High Level Design Specification (HLDS)

for

ASYNCHRONOUS FIFO

Smit Patel,
Swaroop Chandrashekar,
Shashi Kirana Chinchandlahalli Ravanappa,
Gautam Savaliya,

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

20th April 2024

Table of Contents

1. Introduction

1.1 Purpose

The document's goal is to define the specifications for an Asynchronous FIFO (First-In-First-Out) memory queue. This Asynchronous FIFO is intended to transport data across multiple clock domains in a secure manner, which is a typical requirement in digital systems with components that run at different clock frequencies.

1.2 Document Conventions

This sections is to be revised after completing the Section 5 & 6.

1.3 Intended Audience and Reading Suggestions

The document on Asynchronous FIFO Design is primarily designed for hardware design engineers, digital logic designers, FPGA/ASIC designers, system integrators, and quality assurance specialists in the field of electronics. It is also relevant for professors and students focusing in electronics and digital system design. Those having a background in digital systems and a grasp of hardware design principles would gain the most from this reading. The document's technical aspects, notably its applicability in controlling data transfer between multiple time domains, should be addressed with caution.

1.4 Product Scope

The document's scope includes the Asynchronous FIFO's design considerations, functionality, and implementation details. This involves handling full and empty circumstances, using Gray code counters for pointer management, and preserving data integrity during asynchronous clock domain transfers.

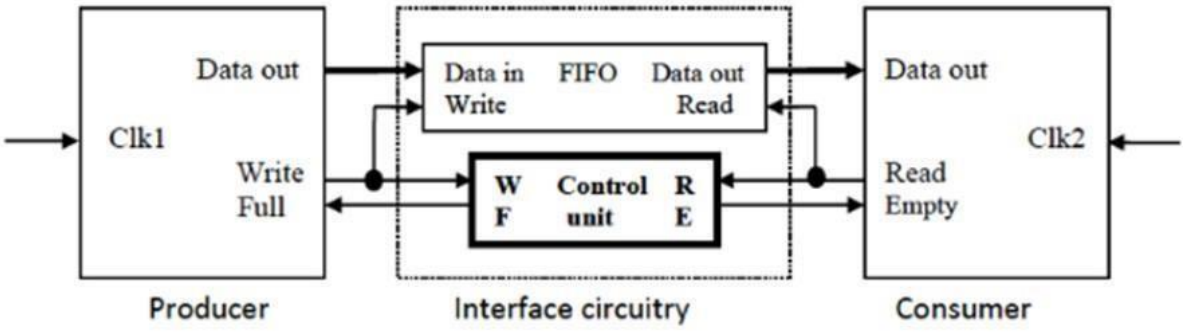
1.5 References

- 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

2. Overall Description

2.1 Product Perspective

The asynchronous FIFO (First In, First Out) buffer is a specialized buffer that is used for data storage and retrieval between two subsystems that operate on different clock domains. It is an important component inside a larger system for handling timing discrepancies and data flow across these subsystems. This product is critical in systems where the producer and consumer modules function independently, ensuring data integrity without the need for operational clock synchronization.



HLDS Figure for ASYNC FIFO

The asynchronous FIFO is portrayed as a major component in this High-Level Design Specification (HLDS), connecting with two independent subsystems: the data producer and data consumer. Each subsystem works on its own clock domain, with the FIFO serving as a data transmission intermediary. The HLDS diagram would normally show the FIFO connected to both subsystems, with arrows indicating data flow direction. Control signals for write and read operations are also shown, demonstrating how the FIFO regulates data queuing and retrieval procedures.

