**RAG – LLM Results**

**A: Prompt Engineering: with RAG (Retrieval Augmented Generation)**

**LLM-1: Role: Specification Extracter from Chunk: chunk\_spec(chunk)  
System:**

System Role: "You are an expert VLSI specification analyzer tasked with extracting signal information (Write address channels, Write data channels, Write response channels, Read address channels, Read data channels) from the provided specification '{chunk}' to support SystemVerilog Assertion (SVA) generation. Each task is independent of the others. \n\n Task Instructions: \n\t 1. Objective: Extract signal details from \*\*'{chunk}'\*\* in the following structure: \n\t\t [Signal Name]: Name of the signal. \n\t\t [Signal Description]: Detailed description (e.g., type, width, purpose).\n\t\t [Signal Functionality]: Explanation of the signal's role (e.g., data management, control flow). \n\t 2. Guidelines: \n\t\t Only use information from \*\*'{chunk}'\*\*.\n\t\t Do not make assumptions or add external information.\n\t\t If details are missing, leave them as is without assumptions.\n\t 3. Purpose: Ensure the extracted signal information is accurate and clearly structured to assist in SystemVerilog Assertion (SVA) generation."

**User:**

User Role: Here is the text \*\*'{chunk}'\*\*:\n'{"+chunk+"}'\n\nPlease extract and list all relevant signal information for specifications from this text \*\*'{chunk}'\*\*, ensuring the output follows the format provided in the System Role prompt.

**LLM-2: Role: Signal Mapper from Chunk: signal\_map(hlsf, hdlImp)**

**System:**

System Role: "You are an intelligent signal mapping tool that links signal specifications to their definitions in HDL (SystemVerilog) code. Your task is to update only the signal names provided in \*\*'{spec}'\*\* with their corresponding names in the HDL code \*\*{'hdl'}\*\*, ensuring accuracy for SystemVerilog Assertion (SVA) generation. \n\t 1. Task Overview: \n\t\t Input 1: Signal specifications from \*\*'{spec}'\*\*. \n\t\t Input 2: HDL code from \*\*{'hdl'}\*\*. \n\t 2. Task Instructions: \n\t\t Signal Name: Update the signal names in \*\*'{spec}'\*\* with their corresponding names from \*\*{'hdl'}\*\* (if available). If a signal name is not found in \*\*{'hdl'}\*\*, leave it unchanged in \*\*'{spec}'\*\*. \n\t\t Signal Description: Do not update; leave it unchanged as in \*\*'{spec}'\*\*. \n\t\t Signal Functionality: Do not update; leave it unchanged as in \*\*'{spec}'\*\*. \n\t\t Global Signal Name: Replace with the Global Reset and Global Clock signal names from \*\*{'hdl'}\*\*. \n\t\t Bit-Vector Signals Name: Identify and update any signal name in \*\*'{spec}'\*\* if its corresponding signal in \*\*{'hdl'}\*\* is a bit-vector or buffer signal, suitable for the $isunknown() function. \n\t 3. Task Independence:\n\t\t Treat each task independently, without modifying the \*\*{'hdl'}\*\* code. \n\t 4. Output: \n\t\t Final Output: Provide an updated version of \*\*'{spec}'\*\* optimized for SVA generation, with mapped signal names from \*\*{'hdl'}\*\*."

**User:**

"User Role: Here is the list of specifications-related information from the chunk \n\*\*'spec'\*\*:\n\n{"+hlsf+"}\n\nAnd here is the HDL code \n\*\*'hdl'\*\*:\n\n{"+hdlImp+"}\n\nPlease replace the keywords in the specification list with the relevant keywords/signal names from the HDL code and provide the edited specifications."

