DESIGNING AN ASYNCHRONOUS FIFO USING VERILOG

[#1]KUKKA RAJ KUMAR, ***Assistant. Professor,***
[#2]PABBATHI VISHNU, ***Assistant. Professor,***
Department of Electronics Communication Engineering,
SREE CHAITANYA INSTITUTE OF TECHNOLOGICAL SCIENCES, KARIMNAGAR, TS.

ABSTRACT:

The First-In, First-Out (FIFO) method is used to manage computer work requests that originate from stacks or queues, assuring that the request with the earliest arrival time receives processing priority. Using the First-In, First-Out (FIFO) logic, data from one clock domain is transmitted to other clock domains upon request. This task is accomplished in the domain of hardware by a collection of flip-flops or read/write memory components. A more efficient method for constructing a First-In-First-Out (FIFO) system is to compare write and read pointers that are generated in separate clock zones and not simultaneously. The method of comparing pointers in an asynchronous FIFO is used to reduce the number of synchronization flip-flops required to construct the FIFO. In order to effectively construct and evaluate the design using this methodology, it is necessary to employ additional methodologies, as outlined in this academic article. Utilizing mixed binary/gray counters that leverage the inherent binary ripple carry logic is one method employed by this design to improve the efficacy of the First-In-First-Out (FIFO) process.

Key Words: Asynchronous FIFO, FIFO Design, Full andEmptyDeduction

DOI Number: 10.48047/nq.2022.20.19.nq99495 **Neuroquantology 2022; 20(19):5240-5244**

1.INTRODUCTION

One clock domain is utilized to sequentially write data values into a FIFO buffer. To read data values from the same FIFO buffer one after the other, a separate clock domain is used. The times of these two clock domains are not synchronized. An asynchronous FIFO is commonly created by using Gray code pointers that are synced into the opposite time domain before issuing synchronous FIFO full or empty status signals. Comparing the pointers concurrently before setting the full or empty status bits is a fascinating and novel approach of creating FIFO full and empty bits.

Full and Empty Deductions

The most difficult component of any FIFO system is getting the full and empty states to work correctly. As a result, something else must distinguish between full and empty. When the two points are equal, the approach divides the address space into four quadrants and decodes them using the two MSBs of the two counters. This indicates whether the FIFO was filled or emptying.

Because the wptr is one quadrant behind the rptr, FIFO is fully operational. It indicates that the memory is "possibly going full" when the write pointer is one region behind the read pointer, as shown. When the direction button is pressed, this occurs.

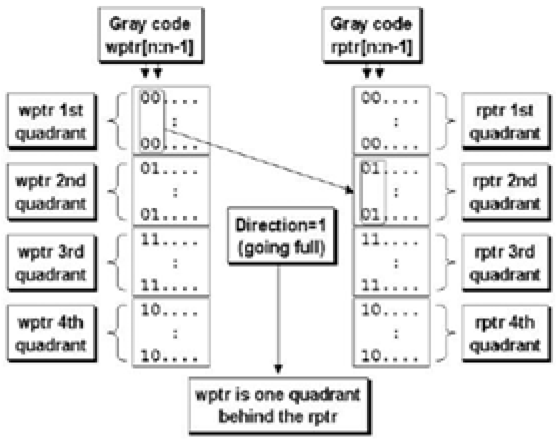


Fig-1:Because the rptr is one quadrant behind the wptr, FIFO runs out of room.

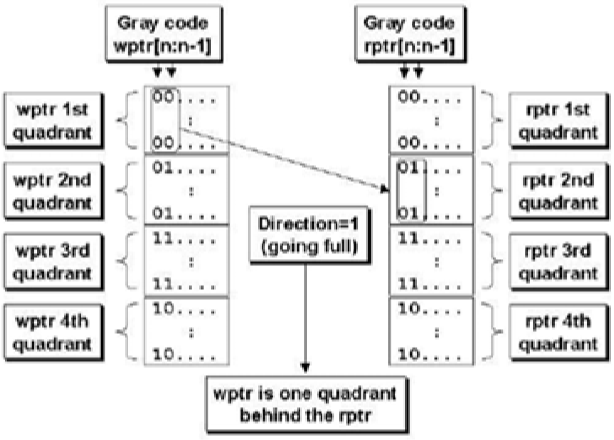


Fig-2:Because the wptr is one quadrant behind the rptr, FIFO is fully operational.