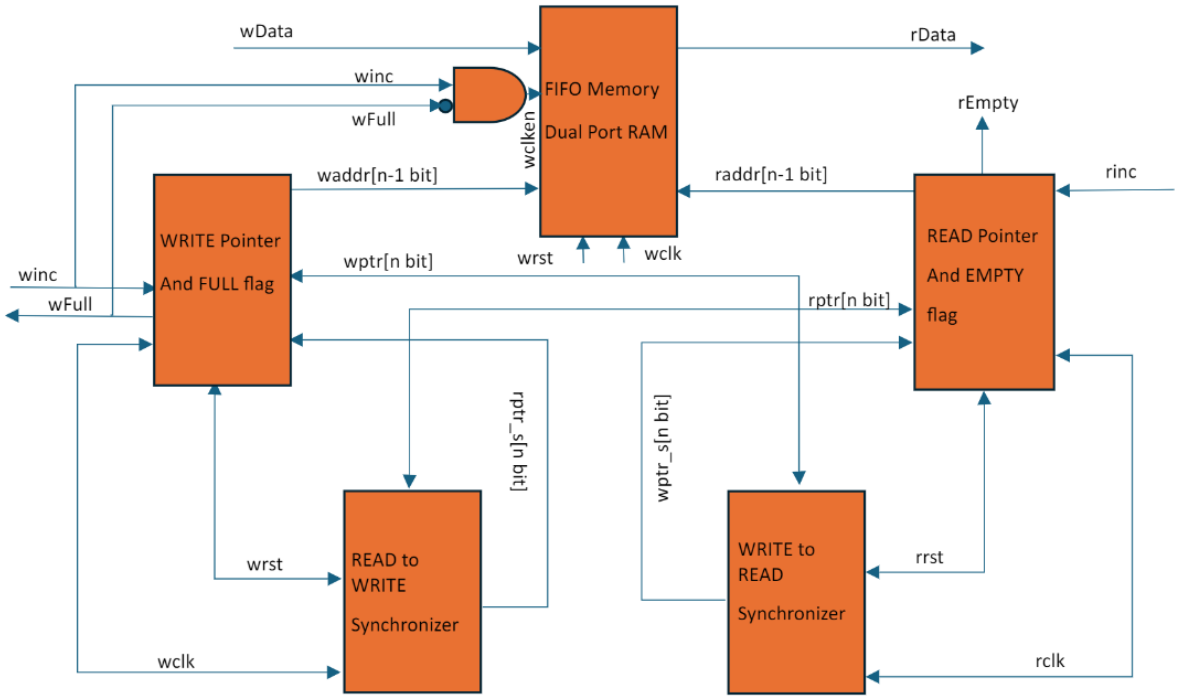
This component is essential for applications including as telecommunications, computer architecture, and digital signal processing, where varied operational speeds between system components are prevalent. The asynchronous FIFO enables error-free and seamless data transfer, making it a critical component of the entire system's efficiency and reliability.

2.2 Product Functions

The functions of the Asynchronous FIFO are as follows

- ***Transferring Data Between Clock Domains:*** *The FIFO is used to safely transfer data between asynchronous clock domains.*
- ***Handling Full, Empty States, Half Full and Half Empty:*** *Developing logic to detect and manage FIFO full and empty states along with half empty and half full.*
- ***Implementation of Gray Code Counters:*** *Using Gray code counters for pointer creation to assure safe data synchronization.*
- ***Managing FIFO Pointers in Synchronous and Asynchronous contexts:*** *Managing FIFO pointers, including the complexity involved in asynchronous contexts.*
- ***Safe Data Synchronization:*** *Maintaining data integrity when moving across clock domains.*
- ***Scalability and Flexibility:*** *Ensuring that the FIFO architecture can be scaled and configured to multiple sizes and configurations.*

The block Diagram of the FIFO given below:



Detailed Block diagram of Asynchronous FIFO

The signals description :

Inputs: *wdata, winc, wclk, wrst_n, rinc, rclk, rrst_n*
Outputs: *rdata, wfull, rempty*
Internal Signals: *waddr, raddr, wptr, rptr, wq2_rptr, rq2_wptr*

2.3 User Classes and Characteristics

- ***Hardware Design Engineers:*** *Individuals who work in the design and development of electronic systems. They would find extensive technical explanations of FIFO (First-In-First-Out) design principles, particularly in asynchronous contexts, quite useful.*
- ***Digital Logic Designers:*** *Users who specialize in designing and implementing logic circuits. They would be interested in the document's emphasis on Gray code counters & pointer management in asynchronous FIFO systems.*
- ***FPGA/ASIC Designers:*** *Designers of field-programmable gate arrays and application- specific integrated circuits will benefit from the document's FIFO design insights, particularly for high-reliability applications.*
- ***Electronics Students:*** *Individuals engaged in academic study or learning about digital system design. The paper will be an excellent learning resource since it provides insights and useful design principles in asynchronous FIFO architecture.*
- ***System Integrators:*** *Individuals who are in charge of integrating numerous hardware components into larger systems. The focus of the document on safe data synchronization across different clock domains is especially related to their work.*

