Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.231 – 237

RESEARCH ARTICLE

DESIGNING RESOURCEFUL COUNTER FOR FIFO

S. Janani[1], S. Thillaikkarasi[2]

PG Scholar (VLSI Design), Sri Eshwar College of Engineering, Coimbatore, TamilNadu, India

Assistant Professor, Department of ECE, Sri Eshwar College of Engineering, Coimbatore, TamilNadu, India

ABSTRACT

First-in first-out memories (FIFOs) have progressed from objectively simple logic functions to high-speed buffers combining large blocks of SRAM. FIFOs are often used to carefully pass data from one clock dominion to another asynchronous clock dominion. Using a FIFO to pass data from one clock domain to another clock domain requires multi- asynchronous clock design techniques. This concept will detail one method that is used to design, synthesize and analyze a safe FIFO between different clock domains using Memory counter that are synchronized into a different clock domain before testing for "FIFO full" or "FIFO empty" conditions. The fully analysed, synthesized counter can be included in FIFO.

Keywords: FIFO, Asynchronous FIFO, Memory Counter

I. INTRODUCTION

In every digital components, there exits the exchange of data between printed circuit boards(PCB). Intermediate storage or buffering always is essential when data reach at the receiving PCB at a high speed, but are processed slowly or unevenly. Buffers here analysed with our everyday life (for example, queue formation in bank, super market etc.,). This queue works with the principle of first come, first served. Buffers acts as the interface between components that work at different speeds or irregularly. In electronic devices, speed determines the data rate, the data rate of the A/D converter is controlled by a quartz crystal. The different data rates are compensated by buffering.

A FIFO is a type of buffer, where the data which is written into a buffer, first comes out. There are other kinds of buffers like the LIFO (last in first out), often called a stack memory, and the shared memory. The buffer can be chosen based on the application. FIFO can be implemented both with software and hardware. Software is more flexible than Hardware. Asynchronous FIFOs are one of FIFOs which is used to securely pass data from one clock domain to another clock domain. An asynchronous FIFO refers to a FIFO design where data values are read and written to a FIFO buffer from one clock domain to another. Asynchronous FIFO that

work properly 99 percentage and plus of the time have design flaws that are typically the most difficult to sense and correct.

Fig.1. shows the common flow of input and output in FIFO. Every memory in which the data word that is written in first also comes out first when the memory is read is a first-in first-out memory. There are three kinds FIFO which can be used according to the applications and their needs. Three kinds of FIFO are Shift Register, Exclusive read/write FIFO, Concurrent read/write FIFO. In Shift register, there are constant number of stored data words and thus, the necessary synchronism between the read and the write processes because a data word must be recite every time one is printed. In Exclusive read/write FIFO, a variable number of stored data words and, because of the internal structure, the necessary synchronism between the read and the write operations. In Concurrent read/write FIFO, a variable number of stored data words and possible asynchronous between the read and the write operation.

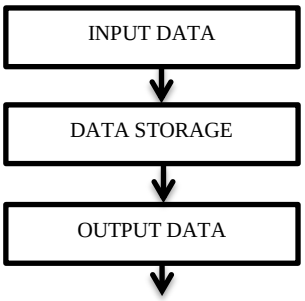

Fig.1. First In First Out flow

This is brief is divided into four sections, the first of which is the Introduction. Section II discuss about the Asynchronous FIFO pointers and clarifies about its method of reading and writing. Section III gives the detail of FIFO through Static Memory using Circular FIFO. Section IV views the Memory Counter for the efficient memory access. Section V provide the relevant simulation result and its analysis. Finally conclusions are drawn in section VI.

II. ASYNCHRONOUS FIFO POINTERS

In order to know FIFO design, one desires to understand in what way the FIFO pointers work. There are mainly two pointers.

1. Write Pointer
2. Read Pointer

The write pointer continuously points to the next word to be written; for that reason, on reset, both pointers are set to zero, which also occurs to be the next FIFO word location to be written. Fig.2. gives the write and read operations, on a FIFO-write operation, the memory location that is piercing to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.

Correspondingly, the read pointer always points to the present FIFO word to be read. Again on reset, both pointers are set to zero, the FIFO is empty and the read pointer is pointing to invalid data (because the FIFO is empty and the empty flag is declared). As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the read pointer that is still addressing the fillings of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic.

The fact that the read pointer is always pointing to the next FIFO word to be read means that the receiver logic does not have to use two clock periods to read the data word. If the receiver first had to increment the read pointer before reading a FIFO data word, the receiver would clock once to output the data word from the FIFO, and clock an additional time to capture the data word into the receiver. That would be pointlessly inefficient.

