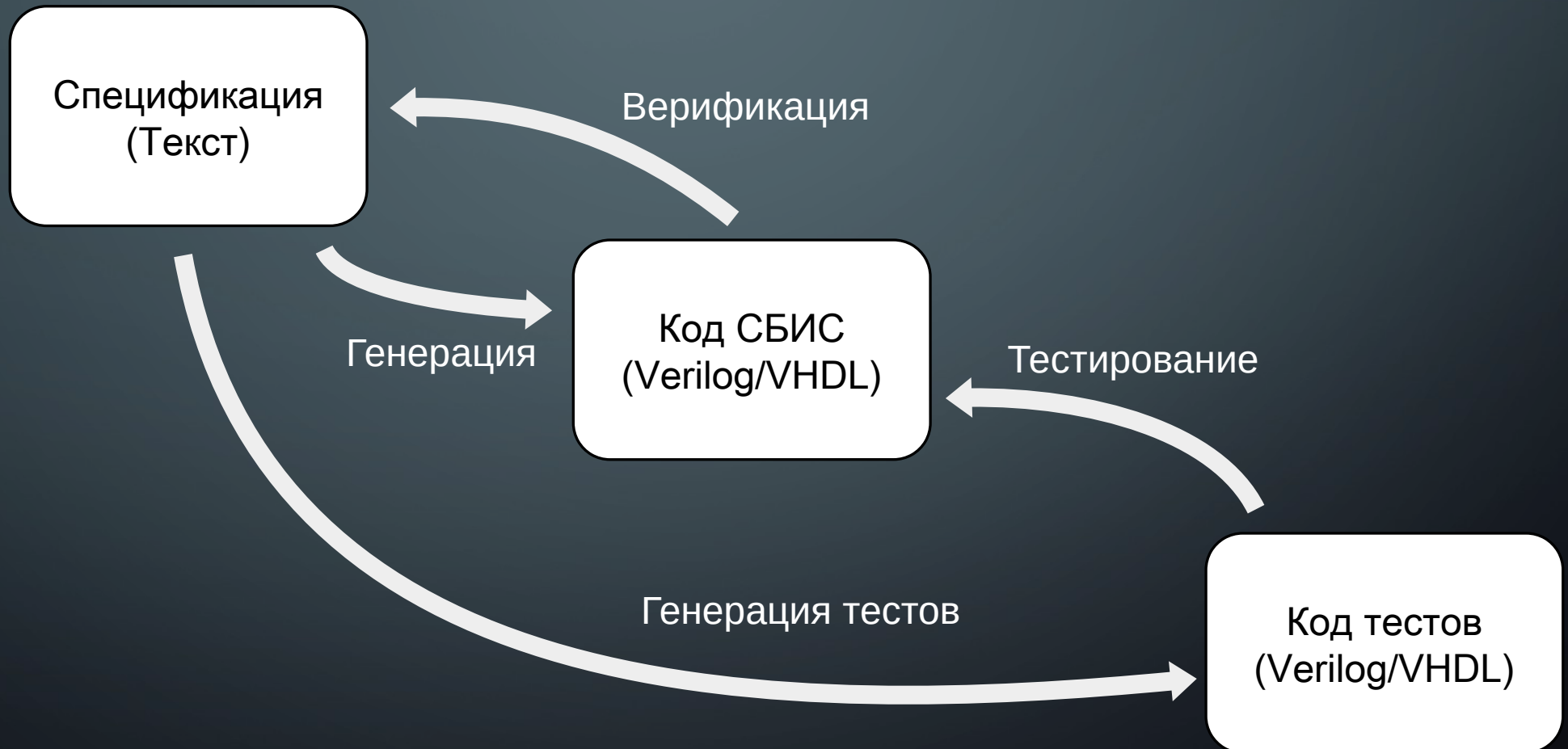




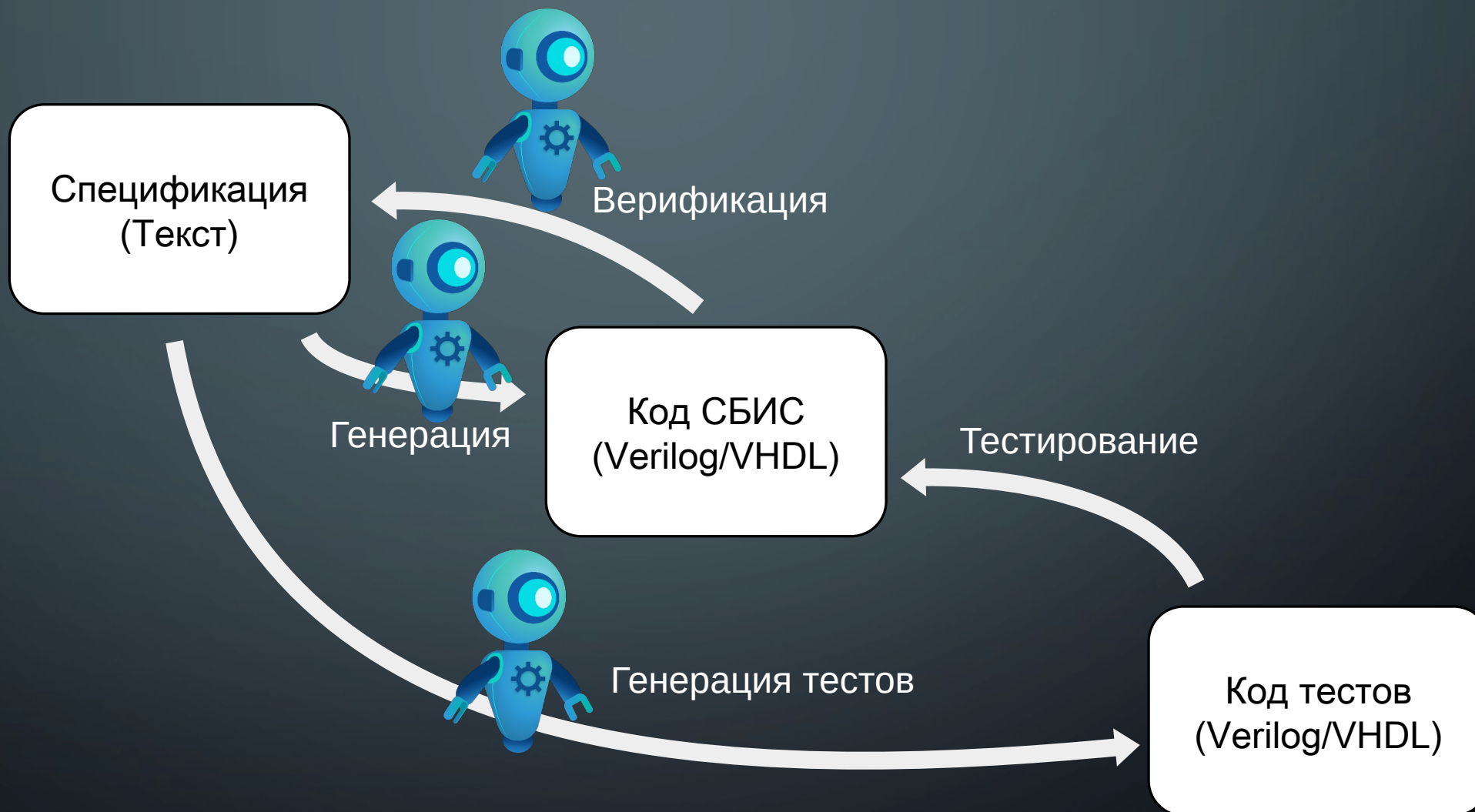
БЯМ для СБИС

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ
С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ



ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)



ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Please act as a professional verilog designer.

Implement a data width conversion circuit that converts 8-bit data input to 16-bit data output. The module provides two output ports: `valid_out`, which indicates the validity of the output data, and `data_out`, which represents the converted 16-bit output data. The first arriving 8-bit data should be placed in the higher 8 bits of the 16-bit data output. The `valid_out` and `data_out` signals are generated in the