High Level Design Specification (HLDS)

for

Asynchronous FIFO

Version: 1.0
Last date of revision: 01/15/2024

Prepared by *SAI SUKITHA PULI*

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

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Sukitha	1/14/2024	Update the design using depth calculation	1.1

1. Introduction

1.1 Purpose

Asynchronous First-In-First-Out (FIFO) are commonly employed for communication between two systems where they have different cock domains or operate in different rates which makes communication easy. Since write and read clocks are not synchronized, it is referred to as asynchronous FIFO. It serves as a buffering mechanism to manage data between producer and consumer systems. Usually, these are used in systems where data needs to pass from one clock domain to another which is generally termed as 'clock domain crossing'. Thus, asynchronous FIFO helps to synchronize data flow between two systems working on different clocks.

1.2 Document Conventions

⇒ The document follows the same font and size for descriptions: Arial-11.
⇒ Italics for any special vocabulary in the sentences.
⇒ All the section headings and sub-headings are Bold Arial font with size 18 and 14 respectively.

1.3 Intended Audience and Reading Suggestions

The HDLS is for industry use for Asynchronous FIFO. It is also very useful for the ones who want to explore more about communication interfaces between two different operating digital systems.

1.4 Product Scope

As Discussed in the above section the FIFO design is specialized in facilitating seamless and *reliable* data transfer between components operating with different clock frequencies.

The scope of Asynchronous FIFOs is mainly in Data Synchronization, where the data loss is prevented between different clock rate communicating systems. They also maintain *reliability and controllability* at the receiver end too. They are very good use in various digital systems used as interfaces. They maintain separate read and write interfaces for data input and output and have status of full/empty operational conditions.

By talking about Asynchronous FIFO, the scope extends to various digital systems in current semiconductor technology for Flow Control, Buffering, Status Indicators and *Data Integrity* which prevents issues like *metastability* to ensure reliable capture of data at the receiving end. And they also seamlessly fit into larger digital systems.

Within the broader context of a system, the Asynchronous FIFO design fits into the communication and data exchange infrastructure. It acts as a bridge between different subsystems, allowing them to communicate seamlessly. Integration points may include connecting the FIFO to the CPU for data exchange, interfacing with network chips for external communication, and being part of the overall SoC architecture to ensure cohesive functionality.

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
- https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/
- Article by Jason Yu: Dual-Clock Asynchronous FIFO in SystemVerilog.
- https://zipcpu.com/blog/2018/07/06/afifo.html
- https://www.verilogpro.com/asynchronous-fifo-design/

2. Overall Description

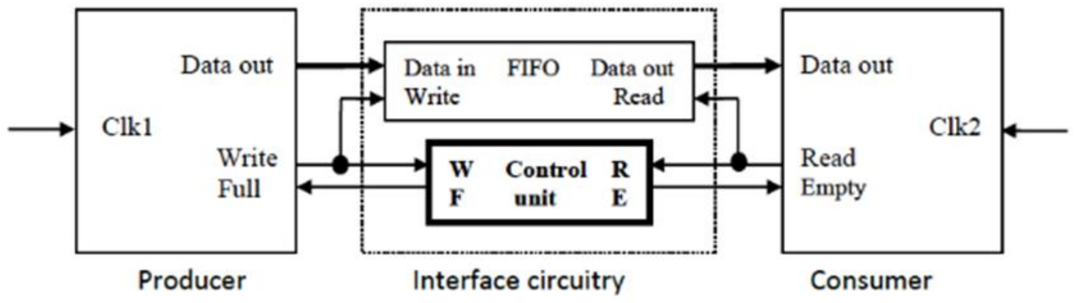
2.1 Product Perspective

The concept of asynchronous FIFO (First-In-First-Out) has been a fundamental part of digital system design for many years. The origin of asynchronous FIFOs can be traced back to the need for handling data transfer between components operating at different clock domains. The specific origins are rooted in digital system design practices, where engineers and designers encountered challenges related to asynchronous communication.

Asynchronous FIFOs have been developed to address issues such as data loss, corruption, and synchronization problems that arise when transferring data between components with independent clocks. The concept and implementation of asynchronous FIFOs have evolved over time to meet the increasing complexity and demands of digital systems, including those in communication interfaces, networking, and other applications.