2.4 Tools and Software

- ***QuestSim:*** *This is the primary simulation tool.*
- ***Accellera UVM:*** *We are using the classes from accellera UVM standards.*

2.5 Design and Implementation Constraints

The FIFO was designed with the assumption that the producer works at 120 MHz and the consumer at 50 MHz, ensuring that data is given and consumed at distinct rates. Furthermore, a maximum write burst size of 1024 is established, with two idle cycles between consecutive reads and three idle cycles between writes, influencing overall data flow and buffer management.

- *Sample design for the above Asynchronous FIFO used as an interface circuitry.*
- *Producer Clk frequency (clk 1): 120 MHz*
- *Consumer Clk frequency (clk 2): 50 MHz*
- *Maximum write burst size: 1024.*
- *Duty cycle: 50%*
- *No. of Idle cycle between successive writes = 3*
- *No. of Idle cycle between successive reads = 2*
- *No. of idle cycles between successive writes is 3 clock cycles. It means that after writing one*
- *data, the Producer block is waiting for 3 clock cycles to initiate the next write. So, it can be*
- *understood that for every 4 clock cycles, one data is written.*
- *The no. of idle cycles between two successive reads is 2 clock cycles, it means that after reading*
- *one data, the consumer block is waiting for 2 clock cycles to initiate the next read. So, it can be*
- *understood that for every 3 clock cycles, one data is read.*
- *Time required to write one data item: 4 x (1/120MHz) = 33nsec*
- *Time required to write all the data in the burst = 1024 x 33nsec= 34099nsec*
- *Time required to read one data item = 3 x (1/50MHz) = 60nsec*
- *So, for every 60 n, the consumer block is going to read one data in the burst. So, in a period of*
- *34099nsec, a maximum 1024 number of data items can be written.*
- *Therefore, the number of data items can be read in a period of 34099nsec =*
- *34099nsec/60nsec=568*
- *The remaining number of bytes to be stored in FIFO = 1024-568 = 456 so a depth of 9 is*
- *required.*

Timing Diagram of Asynchronous FIFO:

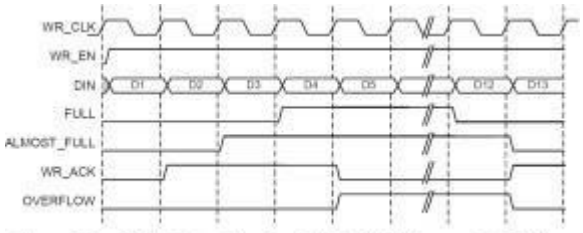


Figure 5-6: **Write Operation for a FIFO with Independent Clocks**

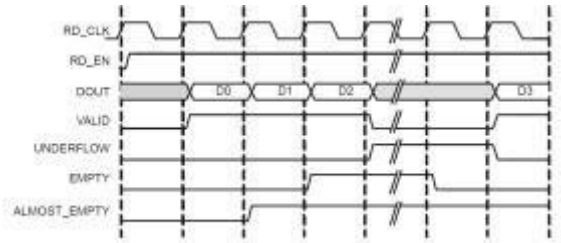


Figure 5-7: **Standard Read Operation for a FIFO with Independent Clocks**

3. External Interface Requirements

3.1 Hardware Interfaces

- ***Memory Module:*** *This module models the storage part of the FIFO. For verification purpose we will be using the SystemVerilog Queue construct. (This is not synthesizable.)*

- ***Control Unit:*** *This module controls the Memory pointers using counters, and other peripheral signals such as memory full, memory empty, etc.,*

