A synthesizable asynchronous FIFO design and simulation in Verilog HDL for clock domain crossing

Juan Pablo Salvatierra
Physics Department
Università degli Studi di Torino
juan.salvatierra@edu.unito.it

***Abstract*—This paper focus on an asynchronous FIFO memory design proposed by Clifford E. Cummings [1]. Here implementation is carried out in Verilog HDL, simulated and synthesized using Xilinx Vivado Design Suite- HLx Editions and a Arty-A7 FPGA board as intended target device. The aim of this work is to discuss implementation and simulation results, in particular how status flags are generated and their reliability.**

***Index Terms*—FIFOs, CDC-FIFO, dual clock FIFO, Clock Domain Crossing, Verilog HDL**

I. Introduction

Whenever communication between two digital systems running at different clock speed and/or phase is needed, a clock domain crossing interface must handle data transition between domains. Among the available solutions asynchronous FIFOs are a very popular and efficient tool, especially when dealing with large control buses or data transfer.
In figure 1 a generic schematic of a dual clock FIFO I/O ports is shown.
The write side (i.e. the transmitter) runs at **w_clk** frequency and sends data to be written into the FIFO through **w_data** bus, **w_en** allows write operation. The read side (i.e. the receiver) runs at **r_clk** frequency and is fetching data from the FIFO through **r_data** bus, when **r_en** allows read operation. The others signals, **w_full** and **r_empty** are status signals the FIFO is sending to the external world. These signals must be properly managed by the transmitter and the receiver in order to avoid overflow (writing into a full FIFO) and underflow (reading from an empty FIFO) conditions, which would lead to data loss and data corruption.
A reliable implementation of the status flags is thus crucial, and it is also a critical part of an asynchronous FIFO's design, especially for what concern generation of the full flag.

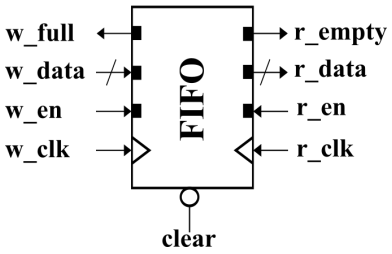

Fig. 1. Dual clock FIFO

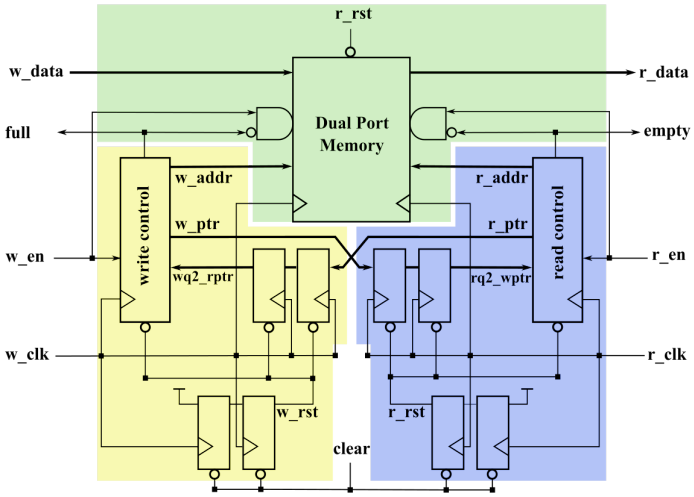

Fig. 2. Design overview

II. Implementation

A. Overview

A schematic of the proposed design is shown in figure 2 and it is widely inspired to the one proposed by Cummings in [1], with minor changes on the read operation and the introduction of a reset synchronizer exploiting techniques illustrated in [2] to allow asynchronous assertion and synchronous deassertion in both write and read domains.
The core of the design is a dual port memory element in which write and read operations are synchronous to their respective clocks and only enabled if w_en && !full / r_en && !empty respectively. The write logic and read logic modules contains: a reset synchronizer, a two stages synchronizer in which the actual clock domain crossing occurs and a control unit in which memory addresses, pointers and status flags are

generated.
The design is parametrized by the address and data number of bits, and allows only implementation of FIFOs with depth of 2^n, where n is the address size.

