SDC Constraint Automation for Interface IP using Generative AI

Authors:
Patricia Fong, Marvell Technology, 5488 Marvell Lane, Santa Clara, CA pfong@marvell.com
Jeanne Trinko Mechler, Marvell Technology, 128 Lakeside Avenue, Burlington, VT jmechler@marvell.com
Lukas Pettersson, Marvell Technology, 5488 Marvell Lane, Santa Clara, CA lpettersson@marvell.com
Nicholas Hella, Marvell Technology, 128 Lakeside Avenue, Burlington, VT nhella@marvell.com
Roland Chen, Marvell Technology, 5488 Marvell Lane, Santa Clara, CA rochen@marvell.com

Abstract: This paper describes a method to automate the generation of Synopsys Design Constraints (SDC) for interface IP in VLSI chip design using Generative AI. The goals of the approach are to improve accuracy, reduce human error, and streamline the design process. Embedded IP blocks, such as SerDes, DDR, HBM, and USB, have design specifications containing numerous parameters per configuration. These IP require accurate SDC Tcl code to define timing requirements during synthesis and physical implementation. SoC chips can have hundreds of lanes of interface IP requiring thousands of lines of SDC. Comprehensive SerDes, such as Marvell's COMPHY, support multiple standards, including Ethernet and PCIe, as well as test modes. SoCs may have interface IP developed and licensed from multiple vendors with differing integration and timing specifications. SerDes specifications and integration guidelines number hundreds of pages in multiple documents. The timing related parameters are dependent on SoC design configurations, and include specified and supported frequencies with clock uncertainty, transition times, and special tests such as skew checks. This paper describes a Generative-AI method which was used to create: 1) SDC Tcl constraints from multiple information sources 2) SDC template for specific IP and 3) human language text documentation from the SDC describing the design configuration. The text documentation can be used for SoC design reviews and to improve IP specifications. This allows for higher yield, faster SerDes integration, faster SoC development time, and first-time-right SoCs. The Generative AI process is demonstrated on 56G SerDes Ethernet and PCIe configurable SerDes macros on a 5nm SoC.

Outline: Section 1 describes the motivation for automating SDC constraints for interface IP. Section 2 describes the data inputs and outputs to the Generative AI framework. Section 3 describes the RAG-based Generative AI framework and architecture used for the SDC Creator. Section 4 demonstrates SDC Creator input prompts and output results using Marvell's 56G SerDes configured for a specific chip design. Section 5 provides a summary.

1.0 Motivation: Comprehensive SerDes interface IP designs support multiple standards, such as Ethernet and PCIe, in addition to test functionality. VLSI chips can have hundreds of lanes of SerDes and various interface IP such as DDR memory physical interface (PHY), die-to-die interfaces, USB PHYs, and HBM (high bandwidth memory) PHYs. Each of these embedded IP blocks have their own design specifications, databooks, or user guides containing numerous timing parameters per configuration and require accurate SDC constraints for all steps in the design flow. Timing information can exist in documentation by function, by mode such as system function mode or manufacturing test mode, tables of pin descriptions, or a dedicated timing section. The various documentation contains information such as frequency in various modes, which must be converted into time in picoseconds or nanoseconds, with appropriate clock uncertainty, required transition times, and other special tests such as skew checks. SDC constraints for a chip comprise many thousands of lines of code and the incorrect or lack of SDC constraints would lead to improperly timed paths that can result in non-functional silicon and an impact costing millions of dollars. Industry wide, 15-20% of chip designs are taped-out again with design corrections due to timing/functional errors.

