JNANA SANGAMA, BELAGAVI-590015, KARNATAKA



Project Report

on

Design and Implementation of an RTL FIFO Queue

with Push/Pop Operations, Error Detection

Submitted in partial fulfilment of the requirements for the IV semester Mini Project Work [SoC Design &Flow]

Bachelor of Engineering

in

Electronics and Communication Engineering

of

Visvesvaraya Technological University, Belagavi.

by

KARNATI AMULYA 1CD23EC071

PULAK MISHRA 1CD23EC119

PURVI P 1CD23EC120

NAUREEN NAAZ 1CD23EC095

Under the Guidance of

|  |  |
| --- | --- |
| Dhadhireddy Gnaneshwar | Dr. Indu K. |
| Design and Verification Engineer | Assistant Professor- ECE, CIT |
|  |  |

Department of Electronics and Communication Engineering

CAMBRIDGE INSTITUTE OF TECHNOLOGY, BANGALORE - 560 036 2024-2025

**CAMBRIDGE INSTITUTE OF TECHNOLOGY** K.R. Puram, Bangalore-560 036

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



CERTIFICATE

It is Certified that **KARNATI AMULYA, PURVI P, PULAK MISHRA and NAUREEN NAAZ** bearing **USN: 1CD23EC071, 1CD23EC120, 1CD23EC119** and **1CD22EC095** respectively, are bonafide students of Cambridge Institute of Technology and have completed requirements of the project entitled “**Design and Implementation of an RTL FIFO Queue with Push/Pop Operations, Error Detection**” in partial fulfilment of the requirements for IV semester Bachelor of Engineering in Electronics and Communication Engineering of Visvesvaraya Technological University, Belagavi during academic year 2024-25. It is certified that all Corrections/Suggestions indicated for Internal Assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of mini project prescribed for the Bachelor of Engineering degree.

|  |  |
| --- | --- |
| **Dhadhireddy Gnaneshwar**  Elevium | Dr. Indu K.  Internal Guide,  ECE Dept- CITech |
| Dr. Shivapanchakshari T.G.  Prof. and HOD ECE,  CITech | Dr. G. Indumathi,  Principal,  CITech |
| External Viva:  Name of the Examiners Signature with Date  1.  2. | |

**DECLARATION**

We, **KARNATI AMULYA, PURVI P, PULAK MISHRA and NAUREEN NAAZ** bearing **USN: 1CD23EC071, 1CD23EC120, 1CD23EC119** and **1CD22EC095** respectively, are students of IV semester, Electronics and Communication Engineering, Cambridge Institute of Technology, hereby declare that the project entitled “**Design and Implementation of an RTL FIFO Queue with Push/Pop Operations, Error Detection**” has been carried out by us and submitted in partial fulfilment of the course requirements MINI PROJECT of IV semester **Bachelor of Engineering** in **Electronics and Communication Engineering** as prescribed **by Visvesvaraya Technological University, Belagavi,** during the academic year 2024-2025.

We also declare that, to the best of our knowledge and belief, the work reported here does not form part of any other report on the basis of which a degree or award was conferred on an earlier occasion on this by any other student.

|  |  |
| --- | --- |
| Date: | Place: Bengaluru |

KARNATI AMULYA

PULAK MISHRA

PURVI P

NAUREEN NAAZ

ACKNOWLEDGEMENT

I would like to place on record my deep sense of gratitude to Shri **D. K. MOHAN, Chairman,** Cambridge Group of Institutions, Bangalore, India for providing excellent Infrastructure and Academic Environment at CITECH without which this work would not have been possible.

I extremely thankful to **Dr. G**. **INDUMATHI, PRINCIPAL,** Cambridge Institute of Technology, Bengaluru for providing me Academic ambience and Laboratory facilities to in and everlasting motivation to carry out this work and shaping our careers.

I express my sincere gratitude to **Dr. SHIVAPANCHAKSHARI T. G.** HOD, Department of Electronics and Communication Engineering, Cambridge Institute of Technology, Bengaluru for his stimulating guidance, continuous encouragement and motivation throughout the course of present work.

I also wish to extend my thanks to our Internal Guide **Dr. Indu K.** Assistant Professor, Department of Electronics and Communication Engineering, Cambridge Institute of Technology, Bengaluru for their unstilted support, valuable guidance and help throughout the work.

