High Level Design Specification (HLDS)

for

Asynchronous FIFO (First In First Out)

Version 1.0

Prepared by Deepthi Chevuru

<ECE-593: Fundamentals of Pre-Silicon Validation – Venkatesh Patil>

Created on: 01/13/2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The Asynchronous FIFO is designed to provide efficient and synchronized data storage and retrieval in a multi-clock domain environment. This document serves as a guide to understand the high-level architecture and functionality of the Asynchronous FIFO system.

1.2 Document Conventions

FIFO	- First In First Out
Producer	- Sender of data who writes data to FIFO
Consumer	- The module which reads data from FIFO for further processing
Asynchronous	- Writes to and Reads from FIFO happen at 2 different clock frequencies
Clock Frequency	- Rate at which the data transfer happens in each module
Idle Cycles	- Clock cycles during which the read or write operation do not happen.
FIFO depth	- FIFO depth refers to the maximum number of data elements that a First-In-First-Out (FIFO) buffer can store or accommodate in a sequential manner.
Bit stream length	- 8 bits
Clock dependency	- Different clocks for Producer and consumer. Rate at which data is written is almost double the rate at which data is read.
Synchronizer	- A synchronizer is a circuit or mechanism designed to mitigate the risk of metastability by ensuring proper synchronization between signals in different clock domains
Metastability	- Metastability is a phenomenon in digital systems where a flip-flop or latch enters an undefined state due to the uncertainty of input signal timing at the sampling edge of a clock.
Status Flags	- Variable which stores status of FIFO if full or empty
Handshaking	- Handshaking is a communication protocol involving a series of coordinated signals or events between sender and receiver to ensure synchronized and reliable data transfer.

1.3 Intended Audience and Reading Suggestions

This document is targeting all the stakeholders involved in design, development and verification of this product. The primary audience include System Architects, Software Developers, Project Managers, Verification Team, Validation Engineers, Maintenance and support teams, Integration teams and Regulatory authorities.

1.4 Product Scope

The Asynchronous FIFO is designed to facilitate asynchronous data transfer between distinct clock domains, ensuring proper synchronization and reliable data storage and retrieval. It can be used in multiple designs like Communication interfaces, Memory interfaces, Clock Domain crossing, etc

Digital Signal Processing (DSP):
In DSP applications, where data is processed using algorithms and filters, asynchronous FIFOs can be used to manage the flow of data between different processing stages that may operate at different rates.

Memory Interfaces:
Asynchronous FIFOs are employed in memory interfaces to enable seamless data transfer between a processing unit and memory components, particularly in situations where the memory subsystem and the processing unit function with distinct clock domains.

Clock Domain Crossing:
Asynchronous FIFOs help in transferring data between these clock domains, ensuring synchronization and preventing data loss or corruption.

Communication between Processors:
Asynchronous FIFOs enable efficient communication between these processors by handling the asynchronous nature of their operation.

Data Buffering and Flow Control:
Asynchronous FIFOs act as buffers between components with varying data arrival rates. They help in managing the flow of data, preventing bottlenecks, and ensuring smooth communication between different parts of the SoC.

Interfacing with External Devices:
Asynchronous FIFOs provide a bridge for transferring data between the external devices and the internal components of the SoC.

Network-on-Chip (NoC) Architectures:
In SoCs with complex communication architectures like NoCs, asynchronous FIFOs facilitate data exchange between different nodes or processing elements. They contribute to efficient and reliable data communication in a networked environment.

Fault Tolerance and Robustness:
Asynchronous FIFOs contribute to the fault tolerance of an SoC by providing a mechanism to handle variations in data arrival times. This enhances the robustness of the overall system.

Custom Interfaces and Protocols:
Asynchronous FIFOs can be customized to accommodate specific communication requirements between different blocks in the SoC.

Network Switching:
In networking applications, asynchronous FIFOs are used to manage the flow of data between different network interfaces or between different components of a network switch. This helps in handling variable data rates and maintaining proper synchronization.

