Innovation & Entrepreneurship Hub for Educated Rural Youth (SURE Trust – IERY)

VENDING MACHINE CONTROLLER

The domain of the Project:
RTL Design and verification (System Verilog, UVM)

Team Mentors (and their designation):

Mr. Satish Devarapalli (Emulation Verification Engineer, Apple)

Mr. Gopinath Polisetty (Lead verification engineering at Quest Global)

Mr. Bhaskar Reddy Jalapu (Staff Engineer Samsung R&D Institute India)

Team Members:

1. Mr. Sharan M, BE - 4th year pursuing ---- Team Leader
2. Mr. Pujari Karthik, B. tech - 4nd year pursuing --- Team member
3. Ms. Nandyala Naga Sunanditha, B. tech - 4th Year pursuing --- Team member
4. Mr. Venkatesh J M, BE - 4nd year pursuing --- Team member

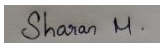
Period of the project

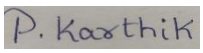
September 2024 to August 2025

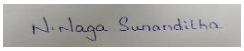
Declaration

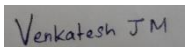
The project titled "**Vending Machine Controller**" has been mentored by **Gopinath sir, Bhaskar sir and Satish sir**, organized by **SURE Trust**, from June 2025 to July 2025, for the benefit of the educated unemployed rural youth for gaining hands-on experience in working on industry relevant projects that would take them closer to the prospective employer. I declare that to the best of my knowledge the members of the team mentioned below, have worked on it successfully and enhanced their practical knowledge in the domain.

Team Members:

1.Sharan M 

2.Pujari Karthik 

3. Nandyala Naga Sunanditha 

4. Venkatesh J M 

Mentor's Name: Mr. Satish Devarapalli
Designation— Emulation Verification Engineer, Apple
Mentor's Name: Mr. Gopinath Polisetty
Designation— Lead verification engineering, Quest Global
Mentor's Name: Mr. Bhaskar Reddy Jalapu
Designation— Staff Engineer Samsung R&D Institute India

Prof. Radhakumari
Executive Director & Founder
SURE Trust

Table of Contents

Executive Summary

This project aimed to **functionally verify a Vending Machine Controller (VMC)** IP using the Universal Verification Methodology (UVM)**.** The controller supports up to 1024 items, multiple currency denominations, and operates in **Reset, Configuration, and Operation modes** through an APB-based configuration interface. A UVM-based verification environment was developed with agents for APB configuration, currency input, item selection, and item dispense, supported by a scoreboard, reference model, and coverage collection. The methodology included both directed and randomized **test sequences** to validate configuration registers, transaction handling, dispense operations, and error scenarios such as empty item conditions and invalid inputs. Key findings show that the design meets its **functional and performance requirements**, including correct dispense decisions, accurate change return, and compliance with latency constraints. High functional coverage was achieved, ensuring thorough validation of the specification. The project demonstrates the effectiveness of **UVM-based verification** and highlights the team's ability to apply industry-standard methodologies to achieve reliable and reusable verification solutions. This work enhances employability by bridging academic training with real-world VLSI verification practices.

Introduction

Background and Context:

The Vending Machine Controller is a widely used digital IP that manages item selection, currency acceptance, and dispense operations in automated vending systems. With the increasing adoption of embedded controllers in daily life applications, ensuring their correctness and reliability becomes highly important. This project was chosen because it combines practical real-world functionality with complex digital design features, making it an ideal case for verification training.

Problem Statement:

The Vending Machine Controller (VMC) manages multiple items, currency inputs, and dispense operations, and must function reliably under various real-time scenarios. Any malfunction, such as incorrect item dispense, wrong change calculation, or misconfigured stock details, can lead to operational failures and customer dissatisfaction. Ensuring the correctness of such a design through rigorous verification is therefore essential. The challenge lies in verifying all possible configurations, transactions, and corner cases in a scalable and reusable manner. This project addresses the problem by developing a **UVM-based verification environment** to validate the VMC design against its specification, ensuring accuracy, reliability, and compliance with industry standards.

Scope:

The scope of this project is centred on the functional verification of the **Vending Machine Controller (VMC)** using System Verilog and UVM. It primarily focuses on validating the three modes of operation—Reset, Configuration, and Operation—along with APB-based register configuration, item selection, currency input, item dispense, and change calculation. The verification effort also includes testing corner cases such as empty item slots, invalid inputs, and latency requirements. However, the scope is limited to functional verification and does not extend to physical design aspects such as synthesis, timing, or low-power verification. The emphasis is on building a **scalable, reusable, and**

coverage-driven UVM environment that reflects industry-standard verification practices.

Limitations of the Project:

1. **Focus on Functional Verification Only**: The project is restricted to verifying functional aspects of the Vending Machine Controller and does not include physical design or backend processes.
2. **No Performance Analysis**: The verification testbench does not measure power, timing closure, or throughput under real hardware constraints.
3. **Exclusion of Low-Power Features**: Power optimization, low-power mode verification, and fault-tolerant features are outside the project scope.
4. **Simulation Environment Constraints**: The results are based on simulation and may not fully capture issues that could arise in silicon implementation.
5. **Specification-Driven**: The verification is limited strictly to the provided specification, with no extensions to additional features or out-of-scope enhancements.

