

RTL Design: A Comprehensive Guide to Understanding and Implementing Register-Transfer Level Design



Umar Waseem
Last updated on 12 Jan, 2024

 FOLLOW



Verilog, a Hardware Description Language (HDL) used to model electronic systems.

This article aims to provide a comprehensive guide to RTL design, including the fundamentals, operations, and advanced concepts while addressing the challenges and nuances that make this design approach valuable.

Digital Fabrication - Electronics - Microchip - Printed Electronics - Transistors

Introduction

Register-Transfer Level (RTL) design is a critical aspect of digital circuit design. It is a design abstraction representing how data flows between hardware registers and the operations performed on that data. RTL design approach is used in creating digital systems where the flow of data between registers, and the operations on that data, are important for the system's functionality.

As digital circuits become more complex, and demand for more efficient systems increases, RTL design has become an essential skill for design engineers alike. It provides a level of abstraction that enables efficient design and verification of complex digital systems. The importance of RTL design is further underscored by its role in the design of microprocessors, digital signal processors, and other complex integrated circuits.

RTL design provides an abstraction that makes it possible to design and verify systems that contain millions of transistors. It provides a manageable way to design complex digital

systems and is a critical step in transforming a high-level system description into a fabricated integrated circuit.

The Basics of RTL Design

Register-Transfer Level (RTL) design is a methodology used for designing digital circuits. The term "register-transfer" refers to the level of abstraction at which the design is considered. [1] At this level, the design is described in terms of the flow of data between registers, and the logic operations that take place on that data.

In RTL design, the designer specifies the data operations to be performed, and the flow of data between registers. The designer does not need to specify how these operations are to be implemented in the hardware. This level of abstraction allows the designer to focus on the high-level function of the system, without getting stuck in the details of the implementation.

RTL design is typically done using a hardware description language (HDL) such as VHDL or Verilog. These languages provide constructs that allow the designer to describe the system in terms of registers, operations, and data flows. The use of an HDL allows the designer to describe the system in a way that is independent of the specific hardware that will be used to implement the system.

Recommended Reading: Guide to Mastering SystemVerilog: Elevate Your Hardware Design and Verification Skills

Understanding Registers

In RTL design, a register is a hardware element that can store a fixed amount of data. The data stored in a register can be used as input to a logic operation, and the result of the operation can be stored back in the register. **Registers** are the basic building blocks of digital systems, and the flow of data between registers is a key aspect of the system's function. [2]

Registers in RTL design are typically binary, meaning they store data in the form of binary digits, or bits. The number of bits a register can store is referred to as the **width** of the register. **For example**, a register that can store 8 bits of data is called an 8-bit register.

The data stored in a register is often referred to as the **state** of the register. The state of a register can change over time as new data is stored in the register. The sequence of states that a register goes through over time is a key aspect of the behavior of the digital system.

In RTL design, registers are used to hold the inputs, outputs, and intermediate results of the logic operations performed. The designer specifies the sequence of operations to be performed and the flow of data between registers to implement the desired function of the system.

Understanding Operations

In RTL design, operations are the logic functions that manipulate and process the data stored in registers. These operations are the fundamental building blocks of digital systems and are responsible for implementing the desired functionality of the system. Operations in RTL design can be broadly categorised into **two types**: arithmetic operations and logical operations.

Arithmetic operations are used to perform mathematical calculations on the data stored in registers. Some common arithmetic operations include addition, subtraction, multiplication, and division. These operations are typically implemented using dedicated hardware elements, such as adders, subtractors, multipliers, and dividers. In RTL design, arithmetic operations are specified using the appropriate constructs in the hardware description language (HDL), such as VHDL or Verilog.

For example, in VHDL, an addition operation can be specified as follows:

```
sum <= a + b;
```

In this example, a and b are input registers, and sum is the output register that stores the result of the addition operation.

Logical operations are used to perform bitwise manipulation of the data stored in registers. Some common logical operations include AND, OR, NOT, XOR, and shift operations. These operations are implemented using basic logic gates, such as AND gates, OR gates, and inverters. In RTL design, logical operations are specified using the appropriate constructs in the HDL.

