International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 09 | Sep 2020 www.irjet.net p-ISSN: 2395-0072

ASYNCHRONOUS FIFO DESIGN USING VERILOG

Lincy D.F[1], S.Thenappan[2]

[1]P.G. Scholar in VLSI DESIGN, Electronics and Communication Engineering Department,
[2]Ass.Professor, Electronics and Communication Engineering Department
[1, 2] GNANAMANI COLLEGE OF TECHNOLOGY, NAMAKKAL, TAMILNADU.

Abstract - *FIFO is an approach for handling program work requests from queues or stacks so that the oldest request is handled first. In hardware, it is either an array of flops or read/write memory that stores data from one clock domain and on request supplies the same data to other clock domains following FIFO logic. An improved technique for FIFO design is to perform asynchronous comparisons between the FIFO write and read pointers that are generated in clock domains and asynchronous to each other. The asynchronous FIFO pointer comparison technique uses fewer synchronization flip-flops to build the FIFO. This method requires additional techniques to correctly synthesize and analyse the design, which are detailed in this paper. To increase the speed of the FIFO, this design uses combined binary/Gray counters that take advantage of the built-in binary ripple carry logic.*

Key Words: Asynchronous FIFO, FIFO Design, Full and Empty Deduction

1. INTRODUCTION

FIFO Using Different Read and Write Logics Design where data values are written sequentially into a FIFO buffer using one clock domain, and the data values are sequentially read from the same FIFO buffer using another clock domain, where the two clock domains are asynchronous to each other. One common technique for designing an asynchronous FIFO is to use Gray code pointers that are synchronized into the opposite clock domain before generating synchronous FIFO full or empty status signals. An interesting and different approach to FIFO full and empty generation is to do an asynchronous comparison of the pointers and then asynchronously set the full or empty status bits.

1.1 Full and Empty Deductions

As with any FIFO design, correct implementation of full and empty is the most difficult part of the design. Therefore, something else has to distinguish between full and empty. The one described is that divides the address space into four quadrants and decodes the two MSBs of the two counters to determine whether the FIFO was going full or going empty at the time the two pointers became equal.

FIFO is going full because the wptr trails the rptr by one quadrant If the write pointer is one quadrant behind the read pointer, this indicates a "possibly going full" situation as shown. When this condition occurs, the direction latch is set.

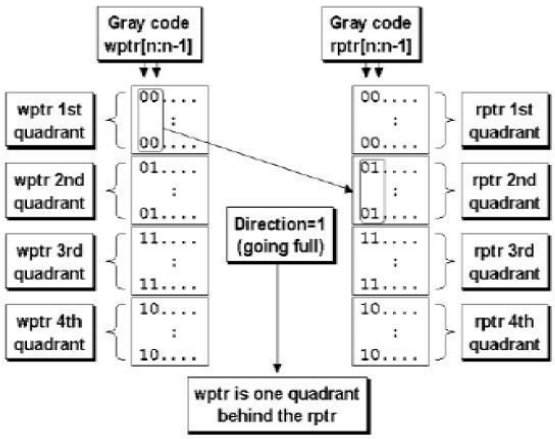


Fig -1: FIFO is going empty because the rptr trails the wptr by one quadrant

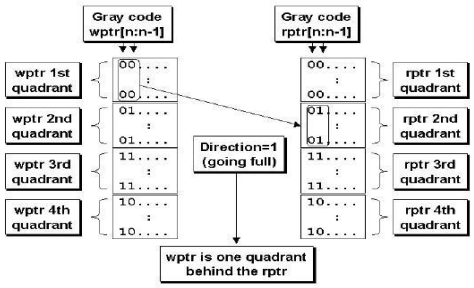


Fig -2: FIFO is going full because the wptr trails the rptr by one quadrant

If the write pointer is one quadrant ahead of the read pointer, this indicates a "possibly going empty" situation as shown. When this condition occurs, the direction latch is cleared.

