CFS VISION

Aligner Datasheet

Table of Contents

1. Overview

The Aligner module takes in an unaligned stream of data and outputs it as an aligned stream of data based on its configuration.

Its purpose if to optimize writes in memory by performing only the writes best suited for the type of memory used in the system.

1.1. Block Diagram

The image below shows the block diagram of the Aligner.

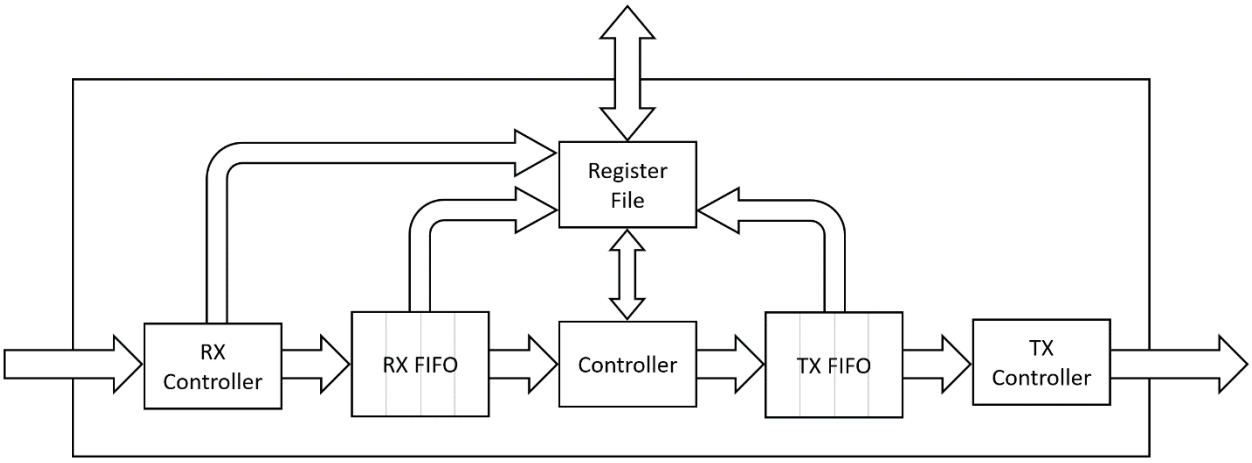

Figure 1 Block Diagram

2. Interface

The Aligner module has several parameters which gives it enough flexibility to be used in a wide variety of systems. The table below shows the available parameters of the module.

Name	Default Value	Description
ALGN_DATA_WIDTH	32	Width, in bits, of the data bus through which the Aligner receives unaligned data (md_rx_data bus) and through which the Aligner sends aligned data (md_tx_data bus). The value must be a power of 2. The minimum legal value for this parameter is 8.
FIFO_DEPTH	8	The depth of the two FIFOs used to handle the received and transmit stream of information.

Table 1 Aligner Parameters

The aligner module uses two types of interfaces:

- A standard AMBA 3 APB for accessing the registers.
- Two interfaces using the same custom MD (Memory Data) protocol:
  - An RX interface through which the Aligner receives the unaligned data.
  - A TX interface through which the Aligner sends the aligned data.

Signal Name	Width	Dir.	Description
clk	1	IN	Clock signal on which the entire module is working.
reset_n	1	IN	Reset signal – active low.
psel	1	IN	APB select.
penable	1	IN	APB enable.
pwrite	1	IN	APB write.
paddr	16	IN	APB address. Bits *paddr[1:0]* are ignored and always treated as equal to 2'b00. This means that all accesses are treated as word (4 bytes) aligned.
pwdata	32	IN	APB write data.
pready	1	OUT	APB ready.
prdata	32	OUT	APB read data.
pslverr		OUT	APB slave error.
md_rx_valid	1	IN	MD RX valid. Once it becomes high, it must stay high until md_rx_ready becomes high.
md_rx_data	ALGN_DATA_WIDTH	IN	MD RX data. It is valid while md_rx_valid is high. It must remain constant until md_rx_ready becomes high.

