ERIFICATION PLAN

## 1. Introduction

### 1.1 Objective

The objective of this verification plan is to define the strategy, scope, and structure for functionally verifying the generic\_intc (Generic Interrupt Controller) RTL module. The verification environment is developed using the Universal Verification Methodology (UVM) and targets functional correctness, configurability, and output behavior of the interrupt controller based on its programmable registers and external inputs.

### 1.2 Scope

This verification effort covers the following aspects of the generic\_intc design:

* Functional behavior of the interrupt controller in response to interrupt inputs
* Configuration of enable, mask, and priority registers
* Proper reflection of interrupt status and clear behavior
* Correct generation of interrupt output (int\_out) in both pulse and level modes
* Polarity control and pulse width configuration
* Integration with UVM register model and protocol interfaces

The initial scope focuses on a sanity-level functional verification, covering register access, interrupt input generation, and output correctness using a scoreboard.

### 1.3 Design Overview

The generic\_intc module is a programmable interrupt controller that manages up to N interrupt sources. It supports:

* Individual enable and mask bits for each source
* Programmable priority per interrupt source
* Status indication and write-1-to-clear logic
* Output generation in either pulse or level mode
* Polarity control and pulse width configuration

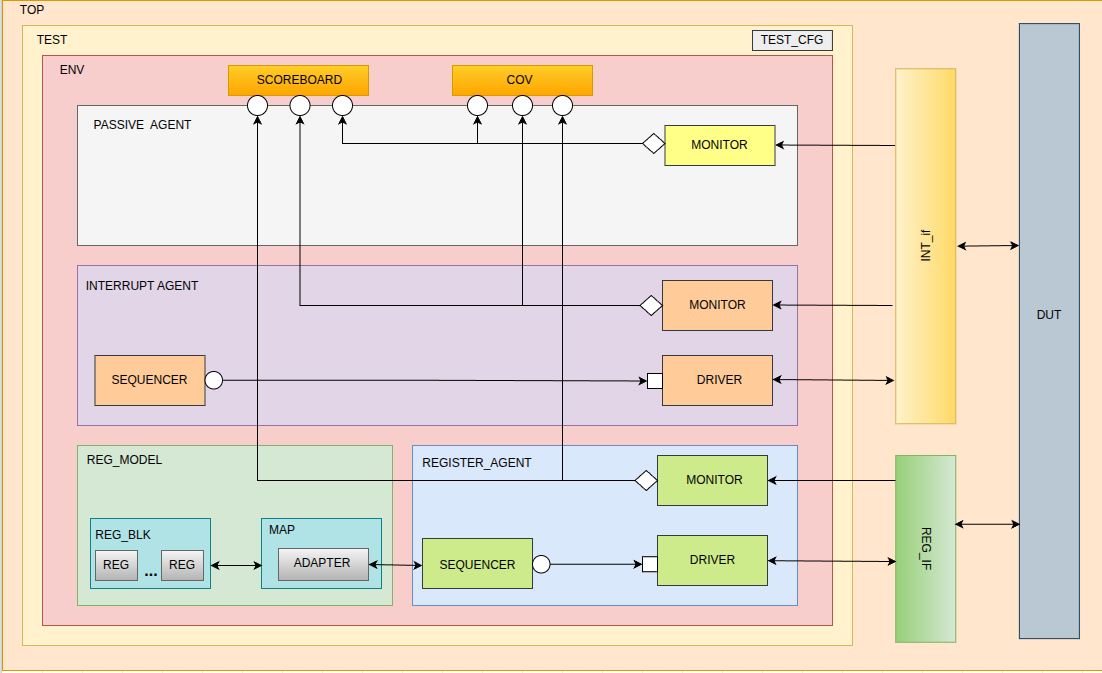
All configuration and status control are achieved via memory-mapped registers accessed through a standard register interface (reg\_if). The module generates a single output (int\_out) based on the highest priority pending and enabled interrupt, respecting mask and mode settings.

## 2. Verification Strategy

### 2.1 Verification Goals

The primary goals of this verification effort are:

* To ensure the functional correctness of the generic\_intc RTL design under different register configurations and interrupt input patterns.
* To verify that the interrupt output (int\_out) behaves according to the selected mode, priority, mask, enable, polarity, and pulse width.
* To validate correct register access using a UVM Register Abstraction Layer (RAL) model.