- ***Working of Gray code counter:*** *Gray code counters are a significant improvement in digital design because they provide a unique approach for eliminating errors in binary counting systems. Their key advantage is that only one bit changes at a time, reducing the possibility of error during the transition between states.This technology is widely used in digital communication and computation, especially where precision and error reduction are critical. Gray code counters are designed in a variety of designs, each adapted to certain speed and efficiency requirements. Depending on the desired speed and complexity, these counters can be built with a single or numerous sets of flip-flops. A prominent variation is the dual n-bit Gray code counter, which effectively tackles conversion issues between different bit Gray codes. Furthermore, Gray code counters are required in FIFO buffers to provide smooth data flow in a variety of digital applications. Because of their design flexibility, they may be adapted to a wide range of digital applications, making them a vital tool in modern electronics and computers.*

4. Product Features

4.1 FIFO memory

- *FIFO memory hardware is a form of data buffering mechanism used in computers to handle and temporarily store data. It works on the basis that the first data to enter the queue is the first data to be processed or outputted. Consider a pipe in which items enter at one end and depart at the other in the same order.*
- *This strategy assures a smooth and orderly flow of data, which is especially advantageous in applications where data must be handled in a sequential manner, such as streaming audio or video, telecommunications, and some real-time computer scenarios.*
- *FIFO memory can be implemented in a variety of ways, such as hardware registers in CPUs, dedicated memory chips, or as part of broader memory management systems. One of the most important advantages of FIFO memory is its simplicity, which enables rapid and efficient data handling without the need for sophisticated algorithms to govern data processing order.*
- *However, because of its simplicity, FIFO memory is not ideal for applications where data must be retrieved or processed out of order. Other types of memory buffers, such as stacks or random access memory (RAM), are more appropriate in such cases.*
- *Overall, FIFO memory is vital for preserving data integrity and order in systems where data processing sequence is critical.*

4.2 Gray Counter

- *Because of their unique qualities, gray code counters are especially valuable in First-In- First-Out (FIFO) memory systems. FIFO systems are data structures in which the first element added to the queue is eliminated first. It is critical in such systems to have reliable and efficient means of tracking read and write positions inside the memory buffer. Gray code counters come in handy here*
- *A Gray code is a binary number system in which two consecutive values differ only by one bit. This single-bit change feature is critical in decreasing errors in digital circuits, especially when many bits are updated at the same time. Gray code counters are used to track the read and write pointers in FIFO systems because they considerably reduce the possibility of erroneous data read/write operations caused by the asynchronous update of multiple bits.*
- *For example, in a normal binary counter, changing from '0111' (7 in decimal) to '1000' (8 in decimal) changes all four bits, which might cause timing problems because each bit does not*

change at the same time. A Gray code counter, on the other hand, alters only one bit at a time, resulting in a more dependable and error-resistant function. This single-bit transition is especially useful in high-speed FIFO systems where timing errors might result in data corruption or loss.

- *Gray code counters, in general, improve the integrity and reliability of FIFO systems by assuring smoother and more precise transitions between states, and hence play an important role in digital data management and transmission.*

4.3 FIFO full , Empty Condition , FIFO Half Full & FIFO Half Empty

- *A FIFO's full and empty conditions are critical for its efficient operation. The condition 'full' indicates that the FIFO buffer has reached its maximum storage capacity and cannot receive any additional data until some of the existing data is read out or deleted. This is necessary in order to avoid data overwriting or loss. To keep track of the read and write positions within the FIFO, hardware normally employs a set of pointers or counters, and when the write pointer catches up to the read pointer, the FIFO is considered full.*
- *The 'empty' state, on the other hand, implies that there are no data elements in the FIFO buffer to be read. This occurs when the read pointer overtakes the write pointer. Attempting to read from the FIFO in such a case would result in underflow, where invalid or stale data could be read.*
- *FIFOs frequently include control logic or status flags to address these scenarios. A full flag, for example, is set when the FIFO is full, and an empty flag is set when it is empty. Other hardware components can utilize these flags to determine when to stop writing to or reading from the FIFO. In systems where timing and data flow control are crucial, such as network routers, data buffers in digital signal processing, and inter-processor communication in multi-core systems, such management assures data integrity and smooth operation.*
- *When the number of data entries in the FIFO equals half of its maximum capacity, it is said to be "half full". In the event that the FIFO can hold N entries in total, the half full condition is: ≥N/2= number of entries in FIFO*
  Total entries in FIFO >= 2/N
- *The FIFO is considered "half empty" when it contains data entries equal to half of its maximum capacity.*

