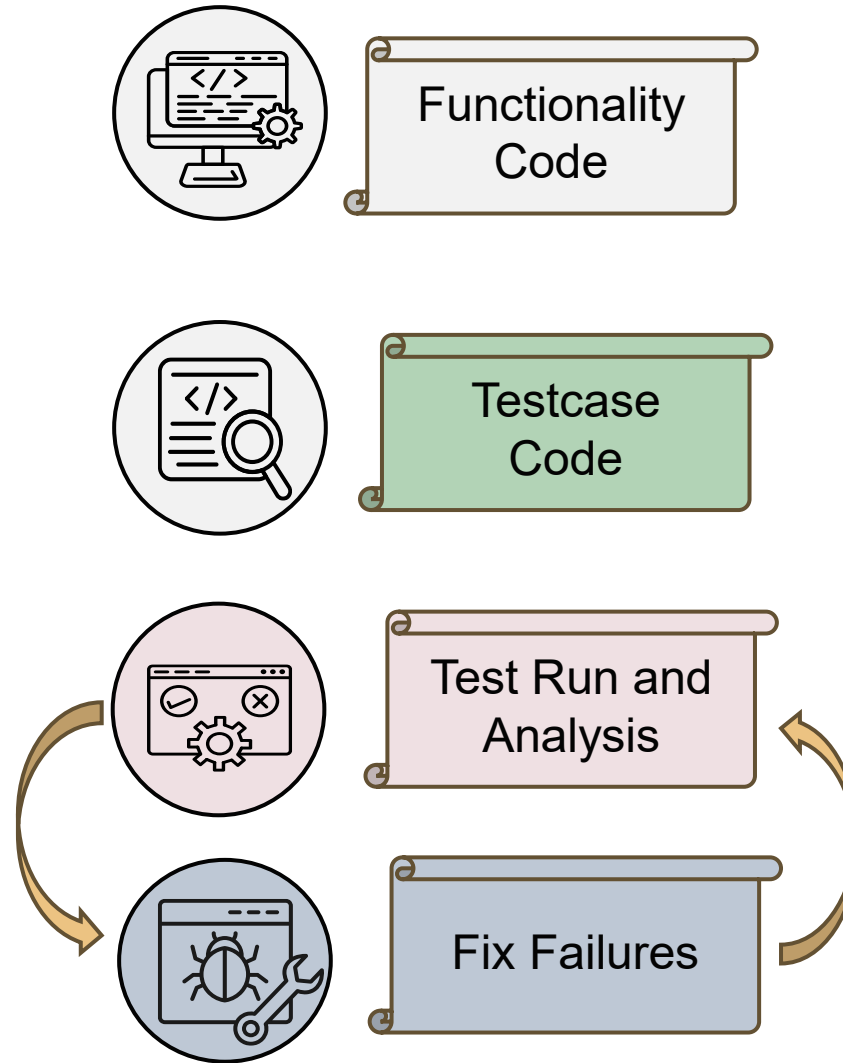
Code

Generate code for IP functionality, testcases and fix if test fails

90min



Code – Add Functionality To Stub Code



Demo – Code Phase



Code phase – Snapshots (1/2)

```
• Tasks - Functionality-Artisan
1 completed · 1 in progress · 11 pending
✓ Analyze existing code structure and architecture
• Searching and extracting register callback requirements from hardware behaviour JSON
  ○ Implement CFG register write callback with configuration lock
  ○ Implement CMD register write callback with command processing
  ○ Implement STATUS register read callback
  ○ Implement KEY register write callbacks with write protection
  ○ Implement DIGEST register write callbacks with write protection
  ○ Implement MSG_LENGTH register write callbacks with write protection
  ○ Implement MSG_FIFO write callback with backpressure modeling
  ○ Implement INTR_STATE, INTR_ENABLE, INTR_TEST callbacks
  ○ Implement WIPE_SECRET register write callback
  ○ Compile the project and fix any errors
  ○ Present implementation summary for approval
```

```
Subagent needs your input!

Context: Implemented Register-Based Transaction Interface functionality for HMAC IP with complete register callbacks, write protection, configuration lock, error detection, and TLM compliance.

Question: I have completed the SystemC TLM implementation for the HMAC IP Register-Based Transaction Interface (HMAC-FUNC-001). Below is a summary of what was implemented. Please review and approve.

Implementation Summary

Files Modified:

1 /home/ashwinp/basavarajh/demo/feb_19/tvastaavp_dvcon/hmac/model/inc/hmac.h
  • Added comprehensive internal state structure with:
    • Hash engine state (hmac_idle, msg_fifo_writable, operation_finalized)
    • FIFO management (fifo_depth, fifo_empty, fifo_full, 32-entry msg_fifo)
    • Message length tracking (msg_length_bits)
    • Internal key storage array (write-only KEY registers)
    • Interrupt enable tracking
  • Added helper function: set_error_and_interrupt()
2 /home/ashwinp/basavarajh/demo/feb_19/tvastaavp_dvcon/hmac/model/src/hmac.cpp
  • Enhanced constructor to initialize all internal state variables
  • Enhanced reset_handler() to properly reset all functionality state
3 /home/ashwinp/basavarajh/demo/feb_19/tvastaavp_dvcon/hmac/model/src/hmac_register.cpp
  • Implemented complete register callbacks with full hardware behavior

Register Callbacks Implemented:

1. INTR_STATE (Interrupt Status)
  • WIC semantics: Write-1-to-clear for all interrupt bits
  • Properly updates internal interrupt state and triggers driver updates

2. INTR_ENABLE (Interrupt Enable)
  • Controls interrupt masking for all three sources
  • Re-evaluates interrupt outputs when enables change

3. INTR_TEST (Interrupt Test)
  • Force-sets interrupt bits for software testing
  • Respects enable masks

4. ALERT_TEST (Alert Test)
  • Forces fatal_fault alert for testing

5. CFG (Configuration Register)
```

Code phase – Snapshots (2/2)

Subagent needs your input!

