Asynchronous Dual Clock FIFO Design (Self-Project)

Devarshi Dhoble, IIT Bombay

Introduction :

The word FIFO stands for First In First Out. The concept of FIFO is used in many branches of life, namely, accounting, stocks rotation and memory management. In memory management, FIFO is used to capture data, essentially in applications where the rate at which it is captured(written) is different from the rate at which it is accessed(read). In many applications where a peripheral device connected to a processor captures and sends data to the processor at a rate, either more or less than the processor speed. For example, normal cameras "record" videos at 60 frames per second. The rate at which the camera sends data to the processor to "process" the data and store in the memory is significantly lower than the processor speed itself, hence the data sent by camera is essentially "asynchronous". For such applications, data needs to cross clock domains from the camera's clock domain to the processor's clock domain. This is done using an asynchronous FIFO of pre-defined "depth" and "width".

While the concept of FIFO is straight forward, the hardware implementation of FIFO is quite a challenge. Mainly because the read and write rates are different. To ensure the FIFO doesn't overflow or underflow and to ensure data integrity, the read domain pointers need to synchronise to write domain and vice-versa to generate a few status signals. These signals can be "full", "empty", "almost full" and "almost empty". The aim of this project is to design a dual clock FIFO with "full" and "empty" status signals and understand the working each block involved in the operations.

Specifications & Design Overview :

- FIFO depth: **16**
- FIFO width: **8 bits**
- Read frequency: **25MHz**
- Write frequency: **50MHz**
- Status signals: **full & empty signals**
- **Separate Synchronous reset** signals for read domains and write domain.
- Generic design, easily scalable with minimal changes in source code.
- For different port sizes, 8-bit write & 4-bit read. **A complete read operation will be executed in 2 read clock cycles**.

Main Block Diagram :

Same Port Sizes :