Video and Image Processing:
Asynchronous FIFOs find applications in video and image processing systems where different stages of processing may operate at different speeds. They help manage the flow of pixel data between various processing units.

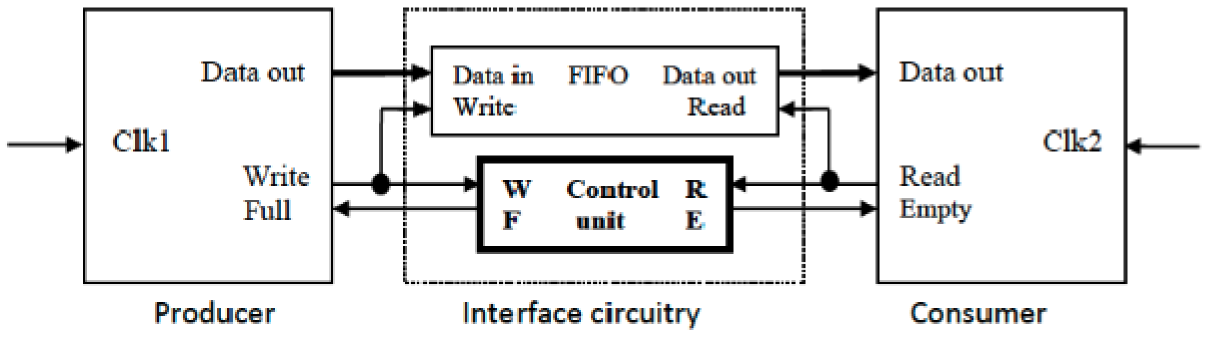
1.5 References

http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf
https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf
www.google.com
https://www.google.com/search?sca_esv=598681343&sxsrf=ACQVn08hZofuJ-X1nkb0iwwH9OQqStfbSg:1705370909887&q=Asynchronous+FIFO+detailed+block+diagram&tbm=isch&source=lnms&sa=X&ved=2ahUKEwjF7aza6eCDAxXHCDQIHey4BZ0Q0pQJegQICxAB&biw=1920&bih=995&dpr=1#imgrc=I-KklrrwVnfe8M
https://escholarship.org/content/qt4mm6706x/qt4mm6706x_noSplash_a6727c85d86d00c6ec405efe8a9ec443.pdf?t=lnq4pt
https://www.google.com/search?q=Half+Interlocked+Asynchronous+handshaking+Protocol&tbm=isch&ved=2ahUKEwjqkpvd7-CDAxUVFzQIHaV4CS8Q2-cCegQIABAA&oq=Half+Interlocked+Asynchronous+handshaking+Protocol&gs_lcp=CgNpbWcQA1CfCFifCGCrC2gAcAB4AIABnQGIAdcBkgEDMS4xmAEAoAEBqgELZ3dzLXdpei1pbWfAAQE&sclient=img&ei=buulZeqjHpWu0PEPpfGI-AI&bih=995&biw=1920#imgrc=0XCjle84Bgaa2M

2. Overall Description

2.1 Product Perspective

The system comprises multiple modules, each serving specific functions. The primary modules include:



Producer Module:
Responsible for writing data to the queue.

Interface Circuitry:
Facilitates asynchronous data transfer between modules. It is in turn a component of two parts - FIFO queue and Control Unit.

FIFO queue:
The queue stores data from producer and sends it to Consumer.

Control Unit:
It maintains the FIFO status like FIFO full and empty which controls the read and write operations.

Consumer Module:
Delivers data read from FIFO to external systems or users.

