SPACEWIRE IP CORE
SPECIFICATION AND ARCHITECTURE

	Name and Function	Date	Signature
Prepared by	Tam LE NGOC		
Verified by	Marc LEFEBVRE		
Approved by			
Authorised by	Marc SOUYRI		

Document type	Nb WBS	Keywords

CharNb 63422
WordsNb 12475
FileName SW_SPEC_ARCHI_02.DOC

DOCUMENT CHANGE LOG

Issue/ Revision	Date	Modification Nb	Modified pages	Observations
0/0				Creation
0/1	01/04/03		14,20,21,22	Adding error response and error interrupt if the TX AHB slave is accessed when the link is not connected. Suppression of the "shall".
0/2	03/04/03		34,20	Adding TX clock selection. Adding complete packet reception interrupt. Adding AHB FIFO full status.

PAGE ISSUE RECORD

Issue of this document comprises the following pages at the issue shown

Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.
all	**0/0**										
all	**0/1**										
all	**0/2**										

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PAGE INTENTIONALLY LEFT BLANK

1 SCOPE

The present document is written in the frame of the ESA 13345/#3 contract " Building block for System on a Chip". It is part of Phase 3 of the contract related to the design of a System On a Chip for Space application. The present activity concerns the design of a Spacewire VHDL core to be integrated in the System On a CHip.

The present document describes the SpaceWire block developed as part of the ScoC project. This document contains the specification and the architecture of the block. The SpaceWire is a serial high speed link compliant with the ECSS-E-50-12 Draft 1 specification (AD11) delivered by ESA. For the SCoC project, the SpaceWire block (SWB) also contains AHB and APB interfaces.

2 DOCUMENTS AND ACRONYMS

2.1 APPLICABLE DOCUMENTS

AD8	*SCOC Requirement Specification*	R&D-RP-SOC-214-MMV, Issue 2, June 2000
AD9	*AMBA™ Specification*	*Rev 2.0, ARM IHI 0011A*
AD10	*Spacecraft Controller On a Chip Architectural Design Document*	*Draft*
AD11	*ECSS-E-50-12 Draft 1 (ESA SpaceWire Specification)*	*March 2001*

2.2 REFERENCE DOCUMENTS

RD21	*System-On-a-Chip Feasibility Study*	December 99, Issue 2, R&D-RP-SOC-154-MMV
RD22	*Spacewire IP Core Hardware User Manual*	December 2001, Issue 0, R&D-SOC-NT-295-V-ASTR

2.3 ACRONYMS

AD	Applicable Document
APB	Advanced Peripheral Bus
AHB	Advanced High-Performance Bus
DMA	Direct Memory Access
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HKPF	Housekeeping Function
HKAPB	Housekeeping Advanced Peripheral Bus
IEEE	Institute of Electrical and Electronics Engineers
IT	Interrupt
LVDS	Low Voltage Differential Signals
SCoC	Spacecraft Controller on a Chip
SWB	SpaceWire Block
RD	Reference Document
RHI	RX Host Interface
SOC	System-On-a-Chip
THI	TX Host Interface