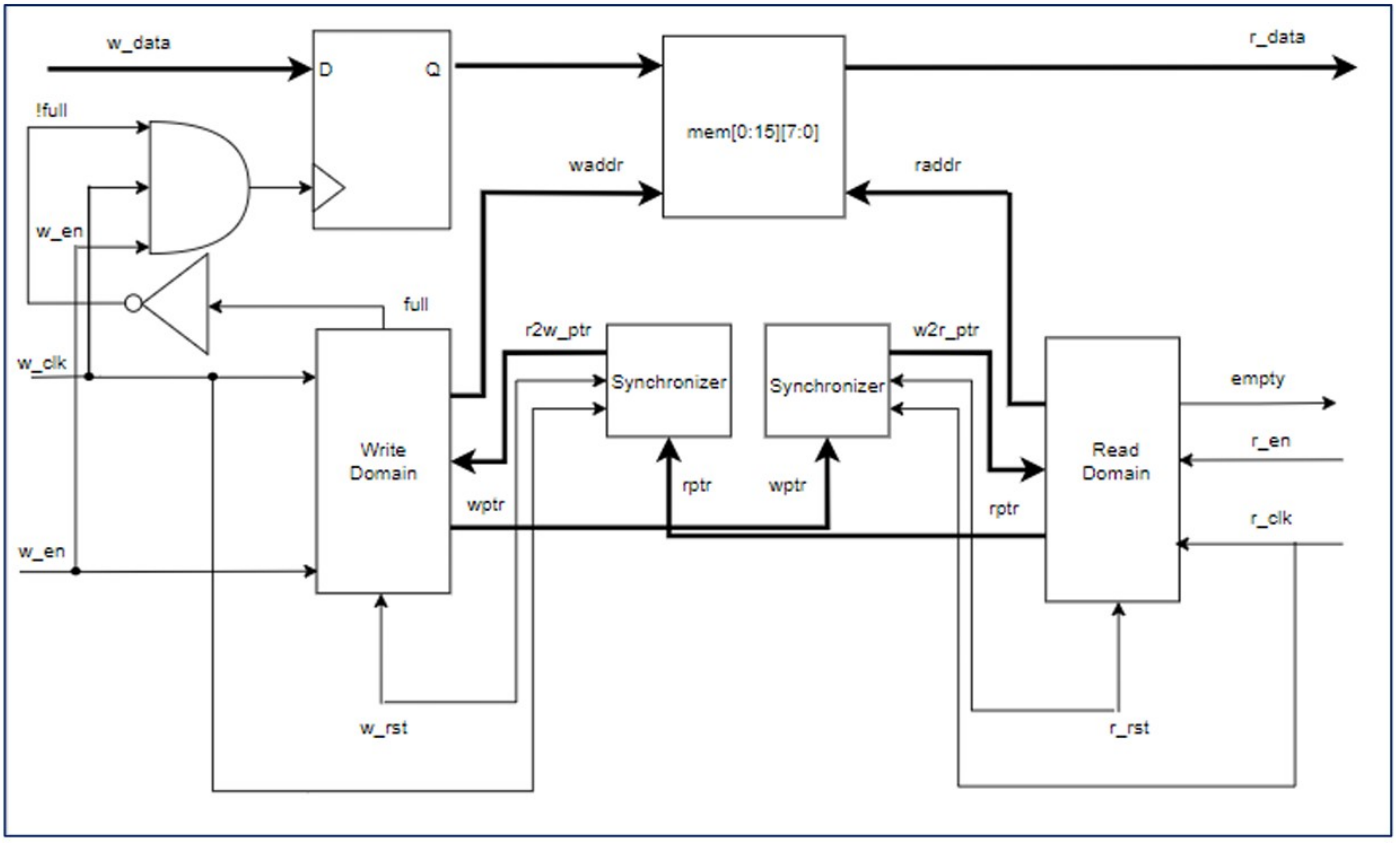

Figure 1: Block Diagram for Same Port Sizes

1. The buffer that will store the write data is the mem block. It is an array type of data structure which has 16 addressable memory locations, and each location has 8 bits of data stored.
2. The address to which data is to be written is generated by the write domain block which primarily generates 3 outputs. The full status signal which controls whether the data can be written to the mem block or not.
3. The waddr signal which controls the address in which the data is written, and the wptr signal which is synchronised into the read domain to produce empty signal. The inputs to this block are w_clk which is the write clock, w_rst which is the write reset signal, w_en which is the write enable signal which signifies that the data present on the data bus is valid, and r2w_ptr which is the synchronised read pointer which is used for generating full status signal. The read domain block is like the write domain block.

Different Port Sizes :