Context: Testcase-Artisan agent analyzing HMAC-FUNC-001 for test-bench generation. The DUT is already implemented and compiled. I need approval to proceed with comprehensive test implementation.

? Question: I have analyzed the HMAC IP implementation and the test case requirements for HMAC-FUNC-001 (Register-Based Transaction Interface). Before proceeding with comprehensive testbench implementation, I'd like to present my test plan for your approval.

Current Status:

- DUT implementation: COMPLETED (code_status.md confirms HMAC-FUNC-001 is STAGE_FUNCTIONALITY_COMPLETED)
- Existing testbench: Basic tests exist (port binding, read/write registers, reset) but they are NOT comprehensive for FUNC-001

Test Plan for HMAC-FUNC-001:

The testcase mapping document identifies 13 test cases for this functionality:

- REG_001: Read-only register access (STATUS, ERR_CODE)
- REG_002: Read-write register access (INTR_STATE, INTR_ENABLE, CFG, DIGEST_0-15, MSG_LENGTH_*)
- REG_003: Write-only register access (INTR_TEST, ALERT_TEST, CMD, WIPE_SECRET, KEY_0-31)
- REG_004: Unaligned address access (negative test)
- REG_010: Register reset values verification
- REG_011: Reset behavior during operation
- REG_020: Reserved bits return 0 (negative test)
- REG_021: Reserved bit fields always read as 0
- REG_030: W1C behavior for INTR_STATE.hmac_done
- REG_031: W1C clearing mechanism (write 1 to clear)
- REG_032: W1C clearing for INTR_STATE.hmac_err
- REG_033: RO bit behavior for INTR_STATE.fifo_empty (write ignored)

Proposed Test Implementation: I will create comprehensive test methods in the testbench that validate:

- 1 All register access semantics (RO, WO, RW, W1C, r0w1c)
- 2 Configuration lock during operation (CFG register)
- 3 Write protection mechanisms (KEY registers during operation)
- 4 Reserved bit handling
- 5 Reset value verification for all registers
- 6 Register field isolation (ensure writes to one field don't affect others)

Would you like me to proceed with implementing these comprehensive tests?

You:

Subagent needs your input!

Context: I am analyzing the HMAC testbench runtime execution as per the Testcase-Analyzer workflow. The test failed on DIGEST register access, and I need your approval to continue with phases 3-5 (Failure Analysis, Root Cause Determination, and Fix Plan Construction).

? Question: I have identified a runtime failure in the HMAC testbench execution. Before I proceed with detailed analysis and fix plan construction, let me summarize what I found:

Runtime Failure Summary:

- Test: REG_002 (Read-Write Register Access for DIGEST registers)
- Failure Point: DIGEST_1 register read-write test
- Error: "READ_ERROR: RESERVED LOCATION 168" (offset 0xA8)
- Symptom: DIGEST_1 write succeeded but read returns previous value (0x12345678 instead of expected 0x12345679)

Initial Observation: The DIGEST registers are defined as a csml_reg_vector with 16 elements starting at offset 0xA4 (164 decimal) with 4-byte stride. DIGEST_1 should be at offset 0xA8 (168 decimal). The warning "RESERVED LOCATION 168" suggests the memory location is not properly accessible, even though write callbacks are registered for all 16 DIGEST registers.

