Asynchronous FIFO Design

Introduction

Synchronous FIFO is a FIFO reads and writes in the same clock domain, while asynchronous FIFO reads and writes in two different clock domains for interconnect and data exchange between various design blocks and entities. In this design where a Gray code pointer is brought in rather than a traditional binary one. The design contains details of the asynchronous FIFO and its basic function mechanism.

Overview

The design is relatively simple compare to regular Verilog projects due to the fact that it's a practice project designed by an intern. The design consists of the following modules:

• fifo_top.v	top level module
• fifomem.v	memory for FIFO data transfer
• async_cmp.v	pure comparison logic for "full" and "empty" status bits
• rptr_empty.v	read pointer and empty-flag logic
• wptr_full.v	write pointer and full-flag logic

The diagram 1 below shows a block diagram of top-level for this asynchronous FIFO design.

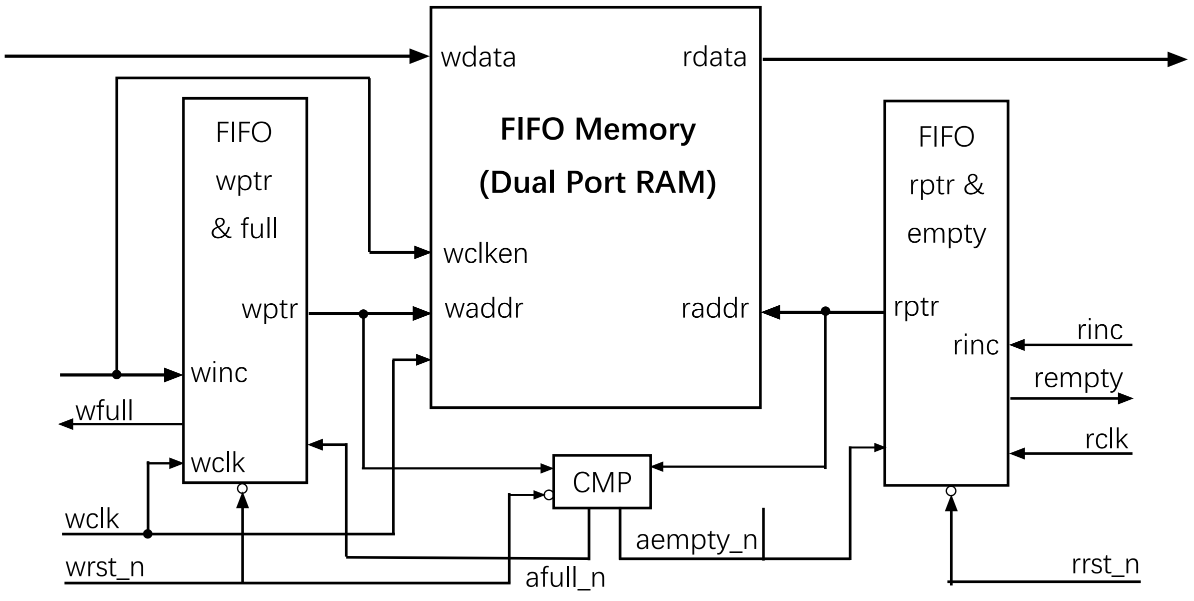


Diagram 1 FIFO top-level view

Signal Descriptions

The table 1 below indicates the input/output signal of the asynchronous FIFO.

Entity	Signal	Type	Description
FIFO Top-level	wdata	input	Serial data input of FIFO
	rdata	output	Serial data output of FIFO
	winc	input	Write pointer increment flag
	rinc	input	Read pointer increment flag
	wfull	output	Full flag stop writing
	rempty	output	Empty flag stop reading
	wrst_n	input	Write reset signal
	rrst_n	input	Read reset signal
	wptr	input	Write pointer function as address signal
	rptr	input	Read pointer function as address signal
CMP	afull_n	wire	
	aempty_n	wire	

Table 1 Signal List

Design Module Descriptions

fifo_top Module

This module connects all targeted signal in a FIFO design. Straight alter on input clock or output clock can fully indicate functions of an asynchronous FIFO, which is to complete data transfer between different clock domains.

fifomem Module

Functioning as the FIFO memory buffer, can be accessed by both read and write clock domains. Here is just one of the compatible memory designs for a FIFO, other memory style can also be adapted in a FIFO design.

async_cmp Module

This module has the function of comparing the asynchronous FIFO pointer, which is used to generate control signals that control assertion of "full" and "empty" status bits. This module only contains combinational logic for the comparison of pointers, no sequential logic included.