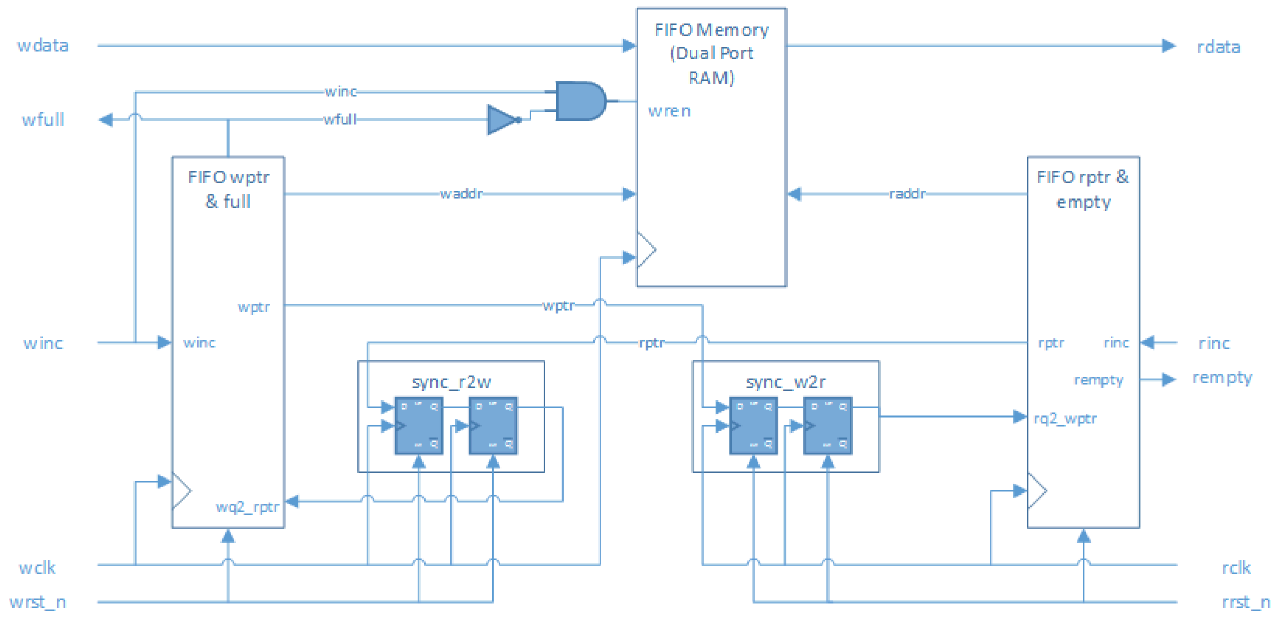
2.2 Product Functions

Producer Module:
Collects data from external sources, such as sensors or input devices. The Input Module is responsible for acquiring raw data inputs from various external sources. It ensures the reliable and accurate reception of data for further processing within the system. This data is then sent to FIFO in bursts of length 1024 at maximum.

Consumer Module:
The data from FIFo is read in burst length of 1024 at maximum with 2 idle cycles in between and transfers to the next modules which will be using the data.

Asynchronous FIFO:
Stores asynchronous data temporarily for synchronized transfer. This module serves as a buffer for storing incoming asynchronous data before it is processed by downstream modules, ensuring proper synchronization between clock domains.

Control Unit:
Stores the required flags to proceed with FIFO operations. When FIFO is full, Producer will not be able to write data to FIFO and when the FIFO is empty, Consumer cannot read from it. Operations happen only based on the status of these flags.



2.3 User Classes and Characteristics

System Administrators:
System administrators play a crucial role in configuring and maintaining the system. They are responsible for managing user accounts, implementing security protocols, and troubleshooting any issues that may arise. System administrators possess advanced skills in system administration and have full access to system settings and configurations.

End Users:
End users are the consumers of processed data, utilizing the system's outputs for decision-making purposes. Their technical expertise varies, but they generally have domain-specific knowledge relevant to their roles. End users have limited access, primarily restricted to viewing and exporting processed data.

Developers:
Developers are involved in the system's development, customization, and integration phases. Proficient in programming languages, system architecture, and software development, developers require full access to the system during the development and testing phases.

Verification Engineers:
Verification engineers contribute to the system development process by designing and executing verification tests. They specialize in ensuring that the system components and features meet specified requirements. Verification engineers collaborate closely with developers during the testing phases.