B. *Control units - gray coded pointers*

To address the FIFO memory and generate the pointers the configuration shown in figure 3 is used in both write and read domains. The memory address is generated by directly taking the n LSBs of a n+1 bits binary counter, which only increments if writing/reading is enabled and the FIFO is not full/empty. The extra bit is needed to test for full and empty conditions. The next value of the binary counter **bnext [n:0]** is basically the pointer, however it is necessary to convert the binary pointer in gray code because it must be sent to the other clock domain through a two stage synchronizer. By using gray coded pointers only one bit flip per clock cycle can occur and the chance of going through metastability in the synchronization stage is highly reduced.

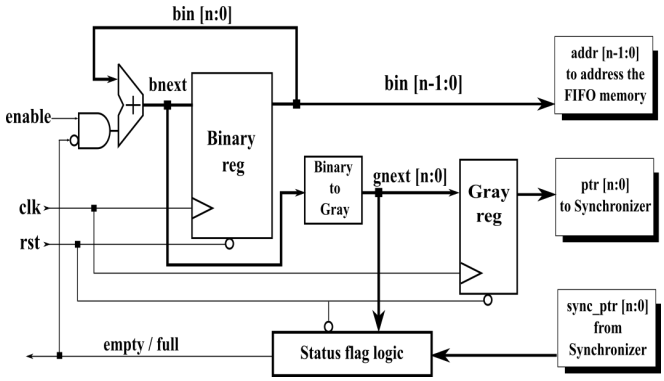

Fig. 3. Control unit schematic

C. *Full and empty conditions*

The proposed design assumes that full flag is generated in the write clock domain and empty flag is generated in the read domain, in order to insure their immediate detection in the respective domains.
The empty condition is the easier to figure out: the FIFO is empty when the read pointer (in the read domain) catches up the write pointer (in the write domain) and both have wrapped around the same number of times, therefore all their bits are equal. To realize this condition the comparison is made between the synchronized write pointer and the read_gnext bus.
To test for the full status the synchronized read pointer is compared against write_gnext, and three conditions must be met for the full status to be true:

- The 1^{st} MSB are unequal because the write pointer has wrapped one more time than the read pointer)
- The 2^{nd} MSB must be opposite too
- All the other bits must be equal

The reason for the second condition lies in the fact that gray codes are specular with respect to the midpoint, and that only the n LSBs of the gray coded pointer are related to the memory address, which is binary coded in this design. To best clarify the concept see the figure below.

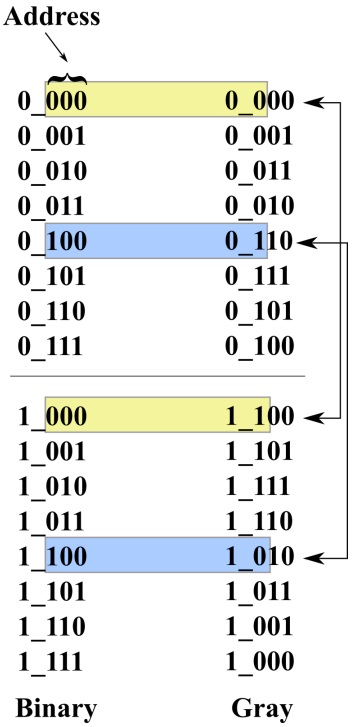

Fig. 4. When the three conditions for full status are met the gray coded pointers addresses the same memory location

D. *Reset*

The proposed design has different active low resets signals for each domain which are intended to be simultaneously and asynchronously set, while removed synchronously with their respective clocks [1]. This can be achieved by using a master asynchronous reset signal (clear in the schematic) and a couple of 2 Flip Flops synchronizers as shown in figure 2.
Upon clear assertion **w_rst** and **r_rst** are also activated, as a consequence all the the synchronizing registers and write/read logic are simultaneously resetted. The empty flag is asserted and the output data bits (r_data) are set to zero. When clear is deasserted after two rising clock edges of their respective domain the reset signals are deasserted.