3 FUNCTIONAL DESCRIPTION

3.1 GLOBAL FUNCTIONALITY DESCRIPTION

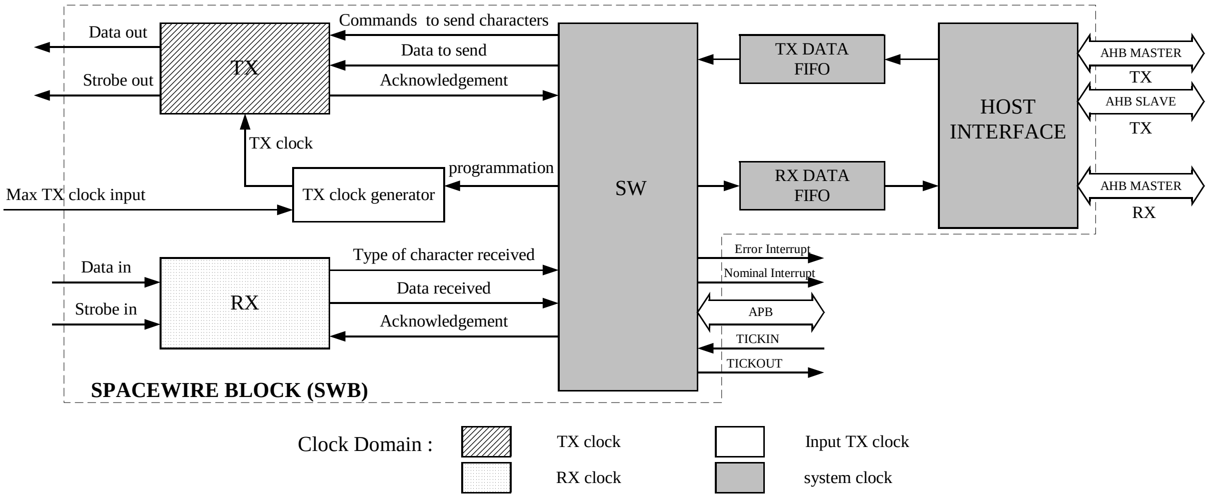


Figure 3.1.1-1 global description

The SWB is a high-speed serial link to transmit and receive packets of data (refer to AD11).

3.1.1 Description of the different blocks

The Host Interface block is an interface with the AMBA AHB and APB buses. It contains the management of the data sent by the host. It manages the storage of data into the host memory.

The TX Data FIFO block is a FIFO containing the data to be transmitted.

The RX Data FIFO block is a FIFO containing the data to be stored into the host memory.

The SW block manages the initialisation protocol. This block selects the character to be transmitted and checks any error occurrence.

The TX block sends the character at the transmission frequency.

The RX block identifies the received character type.

The TX clock generator block generates the clock transmission rate.

3.1.2 Introduction of the interfaces

The basic interface contains clock, test and resetn signals.

The APB interface is used to configure the SWB and to retrieve statuses.

The TX AHB master interface performs the TX DMA.

The TX AHB slave interface is used when the data transmission is in charge of the host.

The RX AHB master interface performs the storage of received data into the host memory.

The link interface brings together the data and strobe signals of the transmission and the reception.

The Time interface manages the transmission and the reception of time code.

The Interrupt interface is used to warn the host when a specific event appears.

3.1.3 Link initialization

Refer to AD11 chapter 8.7.

3.1.4 Transmission function

The SWB receives packets of data from the host through the AHB interface. Two modes are possible for this data transfer. The first one is the AHB master mode, which performs the transfer with a DMA mechanism. With the descriptor of a linked list of packets given by the host, the SWB retrieves 32-bit data of each packet of the linked list. The second one is the AHB slave mode. In this mode, the host transfers the length of the packet and the 32-bit data of the packet to the SWB.

Then the 32-bit data received from the host is split to 9-bit data to be stored into the TX data FIFO. The 9-bit data is composed of 8 bits of real data and 1 bit for particular character such as EOP and EEP (refer to AD11).

When the credit counter is positive (refer to AD11), the SW module fetches the 9-bit data in the TX data FIFO and sends it to the TX module with the right command to transmit this data. When the RX data FIFO free space allows the reception of 8 more bytes, the SW module generates an order to transmit a FCT. To transmit a time code, the TICKIN signal is activated so that the SW module generates the right transfer.

When the TX module receives a command from the SW module, an acknowledgement is generated. Then the character corresponding to the command is transmitted through the LVDS link (Data and Strobe outputs). The TX module automatically transmits NULL characters (refer to AD11) when no other transmission is requested.

The transmission frequency is programmable through the APB interface. The TX clock generator creates the required TX frequency, which can be **up to 4 times** the system clock frequency.

3.1.5 Reception function

The RX module performs the recognition of the received character type. The RX clock is built from the data and strobe input signals (refer to AD11). The RX module also indicates the received characters to the SW module.

Each time that information of character type is received from the RX module, the SW module generates an acknowledgement. Then, following the received character, the SW module manages the credit counter and the outstanding counter. A 9-bit word is stored into the RX data FIFO when a data is received. The SW module also activates the TICKOUT signal when a right time code is received.