When the write pointer is one quadrant ahead of the read pointer, the status "possibly going empty" is displayed. The direction latch is not locked when this occurs.

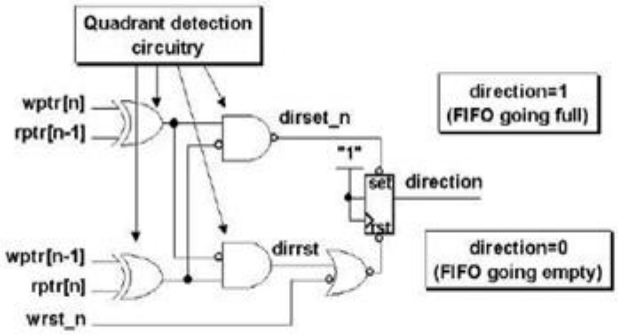


Fig -3: Because the wptr is one quadrant behind the rptr, FIFO is fully operational.

When the FIFO is reset, you may determine it "is going empty" by clearing the direction latch. The direction latch removes the ambiguity in the address identification decoder, therefore setting it up and taking it down is not time-dependent. Because it only requires two 4-input look-up tables, the Xilinx FPGA circuitry for decoding the two wptrMSBs and two rptrMSBs is simple to implement. The second, more challenging issue stems from the fact that the write and read clocks are not in sync. When one or both counters change a large number of bits at about the same moment, comparing them can result in inaccurate decoding leaps. According to the findings of this investigation, only one bit changes from one Gray count sequence to the next. There is no possibility of error during decoding because any decoder or comparator will only move from one excellent output to the next.

FIFO2.v

This is the module that contains all of the clock domains at the highest level. The top module serves as a wrapper for the other FIFO modules required for the architecture. If this FIFO were to be utilized in a larger ASIC or FPGA design, the top-level wrapper would most likely be removed. This would make it easier to group the remaining FIFO modules into their respective clock domains for improved synthesis and static timing analysis.

FIFOmem.v

This FIFO memory buffer is used by both the write and read clock domains. This buffer is most likely a dual-port, synchronous RAM. The FIFO buffer can be utilized with various types of memory.

async_cmp.v

Asynchronous pointer-comparison module signals control the "full" and "empty" state bits, which are set at various times. This section's logic is all combinational comparison logic. This code contains no sequential logic.

rptr_empty.v

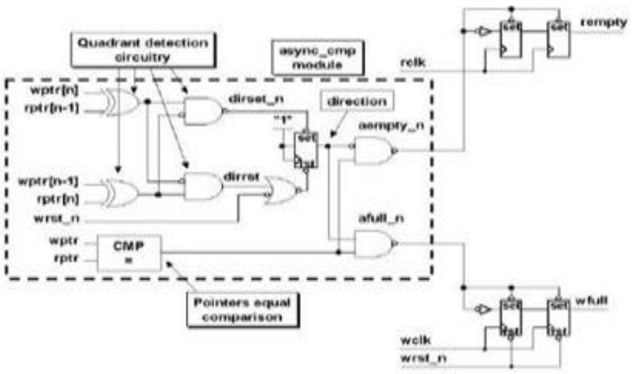
This module contains the FIFO read pointer and empty-flag code, and it is mostly in sync with the read-clock domain. Only when the rptr rises can the aempty_n signal be asserted. When the wptr grows, which is not in sync with rclk but is in sync with the rclk-domain, the signal is de-asserted.

wptr_full.v

This module, which is mainly in sync with the write-clock domain, contains the FIFO write pointer and full-flag logic. Only when wptrincremented and wrst_n occur may afull_n be asserted. This means that asserting the afull_nsignal (an input to this module) occurs concurrently with the wclk domain, but de-asserting it occurs concurrently with the rptrincremented, which is not concurrent with wclk.At Different Times, Empty and Full Production

Asynchronous Generation of Full And Empty

The async_cmp function displays the aempty_n and afull_n asynchronously processed signals. It is de-asserted on the rising edge of a wclk rather than the rising edge of a rclk. Similarly, the afull_n signal is activated for a wclk and deactivated for a rclk. The following read activity will be halted by using the empty signal. The leading edge of the empty signal must be in sync with the read clock, but so must the railing edge. This is accomplished with a two-stage synchronizer that results in r_empty. The symmetrically equivalent approach is used to generate the w_full signal.



nously comparing references to ensure they are both full and empty

Resetting the FIFO

- The first significant FIFO event occurs during the FIFO-reset process. We can see the async_cmp module, the full and empty synchronizers of the wptr_full and rptr_empty modules, and the relationships between these modules. When the FIFO is reset, these modules perform four critical functions.
- The reset signal immediately clears the w_fullflag. Even if you reset, the r_emptyf delay will remain.
- The pointer comparator determines that the pointers are equal because the reset signal clears both FIFO points.
- The direction bit is removed by the reset.
- After the direction bit has been cleared and both values are equal, the empty_n bit is set, which sets the r_emptyflag ahead of time.