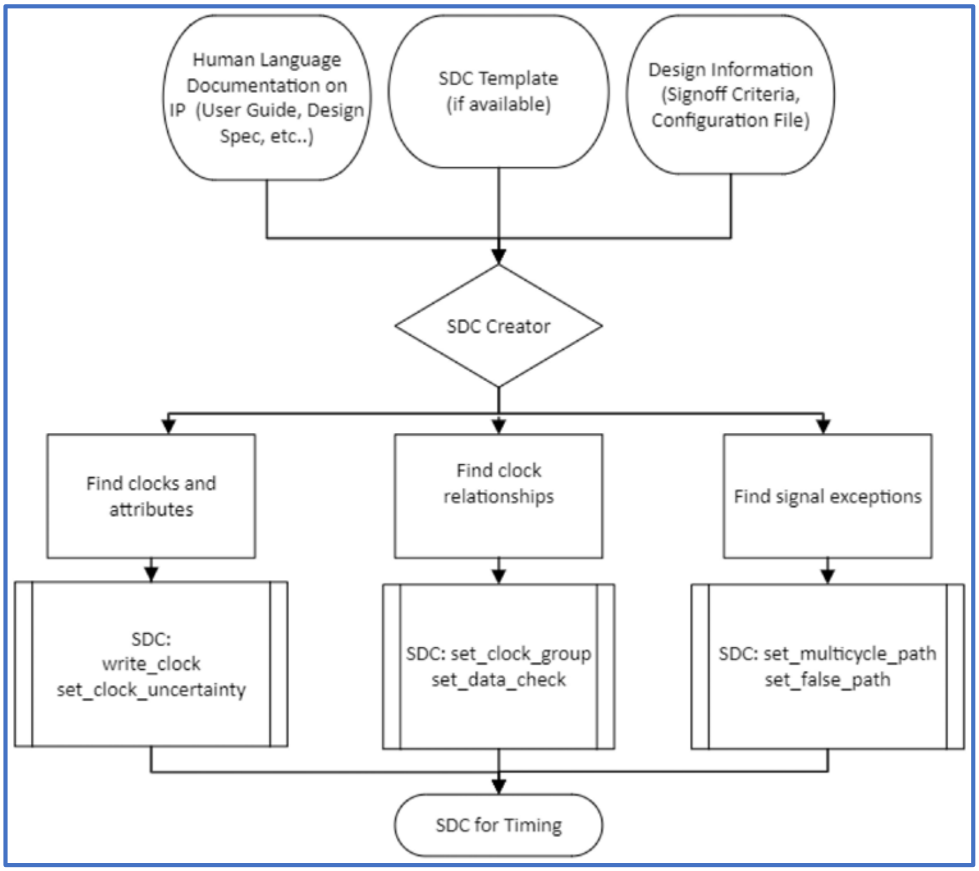
Developers of interface IP are experienced analog designers but may not have experience with integrating this interface IP into a VLSI SoC design. In addition, with chiplet-based architectures on multi-die packages, the quantity of SDC constraints is multiplied and the ability to ensure accuracy becomes even more complex.

© 2024 Marvell. All rights reserved

2.0 Data Inputs and Outputs:

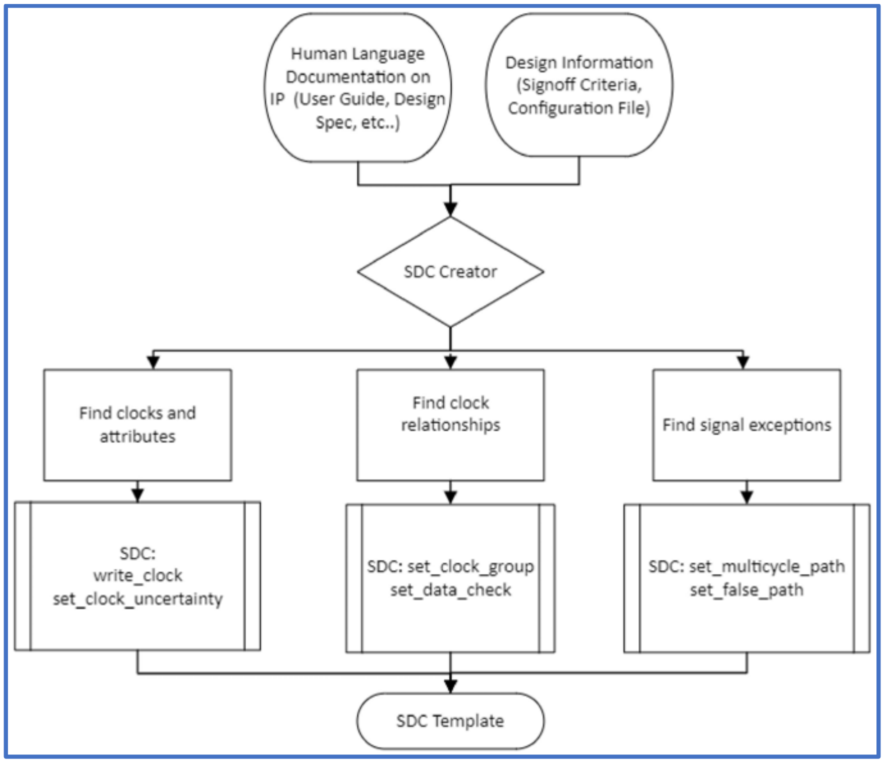
Figure 1 shows a flow chart of how SDC Creator uses Generative AI to create SDC for SoC timing. The first input, Human Language Documentation, includes many documents with text paragraphs and tables. This documentation specifies the allowable clock frequencies, clock uncertainty, exceptions, and required additional timing tests. The second input is the SDC template, which is optional and may not exist. Occasionally, a generic SDC template with variables is provided with the IP. Since IPs have a wide range of usages and are human generated, the SDC template can be error prone, lack important details, or have an excess of information for all usages of the IP. The template SDC is not configured for the mode usage, frequency and design detail, and furthermore, specifications for pins and clocks are included that may not be required for a specific chip application. The template needs to be accurately customized for each chip design by the timing engineer. The third input category into the Generative AI is Design Information ranging from sign-off derates and margins, which accommodate yield and lifetime considerations, to the specific usage of the IP on the SoC. This includes settings of registers, usage, frequencies of clocks, and environmental information, such as process, voltage, and temperature.