Signal Name	Width	Dir.	Description
md_rx_offset	max(1, $\log_2$(ALGN_DATA_WIDTH/8))	IN	MD RX offset. It represents the offset, in bytes, on the md_rx_data bus, from which the valid data starts. It is valid while md_rx_valid is high. It must remain constant until md_rx_ready becomes high. Not all combinations of (offset, size) are legal. The following equation describes the legal combinations: ((ALGN_DATA_WIDTH / 8) + offset) % size == 0
md_rx_size	$\log_2$(ALGN_DATA_WIDTH/8)+1	IN	MD RX size. It represents the size, in bytes, of the valid data from the md_rx_data bus. Value 0 is illegal – must never be used. It is valid while md_rx_valid is high. It must remain constant until md_rx_ready becomes high. Not all combinations of (offset, size) are legal. The following equation describes the legal combinations: ((ALGN_DATA_WIDTH / 8) + offset) % size == 0
md_rx_ready	1	OUT	MD RX ready.
md_rx_err	1	OUT	MD RX error. It is valid only when both md_rx_valid and md_rx_ready are high. Can be high only when md_rx_valid and md_rx_ready are high.
md_tx_valid	1	OUT	MD TX valid. Once it becomes high, it must stay high until md_tx_ready becomes high.
md_tx_data	ALGN_DATA_WIDTH	OUT	MD TX data. It is valid while md_tx_valid is high. It must remain constant until md_tx_ready becomes high.
md_tx_offset	max(1, $\log_2$(ALGN_DATA_WIDTH/8))	OUT	MD TX offset. It represents the offset, in bytes, on the md_tx_data bus, from which the valid data starts. It is valid while md_tx_valid is high. It must remain constant until md_tx_ready becomes high. Not all combinations of (offset, size) are legal. The following equation describes the legal combinations: ((ALGN_DATA_WIDTH / 8) + offset) % size == 0
md_tx_size	$\log_2$(ALGN_DATA_WIDTH/8)+1	OUT	MD TX size. It represents the size, in bytes, of the valid data from the md_tx_data bus. Value 0 is illegal. It is valid while md_tx_valid is high. It must remain constant until md_tx_ready becomes high. Not all combinations of (offset, size) are legal. The following equation describes the legal combinations: ((ALGN_DATA_WIDTH / 8) + offset) % size == 0

Signal Name	Width	Dir.	Description
md_tx_ready	1	IN	MD TX ready.
md_tx_err	1	IN	MD TX error. It is valid only when both md_tx_valid and md_tx_ready are high. Can be high only when md_tx_valid and md_tx_ready are high.
irq	1	OUT	Interrupt request. All the interrupt requests are ORed into this one-bit output signal.

Table 2 Aligner Interface Signals

3. Registers

The Aligner module has several control and status registers accessible though the APB interface.

The following rules govern the scenarios in which the Aligner must return an APB error:

- Any access to a location on which no register is mapped must return an APB error.
- Any write access to a full read-only register must return an APB error.
- Any read access from a full write-only register must return an APB error.
- Illegal write access to the Control register.

The table below list all the registers with their corresponding offset.

Mnemonic	Offset	Name
CTRL	0x0000	Control Register
STATUS	0x000C	Status Register
IRQEN	0x00F0	Interrupt Requests Enable Register
IRQ	0x00F4	Interrupt Requests Register

Table 3 Registers

The register access types are listed in the table below.

Access	Description
RW	Readable-Writable A register field with this access type can be read and written.
RO	Read-Only A register field with this access type can only be read. A write attempt to such a register field will not chance its value.
WO	Write-Only A register field with this access type can only be written. A read attempt will always return value 0.
W1C	Write-1-Clear A register field with this access type can be read. A write attempt with value 1 will clear the register field. A write attempt with value 0 will have no effect on the value of the register field.

Table 4 Register field access types

3.1. Control Register

The Control register contain configurations fields required in order to control the Aligner functionality.

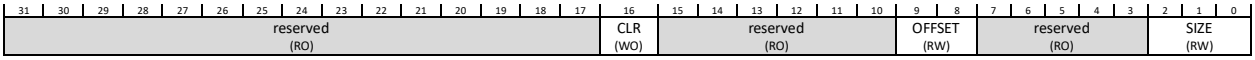

Figure 2 Control register layout

The fields of the Control register are listed in the table below.