Innovation:

This Project has some unique features of Innovation of new ideas Which improve the efficiency Which is shown through the Testcases. These Testcases shows the innovation of the project will helps the users for the better usage. The project included sanity, directed, random, and corner-case tests to validate both normal and edge-case scenarios. Four dedicated agents (APB, item selection, currency, and item dispense) were implemented to ensure modularity and scalability. The project closely followed real-world verification flows, bridging academic learning with professional VLSI verification methodologies. A complete reusable UVM testbench was developed from scratch, incorporating agents, sequences, scoreboard, and coverage.

Project Objectives

Project Objectives and Expected Outcomes:

Develop a UVM-Based Verification Environment for a Vending Machine Controller
Design and implement a complete **System Verilog, UVM testbench** incorporating APB, currency, item selection, and dispense agents, along with a scoreboard, reference model, and functional coverage.
Expected Outcome: A reusable, scalable UVM environment capable of verifying the DUT across all operational modes (Reset, Configuration, Operation).

Verify Functional Correctness of the Controller
Validate item configuration through APB interface, currency input handling, item dispense logic, change calculation, and stock tracking under normal, boundary, and error scenarios.
Expected Outcome: Demonstrated compliance of the DUT with the specification, ensuring correct operation in all supported use cases.

Test Robustness Through Corner-Case and Stress Scenarios
Execute randomized and directed sequences to verify error handling, invalid inputs, empty stock conditions, currency during config mode, latency checks, and long stress tests.
Expected Outcome: Verified resilience of the DUT under extreme and unpredictable usage conditions, ensuring stability and reliability.

Ensure Specification Compliance Using Coverage and Assertions
Apply functional coverage and assertion-based checks for mode transitions, protocol compliance, and latency requirements (<10 cycles).
Expected Outcome: High coverage metrics and assertion results proving complete verification closure of the VMC design.

Align with Industry-Standard Verification Practices
Adopt modern UVM methodology to bridge academic training with real-world SoC verification flows.
Expected Outcome: A professional verification framework and enhanced hands-on skills, directly applicable to semiconductor industry projects.

Project Deliverables:

1. **UVM-Based Verification Environment**
   - Complete layered UVM testbench including APB, currency, item selection, and dispense agents.
   - Scoreboard, reference model, and functional coverage implementation.
2. **Test Sequences and Test Plan**
   - Directed, randomized, error-handling, stress, and integration test cases.
   - Sequences covering configuration, reset, operation, timing, and corner cases.
3. **Verification Reports**
   - Functional coverage and code coverage reports showing verification completeness.
   - Assertion results validating protocol compliance and mode transitions.
4. **Waveform and Debug Evidence**
   - Simulation waveforms and logs demonstrating DUT behaviour for key scenarios.
   - Evidence of latency compliance (<10 cycles) and correct item dispense/change.
5. **Documentation**
   - Project report with objectives, methodology, results, and learning outcomes.
   - Test plan, architecture diagrams, and final verification summary.
6. **Reusable Components**
   - Modular agents and testbench components that can be extended for similar IP blocks.
   - A reference verification framework for academic and industry use.

Methodology and Results

Methods / Technology Used:

1. **Hardware Description and Verification Languages**
    - **System Verilog** for UVM testbench development, including transactions, sequences, assertions, and coverage.
2. **Verification Methodology**
    - **Universal Verification Methodology (UVM)** to build a layered, reusable testbench with agents, sequencers, drivers, monitors, scoreboard, reference model, and coverage collectors.
3. **Simulation Environment**
    - **Synopsys VCS** simulator accessed via **EDA Playground** (cloud-based platform) for compiling, simulating, and debugging the DUT and testbench.
4. **Reference Specification**
    - **Vending Machine Controller specification** provided by SURE ProEd, defining DUT features, interfaces, and timing constraints.
5. **Version Control & Documentation**
    - **GitHub** used to maintain source code, test plan, sequences, and project documentation.

Tools / Software Used:

1. **System Verilog** – Hardware Description and Verification Language used to develop the UVM testbench.
2. **UVM (Universal Verification Methodology)** – Industry-standard methodology for building reusable and scalable test environments.
3. **Synopsys VCS** – Simulator used within EDA Playground for compiling and running System Verilog/UVM testbenches.
4. **EDA Playground** – Cloud-based platform to write, simulate, and debug the verification environment.
5. **GitHub** – Version control and repository for maintaining testbench code, sequences, and project documentation.

Project Architecture:

The verification environment for the **Vending Machine Controller (VMC)** was designed using the **Universal Verification Methodology (UVM)**, following a modular and layered architecture. The testbench is structured to ensure **scalability, reusability, and clarity** while verifying the DUT against its specification.

1. Design Under Test (DUT): Vending Machine Controller

The DUT is a digital IP that manages item configuration, currency input, and item dispensing. It operates in three distinct modes:

- **Reset Mode** – DUT registers and memories reset to default values.
- **Configuration Mode** – Vendor (via APB interface) configures item details, stock counts, and clears dispense counters.
- **Operation Mode** – Customer selects items, inserts currency, and receives dispense or change.