Validation Engineers:
Validation engineers focus on validating the entire system to ensure that it meets user needs and intended use cases. They are responsible for assessing the overall system performance and functionality in alignment with user requirements. Validation engineers collaborate with various stakeholders to ensure the system's overall quality.

External System Integrators:
External system integrators are responsible for integrating the software system with external systems. Possessing expertise in system integration and data exchange protocols, they have access limited to integration interfaces and configurations.

Regulatory Auditors:
Regulatory auditors conduct audits to ensure compliance with industry regulations and standards. Their role requires an understanding of regulatory requirements, and they have access to compliance-related data and system logs.

Technical Support:
Technical support professionals provide assistance to end users and troubleshoot system issues. Proficient in system functionality and troubleshooting techniques, they have access to user support tools and system logs.

Project Managers:
Project managers oversee project progress, resource allocation, and strategic decision-making. Their skills lie in project management, coordination, and effective communication. Project managers have access to project-related data and progress reports

2.4 Tools and Software

Development tools	: QuestaSim.
Languages	: Verilog, SystemVerilog and UVM.
Version Control system	: Git/GitHub.

2.5 Design and Implementation Constraints

FIFO Depth and Latency:
FIFO cannot be designed to store all the data received without getting full. So Depth of FIFO needs to be limited and is a major constraint which impacts the speed of write operations and in turn the processing speed of Producer.

Power Consumption:
In order to handle metastability and counters the power consumption of the whole fifo system could increase which comes with the trade off with increasing system efficiency.

Memory Constraints:
The available RAM and ROM sizes impose limitations on the size and complexity of the software and data structures.

2.6 Assumptions and Dependencies

Clock Domains:
Assume that the clocks are reasonably stable, and any variations are within acceptable limits for proper operation.

Clock Frequencies and Ratios:
Dependencies may include the requirement that the clocks are within certain frequency ratios to ensure reliable data transfer.

Data Width:
Assuming data widths are consistent to avoid issues related to mismatches.

Control Signal Synchronization:
Dependencies may include the proper functioning of the control logic in different clock domains.

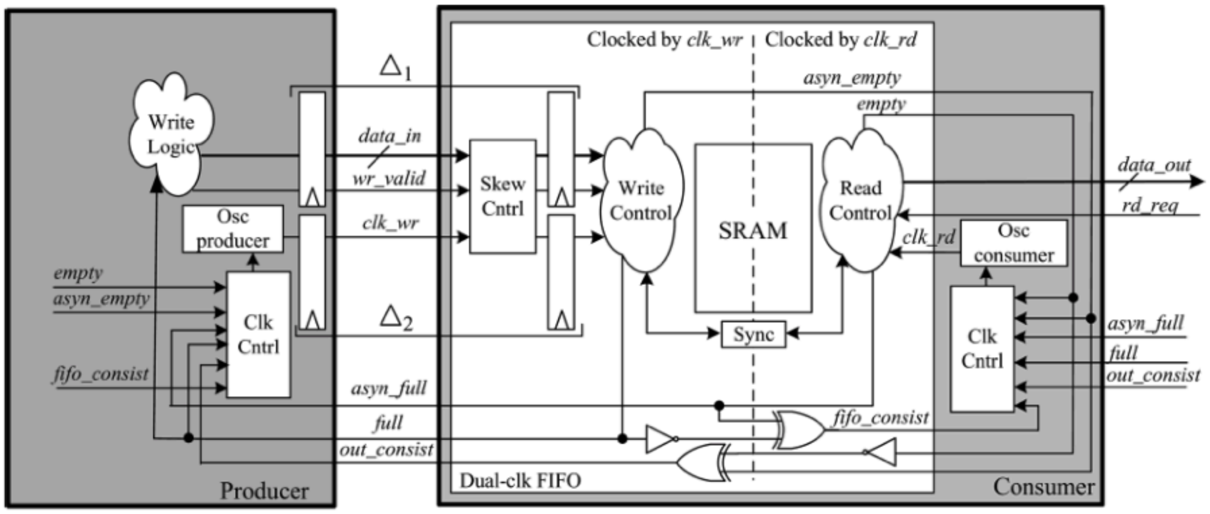
Initialization and Reset Behavior:
On reset or when the system first starts, the fifo is empty, read and write pointer points to the first location of fifo and all of the flags are initialized.