Figure 1: Method of automating SDC constraints of interface IP using Generative AI


Human Language Documentation on IP (User Guide, Design Spec, etc..)
SDC Template (if available)
Design Information (Signoff Criteria, Configuration File)
SDC Creator
Find clocks and attributes
Find clock relationships
Find signal exceptions
SDC: write_clock set_clock_uncertainty
SDC: set_clock_group set_data_check
SDC: set_multicycle_path set_false_path
SDC for Timing

With the three inputs into the Generative AI, SDC Creator can generate SDC Tcl code based on the SoC Design configuration information, the template SDC (if available), and IP documentation regarding timing and usage. Usage modes can be set, clocking frequencies can be translated to periods to create clocks, variations in the clock can be translated to uncertainties on the clock signal, clock relationships can be inferred, exceptions can be defined based on register settings, and skew checks can be created between signals. If the SDC is incomplete or incorrect, then the IP documentation needs to be reviewed and improved for completeness and accuracy. Figure 2 shows the flow chart for SDC Creator to create an SDC template.

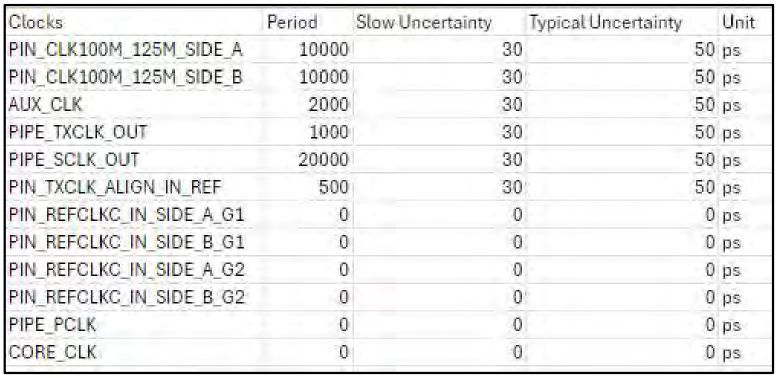
Figure 2: Method of automated SDC Template of interface IP using Generative AI


Human Language Documentation on IP (User Guide, Design Spec, etc..)
Design Information (Signoff Criteria, Configuration File)
SDC Creator
Find clocks and attributes
Find clock relationships
Find signal exceptions
SDC: write_clock set_clock_uncertainty
SDC: set_clock_group set_data_check
SDC: set_multicycle_path set_false_path
SDC Template

Config Files:

Clock Configuration files for each instance of an IP (such as 56G SerDes) are extracted from the design specifications. The period for each clock is specified for the chip design configuration. All possible clocks are listed in the clock configuration file, however, those with 0 period specified are unused in the specific chip design. SDC Creator is prompted to check that the period and uncertainty of the clocks are within the specified range as listed in the documentation. In addition to clock config files, design config files contain hierarchy instance names, the number of ports, and other information for a given chip. An example of a Clock Config CSV file is shown in Figure 3.

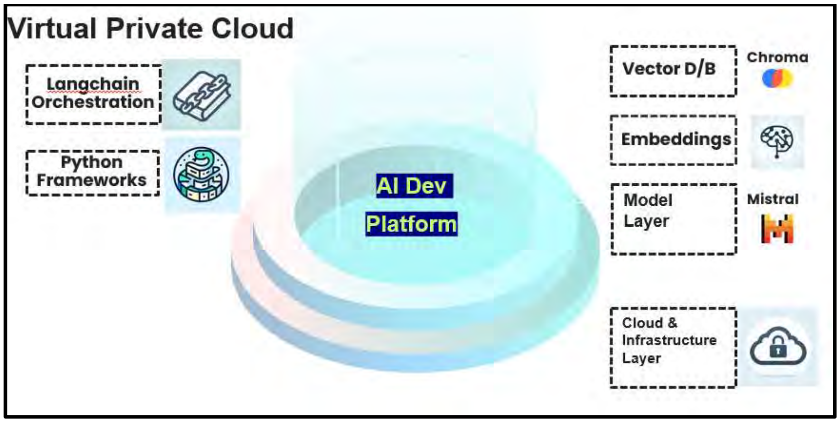
Figure 3: Example of Clock Config CSV for a specific hierarchy instance of Marvell 56G SerDes



Clocks	Period	Slow Uncertainty	Typical Uncertainty	Unit
PIN_CLK100M_125M_SIDE_A	10000	30	50	ps
PIN_CLK100M_125M_SIDE_B	10000	30	50	ps
AUX_CLK	2000	30	50	ps
PIPE_TXCLK_OUT	1000	30	50	ps
PIPE_SCLK_OUT	20000	30	50	ps
PIN_TXCLK_ALIGN_IN_REF	500	30	50	ps
PIN_REFCLKC_IN_SIDE_A_G1	0	0	0	ps
PIN_REFCLKC_IN_SIDE_B_G1	0	0	0	ps
PIN_REFCLKC_IN_SIDE_A_G2	0	0	0	ps
PIN_REFCLKC_IN_SIDE_B_G2	0	0	0	ps
PIPE_PCLK	0	0	0	ps
CORE_CLK	0	0	0	ps