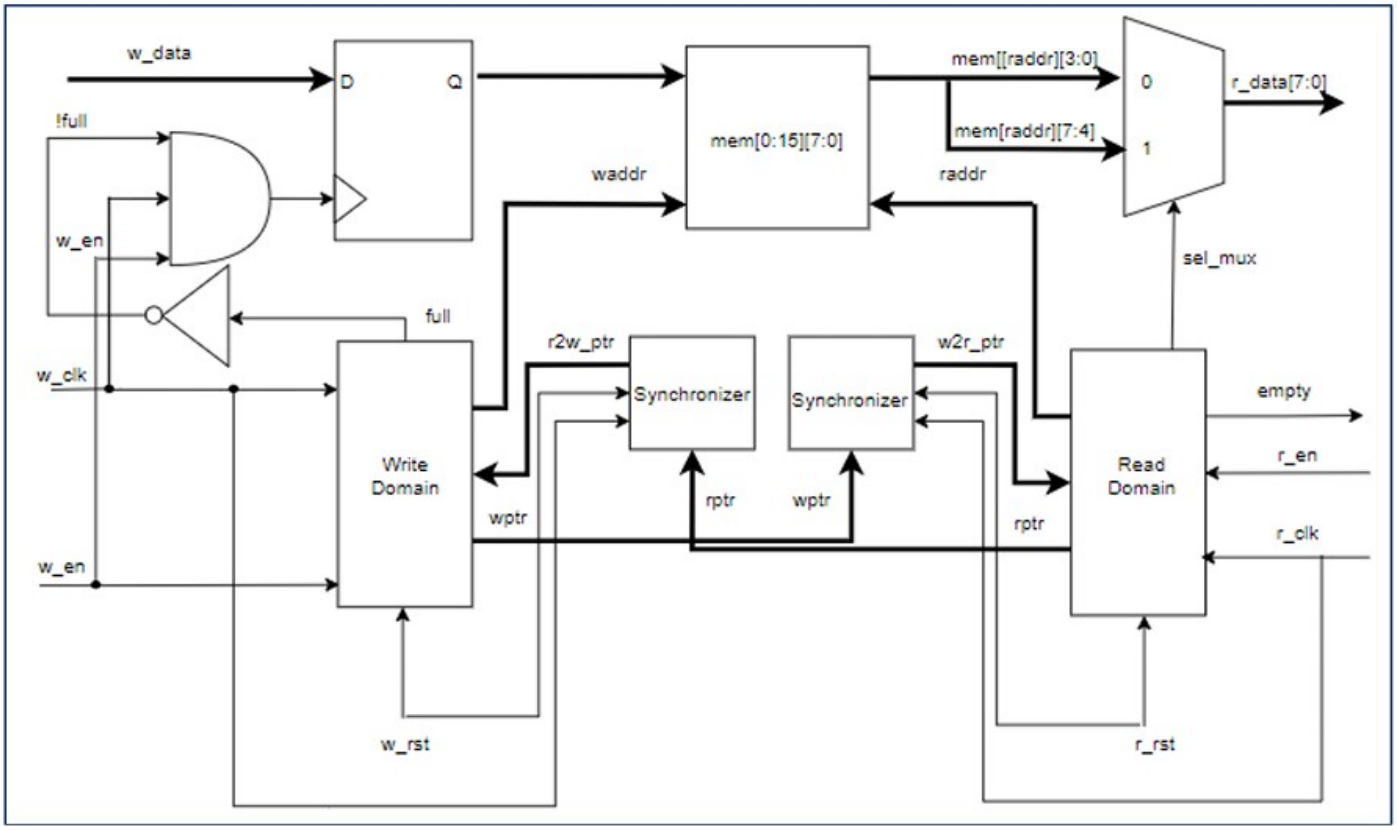

Figure 2: Block Diagram for Different Port Sizes

- For different port size design, write bus is 8bit but the read bus is only 4bit. Evidently, only the read domain circuitry should be modified to accommodate this feature.
- To accommodate reading only 4 bits in one clock cycle or 8bit data in two clock cycles, the important aspect is to hold the read address raddr at the same value for two clock cycles.
- To do this, an output sel_mux is generated which will toggle at every read clock edge. When sel_mux is '0' the raddr signal is not incremented, making the raddr signal unchanged for two read clock cycles. The same signal is used to multiplex the Least Significant 4 bits and Most Significant 4 bits of r_data, which are read in 2 clock cycles.

Individual Blocks :