2.2 Product Functions

The major functions of asynchronous FIFO are to maintain and transfer data between two systems, where it should be able to read and write data from sender and receiver systems.



2.3 User Classes and Characteristics

The product users will mostly be from digitals systems, hardware engineers, semiconductor developers or embedded system engineers where two devices need to be communicating.

Its characteristics, such as efficient synchronization between different clock domains/frequencies, buffering capabilities to manage varying data rates, and compatibility with diverse digital systems, make it valuable for integration into a wide range of designs.

2.4 Tools and Software

It operates within digital systems, it is hardware-skeptic, compatible with various architectures. FIFOs integrate with different environments such as SOC's, CPU's or any networking chips.

It is independent of operating systems and can co-exist normally with different software components or applications within a digital system with flexibility and adaptable to various environments.

Tool: *Questasim*
HDL: *SystemVerilog and UVM*

2.5 Design and Implementation Constraints

- The users should know about the FIFO depth: the larger the product, the larger the area consumption.
- All the data widths and depth should align with specifications with the system. And make sure about the metastability. Here, to avoid metastability we have considered 50% dduty cycle.
- There are few design and implementation constraints such as increased latency due to buffering mechanism which impacts real-time applications.
- Maintain proper data integration techniques.
- Dependencies on Clock signal, error in fifo buffer and i/p or o/p accuracy.

2.6 Assumptions and Dependencies

- Assuming to have gray code counters where there is change in only single bit as multiple signals trigger in same clock edge.
- The FIFO is Positive edge triggered.
- FIFO is full when both write and read pointers have same address except the MSBs, and empty when both pointers are same.
- Below are some assumptions of our Asynchronous FIFO design

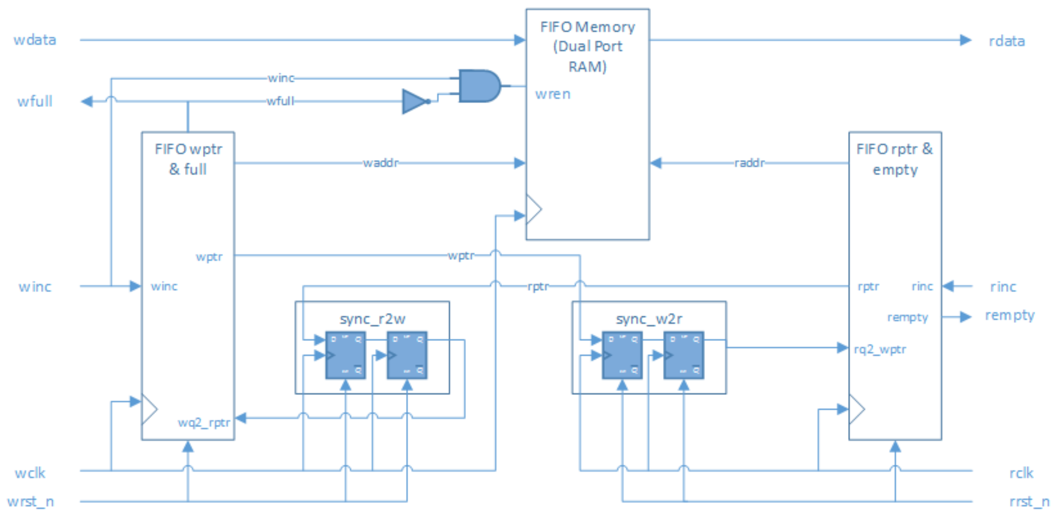
  Clock rates: Producer Clock Frequency = 1Ghz
  Consumer Clock Frequency = 500Mhz
  Duty cycle and Burst sizes are 50% and 200 respected
  And, No. of ideal cycles between successive writes = 2
  No. of ideal cycles between successive writes = 4

3. External Interface Requirements

3.1 Hardware Interfaces

The design is partitioned into the following modules.