When the RX data FIFO is not empty, the host interface fetches its 9-bit data. Each time four 9-bit data are available, the host interface produces a 32-bit word from these four 9-bit data and stores it into the host memory through the AHB bus (master mode).

When any error is detected from the AHB transfer or from the transmission link, the SWB generates an error interrupt to warn the host. The SWB also produces a nominal interrupt to improve the monitoring.

The configuration of the SWB is done through the APB interface.

3.2 FUNCTIONAL MODE DESCRIPTION

Refer to the state diagram of AD11 chapter 8.5.

The SWB supports the following functional modes:

- RESET mode (resetn=0):
  - TX and RX blocks are inactive
  - Host interface is inactive
- ACTIVE mode (resetn=1):
  - TX block is inactive and RX block is active (when entering the ACTIVE mode)
  - Host interface is always on

The transition of the TX and RX states in the active mode is specified by the link initialisation protocol described in AD11.

3.3 DETAILED FUNCTIONALITY DESCRIPTION

3.3.1 TX clock programming

The FREQ_INIT register configures the frequency in the initialisation state.

The FREQ_RUN register configures the frequency in the run state.

For the gated TX clock configuration (see 5.2.1), when the TX_MAX_EN bit is asserted, the FREQ_RUN register value is not taken into account and the transmission frequency is equal to the input TX clock frequency.

It is recommended not to change the FREQ_INIT and FREQ_RUN registers values when the spacewire link is in the RUN state.

See the internal register description paragraph for details.

3.3.2 TX Host Interface (THI)

For the TX function, the SWB has 2 AHB interfaces:

- The master interface allows DMA transfer from the memory (or any other slave on the AHB bus) to the SpaceWire.
- The slave interface allows direct writing of data by an AHB master to the TX.

These interfaces are exclusive and the selection of the active interface is performed through the APB.

3.3.2.1 TX DMA mode

When the SpaceWire is in the TX DMA mode, the host is able to transmit a linked list of packets. The format for an element of the linked list is depicted hereafter:

Size of the packet (16 least significant bits, in bytes)
Address of the first packet data (word aligned)
Address of next linked list element (word aligned)

Tableau 3.3.2-1 linked list element

Writing the address of the descriptor list in a configuration register through the APB slave launches the DMA transfer. By knowing the size of the first packet by reading at this address, the SWB can reach the first data of the packet by reading at next address. With the size of the packet and the address of the first data, the SWB can fetch all the data of the packet. The SpaceWire can perform the same task for the next packet of the linked list. To end the linked list, the last element has a null value in its third field.

The THI makes single transfer on the AMBA AHB to retrieve the 32-bit data from the host.

For the last retrieved data corresponding to the current packet, the THI is able know the number of its valid bytes (this number depends on the packet size). In detail, considering that the data retrieved is

DAT(31:0), if 1 byte is valid, it will be DAT(31:24), if 2 bytes are valid, they will be DAT(31:24) and DAT(23:16) and so on...

The SWB inserts the End Of Packet (EOP) control character at the end of each packet.

If the retrieved packet size is null, the THI will skip the data retrieval process and looks for the next packet in the list.

The THI fetches and deliver all the data rapidly enough to keep the maximum data transfer rate (i.e. avoiding NULL character insertion : NULL character insertion must not be a consequence of the THI management of the AHB).

The TX DMA mode is only effective when the packet contains a big number of bytes.

In the worst case, the packet only contains 1 data byte. In this case, the THI has to perform 4 AHB accesses to fetch the data byte (1 access for the packet size, 1 access for the data address, 1 access for the data and 1 access for the next linked list element). Only 1 out of 4 accesses is used to retrieve the data, so NULL character insertion is inevitable. So for small packets, the host should use the TX slave mode to be effective.

If TX slave access is performed during TX DMA mode, the TX slave block will activate the WRONG_MODE interrupt and an AHB error response is delivered.

The host can monitor the progression in the linked list of packets by reading the descriptor register.

The transfer is aborted by asserting the ABORT_PACKET bit. The THI adds an EEP into the TX FIFO and erases the current data received from the host. The host should launch a new TX DMA only after the ABORT_PACKET auto-reset i.e. after the end of the abortion process.