The DUT has multiple interfaces:

- **System Interface** – Clock, reset, and configuration mode signals.
- **APB Configuration Interface** – For writing and reading configuration registers.
- **Currency Interface** – Accepts currency inputs at a slower asynchronous clock.
- **Item Selection Interface** – Accepts item selection codes.
- **Item Dispense Interface** – Outputs dispense signals and change values.

2. UVM Environment

The testbench consists of four UVM agents, each dedicated to a specific DUT interface:

- **Configuration Agent (cfg_agent)**
  - An active UVM agent that drives the APB interface.
  - Responsible for register read and write operations during configuration mode.

  - Executes sequences such as config_write_item_seq, config_read_item_seq, config_max_items_seq, and config_randomized_item_setup_seq.
- **Currency Agent (currency agent)**
  - Active agent generating various currency patterns.
  - Simulates real-world scenarios like exact payment, overpayment, underpayment, and random currency input.
  - Sequences include exact_currency_dispense_seq, overpayment_with_change_seq, underpayment_seq, and random_currency_seq.
- **Item Agent (item_agent)**
  - Active agent responsible for driving item selection requests.
  - Stimulates DUT with valid and invalid selections, random selections, and multi-item requests.
  - Sequences include invalid_item_select_seq, multi_dispense_item_seq, and random_user_sequence_mix_seq.
- **Dispense Agent (dispense_agent)**
  - A passive monitoring agent connected to the DUT output interface.
  - Captures dispense signals, change values, and latency information.
  - Sends observed transactions to the scoreboard for validation.

3. Scoreboard and Reference Model

The **Scoreboard** plays a crucial role in ensuring correctness:

- It compares the DUT's outputs (item dispense, change values) with predictions generated by the **Reference Model**.
- Validates correctness of dispense logic, accurate change calculation, and adherence to latency constraints (<10 system clocks).
- Tracks total items dispensed, remaining stock, and ensures compliance with specification requirements.

4. Functional Coverage & Assertions

- **Functional Coverage**: Coverage points capture exercised scenarios, including valid/invalid currency, boundary item configurations, multiple item dispensing, and corner cases. This ensures **verification completeness**.

- **Assertions**: Used to validate mode transitions, APB protocol compliance, and output pulse requirements. Assertions ensured that the DUT generated exactly **one clock-wide output pulse per transaction** and ignored illegal conditions (e.g., currency during config mode).

5. Test Sequences

The testbench implemented a comprehensive set of test sequences:

- **Reset Tests** – Verify proper reset of all registers and DUT states.
- **Configuration Tests** – Validate APB transactions, register storage, and item setup.
- **Operation Tests** – Cover normal use cases (exact payment, overpayment, multi-part currency input).
- **Error Handling Tests** – Check robustness for invalid items, unsupported currency, and empty stock.
- **Randomized Tests** – Mix valid and invalid operations to test unpredictable use cases.
- **Stress Tests** – Long-running sequences to validate DUT under high-load conditions.
- **Timing Tests** – Verify output latency (<10 cycles) and single-clock-wide dispense pulses.
- **Integration Tests** – Full-flow scenarios combining configuration, reset, and operation into end-to-end verification.

6. Overall Flow

1. Input agents (**cfg_agent, currency_agent, item_agent**) generate stimulus to drive the DUT.
2. The DUT processes the stimulus and generates outputs via the dispense interface.
3. The **dispense_agent** captures DUT responses and forwards them to the **scoreboard**.
4. The **scoreboard + reference model** validates correctness against expected results.
5. **Coverage collectors and assertions** ensure all features, corner cases, and protocols are verified.

Results:

The Vending Machine Controller was successfully verified using a UVM-based testbench. All functional requirements, including item dispense, change calculation, and mode transitions, were validated against the specification. The DUT met latency requirements (<10 cycles) and handled corner cases such as empty stock, invalid inputs, and unsupported currency correctly, etc. All directed, randomized, and corner-case tests passed successfully. The DUT met its functional, timing, and coverage goals, confirming its correctness and robustness. The project outcomes align with industry-standard verification practices, ensuring reliability of the design.

Final Project Hardware and Working Screenshots

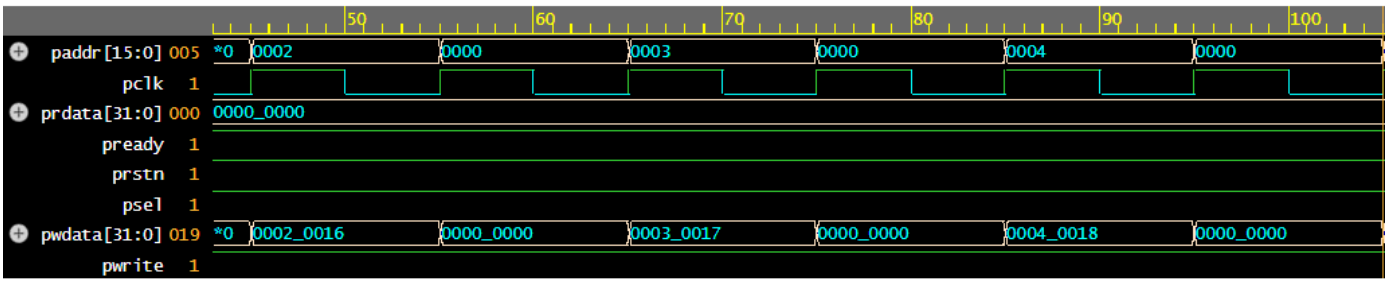


Figure 1. Waveform of configuration agent

The waveform shows the **Configuration Agent (cfg_agent)** performing APB transactions. paddr indicates the register addresses being accessed, while pwdata carries the data written. psel and pwrite are asserted to enable valid write operations. prdata reflects the stored register values during read operations. This verifies that the DUT correctly accepts and updates configuration data through the APB interface.

