Int. J. Adv. Res. Sci. Technol. Volume 4, Issue 1, 2015, pp.253-256.

IJARST

International Journal of Advanced Research in Science and Technology

journal homepage: www.ijarst.com

ISSN 2319 – 1783 (Print)

ISSN 2320 – 1126 (Online)

Implementation of multichannel UART with FIFO

Haritha and D. P. Raju*

Department of ECE, Koushik college of Engineering, Visakapatnam, Andhra Pradesh, India.

****Corresponding Author's Email: haritapriya432@gmail.com***

A R T I C L E I N F O

Article history:

Received 20 Nov. 2014

Accepted 19 Dec. 2014

Available online 20 Jan. 2015

Keywords:

UART Universal Asynchronous Receiver Transmitter

FIFO First In First Out

Gray Counter

A B S T R A C T

To meet modern complex control systems communication demands, the paper presents a multi-channel UART controller based on FIFO (First In First Out) technique. The paper presents design method of asynchronous FIFO and structure of the controller. This controller is designed with FIFO circuit block and UART (Universal Asynchronous Receiver Transmitter) circuit block to implement communication in modern complex control systems quickly and effectively. Form the communication sequence diagrams, it is easily to know that this controller can be used to implement communication when master equipment and slaver equipment are set at different Baud Rate. It also can be used to reduce synchronization error between sub-systems in a system with several sub-systems. The controller is reconfigurable and scalable.

Introduction:

To meet modern complex control systems communication demands, we have to use a multi-channel UART controller based on FIFO technique. In this we have the design method of Asynchronous FIFO and structure of the controller. FIFO: FIFO means first in first out. In this we have two types: Synchronous FIFO, Asynchronous FIFO UART: UART means universal asynchronous receiver transmitter In this we have both receiving and transmitting. This controller can be used to implement communication between transmitter and receiver. This can also be used to reduce synchronization errors between sub-systems. This UART plays a important role in serial communication. Serial communication is used to reduce the distortion of a signal. This serial communication is used for longer distance transmission. It has simple structure.

Asynchronous FIFO: In asynchronous we are using two different clock cycles. Then there is no waste of time. That's why we are using this UART. In asynchronous FIFO,we have two clock domains, which are asynchronous to each other.the data values are written to a FIFO buffer from one clock domain and the data values are read from the same FIFO buffer from another clock domain. Using this asynchronous FIFO, we can send or receive data safely and quickly, because of asynchronous clock domains and time delay also not there.

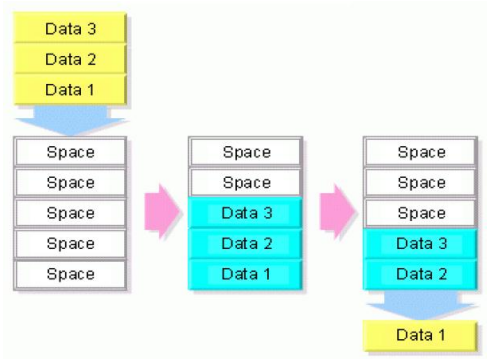
FIFO:

FIFO is a First-In-First-Out memory queue with control logic that manages the read and write operations, generates status flags, and provides optional handshake signals for interfacing with the user logic. It is often used to control the flow of data between source and destination. FIFO can be classified as synchronous or asynchronous depending on whether same clock or different (asynchronous) clocks control the read and write operations. In this project the objective is to design, verify and synthesize a synchronous FIFO using binary coded read and write pointers to address the memory array. FFIO full and empty flags are generated and passed on to source and destination logics, respectively, to pre-empt any overflow or underflow of data. In this way data integrity between source and destination is maintained. The RTL description for the FIFO is written using Verilog HDL, and design is simulated and synthesized using RTL compiler provided by Xilinx software.

In computer programming, FIFO (first-in, first-out) is an approach to 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 store data given from one clock domain and on request supplies with the same data to other clock domain following the first in first out logic. The clock domain that supplies data to FIFO is often referred as WRITE OR INPUT LOGIC and the

clock domain that reads data from the FIFO is often referred as READ OR OUTPUT LOGIC. FIFOs are used in designs to safely pass multi-bit data words from one clock domain to another or to control the flow of data between source and destination side sitting in the same clock domain. If read and write clock domains are governed by same clock signal the FIFO is said to be SYNCHRONOUS and if read and write clock domains are governed by different (asynchronous) clock signals FIFO is said to be ASYNCHRONOUS.