Should I proceed with detailed root cause analysis and fix plan construction?

You:

Impact & Roadmap

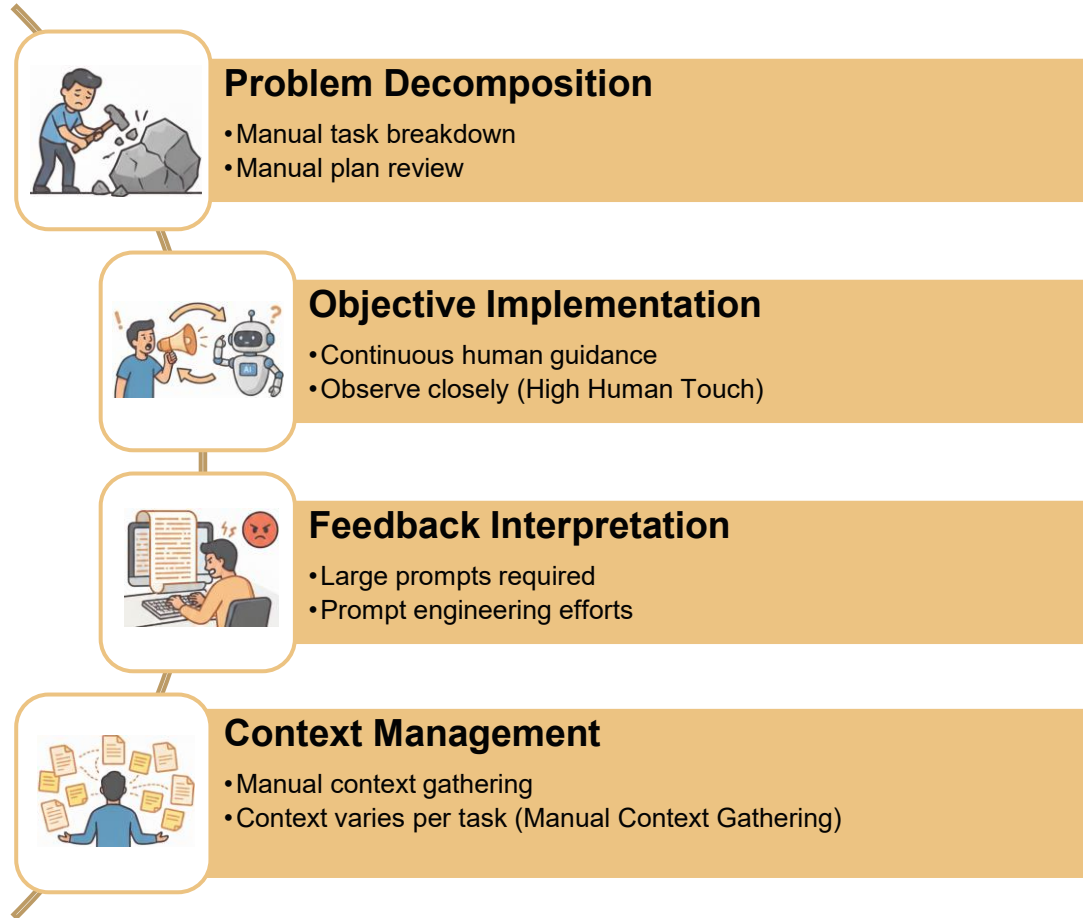
Tvastaa VP impact and roadmap

15min

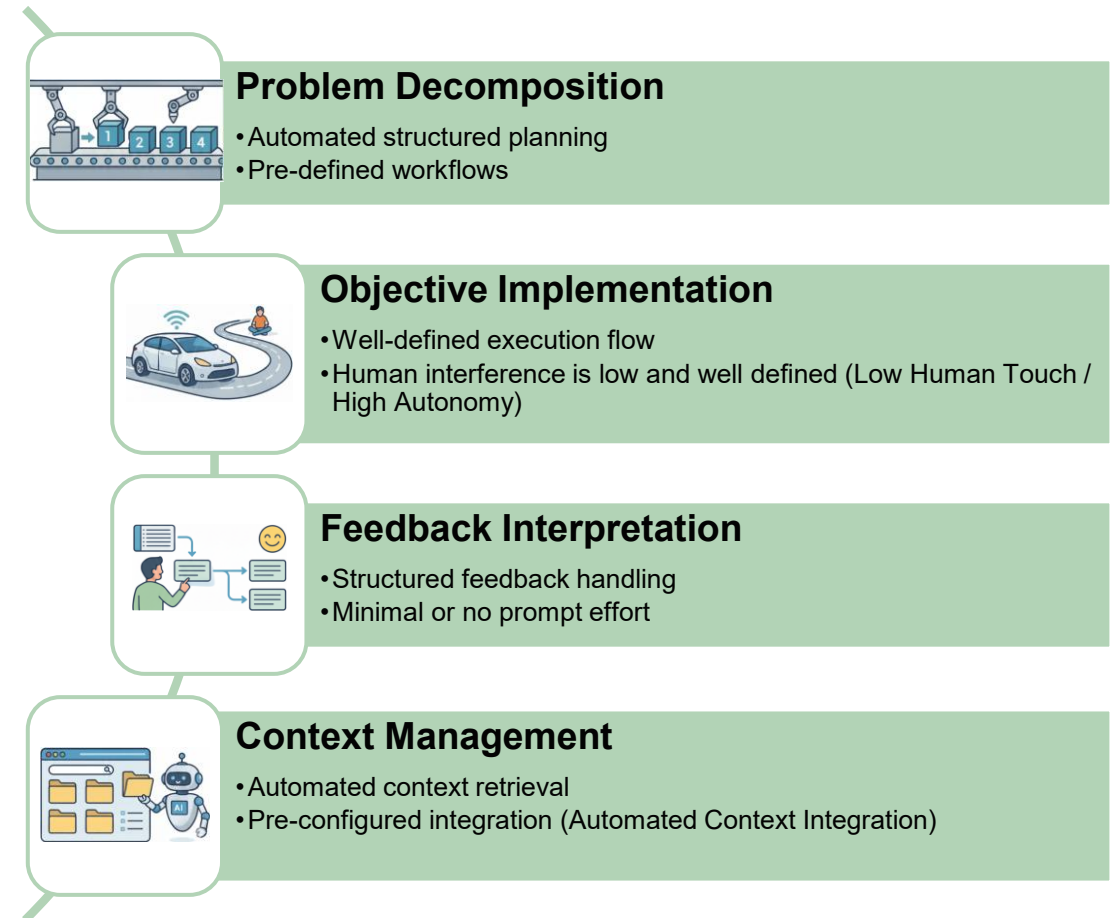


Comparison with Vibe Coding

VIBE CODING

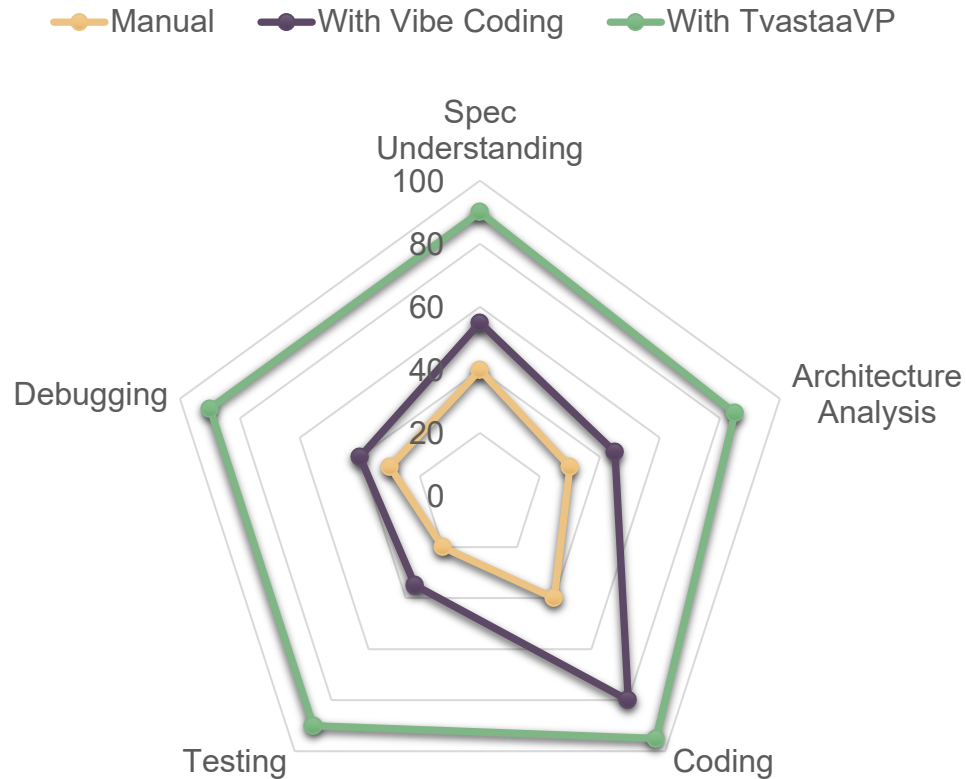


Tvastaa VP AGENTIC WORKFLOW



★ **Tvastaa VP Agentic Workflows BEATS Vibe Coding**

Same Engineer – Improved Workflow



Traditional Approach/Manual