If the THI receives an AHB error response, the TX DMA will stop retrieving data.

To restart the TX DMA after an AHB error reception, the host activates the ABORT_PACKET bit of the management register in order to properly end the current packet transmission. After the autoreset of the ABORT_PACKET bit, the host can launch the TX DMA again.

3.3.2.2 TX slave mode

The TX AHB slave interface doesn't take care of the input addresses.

After reset, the first 32-bit data sent by the host to the THI is the 16-bit size of the packet it wants to transfer (only the 16 least significant bits of the data are taken into account). The size is in bytes. If the THI receives a null packet size, it will not take it into account and will expect to receive another packet size.

Then the host can deliver the data corresponding to this packet. The SWB considers the received data as belonging to the current packet as long as the corresponding byte number does not reach the packet size. Each 32-bit data contains 4 valid bytes, the last 32-bit data contains at least 1 valid byte. Each byte is stored into the TX FIFO.

When the packet size is reached, the THI adds an EOP into the TX FIFO and expects to receive the size of the next packet.

The TX slave is able to accept burst transfer.

The host can stop the current packet transfer by asserting the ABORT_PACKET bit. When this bit is asserted, the THI adds an EEP into the TX FIFO and erases the current received data. The host should access to the TX slave only after the ABORT_PACKET auto-reset i.e. after the end of the abortion process. The next data received from the host is regarded as the size of the next packet to transmit.

When the TX FIFO is full, the THI will generate a split response (AMBA protocol) to the host if a new data transfer is requested. If a split response has been generated and another request (from the same master) is received before the transfer completion, the THI will give another split response.

When the THI receives a request from a master but cannot handle the request, it gives a split response. Before the transfer completion, if another master (different from the first one) requests the THI, the THI will response with an error message.

If the THI receives a request from a master while the link is not connected, the THI will send an error response. Then, a specific interrupt will be generated. If the split has been activated, it will be released.

Only one master should dialogue with the THI to avoid any confusion of data.

3.3.3 RX Host Interface (RHI)

3.3.3.1 The format of the storage

For the RX, the SpaceWire IP has one AHB interface. This interface allows DMA transfers from the RX FIFO to an AHB slave. The RX interface transfers data in packet format to the AHB slave. The format of a packet is described hereafter:

Header (32 bits)
Data (32 bits)
Data (32 bits) …

Tableau 3.3.3-1 Packet format

Header contents:

- bit 31 down to 18 : unused
- bit 17 down to 16 : status
- bit 15 down to 0 : packet size

The 2-bit status indicates the validity of the current packet (complete packet or incomplete packet because of link error, EEP reception or no space left in memory area). The packet size indicates the number of bytes of the packet.

The host allocates two memory areas (1 and 2) and configures 2 sets (one for each area) of three word aligned addresses in the SWB. The first address, called Start_Area Address, represents the beginning of (1 or 2) allocated area, the second address, called End_Packet Address, is close to the (1 or 2) allocated area end and the last address, called End_Area Address, is the real end of the (1 or 2) allocated area.

The couple of three addresses is written in the SpaceWire Configuration registers through the APB bus by the CPU. In addition to these addresses, the host provides a command indicating the validity of the areas.

Practically, the host validates one memory area at least to enable the Rx data transfer from the Rx FIFO to the host memory. For this, the Start_Area, End_Packet and End_Area Registers is programmed and the AREA1_VALID or/and AREA2_VALID bit(s) is set to validate the memory area(s).

The following figure shows the host memory allocation.