next clock cycle after the two data inputs. When there is only one data input, `valid_out` and `data_out` are not generated immediately. Instead, they wait for the arrival of the next data input to complete the concatenation of the two data inputs before generating `valid_out` and `data_out`.

Module name:
`width_8to16`

Input ports:

`clk`: Clock signal used for synchronous operation.
`rst_n`: Active-low reset signal. Defined as 0 for reset and 1 for reset signal inactive.
`valid_in`: Input signal indicating the validity of the input data.
`data_in`: 8-bit input data to be converted.

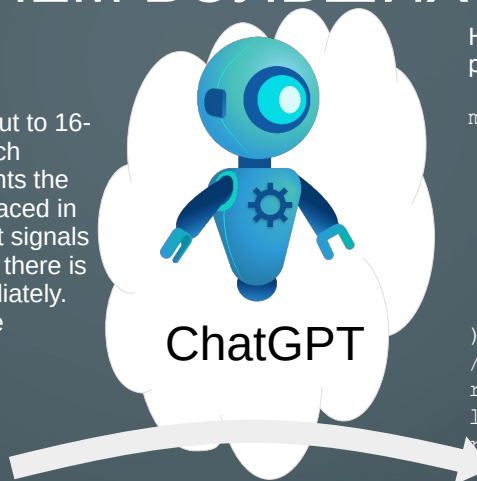
Output ports:

`valid_out`: Output signal indicating the validity of the output data.
`data_out`: 16-bit output data resulting from the width conversion.