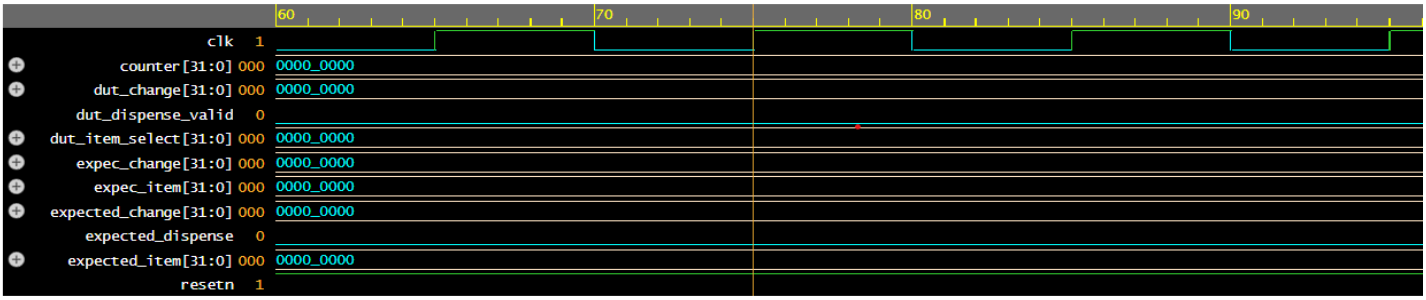


Figure 2. Waveform of checker

The waveform illustrates the **Checker (Scoreboard)** comparing DUT outputs with expected results.

dut_item_select, dut_dispense_valid, and dut_change represent the DUT's actual responses.
expected_item, expected_dispense, and expected_change is generated by the reference model.
The checker ensures that the DUT outputs always match the predicted values. This confirms the correctness of dispense operations, change calculation, and overall DUT behaviour.

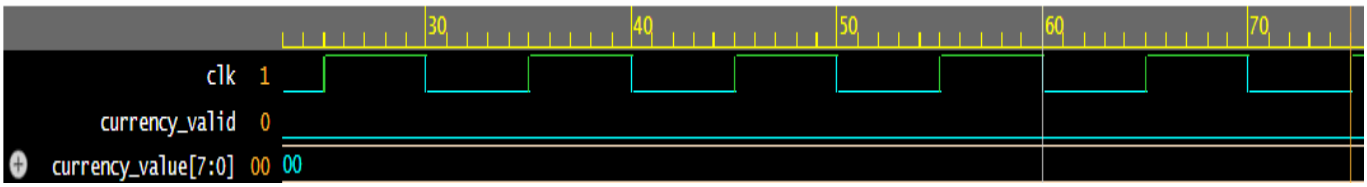


Figure 3. Waveform of Currency agent

The waveform illustrates the activity of the **Currency Agent** driving inputs to the DUT. currency_valid indicates when a valid currency input pulse is applied. currency_value carries the denomination of the inserted currency (5, 10, 15, 20, 50, 100 Rs). The signal is synchronized with the system clock (clk) to ensure proper sampling by the DUT.This demonstrates that the DUT receives valid currency transactions, which are later processed for dispense and change decisions.

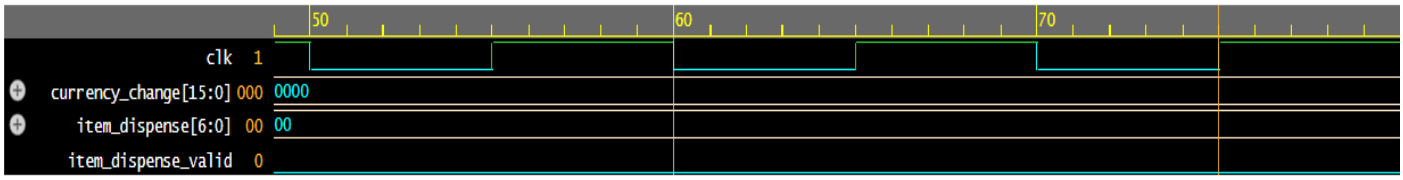


Figure 4. Waveform of Dispense agent

The waveform shows the monitoring activity of the **Dispense Agent** on the DUT outputs. Item_dispense_valid indicates when a valid dispense operation occurs. Item_dispense carries the item ID that has been successfully dispensed. currency_change reflects any balance amount returned to the user.
This confirms that the Dispense Agent correctly captures DUT outputs for validation by the scoreboard.