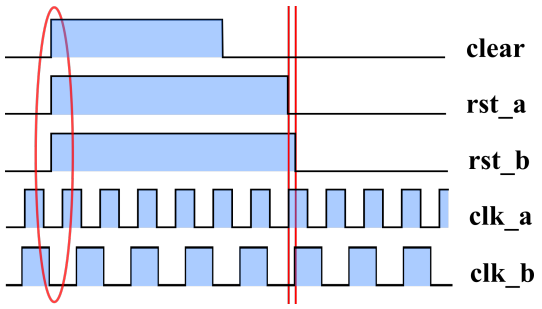

Fig. 5. Reset synchronization

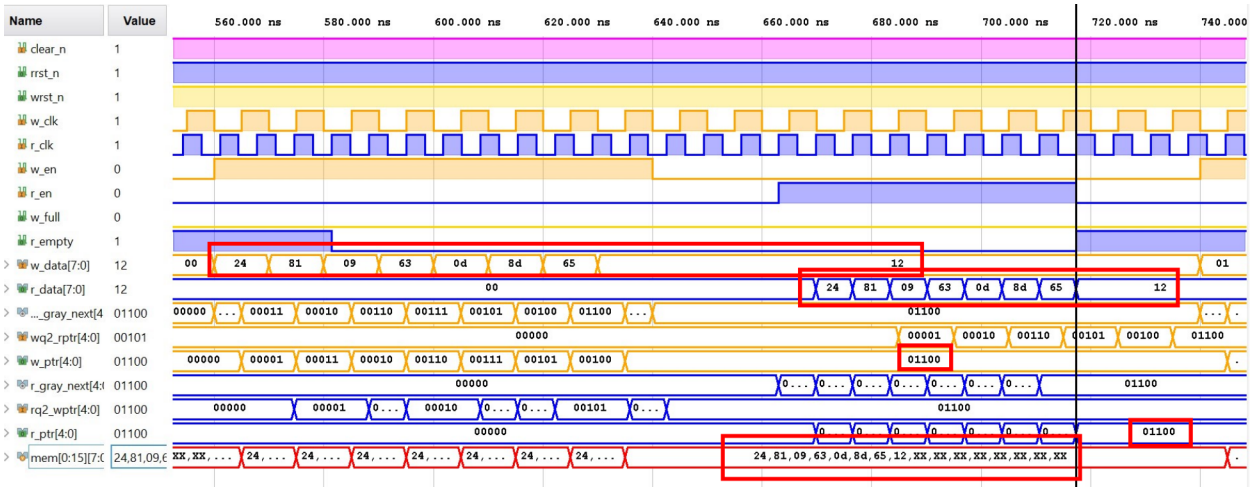

Fig. 6. Write 8 data into the FIFO read 8 data from the FIFO. Highlighted: r_data and w_data, memory content and pointer values at which the empty flag is asserted