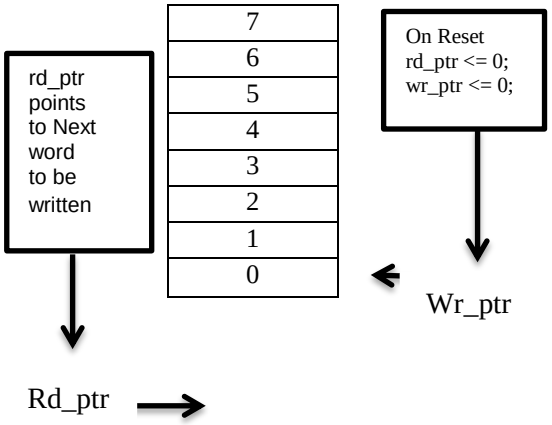

Fig.2. FIFO Pointer

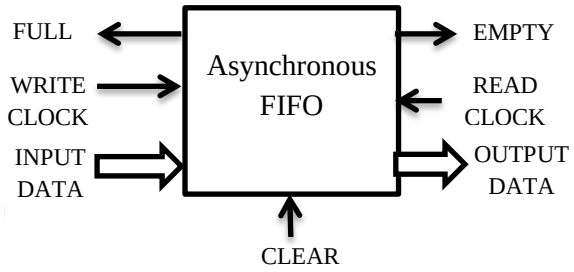

Fig.3. Connections of an Asynchronous FIFO

Fig.3. shows that the control lines WRITE CLOCK and FULL are used to write data. When a data word is to be written into an asynchronous FIFO, it is first required to check whether there is space accessible in the FIFO. This is done by inquiring the FULL status line. If free space is indicated, the data word is applied to the data inputs and written into the FIFO by a clock edge on the WRITE CLOCK input.

In parallel manner, the control lines READ CLOCK and EMPTY are used to read data. In this case, the EMPTY status output has to be inquired before reading, because data can be read out only if it is stored in the FIFO. Then, a clock edge is applied to the READ CLOCK input, causing the first word in the data queue to appear on the data output. The shortcoming of a FIFO of this generous is that the status signals cannot be fully synchronized with the read and write clock.

III. FIFO THROUGH STATIC MEMORY

To pledge the drawback of an initial FIFOs, the architecture should no longer shift the data words through all memory places. The problem is solved by a circular memory with two pointers. In a circular FIFO conception, the memory address of the incoming data is in the write pointer. The address of the first data word in the FIFO that is to be read out is in the read pointer. After reset, both pointers specify the same memory place.

After each write operation, the write pointer is set to the next memory location. The reading of a data word sets the read pointer to the following data word that is to be read out. The read pointer continually follows the write pointer. When the read pointer extents the write pointer, the FIFO is empty. If the write pointer clasps up with the read pointer, the FIFO is full. Fig.4. illustrates the principle of a circular FIFO with two pointers.

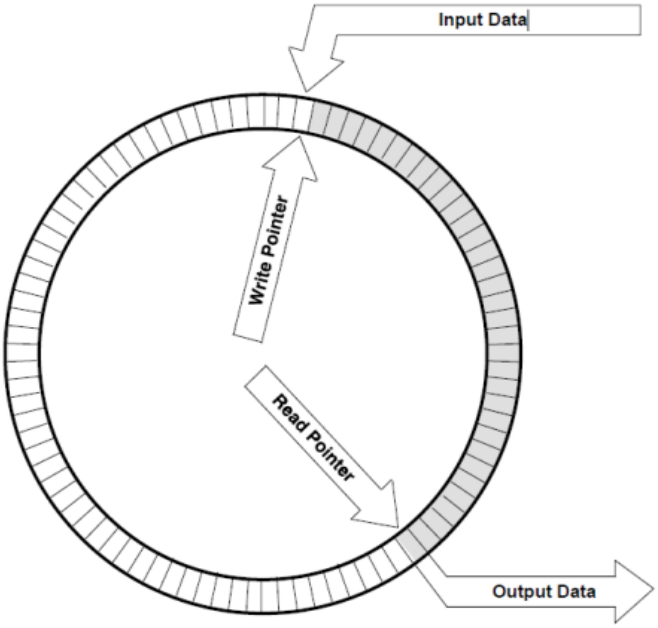

Fig.4. Circular FIFO with two pointers

A. CONVENTION OF FULL AND EMPTY CONDITIONS

The FIFO design in this concept guarantees that the empty flag will be produced in the read-clock domain to guard that the empty flag is detected directly when the FIFO buffer is empty and likewise full flag will be created in the write-clock domain to insure that the full flag is detected immediately when the FIFO buffer is full.