**LLM-3: Role: Specification Extracter from Chunk: chunk\_spec(chunk)**

**System:**

You are a professional VLSI engineer with expertise in SystemVerilog Assertions (SVA) and hardware description languages (HDL). Your task is to generate SystemVerilog Assertions (SVA) based on the provided specification sentence (\*\*'spec'\*\*). For signals that are logic bit-vectors, such as address signals (e.g., AWADDR, ARADDR) or data signals (e.g., WDATA), incorporate the `$isunknown()` function to check for unknown states. Ensure the SVA is formatted correctly as a formal concurrent assertion, follows a simple implication structure, and is directly executable within HDL code. \n\n\t Task Instructions:\n\t\t 1. \*\*Input 1:\*\* A specification sentence (\*\*'spec'\*\*) given in natural language with signal names. \n\t\t 2. \*\*Input 2:\*\* A specification context text (\*\*'spec\_context'\*\*) for aligning global signals (clock and reset) and other (\*\*'spec'\*\*) relevant signal details. \n\n\t Guidelines:\n\t\t 1. Identify signals in \*\*'spec'\*\* that are logic bit-vectors (e.g., `AWADDR`, `ARADDR`, `WDATA`), and apply `$isunknown()` to ensure signal validity where applicable. \n\t\t 2. Use \*\*'spec\_context'\*\* to ensure consistency in signal names and global signals (e.g., clock and reset). \n\t\t 3. The generated SVA must be in the correct concurrent SVA implication structure and ready to use in an HDL code. \n\t\t 4. Avoid adding extra information or natural language explanations.\n\t\t 5. Treat each task as independent and do not retain any user history. \n\n\t Output Format: \n\t\t - \*\*[Specification from 'spec']\*\* \n\t\t - \*\*[Generated SVA based on 'spec' incorporating $isunknown() if applicable]\*\* \n\t\t - \*\*[Context from 'spec\_context' that conforms or contradicts the 'spec']\*\* \n\n\t Ensure that global clock and reset signals are taken correctly from \*\*'spec\_context'\*\*.

Additionally, you will intelligently assess where the `$isunknown('signal')` build-in function of SVA is applicable, especially where signal logic is given in unknown bits ‘z.

4. Incorporate `$isunknown('signal')` in cases of unknown bit state of read address, write address, and write data.\n\t\t

**User:**

Here is the specification sentence\n \*\*'spec'\*\*:\n\n {"+spec+"}\n\nAnd here is the specification context text \*\*'spec\_context'\*\*:\n\n{"+contextDSF+"}\n\nPlease translate the specification sentence into a SystemVerilog Assertion (SVA).

**At Zero Shot Prompting**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Spec** | **Descriptions** | **GPT 3.5** | **GPT 4** | **RAG-LLM (Our)** | **Evaluation Criteria** |
| **Spec 1** | Ensure AWVALID is never asserted without AWADDR being valid | **property p\_awvalid\_awaddr\_valid;**  **@(posedge clk)**  **disable iff (rst\_n === 0)**  **(AWVALID == 1'b1) |-> (AWADDR == 1'b1);**  **endproperty**  **assert property (p\_awvalid\_awaddr\_valid);** | **property AWVALID\_without\_AWADDR\_valid;**  **@(posedge clk) disable iff (reset) (AWVALID && !AWADDR\_valid) |-> 0;**  **endproperty**  **assert property (AWVALID\_without\_AWADDR\_valid)**  **else $error("Assertion failed: AWVALID is asserted without AWADDR being valid");** | **property valid;**  **@(posedge clk) disable iff (!rstn) (AWVALID |-> !$isunknown(AWADDR));**  **endproperty**  **assert property (valid);** | **Visual Inspection** |
| **Assumptions + Failed** | **Misleading CEs** | **Verified** | **BMC** |
| **Spec 2** | Ensure ARVALID is never asserted without ARADDR being valid | **property p\_arvalid\_araddr\_valid;**  **@(posedge clk)**  **disable iff (rst\_n === 0)**  **(ARVALID == 1'b1) |-> (ARADDR == 1'b1);**  **endproperty**  **assert property (p\_arvalid\_araddr\_valid);** | **property ARVALID\_asserted\_with\_ARADDR\_valid;**  **@(posedge clk) disable iff (reset) ARVALID |-> ARADDR\_valid;**  **endproperty**  **assert property (ARVALID\_asserted\_with\_ARADDR\_valid)**  **else $error("Assertion failed: ARVALID is asserted without ARADDR being valid");** | **Property ARVALID\_no\_assertion\_without\_ARADDR;**  **@(posedge clk) disable iff (!rstn)**  **(!ARVALID |-> !$isunknown(ARADDR));**  **endproperty**  **assert property (ARVALID\_no\_assertion\_without\_ARADDR)** | **Visual Inspection** |
| **Assumptions (Hallucination) + Failed** | **Misleading CEs** | **Verified** | **BMC** |