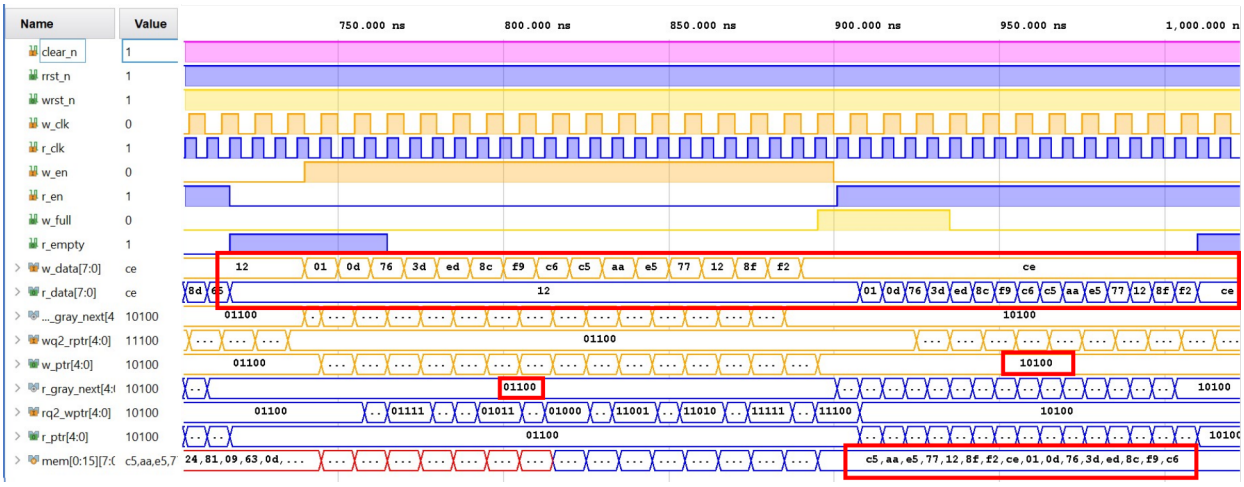

Fig. 7. Write until full read until empty. Highlighted: r_data and w_data, memory content and pointer values at which the full flag is asserted

III. Simulation results

To carry out the simulation the following parameters has been used:

- **ADDRESS_SIZE** = 4 (DEPTH=16)
- **DATA_SIZE** = 8
- **w_clk** period = 10 ns (100 MHz)
- **r_clk** period = 6.76 ns (147.93 MHz)

The testbench which has the following steps:

- Perform an initial clear assertion
- Write a bunch of data into the FIFO
- Read the data from the FIFO
- Write until full
- Read until empty

Simulation results are also compared with the ones obtained using the same testbench, and the FIFO IP Core provided by Xilinx and compiled with (almost) equal parameters to the one implemented in this paper.

From the above figures it can be seen that first in first out queueing is respected, all the FIFO locations are writable (this is not always true,like in the Xilinx's FIFO IP Core), reading and writing follows the frequency of the respective clock and the status flags are asserted exactly when the pointers points to the same location of the FIFO (i.e. the previously seen conditions are met).

Nonetheless, flags deassertion is quite pessimistic: it takes a few clock cycle of the respective domain to detect the change in the opposite domain's pointer.

Although not shown here, it has been checked at simulation level that trying to write into a full FIFO is not allowed (memory content do not change even if w_data keeps changing and write enable is high), and that trying to read from an empty FIFO is not allowed too (r_data do not change and it's fixed at the last read value, even if read enable is high).

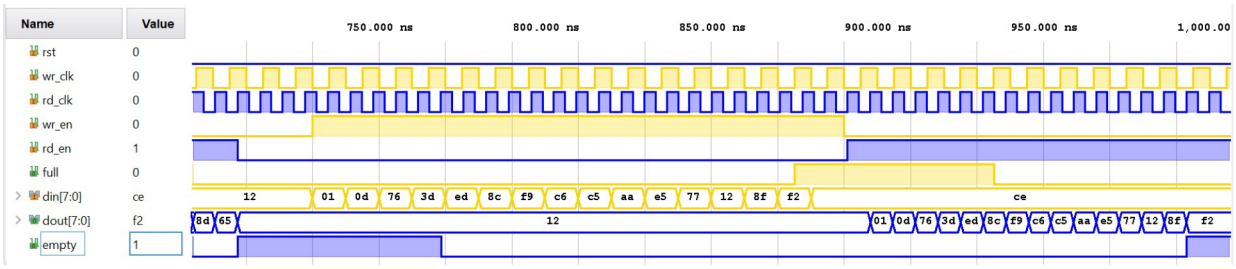

Fig. 8. Simulation results using the Xilix's IP core

The above figure 8 refers to simulation using the FIFO IP core. The main difference between the proposed design and the compiled IP core behaviours are two: the asynchronous reset is active high and there are only 15 writable location (depth was selected 16). In fact, full flag is asserted when the 15^{th} word is written and even if data in keep changing the value is not written into the memory. Status flags deassertion are also pessimistic.

IV. SYNTHESIS

Synthesis has been carried out using Vivado's synthesis tool,using both block ram and distributed ram for the dual port memory.Then post synthesis STA (static timing analysis) has been performed. Post synthesis schematics and utilization reports are shown in the following figures and tables.