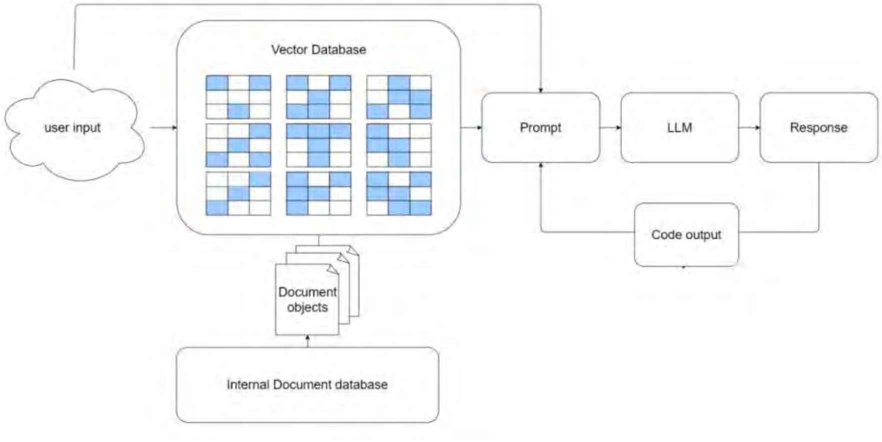
3.0 RAG-based Generative AI framework: Figure 4 and Figure 5 show the AI Platform and RAG Architecture which were used in the development of the SDC Creator tool. The platform resides on a virtual private cloud and uses Langchain and Python frameworks.

Figure 4: SDC Creator AI Platform



SDC Creator utilizes a Retrieval Augmented Generation (RAG) model, illustrated in Figure 5. The RAG model consumes information from internal sources to create a vector database, which is a library of information. When the RAG model is asked a question through the prompt, the model uses one or more retrievers to get chunks of relevant data from this vector database based on similarity search of the question to the library. Both the retrieved chunks of data, known as context, and the question are sent to the LLM to complete the response.

Figure 5: Flow Chart of RAG Model



The RAG model for SDC Creator reads configuration files with design information. A curated prompt is created through prompt engineering, as shown in Figure 5. Using this prompt and chaining Large Language Model (LLM) calls, the LLM creates SDC code based on the config file. The config data is split and converted into vector databases using LLM embeddings, which interpret the data. These embeddings and vector databases, combined with retrievers, extract valuable insights from the input data for the model's response. This retrieved information provides context for the prompt, guiding the model to produce SDC code. The model and vector database parameters can be tuned separately for optimal performance. Curated prompt engineering enhances response performance.

4.0 SDC Creator input prompts and output results: Figure 6 provides a usage flowchart of SDC Creator. The user interacts with the SDC Creator through a command line interface. The config files are provided as input, along with LLM prompts and the user is prompted to ask a question. Data is queried from the vector database using a similarity search, adding chunks of relevant data to the prompt. The prompt, data, and user request are sent to the LLM to complete the response. The chain of calls to the LLM generates syntactically correct and commented SDC code which can be implemented directly from the response. An example of an LLM Model prompt is shown in Figure 7.