For example, in Verilog, an AND operation can be specified as follows:

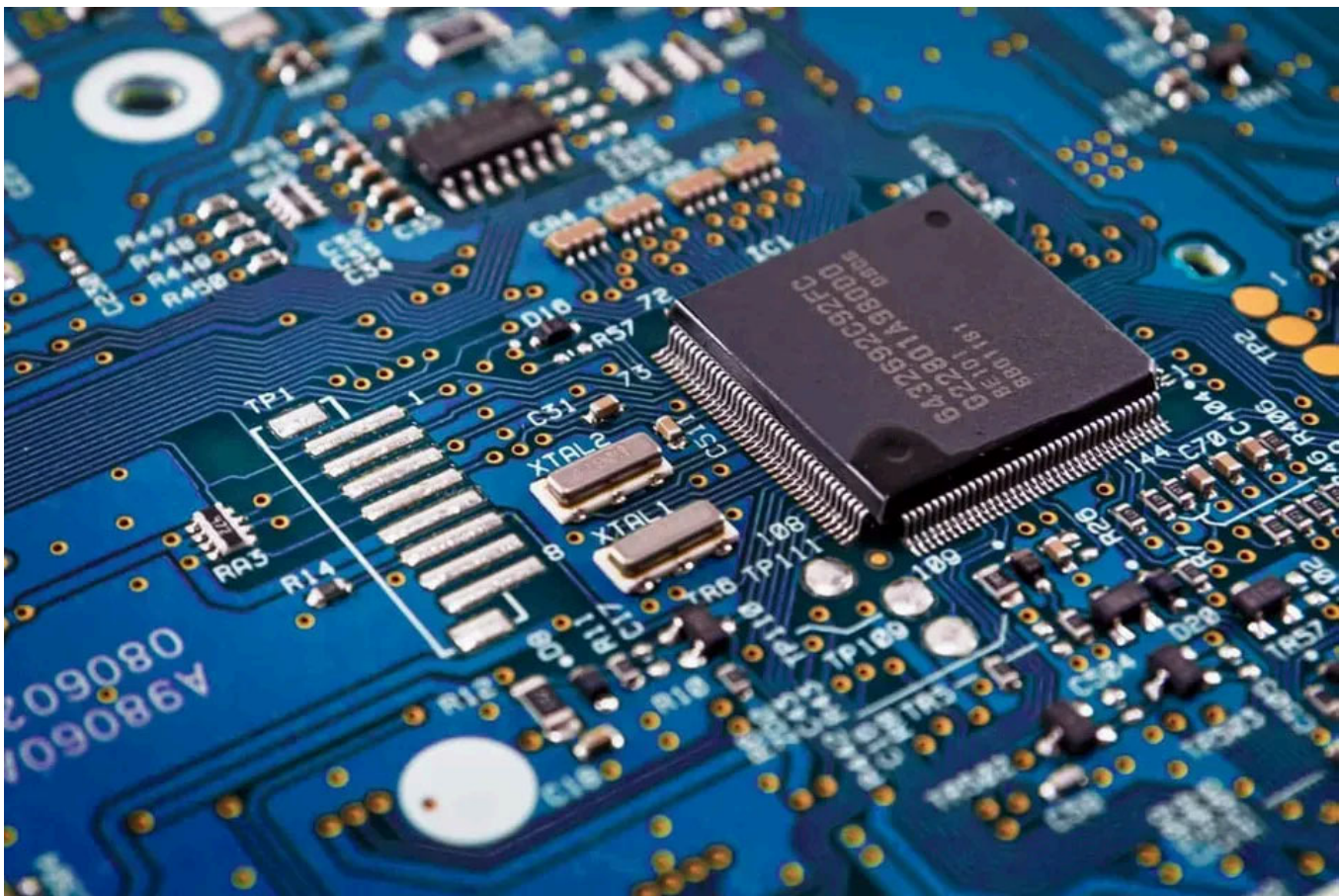
```
assign result = a & b;
```

In this example, a and b are input registers, and result is the output register that stores the result of the AND operation.

In RTL design, operations are used to implement the desired functionality of the system by processing the data stored in registers. The designer specifies the sequence of operations to be performed and the flow of data between registers to achieve the desired system behavior. The designer can efficiently design and verify complex digital systems by focusing on the operations and data flow at the RTL level.

Synchronous and Asynchronous Logic

At the heart of digital circuit design, two fundamental approaches orchestrate data flow: synchronous and asynchronous logic. [3] Within RTL (Register Transfer Level) design, they each play distinct roles, shaping how systems operate and interact. While synchronous logic relies on a global clock signal to guide operations, asynchronous logic empowers elements to move independently, guided by local signals.