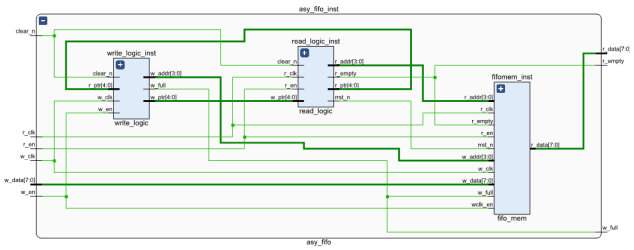

Fig. 9. Post synthesis block diagram

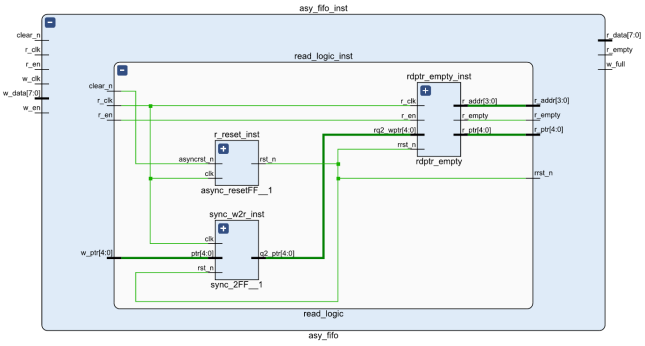

Fig. 10. read logic block

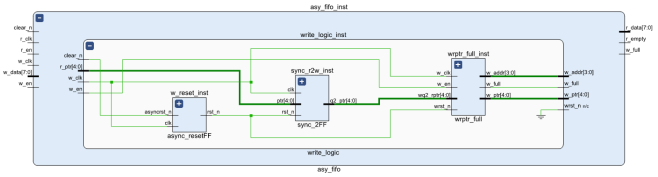

Fig. 11. write logic block

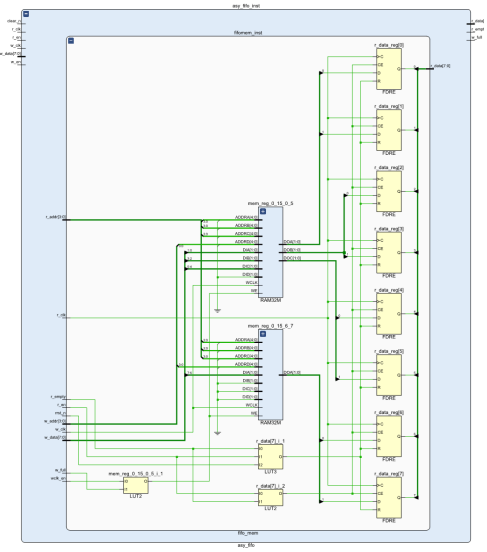

Fig. 12. Synthesis using distributed RAM

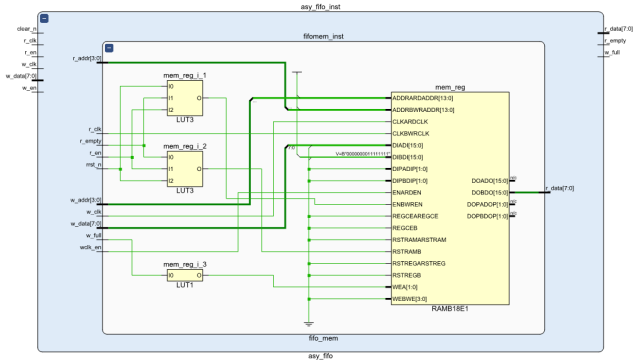

Fig. 13. Synthesis using block RAM

TABLE I
RESOURCES UTILIZATION USING BLOCK RAM

Site Type	*Used*	*Fixed*	*Available*	*Util%*
Slice LUTs	25	0	20800	0.12
LUT as Logic	25	0	20800	0.12
LUT as Memory	0	0	9600	0.00
Slice Registers	46	0	41600	0.11
Register as Flip Flop	46	0	41600	0.11
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00
Block RAM TILE	0.5	0	50	1.00
RAMB36/FIFO	0	0	50	0.00
RAMB18	1	0	100	1.00