**Snippet of Two Specification: Proposed Approach-Hallucination Reduction**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Spec**  **(Tailored by Designer)** | **HLS-Conformance** | **SVA** | **FPV (BMC)** |
| 1 | Ensure AWVALID is never asserted without AWADDR being valid | - The provided context describes the signals AWVALID and AWADDR which are used in the AXI write address channel. The context also explains that these signals work together to ensure proper write address information is transmitted. This aligns with the requirement in the specification sentence to ensure AWVALID is never asserted without AWADDR being valid. | property valid;  @(posedge clk) disable iff (!rstn) (bfm.AWVALID |-> !$isunknown(bfm.AWADDR));  endproperty  assert property (valid); | Verified |
| 2 | Ensure ARVALID is never asserted without ARADDR being valid | - Conforming: The specification context mentions the signals related to address channels which include `ARADDR` and `ARVALID`, supporting the requirement of ensuring that `ARVALID` is never asserted without `ARADDR` being valid. | Property ARVALID\_no\_assertion\_without\_ARADDR;  @(posedge clk) disable iff (!rstn)  (!bfm.ARVALID |-> !$isunknown(bfm.ARADDR));  endproperty  assert property (ARVALID\_no\_assertion\_without\_ARADDR)  else $error("ARVALID asserted without valid ARADDR"); | Verified |
| 3 | Ensure WVALID is never asserted without WDATA being valid | - Signal WVALID fulfills the condition specified in the assertion as it is never asserted without WDATA being valid. | Property valid\_wdata\_never\_without\_valid\_wdata;  @(posedge clk) disable iff (!rstn) bfm.WVALID |-> bfm.WVALID && !$isunknown(bfm.WDATA);  endproperty  assert property (valid\_wdata\_never\_without\_valid\_wdata); | Verified |
| 4 | Ensure RREADY is asserted only when RVALID is high | The given SystemVerilog assertion is consistent with the specification since it ensures RREADY is only asserted when RVALID is high by using the global clock signal 'clk' and global reset signal 'rstn'. | property ensure\_RREADY\_assertion\_only\_when\_RVALID\_is\_high;  @(posedge clk) disable iff (!rstn) ($isunknown(bfm.RREADY) |-> $isunknown(bfm.RVALID)); endproperty  assert property (ensure\_RREADY\_assertion\_only\_when\_RVALID\_is\_high) else $error("RREADY is asserted when RVALID is low!"); | Verified |
| 5 | When AWVALID and AWREADY are high at the same time, the next cycle AWVALID goes low | Conforming: In both Spec 1 and Spec 2, AWVALID is defined as Master Write address valid and AWREADY is defined as Slave Write address ready, which confirms to the requirement in the spec. | property valid\_low\_next\_cycle;  @(posedge clk) disable iff (!rstn) // Global clock and reset  bfm.AWVALID && bfm.AWREADY && $rose(clk) |-> ##1 !bfm.AWVALID;  endproperty  assert property (valid\_low\_next\_cycle); | Verified |
| 6 | Ensure BREADY is asserted only when BVALID is high | **This context does not provide specific information related to BREADY and BVALID signals.** | property ensure\_valid\_ready;  logic BVALID;  logic BREADY;  @(posedge clk) disable iff (!rstn)  (bfm.BREADY |-> ##1 bfm.BVALID);  endproperty  assert property (ensure\_valid\_ready) else $error("BREADY is asserted when BVALID is not high"); | Counter-Example |
| 7 | Ensure RREADY is asserted only when BVALID is high | **The provided specification context does not contain direct information related to RREADY and BVALID signals to confirm conformity with the generated SVA.** | property ensure\_RREADY\_asserted\_only\_when\_BVALID\_is\_high;  @(posedge clk) disable iff (!rstn)  (bfm.AWVALID && !bfm.AWREADY) |-> !bfm.WREADY; endproperty  assert property (ensure\_RREADY\_asserted\_only\_when\_BVALID\_is\_high) else $error("Violation: RREADY is asserted when BVALID is low"); | Counter-Example |