- fifo – top level wrapper module
- fifomemory – the FIFO memory buffer that is accessed by the write and read clock domains
- synchronizer_r2w – 2 flip-flop synchronizers to synchronize read pointer to write clock domain
- synchronizer_w2r – 2 flip-flop synchronizers to synchronize write pointer to read clock domain
- readptr_empty – synchronous logic in the read clock domain to generate FIFO empty condition
- writeptr_full – synchronous logic in the write clock domain to generate FIFO full condition



3.2 Software Interfaces

The Asynchronous FIFO (First-In-First-Out) primarily interacts at the hardware level within digital systems and does not directly rely on specific software components. However, its integration may involve interactions with hardware description languages (HDLs) such as SystemVerilog, as well as synthesis tools compatible with these languages such as Questasim.

4. Product Features

All the Product Features will be updated in next versions while implementing the design and verifying it.

4.1 Product Feature 1: FIFO Memory

Here are the Depth Calculations of the Asynchronous design to be calculated for our FIFO memory.

Depth Calculation:

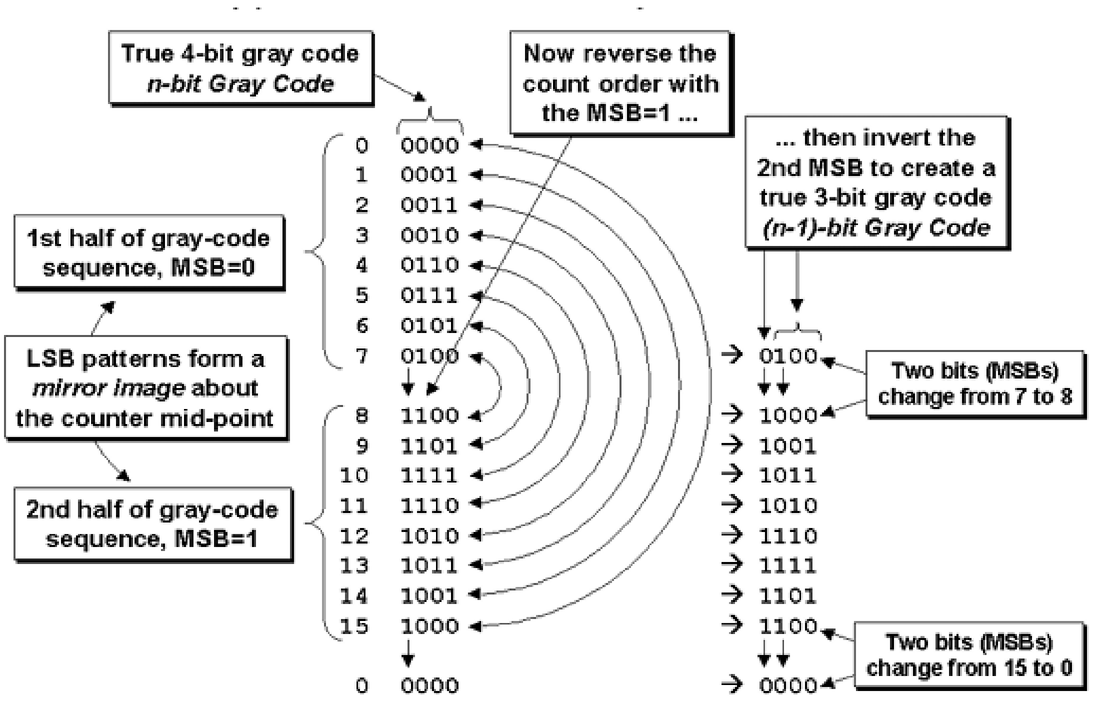
⇒ Time required to write 1 data = 3 × 1/1Ghz = 3 ns
⇒ Time required to read 1 data = 5 × 1/500Mhz = 10ns
⇒ Time to write all data in burst = 200 × 3ns = 600ns
For every 10ns, consumer reads 1 data from burst
For every 600ns, 200 data items are written as per our burst size

⇒ No. of data items can be read in 600ns = 600/10 = 60
⇒ The remaining no. of bytes to be stored in FIFO = 200- 60 = 120
Therefore, Minimum depth according to our assumptions/design should be **120.**