Write Domain

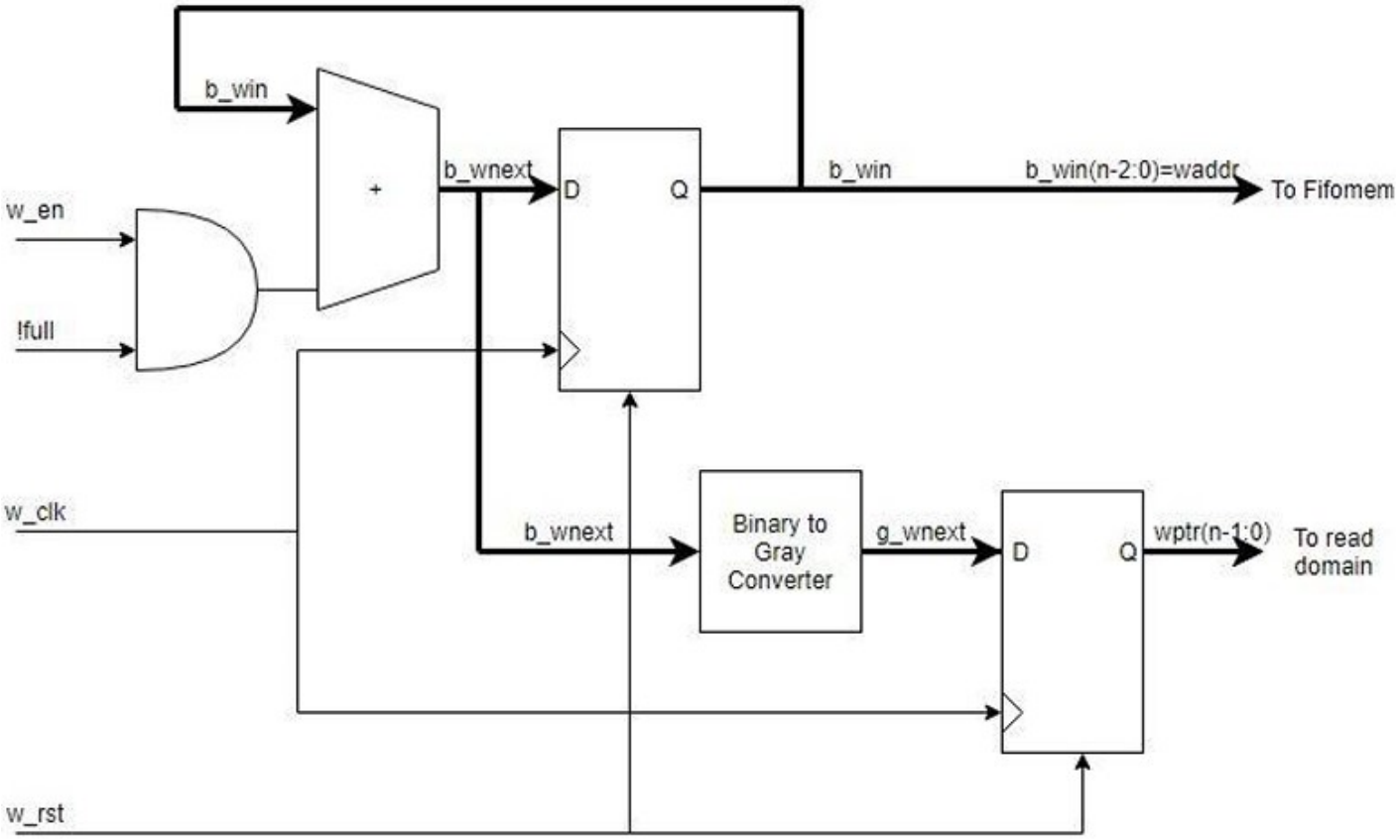

Figure 3: Block Diagram for Write Domain

- The write domain generates waddr, which will be used to write data in the corresponding location. Generation of waddr is done through conditional incrementor circuit which is then synchronised with the w_clk using a D flipflop.
- The condition for incrementor is simple, the waddr is incremented only when w_en is asserted and the FIFO is not full.
- For empty signal generation, Gray code converter is used. The reasons behind the use of Gray code converter will be explained in the synchroniser block. The Gray code version of binary counter is registered through a D flipflop(wptr) which is then sent to the synchroniser.

Read Domain : Same Port Sizes

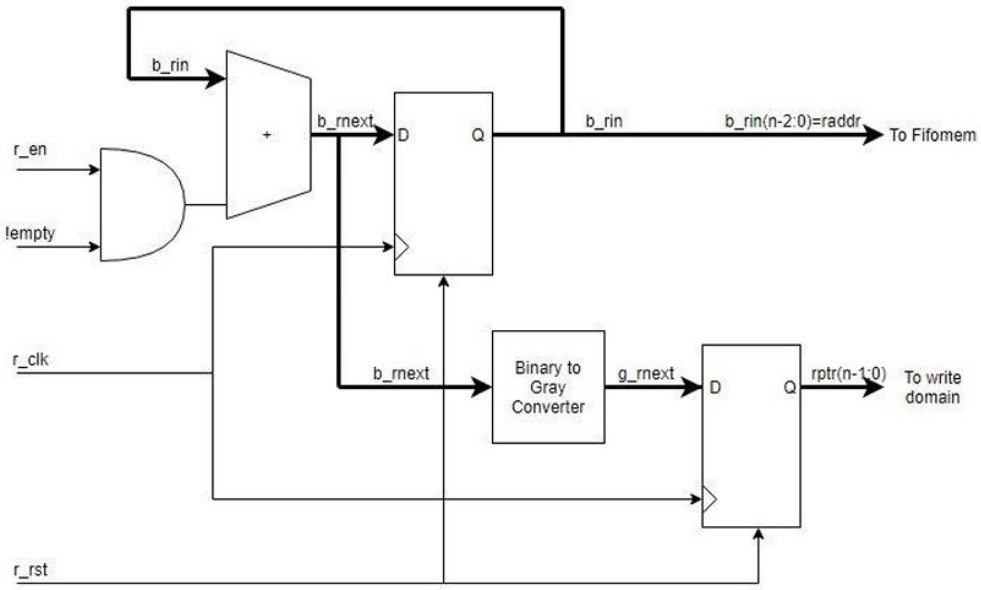

Figure 4: Block Diagram for Read Domain with Same Port Sizes

The read domain block with same port size is like the write domain. The Gray code block generates rptr signal through a flipflop. This rptr signal is sent to the synchroniser, which synchronises the pointer to the write domain.