Implementation:

The `data_out` register is triggered on the positive edge of the clock signal (`posedge clk`) or the negative edge of the reset signal (`negedge rst_n`). On reset, indicated by `!rst_n`, the `data_out` register is cleared to 0. If the input data is valid (`valid_in`) and the flag signal is active, the `data_out` register is updated by concatenating the contents of the `data_lock` register (8 bits) and the `data_in` register (8 bits) to form a 16-bit output. The first valid data is temporarily stored, and when the second valid data is inputted, they are concatenated to produce the output `valid_out` and `data_out`.

Give me the complete code.



Генерация

Here is a complete and professional Verilog implementation of the `width_8to16` module as per your requirements:

```
module width_8to16 (
    input      clk,          // Clock signal
    input      rst_n,        // Active-low reset
    input      valid_in,     // Indicates when data_in is valid
    input [7:0] data_in,     // 8-bit input data

    output reg  valid_out,   // Indicates when data_out is valid
    output reg [15:0] data_out // 16-bit output data
);
// Internal registers
reg  data_lock_valid; // Flag indicating a byte has been
locked
reg [7:0] data_lock; // Temporarily holds the first byte

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        data_out      <= 16'b0;
        valid_out     <= 1'b0;
        data_lock_valid <= 1'b0;
        data_lock      <= 8'b0;
    end else begin
        valid_out <= 1'b0; // Default to invalid each cycle unless
new output is formed

        if (valid_in) begin
            if (data_lock_valid) begin
                // Second byte has arrived; form 16-bit word
                data_out      <= {data_lock, data_in}; // High
byte first
                valid_out     <= 1'b1;
                data_lock_valid <= 1'b0; // Clear lock
            end else begin
                // Store the first 8-bit input
                data_lock      <= data_in;
                data_lock_valid <= 1'b1;
            end
        end
    end
end
endmodule
```


ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Please act as a professional verilog designer. Validate the following verilog code:

```
module width_8to16 (
    input      clk,          // Clock signal
    input      rst_n,        // Active-low reset
    input      valid_in,     // Indicates when data_in is valid
    input [7:0] data_in,     // 8-bit input data

    output reg      valid_out, // Indicates when data_out is valid
    output reg [15:0] data_out // 16-bit output data
);
// Internal registers
reg      data_lock_valid; // Flag indicating a byte has been locked
reg [7:0] data_lock;      // Temporarily holds the first byte

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        data_out      <= 16'b0;
        valid_out      <= 1'b0;
        data_lock_valid <= 1'b0;
        data_lock      <= 8'b0;
    end else begin
        valid_out <= 1'b0; // Default to invalid each cycle
        unless new output is formed

            if (valid_in) begin
                if (data_lock_valid) begin
                    // Second byte has arrived; form 16-bit word
                    data_out      <= {data_lock, data_in}; // High
byte first
                    valid_out      <= 1'b1;
                    data_lock_valid <= 1'b0; // Clear lock
                end else begin
                    // Store the first 8-bit input
                    data_lock      <= data_in;
                    data_lock_valid <= 1'b1;
                end
            end
        end
    end
end
endmodule
```



Валидация

As a professional Verilog designer, I'll validate the provided Verilog code for the width_8to16 module, which converts an 8-bit input data stream to a 16-bit output data stream by concatenating two consecutive valid 8-bit inputs. Below is a detailed analysis of the code, including its functionality, correctness, potential issues, and suggestions for improvement.

Code Overview

The module takes an 8-bit input data (data_in) with a validity signal (valid_in) and produces a 16-bit output data (data_out) with a corresponding validity signal (valid_out). It uses a clock (clk) and an active-low reset (rst_n). The module stores the first valid 8-bit input in a register (data_lock) and waits for the second valid input to form a 16-bit output by concatenating the stored byte with the new byte.

Validation Checklist

- Syntax Correctness
- Functional Correctness

Reset Behavior

- Timing and Synchronization

- Edge Cases and Robustness

- Coding Style and Best Practices

1. Syntax Correctness

Status: The Verilog code is syntactically correct.

Details:

All module ports, internal registers, and signals are properly declared.

The always block uses the correct sensitivity list for a synchronous design (posedge clk or negedge rst_n).

No syntax errors (e.g., missing semicolons, incorrect keywords) are present.

The code adheres to Verilog-2001 or later standards (e.g., port declaration style).

2. Functional Correctness

Intended Functionality: The module combines two consecutive 8-bit inputs into a single 16-bit output when valid_in is high, with valid_out indicating when data_out is valid.

Analysis:

Reset State: On active-low reset (rst_n = 0), all registers (data_out, valid_out, data_lock, data_lock_valid) are cleared to 0, which is correct for initialization.

First Byte: When valid_in is high and data_lock_valid is low (no byte stored), the input data_in is stored in data_lock, and data_lock_valid is set to 1. No output is produced (valid_out remains 0), which is correct.

Second Byte: When valid_in is high and data_lock_valid is high (a byte is stored), the module concatenates data_lock (high byte) with data_in (low byte) to form data_out, sets valid_out to 1, and clears data_lock_valid. This matches the intended functionality.

