IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
ISSN: 2319 – 4200, ISBN No. : 2319 – 4197 Volume 1, Issue 3 (Nov. - Dec. 2012), PP 32-37
www.iosrjournals.org

Asynchronous FIFO Design with Gray code Pointer for High Speed AMBA AHB Compliant Memory controller

G.Ramesh[1], V.Shivaraj Kumar[2], K.Jeevan Reddy[3]
Dept. of Electronics and Communication, JNTUH, India

Abstract: *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 analyze 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. This FIFO design is used to implement the AMBA AHB Compliant Memory Controller. Which means, Advanced Microcontroller Bus Architecture compliant Microcontroller .The MC is designed for system memory control with the main memory consisting of SRAM and ROM.*
Keywords: *AMBA, AHB ,FIFO, Gray Counter, ,Memory Controller*

I. Introduction

An asynchronous FIFO refers to a FIFO 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 asynchronyous 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.This paper discusses the FIFO design style with asyn-chronous pointer comparison and asynchronous full and empty generation. Important details relating to this style of asynchronous FIFO design are included. The FIFO style implemented in this paper uses efficient Gray code counters, whose implementation is described in the next section.

II. Gray Code Counter

One Gray code counter style uses a single set of flip-flops as the Gray code register with accompanying Gray-to binary conversion, binary increment, and binary-to-Gray conversion. A second Gray code counter style, the one described in this paper, uses two sets of registers, one a binary counter and a second to capture a binary-to-Gray converted value. The intent of this Gray code counter is to utilize the binary carry structure, simplify the Gray-to-binary conversion; reduce combinational logic, and increase the upper frequency limit of the Gray code counter. The binary counter conditionally increments the binary value, which is passed to both the inputs of the binary counter as the next-binary-count value, and is also passed to the simple binary-to-Gray conversion logic, consisting of one 2-input XOR gate per bit position. The converted binary value is the next Gray-count value and drives the Gray code register inputs. Fig:1 shows the block diagram for an n-bit Gray code counter.

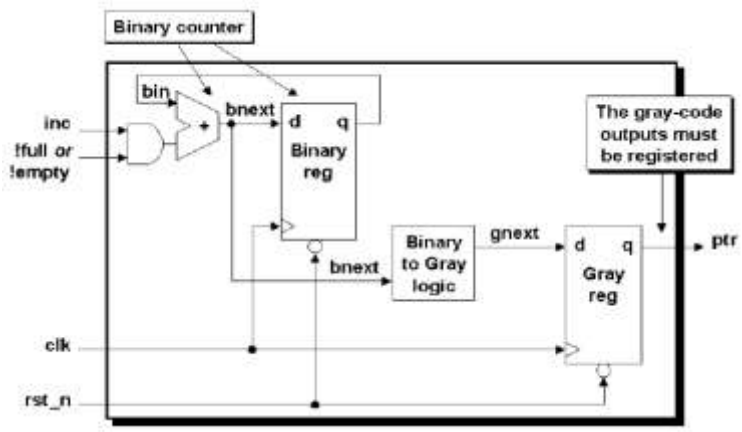

Figure 1.n-bit Gray code counter.

This implementation requires twice the number of flip-flops, but reduces the combinatorial logic and can operate at a higher frequency. In FPGA designs, availability of extra flip-flops is rarely a problem since FPGAs typically contain far more flip-flops than any design will ever use. In FPGA designs, reducing the amount of combinational logic frequently translates into significant improvements in speed. The *ptr* output of the block diagram in figure.1is an n-bit Gray code pointer.

III. Full & Empty Detection

There are two problems with the generation of full and empty First, both full and empty are indicated by the fact that the read and write pointers are identical. Therefore, something else has to distinguish between full and empty. One known solution to this problem appends an extra bit to both pointers and then compares the extra bit for equality (for FIFO empty) or inequality (for FIFO full), alongwith equality of the other read and write pointer bits.

Another solution 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.