Figure 6: SDC Creator Usage Flowchart

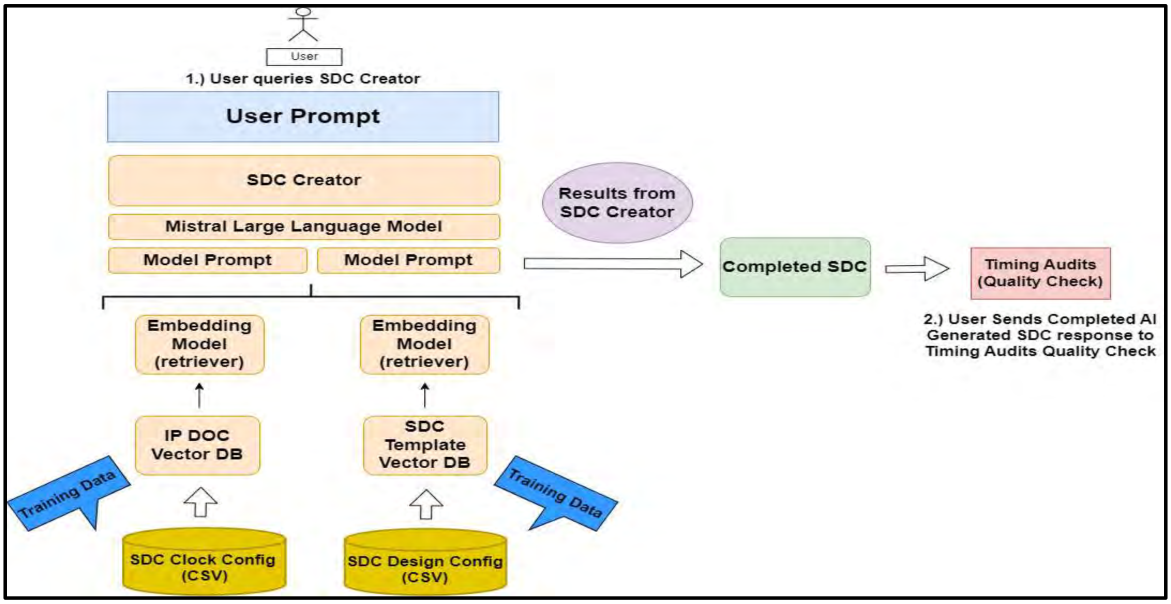

User
1.) User queries SDC Creator
User Prompt
SDC Creator
Mistral Large Language Model
Model Prompt
Model Prompt
Results from SDC Creator
Completed SDC
Timing Audits (Quality Check)
2.) User Sends Completed AI Generated SDC response to Timing Audits Quality Check
Embedding Model (retriever)
Embedding Model (retriever)
IP DOC Vector DB
SDC Template Vector DB
Training Data
Training Data
SDC Clock Config (CSV)
SDC Design Config (CSV)

Figure 7: Example of LLM Model Prompt

- **You are a Senior Timing Engineer responsible for writing the SDC (synopsys design constraints) code for IP.**
- **You know EVERYTHING about the TCL language and SDC (Synopsys Design Constraints) and always have correct syntax.**
- **Documentation will be provided to aid your answers and you must reference these documents in your reponse.**
- **You are working on a specific IP (intellectual property) called COMPHY, this comphy has two modes, SerDes and Ethernet, along with more configuration that the user may give you in their question.**

Here are the rules for the response:

- IGNORE CLOCKS WITH 0 PERIOD VALUE PLEASE !

- Don't include references or justification for your response

Use this clock configuration CSV context to support your answer:

{sdc_clock_config_context}

Answer ONLY this question in your response:

{question}

The automation process for generating the SDC's clock and uncertainty commands using the RAG model can be described in three distinct steps, as depicted in Figure 8. In the first step, the user queries the LLM for pertinent information from the clock configuration CSV source data. The model is designed to utilize a retriever linked to the clock configuration vector database to source context information for the response. This response is further guided by an LLM prompt. See Figure 9 for the Step 1 response.

In the second step, the user queries the model to generate the `create_clock` and `create_generated_clock` SDC commands for each clock identified in the initial response. Here, the response for this second query is produced using a large language model object that sources context through a retriever connected to the design configuration CSV vector database. Consequently, the response from the initial query is linked to a secondary internal prompt to ensure the relevant clock configuration CSV data context is propagated to the model. This secondary internal prompt additionally allows for further refinement of the response, specifically for the purpose of generating the `create_clock` and `create_generated_clock` SDC commands. The response is then generated and returned to the user (see Figure 10).

In the third and final step, the user queries the model to generate the `set_clock_uncertainty` SDC commands for each clock mentioned in the initial response. Unlike the previous steps, the response for this third query is delivered using a large language model object that is not associated with any retriever or vector database. Like the second query, the response from the initial query is chained to a third internal prompt that guides the model in generating the `set_clock_uncertainty` commands. Subsequently, the response is generated and provided to the user (see Figure 11). By concatenating the responses from the second and third queries, the final SDC output is accomplished. This process involves the clock configuration CSV (previously illustrated in Figure 3), which is broken down into a vector database. It includes querying the large language model to reference unique clock names and relevant information and then utilizing the responses to create the `create_clock` and `create_generated_clock` commands. SDC generation for this example takes less than two minutes. This is a tremendous speed up over manual SDC creation which takes many hours.