Read Domain : Different Port Sizes

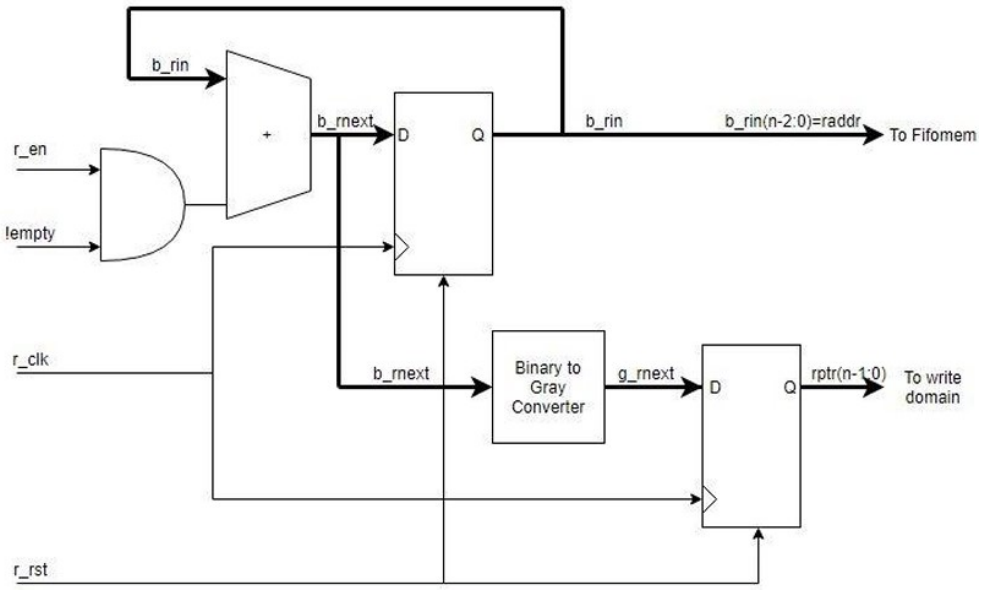

Figure 5: Block Diagram for Read Domain with Same Port Sizes

- For different port size, there is an additional mux and flipflop which will generate a signal sel which toggles at every r_clk rising edge.
- When sel is '1' the conditional incrementor is enabled and it increments the value of raddr, whereas when sel is '0' which happens in the next immediate rising edge, the incrementor is disabled, causing the raddr to retain its previous value.
- This enables the raddr to retain the same value for two read clock cycles and the corresponding data can be read from the buffer.

Synchronizer :

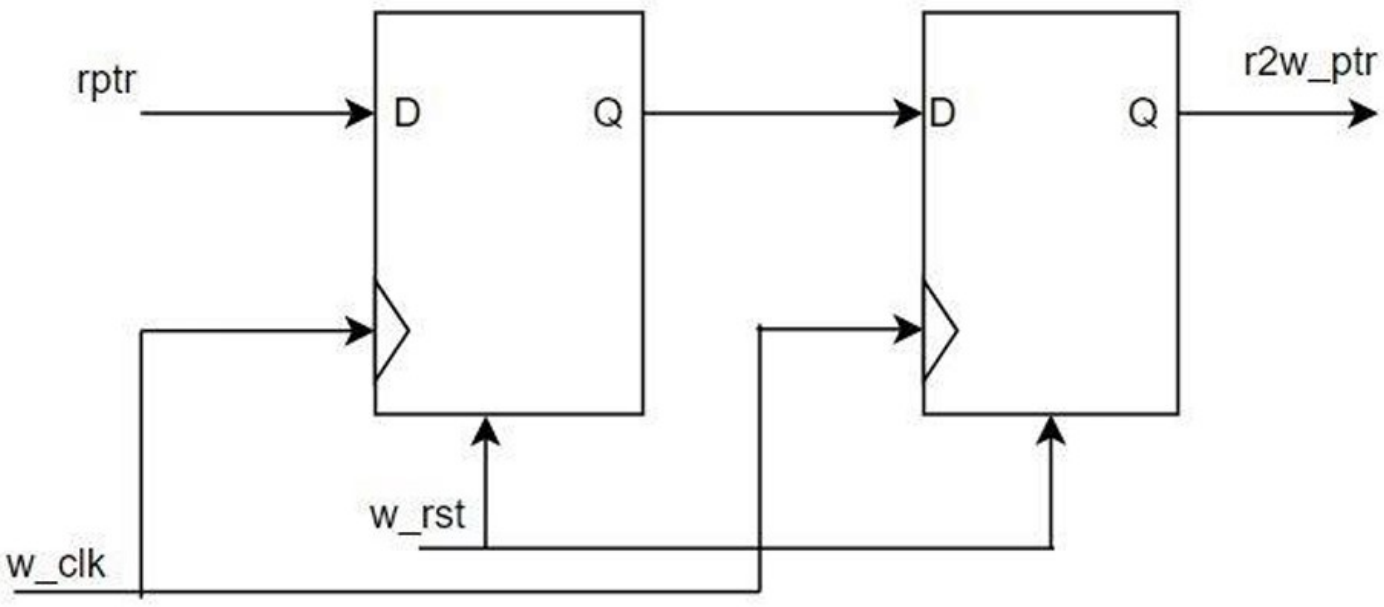

Figure 6: Block Diagram for Synchronizer

- The function of synchroniser is to synchronise asynchronous inputs to a particular clock frequency. Here, the rptr signal coming from the read domain is synchronised into the write domain.
- A 2-stage synchroniser is used for this purpose. Using two stages will reduce the chance of driving the signal into metastability as it increases the metastability resolution time. The synchroniser produces a r2w_ptr.
- Similar Synchroniser circuit is used for the wptr, producing a w2r_ptr.