Default Behavior: valid_out is set to 0 at the start of each clock cycle unless a new 16-bit output is formed, ensuring valid_out is only high for one cycle when a valid 16-bit output is produced.

Correctness: The logic correctly implements the 8-bit to 16-bit conversion by pairing consecutive valid inputs.

...

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Can you validate the code above against the following specification?

Implement a data width conversion circuit that converts 8-bit data input to 16-bit data output. The module provides two output ports: `valid_out`, which indicates the validity of the output data, and `data_out`, which represents the converted 16-bit output data. The first arriving 8-bit data should be placed in the higher 8 bits of the 16-bit data output. The `valid_out` and `data_out` signals are generated in the next clock cycle after the two data inputs. When there is only one data input, `valid_out` and `data_out` are not generated immediately. Instead, they wait for the arrival of the next data input to complete the concatenation of the two data inputs before generating `valid_out` and `data_out`.

Module name:
`width_8to16`

Input ports:

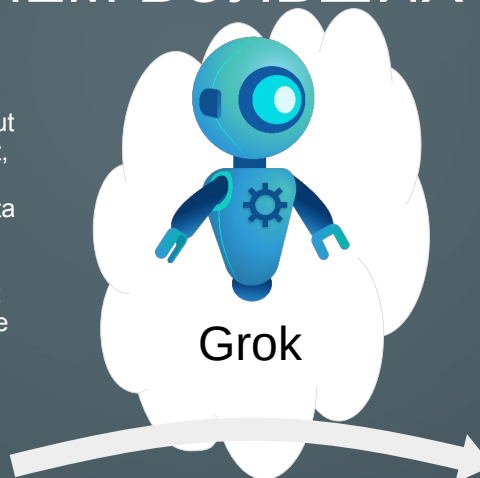
`clk`: Clock signal used for synchronous operation.
`rst_n`: Active-low reset signal. Defined as 0 for reset and 1 for reset signal inactive.
`valid_in`: Input signal indicating the validity of the input data.
`data_in`: 8-bit input data to be converted.

Output ports:

`valid_out`: Output signal indicating the validity of the output data.
`data_out`: 16-bit output data resulting from the width conversion.

Implementation:

The `data_out` register is triggered on the positive edge of the clock signal (`posedge clk`) or the negative edge of the reset signal (`negedge rst_n`). On reset, indicated by `!rst_n`, the `data_out` register is cleared to 0. If the input data is valid (`valid_in`) and the flag signal is active, the `data_out` register is updated by concatenating the contents of the `data_lock` register (8 bits) and the `data_in` register (8 bits) to form a 16-bit output. The first valid data is temporarily stored, and when the second valid data is inputted, they are concatenated to produce the output `valid_out` and `data_out`.



Валидация

1. Module Name and Ports

Specification: Module named `width_8to16` with inputs (`clk`, `rst_n`, `valid_in`, `data_in[7:0]`) and outputs (`valid_out`, `data_out[15:0]`).

Code:

Module name: Correctly defined as `width_8to16`.

Inputs: `clk`, `rst_n`, `valid_in`, `data_in[7:0]` are all correctly declared.

Outputs: `valid_out` and `data_out[15:0]` are correctly declared as `reg` types.

Status: Compliant.

2. Reset Behavior

Specification: On reset (`!rst_n`), `data_out` is cleared to 0. (Implies other registers like `valid_out`, `data_lock`, and `data_lock_valid` should also be initialized appropriately.)

Code:

Reset condition: if (`!rst_n`) in the always block.

Resets:

`data_out <= 16'b0`: Clears output data (matches specification).

`valid_out <= 1'b0`: Ensures no spurious output (reasonable, not explicitly mentioned but implied).

`data_lock <= 8'b0`: Clears stored byte (reasonable).

`data_lock_valid <= 1'b0`: Clears flag (reasonable).

Reset is synchronous (checked on `posedge clk`), despite the specification mentioning `negedge rst_n`, which suggests an asynchronous reset.

Status: Partially Compliant. The code uses a synchronous reset, while the specification implies an asynchronous reset due to the `negedge rst_n` in the sensitivity list. This is a potential mismatch that needs clarification or correction.

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Please act as a professional verilog designer.

Implement a data width conversion circuit that converts 8-bit data input to 16-bit data output. The module provides two output ports: `valid_out`, which indicates the validity of the output data, and `data_out`, which represents the converted 16-bit output data. The first arriving 8-bit data should be placed in the higher 8 bits of the 16-bit data output. The `valid_out` and `data_out` signals are generated in the next clock cycle after the two data inputs. When there is only one data input, `valid_out` and `data_out` are not generated immediately. Instead, they wait for the arrival of the next data input to complete the concatenation of the two data inputs before generating `valid_out` and `data_out`.

Module name:
`width_8to16`

Input ports:

`clk`: Clock signal used for synchronous operation.
`rst_n`: Active-low reset signal. Defined as 0 for reset and 1 for reset signal inactive.
`valid_in`: Input signal indicating the validity of the input data.
`data_in`: 8-bit input data to be converted.