3. External Interface Requirements

3.1 Hardware Interfaces

The system interfaces with external entities, including:

Input Devices: Devices or systems providing data inputs to the Input Module.
Output Devices: Devices or systems receiving processed data from the Output Module.



Producer Module:

The producer module in the FIFO system is responsible for managing the input data stream and efficiently writing data into the FIFO buffer. Its primary objective is to ensure a seamless and controlled flow of data from the source to the FIFO, adhering to the specified data transfer protocols.

Functions:

Data Input:
The producer module accepts incoming data from an external source, which could be a sensor, another module, or an external communication interface.

Data Formatting:
It is equipped to format or preprocess the incoming data, ensuring compatibility with the data format expected by the FIFO.

Write Data to FIFO:
The producer module controls the initiation of write operations to the FIFO. It manages the timing and sequencing of these operations to prevent data collisions and maintain the integrity of the stored data.

Write Enable Signal:
Status signals indicate the readiness of the producer, the occurrence of write operations, or any relevant status updates. The timing and synchronization ensure that the data is correctly captured by the FIFO.

FIFO Module:

The FIFO module within the system is designed to manage the orderly and sequential storage and retrieval of data. It follows the First-In-First-Out principle, ensuring that data written into the FIFO is read out in the same order. The module serves as a crucial buffer for handling data transfer between components that may operate at different rates or in different clock domains.

Functions:

Sequential Data Storage:
The FIFO module stores incoming data in a sequential manner, preserving the order in which it was written. This ensures that the oldest data is read out first.

Write and Read Pointers:
The module employs write and read pointers to keep track of the current write and read positions within the FIFO buffer. These pointers determine where the next write or read operation will occur.

FIFO Depth:
FIFO depth is 456 as calculated in section 4.2. It means that the fifo can store 456 data chunks at any particular point of time.

Clock Domain Synchronization:
To handle scenarios where the FIFO interacts with components in different clock domains, the module incorporates proper synchronizers. These synchronizers mitigate risks associated with clock domain crossings. Details of synchronizers are discussed in section 4.4.

Control Signals:
The FIFO module utilizes control signals, such as write enable (WE) and read enable (RE), to coordinate data transfer operations. These signals facilitate the initiation of write and read operations. Control signal details are discussed in section 4.5.

Full and Empty Flags:
The module maintains flags indicating whether the FIFO is full or empty. These flags assist in flow control and prevent potential overflow or underflow situations. Details in section 4.5.

Control Unit:
The FIFO Control Unit serves as the central coordinating entity for the FIFO module, managing the orderly flow of data into and out of the buffer. It is designed to ensure synchronized and error-free communication between the producer and consumer modules while maintaining the integrity of the stored data.

FIFO Consumer Module:

The FIFO Consumer Module is designed to efficiently retrieve and process data from the FIFO buffer while maintaining synchronization with the FIFO Control Unit and the overall system. Its primary function is to ensure the orderly consumption of data based on the First-In-First-Out principle.

Functions:

Read Operations:
The consumer module initiates read operations to retrieve data from the FIFO buffer. It adheres to the sequential order of data storage, reading the oldest data first.

Data Processing:
Processes the retrieved data according to the intended functionality. This may involve data analysis, transformation, or further transmission to downstream components.

3.2 Software Interfaces

Producer Interface:
This interface gets the input data from external interfaces like sensors memory etc to which it is interfaced. This module is interfaced with FIFO module in the RTL design.