FIFO full and FIFO empty flags are of great concern as no data should be written in full condition and no data should be read in empty condition, as it can lead to loss of data or generation of non-relevant data. The full and empty conditions of FIFO are controlled using binary or gray pointers. In this report we deal with gray pointers only since we are designing ASYNCHRONOUS FIFO. The binary pointers are used for generating full and empty conditions for SYNCHRONOUS FIFO. The reason why they are used is beyond the scope of this document.

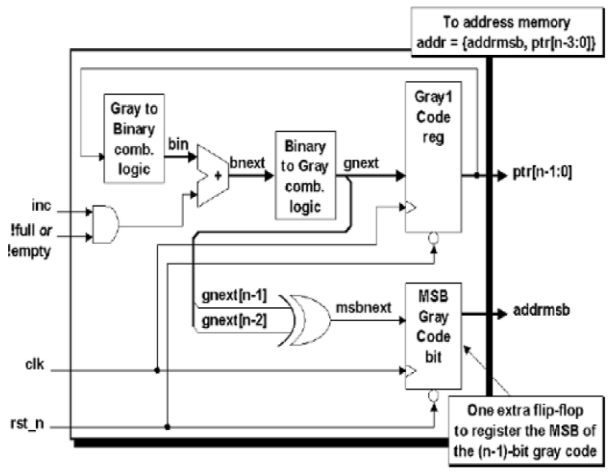


Data Flow through FIFO

Gray code counter:

Gray code counter basics:

The first fact about a Gray code is that the code distance between any two adjacent words is just 1 (only one bit can change from one Gray count to the next). The second fact is that most useful Gray code counters must have power-of-2 counts in the sequence.



Dual n bit gray counter

A dual n-bit Gray code counter is a Gray code counter that generates both an n-bit Gray code sequence (described in section 3.2) and an (n-1)-bit Gray code sequence. The (n-1)-bit Gray code is simply generated by doing an exclusive-or operation on the two MSBs of the n-bit Gray code to generate the MSB for the (n-1)-bit Gray code.

The binary-value incrementer is conditioned with either an "if not full" or "if not empty" test as shown in Figure 3, to insure that the appropriate FIFO pointer will not increment during FIFO-full or FIFO-empty conditions that could lead to overflow or underflow of the FIFO buffer. If the logic block that sends data to the FIFO reliably stops sending data when a FIFO full condition is asserted, the FIFO design might be streamlined by removing the full-testing logic from the FIFO write pointer. The FIFO pointer itself does not protect the FIFO buffer from being overwritten, but additional conditioning could be added to the FIFO memory buffer to insure that a write enable signal could not be activated during a FIFO full condition.

An additional "sticky" status bit, either ovf (overflow) or unf (underflow), could be added to the pointer design to indicate that an additional FIFO write operation occurred during full or an additional FIFO read operation occurred during empty to indicate error conditions that could only be cleared during reset.

A safe, general purpose FIFO design will include the above safeguards at the expense of a slightly larger and perhaps slower implementation. This is a good idea since a future co-worker might try to copy and reuse the code in another design without understanding all of the important details that were considered for the current design.

Implementation:

UART Transmitter:

The input-output signals of the transmitter are discussed here. The input signals are provided by the host processor, and the output signals control the movement of data in the UART. The architecture of the transmitter will consists of controller, a data register (XMT_datareg), a data shiftreg (XMT_shftreg), and a status register (bit_count) to count the bits that are transmitted. The status reg will be included with the data path unit the controller has the following inputs:

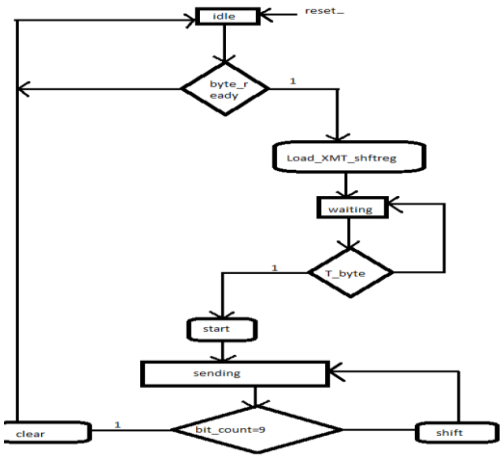
Byte_ready: asserted by host machine to indicate that Data_Bus has valid data

Load_XMT_datareg: assertion transfers Data _Bus to the transmitter data storage reg.