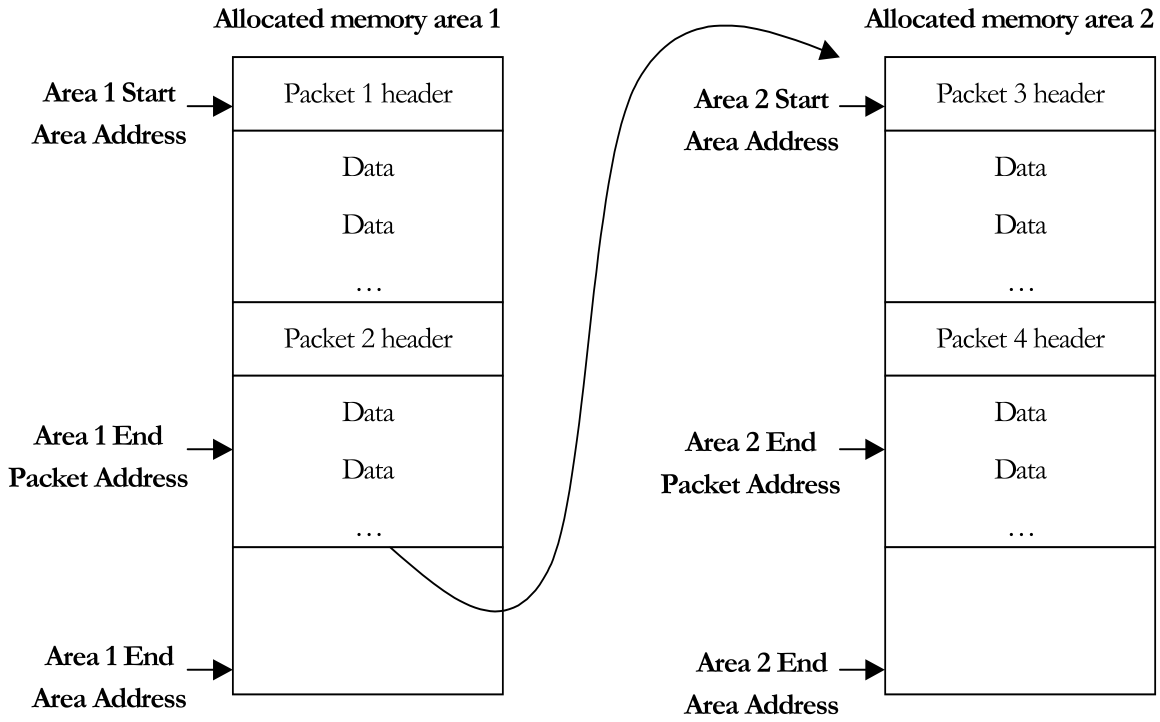


Figure 3.3.3-1 Storage format

3.3.3.2 The functionality of the RHI

The following figure shows how the RHI stores data into the host memory.