4.2 Product Feature 2: Top- Level RTL Interface Signals

Signal	**Description**
SIZE	PARAMETER number of entries in FIFO
Clock_P	Producer Clock rate
Clock_C	Consumer Clock rate
Reset	Reset, active high
FIFO_Full	Signal when the FIFO is full
FIFO_AFull	Signal when FIFO is almost full
FIFO_Empty	Signal when FIFO is almost empty
FIFO_AEmpty	Signal when the FIFO is empty
Read Enable	Delete an element from the FIFO (increment read pointer)
Read Data	Entry consumed from deleting an element from the FIFO
Write Enable	Push an entry onto the FIFO (increment write pointer)
Write Data	Entry to add into the FIFO
Write Clear	Undo last written entry (decrement write pointer)

4.3 Product Feature 3: Gray Code counter:

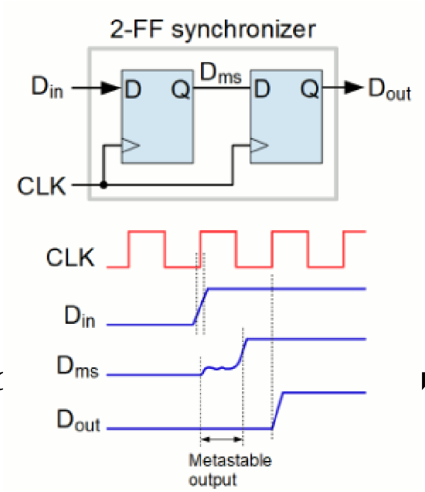

True 4-bit gray code
n-bit Gray Code
Now reverse the count order with the MSB=1 ...
... then invert the 2nd MSB to create a true 3-bit gray code
(n-1)-bit Gray Code
1st half of gray-code sequence, MSB=0
LSB patterns form a mirror image about the counter mid-point
2nd half of gray-code sequence, MSB=1
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

A Gray counter in Asynchronous FIFO (First-In-First-Out) system is a specific type of counter employed to address and manage the storage locations within the FIFO structure. Unlike binary counters where adjacent values have multiple bit transitions, gray counters ensure that only **one-bit** changes at a time between consecutive values which reduces metastability of many bits when compared to conventional binary counter. This characteristic minimizes the potential for errors during transitions, making gray counters particularly suitable for asynchronous designs where signal transitions may not occur simultaneously.

So, the design choice of using a Gray counter aligns with the goal of reducing the likelihood of metastability issues that can arise in asynchronous systems to ensure accurate addressing of memory locations and smooth data flow within the FIFO structure.

4.4 Product Feature 4: Read and Write Synchronizers

As the clock domains for both consumer and producer are different, the design need some



synchronizers for cross domain clocking. The above considered gray counter does not sample all the values for full and empty conditions which requires synchronizers.
Here, we are considering 2 flip flop synchronizers to provide some synchronization between two different clock signals.

4.5 Product Feature 5: FIFO Full and Empty Conditions.

In asynchronous FIFO systems, the coordination between read and write pointers is crucial for maintaining data integrity, especially when different clock domains are involved. Careful management of these pointers, often facilitated by specialized counter mechanisms like Gray counters, helps prevent issues such as data collisions and ensures smooth and reliable data transfer within the FIFO structure. Efficient control of read and write pointers contributes to the overall effectiveness of the FIFO in managing asynchronous data flow between different components of a system.

FIFO Full:
Raise a full flag or signal when the number of stored elements reaches the maximum capacity of the FIFO. Prevent data overflow by signaling that the FIFO cannot accept more input until some data is read or removed.

FIFO Empty:
Raise an empty flag or signal when there are no elements in the FIFO. Ensure that valid data is available for retrieval, preventing reading from an empty FIFO.

FIFO Almost Full:
Set a threshold level below the maximum capacity and raise a corresponding flag or signal when the number of entries approaches this threshold. Allow the system to take preventive actions or optimizations when the FIFO is close to being full, avoiding potential overflow issues.

FIFO Almost Empty:
Set a threshold level above zero and raise a flag or signal when the number of entries approaches this threshold from the higher end. Enable the system to initiate actions or optimizations when the FIFO is close to being empty, ensuring a continuous flow of data.

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.>