Empty Logic Generation :

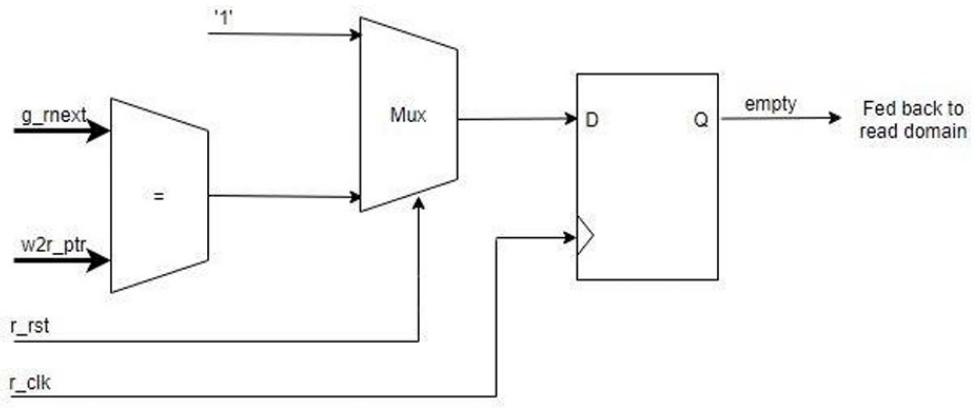

Figure 7: Block Diagram for Empty Logic Generation

Empty logic generation is straight forward. When the gray code version of read pointer and the synchronised write pointer become equal, the empty flag is asserted, implying the data on the read bus is invalid.

Full Logic Generation :

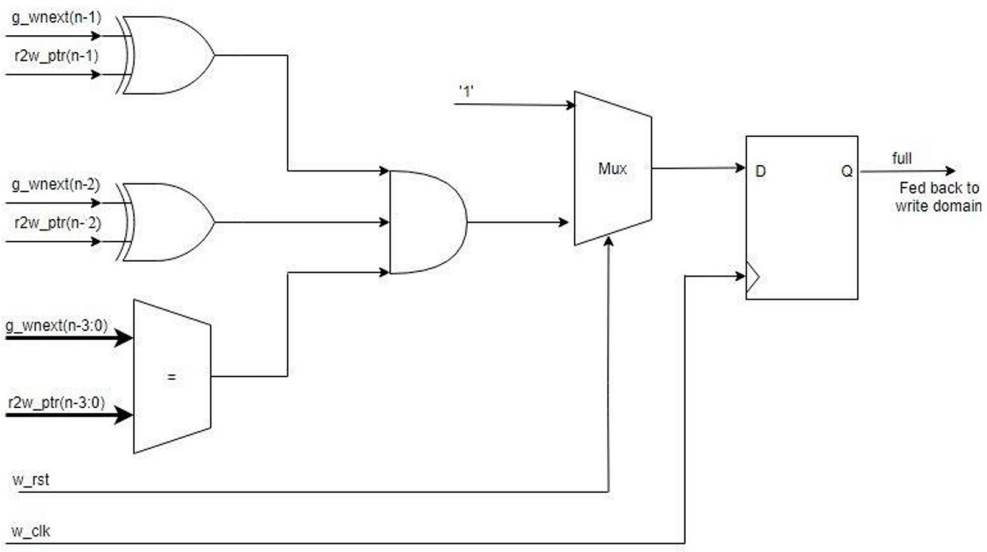

Figure 8: Block Diagram for Full Logic Generation

When the two most significant bits are complementary to each other and the remaining bits are same, then the full flag is asserted. The two MSBs being complementary only implies that the write pointer has wrapped around the FIFO array once and has come to the read pointer location. With this, both status signals are generated, which are very important to prevent underflow and overflow.

Results :