Parallel-In, Parallel-Out, Universal ShiftRegister

The parallel-in/parallel-out shift register's function is to take parallel data, shift it, and then send it out. A universal shift register can execute multiple things at once and has a capability known as "parallel-in/parallel-out."

Four bits of data are delivered to a shift register that works with parallel-in and parallel-out at DA DB DC DD. The mode control, which can accept multiple inputs, determines whether to load in parallel or shift. Some real gadgets may also allow you to adjust the way you shift using the mode control. Every clock pulse will cause one bit of data to be transferred. The values that have been altered are in the outputs QA QB QC QD.

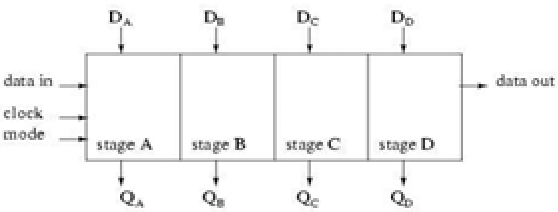


Fig-5:4-Stage Shift Register with Parallel-in and Parallel-Out

The terms "data in" and "data out" are used to cascade several steps. While we investigate this, we can only cascade information for right shifting. To allow the left-shift data flow, two left-pointing signals labeled "data in" and "data out" could be inserted above. The diagram below depicts the components of a right shifting parallel-in/parallel-out shift register. Even though they are not strictly necessary for the parallel-in/parallel-out shift register, the tri-state buffers are included in the device shown below. Because the 74LS395 is so similar to our idea of a perfect right shifting parallel-in/parallel-out shift register, we utilize a very simplified version of the information above from the data sheet. When data enters the FFs, the AND-OR multiplexer is controlled by LD/SH. If LD/SH'=1, which activates the top four AND gates, DA, DB, DC, and DD can be applied to the four FF data inputs at the same time. Keep an eye out for the inverter bubble at the four FFs' clock inputs. This demonstrates that the 74LS395 stores data on the low to high shift's negative going clock. At the following clock down, the four bits of data will be timed in tandem from DA DB DC DD to QA QB QC QD. This "real part" must have OC' set to low so that data can be accessible via the output pins as well as the internal FFs.

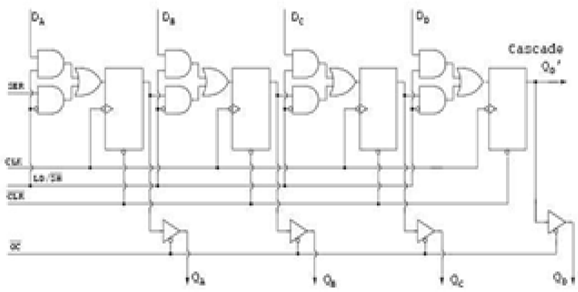


Fig-6:Inside and outside parallelism Taking turns with the tri-state What It Is

If LD/SH'=0 on the next negative clock edge, the previously loaded data can be shifted to the right one bit. The data would leave our 4-bit shift register for good after four clocks. If our device was not cascaded from QD' to SER of another device, the data would be lost.

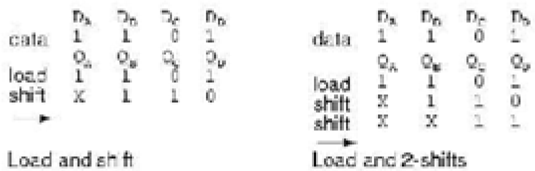


Fig -7: Parallelism both inside and outside the Transfer Register Above is a data pattern for the sources DA, DB, DC, and DD.

The code is forwarded to QA QB QC QD. It is then shifted slightly to the right. The incoming data, represented by the letter X, is unknown to us. If the

eISSN1303-5150

input (SER) were grounded, we would know what data (0) was transferred in. It also illustrates that the right is moving two spaces, which necessitates the use of two clocks.