Output ports:

`valid_out`: Output signal indicating the validity of the output data.
`data_out`: 16-bit output data resulting from the width conversion.

Implementation:

The `data_out` register is triggered on the positive edge of the clock signal (`posedge clk`) or the negative edge of the reset signal (`negedge rst_n`). On reset, indicated by `!rst_n`, the `data_out` register is cleared to 0. If the input data is valid (`valid_in`) and the flag signal is active, the `data_out` register is updated by concatenating the contents of the `data_lock` register (8 bits) and the `data_in` register (8 bits) to form a 16-bit output. The first valid data is temporarily stored, and when the second valid data is inputted, they are concatenated to produce the output `valid_out` and `data_out`.

Give me the complete code.



Генерация

Here's the implementation of the `width_8to16` module in Verilog:

```
// Width Conversion Module (8-bit to 16-bit)
module width_8to16(
    input clk,
    input rst_n,
    input valid_in1, // First valid data signal
    input [7:0] data_in1, // First 8-bit input data
    input valid_in2, // Second valid data signal
    output reg [15:0] data_out, // Converted 16-bit output data
    output reg valid_out // Validity of the output data
);

reg [7:0] data_lock; // Register to store first valid 8-bit data

always @(posedge clk) begin
    if (~rst_n) begin
        data_out <= 0;
        valid_out <= 1'b0;
    end else if (valid_in1 && (!valid_in2)) begin
        data_out <= data_lock;
        valid_out <= 1'b1;
    } else if (valid_in1 && valid_in2) begin
        data_lock[7:0] = data_in1;
        data_lock[15:8] = data_in2;
        data_out = {data_lock, data_in2};
        valid_out <= 1'b1;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (rst_n) begin
        data_lock <= 0;
    end
end

endmodule
```


ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Please act as a professional verilog designer. Validate the following verilog code:

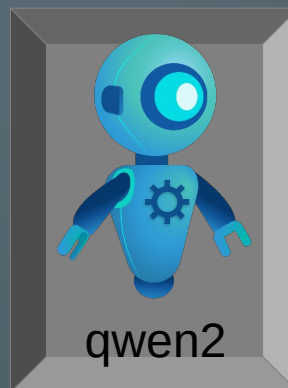
```
// Width Conversion Module (8-bit to 16-bit)
module width_8to16(
    input clk,
    input rst_n,
    input valid_in1, // First valid data signal
    input [7:0] data_in1, // First 8-bit input data
    input valid_in2, // Second valid data signal
    output reg [15:0] data_out, // Converted 16-bit
    output data
    output reg valid_out // Validity of the output data
);

reg [7:0] data_lock; // Register to store first valid
bit data

always @(posedge clk) begin
    if (~rst_n) begin
        data_out <= 0;
        valid_out <= 1'b0;
    end else if (valid_in1 && (!valid_in2)) begin
        data_out <= data_lock;
        valid_out <= 1'b1;
    } else if (valid_in1 && valid_in2) begin
        data_lock[7:0] = data_in1;
        data_lock[15:8] = data_in2;
        data_out = {data_lock, data_in2};
        valid_out <= 1'b1;
    end
end

always @(posedge clk or negedge rst_n) begin
    if (rst_n) begin
        data_lock <= 0;
    end
end

endmodule
```



Валидация

As a professional Verilog designer, let's validate the provided code snippet for the width conversion module from 8-bit to 16-bit.

The given Verilog code is designed to convert an incoming stream of 8-bit data (either one or two at a time) into a single 16-bit output. The key components and their functions are:

- Input signals: clk for clock input, rst_n for asynchronous reset signal, valid_in1 and valid_in2 for the validity of incoming 8-bit data, data_in1 for the first 8-bit input data stream, and data_out as the output that holds the converted 16-bit data.
- Internal logic: An always block handles clock events (posedge clk) to update the internal state based on valid inputs.
 - When only one of valid_in1 or valid_in2 is active (i.e., both are not active simultaneously), the output holds a single piece of data from either data_in1 or data_in2.
 - If both signals valid_in1 and valid_in2 are active, they combine an incoming 8-bit value with the previous 8-bit input to form a 16-bit output.
- Reset handling: The reset signal (rst_n) ensures that all internal states (data_lock, data_out, valid_out) are initialized correctly after the reset event.

Verification Steps:

To verify this design, consider the following points and test scenarios:

Test Case 1: Single Input Stream

- Initialize system with no active inputs.
- Apply a single valid input through data_in1 (valid_in1 is true) followed by clock edge. Verify that data_out correctly captures the value of data_in1, and valid_out becomes true.

Test Case 2: Dual Input Streams

- Repeat with both inputs active (valid_in1 and valid_in2) at the same time.
- Provide two valid streams (data_in1 and data_in2).
- Ensure that a new output is formed by combining the last value from data_in1, followed by the current 8-bit input from data_in2.
- Check that valid_out toggles between false and true accordingly.