Section 3: UVM Testbench Architecture

### 1. Test

The top-level UVM test class controls the overall simulation. It instantiates the env, configures register models and sequences, and starts stimulus generation. It Coordinates both:

* The register configuration sequences
* The interrupt-driving sequences

### 2. Environment (env)

This is the central container that holds all UVM components:

* reg\_model
* register\_agent
* interrupt\_agent
* passive\_agent
* scoreboard
* coverage collector

### 3. Register Model (reg\_model)

This block implements the UVM RAL (Register Abstraction Layer) for the INTC:

* REG\_BLK: Includes all register classes (int\_enable, int\_mask, int\_status, out\_mode, etc.)
* MAP: Connects the register block to the DUT address map
* ADAPTER: Translates between register transactions and bus transactions
* The register model is used to drive and read configuration registers via reg\_if.

### 4. Register Agent

A UVM active agent responsible for accessing the DUT registers:

* Driver: Converts register sequence items into reg\_if transactions.
* Sequencer: Hosts register sequences (e.g., for initialization or random config).
* Monitor: Passively observes register activity.

This agent is bound to reg\_if.

### 5. Interrupt Agent

An active agent responsible for driving int\_in signals:

* Driver: Drives int\_in[N-1:0] values based on sequence items.
* Sequencer: Runs sequences that apply various interrupt stimulus patterns.
* Monitor: Samples and forwards observed int\_in activity to the scoreboard.

This agent is bound to int\_if.

### 6. Passive Agent

A passive agent that observes the int\_out output from the DUT:

* Monitor: Samples int\_out value from the DUT.
* Sends this sampled value to:
  + Scoreboard for checking correctness
  + Coverage collector for functional coverage

### 7. Scoreboard

* Compares the expected int\_out value (based on int\_in, enable/mask/prio registers) with the actual output observed by the passive agent.
* Checks for correctness of priority encoding, masking, and pulse/level behavior.

### 8. Coverage Collector

* Collects coverage on key control register values and output behavior:
  + Register configuration values (enable, mask, out\_mode, etc.)
  + Interrupt triggering scenarios
  + Output pulse/level correctness

### 9. Interfaces

* reg\_if: Connects the register agent with the DUT's memory-mapped register interface.
* int\_if: Connects the interrupt agent and passive agent to the DUT’s int\_in and int\_out signals.

### 4. Feature Extraction

|  |  |  |
| --- | --- | --- |
| Feature | Feature Description | Details |
| Parameterized Number of Interrupt Sources (N) | Supports configurable number of interrupt lines like 8, 16, 32, etc. | Testbench supports int\_in[N-1:0], int\_enable[N-1:0], etc. across varying N values. |
| Enable/Disable Control (int\_enable) | Enable individual interrupt sources using int\_enable register. | - Interrupt with enable = 0 should not affect int\_out.- Interrupt with enable = 1 should propagate to output if not masked. |
| Masking Support (int\_mask) | Allows masking individual interrupt sources. | - If interrupt is masked, int\_out must remain inactive even if interrupt is enabled and asserted.- When unmasked, int\_out should assert if interrupt is active and enabled. |
| Programmable Priority Per Source | Each source can have a different priority level, used for arbitration. | - When multiple interrupts are active, the one with the highest priority should be reflected in int\_out and int\_vector. |
| Status Register (int\_status) | Indicates which interrupts are pending and valid. | - Assert a valid interrupt → int\_status[i] = 1.- Write 1 to int\_clear[i] → int\_status[i] = 0. |
| Output Polarity Control (out\_polarity) | Output can be active-high or active-low. | - out\_polarity = 1 → int\_out = 1 for valid interrupt.- out\_polarity = 0 → int\_out = 0 for valid interrupt. |
| Output Mode (Level or Pulse) | Supports level-sensitive and pulse-based output signaling. | - Level mode: int\_out stays active as long as pending interrupt exists.- Pulse mode: Single pulse is generated per new interrupt. |
| Pulse Width Control (pulse\_width) | Defines width (duration) of pulse output in pulse mode. | - pulse\_width = X → int\_out pulse remains asserted for X clock cycles. |
| Clear Register (int\_clear) | Software-triggered clearing of pending interrupts. | - Writing 1 to int\_clear[i] clears int\_status[i]. - If no valid interrupts remain → int\_out is deasserted. |