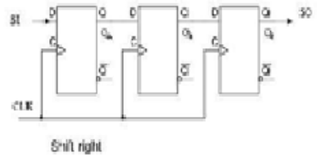
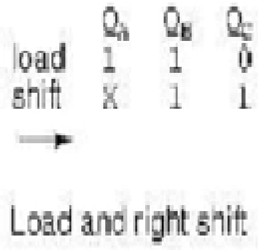


Fig-8:ShiftRight

The graphic above depicts the hardware used to shift data to the right. This figure is too simple to deal with, other than to demonstrate how simple it is in comparison to the figures that will follow.



The data that has been shifted to the right is displayed above to be compared to the previous right-shifter.

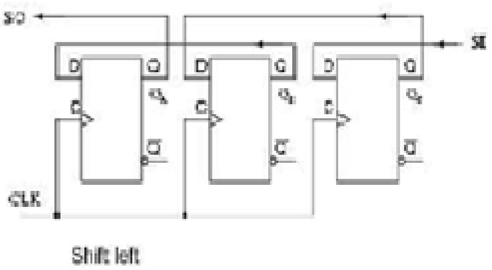
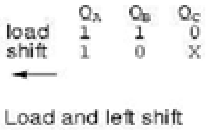


Fig-9:ShiftLeft

To shift left, we must first change the FFs. than the right gear that came before it. SI and SO are also in the wrong places. SI is relocating to QC. Transitioning from QC to QB. Moving from QB to QA. When QA cuts the link to SO, another shifter SI may be affected.



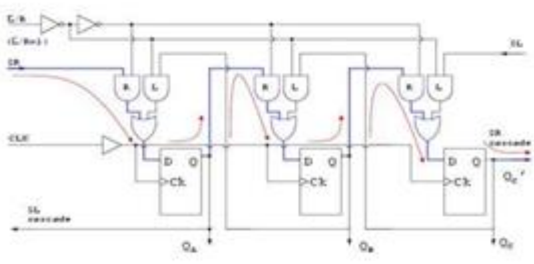


Fig-10:Left or right shift, Right Action

The letters L' and R can be used to move the imaginary shift register illustrated above in either direction. Setting L'/R=1 changes the typical direction, which is to the right. When L'/R equals 1, the multiplexer AND gates R are activated.

The data enters at SR, passes through QA, QB, and QC, and then exits at SR cascade. If this pin is used, the SR of something else may shift to the right. What happens if we set L'/R to zero?

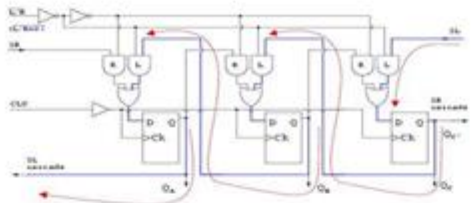


Fig-11:The shift left/right key and the left action key

When L'/R=0, the multiplexer AND gates labeled L turn on, resulting in the identical path as shown by the arrows in the preceding "shift left" figure. The data enters at SL, passes through QC, QB, and QA, and finally exits at SL cascade. If this pin is utilized, the SL of something else may shift to the left. The simplicity of the two images above that demonstrate "shift left/right register" is the nicest part. The left-right setting L'/R=0 is straightforward. The parallel data loading mentioned in the section title is required for a business part. This is seen in the following image. Now that we know how to utilize the L'/R gates to shift to the left and right, let's add the SH/LD', shift/load, and "load" AND gates to allow data to be loaded in parallel from inputs DA, DB, and DC. If SH/LD' is 0 and gates R and L are both turned off. BUT gates "load" to transport data from DA to DB to DC to the FF data ports. The data will be sent to QA QB QC by the next clock CLK.

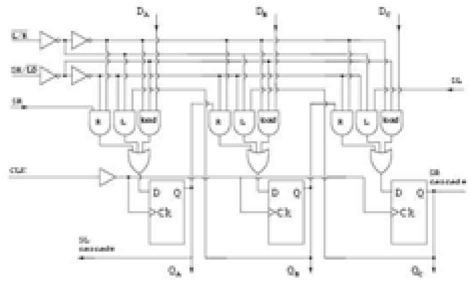


Fig-12:Load and shift (left/right).

IfSH/LD'ischangedtoSH/LD'=1,theANDgateslabelled"load" are disabled, allowing the left/ right control L'/R toset the direction of shift on the L or R AND gates. Shifting isas in the previous figure The only thing needed to produce aviable integrated deviceis to add the fourth AND gate to the multiplexer as alluded for the 74ALS299. This is shown in the next section for that part.