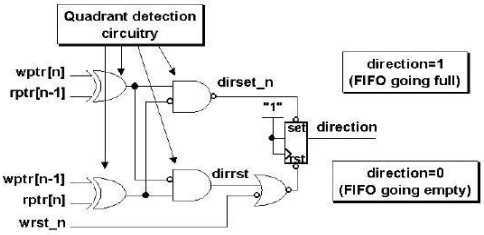


Fig -3: FIFO is going full because the wptr trails the rptr by one quadrant

When the FIFO is reset the direction latch is also cleared to indicate that the FIFO "is going empty". Setting and resetting the direction latch is not timing-critical, and the direction

latch eliminates the ambiguity of the address identity decoder. The Xilinx FPGA logic to implement the decoding of the two wptrMSBs and the two rptrMSBs is easily implemented as two 4-input look-up tables. The second, and more difficult, problem stems from the asynchronous nature of the write and read clocks. Comparing two counters that are clocked asynchronously can lead to unreliable decoding spikes when either or both counters change multiple bits more or less simultaneously. The solution described in this paper uses a Gray count sequence, where only one bit changes from any count to the next. Any decoder or comparator will then switch only from one valid output to the next one, with no danger of spurious decoding glitches.

1.2 FIFO2.v

This is the top-level wrapper-module that includes all clock domains. The top module is only used as a wrapper to instantiate all of the other FIFO modules used in the design. If this FIFO is used as part of a larger ASIC or FPGA design, this top-level wrapper would probably be discarded to permit grouping of the other FIFO modules into their respective clock domains for improved synthesis and static timing analysis.

1.3 FIFOmem.v

This is the FIFO memory buffer that is accessed by both the write and read clock domains. This buffer is most likely an instantiated, synchronous dual-port RAM. Other memory styles can be adapted to function as the FIFO buffer.

1.4 async_cmp.v

This is an asynchronous pointer-comparison module that is used to generate signals that control assertion of the asynchronous "full" and "empty" status bits. This module only contains combinational comparison logic. No sequential logic is included in this module.

1.5 rptr_empty.v

This module is mostly synchronous to the read-clock domain and contains the FIFO read pointer and empty-flag logic. Assertion of the aempty_n signal (an input to this module) is synchronous to the rclk- domain, since aempty_n can only be asserted when the rptr incremented, but de-assertion of the aempty_n signal happens when the wptr increments, which is asynchronous to rclk.

1.6 wptr_full.v

This module is mostly synchronous to the write-clock domain and contains the FIFO write pointer and full-flag logic. Assertion of the afull_nsignal (an input to this module) is synchronous to the wclk-domain, since afull_ncan only be asserted when the wptrincremented (and wrst_n), but de-assertion of the afull_nsignal happens when the rptrincrements, which is asynchronous to wclk.

1.7 Asynchronous Generation of Full And Empty

In the async_cmp shown is aempty_n and afull_n are the asynchronously decoded signals. The aempty_n signal is asserted on the rising edge of an rclk, but is de-asserted on the rising edge of a wclk. Similarly, the afull_n signal is asserted on a wclk and removed on an rclk. The empty signal will be used to stop the next read operation, and the leading edge of aempty_n is properly synchronous with the read clock, but the railing edge needs to be synchronized to the read clock. This is done in a two-stage synchronizer that generates r_empty. The w_full signal is generated in the symmetrically equivalent way.

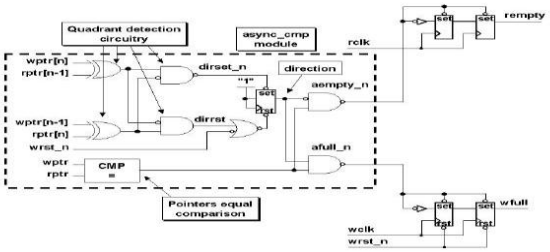


Fig -4: Asynchronous pointer comparison to assert full and empty

1.8 Resetting the FIFO

The first FIFO event of interest takes place on a FIFO-reset operation. When the FIFO is reset, four important things happen within the async_cmp module and accompanying full and empty synchronizers of the wptr_full and rptr_empty modules (the connections between the async_cmp, wptr_full and rptr_empty modules are shown.