Any circuit, from asynchronous to synchronous, may be designed using the RTL

Any digital circuit, whether combinational or sequential, can be designed using RTL. RTL logic can be implemented in a similar fashion to event-based or clock-based logic. The event-based reset logic in that code, which is independent of other events, allows the circuit to be reset, as shown in the figure below. The TL logic for asynchronous logic is shown in the figure:

```

async_reset process(clk, rst) is
begin
    if rst = '1' then
        q <= '0' ;
    elsif rise_edge(clk) then
        q <= d ;
    end if ;
end process async_reset;

```

Additionally, events in synchronous logic depend on the synchronisation of the clock as depicted below.

```

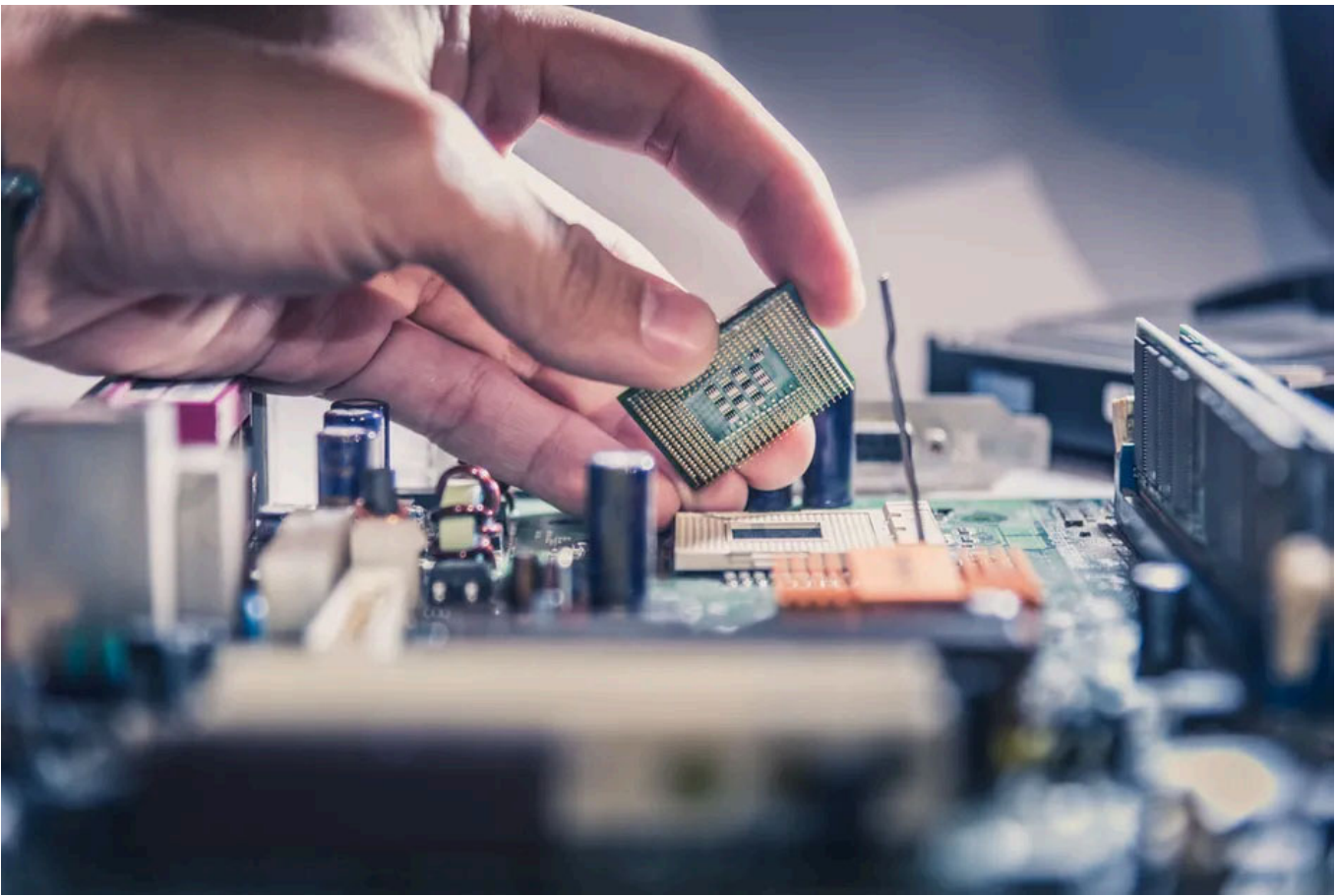
sync_reset process(clk) is
begin
    if rise_edge(clk) then
        if rst = '1' then
            q <= '0' ;
        else
            q <= d ;
        end if ;
    end if;
end process sync_reset;

```

In general, synchronous logic is the preferred choice for most digital systems due to its predictability and ease of design. However, asynchronous logic can offer advantages in specific applications where power consumption or performance is critical.

The RTL Design Process

The RTL design process is a series of steps that transform a high-level system description into a low-level implementation that can be fabricated onto a chip. This process, also known as **front end** involves several key stages, including high-level synthesis, RTL coding, and RTL verification.



Typical computer chip design with connecting pins

The first stage of the RTL design process is **high-level synthesis**, where a high-level description of the system is transformed into an RTL description. This involves translating the system's behavior, specified in a high-level language such as C or C++, into an equivalent RTL description in a hardware description language (HDL) like VHDL or Verilog.