2.DESIGN AND ANALYSE A SYNCHRONOUS FIFO

Create and test a FIFO with various read and write logics. The data in each of the 64 sources we examined was 32 bits long. The oldest request is dealt with first when utilizing the FIFO approach to handle program work requests from stacks or lines. A collection of flip-flops or read/write memory with FIFO logic is used in hardware to store data from one clock domain and deliver it to other clock domains when requested. The clock domain that delivers data to the FIFO is referred to as "write logic," whereas the clock domain that receives data from the FIFO is referred to as "read logic."

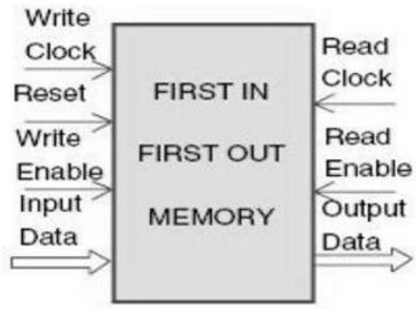


Fig-13:FIFOmemoryI/O

3.**CONCLUSIONS**

When developing an asynchronous FIFO, you must consider every detail, such as how to produce pointers and the distinction between full and empty generation. When critical aspects are overlooked, the design is frequently incorrect and easily verifiable. FIFO design issues are typically discovered by simulating a gate-level FIFO architecture and noting real delays on the back. Gray code pointers are used to safely sync FIFO pointers into the opposite clock domain. Gray code pointers are used to safely sync FIFO pointers into the opposite clock domain.The most difficult aspect of a FIFO plan may be determining the full status. Dual n-bit Gray code counters can be used to synchronize an n-bit pointer with the other clock domain as well as to perform "full" comparisons with a (n-1)-bit pointer. When doing FIFO design, another handy thing you can do is sync binary FIFO pointers using the methods provided. The FIFO-empty state is easily obtained by comparing and equaling the synchronized n-bit write pointer to the n-bit read pointer. The methods suggested in this article should be able to work with asynchronous clocks with small to big discrepancies.

REFERENCES

1. N.Verma,“AnalysistowardsminimizationoftotalSRAMenergy over active and idle operating modes,” IEEETrans. on VLSI Systems, Vol. 19, No. 9, pp. 1695-1703,Sept.2010.
2. K.Nii,etal.,“A65nmultra-high-densitydual-portSRAMwith 0.71um28T-cellforSoC,” IEEE Symp. on VLSICircuits, pp. 130-131, 2006.R. Nicole, “Title of paper with only first word capitalized,”J.NameStand.Abbrev.,inpress.
3. W.-H. Du, et al, “An Energy-Efficient 10T SRAM-based FIFO Memory Operating in Near-/Sub-threshold Regions,” IEEE System-on-Chip Conference, pp. 19-23,Sept.2011.K.Elissa,“Titleofpaperifknown,”unpublished.
4. D. Markovic, et al., “Ultralow-power design innear-threshold region,” IEEE Proceedings, vol. 98, no 2, pp.237-252,Feb.2010.
5. I.-J. Chang, et al., “A 32 kb 10T Sub-Threshold SRAM Array with Bit-Inter leaving and Differential Read Scheme in 90 nm CMOS,” IEEE Journal of Solid-StateCircuits,pp650-658,Feb.2009.K.Elissa,“Titleofpaperifknown,”unpublished.
6. Y.-T.Chiu,etal., “Subthreshold Asynchronous FIFO Memory for Wireless Body Area Networks (WBANs)”,International Symposium on Medical Information and Communication Technology(ISMICT),March2010.
7. M.-H. Tu, et al, “Single-ended Subthreshold SRAM withAsymmetrical Write/Read-Assist,” in IEEE Trans. onCircuitsandSystems,Vol.57,No.12,pp.3039-3047,Dec.2010.K.Elissa,“Titleofpaperifknown,”unpublished.
8. M.-T.Chang,etal.,“ARobustUltra-LowPowerAsynchronous FIFO Memory with Self-Adaptive PowerControl,”IEEESystem-on-ChipConference,pp.175-178,2008.
9. W.-H.Du,etal.,“A2kbbuilt-inrow-controlleddynamicvoltage scaling near-/sub-threshold FIFO memory forWBANs," IEEE International Symposium on VLSI Design,Automation,and Test(VLSI-DAT,pp.1-4,2012.