Mnemonic	Offset	Size	Access	Reset	Description
SIZE	0	3	RW	1	Size, in bytes, of the aligned data. Trying to write value 0 in this field will return an APB error. Trying to write an illegal combination of (SIZE, OFFSET) will return an APB error.
reserved	3	5	RO	0	Reserved field.
OFFSET	8	2	RW	0	Offset, in bytes, of the aligned data. Trying to write an illegal combination of (SIZE, OFFSET) will return an APB error.
reserved	10	6	RO	0	Reserved field.
CLR	16	1	WO	0	Clear the status counter (CNT_DROP) when writing 1. Writing 0 has no effect. A read will always return value 0.
reserved	17	15	RO	0	Reserved field.

Table 5 Control register fields

3.2. Status Register

The Status register contains status information about the state in which the Aligner is at some moment in time.

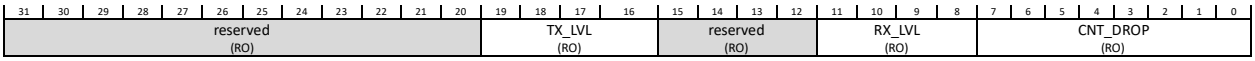

Figure 3 Status register layout

The fields of the Control register are listed in the table below.

Mnemonic	Offset	Size	Access	Reset	Description
CNT_DROP	0	8	RO	0	Number of unaligned accesses, which are dropped. Once it reaches the maximum value the counter will not wrap.
RX_LVL	8	4	RO	0	Fill level of the RX FIFO.
reserved	12	4	RO	0	Reserved field.
TX_LVL	16	4	RO	0	Fill level of the TX FIFO.
reserved	20	12	RO	0	Reserved field.

Table 6 Status register fields

3.3. Interrupt Requests Enable Register

The Interrupt Requests Enable register contains fields to enable each individual interrupt.

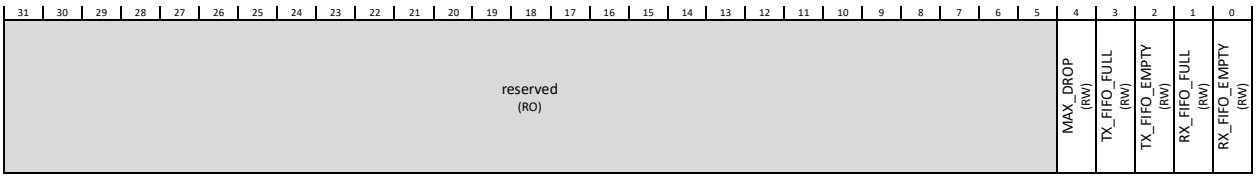

Figure 4 Interrupt Requests Enable register layout.

The fields of the Control register are listed in the table below.

Mnemonic	Offset	Size	Access	Reset	Description
RX_FIFO_EMPTY	0	1	RW	0	Enable IRQ.RX_FIFO_EMPTY in the irq output.
RX_FIFO_FULL	1	1	RW	0	Enable IRQ.RX_FIFO_FULL in the irq output.
TX_FIFO_EMPTY	2	1	RW	0	Enable IRQ.TX_FIFO_EMPTY in the irq output.
TX_FIFO_FULL	3	1	RW	0	Enable IRQ.TX_FIFO_FULL in the irq output.
MAX_DROP	4	1	RW	0	Enable IRQ.MAX_DROP in the irq output.
reserved	5	27	RO	0	Reserved field.

Table 7 Interrupt Requests Enable register fields.

3.4. Interrupt Requests Register

The Interrupt Requests register contains the state of each available interrupt.

