

Async FIFO

MAS

Intel Corporation  
Intel Secret

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev** | **Date** | **Owner** | **Comment** |
| 0.1 | 03/10/18 | Ori Yefet & Omri Dassa | Initial revision |

Table of Contents

[1. FIFO overview 4](#_Toc373260318)

[1.1 Block key features 4](#_Toc373260319)

[1.2 Block external interface 4](#_Toc373260320)

[2. FIFO block diagram 5](#_Toc373260321)

[3. Data and control flows 6](#_Toc373260322)

[3.1 Flow push description 6](#_Toc373260323)

[4. FIFO clock and resets domain partitions 6](#_Toc373260324)

[5. FIFO internal performance/latency assumptions 6](#_Toc373260325)

[6. Sub blocks description 7](#_Toc373260326)

[6.1 Cross\_clock 7](#_Toc373260327)

[6.1.1 Cross\_clock block diagram 7](#_Toc373260328)

[6.1.2 Cross\_clock special logic 7](#_Toc373260333)

[7. MISC 7](#_Toc373260338)

[7.1 Verification aspects 7](#_Toc373260339)

[7.1.1 Assertions 7](#_Toc373260340)

[8. BLOCK TIMING 8](#_Toc373260326)

Table of figures

[Figure 1 – FIFO external interfaces 4](#_Toc373260344)

[Figure 2 - FIFO block diagram 5](#_Toc373260345)

[Figure 3 – Cross\_clock block diagram 5](#_Toc373260345)

[Figure 4 - Interface A timing 8](#_Toc373260347)

Table of tables

Table 1 – FIFO external interfaces 4

[Table 2 - Flow push description 6](#_Toc373260349)

# FIFO overview

The block read and write values of a data flow. it enables a clients to save and recover values from the data flow by a protocol of “First In First Out”. This protocole recovers information in the same order it was delivered.

The block is located between 2 blocks that one is to transfer information to the other.

The block will have the abilty of working in any depth greater than 4 determined by parameter.

The block will be able to receive data of any size determined by parameter.

The block is working under 2 clock domains – one for the writing and the other for retreiving.

## Block key features

1. Pop – reads a value that was saved to the FIFO.
2. Push – writes a value from the data flow to the FIFO.
3. Empty – notifying when FIFO is empty.
4. Full – notifying when FIFO is full.

## Block external interface



Figure 1 – Async FIFO external interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| Signal/Bus | Direction | Size | Comment |
| Clk\_rdd | input | 1 bit | The clock of the reading domain |
| Pop | input | 1 bit | retrieves an element when is set |
| Empty | output | 1 bit | Is set when FIFO is empty |
| Data\_in | input | DATA\_SIZE Parameter | data to insert the FIFO |
| Clk\_wrd | input | 1 bit | The clock of the writing domain |
| Push | input | 1 bit | Inserts the element in data\_in when sets |
| Full | output | 1 bit | Is set when FIFO is full |
| Data\_out | output | DATA\_SIZE Parameter | data to extract from the FIFO when pop is set |
| Rst | input | 1 bit | Resets all enables when off. |

Table 2 – Async FIFO external interfaces

# FIFO block diagram



Figure 3 FIFO block diagram

Controllers – each receives the inputs of FIFO block and determines the action that ought to be executed. Delivers the pointers to register\_file. Works on the read/write clk domain and resets the pointers and the flags on ~rst.

Register\_file – receives the data from external and the pointers from the controller and execute the acceptable operation- pop/push data in/out of the FIFO.

Write/Read logic – computes the next state of the controllers’ enables – pop&(~empty) / push&(~full).

Full/Empty logic – computes the full and empty signals.

cross\_clock – delivering the read/write ptrs of the reg\_file from each clock domain to the other using grey code and synchronizer.

# Data and control flows

* + - Data flow: data is divided to 2 orientations – read and write.
      * Write: data enters the block from data\_in port. The data is written to the FIFO registers as long as the full flag is not set. By the next clock of the write orientation clock domain, the data will be valid in the register file.
      * Read: data is presented on the data\_out port. Data is valid by the next cycle of the read orientation clock domain as long as the empty flag is not set.
    - Control flow:
      * data is written and being read to and from the reg-file by “read” and “write” ptrs to the register.
      * Ptrs are being incremented every valid reading and writing operation.
      * Reading and writing operation are controlled by “read” and “write” enables and are being determined as specified in the block diagram.
      * Full and empty flags are being set when ptrs of the “read” and “write” overlaps. In order to determine whether the fifo is full or empty, ptrs are holding an extra bit that suggest full state or empty state.
      * Ptrs cross domain with cross\_clock sub-block as described on section 6.