1. The reset signal directly clears the w_fullflag. The r_emptyf lag is not cleared by a reset.
2. The reset signal clears both FIFO pointers, so the pointer comparator asserts that the pointers are equal.
3. The reset clears the direction bit.
4. With the pointers equal and the direction bit cleared, the empty_n bit is asserted, which pre-sets the r_emptyflag.

1.9 Parallel-In, Parallel-Out, Universal Shift Register

The purpose of the parallel-in/ parallel-out shift register is to take in parallel data, shift it, then output it as shown below. A universal shift register is a do-everything device in addition to the parallel-in/ parallel-out function.

We apply four bit of data to a parallel-in/ parallel-out shift register at DA DB DC DD. The mode control, which may be

multiple inputs, controls parallel loading vs shifting. The mode control may also control the direction of shifting in some real devices. The data will be shifted one bit position for each clock pulse. The shifted data is available at the outputs QA QB QC QD.

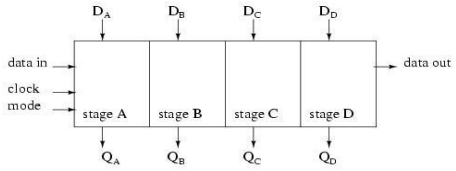


Fig -5: Parallel-in, Parallel-out Shift Register with 4-Stages

The "data in" and "data out" are provided for cascading of multiple stages. Though, above, we can only cascade data for right shifting. We could accommodate cascading of left-shift data by adding a pair of left pointing signals, "data in" and "data out", above. The internal details of a right shifting parallel-in/ parallel-out shift register are shown below. The tri-state buffers are not strictly necessary to the parallel-in/ parallel-out shift register, but are part of the real- world device shown below. The 74LS395 so closely matches our concept of a hypothetical right shifting parallel-in/ parallel-out shift register that we use an overly simplified version of the data sheet details above. LD/SH' controls the AND-OR multiplexer at the data input to the FF's. If LD/SH'=1, the upper four AND gates are enabled allowing application of parallel inputs DA DB DC DD to the four FF data inputs. Note the inverter bubble at the clock input of the four FFs. This indicates that the 74LS395 clocks data on the negative going clock, which is the high to low transition. The four bits of data will be clocked in parallel from DA DB DC DD to QA QB QC QD at the next negative going clock. In this "real part", OC' must be low if the data needs to be available at the actual output pins as opposed to only on the internal FFs.

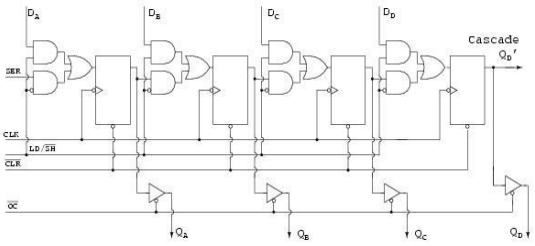


Fig -6: Parallel-in, Parallel-out Shift Register with tri-state Output

The previously loaded data may be shifted right by one bit position if LD/SH'=0 for the succeeding negative going clock edges. Four clocks would shift the data entirely out of our 4-bit shift register. The data would be lost unless our device was cascaded from QD' to SER of another device.

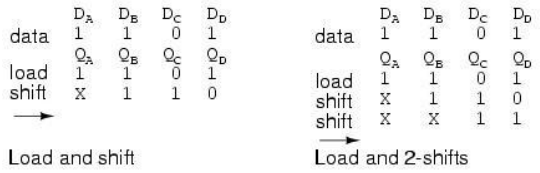


Fig -7: Parallel-in, Parallel-out Shift Register

Above, a data pattern is presented to inputs DA DB DC DD. The pattern is loaded to QA QB QC QD . Then it is shifted one bit to the right. The incoming data is indicated by X, meaning we do no know what it is. If the input (SER) were grounded, for example, we would know what data (0) was shifted in. Also shown, is right shifting by two positions, requiring two clocks.