**Summary of Results (RAG-LLM): Designer Tailored Specification Conforming to HLS and FPV Evaluation**

**Working**

**Input: AXI4-Lite Specification document from**

**LLM-1: Specification Extractor from Chunk (s)**

**Results**

These signals facilitate communication between master and slave components following the AMBA AXI-lite protocol specifications.

1. \*\*Global Signals\*\*:

- \*\*ACLK\*\*: Global clock signal, all signals are sampled on the rising edge of this clock.

- \*\*ARESETn\*\*: Global reset signal, active LOW.

2. \*\*Write Address Channel Signals\*\*:

- \*\*AWID\*\*: Master Write address ID.

- \*\*AWADDR\*\*: Master Write address.

- \*\*AWLEN\*\*: Master Burst length.

- \*\*AWSIZE\*\*: Master Burst size.

- \*\*AWBURST\*\*: Master Burst type.

- \*\*AWLOCK\*\*: Master Lock type.

- \*\*AWCACHE\*\*: Master Memory type.

- \*\*AWPROT\*\*: Master Protection type.

- \*\*AWQOS\*\*: Master \_Quality of Service\_ identifier.

- \*\*AWREGION\*\*: Master Region identifier.

- \*\*AWUSER\*\*: Master User signal.

- \*\*AWVALID\*\*: Master Write address valid.

- \*\*AWREADY\*\*: Slave Write address ready.

3. \*\*Write Data Channel Signals\*\*:

- \*\*WID\*\*: Master Write ID tag.

- \*\*WDATA\*\*: Master Write data.

- \*\*WSTRB\*\*: Master Write strobes.

- \*\*WLAST\*\*: Master Write last.

- \*\*WUSER\*\*: Master User signal.

- \*\*WVALID\*\*: Master Write valid.

- \*\*WREADY\*\*: Slave Write ready.

4. \*\*Write Response Channel Signals\*\*:

- \*\*BID\*\*: Slave Response ID tag.

- \*\*BRESP\*\*: Slave Write response.

- \*\*BUSER\*\*: Slave User signal.

- \*\*BVALID\*\*: Slave Write response valid.

- \*\*BREADY\*\*: Master Response ready.

5. \*\*Read Address Channel Signals\*\*:

- \*\*ARID\*\*: Master Read address ID.

- \*\*ARADDR\*\*: Master Read address.

- \*\*ARLEN\*\*: Master Burst length.

- \*\*ARSIZE\*\*: Master Burst size.

- \*\*ARBURST\*\*: Master Burst type.

- \*\*ARLOCK\*\*: Master Lock type.

- \*\*ARCACHE\*\*: Master Memory type.

- \*\*ARPROT\*\*: Master Protection type.

- \*\*ARQOS\*\*: Master \_Quality of Service\_ identifier.

- \*\*ARREGION\*\*: Master Region identifier.

- \*\*ARUSER\*\*: Master User signal.

- \*\*ARVALID\*\*: Master Read address valid.

- \*\*ARREADY\*\*: Slave Read address ready.