The next stage is **RTL coding**, where the RTL description is further refined and optimised. This involves writing the RTL code that describes the data flow and operations of the system, and optimising this code to meet the design requirements.

The final stage is **RTL verification**, where the correctness of the RTL design is checked. This involves simulating the RTL design to ensure it behaves as expected, and formally verifying the design to prove its correctness.

Let's take a closer look at each of these stages.

High-Level Synthesis

High-level synthesis (HLS) is the process of transforming a high-level system description into an equivalent RTL description. This involves translating the system's behavior, specified in a

high-level language such as **C** or **C++**, into an equivalent RTL description in a hardware description language (HDL) like **VHDL** or **Verilog**.

HLS is a critical step in the RTL design process as it allows designers to describe the system's behavior in a high-level language, which is easier to understand and write than low-level HDL code. This not only speeds up the design process but also reduces the chance of errors.

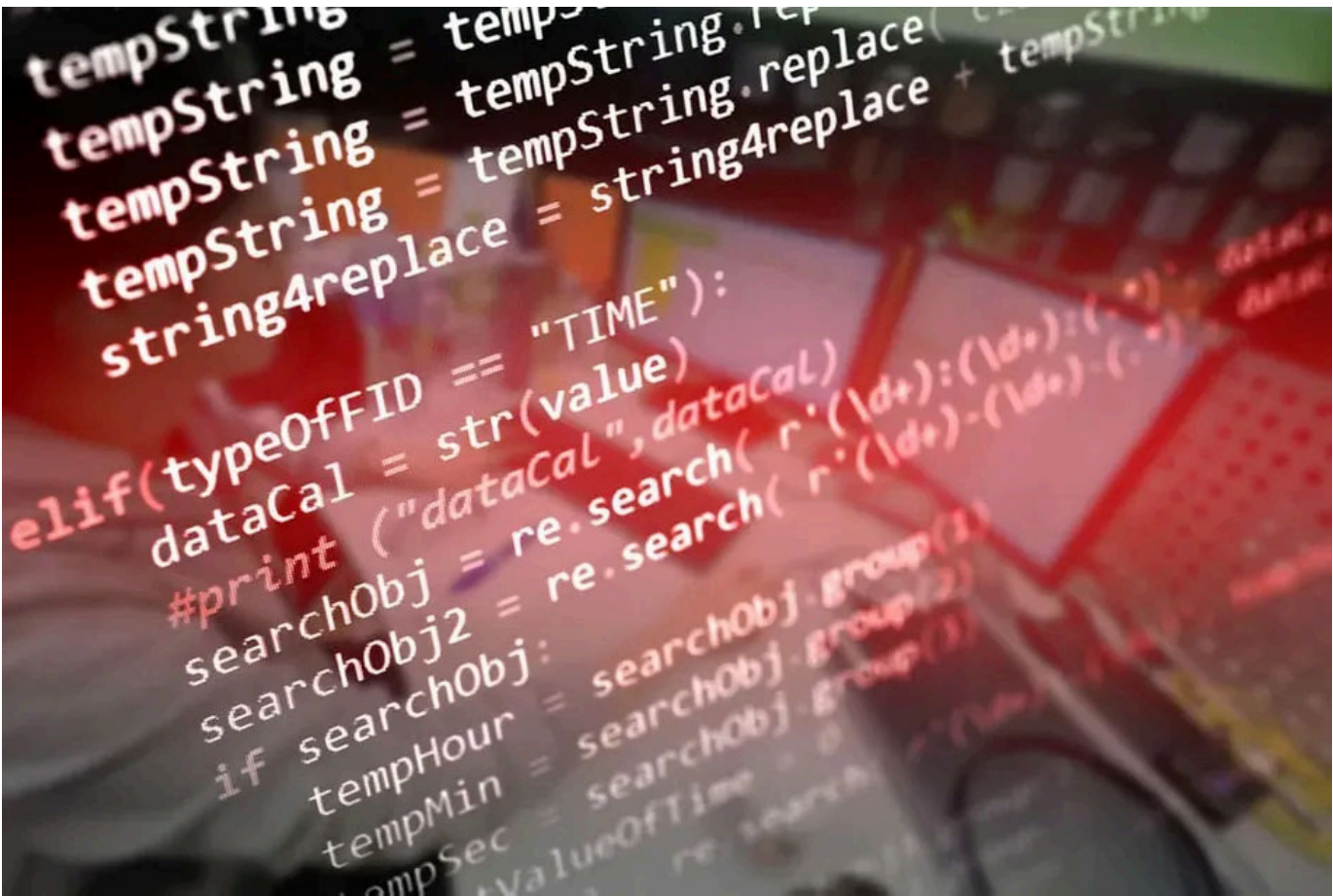
During HLS, the high-level system description is analyzed, and the data flow and operations of the system are extracted. These are then mapped onto an RTL description, which describes the system in terms of registers, operations, and data flows.