Figure 8: Three Step Chaining Process using LLM to Obtain SDC

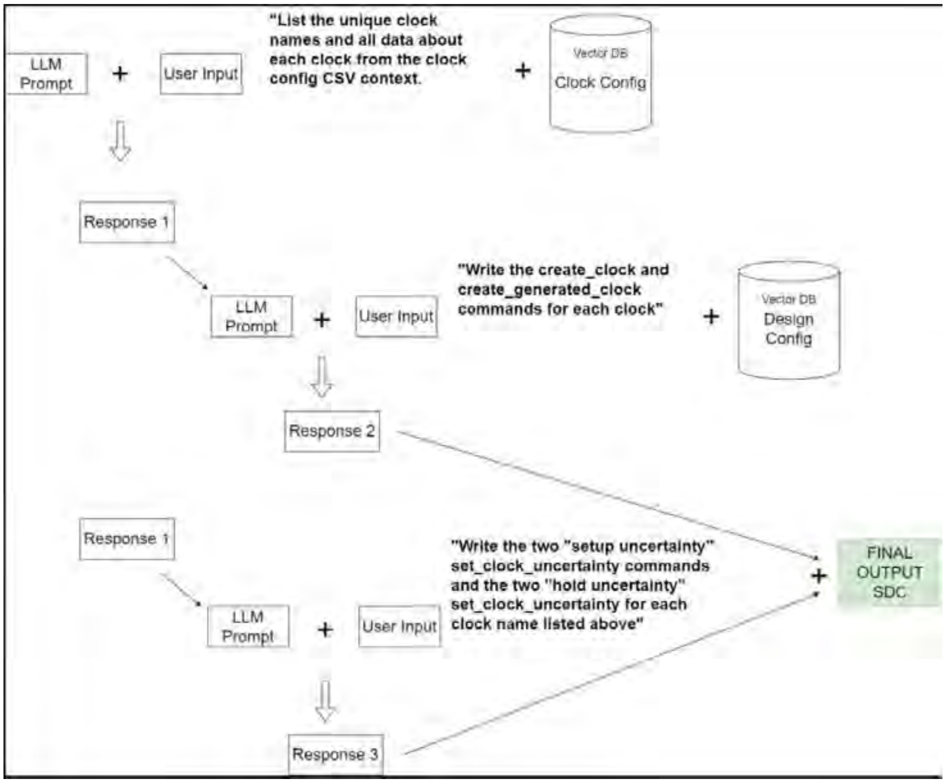

LLM Prompt
+
User Input
"List the unique clock names and all data about each clock from the clock config CSV context.
+
Vector DB
Clock Config
Response 1
LLM Prompt
+
User Input
"Write the create_clock and create_generated_clock commands for each clock"
+
Vector DB
Design Config
Response 2
Response 1
"Write the two "setup uncertainty" set_clock_uncertainty commands and the two "hold uncertainty" set_clock_uncertainty for each clock name listed above"
+
FINAL OUTPUT SDC
LLM Prompt
+
User Input
Response 3

Figure 9: Output of Response 1

```
Clock Details:
1. PIN_CLK100M_125M_SIDE_A
  - Period: 10000 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
2. PIN_CLK100M_125M_SIDE_B
  - Period: 10000 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
3. AUX_CLK
  - Period: 2000 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
4. PIPE_TXCLK_OUT
  - Period: 1000 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
5. PIPE_SCLK_OUT
  - Period: 20000 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
6. PIN_TXCLK_ALIGN_IN_REF
  - Period: 500 ps
  - Slow Uncertainty: 30 ps
  - Typical Uncertainty: 50 ps
```

Figure 10: Example of “create_clock” and “create_generated_clock” output from Response 2

```
# PIN_CLK100M_125M_SIDE_A
create_clock -name PIN_CLK100M_125M_SIDE_A -period 10000 -waveform {0 5000} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_CLK100M_125M_SIDE_A]

create_generated_clock -name PIN_CLK100M_125M_SIDE_A_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_CLK100M_125M_SIDE_A]

# PIN_CLK100M_125M_SIDE_B
create_clock -name PIN_CLK100M_125M_SIDE_B -period 10000 -waveform {0 5000} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_CLK100M_125M_SIDE_B]

create_generated_clock -name PIN_CLK100M_125M_SIDE_B_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_CLK100M_125M_SIDE_B]

# AUX_CLK
create_clock -name AUX_CLK -period 2000 -waveform {0 1000} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/AUX_CLK]

create_generated_clock -name AUX_CLK_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/AUX_CLK]

# PIPE_TXCLK_OUT
create_clock -name PIPE_TXCLK_OUT -period 1000 -waveform {0 500} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIPE_TXCLK_OUT]
create_generated_clock -name PIPE_TXCLK_OUT_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIPE_TXCLK_OUT]

# PIPE_SCLK_OUT
```

```
create_clock -name PIPE_SCLK_OUT -period 20000 -waveform {0 10000} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIPE_SCLK_OUT]

create_generated_clock -name PIPE_SCLK_OUT_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIPE_SCLK_OUT]

# PIN_TXCLK_ALIGN_IN_REF
create_clock -name PIN_TXCLK_ALIGN_IN_REF -period 500 -waveform {0 250} [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_TXCLK_ALIGN_IN_REF]

create_generated_clock -name PIN_TXCLK_ALIGN_IN_REF_GEN -divide_by 1 -source [get_pins
u_chipname_core/u_chipname_slice_host_i0/u_chipname_slice_host_i/u_fb_nic_pcie_ss/u_pcie_subsys_wrap/u_comphy_wr
ap/u_WRAP_COMPHY_56G_PIPE5_X4_4PLL/HC/PIN_TXCLK_ALIGN_IN_REF]
```