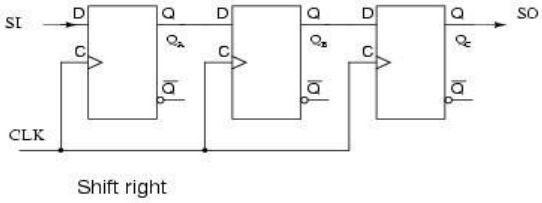
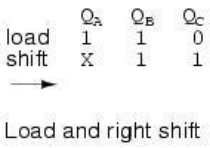


Fig -8: Shift Right

The above figure serves as a reference for the hardware involved in right shifting of data. It is too simple to even bother with this figure, except for comparison to more complex figures to follow.



Right shifting of data is provided above for reference to the previous right shifter.

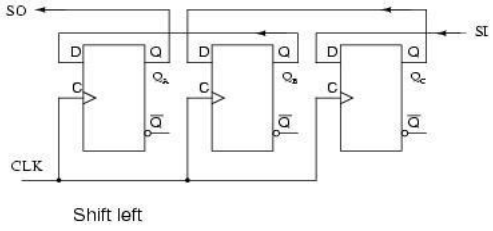
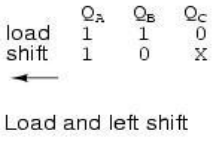


Fig -9: Shift Left

If we need to shift left, the FFs need to be rewired. Compare to the previous right shifter. Also, SI and SO have been reversed. SI shifts to QC. QC shifts to QB. QB shifts to QA. QA leaves on the SO connection, where it could cascade to another shifter SI.



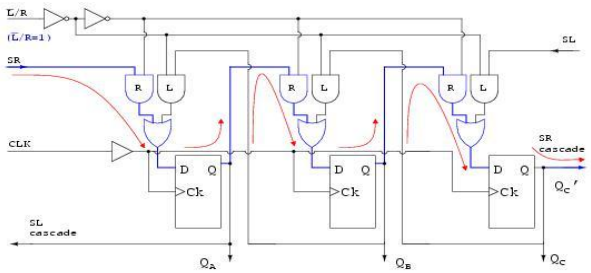


Fig -10: Shift Left/Right, Right Action

What we have above is a hypothetical shift register capable of shifting either direction under the control of L'/R. It is setup with L'/R=1 to shift the normal direction, right. L'/R=1 enables the multiplexer AND gates labelled R.

Data shifts in at SR, to QA, to QB, to QC, where it leaves at SR cascade. This pin could drive SR of another device to the right. What if we change L'/R to L'/R=0?

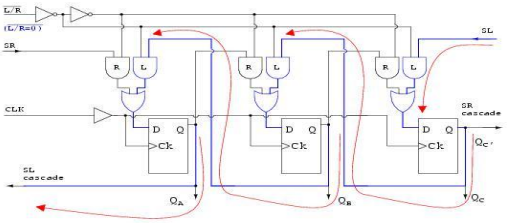


Fig -11: Shift Left/Right Register, Left Action

With L'/R=0, the multiplexer AND gates labelled L are enabled, yielding a path, shown by the arrows, the same as the above "shift left" figure. Data shifts in at SL, to QC, to QB, to QA, where it leaves at SL cascade. This pin could drive SL of another device to the left. The prime virtue of the above two figures illustrating the "shift left/ right register" is simplicity. The operation of the left right control L'/R=0 is easy to follow. A commercial part needs the parallel data loading implied by the section title. This appears in the figure below. Now that we can shift both left and right via L'/R, let us add SH/LD', shift/ load, and the AND gates labelled "load" to provide for parallel loading of data from inputs DA DB DC. When SH/LD'=0, AND gates R and L are disabled, AND gates "load" are enabled to pass data DA DB DC to the FF data inputs. the next clock CLK will clock the data to QA QB QC.