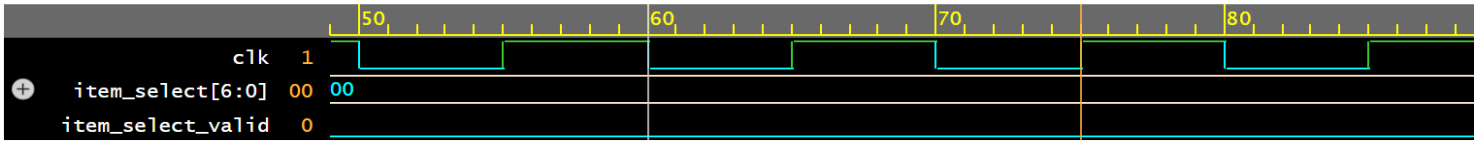


Figure 5. Waveform of Item select agent

The waveform shows the behaviour of the **Item Select Agent** driving item selection inputs. Item_select_valid indicates when a valid item selection request is applied. Item_select carries the ID of the chosen item that the user wants to purchase. The signal transitions are synchronized with the system clock (clk) for proper sampling. This confirms that item selection inputs are correctly generated and applied to the DUT for verification.

```
`include "uvm_macros.svh"
import uvm_pkg::*;
`include "tb_includes.sv"

module top;

  logic clk = 0;
  always #5 clk = ~clk;

  logic pclk = 0;
  always #5 pclk = ~pclk;

  checker_if     checker_vif(clk);
  ctrl_if        ctrl_vif(clk);
  cfg_if         cfg_vif(pclk);
  currency_if    currency_vif(clk);
  item_select_if #(.MAX_ITEMS(MAX_ITEMS)) item_vif(clk);
  dispense_if #(.MAX_ITEMS(MAX_ITEMS))   dispense_vif(clk);
```

Figure 6. Top level snippet

The code defines the **top-level testbench module (top)**.
Two clocks are generated: clk for the main system and pclk for the APB configuration interface.
Multiple **virtual interfaces** (checker_if, cfg_if, currency_if, item_select_if, dispense_if) are instantiated.
These interfaces connect the DUT with corresponding UVM agents for stimulus and monitoring.
This structure ensures synchronization between the DUT and all verification components.

```
vending_machine #(.MAX_ITEMS(MAX_ITEMS), .MAX_NOTE_VAL(100)) dut (
  .clk(clk),
  .rstn(ctrl_vif.rstn),
  .cfg_mode(ctrl_vif.cfg_mode),

  .pclk(pclk),
  .prstn(cfg_vif.prstn),
  .paddr(cfg_vif.paddr),
  .psel(cfg_vif.psel),
  .pwrite(cfg_vif.pwrite),
  .pwdata(cfg_vif.pwdata),
  .prdata(cfg_vif.prdata),
  .pready(),

  .currency_valid(currency_vif.currency_valid),
  .currency_value(currency_vif.currency_value),

  .item_select_valid(item_vif.item_select_valid),
  .item_select(item_vif.item_select),

  .item_dispense_valid(dispense_vif.item_dispense_valid),
  .item_dispense(dispense_vif.item_dispense),
  .currency_change(dispense_vif.currency_change)
);
```

Figure 7. Virtual interface snippet

The vending machine DUT is instantiated with parameters for maximum items and maximum note value (set to 100).A **virtual interface** is used to connect DUT ports with different functional interfaces in the testbench. The **control interface** (ctrl_vif) handles reset and configuration mode signals. The **configuration interface** (cfg_vif) connects APB signals like address, write, read, and ready. The **currency, item, and dispense interfaces** manage user inputs (currency, item selection) and outputs (item dispense, change return).

```
 run_test("vending_test");            //  vending test1
 // run_test("reset_mode_test");         //  vending_test2
 // run_test("config_mode_test");        //  vending_test3
// run_test("error_operation_test");     //  vending_test4
/   run_test("latency_output_test");     //  vending_test5
 // run_test("operation_random_test");   //  vending_test6
 // run_test("operation_basic_test");    //  vending_test7
  //run_test("config_negative_test");    //  vending_test8
 //run_test("config_full_test");         //  vending_test9
 // run_test("config_edge_test");        //  vending_test10
 // run_test("config_random_test");      //  vending_test11
```