rptr_empty Module

The name of this module is an acronym of "read pointer empty". This module is mostly synchronous to the read-clock(*rclk*) domain and also contains the FIFO read pointer and empty-flag logic. *aempty_n* signal (as an input to this module) is synchronous to the *rclk* domain. In this module where *aempty_n* is asserted, *rptr* always incremented before that and the de-assertion of *aempty_n* only happens when *wptr* increments, what is also asynchronous to *rclk*.

wptr_ full Module

Similarly, this module is basically synchronous to the read-clock domain and contains the FIFO write pointer and full-flag logic. Also, the *afull_n* signal is synchronous to *wclk*, since *afull_n* can only be asserted when *wptr* incremented, and de-assertion of the *afull_n* signal happens when *rptr* increments.

Design Flow

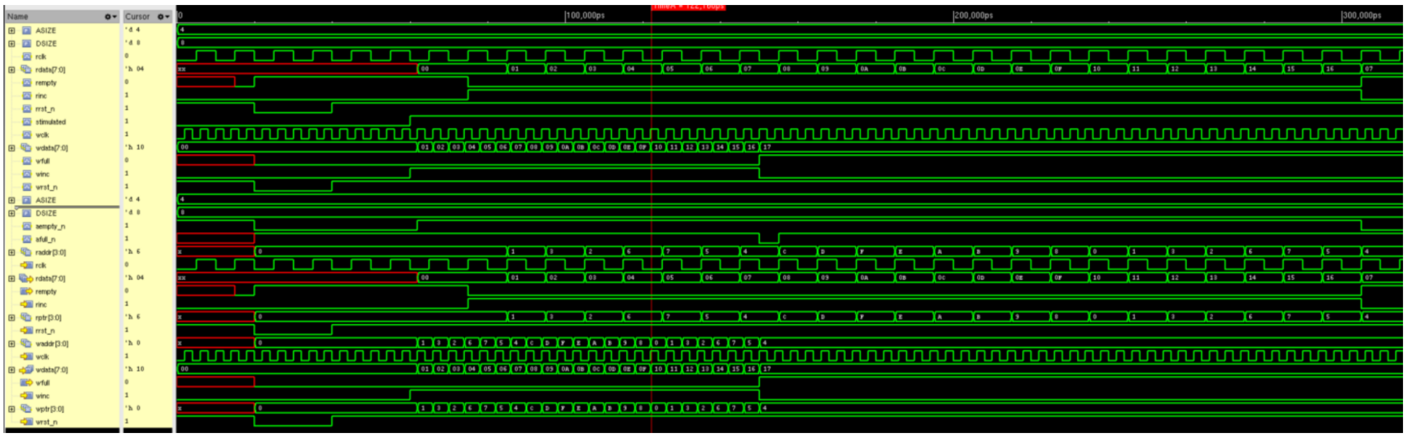
Cadence design tools are used for synthesis and simulation. File structure of this design is simple since the design itself is quite straight forward. Waveform was viewed in SimVision. Linux version CentOS 6.10

Timing Diagrams

The following timing diagrams show the major timing scenarios in simulation.

The input data line writes a series of data (here as an incremental sequence) to the memory with Gray coded addresses, and reads from memory with the same order from the write addresses order. Here from the waveform diagram we can tell that the writing procedure immediately stopped right after the "full" flag was pulled high, which was under the control of the CMP module that compares write pointer and read pointer that generates *afull_n* flag and *aempty_n* flag, controlling the increment of read/write pointers. Same situation happened in the reading procedure that reading from memory was immediately suspended when "empty" flag was pulled high.

Figure 1 Overall view of fixed pattern FIFO stimulation



While the fixed pattern stimulation above reveals the expected input and output from design, randomize stimulate is still inevitable for a thorough design simulation in case some unexpected results generated.

Asynchronous FIFO test targets

In this case, random input value was implemented to verify if this FIFO design can tackle the random input scenario.