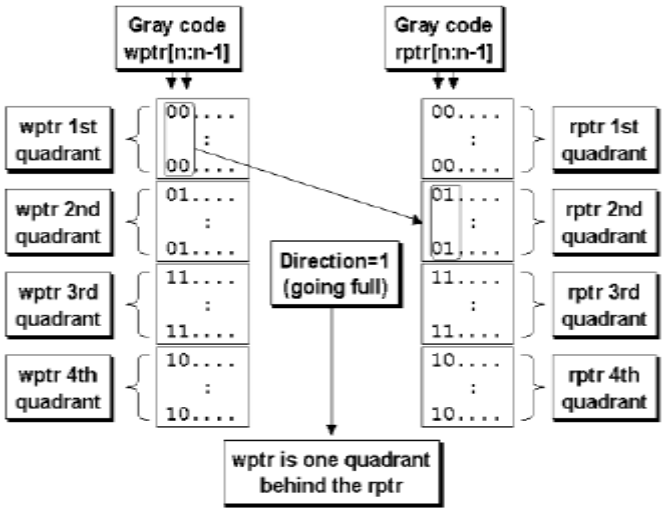

Figure.2: FIFO is going full because the wptr trails the rptr by one quadrant If the write pointer is one quadrant behind the read pointer, this indicates a "possibly going full" situation as shown in Fig: 2. When this condition occurs, the direction latch of Figure 4 is set.

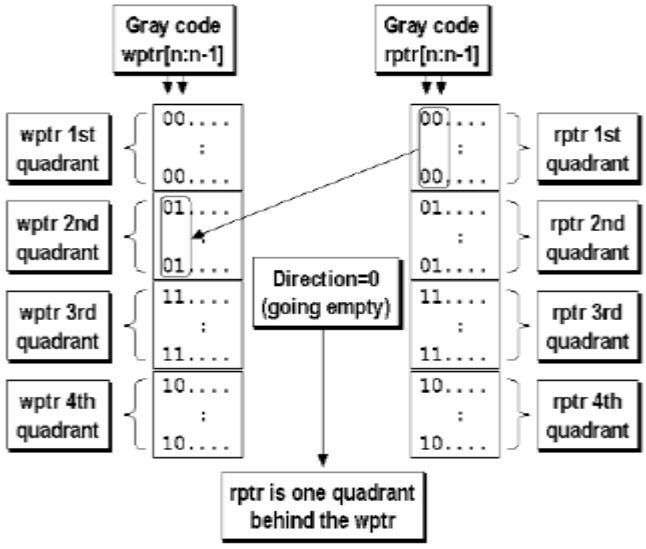

Figure 3.FIFO is going empty because the rptr trails the wptr by one quadrant

If the write pointer is one quadrant ahead of the read pointer, this indicates a "possibly going empty" situation as shown in Fig: 3. when this condition occurs, the direction latch of Fig: 4 is cleared.

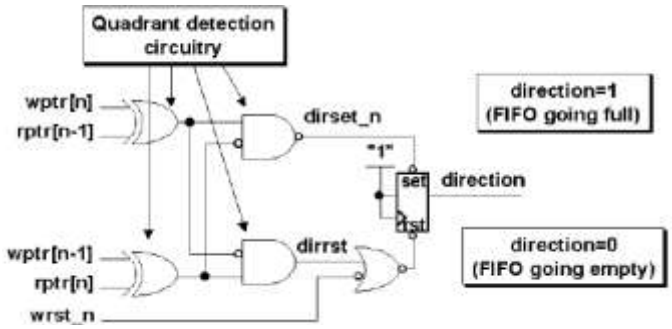

Figure 4.FIFO direction quadrant detection circuitry

When the FIFO is reset the direction latch is also cleared to indicate that the FIFO "is going empty" (actually, it is empty when both pointers are reset). Setting and resetting the direction the direction latch eliminates the ambiguity of the address identity decoder.