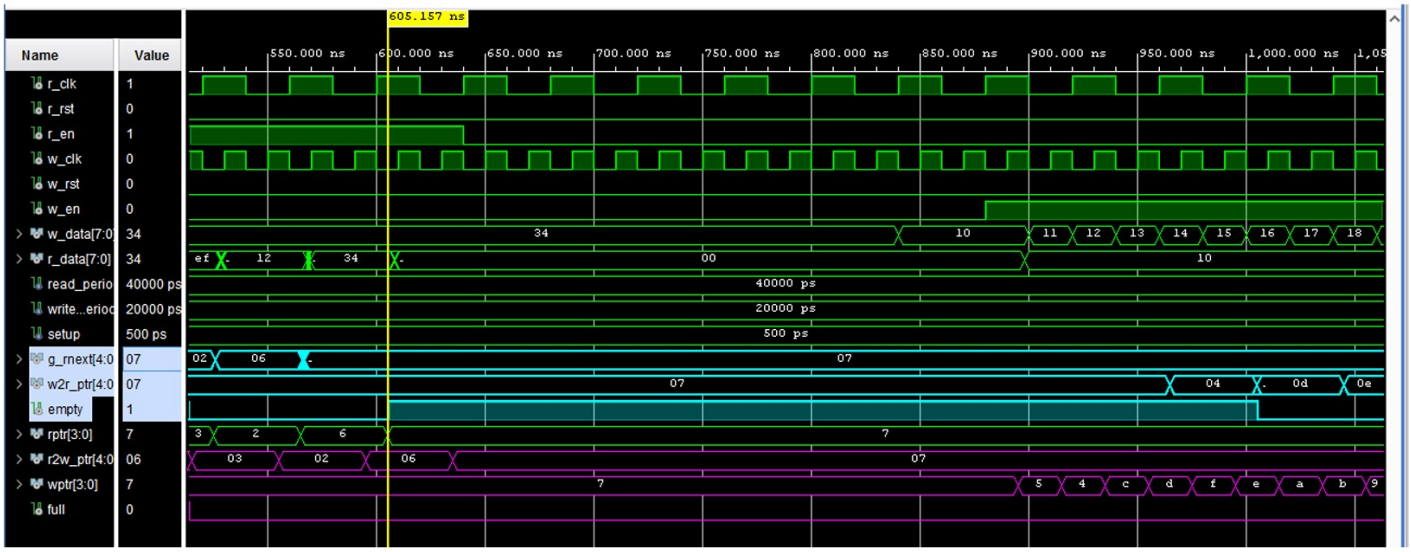

Figure 9: Empty flag asserted for same port sizes

The empty flag is asserted as soon as the gray code version of the read pointer and the synchronized write pointer become equal. This proves that the empty condition is detected immediately.

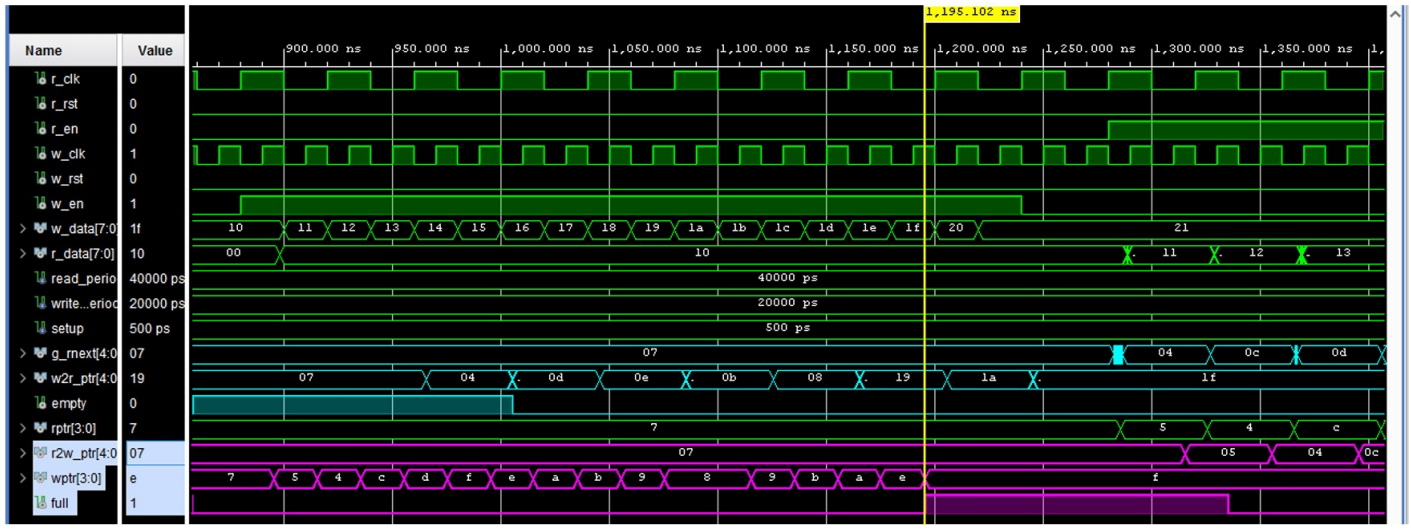

Figure 10: Full flag asserted for same port sizes

the full flag is asserted as soon as the gray code version of the read pointer and the synchronised write pointer meet the above-mentioned conditions.