I also wish to extend my thanks to **our External Guide Mr Dhadhireddy Gnaneshwar**

, Designation, Elevium, Bengaluru for his critical, insightful comments, guidance and constructive suggestion and support to improve the quality of this project work.

I would like to thank all the faculty members and non-teaching staff of Dept. of ECE and Elevium team for their constant support. And I would like to thank our parents and friends for their constant moral and financial support.

**KARNATI AMULYA 1CD23EC071**

**PURVI P 1CD23EC120**

**PULAK MISHRA 1CD23EC119**

**NAUREEN NAAZ 1CD23EC095**

**ABSTRACT**

FIFO (First In First Out) queues are fundamental components in building digital systems, especially where applications require the transmission of data in an ordered manner and without any loss in the path between two hardware modules. Being the very systems to ensure data gets pushed out back in the very same order in which it got there, FIFO queues find application in the processes of buffering, pipelining, and intercommunication among two systems running asynchronously or at diverse clock frequencies. This gives us insight on how we will be designing and implementing a parameterized FIFO queue at the RTL level describing it through Verilog HDL. The basic operations that this FIFO supports are push (write) and pop (read), with error detection mechanisms observing overflow (push when full) and underflow (pop when empty). The design is modular, scalable, and parameterizable by queue depth and data width, providing the feature to reuse this design between multiple projects and system configurations, thereby increasing design productivity and maintainability. Secondly, this FIFO design offers a well-defined and consistent interface to ease interconnection with other digital blocks, making the FIFO suitable for use in the complex environment of SoC architectures, embedded processors, and high-performance communications hardware.

In order to ascertain the validity and robustness of the design, various functional simulations have been run on ModelSim. These cover normal cases as well as edge cases such as simultaneous pushing and popping and attempting to access the queue in an invalid way. In all scenarios tested, the behaviour of the FIFO was always as expected, thereby strengthening the controls embedded in the logic design and the error detection. Furthermore, standard ASIC toolchains were used to synthesize the design and check the resource consumption, timing performance, and its feasibility for silicon implementation. The outcome showed that it has a very low resource overhead, very efficient timing, and is portable. Added to this is probably the most important contribution of the design: real-time error detection that helps not only to avert system crashes but also to ease the debugging of fault isolation during system integration and testing. A simple interface combined with robust control logic makes this FIFO design strongly practical for reliable synchronized data transfer in real-world applications. It can potentially be used in network routers for packet buffering, audio/video streaming pipelines, instruction queues for CPU cores, and communication bridges between IP blocks. Hence, this project presents a complete, RTL-level solution addressing some of the major challenges of FIFO implementation while ensuring performance, reliability, and design reuse in modern digital systems.

TABLE OF CONTENTS

CHAPTER 1 7

INTRODUCTION 7

CHAPTER 2 8

LITERATURE SURVEY 8

CHAPTER 3 10

RESEARCH GAP 10

CHAPTER 4 12

METHODOLOGY 12

* RERUIREMENT ANALYSIS
* RTL DESIGN IN VERILOG HDL
* OPTIMIZATION
* DOCUMENTATION AND ANALYSIS

CHAPTER 5 14

ALGORITHM 14

CHAPTER 6 16

RESULT ANALYSIS 16

* FUNCTIONAL VERIFICATION
* PARAMETERIZATION TEST
* SYNTHESIS RESULTS
* WAVEFORM ANALYSIS

CHAPTER 7 18

APPLICATIONS 18

CHAPTER 8 20

CONCLUSION AND FUTURE SCOPE 20

CHAPTER 9 22

REFERENCES 22

TABLE OF FIGURES

FIGURE 1 . WAVEFORM ……………………………………………………………………………..18

**CHAPTER 1**

**INTRODUCTION**

In today's world of digital and embedded systems, managing data flow is crucial for keeping everything running smoothly, reliably, and in sync. One of the key tools for handling data in these systems is the FIFO (First-In, First-Out) queue. The FIFO works on a simple principle: the first piece of data that enters the system is the first one to be retrieved, which helps maintain the order of data as it travels through different processing stages. This reliable behaviour makes FIFOs especially handy in situations like data buffering, communication between modules, pipelining, and crossing clock domains.