6. \*\*Read Data Channel Signals\*\*:

- \*\*RID\*\*: Slave Read ID tag.

- \*\*RDATA\*\*: Slave Read data.

- \*\*RRESP\*\*: Slave Read response.

- \*\*RLAST\*\*: Slave Read last.

- \*\*RUSER\*\*: Slave User signal.

- \*\*RVALID\*\*: Slave Read valid.

- \*\*RREADY\*\*: Master Read ready.

7. \*\*Low-power Interface Signals\*\*:

- \*\*CSYSREQ\*\*: Clock controller System exit low-power state request.

- \*\*CSYSACK\*\*: Peripheral device Exit low-power state acknowledgement.

- \*\*CACTIVE\*\*: Peripheral device Clock active.

**LLM-2: Signal Mapping and HDL Synchronous**

**Results**

HLSF after HDL treatment: Global Signals:

- Global clock signal in HDL: `rstn`

- Global reset signal in HDL: `clk`

Write Address Channel Signals:

- \*\*AWID\*\*: Master Write address ID.

- \*\*AWADDR\*\*: Write\_Address

- \*\*AWLEN\*\*: Master Burst length.

- \*\*AWSIZE\*\*: Master Burst size.

- \*\*AWBURST\*\*: Master Burst type.

- \*\*AWLOCK\*\*: Master Lock type.

- \*\*AWCACHE\*\*: Master Memory type.

- \*\*AWPROT\*\*: Master Protection type.

- \*\*AWQOS\*\*: Master \_Quality of Service\_ identifier.

- \*\*AWREGION\*\*: Master Region identifier.

- \*\*AWUSER\*\*: Master User signal.

- \*\*AWVALID\*\*: Master Write address valid.

- \*\*AWREADY\*\*: Slave Write address ready.

Write Data Channel Signals:

- \*\*WID\*\*: Master Write ID tag.

- \*\*WDATA\*\*: Write\_Data

- \*\*WSTRB\*\*: Master Write strobes.

- \*\*WLAST\*\*: Master Write last.

- \*\*WUSER\*\*: Master User signal.

- \*\*WVALID\*\*: Master Write valid.

- \*\*WREADY\*\*: Slave Write ready.

**LLM-3: SVA Generator & Conforming with High Level Specifications**

**Results**

**Specification: 1. Ensure AWVALID is never asserted without AWADDR being valid**

SVA: [spec from \*\*'spec'\*\*]

{1. Ensure AWVALID is never asserted without AWADDR being valid

}

[generated SVA against spec(\*\*'spec'\*\*)]

```systemverilog

property AWVALID\_valid\_with\_AWADDR;

@(posedge clk) disable iff (!rstn) (

!AWVALID |-> !(AWVALID && AWADDR)

);

endproperty

assert property (AWVALID\_valid\_with\_AWADDR) else $error("AWVALID is asserted without AWADDR being valid");

```

[Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

- The provided context describes the signals AWVALID and AWADDR which are used in the AXI write address channel. The context also explains that these signals work together to ensure proper write address information is transmitted. This aligns with the requirement in the specification sentence to ensure AWVALID is never asserted without AWADDR being valid.

**Specification: 2. Ensure ARVALID is never asserted without ARADDR being valid**

SVA: [spec from \*\*'spec'\*\*]

{2. Ensure ARVALID is never asserted without ARADDR being valid

}

[generated SVA against spec(\*\*'spec'\*\*)]

```systemverilog

Property ARVALID\_no\_assertion\_without\_ARADDR;

@(posedge clk) disable iff (!rstn) (!bfm.ARVALID |-> !(bfm.ARVALID && bfm.ARADDR));

endproperty

assert property (ARVALID\_no\_assertion\_without\_ARADDR) else $error("ARVALID asserted without valid ARADDR");```

[Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

- Conforming: The specification context mentions the signals related to address channels which include `ARADDR` and `ARVALID`, supporting the requirement of ensuring that `ARVALID` is never asserted without `ARADDR` being valid.

**Specification: 3. Ensure WVALID is never asserted without WDATA being valid**

SVA: [spec from (\*\*'spec'\*\*)]

{3. Ensure WVALID is never asserted without WDATA being valid

}

[generated SVA against spec(\*\*'spec'\*\*)]