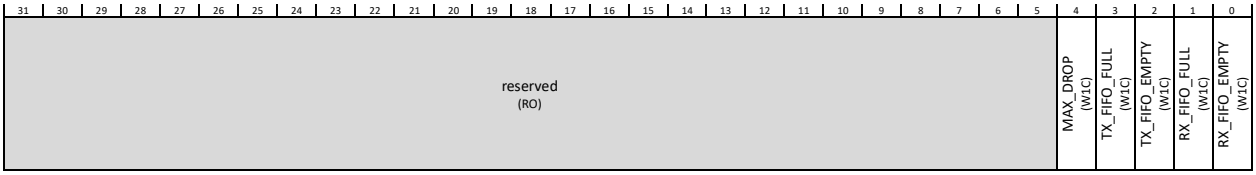

Figure 5 Interrupt Requests register layout

The fields of the Control register are listed in the table below.

Mnemonic	Offset	Size	Access	Reset	Description
RX_FIFO_EMPTY	0	1	W1C	0	RX FIFO became empty (sticky). If cleared while RX FIFO is still empty, the interrupt will not be set immediately. It will be set again once the RX FIFO becomes empty (e.g. STATUS.RX_LVL transitions from 1 to 0).
RX_FIFO_FULL	1	1	W1C	0	RX FIFO became full (sticky). If cleared while RX FIFO is still full, the interrupt will not be set immediately. It will be set again once the RX FIFO becomes full (e.g. STATUS.RX_LVL transitions from MAX-1 to MAX).
TX_FIFO_EMPTY	2	1	W1C	0	TX FIFO became empty (sticky). If cleared while TX FIFO is still empty, the interrupt will not be set immediately. It will be set again once the TX FIFO becomes empty (e.g. STATUS.TX_LVL transitions from 1 to 0).
TX_FIFO_FULL	3	1	W1C	0	TX FIFO became full (sticky). If cleared while TX FIFO is still full, the interrupt will not be set immediately. It will be set again once the TX FIFO becomes full (e.g. STATUS.TX_LVL transitions from MAX-1 to MAX).
MAX_DROP	4	1	W1C	0	STATUS.CNT_DROP reached its maximum value (sticky). If cleared while STATUS.CNT_DROP is still MAX interrupt will not be set immediately. It will be set again once the STATUS.CNT_DROP becomes MAX (e.g. STATUS.CNT_DROP transitions from MAX-1 to MAX).
reserved	5	27	RO	0	Reserved field.

Table 8 Interrupt Requests register fields.

4. Functionality

4.1. Data Alignment

The role of the Aligner module is to take an unaligned stream of data and send it out as an aligned stream of data. The alignment is done based on the CTRL.SIZE and CTRL.OFFSET register fields.

The figures below show how the alignment is done based on different values for CTRL.SIZE and CTRL.OFFSET, while ALGN_DATA_WIDTH is 32.