Test Case 3: Resetting

- Apply the reset signal (rst_n) while the system has valid inputs.
- Verify that all outputs are cleared to their initial state (e.g., data_out=0, valid_out=0) upon rising edge of the clock.

Implementation Verification:

1. Simulate the circuit: Utilize a tool like ModelSim or Vivado's Simulation environment for hardware description language verification.
2. Testbench creation: Write a testbench that generates various input patterns and observes corresponding output behavior based on defined scenarios above.
3. Expected vs Actual Output Comparison: Compare actual simulation results with expected behavior according to the logic described in the code.

Conclusion:

By systematically testing these cases, you can ensure that the width conversion module functions correctly under different operating conditions, validating its design implementation for both edge cases and regular operation.

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

Can you validate the code above against the following specification?

Implement a data width conversion circuit that converts 8-bit data input to 16-bit data output. The module provides two output ports: `valid_out`, which indicates the validity of the output data, and `data_out`, which represents the converted 16-bit output data. The first arriving 8-bit data should be placed in the higher 8 bits of the 16-bit data output. The `valid_out` and `data_out` signals are generated in the next clock cycle after the two data inputs. When there is only one data input, `valid_out` and `data_out` are not generated immediately. Instead, they wait for the arrival of the next data input to complete the concatenation of the two data inputs before generating `valid_out` and `data_out`.

Module name:
`width_8to16`

Input ports:

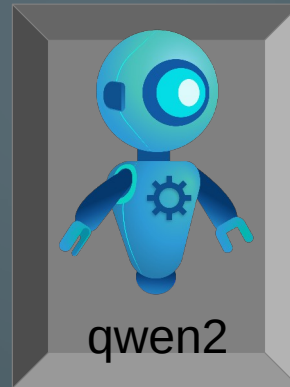
`clk`: Clock signal used for synchronous operation.
`rst_n`: Active-low reset signal. Defined as 0 for reset and 1 for reset signal inactive.
`valid_in`: Input signal indicating the validity of the input data.
`data_in`: 8-bit input data to be converted.

Output ports:

`valid_out`: Output signal indicating the validity of the output data.
`data_out`: 16-bit output data resulting from the width conversion.

Implementation:

The `data_out` register is triggered on the positive edge of the clock signal (`posedge clk`) or the negative edge of the reset signal (`negedge rst_n`). On reset, indicated by `!rst_n`, the `data_out` register is cleared to 0. If the input data is valid (`valid_in`) and the flag signal is active, the `data_out` register is updated by concatenating the contents of the `data_lock` register (8 bits) and the `data_in` register (8 bits) to form a 16-bit output. The first valid data is temporarily stored, and when the second valid data is inputted, they are concatenated to produce the output `valid_out` and `data_out`.



Валидация

The code description above appears to accurately implement the specifications provided for converting an 8-bit data input into a 16-bit output data width conversion circuit. Let's analyze it against the requirements:

Module Name:

- `width_8to16`: The name follows the requirement of indicating that this module performs width conversion from 8 bits to 16 bits.

Input Ports:

- `clk`: Represents the clock signal, which is essential for synchronous operation.
- `rst_n`: Active-low reset signal, allowing the circuit to be reset when needed.
- `valid_in`: Signal indicating if the input data is valid, aligning with the requirement of having a validity flag for each data arrival.
- `data_in`: 8-bit input data that needs conversion.

Output Ports:

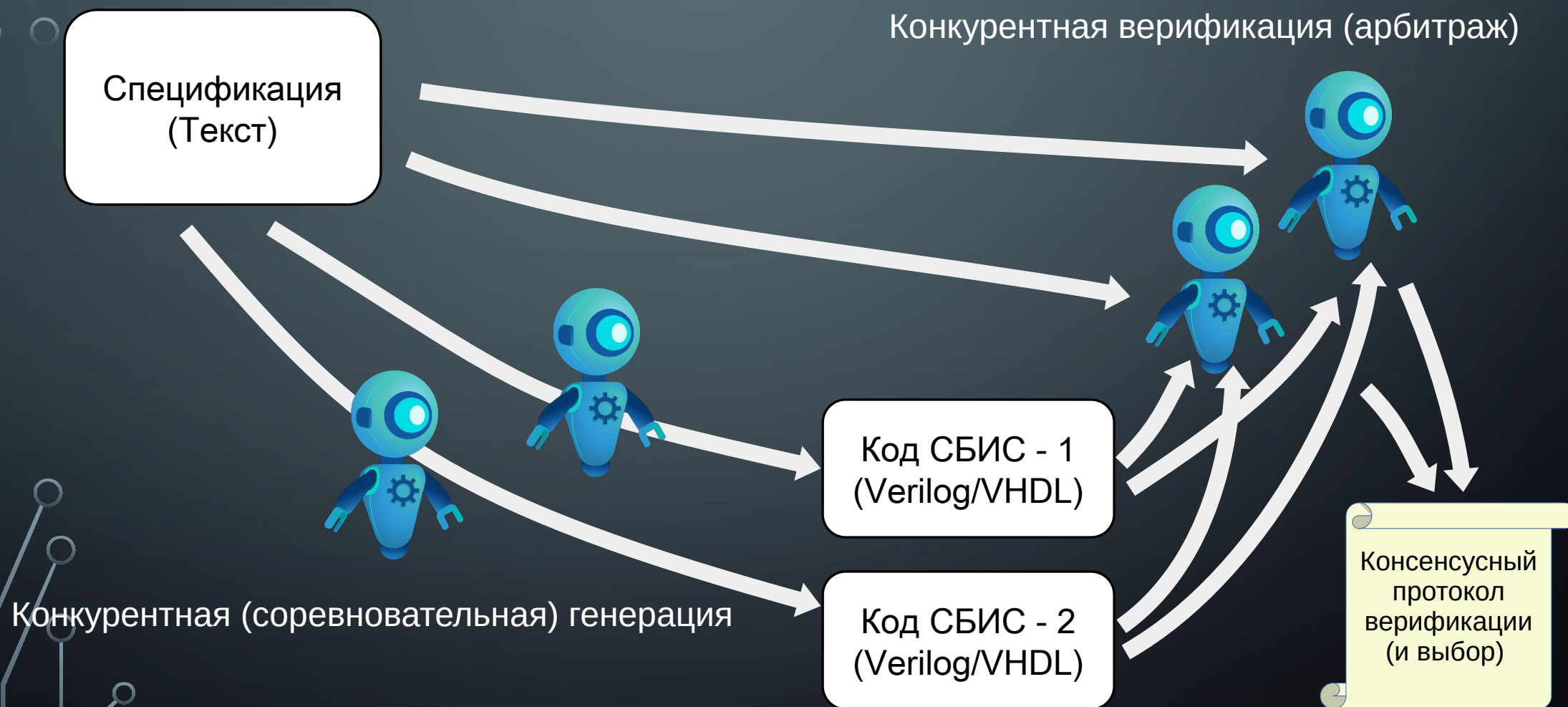
- `valid_out`: A signal indicating whether the output data is valid, which matches the requirement to generate this signal after receiving two inputs and ensuring both are valid.
- `data_out`: The resulting 16-bit data from converting the input. This meets the specification of a 16-bit output width.

...

Conclusion:

Overall, the provided code description effectively addresses all the requirements specified for implementing a data width conversion circuit from 8-bit to 16-bit inputs by ensuring proper synchronization, handling of reset and valid signals, and correct data concatenation logic. The code is described in terms that align well with the given specifications, suggesting it would perform as intended under those conditions.

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM) - ПОВЫШЕНИЕ НАДЕЖНОСТИ АНСАМБЛИРОВАНИЕМ

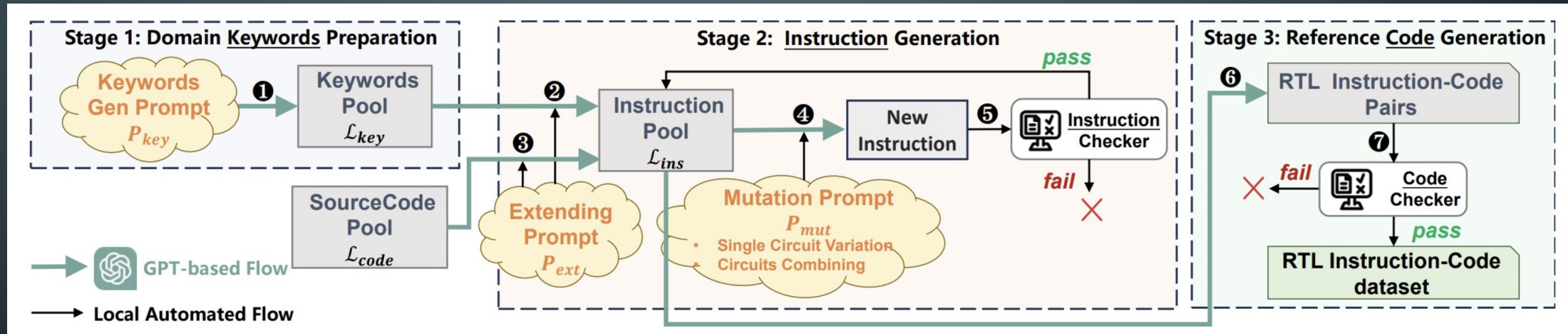


ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM) - НА ЧЕМ ОБУЧАТЬ МОДЕЛИ И СТРОИТЬ ПРОМПТЫ?