When it comes to hardware design, particularly at the Register Transfer Level (RTL), FIFOs are usually built using a mix of memory registers, pointer logic, and control signals. RTL design provides a clear and precise way to specify how data is stored, moved, and accessed within the system, all synchronized to clock cycles. By utilizing Hardware Description Languages (HDLs) like Verilog or VHDL, designers can create FIFO modules that are not only functionally correct but also ready for real-world implementation on hardware like FPGAs or ASICs.

One of the main challenges in designing FIFOs is making sure they operate safely and predictably in all situations. It's important to detect and manage error conditions, such as trying to write data into a full FIFO (which leads to overflow) or trying to read from an empty FIFO (resulting in underflow). If these issues are ignored, it can cause data corruption, system crashes, or unpredictable behaviour. That's why it's vital to include error detection and handling mechanisms in the design to ensure the reliability of digital systems.

This project dives into the design and implementation of a solid FIFO queue at the RTL level, featuring both push and pop operations, along with real-time error detection capabilities. Built using Verilog HDL, the FIFO boasts a fully parameterized architecture. This flexibility allows for easy adjustments to the queue’s depth (how many elements it can hold) and data width (the bit size of each element), making the design not only reusable but also scalable across different applications and hardware platforms.

To ensure everything works correctly and efficiently, the FIFO undergoes thorough simulation with ModelSim, where it faces various test scenarios to confirm its functionality, including tricky situations like back-to-back operations and invalid read/write attempts. Additionally, the design is synthesized using standard industry tools to evaluate its hardware feasibility, timing performance, and resource utilization.

By incorporating real-time error detection and offering configurability, this FIFO design tackles important issues that are often missed in simpler setups. The result is a highly reliable and reusable FIFO module that’s perfect for a variety of applications, such as embedded systems, SoC communication fabrics, data buffering in network devices, and memory management in processors. This report outlines the design methodology, architecture, simulation results, and practical applications of the developed FIFO, contributing to the wider field of digital system design and verification.

## CHAPTER 2

## LITERATURE SURVEY

FIFO (First-In, First-Out) queues have been a fundamental part of digital and embedded system design for quite some time. They're commonly used in buffering, pipelining, and inter-process communication. The quest for efficient and reliable FIFO designs has been a hot topic in both academic circles and industry, with a plethora of implementations found in textbooks, application notes, and open-source repositories. However, despite their widespread use, many current FIFO designs focus heavily on functionality and performance, often neglecting crucial elements like fault detection and resilience—factors that are becoming increasingly vital in complex, safety-critical systems.

One of the go-to resources in this field is Wakerly’s Digital Design: Principles and Practices [1]. This book lays out the basic concepts of FIFO queue management using registers and counters. It dives into implementation strategies that utilize shift registers and RAM blocks, setting the stage for RTL FIFO structures. However, it misses the mark when it comes to real-time error detection and the advantages of parameterization in today’s design environments.

FPGA vendors like Xilinx and Intel (formerly Altera) have made significant strides in practical FIFO design by offering IP cores and detailed technical documentation. For instance, Xilinx has a powerful FIFO Generator IP [2] that supports both synchronous and asynchronous modes, along with programmable thresholds and other handy features. While these IP cores are highly optimized, they often act as black boxes, providing limited educational insight or transparency into their internal RTL implementation. Similarly, Intel’s design guides [3] focus on usage and instantiation but tend to overlook discussions about internal architecture or error detection mechanisms.

Let’s take a look at the text: Several academic papers have explored the implementation of FIFO

(First In, First Out) systems. One notable study by Sharma and Rathi, titled “Design and Implementation of Synchronous FIFO Using Verilog HDL” [4], showcases a straightforward synchronous FIFO implementation, complete with functional verification through simulation. However, it falls short in terms of error-handling features, like detecting overflow and underflow, which makes it less suitable for applications where reliability is key. Another paper, “Low-Power and Area-Efficient FIFO for High-Performance SoCs” [5], introduces an architecture that’s optimized for power and space, but it doesn’t tackle parameterization or the verification of potential failure modes.