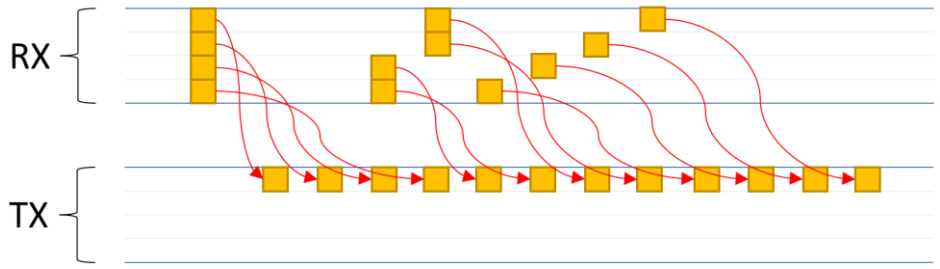

Figure 6 Data alignment when CTRL.SIZE = 1 and CTRL.OFFSET = 0

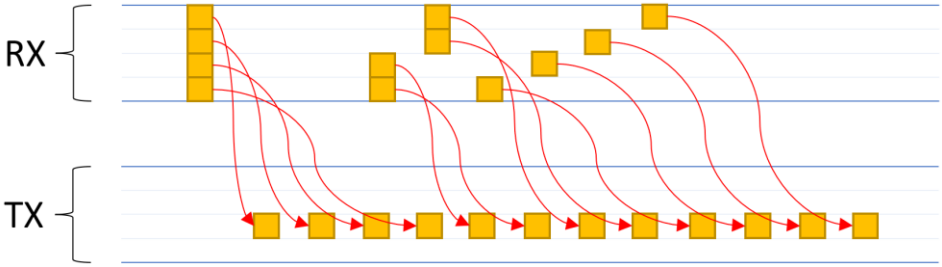

Figure 7 Data alignment when CTRL.SIZE = 1 and CTRL.OFFSET = 2

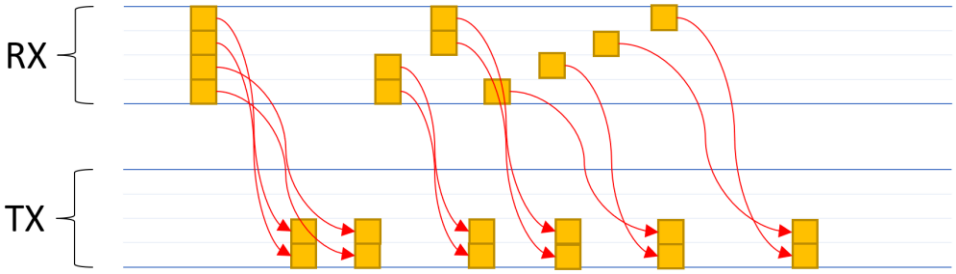

Figure 8 Data alignment when CTRL.SIZE = 2 and CTRL.OFFSET = 2

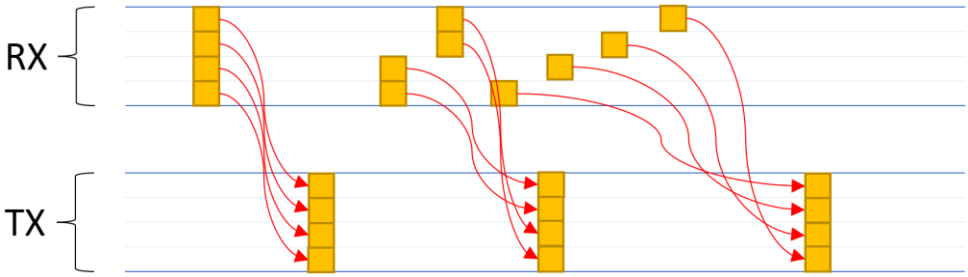

Figure 9 Data alignment when CTRL.SIZE = 4 and CTRL.OFFSET = 0

For both RX and TX interfaces the same custom Memory Data (MD) protocol is used.

The figure below shows how the RX and TX interface behave for a setup in which CTRL.SIZE = 11, CTRL.OFFSET = 0 and ALGN_DATA_WIDTH is 32.