FIFO Interface:
This RTL module gets data input from producer interface which is a stream of data with 1024 burst length along with we signal. Data is accepted and written to FIFO only if the FIFO is not full. It also interfaces with consumer module which reads data from fifo when it's not empty.

Consumer Interface:
This module interfaces with FIFO and the external system which consumes the data which is red from the fifo.

Testbenches:
Testbenches are written for each individual module to test the functionality along with a test bench which tests the overall functionality of the product.

The producer interface test bench acts as the external interface providing the input data and checking the data which is sent to fifo and the handshaking signal.
For the fifo interface the independent test range provides the right data along with the handshaking signal and checks the read data which is available for read along with the read enable signal.
For the Consumer interface individual test bench provides the data to be read along with the read enable signal and checks the working of the module.

The overall test bench interfaces with the producer interface as an input sensor providing required data and also interfaces with the consumer module testing the data which is read from fifo.

4. Product Features

4.1 Clock Domains

There are two different clock domains for Producer and Consumer.
Producer is responsible for writing data into FIFO and operates at a frequency of 120 MHz and no idle cycles.
Consumer reads data from FIFO and operates at 200 MHz frequency and has 2 idle cycles for read.
All the interfaces work at positive edge of clock.

4.2 FIFO Memory and depth

A FIFO (First-In-First-Out) memory buffer is a storage structure that follows the FIFO principle, where the first data item that enters the buffer is the first one to be retrieved. It acts as a temporary data storage mechanism, commonly used in digital systems to manage the flow of data between two asynchronous processes or components.

FIFO depth Calculation:

Writing frequency = fA = 120 MHz.
Reading Frequency = fB = 200MHz.
Burst Length = No. of data items to be transferred = 1024.
No. of idle cycles between two successive writes is = 0.
No. of idle cycles between two successive reads is = 2.

The number of idle cycles between two consecutive writes is zero clock cycles. This indicates that the producer continuously writes data to the FIFO. Therefore, it can be inferred that for every clock cycle, one piece of data is written.

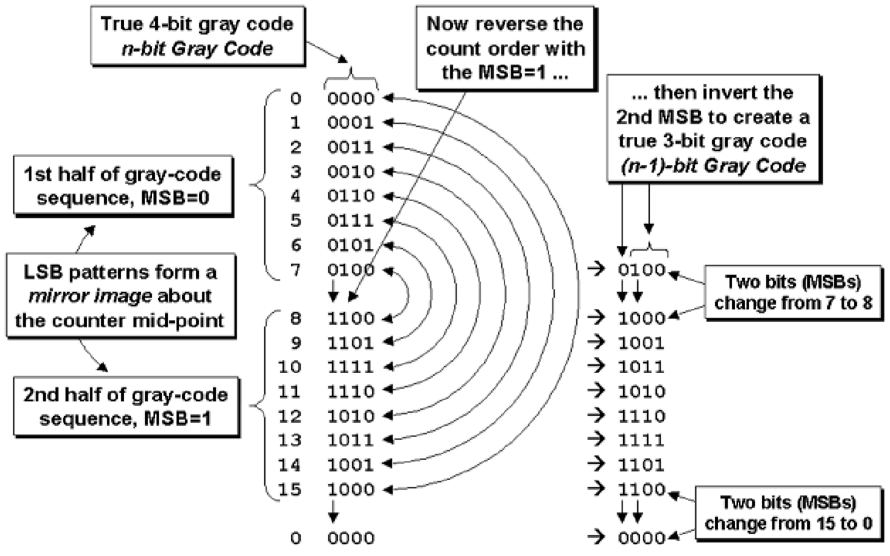
The number of idle cycles between two consecutive reads is two clock cycles. This implies that after reading one piece of data, the consumer waits for 2 clock cycles before initiating the next read. Consequently, it can be inferred that for every three clock cycles, one piece of data is read.