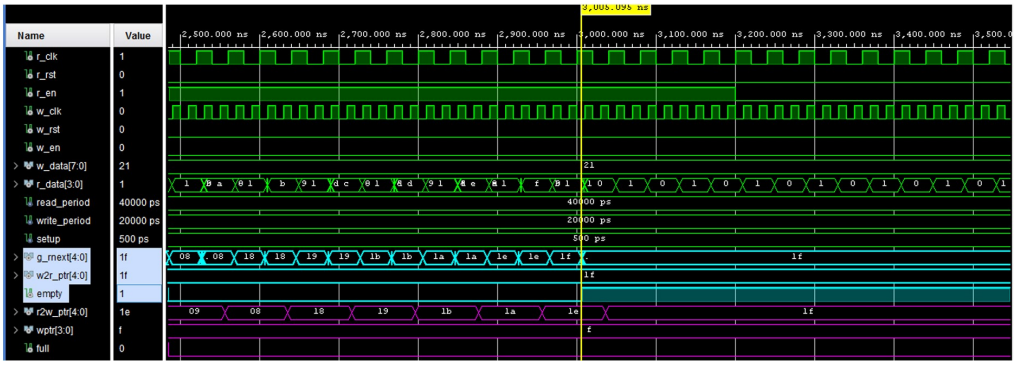

Figure 11: Empty flag asserted for different port size

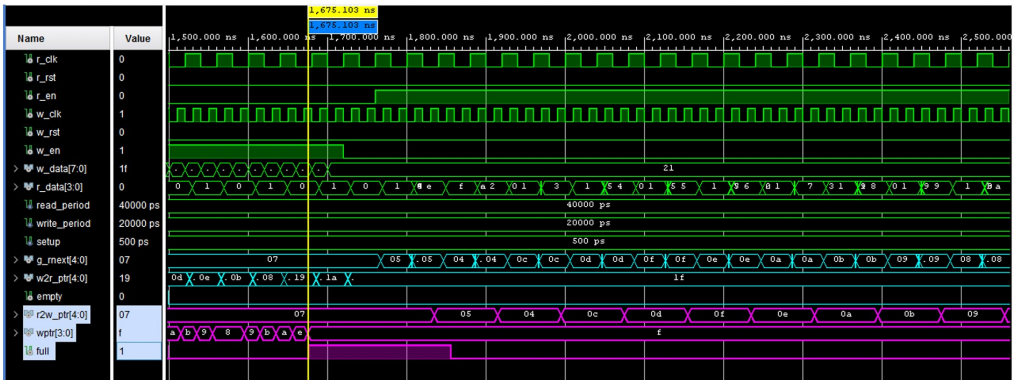

Figure 12: Full flag asserted for different port size

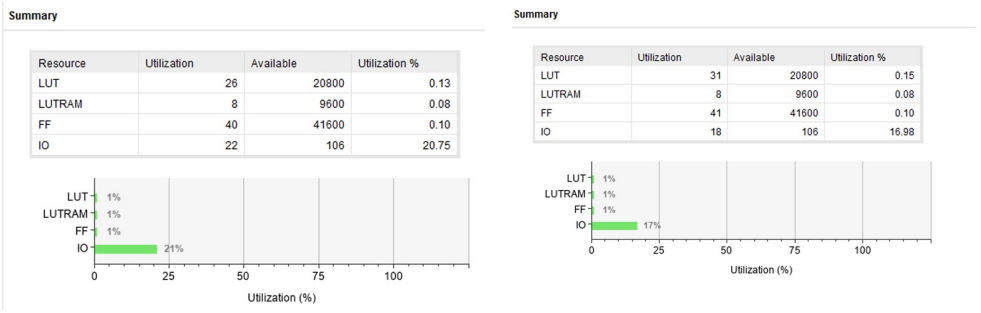

Figure 13: Utilisation for Same port sizes and Different port sizes

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	17.248 ns	Worst Hold Slack (WHS):	0.131 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	120	Total Number of Endpoints:	120	Total Number of Endpoints:	58

All user specified timing constraints are met.

Figure 14: Timing report for Same port sizes

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	17.071 ns	Worst Hold Slack (WHS):	0.167 ns	Worst Pulse Width Slack (WPWS):	8.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	121	Total Number of Endpoints:	121	Total Number of Endpoints:	59

All user specified timing constraints are met.

Figure 15: Timing report for different port sizes

Conclusion :

- The status flags are asserted exactly when the corresponding conditions are met. It is important to implement the status flags logic correctly as it is critical to ensure that the FIFO does not overflow or underflow.
- The array data type used to code the FIFO buffer is inferred as Distributed RAM which is implemented as a collection of SLICEM blocks in the FPGA.

References :

- Simulation and Synthesis techniques for Asynchronous FIFO Design - Sunburst paper on dual clock FIFO.
- Nandland videos on CDC & dual clock FIFO.