For example, consider a high-level description of a system that performs a simple arithmetic operation, such as adding two numbers. The HLS process would analyse this description, extract the addition operation and the data flow between the input and output, and map these onto an RTL description that specifies the addition operation and the flow of data between registers.

HLS tools, such as **Cadence Stratus** or **Synopsys' Catapult**, automate this process, allowing designers to quickly and accurately generate RTL descriptions from high-level system descriptions. [4] These tools also provide optimization features that can improve the performance, power, and area of the resulting RTL design.

RTL Coding

RTL coding is the process of writing the RTL description of a digital system using a hardware description language (HDL) such as VHDL or Verilog. This stage of the RTL design process involves specifying the data flow and operations of the system in terms of registers, operations, and data flows. RTL coding is crucial for defining the system's behavior and ensuring that it meets the design requirements.



RTL Coding in Command Prompt

During RTL coding, the designer writes the HDL code that describes the system's registers, operations, and data flows. This code is typically organised into modules, which are reusable blocks of code that represent specific parts of the system. Modules can be instantiated multiple times within a design, allowing for efficient reuse of code and simplification of the overall design.

VHDL and Verilog are the two most widely used HDLs for RTL coding. Both languages provide constructs for describing registers, operations, and data flows, as well as control structures for specifying the sequence of operations.

VHDL (VHSIC Hardware Description Language) is a strongly typed, concurrent language that provides a high level of abstraction for describing digital systems. It supports a wide range of data types and constructs, making it suitable for describing complex systems. VHDL code is typically organised into entities and architectures, where entities define the interface of a module, and architectures define the behavior of the module. [5]

For example, a simple VHDL module that implements a 4-bit adder might look like this:

```
entity adder is
  port (
```

```

    a, b: in std_logic_vector(3 downto 0);
    sum: out std_logic_vector(3 downto 0)
);
end entity adder;

```

```

architecture behavior of adder is
begin
    sum <= a + b;
end architecture behavior;

```

Verilog is a weakly typed, concurrent language that provides a lower level of abstraction than VHDL. It is well-suited for describing digital systems at the gate, register-transfer, and behavioral levels. Verilog code is typically organized into modules, which define both the interface and behavior of a part of the system.

For example, a simple Verilog module that implements a 4-bit adder might look like this:

```

module adder (
    input [3:0] a, b,
    output [3:0] sum
);
    assign sum = a + b;
endmodule

```

During the RTL coding stage, the designer must ensure that the HDL code accurately describes the desired behavior of the system and meets the design requirements. This may involve optimizing the code for performance, power, and area, as well as ensuring that the code is modular and reusable. Once the RTL coding is complete, the design moves on to the RTL verification stage, where the correctness of the RTL description is checked.

RTL Verification

RTL verification is the final stage in the RTL design methodology, where the correctness of the RTL design is checked. This stage is crucial to ensure that the RTL description accurately represents the intended behavior of the digital system and meets the design requirements. There are **two** main **methods** used in RTL verification: simulation and formal verification.

Formal verification tools, such as **Cadence's JasperGold** or **Synopsys' VC Formal**, use mathematical algorithms to exhaustively explore the state space of the design and prove the correctness of the formal properties. This provides a higher level of assurance of the design's correctness than simulation, as it can prove the correctness of the design for all possible input vectors, not just a subset. **UVM** is also a standardized framework for creating reusable and scalable testbenches, streamlining verification processes in RTL design.

For example, in a formal verification of a 4-bit adder, the formal properties might specify that for any pair of 4-bit numbers, the output of the adder is the sum of these numbers. The formal verification tool would then mathematically prove that this property holds for the RTL description of the adder.

RTL verification is a critical stage in the RTL design process, as it ensures that the RTL description accurately represents the intended behavior of the system and meets the design requirements. By using simulation and formal verification methods, designers can confidently verify the correctness of their RTL designs and proceed to the next stages of the design process.

Recommended Reading: Verilog vs VHDL: A Comprehensive Comparison

Advanced Concepts in RTL Design

Advanced concepts in RTL design involve techniques and methodologies that go beyond the basic RTL design process. These include RTL partitioning, pipelining, and clock domain crossing, among others. These advanced concepts are used to manage the complexity of large designs, optimise the performance, power, and area of the design, and handle special design requirements.