Time required to write one data item = 1/120M = 8.33 nSec.
Time required to write all the data in the burst = 1024 * 8.33 nSec. ~= 8530 nSec.
Time required to read one data item = 3 * 1/200MHz = 15 nSec.
So, for every 15 nSec, the Consumer is going to read one data item in the burst.
So, in a period of 8530 nSec, 8530/15 = 568.66 ~= 568 no. of data items can be read.
The remaining no. of bytes to be stored in the FIFO = 1024 - 568 = 456.
So, the FIFO must be capable of storing 456 data items.
Depth of FIFO = 456.

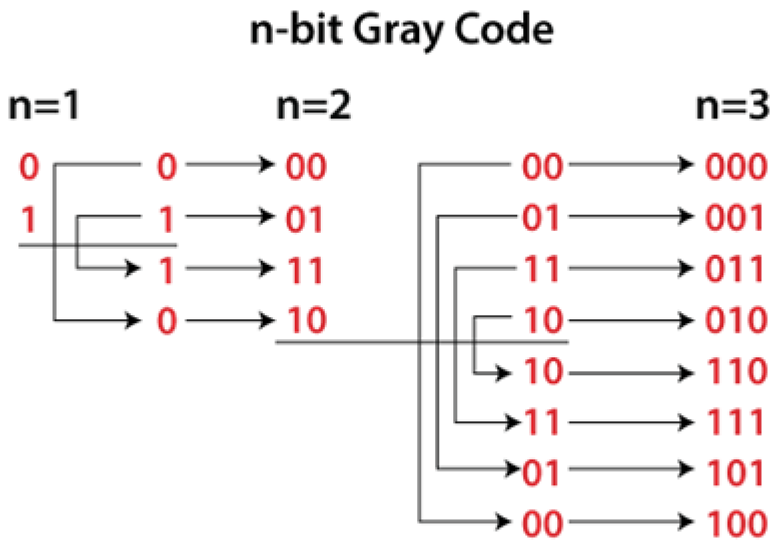
4.3 Counters for pointers

We have 2 pointers - one for read and one for write and they have to point to each location in the FIFO buffer. So length of these pointers will be 9 bits as 8 bits can only count till 255.
Here, a Gray code counter might be employed as a means of generating addresses or pointers within the buffer. As gray code implements one bit change for every increment it has immense advantage over using binary code for counters. A few advantages include reducing the risk of metastability over multiple bits, reduction in power consumption, reducing glitches and stable addressing.

4 bit Gray code sequence is shown below:


True 4-bit gray code
n-bit Gray Code
Now reverse the count order with the MSB=1 ...
... then invert the 2nd MSB to create a true 3-bit gray code
(n-1)-bit Gray Code
1st half of gray-code sequence, MSB=0
LSB patterns form a mirror image about the counter mid-point
2nd half of gray-code sequence, MSB=1
Two bits (MSBs) change from 7 to 8
Two bits (MSBs) change from 15 to 0
0 0000
1 0001
2 0011
3 0010
4 0110
5 0111
6 0101
7 0100
8 1100
9 1101
10 1111
11 1110
12 1010
13 1011
14 1001
15 1000
0 0000
0100
1000
1001
1011
1010
1110
1111
1101
1100
0000

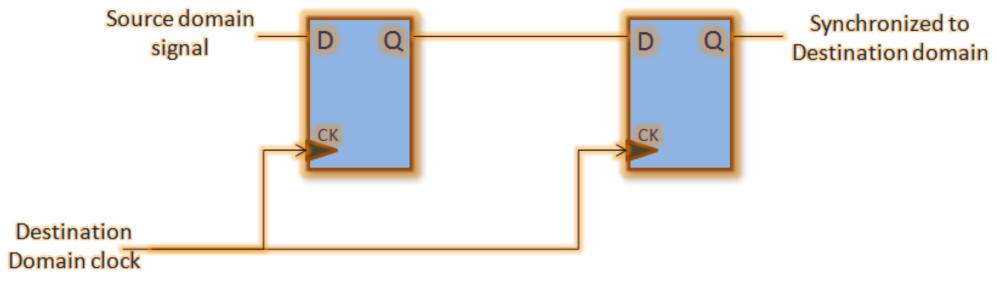
Generation of gray code for n-bit:



4.4 Synchronizers

Synchronizers play a crucial role in mitigating the risk of metastability, especially in systems involving multiple asynchronous clock domains. When the FIFO is empty and a write operation occurs, the update of the read pointer may coincide with a clock trigger, introducing the possibility of metastability. Similarly, when the FIFO is full and a read operation is executed, there is a risk of metastability.