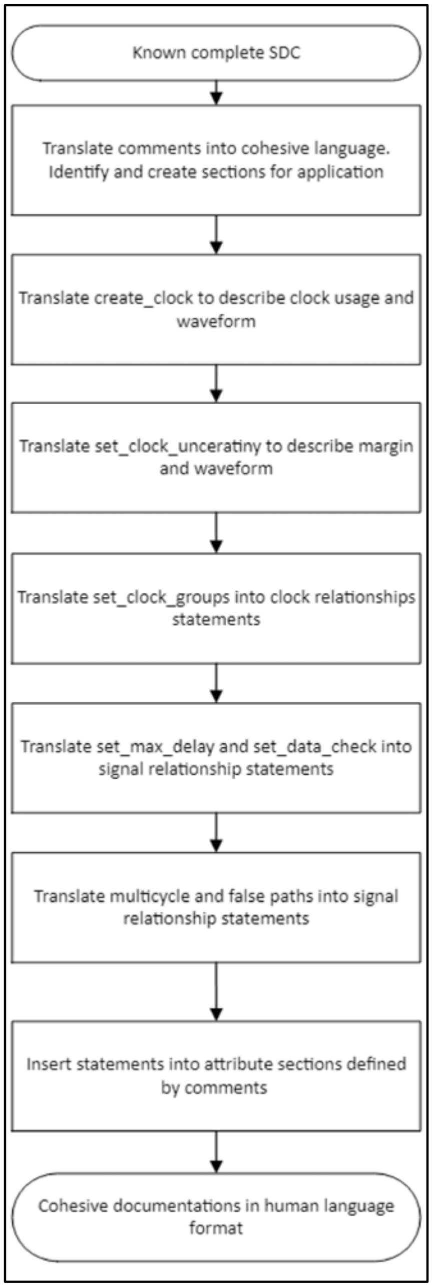
Figure 11: Generate “set_clock_uncertainty” output from Response 3

```
# PIN_CLK100M_125M_SIDE_A
set_clock_uncertainty -setup 30 -fall_from [get_clocks PIN_CLK100M_125M_SIDE_A] -rise_to [get_clocks
PIN_CLK100M_125M_SIDE_A]
set_clock_uncertainty -setup 50 -rise_from [get_clocks PIN_CLK100M_125M_SIDE_A] -fall_to [get_clocks
PIN_CLK100M_125M_SIDE_A]
set_clock_uncertainty -hold 30 -fall_from [get_clocks PIN_CLK100M_125M_SIDE_A] -rise_to [get_clocks
PIN_CLK100M_125M_SIDE_A]
set_clock_uncertainty -hold 50 -rise_from [get_clocks PIN_CLK100M_125M_SIDE_A] -fall_to [get_clocks
PIN_CLK100M_125M_SIDE_A]
# PIN_CLK100M_125M_SIDE_B
set_clock_uncertainty -setup 30 -fall_from [get_clocks PIN_CLK100M_125M_SIDE_B] -rise_to [get_clocks
PIN_CLK100M_125M_SIDE_B]
set_clock_uncertainty -setup 50 -rise_from [get_clocks PIN_CLK100M_125M_SIDE_B] -fall_to [get_clocks
PIN_CLK100M_125M_SIDE_B]
set_clock_uncertainty -hold 30 -fall_from [get_clocks PIN_CLK100M_125M_SIDE_B] -rise_to [get_clocks
PIN_CLK100M_125M_SIDE_B]
set_clock_uncertainty -hold 50 -rise_from [get_clocks PIN_CLK100M_125M_SIDE_B] -fall_to [get_clocks
PIN_CLK100M_125M_SIDE_B]
# AUX_CLK
set_clock_uncertainty -setup 30 -fall_from [get_clocks AUX_CLK] -rise_to [get_clocks AUX_CLK]
set_clock_uncertainty -setup 50 -rise_from [get_clocks AUX_CLK] -fall_to [get_clocks AUX_CLK]
set_clock_uncertainty -hold 30 -fall_from [get_clocks AUX_CLK] -rise_to [get_clocks AUX_CLK]
set_clock_uncertainty -hold 50 -rise_from [get_clocks AUX_CLK] -fall_to [get_clocks AUX_CLK]
# PIPE_TXCLK_OUT
set_clock_uncertainty -setup 30 -fall_from [get_clocks PIPE_TXCLK_OUT] -rise_to [get_clocks PIPE_TXCLK_OUT]
set_clock_uncertainty -setup 50 -rise_from [get_clocks PIPE_TXCLK_OUT] -fall_to [get_clocks PIPE_TXCLK_OUT]
set_clock_uncertainty -hold 30 -fall_from [get_clocks PIPE_TXCLK_OUT] -rise_to [get_clocks PIPE_TXCLK_OUT]
set_clock_uncertainty -hold 50 -rise_from [get_clocks PIPE_TXCLK_OUT] -fall_to [get_clocks PIPE_TXCLK_OUT]
# PIPE_SCLK_OUT
set_clock_uncertainty -setup 30 -fall_from [get_clocks PIPE_SCLK_OUT] -rise_to [get_clocks PIPE_SCLK_OUT]
set_clock_uncertainty -setup 50 -rise_from [get_clocks PIPE_SCLK_OUT] -fall_to [get_clocks PIPE_SCLK_OUT]
set_clock_uncertainty -hold 30 -fall_from [get_clocks PIPE_SCLK_OUT] -rise_to [get_clocks PIPE_SCLK_OUT]
set_clock_uncertainty -hold 50 -rise_from [get_clocks PIPE_SCLK_OUT] -fall_to [get_clocks PIPE_SCLK_OUT]
# PIN_TXCLK_ALIGN_IN_REF
set_clock_uncertainty -setup 30 -fall_from [get_clocks PIN_TXCLK_ALIGN_IN_REF] -rise_to [get_clocks
PIN_TXCLK_ALIGN_IN_REF]
set_clock_uncertainty -setup 50 -rise_from [get_clocks PIN_TXCLK_ALIGN_IN_REF] -fall_to [get_clocks
PIN_TXCLK_ALIGN_IN_REF]
set_clock_uncertainty -hold 30 -fall_from [get_clocks PIN_TXCLK_ALIGN_IN_REF] -rise_to [get_clocks
PIN_TXCLK_ALIGN_IN_REF]
set_clock_uncertainty -hold 50 -rise_from [get_clocks PIN_TXCLK_ALIGN_IN_REF] -fall_to [get_clocks
PIN_TXCLK_ALIGN_IN_REF]
```