Here, in this FIFO design a status counter will be take care of FIFO full and empty conditions. This status counter will be incremented on every write and will be decremented on every read operation. When the status counter extents the maximum FIFO depth it will declare FIFO full signal and when its value is zero it will declare FIFO empty signal. This status counter also guarantee that there is no loss of data i.e. it ensures the "write before read conditions" and "read before wire condition". On reset the status counter is set to zero value and empty signal will be asserted.
The write enable and read enable will be reliant on write request and read request and empty and full bit.
Write enable == write req. and ~ (not of) full
Read enable == read req. and ~ (not of) empty

IV. MEMORY COUNTER

Memory is the process in which information is encoded, stored, and retrieved. In computing, memory refers to the physical devices used to store programs or data on a temporary or permanent basis for use in computer or digital electronic devices. Memory is an important category of Intelligent Property. Circuits are analog design that must be carefully designed. Apart from storage, counter is a device which stores the number of times a particular event or process has occurred, often in relationship to a clock signal.

It checks whether the queue is empty, full, increment or decrement. Counter is of many types such as binary/non-binary and asynchronous types. There is propagation delay when data is transmitted from one device to another due to capacitance of the device. This delay can be reduced by Decade counter, Modulus six counter etc., Further to reduce the delay transmission from one domain to another in memory, memory counter structure is used.

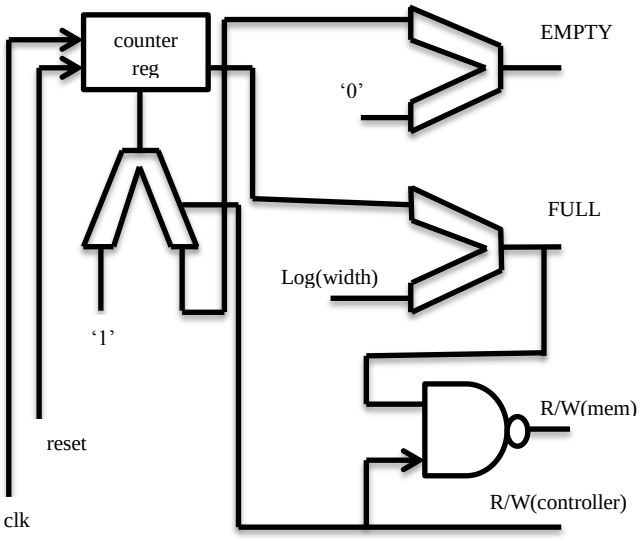

Fig.5. Block diagram of Memory Counter

Fig.5. illustrate the solution for high speed counter. The common approach is to add clock and reset options for the normal operation of the counter. The pointer in counter can be incremented when the data is written in stack of memory and it gets decremented when data is read from stack. The counter register is formed by adding D flip-flop of 8-bit, it inhibits passage of 8-bit data into one of the comparators.

There are two comparators, one has a constant value of "zero" and the other has constant value of "log(width)", these constant values are compared with the data values from counter register. It then intimate the Full and Empty condition of array of address.

When the comparator gives the output of "Empty", automatically Read/Write(controller) bit from controller reaches the multiplexer. Based on this input, counter gets incremented, thus the writing operation exists. In contrary, counter gets decremented during read operation. NAND gate gets the input from memory array. This means that the counter described in this area is used to make the memory access very quickly and it helps to reduce the delay propagation of data.

V. RESULTS

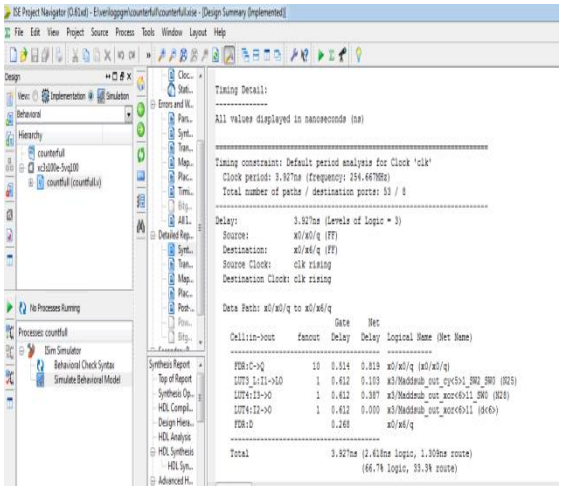

Fig.6. Timing analysis-1

Fig.6, gives the timing analysis for the data transmission from source x0/x0/q to the destination x0/x6/q, which acquire the total delay of 3.927ns.