IV. First In First Out [Fifo]

This FIFO implementation synchronizes the pointers from one clock domain to another before generating full and empty flags. The FIFO style provides asynchronous comparison between Gray code pointers to generate an asynchronous control signal to set and reset the full and empty flip-flops. The block diagram for FIFO is shown in Fig: 5.

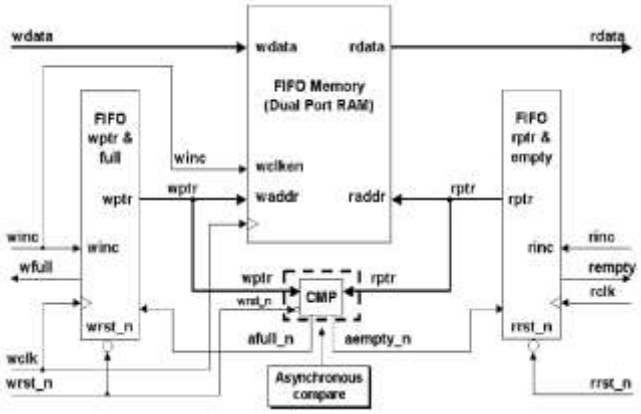

Figure 5. FIFO2 partitioning with asynchronous pointer comparison logic

4.1. Asynchronous generation of full and empty

In the async_cmp shown in Fig: 6, 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 trailing edge needs to be synchronized to the read clock. This is done in a two-stage synchronizer that generates rempty. The wfull signal is generated in the symmetrically equivalent way.

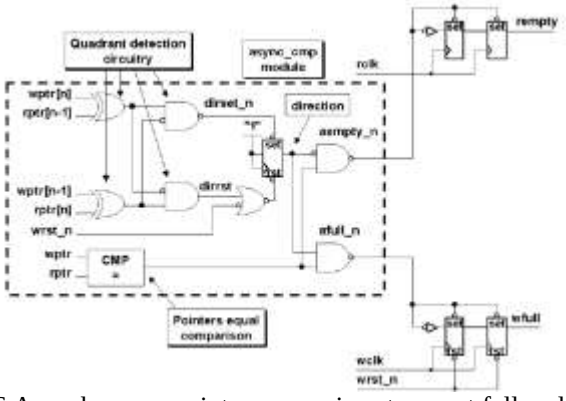

Figure 6.Asynchronous pointer comparison to assert full and empty

4.2. 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 ac-companying 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 in Fig 7.

1. The reset signal directly clears the wfull flag. The rempty flag 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 aempty_n bit is asserted, which presets the rempty flag.

4.3. FIFO writes & FIFO full

The second FIFO operational event of interest takes place when a FIFO-write operation takes place and the wptr is incremented. At this point, the FIFO pointers are no longer equal so the aempty_n signal is de-asserted, releasing the preset control of the rempty flip-flops. After two rising edges on rclk, the FIFO will de-assert the rempty signal. Because the de-assertion of aempty_n happens on a rising wclk and because the rempty signal is clocked by the rclk, the two-flip-flop synchronizer as shown in Fig: 8 are required to remove meta-stability that could be generated by the first rempty flip-flop. The second FIFO operational event of interest takes place when the wptr increments into the next Gray code quadrant beyond the rptr. The direction bit is cleared.

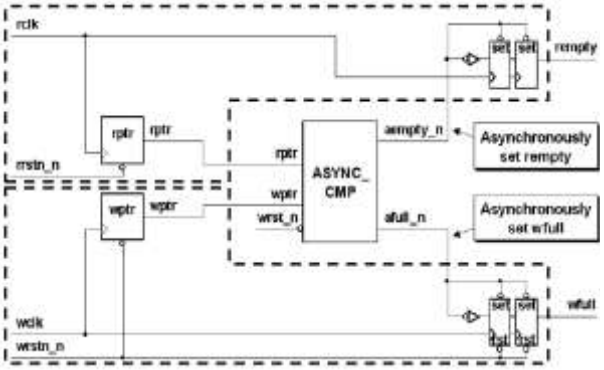

Figure 7.async_cmp module connection to rptr_empty and wptr_full modules.