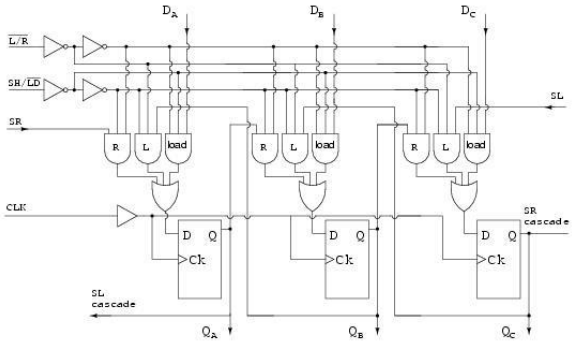


Fig -12: Shift Left/Right Load

If SH/LD' is changed to SH/LD'=1, the AND gates labelled "load" are disabled, allowing the left/ right control L'/R to set the direction of shift on the L or R AND gates. Shifting is as in the previous figure The only thing needed to produce a viable integrated device is 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 ASYNCHRONOUS FIFO

Design and analyze FIFO using different read and write logics. We have considered 64 inputs, each having 32- bit data. FIFO is an approach for handling program work requests from queues or stacks so that the oldest request is handled first. In hardware, it is either an array of flops or read/write memory that stores data from one clock domain and on request supplies the same data to other clock domains following FIFO logic. Clock domain that supplies data to FIFO is often referred to as write or input logic, and clock domain that reads data from FIFO is often referred to as read or output logic

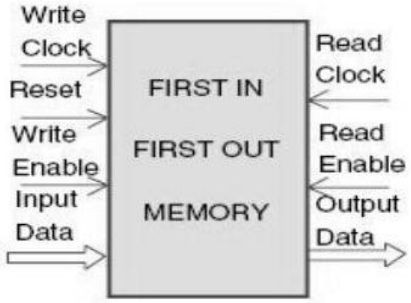


Fig -13: FIFO memory I/O

3. CONCLUSIONS

Asynchronous FIFO design requires careful attention to details from pointer generation techniques to full and empty generation. Ignorance of important details will generally result in a design that is easily verified but is also wrong. finding FIFO design errors typically requires simulation of a gate-level FIFO design with back annotation of actual delays. Synchronization of FIFO pointers into the opposite clock domain is safely accomplished using Gray code pointers Synchronization of FIFO pointers into the opposite clock domain is safely accomplished using Gray code pointers .Generating the FIFO -full status is perhaps the hardest part of a FIFO design. Dual n-bit Gray code counters are valuable to synchronize and n-bit pointer into the opposite clock domain and to use an (n-1)-bit pointer to do “full” comparison. Synchronizing binary FIFO pointers using techniques described is another worthy technique to use when doing FIFO design .Generating the FIFO -empty status is easily accomplished by comparing-equal the n-bit read pointer to the synchronized n-bit write pointer. The techniques described in this paper should work with asynchronous clocks spanning small to large differences.

REFERENCES

[1] N. Verma, "Analysis towards minimization of total SRAM energy over active and idle operating modes," IEEE Trans. on VLSI Systems, Vol. 19, No. 9, pp. 1695-1703, Sept. 2010.

[2] K. Nii, et al., "A 65 nm ultra-high-density dual-port SRAM with 0.71um2 8T-cell for SoC," IEEE Symp. on VLSI Circuits, pp. 130-131, 2006.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[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, "Title of paper if known," unpublished.

[4] D. Markovic, et al., "Ultralow-power design in near-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- Interleaving and Differential Read Scheme in 90 nm CMOS," IEEE Journal of Solid-State Circuits, pp 650-658, Feb. 2009.K. Elissa, "Title of paper if known," unpublished.

[6] Y.-T. Chiu, et al., "Subthreshold Asynchronous FIFO Memory for Wireless Body Area Networks (WBANs)", International Symposium on Medical Information and Communication Technology (ISMICT), March 2010.

[7] M.-H. Tu, et al, "Single-ended Subthreshold SRAM with Asymmetrical Write/Read-Assist," in IEEE Trans. on Circuits and Systems, Vol. 57, No. 12, pp.3039-3047, Dec. 2010. K. Elissa, "Title of paper if known," unpublished.

[8] M.-T. Chang, et al., "A Robust Ultra- Low Power Asynchronous FIFO Memory with Self-Adaptive Power Control," IEEE System-on-Chip Conference, pp. 175-178, 2008.

[9] W.-H. Du, et al., "A 2kb built-in row-controlled dynamic voltage scaling near-/sub-threshold FIFO memory for WBANs," IEEE International Symposium on VLSI Design, Automation, and Test (VLSI- DAT, pp.1-4, 2012.