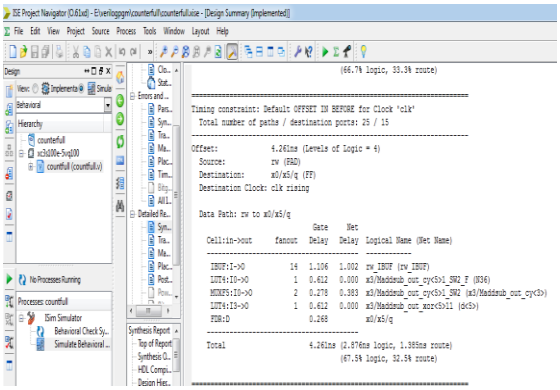

Fig.7. Timing analysis-2

Fig.7, shows the timing analysis for the data transmission from source rw to the destination x0/x5/q, thus the total delay obtained was 4.261ns.

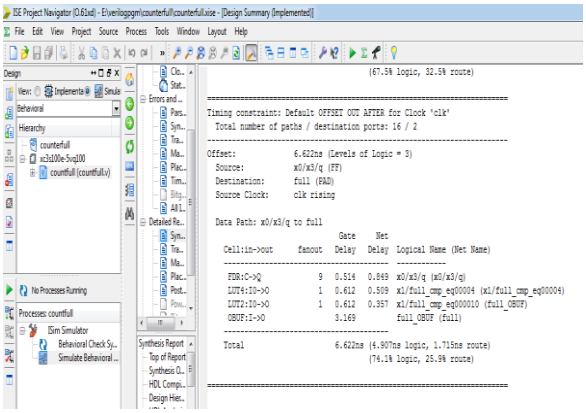

Fig.8. Timing analysis-3

Fig.8, shows the analysis of delay while the data transmitted from source x0/x3/q to final, here it acquire the total delay of 6.622ns.

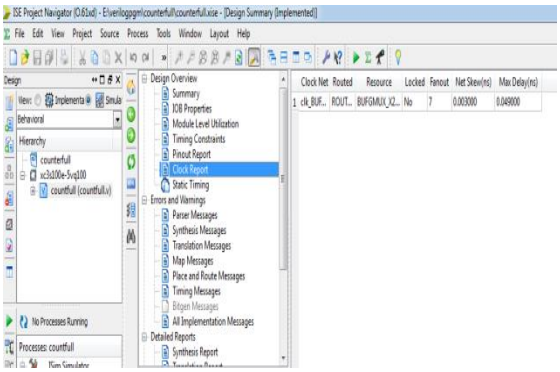

Fig.9. Clock analysis

Fig.9, exemplifies the clock report clearly.

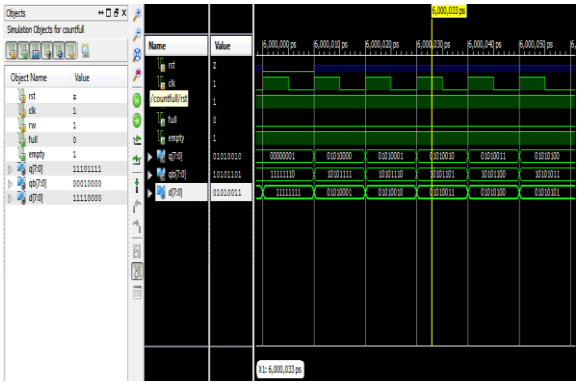

Fig.10. Simulation result of Memory Counter

Fig.10, shows the simulation output of Memory Counter. This briefly explains the increment and decrement of counter for every write and read processes.

VI. CONCLUSION

Asynchronous FIFO design requires careful design and attention from pointer creating techniques to full and empty generation. Elimination of important details will usually result in a design that can be easily verified but it may be wrong. Finding errors typically requires simulation of a gate-level FIFO design.

A safe FIFO can be interpreted by utilising memory-counter for the high speed memory access and efficiency can be increased.

REFERENCES

1. IEEE paper "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Design",

2. IEEE standard Verilog Reference Manual, IEEE standard 2001,

3. Design Compiler User Guide from Synopsys,

4. Verilog HDL by Samir Palnitkar.