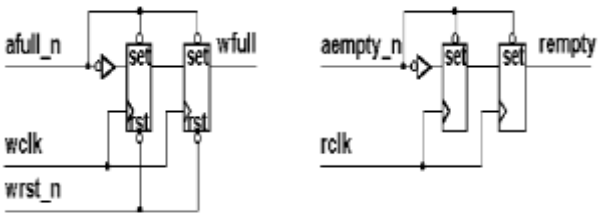

Figure 8 .Asynchronous empty and full generation

4.4. FIFO reads & FIFO empty

when the rptr increments into the next Gray code quadrant beyond the wptr. The direction bit is again set (but it was already set).when the rptr is within one quadrant of catching up to the wptr. At this instant, the dirrst bit of Fig 6 is asserted high, which clears the dire-ction bit. This means that the direction bit is cleared long before the FIFO is empty and is not timing critical to assertion of the aempty_n signal. When the rptr catches up to the wptr (and the direction bit is zero). The aempty_n signal presets the rempty flip-flops. The aempty_n signal is asserted on a FIFO-read operation and is synchronous to the rising edge of the rclk; therefore, asserting empty is synchronous to the rclk.

Finally, when a FIFO-write operation takes place and the wptr is incremented. At this point, the FIFO pointers are no longer equal so the aempty_n signal is de-asserted, releasing the preset control of the rempty flip-flops. After two rising edges on rclk, the FIFO will de-assert the rempty signal. Because the de-assertion of aempty_n happens on a rising wclk and because the rempty signal is clocked by the rclk, the two-flip-flop synchronizer.

4.5. Full and Empty critical timing paths

Using the asynchronous comparison technique described in this paper, there are critical timing paths associated with the generation of both the rempty and wfull signals. The rempty critical timing path, shown in Fig :9 , consists of (1) the rclk-to-q incrementing of the rptr, (2) comparison logic of the rptr to the wptr, (3) combining the comparator output with the direction latch output to Generate the aempty_n signal, (4) presetting the rempty signal, (5) any logic that is driven by the rempty signal, and (6) resultant signals meeting the setup time of any down-stream flip-flops clocked within the rclk domain. This critical timing path has a symmetrically equivalent critical timing path for the generation of the wfull signal, also shown in Fig: 9.

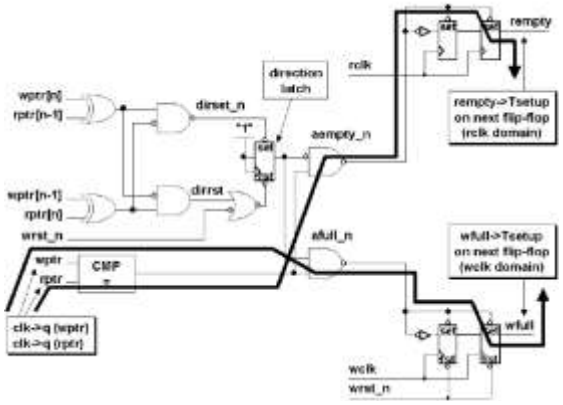

Figure 9 .Critical timing paths for asserting rempty and wfull.

V. Implementation Of Amba Ahb Memory Controller

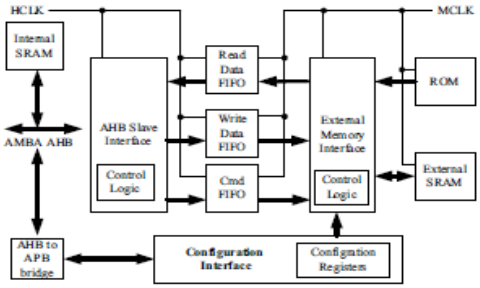

Figure 10.Architecture of AHB-MC

In order to improve the speed constraint and reduce the number of flip flops gray counter FIFO is Included. simulation results of FIFO shows gray counter FIFO.it is practically proved that, Gray Counter FIFO offers high speed for reading from and writing into the ROM,SRAM memories.FIFO plays a vital role in implementations of Memory controller for avoiding Memory latancies.Number of flip flop's are reduced compared to conventional FIFO styles.AMBA Memory controller has two types of clock domains.