RTL Partitioning

RTL partitioning is a technique used to manage the complexity of large RTL designs. It involves dividing the design into smaller, more manageable parts, or partitions, each of which can be designed and verified independently. This not only simplifies the design process but also allows for parallelism, where different partitions can be designed and verified by different designers or teams at the same time.

Partitioning can be done in several ways, depending on the requirements of the design. One common method is **functional partitioning**, where the design is divided based on its

functional blocks. For example, in a microprocessor design, the processor core, memory, and input/output (I/O) interfaces might be separate partitions.

Another method is **hierarchical partitioning**, where the design is divided into a hierarchy of partitions. This is often used in complex designs that consist of several layers of hierarchy, with each layer representing a different level of abstraction.

Partitioning can also be done based on the physical characteristics of the design, such as low-power domains or clock domains. This is known as **physical partitioning** and is used for power estimation and timing analysis of design.

RTL partitioning is a powerful technique for managing the complexity of large designs. By dividing the design into smaller, more manageable parts, designers can focus on the details of each part without being overwhelmed by the complexity of the whole design. This not only simplifies the design process but also improves the quality of the design, as each part can be thoroughly designed and verified.

RTL Optimization

RTL optimization is a critical step in the RTL design process that aims to improve the performance, power, and area (PPA) of the design. [6] This involves refining the RTL description to make it more efficient and effective in meeting the design requirements.

Performance optimization involves improving the speed of the design. This can be achieved by techniques such as pipelining, where the design is divided into stages that can be executed in parallel, and parallelism, where multiple operations are performed simultaneously. **For example**, in a processor design, instruction fetching, decoding, execution, and write-back can be pipelined to improve the processor's instruction throughput.

Power optimization involves reducing the power consumption of the design. This can be achieved by techniques such as clock gating, where the clock signal is disabled for parts of the design that are not in use, and power gating, where the power supply is cut off to parts of the design that are not in use. **For example**, in a mobile device design, power gating can be used to turn off the GPS module when it is not in use to save power.

Area optimization involves reducing the size of the design. This can be achieved by techniques such as resource sharing, where multiple parts of the design share the same resources, and constant propagation, where constants are propagated through the design to simplify operations. **For example**, in a digital signal processing (DSP) design, multiple filters can share the same multiplier to save area.

RTL optimization is performed using RTL synthesis tools, such as **Synopsys' Design Compiler** or **Cadence's Genus**. These tools analyze the RTL description and apply various optimization techniques to improve the PPA of the design. The designer can guide the optimization process by setting constraints on the performance, power, and area of the design, and the synthesis tool will try to meet these constraints while optimizing the design.

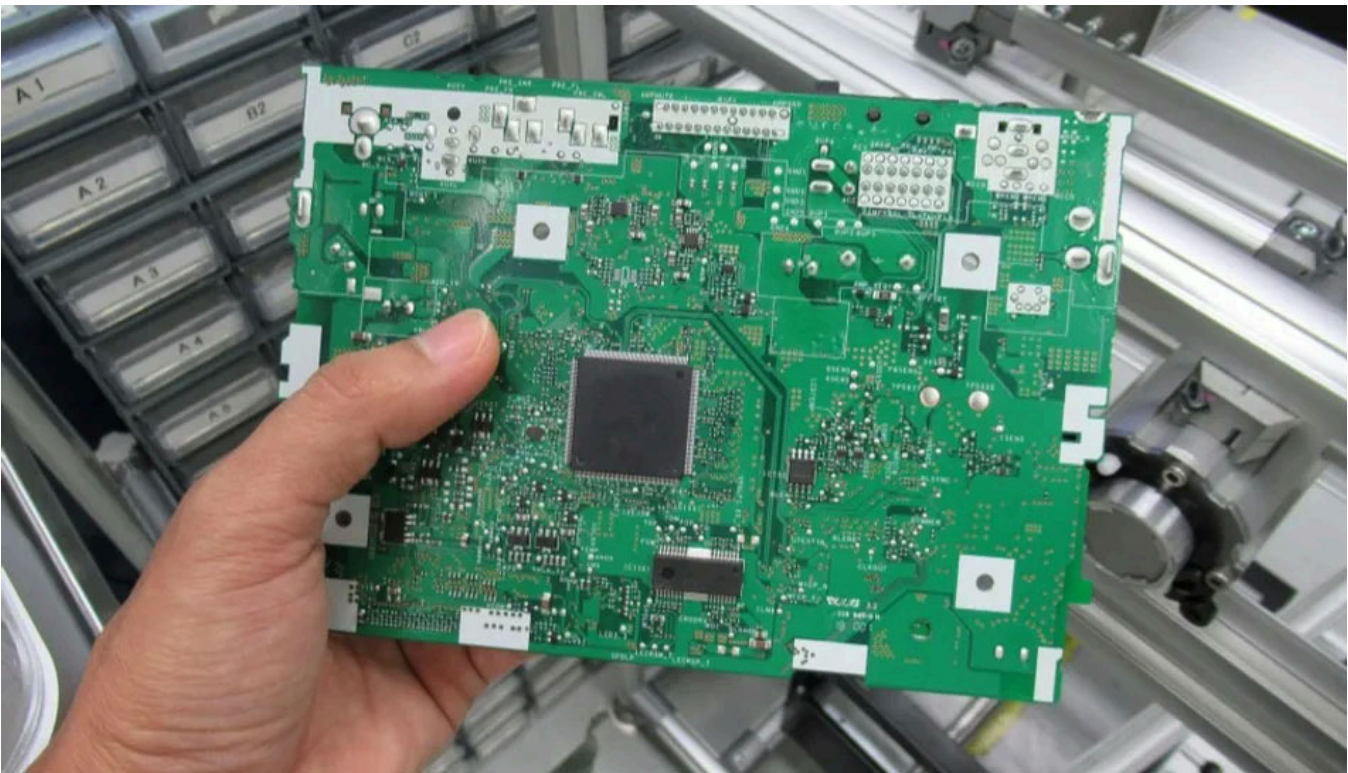
RTL optimization is a complex and challenging task that requires a deep understanding of the design and the optimization techniques. However, it is a crucial step in the RTL design process, as it can significantly improve the PPA of the design and make it more competitive in the market.