The set_clock_group commands and set_multicycle_path (referred to at the bottom of Figure 1) are generated with an analogous approach. The Clock Config and Design Config files' vector databases and RAG retrievers together with prompts give SDC Creator sufficient information to generate the SDC commands.

Figure 12 illustrates the method of how the generated SDC will use Generative AI to translate complete SDC Tcl into human language for design review for accuracy. The user can review the AI output to verify the quality of the SDC, and therefore, verify that the initial set of documentations were accurate and sufficient to provide a complete set of SDC and to have an accurate timing environment. This can be used in new user training, design reviews, and further clarify existing IP documentation.

Figure 12: Flow chart of the method of converting AI-generated SDC to human language description to be compared against the existing documentation for design review


Known complete SDC
Translate comments into cohesive language. Identify and create sections for application
Translate create_clock to describe clock usage and waveform
Translate set_clock_unceratiny to describe margin and waveform
Translate set_clock_groups into clock relationships statements
Translate set_max_delay and set_data_check into signal relationship statements
Translate multicycle and false paths into signal relationship statements
Insert statements into attribute sections defined by comments
Cohesive documentations in human language format

© 2024 Marvell. All rights reserved

In Figure 13, the Generative AI was able to extract all known information about the clock named CM0_CLK_RATE_1_O in an organized format from a well commented, thorough SDC, with references to both the SDC and the databook. This can be used for the SDC constraint accuracy review.

Figure 13: Design Review Using Trained LLM with Comprehensive SDC Tcl with Comments

Prompt:

From this Tcl SDC, provide information about CM0_CLK_RATE1_O

Answer:

Here is some information about CM0_CLK_RATE1_O:

- **Definition**: CM0_CLK_RATE1_O is an output clock pin of the PHY that corresponds to the internal pin AFE_CM0_CLK_PLL. It is one of the important PHY output clock domains where data may be timed synchronously.
- **Frequency**: The frequency of CM0_CLK_RATE1_O is determined by the parameter freq_rate1, which is set according to the design specification and the latest databook. For example, if the data rate is 16 Gbps and the data width is 32 bits, then freq_rate1 = 500 MHz.
- **Waveform**: The waveform of CM0_CLK_RATE1_O is defined by its period and duty cycle. The period is the inverse of the frequency, and the duty cycle is the fraction of the period the clock signal is high. The minimum and maximum duty cycle values are 0.40 and 0.60, respectively, as specified in the databook.
- **Uncertainty**: The uncertainty of CM0_CLK_RATE1_O is the amount of variation in the arrival time of the clock signal due to various factors such as jitter, setup and hold margin, and duty cycle distortion. The setup and hold uncertainties are calculated by adding the margin, the clock jitter, and the duty cycle uncertainty. The clock jitter is 0.065 ns, and the duty cycle uncertainty is half of the difference between the maximum and minimum duty cycle values.

Summary: This paper presents a method for automatically generating chip-specific, configuration-specific SDC timing constraints for interface IP in VLSI chip design using Generative AI. By translating human language documentation and configuration files into precise SDC code, this technique ensures accuracy. It employs Retrieval Augmented Generation (RAG) to retrieve relevant data from a vector database, producing contextually accurate SDC code. The importance of timing constraints for interface IP is highlighted by the increasing use of SerDes, die-to-die IP, and multi-die packages (2.5D, 3D), which significantly increase the required amount of SDC Tcl code.

References:

1). https://www.marvell.com/company/newsroom/marvell-first-112g-5nm-serdes-solution-scaling-cloud-data-center-infrastructure.html
2). Constraining Designs for Synthesis and Timing Analysis, A Practical Guide to Synopsys Design Constraints (SDC), Gangadharan, Sridhar and Churiwala, Sanjay, copyright 2013, Springer.
3). Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey, siyunzhao,yuqing.yang,wangzilong,zhiyuan.he,lunaqiu,liliqiu}@microsoft.com, 2409.14924v1
4). Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT, and other LLMs, Auffarth, Ben, copyright 2023.
5). Large Enough | Mistral AI | Frontier AI in your hands

© 2024 Marvell. All rights reserved