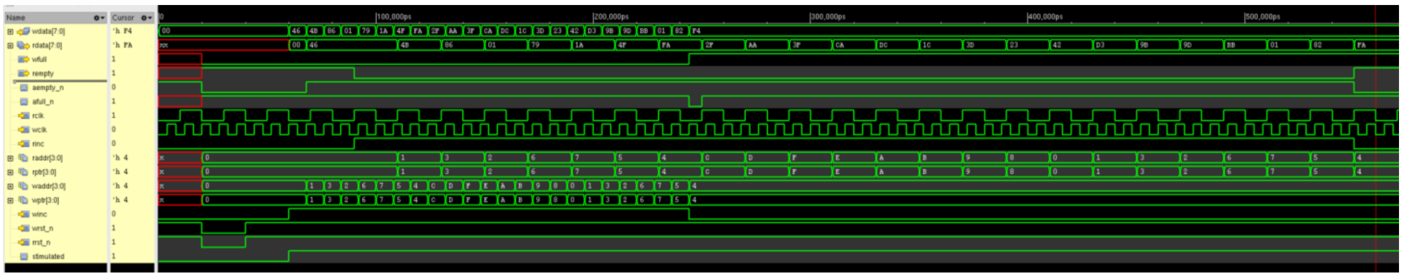


Figure 2 Random Input

Test of FIFO reads and writes with different clock frequency, the following figure is a test of a commonly used on board oscillator frequency ratio.

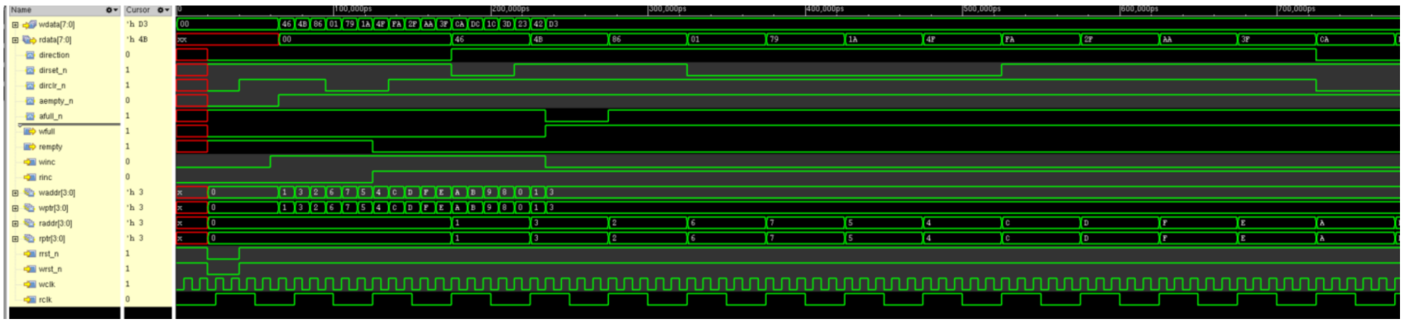


Figure 3 Test between various clock frequency ratio

In this stimulus case, “full” flag is force high to verify whether this FIFO design responds as expected that “full” flag controls data input and writing to the memory. And the result is promising.