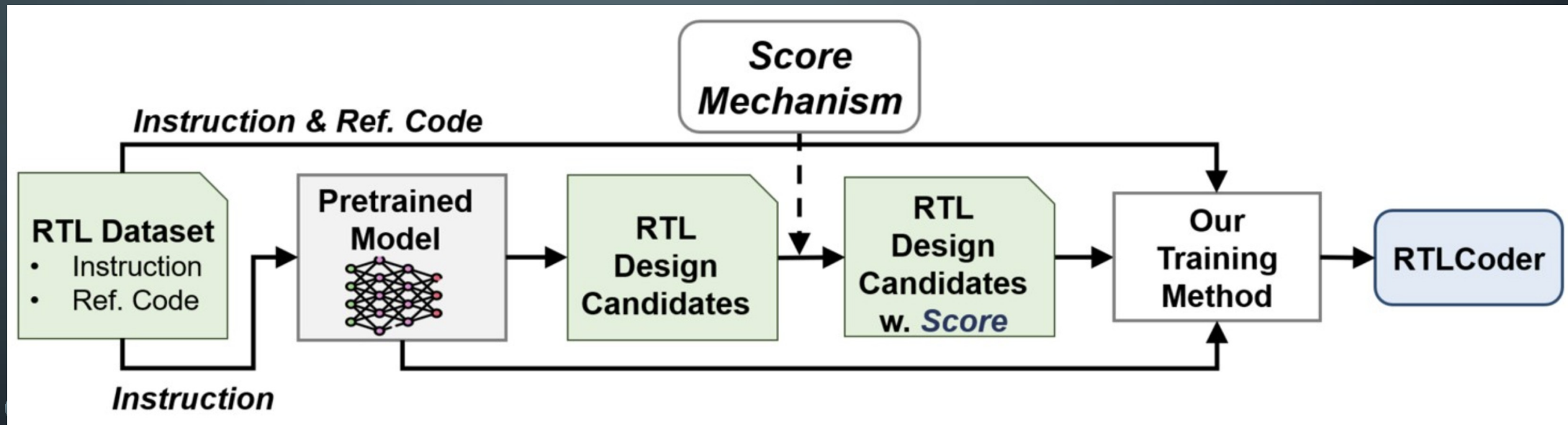
Works	New Training Dataset	New LLM Model	Outperform GPT-3.5
Prompt Engineering [3, 4, 14, 15, 22]	N/A	N/A	N/A
Thakur et al. [21] from NYU	Open-Source	Open-Source	No
VerilogEval [12] & ChipNeMo [11] from NVIDIA	Closed-Source	Closed-Source	Comparable
RTLCoder from HKUST	Open-Source	Open-Source	Yes

TABLE 1: LLM-based works on design RTL generation (e.g., Verilog).

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM) - КАК УПРАВЛЯТЬ ГЕНЕРАЦИЕЙ?



ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM) - КАК ДО-ОБУЧАТЬ МОДЕЛИ?



ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

ВЫВОДЫ:

1. КАК ОБЛАЧНЫЕ, ТАК И ЛОКАЛЬНО УСТАНОВЛИВАЕМЫЕ БЯМ (LLM) «ИЗ КОРОБКИ» МОГУТ КАК ГЕНЕРИРОВАТЬ КОД VERILOG, ТАК И ВЕРИФИЦИРОВАТЬ ЕГО
2. ВОЗМОЖНО БОЛЕЕ ТОНКОЕ УПРАВЛЕНИЕ И ГЕНЕРАЦИЕЙ И ВЕРИФИКАЦИЕЙ, А ТАКЖЕ ОБУЧЕНИЕ СОБСТВЕННЫХ МОДЕЛЕЙ

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

ВОПРОСЫ ДЛЯ РЕШЕНИЯ (РИСКИ):

1. ВОЗМОЖНОСТЬ МИНИМИЗАЦИИ “ГАЛЛЮЦИНАЦИЙ”
2. ВОЗМОЖНОСТЬ КОМПАКТИЗАЦИИ РЕЗУЛЬТАТОВ
ВЕРИФИКАЦИИ ДЛЯ ОПЕРАТИВНОГО АНАЛИЗА
3. ВОЗМОЖНОСТЬ ФОРМАЛИЗАЦИИ И АВТОМАТИЗАЦИИ
ПРОЦЕССА ОПЕРАТИВНОГО АНАЛИЗА И ПРИНЯТИЯ
ОТВЕТСТВЕННОСТИ ЗА ЕГО РЕЗУЛЬТАТЫ
4. ВОЗМОЖНОСТИ ДО-ОБУЧЕНИЯ НА СОБСТВЕННЫХ ДАННЫХ
(КОДА И СПЕЦИФИКАЦИЙ)

ПРОЕКТИРОВАНИЕ СВЕРХБОЛЬШИХ ИНТЕГРАЛЬНЫХ СХЕМ С ИСПОЛЬЗОВАНИЕМ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (LLM)

КОНТАКТЫ:

Антон Колонин, к.т.н.,
в.н.с. ЦИИ НГУ
anton.kolonin@yandex.ru
a.kolonin@g.nsu.ru
+7-913-925-00-58
@akolonin