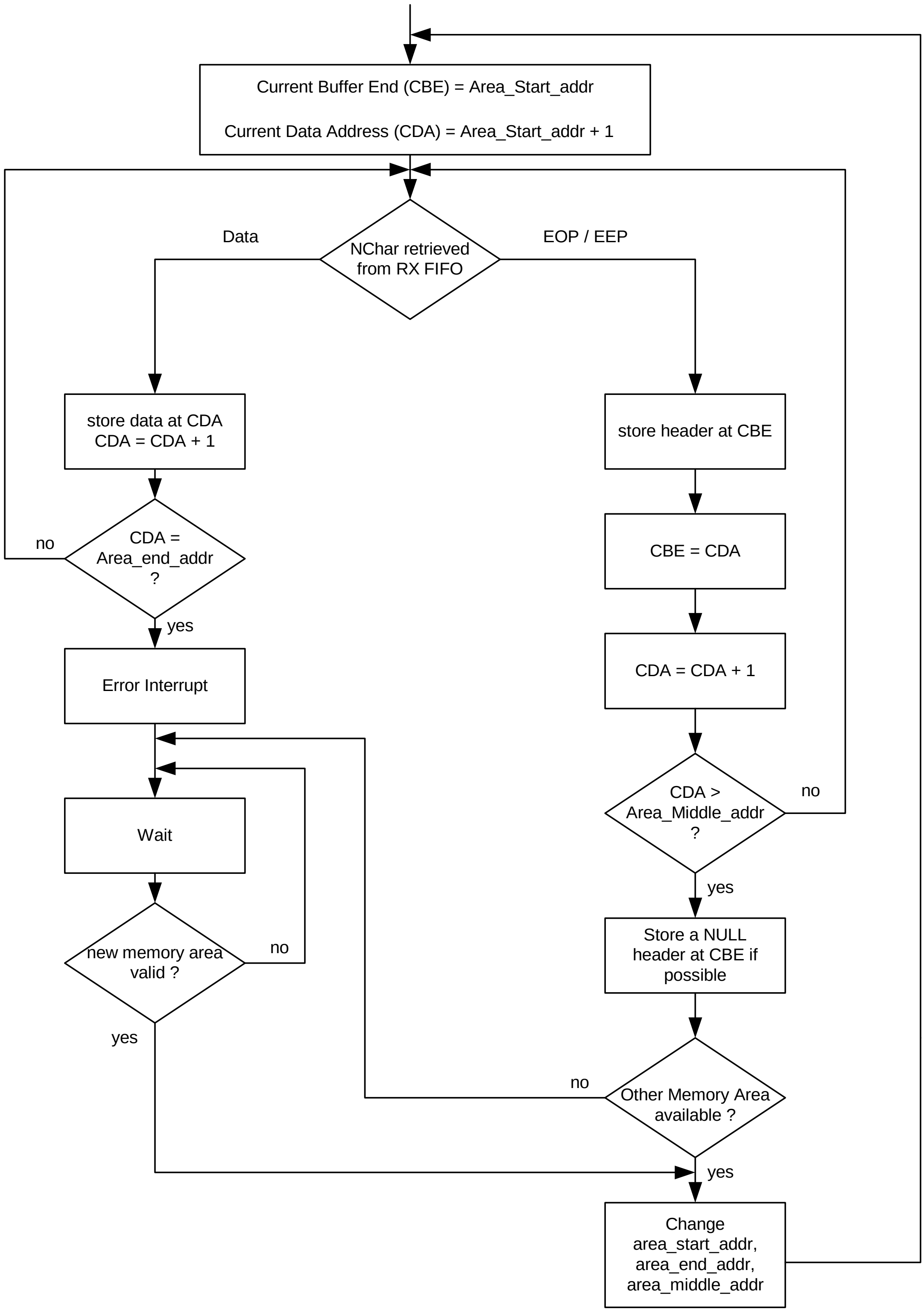


Figure 3.3.3-2 RX storage

The RHI indicates the current end of the currently used area (1 or 2), called Current Buffer End. When the area is empty, the Current Buffer End is the Start Address of this area. When an entire packet is stored into the used area, the Current Buffer End indicates the address of the next packet header. The EOP or EEP is not written into the host memory. Depending on the packet size, the last data of the packet may contain 1, 2, 3 or 4 valid bytes. The rule to determine which ones are valid is the same as in TX Host Interface. The Current Buffer End is only updated when the entire current packet is stored.

The RHI indicates the used memory area (AREA1_USED or AREA2_USED bit in the management register) so that the host can follow the storage progression by monitoring the Current Buffer End.

The RHI uses the Start Address to start writing packets in host memory in a given area (1 or 2). The RHI first attempts to use area 1. During packet data reception and up to the reception of the end of packet, the RHI leaves the packet header empty. When an end of packet is detected in the Rx FIFO, the RHI fills the header of the packet that it had just finished to write. The status will be set to "01" if an EEP has been retrieved from the Rx FIFO, it will be set to "10" if there is no space left in the memory area to complete the packet transfer, otherwise the status bit will be "00". The packet size is filled with the number of bytes of the packet.

In case the RX receives data and no area is allocated, the SWB generates the NO_AREA_VALID interrupt.

If the SWB receives a NULL character and the link is not enabled, the LINK_NOT_ENABLED interrupt will be generated to warn the host.

3.3.3.3 Reaching the End_Packet Address

If the RHI reaches the End_Packet Address during a packet transfer, it will write a null header into the current memory area after the last data of the current packet. If no space is available, the null header will not be written. Then the current AREA1_USED or AREA2_USED bit is reset. The SWB then invalidates the current memory area by resetting the AREA1_VALID or AREA2_VALID bit. The re-validation of the area or the definition of a new area is in charge of the host.