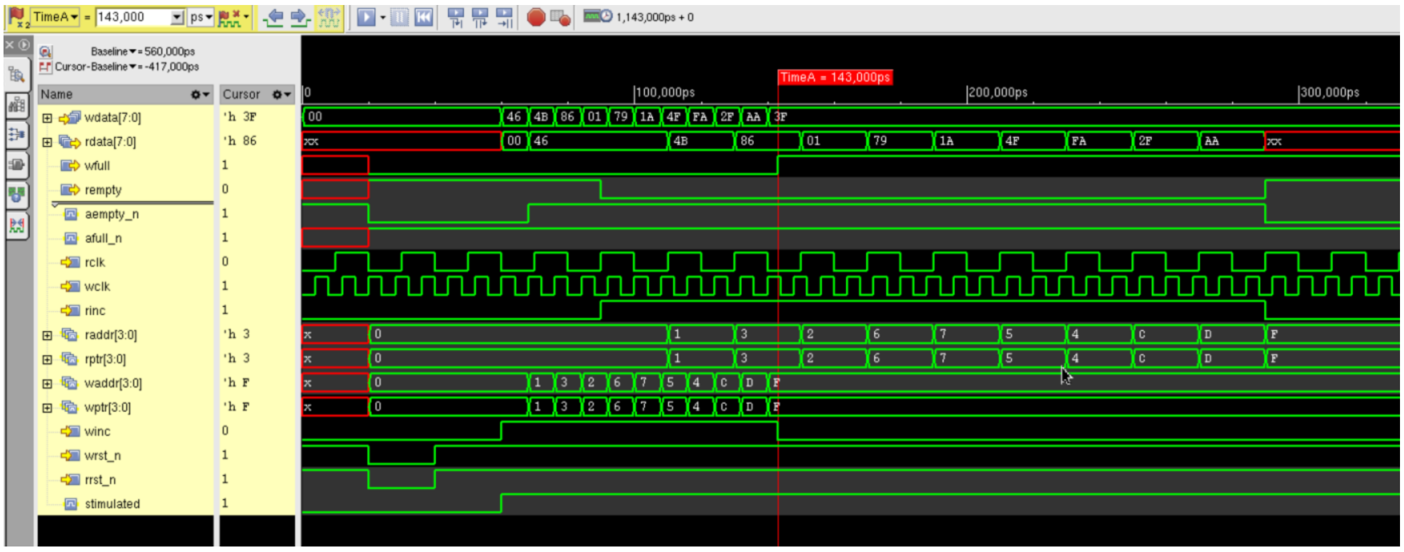


Figure 4 Test of manually toggled “full” flag

Reset signal testing. After reset signal implemented, default signal value is correct and FIFO working properly. From the waveform we can tell that the reset signal successfully controlled the expected reset behavior of this FIFO design.

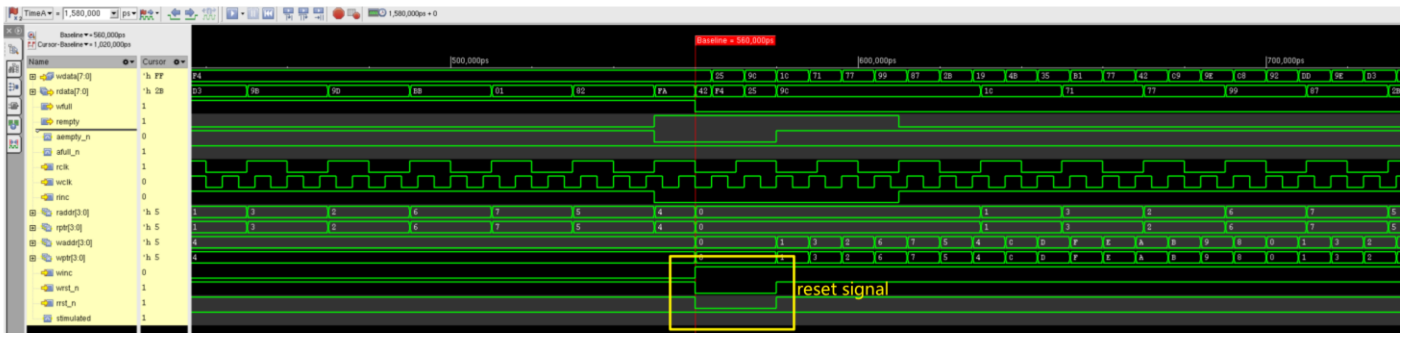


Figure 5 Reset test

One last thing about testing that is when the reading frequency if higher than the writing frequency, in this case shown in the figure below, a FIFO depth of "1" will be sufficient since there will not be any data loss when the reading is faster than writing (Straight data transfer from one register to the other). If not so, circumstance in figure 5 will occur, since reading operation happens when FIFO is not empty and previous data was not fully covered by new data written. In this case, just straight connect the transfer or manually set the depth of FIFO to "1".

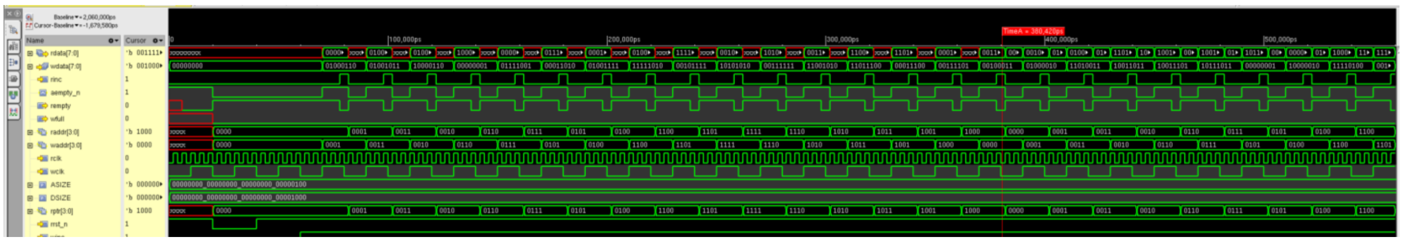


Figure 5 Reading clock frequency higher than writing clock frequency

Design Hierarchy Summary

Table 2 lists the resource utilization in this FIFO design and shows the design hierarchy summary.

	Instances	Unique
Modules	6	6
Registers	19	19
Scalar wires	13	-
Vectored wires	11	-
Always blocks	11	11
Initial blocks	1	1
Continuous assignments	11	12
Pseudo assignments	8	8

Table 2 Design Hierarchy Summary

Revision History

Date	Version	Alteration Summary
SEP20 2022	0.1	Project Starts
SEP22 2022	0.2	Project fully tested and closed