1) AHB clock domain (HCLK).
2) Memory clock domain (MCLK).

Synchronization must require for proper communication between memory and AHB modules. Here, Gray counter FIFO provides synchronization between these two domains.

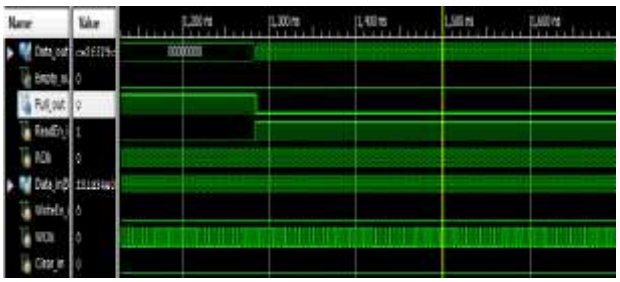

Figure 11.Gray pointer FIFO simulation

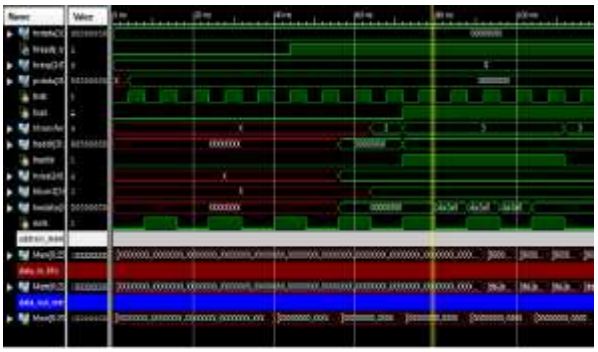

Figure 12. Writing the data into memory

Figure 13. Reading the data from memory

VI. Conclusion

This paper describes an efficient technique to implement a high-speed asynchronous FIFO, using dual-port RAMs addressed by Gray counters this design uses an asynchronous comparator for detecting full and empty status. The technique described implements an asynchronous assertion of the full and empty flags that requires more effort to analyze for static timing verification. The technique described also does not have registered full and empty status flags, so care must be taken to insure that the generation of these flags meets the required timing to recognize assertion of full and empty in the rest of the system.

References

[1] Clifford E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 2nd paper. Also available at www.sunburst-design.com/papers

[2] Clifford E. Cummings, "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs," *SNUG 2001 (Synopsys Users Group Conference, San Jose, CA, 2001) User Papers*, March 2001, Section MC1, 3rd paper. Also available at www.sunburst-design.com/papers

[3] Clifford E. Cummings and Don Mills, "Synchronous Resets? Asynchronous Resets? I am So Confused! How Will I Ever Know Which to Use?" *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 1st paper. Also available at www.sunburst-design.com/papers

[4] Frank Gray, "Pulse Code Communication." United States Patent Number 2,632,058. March 17, 1953.

[5] John O'Malley, *Introduction to the Digital Computer*, Holt, Rinehart and Winston, Inc., 1972, pg. 190.

[6] Peter Alfke, "Asynchronous FIFO in Virtex-II™ FPGAs," Xilinx techXclusives, downloaded from www.xilinx.com/support/techXclusives/fifo-techX18.htm

[7] "Prime Cell AHB SRAM/NOR Memory Controller", Technical Reference Manual, ARM Inc. Building an AMBA AHB compliant Memory Controller Hu Yueli1,2, Yang Ben1,2 Key Laboratory of Advanced Display and System Applications, Ministry of Education, College of Mechanical and Electronic Engineering and Automation, Shanghai University, Shanghai 200072, China