Challenges and Solutions in RTL Design

RTL design, while a powerful methodology for digital system design, presents several challenges. These include handling design complexity, managing power consumption, and ensuring design correctness. However, various solutions have been developed to address these challenges, including RTL partitioning, power optimization techniques, and formal verification methodology.

Handling Design Complexity

One of the main challenges in RTL design is managing the complexity of large designs. As the size and complexity of digital systems continue to grow, the task of designing and verifying these systems at the RTL level becomes increasingly difficult. This complexity can lead to longer design times, increased risk of design errors, and difficulty in meeting performance, power, and area requirements.



Complex designed PCB

RTL partitioning is a key technique for handling design complexity. By dividing the design into smaller, more manageable parts, designers can focus on the details of each part without being overwhelmed by the complexity of the whole design. This not only simplifies the design process but also improves the quality of the design, as each part can be thoroughly designed and verified.

Another approach to managing design complexity is the use of **high-level synthesis (HLS) tools**. These tools allow designers to describe the design at a higher level of abstraction using a high-level programming language, such as C or C++. The HLS tool then automatically generates the RTL description from the high-level description. This not only reduces the complexity of the design process but also allows designers to leverage their software programming skills.

Design reuse is another effective strategy for managing design complexity. By reusing proven design blocks, or IP cores, designers can reduce the amount of new RTL code that needs to be written and verified. This not only saves design time but also reduces the risk of design errors.

By using techniques such as RTL partitioning, high-level synthesis, and design reuse, designers can effectively manage the complexity of large designs and produce high-quality, efficient digital systems.

Ensuring Design Correctness

Ensuring design correctness is another significant challenge in RTL design. Given the complexity of modern digital systems, verifying that the RTL description accurately represents the intended behavior of the system is a non-trivial task. Errors in the RTL description can lead to functional errors in the final system, which can be costly to fix if not detected early in the design process.

Simulation is a widely used method for checking design correctness. By applying a set of input vectors to the RTL design and observing the resulting output vectors, designers can check whether the design behaves as expected. However, due to the exponential growth of the state space with the size of the design, it is impossible to simulate all possible input vectors for large designs. Therefore, creating a comprehensive set of test vectors that can effectively exercise all functional aspects of the design is a challenging task.

Formal verification offers a solution to this challenge. By expressing the intended behavior of the system as a set of formal properties, and using a formal verification tool to mathematically prove that these properties hold for the RTL design, designers can ensure the correctness of the design for all possible input vectors. However, writing the formal properties and interpreting the results of the formal verification tool requires a high level of expertise.

Another approach to ensuring design correctness is the use of **assertion-based verification (ABV)**. In ABV, designers write assertions, which are statements of intended behavior, in the RTL code. [7] These assertions are then checked during simulation or formal verification. If an assertion fails, it indicates a discrepancy between the intended and actual behavior of the design.

Linting is another technique used to ensure design correctness. Linting tools perform static analysis of the RTL code to check for common coding errors, adherence to coding standards, and potential synthesis and timing issues. By catching these issues early in the design process, linting tools can help improve the quality of the RTL code and reduce the risk of design errors.