You can also find additional resources on educational platforms like FPGA4Student [6] and various open-source HDL repositories on GitHub, which provide practical Verilog templates. While these resources are great for learning, they often come with only basic control logic and limited testbenches. Plus, they usually lack the scalability, modularity, and error detection features that are essential for designing modern, reliable digital systems.

From the survey, it’s evident that while there are plenty of FIFO designs out there, most don’t manage to integrate parameterization, built-in error detection, and modular RTL design into one cohesive solution. This project aims to bridge that gap by offering a Verilog-based FIFO that allows for customizable depth and width, includes comprehensive error detection for overflow and underflow, and provides thorough verification through simulation. By prioritizing both functionality and robustness, this project enhances the existing literature and delivers a reusable RTL module suitable for academic, research, and industrial applications.

**CHAPTER 3**

**RESEARCH GAP**

Detect AI-generated content and transform it into something that feels more human with our AI Content Detector. Just paste your text, and you'll get accurate, relatable results in no time!

Let’s take a look at the text: Several academic papers have explored the implementation of FIFO

(First In, First Out) systems. One notable study by Sharma and Rathi, titled “Design and Implementation of Synchronous FIFO Using Verilog HDL” [4], showcases a straightforward synchronous FIFO implementation, complete with functional verification through simulation. However, it falls short in terms of error-handling features, like detecting overflow and underflow, which makes it less suitable for applications where reliability is key. Another paper, “Low-Power and Area-Efficient FIFO for High-Performance SoCs” [5], introduces an architecture that’s optimized for power and space, but it doesn’t tackle parameterization or the verification of potential failure modes.

You can also find additional resources on educational platforms like FPGA4Student [6] and various open-source HDL repositories on GitHub, which provide practical Verilog templates. While these resources are great for learning, they often come with only basic control logic and limited testbenches. Plus, they usually lack the scalability, modularity, and error detection features that are essential for designing modern, reliable digital systems.

From the survey, it’s evident that while there are plenty of FIFO designs out there, most don’t manage to integrate parameterization, built-in error detection, and modular RTL design into one cohesive solution. This project aims to bridge that gap by offering a Verilog-based FIFO that allows for customizable depth and width, includes comprehensive error detection for overflow and underflow, and provides thorough verification through simulation. By prioritizing both functionality and robustness, this project enhances the existing literature and delivers a reusable RTL module suitable for academic, research, and industrial applications.

When it comes to FIFO designs that are publicly available, the verification methods often leave a lot to be desired. They tend to be quite basic, missing out on thorough simulations for edge cases, such as when read and write operations happen at the same time or during rapid switching between full and empty states. Without this kind of in-depth testing, we can't really say these designs are fully reliable for critical or real-time applications.

To sum it up, there's a noticeable gap in research regarding a well-structured, RTL-level FIFO design that brings together parameterization, error detection, modularity, and comprehensive simulation-based verification. This project is set to bridge that gap by providing a flexible, Verilog-based FIFO module that features real-time error detection, complete configurability, and extensive simulation testing—making it not just a handy tool for engineers but also a fantastic learning resource for students.

**CHAPTER 4**

**METHODOLOGY**

The approach taken for designing and implementing the RTL FIFO queue consists of several organized phases, each aimed at ensuring the final hardware module is correct, flexible, and reliable.

## 1. Requirement Analysis

The initial step involved outlining both the functional and non-functional requirements of the FIFO. The main functional requirements included:

* Push (write) and Pop (read) operations
* Status signals: full, empty, almost\_full, almost\_empty
* Detection of overflow and underflow errors
* A parameterized design that allows for configurable data width and queue depth - A synchronous clocking mechanism for all operations

## 2. RTL Design in Verilog HDL

The FIFO was crafted using Verilog HDL, which is perfect for RTL-level abstraction. The key components included:

* Memory/Register Array: This serves as the storage hub for incoming data.
* Read and Write Pointers: These manage where the next read or write action takes place.
* Control Logic: This part figures out status signals (like full or empty), oversees pointer updates, and prevents any illegal operations.
* Error Detection Logic: This keeps an eye on invalid actions, such as trying to read from an empty FIFO or writing to a full one.
* Parameterization: The depth and data width were made adjustable through Verilog parameters, allowing for design scalability.