If the other allocated area is valid, the RHI will continue transferring the next packet into the other available allocated area. If the other area is not valid, the RHI will generate the NO_AREA_VALID interrupt and stops the packet transfer until the host provides another available area.

3.3.3.4 Reaching the End_Area Address

If the RHI reaches the End_Area Address before ending writing the current packet, the RHI will write the packet header with the current number of bytes written in the buffer as packet size and with the status "10" indicating that the packet contains no error but is incomplete. **The status "10" will be generated even if all the packet data are written**.

The EXCEED_MEM interrupt is activated.

The remaining data of the packet is written in another available memory area and is considered as an entire packet. So it is the host responsibility to concatenate the beginning of the packet with the end of the packet (the packet can be split between areas 1 and 2) or to perform any other recovery actions (link restart,...).

3.3.3.5 AHB error occurrence

When the RHI receives an error response, the data storage stops without ending the current packet storage. Both memory areas become invalid.

The storage starts again when a memory area is validated. The last part of the incompletely stored packet is written into the new valid memory area.

3.3.3.6 Advice

The space between the End Packet Address and the End Area Address should be at least equal to the maximal size of a packet (expected to be received by the host) in order to guarantee a safe protocol.

3.3.4 Format of the words stored in the TX and RX FIFOs:

The TX and RX FIFOs contain 9-bit words.

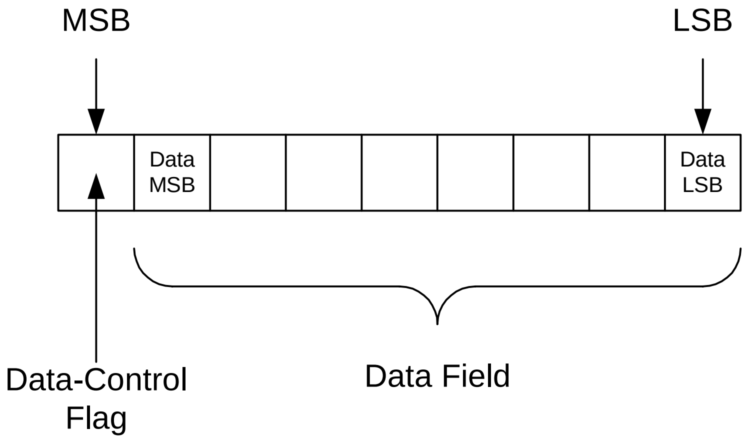


Figure 3.3.4-1 FIFO word format

	Control flag	Data Bits(MSB…LSB)	Meaning
TX FIFO	0	XXXXXXXX	8-bit data
	1	XXXXXXX0	EOP
	1	XXXXXXX1	EEP
RX FIFO	0	XXXXXXXX	8-bit data
	1	00000000	EOP
	1	00000001	EEP

Tableau 3.3.4-1 Word meaning

See the AD11 for details.

3.3.5 The interrupts

The Internal Register Description paragraph shows all the interrupts. An interrupt remains active until its reset.

When an interrupt is asserted, the host should perform all the corresponding tasks before clearing this interrupt by writing into the IT_RESET register.

The ITMASKREG allows inhibiting the output interrupt signals (nominal interrupt and error interrupt) but doesn't inhibit the interrupt register. Interrupt register bits will still be set.

3.3.6 Time Code transmission and reception

To send a time code, the host either generates a pulse on the TICKIN_CTM input signal or asserts the TICKIN bit of the management register. A time code is sent when a rising edge is detected on TICKIN_CTM or TICKIN.

It is possible to initialise the time code value by writing in the time code register (TIMESEND_REG byte).

When a correct time code is received, the SWB generates a pulse on the TICKOUT_CTM output signal and the TICKOUT interrupt is asserted.

The received time code value is in the time code register (TIMEREC_REG byte).

3.3.7 Test mode

The test mode is activated when TEST_MODE_HARD and TEST_MODE_SOFT are high.

The test mode allows invalidating the host memory areas. See the Internal Register Description paragraph.