## Flow push description

Example to a push feature flow:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Signals: push = “1”, data\_in = “value”, rst = “0” |
| 2 | Controller: checks if FIFO is full before changing the pointers. Assume ~full. |
| 3 | Controller: checks if FIFO is full after inserting. |
| 4 | Register\_file: stores the data to the FIFO – reg\_file[wr\_add] = wr\_d, raising full flag in case the FIFO is full. |

Table 1 - Flow push description

# FIFO clock and resets domain partitions

Clock domains:

* Pop domain – works on rising edge, operates the read-controller and the “read” ptrs and enables are being determined by its domain.
* Push domain - works on rising edge, operates the write-controller and the “write” ptrs and enables are being determined by its domain.

Reset domain:

The structure will support an asynchronous reset and will do the following upon reset:

* Sets empty to “1”.
* Sets full to “0”.
* Sets all pointers to 0.

# FIFO internal performance/latency assumptions

Data\_out – valid to the client the next cycle pop signal was set.

Empty/Full – valid the next cycle.

# Sub blocks description

## Cross\_clock

The sub block receives data on one clock domain and delivers it to another clock domain.

The block uses grey code and synchronizer.

### Cross\_clock block diagram



Figure 4 - [sub\_block] block diagram

The data input is coded to grey-code so to avoid metastable-bus issues and then sampled before synchronized. Output is being decoded back to binary.

### Cross\_clock special logic

* Bin to grey: XOR with right-shifted data. MSB is copied as is.
* Synchronizer: 2 flops samples. (subs for the generic one – for practice)
* Grey to bin: every bit of the data bus is being XORed with the bus more significant bits. MSB is copied as is.

# MISC

## Verification aspects

* Client tries to pop from an empty fifo.
* Client tries to push to a full fifo.
* Jasper Gold

### Assertions & Assumes

* Warning when client tries to pop from an empty fifo.
* Warning when client tries to push to a full fifo.

In either case – simulation will not be toppled.

* Formal AV by Jasper Gold:
  + ~(empty & full).
  + (empty & push) ##4 ~empty
  + (full & pop) ##4 ~full

# Block Timing

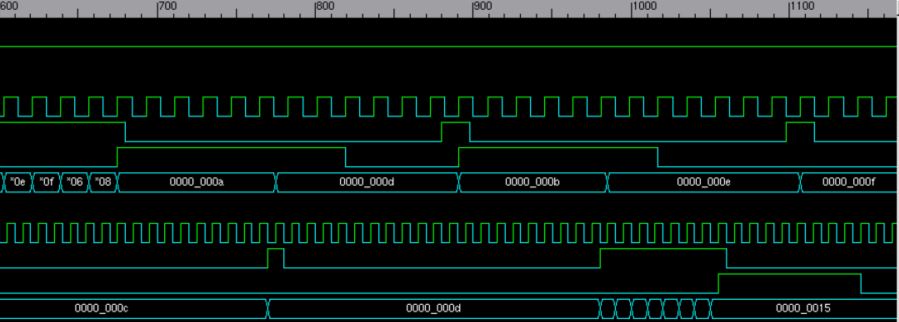
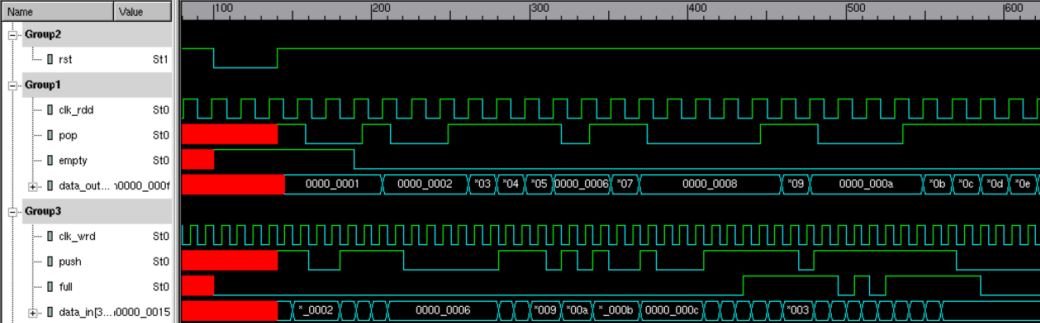
 

Figure 3 FIFO Timing