Figure 8. Testcases Name

These Tests are used for our Verification project which include different scenarios of the sequences which accumulated as the separate tests. More information about the sequences and the test Scenarios are provided in the GitHub Link.

```
Name                        Type                          Size  Value
-------------------------------------------------------------------
uvm_test_top                vending_test                  -     @370
  env                       vending_env                   -     @383
    cfg_ag                  cfg_agent                     -     @393
      drvr                  cfg_driver                    -     @625
        rsp_port            uvm_analysis_port             -     @644
        seq_item_port       uvm_seq_item_pull_port        -     @634
      mon                   cfg_monitor                   -     @654
        cfg_ap              uvm_analysis_port             -     @663
      seqr                  cfg_sequencer                 -     @488
        rsp_export          uvm_analysis_export           -     @497
        seq_item_export     uvm_seq_item_pull_imp         -     @615
        arbitration_queue   array                         0     -
        lock_queue          array                         0     -
        num_last_reqs       integral                      32    'd1
        num_last_rsps       integral                      32    'd1
    currency_ag             currency_agent                -     @402
      drvr                  currency_driver               -     @813
        rsp_port            uvm_analysis_port             -     @832
        seq_item_port       uvm_seq_item_pull_port        -     @822
      mon                   currency_monitor              -     @842
        currency_ap         uvm_analysis_port             -     @851
      seqr                  currency_sequencer            -     @676
        rsp_export          uvm_analysis_export           -     @685
        seq_item_export     uvm_seq_item_pull_imp         -     @803
        arbitration_queue   array                         0     -
        lock_queue          array                         0     -
        num_last_reqs       integral                      32    'd1
        num_last_rsps       integral                      32    'd1
    dispense_ag             dispense_agent                -     @420
      mon                   dispense_monitor              -     @865
        dispense_ap         uvm_analysis_port             -     @874
    item_ag                 item_agent                    -     @411
      drvr                  item_driver                   -     @1023
        rsp_port            uvm_analysis_port             -     @1042
        seq_item_port       uvm_seq_item_pull_port        -     @1032
      mon                   item_monitor                  -     @1052
        item_ap             uvm_analysis_port             -     @1061
      seqr                  item_sequencer                -     @886
        rsp_export          uvm_analysis_export           -     @895
        seq_item_export     uvm_seq_item_pull_imp         -     @1013
        arbitration_queue   array                         0     -
        lock_queue          array                         0     -
        num_last_reqs       integral                      32    'd1
        num_last_rsps       integral                      32    'd1
    scoreboard              top_scoreboard                -     @478
      a_chk                 apb_checker                   -     @1121
        imp                 uvm_analysis_imp              -     @1130
      c_chk                 currency_checker              -     @1083
        imp                 uvm_analysis_imp              -     @1092
      d_chk                 dispense_checker              -     @1140
        imp                 uvm_analysis_imp              -     @1149
      i_chk                 item_checker                  -     @1102
        imp                 uvm_analysis_imp              -     @1111
      ref_m                 vending_ref_model             -     @1074
    v_scoreboard            vending_scoreboard            -     @429
      cfg_export            uvm_analysis_imp_cfg          -     @468
      currency_export       uvm_analysis_imp_currency     -     @448
      dispense_export       uvm_analysis_imp_dispense     -     @458
      item_export           uvm_analysis_imp_item         -     @438
-------------------------------------------------------------------
```

The above provided data are the UVM Topology which should be included for us to see the weightage of the variable or type is holding a certain value. The below provided the output of the test and also the errors which we faced.

```
UVM_INFO vending_test.sv(31) @ 0: uvm_test_top [VENDING_TEST] Starting test sequences...
UVM_INFO dispense_monitor.sv(28) @ 5: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
UVM_INFO dispense_checker.sv(35) @ 5: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           0
UVM_INFO dispense_checker.sv(48) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
UVM_INFO dispense_monitor.sv(28) @ 15: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
UVM_INFO dispense_checker.sv(35) @ 15: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           0
UVM_INFO dispense_checker.sv(48) @ 15: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 15: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
```

1.The starting of the vending test is to check the conditions which checks the provided cfg,item_sel,currency sequences.

```
UVM_INFO vending_test2.sv(30) @ 0: uvm_test_top [RESET_TEST] Running reset test...
UVM_INFO dispense_monitor.sv(28) @ 5: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
UVM_INFO dispense_checker.sv(35) @ 5: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           0
UVM_INFO dispense_checker.sv(48) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
```

2.The 2nd test is the reset test which is also provided with the sanity test that should be implemented as per the specification of the project.

```
UVM_INFO vending_test3.sv(34) @ 0: uvm_test_top [CONFIG_TEST] Configuration write sequence started
UVM_INFO dispense_monitor.sv(28) @ 5: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
UVM_INFO dispense_checker.sv(35) @ 5: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           0
UVM_INFO dispense_checker.sv(48) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 5: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
```

```
UVM_ERROR apb_checker.sv(34) @ 815: uvm_test_top.env.scoreboard.a_chk [APB_CHECKER] value not matched ..... dut_val=7, ref_val=0
UVM_ERROR apb_checker.sv(37) @ 815: uvm_test_top.env.scoreboard.a_chk [APB_CHECKER] avail not matched ..... dut_avail=205, ref_avail=0
UVM_ERROR apb_checker.sv(40) @ 815: uvm_test_top.env.scoreboard.a_chk [APB_CHECKER] disp not matched ..... dut_disp=171, ref_disp=0
```

3.The third test which allows the verification into the configuration mode and it should be implemented with different checkers and the errors we faced here are the important because of the task which provided should not be matched because of the sequences of APB criteria.

```
UVM_INFO dispense_monitor.sv(28) @ 55: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
UVM_INFO dispense_checker.sv(35) @ 55: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           0
UVM_INFO dispense_checker.sv(48) @ 55: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 55: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
UVM_INFO cfg_monitor.sv(37) @ 55: uvm_test_top.env.cfg_ag.mon [CFG_MONITOR] WRITE: addr=0x2 data=0x20016
UVM_INFO apb_checker.sv(24) @ 55: uvm_test_top.env.scoreboard.a_chk [APB_CHECKER] Configured item = 2 : val=22, avail=2
UVM_INFO dispense_monitor.sv(28) @ 65: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
```

4.The Fourth test is the error operation test which explain with different scenarios but in general the error should be accumulated in this test which will satisfy through Dispense Monitor.