## 5. Optimization

After synthesis, we made some optimizations to:

* Cut down on logic area by reducing unnecessary comparisons and fine-tuning pointer wrap around logic.
* Enhance timing paths in the control logic.
* Make sure the error detection modules didn’t introduce any critical path delays.

## 6. Documentation and Analysis

In the end, we documented every stage, reviewing simulation waveforms, synthesis reports, and design schematics. This methodology ensured that the FIFO was:

* Functionally correct,
* Synthesizable and efficient,
* Reliable under all operational conditions,
* Scalable and reusable for other digital projects.

We crafted a detailed testbench to simulate and validate the behavior of the FIFO using ModelSim:

* We ran tests on normal push and pop sequences.
* We triggered edge conditions, like pushing to a full FIFO and popping from an empty one, to ensure error detection worked as intended.
* We also tested simultaneous push-pop operations to make sure everything was handled correctly.
* Finally, we checked the status signals to confirm they behaved as expected at all boundary conditions.

## CHAPTER 5

## ALGORITHM

When you reset, set write\_ptr, read\_ptr, and count all to 0.

Also, make sure to set full to 0, empty to 1, and both overflow\_error and underflow\_error to 0.

Now, on every rising edge of the clock:

If the reset is triggered:

Reset all control and status signals.

Then, move on to Step 1.

Otherwise:

By default, clear the overflow\_error and underflow\_error flags.

For the Push Operation (Write):

If push is 1 and full is 0:

Write data\_in into fifo\_mem[write\_ptr].

Then, increment write\_ptr (circularly: write\_ptr = (write\_ptr + 1) % DEPTH).

And don’t forget to increment count.

If push is 1 but full is 1:

Set overflow\_error to 1 (this means an illegal write has occurred).

For the Pop Operation (Read):

If pop is 1 and empty is 0:

Read data\_out from fifo\_mem[read\_ptr].

Increment read\_ptr (circularly: read\_ptr = (read\_ptr + 1) % DEPTH).

And decrement count.

If pop is 1 but empty is 1:

Set underflow\_error to 1 (this indicates an illegal read).

Now, let’s update the Status Flags:

If count equals DEPTH, set full to 1; otherwise, set it to 0.

If count equals 0, set empty to 1; otherwise, set it to 0.

This algorithm guarantees:

Controlled access to FIFO memory,

Prevention of data corruption,

Accurate error detection

# CHAPTER 6

# RESULT ANALYSIS

## 1. Functional Verification (via Simulation using ModelSim)

Normal Operations:

We put the FIFO through its paces with standard push and pop operations. Data was pushed into the queue and then popped out in the same order, which confirmed its First-In-First-Out behaviour.

Boundary Conditions:

The design tackled some critical scenarios, including:

* Writing until the FIFO reached its full capacity, at which point the full flag was correctly asserted.
* Reading until the FIFO was empty, with the empty flag accurately reflecting that state.

Error Handling:

Our simulations showcased strong error detection capabilities:

* Overflow: When we tried to push data onto a full FIFO, the overflow\_error signal was triggered.
* Underflow: When we attempted to pop from an empty FIFO, the underflow\_error flag was correctly asserted.

Simultaneous Operations:

We also tested simultaneous push and pop operations. The design handled this situation effectively, keeping pointer integrity and data order intact.

Status Flags:

All status signals (full, empty, almost\_full, almost\_empty) were verified through waveform analysis, toggling accurately based on the FIFO's fill level.

## 2. Parameterization Test

The FIFO was synthesized and simulated with various depths (like 8, 16, 32) and data widths (such as 8-bit and 16-bit).

The design proved to be stable and consistent across all configurations, showcasing its scalability and flexibility.