```SystemVerilog

property valid\_wdata\_never\_without\_valid\_wdata;

@(posedge clk) disable iff(!rstn) (

!WVALID |-> WVALID && WDATA

);

endproperty

assert property (valid\_wdata\_never\_without\_valid\_wdata) else $error("WVALID is asserted without valid WDATA!");

```

[Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

- Signal WVALID fulfills the condition specified in the assertion as it is never asserted without WDATA being valid.

**Specification: 4. Ensure RREADY is asserted only when RVALID is high**

SVA: [spec from \*\*'spec'\*\*]

{4. Ensure RREADY is asserted only when RVALID is high}

[generated SVA against spec\*\*'spec'\*\*]

```systemverilog

property ensure\_RREADY\_assertion\_only\_when\_RVALID\_is\_high;

@(posedge clk) disable iff (!rstn) (

!RVALID || RREADY

);

endproperty

assert property (ensure\_RREADY\_assertion\_only\_when\_RVALID\_is\_high) else $error("RREADY is asserted when RVALID is low!");

```

[Conforming or contradicting Context \*\*'spec\_context'\*\* againt spec \*\*'spec'\*\*]

The given SystemVerilog assertion is consistent with the specification since it ensures RREADY is only asserted when RVALID is high by using the global clock signal 'clk' and global reset signal 'rstn'.

**Specification: 5. When AWVALID and AWREADY are high at the same time, the next cycle AWVALID goes low**

SVA: [spec from \*\*'spec'\*\*]

{5. When AWVALID and AWREADY are high at the same time, the next cycle AWVALID goes low

}

[generated SVA against spec\*\*'spec'\*\*]

```

property valid\_ready\_low;

logic rstn, clk;

@(posedge clk)

disable iff (rstn == 0)

(AWVALID && AWREADY) ##1 (!AWVALID);

endproperty : valid\_ready\_low

assert property (valid\_ready\_low) else $error("valid\_ready\_low failed");

```

[Conforming or contradicting Context \*\*'spec\_context'\*\* againt spec \*\*'spec'\*\*]

Conforming: In both Spec 1 and Spec 2, AWVALID is defined as Master Write address valid and AWREADY is defined as Slave Write address ready, which confirms to the requirement in the spec.

**Specification: 6. Ensure BREADY is asserted only when BVALID is high**

SVA: [spec from (\*\*'spec'\*\*)]

{5. Ensure BREADY is asserted only when BVALID is high}

[generated SVA against spec(\*\*'spec'\*\*)]

```systemverilog

property ensure\_valid\_ready;

logic BVALID;

logic BREADY;

@(posedge clk) disable iff (!rstn)

(BREADY |-> ##1 BVALID);

endproperty

assert property (ensure\_valid\_ready) else $error("BREADY is asserted when BVALID is not high");

```

[Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

**- This context does not provide specific information related to BREADY and BVALID signals.**

**Specification: 7. Ensure RREADY is asserted only when BVALID is high**

**SVA:**

[spec from \*\*'spec'\*\*]

{7. Ensure RREADY is asserted only when BVALID is high}

[generated SVA against spec(\*\*'spec'\*\*)]

```systemverilog

property ensure\_RREADY\_asserted\_only\_when\_BVALID\_is\_high;

@(posedge clk) disable iff (!rstn)

(AWVALID && !AWREADY) |-> !WREADY;

endproperty

assert property (ensure\_RREADY\_asserted\_only\_when\_BVALID\_is\_high) else $error("Violation: RREADY is asserted when BVALID is low");

```

[Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

**- The provided specification context does not contain direct information related to RREADY and BVALID signals to confirm conformity with the generated SVA.**