5.The fifth test is the latency test which will verifies through the waveforms and it will shows about the logic of less than 10 system clocks.

```
UVM_INFO dispense_checker.sv(35) @ 1415: uvm_test_top.env.scoreboard.d_chk [dispense_CHECK] expected_item:           2
UVM_INFO dispense_checker.sv(48) @ 1415: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] DUT dispensed=0, expected=0
UVM_INFO dispense_checker.sv(53) @ 1415: uvm_test_top.env.scoreboard.d_chk [DISPENSE_CHECKER] Change : DUT=0, Expected=0
UVM_ERROR vending_test6.sv(60) @ 1415: uvm_test_top [RAND_MULTIITEM_TEST] Both item & currency random sequences done.
UVM_INFO dispense_monitor.sv(28) @ 1425: uvm_test_top.env.dispense_ag.mon [DISPENSE_MONITOR] Item Dispensed: 0, Change Returned: 0, item_select :   0
```

6.The sixth test is the random test which will interfere any scenarios of the sequences which will be executed and the errors will appear because of the condition of sequences inference we provided it as just a rare scenario.The other Test which we implemented will be on the configuration basis.

EDA PLAYGROUND : https://edaplayground.com/x/w234

Learning and Reflection

Learning and Reflection:

Overall experience:

Team Member	Role	Technical Learnings	Management / Soft Skills	Overall Experience
Sharan M	Configuration Agent	Learned UVM-based testbench design, APB transaction handling, writing and executing configuration sequences, analysing waveform outputs	Improved coordination with team members, time management during verification tasks	Gained in-depth understanding of configuration flows and verification methodology; overall felt confident in handling complex test scenarios
Karthik pujari	Item Selection Agent	Developed skills in designing and implementing item selection logic, handling APB transactions, debugging selection sequences	Learned effective communication within the team, reporting issues clearly	Felt hands-on experience in agent-level verification; enhanced understanding of end-to-end item selection process
Venkatesh J M	Currency Agent	Gained experience in designing and testing currency handling sequences, reference model integration, waveform verification	Improved collaborative problem-solving and constructive feedback during testing	Overall exposure to currency agent design and validation increased technical confidence and understanding of system behaviour
Naga Sunanditha	Dispense Agent	Learned to implement and verify dispense logic, integrate with other agents, and analyse timing of outputs	Developed systematic documentation and reporting habits, coordination with other members for seamless agent interaction	Gained practical understanding of dispense mechanisms and workflow integration; overall experience strengthened verification skills.

Achievements

- Successfully developed a **UVM-based verification environment** for the Vending Machine Controller IP.
- Designed and implemented **modular agents** (configuration, currency, item selection, and dispense) ensuring reusability and scalability.
- Verified key features of the DUT including **APB configuration, item setup, currency validation, dispense logic, and change calculation**.
- Achieved **comprehensive functional coverage and code coverage**, validating correctness across normal, error, and corner-case scenarios.
- Gained hands-on experience in **virtual interfaces, score boarding, randomization, and sequence creation**.
- Integrated all components to perform **end-to-end testing**, covering reset, operation, error handling, and stress scenarios.
- Strengthened teamwork, debugging, and **industry-oriented verification skills** through collaborative environment building and problem-solving.

Learnings (Sharan M)

- **Technical Knowledge**:
  - Gained in-depth understanding of **APB protocol**, its timing, and register read/write verification.
  - Learned to implement a **UVM configuration agent** with sequencer, driver, and monitor, and connected it to the scoreboard for checking.
  - Improved skills in **virtual interfaces and modular agent design** for scalable verification environments.
  - Acquired hands-on practice with **functional coverage, randomization, and sequence creation** for corner-case testing.
  - Enhanced debugging skills by integrating multiple agents and resolving environment connectivity issues.
- **Management/Process Knowledge**:
  - Learned to plan sequences systematically**.**
  - Improved task ownership as a Team Lead and **coordination with other agents by others** for smooth integration.
  - Experienced the importance of incremental build and testing**,** instead of attempting full integration at once.

Experience (Reflection)

- Working on the **configuration agent** gave me confidence in handling **bus protocols** and developing reusable verification components.
- During **integration**, I experienced real challenges like mismatched interfaces, synchronization issues, and scoreboard alignment—but solving them deepened my understanding of the **UVM flow**.
- The project enhanced my ability to **think from both the design and verification perspective**, ensuring specification features are fully testable.
- Overall, this was a highly practical and industry-relevant experience**,** improving not just technical skills, but also problem-solving, patience, and teamwork mindset.

Learnings (Naga Sunanditha)

Technical

● Gained a good understanding of UVM methodology and how it is applied in real projects.
● Learned about the UVM testbench architecture, including environment, agents, sequencer, driver, monitor, scoreboard, and coverage.
● Understood how different UVM components interact through TLM ports (analysis ports, exports, etc.) to build a reusable and modular environment.
● Practiced writing sequences and test cases for both directed and random scenarios.
● Specifically, I worked on the dispense agent (passive), which monitors DUT outputs (item_dispense_valid, item_dispense, currency_change).
● Implemented the checker/scoreboard to validate correctness, timing, and error conditions.
● Developed test cases like exact payment, overpayment with change, underpayment, zero stock, and invalid item selection.