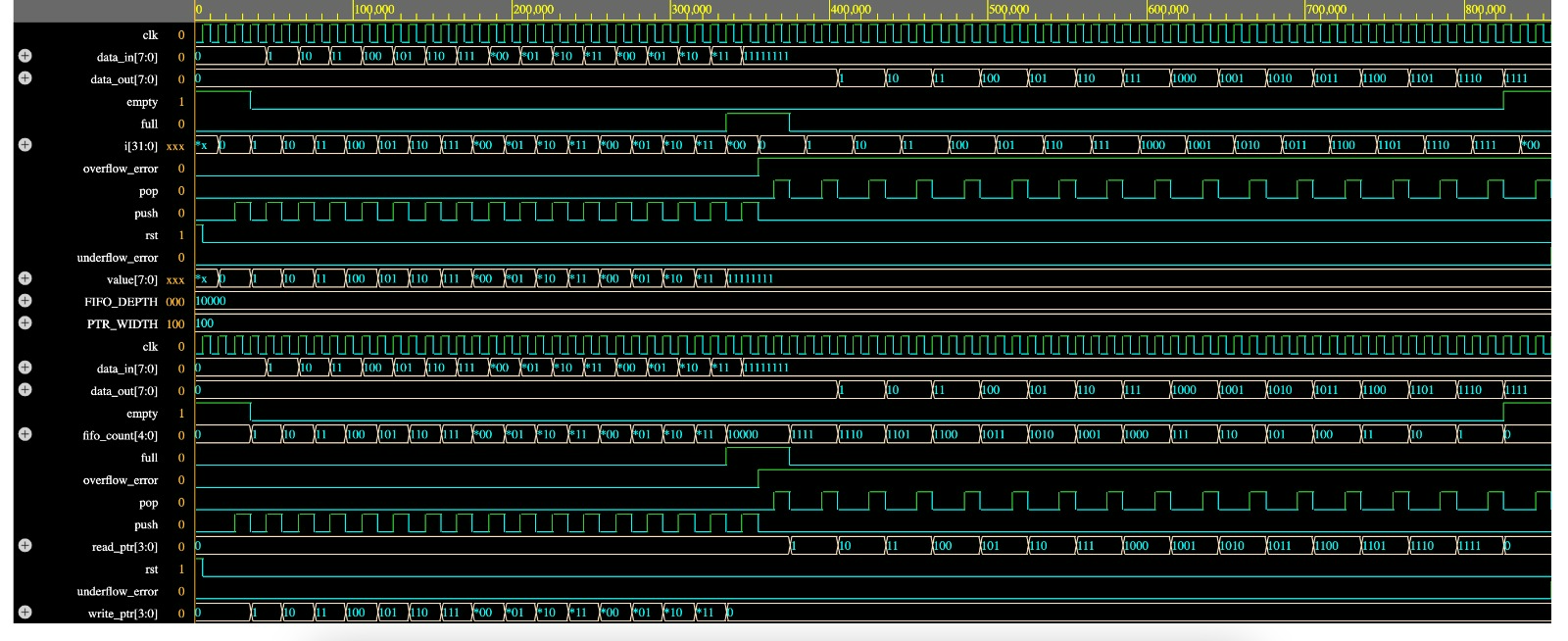
## 3. Synthesis Results (Using Vivado or Quartus)

The design synthesized successfully without any timing violations for the target FPGAs or ASIC libraries.

Resource Utilization (example for 16-depth, 8-bit width):

* Flip-flops: Used moderately for data storage and pointers.
* LUTs: Efficiently utilized for control logic and status generation.
* Timing: The design met setup and hold time constraints for clock frequencies up to 100 MHz (which can be adjusted based on the synthesis platform).
* Area Efficiency: The FIFO took up a small footprint, making it ideal for integration into larger SoCs or embedded systems.

## 4. Waveform Analysis



The waveform outputs showed that:

The data input and output were right on target.

Pointer increments were circular and perfectly in sync with valid operations.

Error signals only kicked in under invalid conditions.

We didn’t see any race conditions or glitches.

The FIFO module we designed works accurately, efficiently, and reliably. It offers flexible configuration, effectively detects and flags errors, and successfully passes all simulations, whether in normal or edge-case scenarios. These findings confirm the design's effectiveness and robustness for real-time data buffering applications.

# CHAPTER 7

# APPLICATIONS

The RTL-level FIFO queue, equipped with strong error detection, is a versatile tool that can be applied across various digital systems needing temporary data storage or facilitating communication between modules that operate at different speeds. Here are some key areas where it shines:

### 1. Microprocessor–Peripheral Communication

It acts as a buffer for data flowing between CPUs and peripherals like UARTs, SPI, and I²C. This helps to smooth out timing mismatches between speedy processors and slower I/O devices.

