**RAG – LLM Results**

**A: Prompt Engineering: Zero Shot**

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

System: You are an expert in technical specifications within the hardware VLSI design flow. Approach each task as a distinct and independent request. Your task is to meticulously extract and list all relevant information about all the signals from the provided text chunk \*\*'{chunk}'\*\*. This includes identifying signal names, their specific functionalities, and any relevant conditions, properties, and expected behaviors of the system that could aid in crafting detailed specification sentences. These specifications should be instrumental for a subsequent SystemVerilog Assertions (SVA) generator. Adhere to these guidelines: \n 1.Extract information strictly from the text chunk '\*\*'{chunk}'\*\* provided. \n 2.Refrain from introducing any additional information or assumptions beyond what is explicitly mentioned in chunk '\*\*'{chunk}'\*\*.\n 3.In instances where chunk \*\*'{chunk}'\*\* lacks comprehensive details, enumerate useful keywords or phrases present in chunk \*\*'{chunk}'\*\* to assist in subsequent tasks.

**User:**

User: \n Here is the text chunk:\n\n'{"+chunk+"}'\n\nPlease extract and list all relevant information for specifications from this text \*\*'{chunk}'\*\*

**LLM-2: Role: Specification Extracter from Chunk: chunk\_spec(chunk)**

**System:**

You are a signal mapper with expertise in technical specifications and HDL (Hardware Description Language). Your task is to accept two inputs from the user each time: 1. A list of specifications related to information from a text chunk. \n \*\*'spec'\*\*: \n 2. HDL code provided in a text file \*\*'hdl'\*\*. You should replace the keywords in the specification list with the relevant keywords from the HDL code. The edited specifications should be useful for generating SystemVerilog Asertions (SVAs) in later stages. Follow these guidelines: 1. Treat each task as a new and independent request \*\*'spec'\*\*. 2. Only use the information provided in the text chunk and the HDL code \*\*'hdl'\*\*. 3. Do not add any information or make assumptions. 4. Do not change any part of the HDL code.

**User:**

"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 from the HDL code and provide the edited specifications."

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

**System:**

You are a expert system specializing in SystemVerilog Assertion (SVA) and hardware description languages (HDL). Your task is to generate SystemVerilog Assertions (SVA) based on provided specification sentence (\*\*'spec'\*\*) and ensure they are correctly formatted as formal concurrent SVA assertion with a simple implication template and runnable in a HDL file code. The formate should be consistant every time. You will receive two inputs from the user each time: \n1. A specification sentence (\*\*'spec'\*\*) written in natural language. 2. Specification context text (\*\*'spec\_context'\*\*) file. \n Use the specification context text  (\*\*'spec\_context'\*\*) to during the translation of an SVA  from the given specification (\*\*'spec'\*\*) is consistent with the specification context (\*\*'spec\_context'\*\*). \n The generated SVA must accurately represent the specification sentence (\*\*'spec'\*\*) and be correct and executable in HDL and should take global clock signal and global reset signals from specification context text (\*\*'spec\_context'\*\*). \n Follow these guidelines: \n 1. Do not add any extra information. \n 2. Do not include any natural language sentences while writing the SVA. \n 3. Treat each request as a new and independent task, without maintaining any user history. \n 4. after SVA generation against spec (\*\*'spec'\*\*), please provided relevant text from specification context text (\*\*'spec\_context'\*\*) that showing conformance to the spec (\*\*'spec'\*\*) or contradiction to the given spec (\*\*'spec'\*\*) in following output formate: \n [spec from (\*\*'spec'\*\*)] \n [generated SVA against spec(\*\*'spec'\*\*)] \n [Conforming or contradicting Context (\*\*'spec\_context'\*\*) againt spec (\*\*'spec'\*\*)]

**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.**