To address these concerns, two synchronizers have been incorporated—one dedicated to read operations and the other to write operations. Both synchronizers adopt a one-level design, utilizing single D flip-flops that operate on the clocks of the producer and consumer. These synchronizers serve to reduce the likelihood of metastability during read and write operations in the system.



4.5 Control Unit

Control unit is responsible to maintain the status of FIFO and make sure the operations are performed without any glitches. It implements two flags to maintain the status of FIFO.

FIFO Flags:
empty - indicates whether the FIFO is empty.
full - indicates whether the FIFO is full.

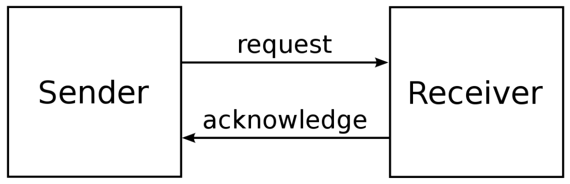
empty flag:
empty flag will be generated in the read-clock domain to ensure that the empty flag is detected immediately when the FIFO buffer is empty, that is, the instant that the read pointer catches up to the write pointer.

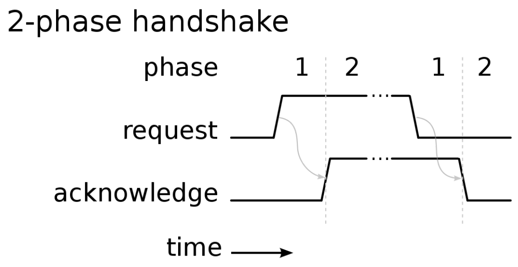
full flag:
full flag will be generated in the write-clock domain to ensure that the full flag is detected immediately when the FIFO buffer is full, that is, the instant that the write pointer catches up to the read pointer.

4.6 Handshake Protocol

A handshaking protocol is crucial for coordinating and synchronizing communication between different components, especially in scenarios where asynchronous data transfer occurs.

The protocol used here is Half Interlocked Asynchronous handshaking Protocol in which the sender sends data and a data ready signal till an acknowledgement is received from the receiver. Once acknowledgement is received, new data is sent again with a data ready signal.





Signals:
‘we’ is asserted by the producer to indicate that valid data is present.
‘wa’ is asserted by FIFO that data is written to FIFO and write pointer points to next location.
‘re’ is asserted by the FIFO that next location data is available on the port to read.
‘wa’ is asserted by consumer that data is read from FIFO and read pointer points to next location.

Write Operation:
The write operation occurs when we is asserted and full is de-asserted. Data (data_in) is written into the FIFO.

Read Operation:
The read operation occurs when re is asserted and empty is de-asserted. Data (data_out) is read from the FIFO.

5. Logic Design

5.1 <Your work Directory Structure>

<How is your design and simulation repository, where to find what?>

5.2 <Design modules>

<what will be your hardware coding style, how many modules, how many, how they will be connected to each other, how will they interact with each other and to the outer world>

5.3 <SystemVerilog abstraction Features used>

<List all the SystemVerilog features you are going to use to take advantage of abstraction, parametrization, scaling your design, etc.>

5.4 <Simulation, Tools, Directory Structure>

<What simulation tools used, what aspects will be shown transcripts, waveforms etc.>

6. Verification

<Describe how you plan to verify your design.>

6.1 Testbench Style

6.2 Testing Strategies

6.3 Test case scenarios

6.4 Others

Summary

Appendix A: Glossary

<Nomenclature used in your document.>