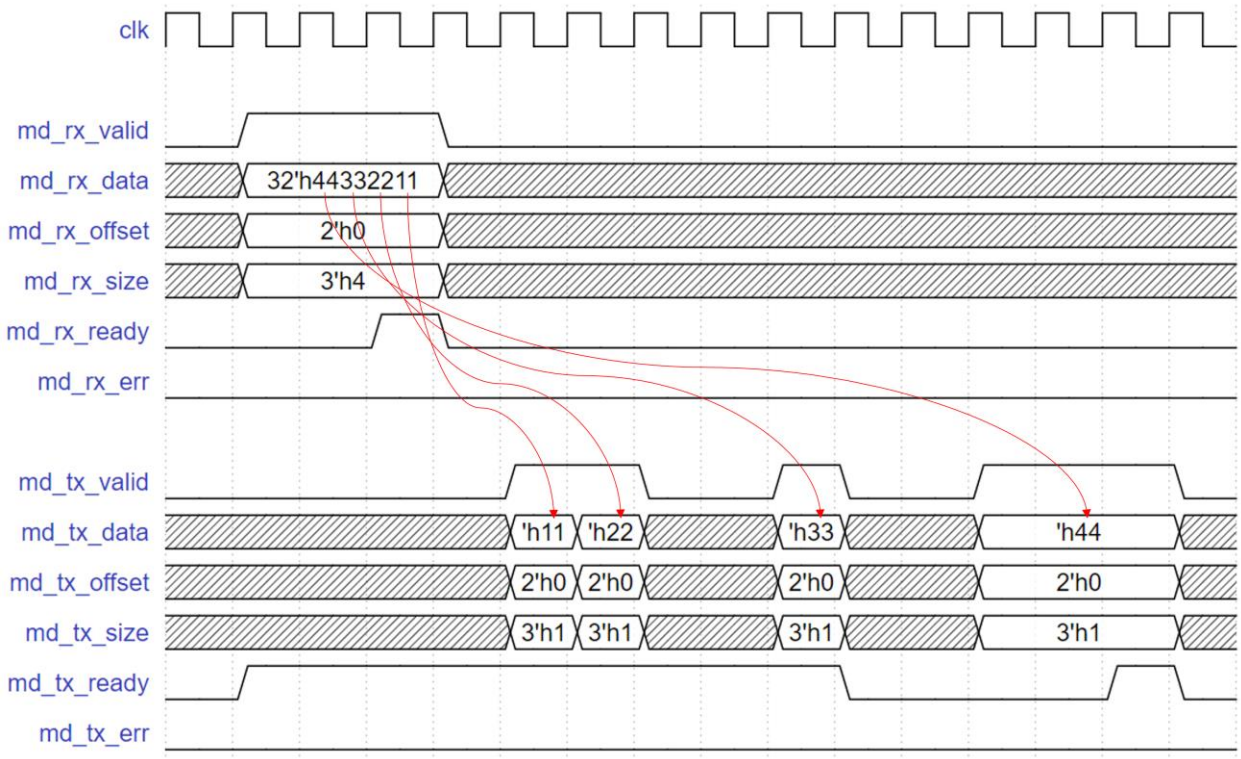

Figure 10 RX and TX interfaces during data alignment when CTRL.SIZE = 1 and CTRL.OFFSET = 0

4.2. Flow Control

The flow of information comes from the RX interface, passes through the Aligner, and goes out on the TX interface.

Inside the Aligner, it passes through all the submodules presented in Figure 1 Block Diagram.

4.2.1. RX Controller

The RX Controller has two main responsibilities.

First, it controls the backpressure on the RX interface, via *md_rx_ready*, to pause the incoming traffic when the RX FIFO is full. So, whenever the RX FIFO has room to receive another data, the RX Controller will set *md_rx_ready* to 1. But when the RX FIFO is full, the RX Controller will set *md_rx_ready* to 0.

Second, it determines if the incoming MD transfer on the RX interface is legal or not. A legal transfer must satisfy the following equation:

$$((ALGN\_DATA\_WIDTH / 8) + offset) \% size == 0$$

If this equation is not satisfied, then the RX Controller responds back with error (*md_rx_err* is set to 1) and the drop counter is incremented (STATUS.CNT_DROP) if it is not already at its maximum value.

Once STATUS.CNT_DROP reached its maximum value it will remain at this value regardless how many illegal MD transfers are received. This counter can only be reset to 0 by writing value 1 to CTRL.CLR.

There is an interrupt request generated whenever STATUS.CNT_DROP reaches its maximum value.

When this happens, IRQ.MAX_DROP is set to 1 and it will remain unchanged until it is cleared (write with value 1 in IRQ.MAX_DROP) regardless if STATUS.CNTDROP is no longer at its maximum value (sticky behaviour).

4.2.2. RX FIFO

The RX FIFO (First In First Out) is responsible with buffering the incoming data traffic, until the Controller is ready to align a new data.

The fill level of this RX FIFO is reflected in STATUS.RX_LVL register field.

There is an interrupt request generated whenever RX FIFO becomes empty.

When this happens, IRQ.RX_FIFO_EMPTY is set to 1 and it will remain unchanged until it is cleared (write with value 1 in IRQ. RX_FIFO_EMPTY) regardless if RX FIFO is no longer empty (sticky behaviour).

There is an interrupt request generated whenever RX FIFO becomes full.

When this happens, IRQ.RX_FIFO_FULL is set to 1 and it will remain unchanged until it is cleared (write with value 1 in IRQ. RX_FIFO_FULL) regardless if RX FIFO is no longer full (sticky behaviour).

4.2.3. Controller

The Controller is responsible for doing the actual alignment of the data, based on CTRL.SIZE and CTRL.OFFSET configuration.