T_byte: assertion initiates transmission of a byte of a data, including the stop, start, and parity bits.

Bit_count: counts bits in the word during transmission.

The state machine of the controller forms the following output signals that control the data path of the transmitter.



Flow chart of Transmitter

UART Receiver:

The UART receiver has the task of receiving the serial bit stream of data, removing the start-bit, and transferring the data in a parallel format to a storage reg connected to the host data bus. The data arrives at standard bit rate, but it is not necessarily sync with the internal clock at the host of receiver. This issue of sync is resolved by generating a local clock at higher frequency and using it to sample received data in a manner that preserves the integrity of the data. In the scheme used here, the data, assumed to be in a 10-bit format, will be sampled at a rate determined by Sample clock, which is generated at the receivers host. The cycles of Sample clock will be counted to ensure that the data are sampled in the middle of a bit time. The sampling algorithm must (1) verify that a start bit has been received, (2) generate samples from 8 bits of the data, and (3) load the data on to the local bus.

Although a higher sampling frequency could be used, the frequency of Sample clock in this example is 8 times (known) frequency of a bit clock that transmitted the data. This ensures that a slight misalignment between the leading edge of a cycle of Sample clock and the arrival of the start-bit will not compromise sampling scheme, because the sample will still be taken within the interval of time corresponding to a transmitted bit. the arrival of a start-bit will be determined by successive samples of value 0 after input data go slow. Then 3 additional samples will be taken to confirm that a valid start-bit has arrived. Thereafter,8 successive bits will be sampled at approximately the centre of their bit times. Under worst case conditions of misalignment, the sample is taken a full cycle of Sample clock ahead of the actual center of the bit time, which is a tolerable skew.

The high level block diagram shows the input-output signals of a state-machimne controller that will interface with the host processor and direct the receiver sampling scheme. The state machine has the following inputs:

Read_not_ready_in: signals that the host is not ready to receive data

Serial_in: serial bit stream received by the unit

Reset_: active low reset

Sample_conter: counts the samples of a bit

Bit_counter: counts the bits that have been sampled

The sate machine produces the following outputs:

Read_not_ready_out: signals that the receiver has received 8 bits

Inc_sample_counter: incements Sample_counter

clr_sample_counter: clear Sample_counter

Inc_Bit_counter: incements Bit_counter

clr_Bit_counter : clear Bit_counter

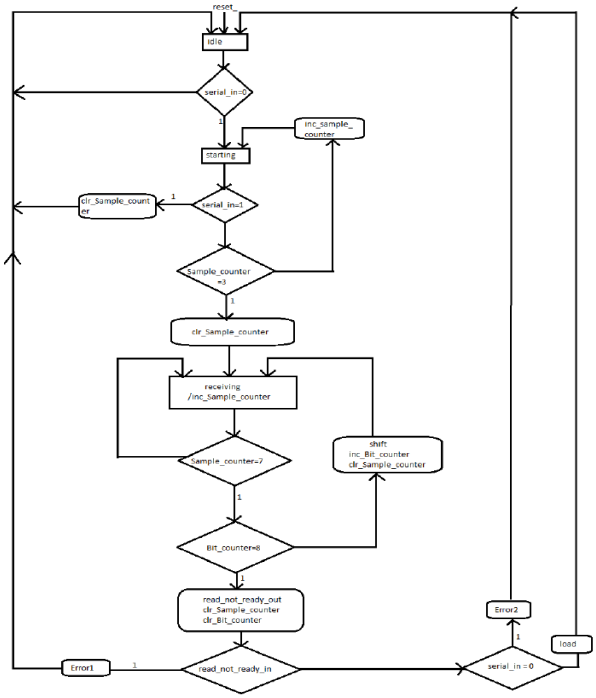
shift : causes RCV_shftreg to shift towards the LSB

load : causes RCV_shftreg to transfer the data to RCV_datareg

Error1 : asserts if host is not ready to receive the data after last bit has been sampled

Error2 : asserts if the stop-bit is missing.