Management & Collaboration Learnings

● Learned to coordinate with team members' agents and integrate my components into the environment.
● Improved time planning, modular design, and communication skills during team-based verification.

Experience

● This project gave me hands-on exposure to UVM-based verification of a complex design (Vending Machine Controller).
● Working on the dispense agent and checker taught me that verification is not only about generating inputs but also about ensuring the DUT behaves correctly under all possible conditions.
● Developing test cases for error handling and boundary conditions improved my debugging and analysis skills.
● I gained confidence in building scalable and reusable verification components, especially scoreboards and passive monitors.
● Overall, I found this experience challenging but rewarding, as it helped me grow technically in System Verilog & UVM and personally in teamwork and problem-solving

Learnings (Karthik)

Technical

Gained a solid understanding of UVM methodology and testbench structure (agents, sequences, drivers, monitors, and scoreboards).
Designed and implemented the currency agent (active) to drive currency_valid pulses and currency_value into the DUT.
Learned how to create directed and randomized test cases for currency-related scenarios such as exact payment, overpayment with change, underpayment, and unsupported currency values.
Worked on the checker/scoreboard side to validate currency transactions against expected DUT behaviour.
Understood how coverage metrics help ensure all currency input patterns and edge cases are verified.
Got hands-on experience in integration of multiple agents (cfg, item, currency, dispense) into a common environment.

Team & Management

Helped with integration of all agents in the verification environment and ensured smooth communication through analysis ports.
Collaborated with teammates by running combined tests and debugging failures. Identified bugs in the DUT during test runs (e.g., dispense mismatches,

incorrect change, FSM misbehaviour), especially when cross-checking with verification tests.
Learned the importance of coordination, debugging discipline, and reporting errors clearly in a verification project.

Experience

The project provided practical exposure to currency-related verification in UVM.
Building the currency agent and its test cases gave confidence in handling asynchronous input interfaces.
Playing a key role in environment integration gave good learning on how multiple UVM components connect and work together.
Actively debugging and finding DUT issues, especially when validating against the scoreboard was both challenging and rewarding.
Overall, this experience improved both technical skills in UVM and team skills in integration and debugging.

Learnings(Venkatesh JM)

1. Learned to structure a complete **UVM verification environment** with agents, scoreboard, and reference model for reusability.
2. Gained hands-on experience in **currency agent design** (sequencer, driver, monitor) and writing **directed/randomized sequences**.
3. Understood the role of **reference model and scoreboard** in creating a self-checking testbench.
4. Strengthened knowledge of **APB protocol verification** and configuration interface testing.
5. Learned **cross-agent coordination** (currency, item, cfg, dispense) for realistic environment behaviour.
6. Explored **error handling and negative tests** like unsupported currency and invalid item selections.
7. Applied **functional/code coverage** techniques to ensure verification completeness.
8. Improved **debugging skills** in UVM (virtual interface binding, TLM connections).
9. Learned the importance of **specification-driven stimulus generation** for realistic and legal test cases.

Experience

This project gave me end-to-end experience in verification of a digital IP using UVM. Although I contributed across the environment, my primary focus was on the currency agent, where I gained hands-on knowledge of sequencer-driver-monitor connectivity, transaction modelling, and self-checking verification flow. By working on both positive and negative test scenarios, I reinforced my understanding of FSM-based DUT behaviour (dispense, change calculation, error handling). Debugging connectivity, synchronization, and analysis port issues improved my confidence in handling real-world UVM problems.

Conclusion and Future Scope

Conclusion

The primary objective of this project was to **verify the Vending Machine Controller IP** against its functional specification. A complete **UVM-based verification environment** was developed consisting of configuration, currency, item selection, and dispense agents, along with a scoreboard and reference model for automated checking. Various test scenarios including reset, configuration, basic operation, error handling, timing checks, random sequences, and stress tests were successfully executed. The environment achieved significant **functional and code coverage**, ensuring correctness, robustness, and compliance of the design with the given requirements. Overall, the project provided a strong learning platform for applying **System Verilog and UVM methodologies** in a real verification flow.

Future Scope

Looking ahead, the verification environment for the Vending Machine Controller can be expanded in several exciting directions:

- **AI/ML-driven Verification**: Apply machine learning techniques to generate smarter random test sequences, automatically detect corner cases, and predict coverage gaps.

- **Portable Stimulus Integration**: Use Accellera's Portable Stimulus Standard (PSS) to create reusable test scenarios across simulation, emulation, and FPGA prototyping platforms.
- **Real-Time Prototyping**: Map the design onto an FPGA and connect it with actual hardware inputs (buttons, coin acceptors, displays) to validate both simulation and hardware behaviour.
- **Security and Fault Tolerance**: Extend tests to cover security aspects (e.g., invalid inputs, counterfeit currency) and study design resilience under fault injection.
- **IoT-Enabled Extensions**: Explore integration of the controller with IoT-based monitoring systems where stock and transactions can be verified and analysed remotely.

By pursuing these directions, the project not only ensures **functional correctness** but also evolves toward **next-generation verification practices** that combine automation, scalability, and real-world adaptability.