TABLE II
FPGA PRIMITIVES USING BLOCK RAM

Ref Name	*Used*	*Functional Category*
FDCE	45	Flop & Latch
IBUF	13	IO
OBUF	10	IO
LUT6	8	LUT
LUT3	8	LUT
LUT1	7	LUT
LUT5	6	LUT
LUT4	4	LUT
BUFG	2	Clock
RAMB18E1	1	Block Memory
FDPE	1	Flop & Latch

TABLE III
RESOURCES UTILIZATION USING DISTRIBUTED RAM

Site Type	*Used*	*Fixed*	*Available*	*Util%*
Slice LUTs	33	0	20800	0.16
LUT as Logic	25	0	20800	0.12
LUT as Memory	8	0	9600	0.08
LUT as Distributed RAM	8	0	-	-
LUT as Shift Registers	0	0	-	-
Slice Registers	54	0	41600	0.13
Register as Flip Flop	54	0	41600	0.13
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

V. STATIC TIMING ANALYSIS

Timing analysis is carried out without timing violations if any path between the clocks is set false, or equivalently if the constraint set _clock_groups -asynchronous is used with w_clk and r_clk. However, this approach may be too relaxed and may result in an underconstrained CDC circuit [4].
Following suggestions reported in [4], [5] the only constraints here used to carry out static timing analysis are of the type set_max_delay -datapath only, from the gray pointer registers in the source domain to the first flip flop of the synchronizer in the destination domain, with the delay value set to the destination clock period. This approach led to a post synthesis timing report without violations when using block RAM based FIFO and unrelated clock periods.
However, when using distributed RAM setup violations with a -2.060 ns[1] WNS (worst negative slack) on the path between w_clk and the r_data output registers occurred if the clock periods were totally unrelated. If the clock periods were related (e.g. 100 MHz , 50 MHz) setup violations in the previously mentioned path were not observed by the tool.

TABLE IV
FPGA PRIMITIVES USING DISTRIBUTED RAM

Ref Name	*Used*	*Functional Category*
FDCE	45	Flop & Latch
IBUF	13	IO
RAMD32	12	Distributed Memory
OBUF	10	IO
LUT6	8	LUT
FDRE	8	Flop & Latch
LUT3	7	LUT
LUT5	6	LUT
LUT1	6	LUT
RAMS32	4	Distributed Memory
LUT4	4	LUT
LUT2	2	LUT
BUFG	2	Clock
FDPE	1	Flop & Latch

VI. CONCLUSIONS

The asynchronous FIFO design proposed in this paper is a general but safe design, as it respect two golden rules of FIFOs: do not allow write when full, do not allow read when empty, or in other words it shouldn't go in overflow or underflow conditions.
Simulation results shows the expected behaviour and are comparable with the ones obtained using the Xilinx IP Core. Cummings also proposed another FIFO design in [3] , which involves asynchronous pointer comparison, however Cummings himself discourage the reader from using it and go instead for the design described in [1]. Including this FIFO in a design may require modifications and/or added features such as almost empty and almost full flags, depending on clock frequencies more synchronization stages in the CDC might be also needed to avoid metastability propagation. Properly constraining an asynchronous FIFO is hard and also design dependent, what is here described should be a valid (perhaps not exhaustive) guideline for FIFOs based on two stage synchronizers as CDC circuits.

REFERENCES

[1] Clifford E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," *SNUG 2002 2 (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 and Don Mills, "Synchronous Resets? Asynchronous Resets? I am So Confused! How Will I Ever Know Which to Use?," *SNUG 2002 2 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers,* March 2002, Section TB2, 1st paper. Also available at www.sunburst-design.com/papers/

[1] usign the same clock frequencies used for simulation

[3] Clifford E. Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons,” *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers,* March 2002, Section TB2, 3rd paper.Also available at
www.sunburst-design.com/papers/

[4] https://forums.xilinx.com/t5/Vivado-TCL-Community/set-clock-groups-and-constraint-propagation/td-p/407547

[5] https://forums.xilinx.com/t5/Xilinx-IP-Catalog/Constraining-asynchronous-FIFO/td-p/734824