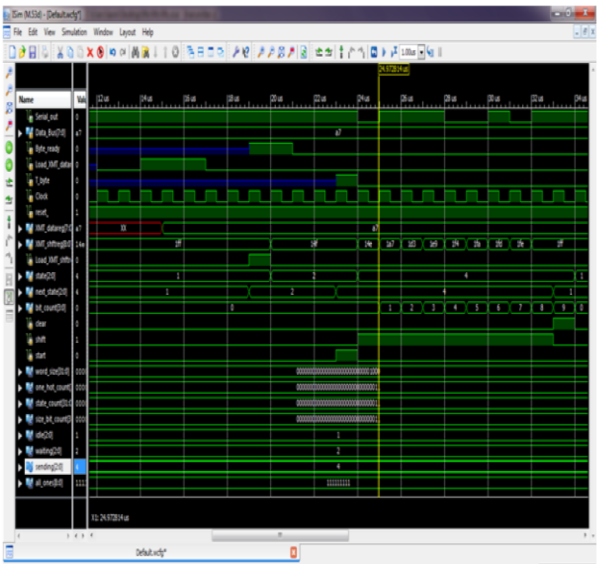
The ASM chart of a state machine controller is shown in figure. The machine has three states :idle, starting and receiving. transition between states are synchronized by Sample_clk.

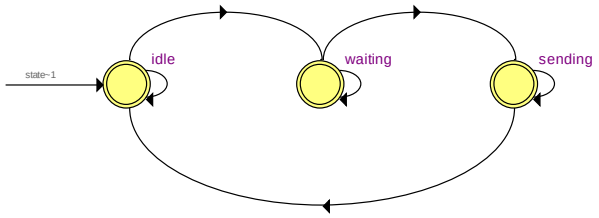


Flow chat of receiver

Simulation Results:

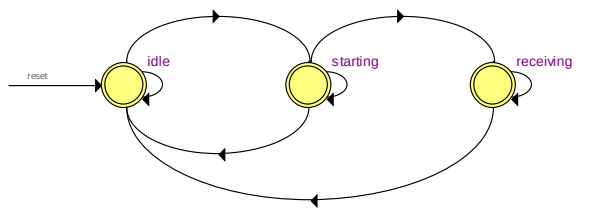
Transmitter output:





FSM of transmitter generated from simulation

Receiver output:



FSM of receiver generated from simulation

Conclusion:

This paper introduces a method to design asynchronous FIFO. And using synchronous FIFO technique implements a multi-channel UART controller based on SRAM with high speed and high reliability. The controller can be used to implement communications in complex system with different Baud Rates of sub-controllers. And it also can be used to reduce time delays between sub-controllers of a complex control system to improve the synchronization of each sub-controller. The controller is reconfigurable and scalable. The SRAM is reconfigurable, so the controller's fault is that it would be influenced by the radiation from surroundings and by short-time pulses easily.

About Authors:

Haritha is a P.G student of Department of ECE, Koushik college of Engineering, Visakapatnam, Andhra Pradesh, India. And D. P. Raju is presently working as Assistant Professor in the Department of ECE, Koushik college of Engineering, Visakapatnam, Andhra Pradesh, India.

References:

1. S. E. Lyshevski, "Control Systems Theory with Engineering Applications", Birkhauser Boston, 2001
2. L. K. Hu and Q.CH. Wang, "UART-based Reliable Communication and performance Analysis" , Computer Engineering, Vol 32 No. 10,May 2006, pp15-21
3. F.S. Pan, F. ZHAO, J. Xi and Y. Luo, "Implement of Parallel signal Processing System Based on FPGA and Multi-DSP", Computer Engineering Vol 32, No. 23, Dec 2006, pp247-249
4. C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design", SNUG San Jose 2002
5. C. E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons", SNUG San Jose 2002
6. Vijay A. Nebhrajan, "Asynchronous FIFO Architectures", www.eebyte.com
7. X., Yang, "Industrial Data Communication and Control Networks", Beijing: TUP, 2003.6
8. B. Zeidman, "Designing with FPGAs & CPLDs", CMP Books, 2002