By using a combination of simulation, formal verification, assertion-based verification, and linting, designers can ensure that the RTL description accurately represents the intended behavior of the system and meets the design requirements.

Recommended Reading: ASIC vs FPGA in chip design

Implementing RTL Design in a Real-World Scenario

A real-world example of implementing RTL design can be seen in the development of a **digital signal processing (DSP) system** for a wireless communication device. The DSP system is responsible for processing the digital signals received from the wireless interface, and performing operations such as filtering, modulation, and demodulation.

The first step in the RTL **design** process was to define the specifications of the DSP system. These specifications included the types of signals to be processed, the required processing operations, and the performance, power, and area requirements of the system.

Next, the RTL description of the DSP system was **written** using a hardware description language, such as VHDL or Verilog. The RTL description defined the data paths and control logic of the DSP system, including the registers, arithmetic units, and control units.

The RTL description was then verified using simulation and formal verification methods. For simulation, a set of test vectors was created based on the system's specifications, and these vectors were applied to the RTL design using a simulation tool. The output vectors were observed and compared with the expected results to check the correctness of the design.

For formal **verification**, a set of formal properties was written to express the intended behavior of the DSP system. A formal verification tool was used to mathematically prove that these properties hold for the RTL design.

After the RTL design was verified, it was **optimized** using an RTL synthesis tool. The tool analyzed the RTL description and applied various optimization techniques to improve the performance, power, and area of the design. The designer guided the optimization process by setting constraints on the performance, power, and area of the design.

Finally, the optimized RTL design was **translated** into a gate-level netlist using the synthesis tool. The netlist was then used for physical design, where the layout of the DSP system was created for fabrication. Testbench is used as a virtual environment to simulate and verify the functionality and timing of an RTL design before physical fabrication in VLSI.

This case study illustrates the application of RTL design in a real-world scenario. Despite the complexity of the DSP system, the RTL design process enabled the designers to effectively manage the design complexity, ensure the correctness of the design, and optimize the performance, power, and area of the system.

Conclusion

RTL design plays a pivotal role in the development of digital systems. It serves as the bridge between the conceptual, high-level system design and the physical, gate-level design. RTL design allows designers to focus on data paths and control logic without worrying about low-level details. RTL design significantly contributes to the competitiveness of digital systems in the market.

With the evolution of the size and complexity of digital systems, new challenges arise, and new solutions are developed. Techniques such as high-level synthesis, assertion-based verification, and power optimization are continually evolving to meet the changing needs of the industry. RTL design is an evolving field with new challenges and solutions.

FAQs

Q. What is the difference between RTL design and gate-level design?

A. RTL design and gate-level design are two levels of abstraction in the design of digital systems. RTL design focuses on the data paths and control logic of the system, while gate-level design focuses on the implementation of the system using logic gates. RTL design is closer to the system's functionality and is easier to understand and modify, while gate-level design is closer to the physical implementation and provides more accurate estimates of the system's performance, power, and area.

Q. What is the role of simulation in RTL design?

A. Simulation plays a crucial role in verifying the correctness of the RTL design. By applying a set of input vectors to the RTL design and observing the resulting output vectors, designers can check whether the design behaves as expected. However, due to the large state space of modern digital systems, creating a comprehensive set of test vectors is challenging.

Q. What is formal verification, and how is it used in RTL design?

A. Formal verification is a method for ensuring the correctness of the RTL design. It involves

expressing the intended behavior of the system as a set of formal properties and using a formal verification tool to mathematically prove that these properties hold for the RTL design.

Q. What is RTL synthesis and why is it important?

A. RTL synthesis is the process of translating the RTL design into a gate-level netlist. It involves analyzing the RTL description, applying various optimization techniques to improve the performance, power, and area of the design, and generating the gate-level netlist. RTL synthesis is important because it bridges the gap between the digital design and the physical design, and it plays a crucial role in meeting the design requirements.

References

- [1] Wikipedia. Register Transfer Level [Cited 2024 January 10] Available at: [Link](#)
- [2] Telecomtrainer. RTL Register Transfer Level [Cited 2024 January 10] Available at: [Link](#)
- [3] Tutorialspoint. Difference between Synchronous and Asynchronous Sequential Circuits [Cited 2024 January 10] Available at: [Link](#)
- [4] Cadence. Stratus High-Level Synthesis [Cited 2024 January 10] Available at: [Link](#)
- [5] Medium. VHDL — Understanding the Hardware Description Language [Cited 2024 January 10] Available at: [Link](#)
- [6] synopsys. Optimizing the RTL Design Flow with Real-Time PPA Analysis [Cited 2024 January 10] Available at: [Link](#)
- [7] ACM. A Survey on Assertion-based Hardware Verification [Cited 2024 January 10] Available at: [Link](#)