- Manual spec study – gaps and assumptions
- Architecture quality depends on experience
- Hand-written SystemC, slow iterations
- Sparse testbenches and weak coverage
- Debug mostly reactive

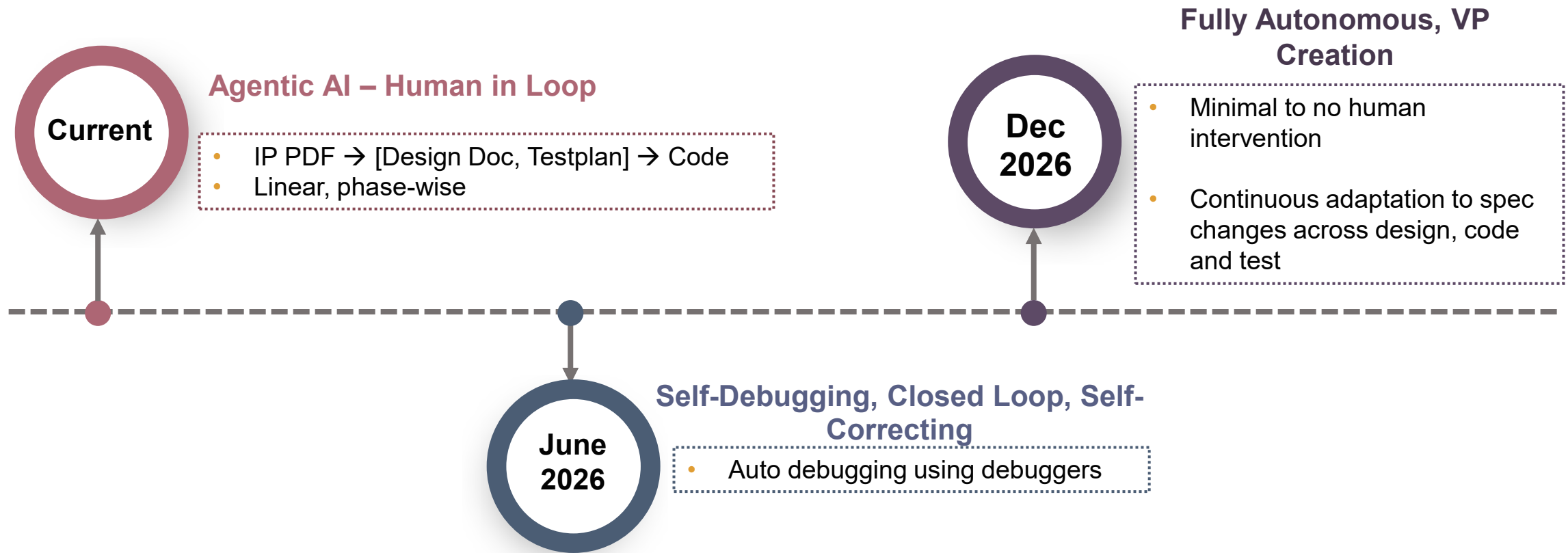
With Vibe Coding

- Repeated prompting to refine output
- Partial spec comprehension
- Code-first, architecture later
- Testbench mostly manual
- Debug driven by user feedback

With Tvastaa VP

- Guided spec understanding & extraction
- Assisted architecture and design decisions
- Structured SystemC generation
- Testbench created alongside model
- Faster debugging and fixes

Roadmap



Take Aways..

Tvastaa VP Value Proposition

- ✓ Faster spec-to-model turnaround
- ✓ Reduced dependence on SystemC human experts
- ✓ Reduced design and documentation efforts
- ✓ Consistent code quality across engineers
- ✓ Automated test generation and debugging
- ✓ Faster iteration and prototype availability

Thank You

www.vayavyalabs.com