Every time the RX FIFO has some data available it will take it out, align it, and send it to the TX FIFO.

When the TX FIFO is full, the Controller will wait for it to become not full so that it can push inside new data.

4.2.4. TX FIFO

The TX FIFO is responsible with buffering the outgoing data traffic, until the TX Controller is ready to sent it out on the MD TX interface.

The fill level of this TX FIFO is reflected in STATUS.TX_LVL register field.

There is an interrupt request generated whenever TX FIFO becomes empty.

When this happens, IRQ.TX_FIFO_EMPTY is set to 1 and it will remain unchanged until it is cleared (write with value 1 in IRQ. TX_FIFO_EMPTY) regardless if TX FIFO is no longer empty (sticky behaviour).

There is an interrupt request generated whenever TX FIFO becomes full.

When this happens, IRQ.TX_FIFO_FULL is set to 1 and it will remain unchanged until it is cleared (write with value 1 in IRQ. TX_FIFO_FULL) regardless if TX FIFO is no longer full (sticky behaviour).

4.2.5. TX Controller

The TX Controller is responsible for taking the data out of the TX FIFO and send it on the MD TX interface.

4.3. Register Access

The registers are accessible via a standard AMBA 3 APB.

Based on the synchronization requirements, some of the accesses might introduce several wait states.

However, it is illegal to have more than 5 wait states in an APB transfer.

The two least significant bits of the address (*paddr[1:0]*) are ignored and treated as they are always 2'b00. This means that all accesses are considered word (4 bytes) aligned.

There are several scenarios for which an APB transfer can end with an error (*pslverr* equal to 1).

The Aligner will respond with APB error in any of the scenarios listed below:

- Access to an unmapped location
- Write access to STATUS register
- Write access to CTRL with an illegal combination of new values for CTRL.OFFSET and CTRL.SIZE

4.4. Interrupt Requests

The Aligner module has several interrupt requests to allow software routines to be triggered every time a particular event is happening.

All the interrupts are ORed together to form the *irq* output of the Aligner.

Each interrupt is enabled, to contribute to the *irq* output, via the enable configuration from the Interrupt Requests Enable Register.

Whenever the event which triggers an interrupt is emitted, the corresponding bit in Interrupt Requests Register is set, regardless of the enable bit from Interrupt Requests Enable Register.

Whenever the event which triggers an interrupt is emitted, the corresponding interrupt is generated as a pulse (equal to 1 for one clock cycle).

The figure below shows how the *irq* output is generated based on the events part of the interrupt requests.

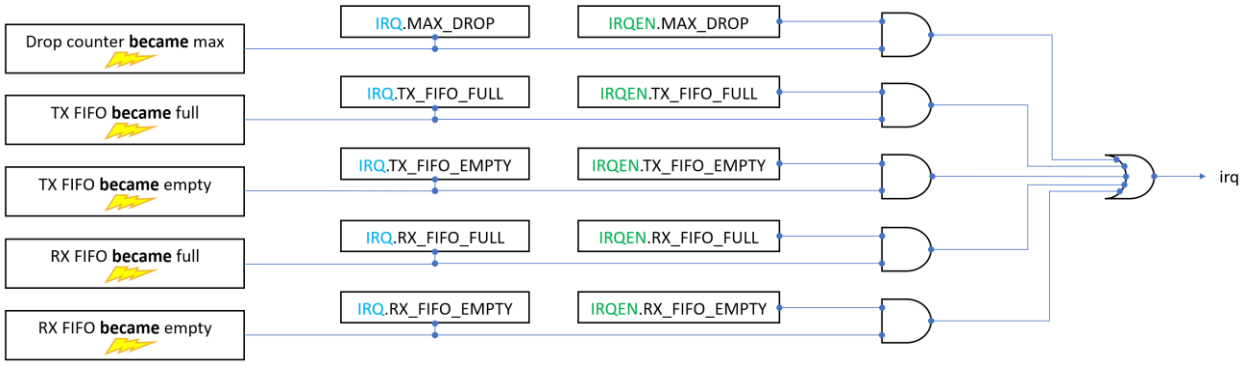

Figure 11 Logic for generating irq output