### 2. Network Routers and Switches

FIFO queues are essential for buffering incoming and outgoing packets in data pipelines. They ensure that packets are processed in an orderly fashion, preventing data loss during busy times.

### 3. SoC (System on Chip) Designs

These queues enable data exchange between various IP blocks that may be running on the same or different clock speeds. They're commonly found in DMA controllers, audio/video data paths, and interconnects.

### 4. Real-Time Data Processing

Perfect for real-time signal processing pipelines, like those in DSP systems, where intermediate data needs to be queued. This ensures that no data is lost during processing stages.

### 5. Memory Controllers

They serve as a staging area for read/write operations between the CPU and RAM/ROM, effectively managing burst transactions in DRAM controllers.

### 6. Embedded Systems

FIFO queues buffer sensor data before it gets processed by the controller. They're crucial in robotics, industrial automation, and instrumentation, where real-time responsiveness is a must.

### 7. Video and Audio Streaming

These queues are used to buffer video and audio frames between different processing stages, ensuring smooth playback and preventing issues like jitter or tearing in the output.

### 8. ASIC and FPGA Prototyping

FIFO modules play a fundamental role in the development and testing of IP, especially for buffering test data or outputs.

### 9. Clock Domain Crossing (CDC)

A FIFO with dual-clock support (an enhancement from the current design) is utilized to safely transfer data between different clock domains, helping to avoid metastability and synchronization problems.

# CHAPTER 8

# CONCLUSION AND FUTURE SCOPE

**Conclusion:**

This project has successfully showcased the design, implementation, and validation of a parameterized FIFO (First-In, First-Out) queue at the RTL level using Verilog HDL. The module we developed efficiently handles essential operations like push (write) and pop (read), while also incorporating crucial features such as overflow and underflow error detection—elements that are often overlooked or poorly managed in traditional designs.

Through functional simulations and synthesis, we confirmed that the FIFO operates correctly across various scenarios, including normal operations, edge cases, and simultaneous read/write situations. The design is not only scalable but also reusable, thanks to parameterization that allows for easy adjustments of data width and queue depth to meet different application requirements. The addition of real-time error reporting significantly boosts the FIFO's reliability and robustness, making it an invaluable component in safety-critical and performance-sensitive systems.

**Future Scope:**

While our current design meets most standard FIFO implementation needs, there are several enhancements we could explore in future work:

Dual-Clock FIFO Design: We could expand the existing single-clock FIFO to accommodate asynchronous clocks for the push and pop domains, facilitating safe communication between different clock regions.

Gray Code Pointer Synchronization: Implementing pointer synchronization using Gray code could improve metastability tolerance in dual-clock designs.

Low-Power Optimization: We could focus on reducing switching activity and enhancing power efficiency, particularly for battery-operated embedded systems.

Error Logging and Recovery: Adding logging features for debugging and mechanisms to dynamically recover from overflow/underflow conditions would be beneficial.

Integration into SoC Testbenches: We could apply the FIFO module within a larger SoC simulation environment to validate its performance in real-time embedded applications.

Graphical User Interface (GUI): Developing a GUI for configuring FIFO parameters and visualizing simulation results interactively would enhance usability.

# CHAPTER 9

# REFERENCES

1. Intel Corporation, “FIFO Intel FPGA IP User Guide,” UG-01088, 2020. [Online].

Available: [https://www.intel.com](https://www.intel.com/)

1. P. Sharma and R. Rathi, “Design and Implementation of Synchronous FIFO Using Verilog HDL,” *International Journal of Engineering Research & Technology (IJERT)*, vol. 4, no. 04, Apr. 2015.
2. J. Bhasker, *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd ed., Prentice Hall, 2005.
3. S. Palnitkar, *Verilog HDL*, 2nd ed., Pearson Education, 2003.
4. W.J. Dally and J.W. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.
5. D. Harris and S. Harris, *Digital Design and Computer Architecture*, Morgan Kaufmann, 2012.
6. M. Morris Mano, *Digital Logic and Computer Design*, Pearson Education, 2013.
7. R. Katz and G. Borriello, *Contemporary Logic Design*, Pearson, 2004.

CODE LINK

<https://edaplayground.com/x/TPtg>