## 5. Test Plan

The following test cases are defined to verify different functional scenarios of the generic\_intc module. Each test targets specific combinations of register configurations, input assertions, and output expectations to validate masking, priority, polarity, mode, pulse width, and corner cases.

|  |  |
| --- | --- |
| Test Name | Test Description |
| single\_int | Enable only one source, assert it, and check int\_out, int\_status, and priority. |
| multiple\_ints\_diff\_prio | Assert multiple sources with different priorities; verify highest one is selected. |
| pulse\_mode\_trigger | Set to pulse mode, assert input, verify int\_out as a 1-cycle pulse. |
| level\_mode\_trigger | Set to level mode, assert input, verify int\_out remains high while pending. |
| same\_priority\_test | Assert multiple sources with same priority; check deterministic priority handling. |
| dut\_reset\_test | Apply DUT reset; check default register values (e.g., INT\_ENABLE, MASK, STATUS = 0). |
| reg\_read\_write | Write and read all registers; verify data integrity. |
| clear\_register | Assert masked interrupt, then write to INT\_CLEAR; check INT\_STATUS becomes 0. |
| status\_register | Assert masked interrupt; check int\_out = 0, but INT\_STATUS = 1. |
| multiple\_ints\_one\_enabled | Enable only one source, assert several; only the enabled one should affect output. |
| all\_masked\_interrupts | Mask all sources, assert inputs; expect no int\_out. |
| all\_enabled\_unmasked | Enable and unmask all; assert multiple; check priority selection. |
| mask\_then\_trigger | Mask source first, then assert; expect no output. |
| active\_high\_polarity\_level | Set polarity = active high and mode = level; assert input; check int\_out. |
| active\_high\_polarity\_pulse | Set polarity = active high and mode = pulse; assert input; verify pulse output. |
| active\_low\_polarity\_level | Set polarity = active low and mode = level; assert low input; check output. |
| active\_low\_polarity\_pulse | Set polarity = active low and mode = pulse; assert low input; verify output pulse. |
| active\_high\_width\_test | Change pulse\_width (active\_high) and check actual pulse duration on int\_out. |
| active\_low\_width\_test | Change pulse\_width (active\_low) and check output pulse duration. |
| write\_readonly\_register | Attempt to write to INT\_STATUS (RO); verify it has no effect. |
| unknown\_value\_reg | Write 'x' to a register field; observe DUT behavior (should be ignored or flagged). |

* Compile Time and Run Time Switch Usage:

|  |  |  |
| --- | --- | --- |
| SWITCH | RUN/COMPILE TIME | USAGE |
| TRANSACTION\_COUNT | RUN TIME | IN MONITOR (INTERRUPT AND INT\_OUT) and SEQUENCES (INTERRUPUT AND REG) |
| int\_exp\_pkt | RUN TIME | IN SCB FOR COMPARISON WITH ACTUAL INT PKT |
| reg\_exp\_pkt | RUN TIME | IN SCB FOR COMPARISON WITH ACTUAL REG PKT |
| out\_exp\_pkt | RUN TIME | IN SCB FOR COMPARISON WITH ACTUAL OUT PKT |
| no\_of\_sources | COMPILE TIME | IN REG BLOCK, INT SEQ ITEM AND SEQ, INTERRUPT AND INT\_OUT MONITOR AND ENV |
| int\_in | RUN TIME | IN INTERRUPT SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| int\_enable | RUN TIME | IN REG SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| int\_mask | RUN TIME | IN REG SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| out\_mode | RUN TIME | IN REG SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| out\_polarity | RUN TIME | IN REG SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| pulse\_width | RUN TIME | IN REG SEQ FOR GETTING THE VALUE FROM PLUSARGS |
| WAVES | RUN TIME | FOR WAVEFORM DUMP |