4.4 Synchronizers

- *Synchronizers play an important role in the design of asynchronous FIFOs (First-In, First- Out memory queues) that operate across multiple time domains. Data is written into an asynchronous FIFO under one clock domain and read from it under another. Because of the disparity in clock domains, timing concerns like as metastability might occur, when the data signal is stuck in an indeterminate state during the transition time.*
- *Synchronizers are used to mitigate this. A typical synchronizer is a set of connected flip- flops. The first flip-flop captures an asynchronous signal on the receiving clock domain's edge, however this signal could be metastable. The risk of metastability influencing the system is considerably lowered by passing this signal through several flip-flop stages, as each flip-flop stage provides additional time for the signal to settle into a stable state.*
- *The selection and design of synchronizers is critical because they must strike a compromise between lowering metastability hazards, and the delay induced by the synchronizer chain. The number of stages in a synchronizer is frequently determined by a trade-off between the frequency of the clocks involved and the acceptable level of system performance and reliability.*

5. Logic Design

5.1 Your work Directory Structure

- *Design.sv contains System Verilog source files.*
- *Testbench.sv contains testbench files for simulation.*
- *Run.do file to compile and run the simulation.*

5.2 Design Modules

- *Hardware Coding Style: System Verilog is used for RTL engineering*

Modules:

- *The FIFO feature is implemented by the asynchronous FIFO module (design.sv).*
- *Testbench (testbench.sv): This is where you test the asynchronous FIFO module.*
- *Interconnection: The delayed FIFO module can be used in other modules to connect them or test stands. The right signals will be linked for data input, data output, control signals, and time domains.*
- *Interaction: The asynchronous FIFO module has input and output ports that let it talk to outside things like the testbench and other modules. Signals like data in, data out, read enable, write enable, and clock allow it to talk to each other.*

5.3 System Verilog abstraction Feature used

For scale and freedom, parameters are used to set things like FIFO depth, data width, and so on.

5.4 Simulation, Tools, Directory Structure

- *Simulation Tools: Model Sim or any other software that works with it.*

Parts of the simulation:

- *Transcripts: These are log files that show mistakes, messages, and the progress of the exercise.*
- *Waveforms: Made to see how signals and modules interact during modelling. Structure of the directory:*
- *System Verilog code files are in Design.sv.*
- *The file Testbench.sv has testbench files for modelling.*
- *Run.do file to make the exercise work and run it.*

6. Verification

6.1 Testbench Style

- *The Universal Verification Methodology (UVM) framework will be used to write the test bench in System Verilog.*
- *We will use Transaction-Level Modelling (TLM) to hide the details of transactions that happen between the testbench and the asynchronous FIFO module.*

- *Random Stimulus Generation with Limits: To fully test the FIFO functionality, random test cases will be created within certain limits.*

6.2 Testing Strategies

- *Functional Testing: The asynchronous FIFO module's basic functions, such as data storage, recovery, and border conditions, will be checked by test cases.*

- *Performance Testing: The FIFO's performance will be checked by test cases that look at things like speed, delay, and how well resources are used in different situations.*

- *Corner Case Testing: Unique situations and edge conditions will be checked to make sure the design is strong and correct.*

- *Stress Testing: The FIFO will be put through stressful situations to see how it works in high-traffic, overflow, and underflow situations. Test case scenarios*

6.3 Test case scenarios.

- *Write and Read Operations: Test cases will be made to make sure that write and read operations work correctly in a range of clock domains and data lengths.*
- *Full and Empty situations: Test cases will check how the FIFO works when it is full or empty, making sure that these situations are handled.*
- *Boundary Conditions: Test cases will make sure that the FIFO works correctly when it gets close to its capacity limits for overflow/underflow and wrapping.*