3.4 INTERNAL REGISTER DESCRIPTION

All the SWB registers are accessible through the APB interface. The input address is interpreted as a byte address, as per AD9. Since each access is a word access, the two least significant address bits are assumed always to be zero. Only address bits 6:2 are decoded. Misaligned addressing is not supported. For read accesses, data output is produced combinatorially from the address.

3.4.1 Global description

Register name	Address (Hex)	Read/Write	Remark	Reference
Management	------00	r/w	link management and status.	Tableau 2
Interrupt	------04	r	Interrupt status.	Tableau 3
Current Buffer End	------08	r	Current end of the used memory area.	Tableau 4
Start Address 1	------0C	r/w	Memory area 1 start address	Tableau 5
Start Address 2	------10	r/w	Memory area 2 start address	Tableau 6
Middle Address 1	------14	r/w	Memory area 1 packet end	Tableau 7
Middle Address 2	------18	r/w	Memory area 2 packet end	Tableau 8
End Address 1	------1C	r/w	Memory area 1 end address	Tableau 9
End Address 2	------20	r/w	Memory area 2 end address	Tableau 10
Descriptor	------24	r/w	First address of the linked list of packets	Tableau 11
Time Out	------28	r/w	Time out programmation	Tableau 12
Interrupt Mask	------2C	r/w		Tableau 13
Interrupt reset	------30	w		Tableau 14
Interrupt set	------34	w		Tableau 15
Time Code	------38	r/w	Time code programmation and status	Tableau 16
Additional status register	------3C	r		Tableau 17

3.4.2 Detailed description

Management Register: address 00H

Bits	Name	Reset Value	Function	r/w
31-24	FREQ_INIT	0	Configuration of the TX clock frequency used during the link initialisation. See 5.2.1. ➢ In the gated TX clock configuration, the input TX clock frequency is divided by **2(FREQ_INIT+1)** then is used as TX frequency. ➢ In the not-gated TX clock configuration, the input TX clock frequency is divided by **(FREQ_RUN+1).**	r/w
23-16	FREQ_RUN	0	Configuration of the TX clock frequency used after the link initialisation. See 5.2.1. ➢ In the gated TX clock configuration, the input TX clock frequency is divided by **2(FREQ_RUN+1)** then is used as TX frequency if TX_MAX_EN=0. ➢ In the not-gated TX clock configuration, the input TX clock frequency is divided by **(FREQ_RUN+1).**	r/w
15-13	ST_TRANS	0	Status showing the link initialisation progression. 0 to 4: initialisation state (0=ErrorReset, 1=ErrorWait, 2=Ready, 3=Started, 4=Connecting) 5: run state	r
12	TICKIN	0	The host can transmit a time code by asserting this bit. Writing a '1' launches the time code transmission Writing a '0' has no effect	w
11	LINK_DISABLED	0	The link is disabled. 0: link not disabled 1: link disabled	r/w
10	LINK_START	0	The link can start. 0: link not started 1: link started	r/w
9	AUTOSTART	0	The link automatically starts when a NULL character is received. 0: autostart off 1: autostart on	r/w
8	TX_MAX_EN	0	**This bit is only used with the gated TX clock configuration**. In run state, the TX frequency used will be the same as the input TX clock frequency if this bit is	r/w

Bits	Name	Reset Value	Function	r/w
			asserted. See 5.2.1. 0: max TX frequency Off 1: max TX frequency On	
7	DMA_RUNNING	0	This bit indicates if the DMA is running or not. 0: DMA is not running 1: DMA is running	
6	AHB_MODE_TX	0	The host has 2 possibilities to transfer data to the SWB. 0: TX AHB slave 1: TX AHB master (DMA)	r/w
5	TEST_MODE_SOFT	0	When TEST_MODE_HARD input signal and TEST_MODE_SOFT are asserted, the test mode is active.	r/w
4	ABORT_PACKET	0	Abortion of the data transfers. Writing a '1' launches the abortion process. **Writing a '0' has no effect**. This bit is automatically reset when the abortion process ends.	r/w
3	AREA2_USED	0	This bit is asserted when the host memory area 2 is used to store the data from the SWB.	r
2	AREA1_USED	0	This bit is asserted when the host memory area 1 is used to store the data from the SWB.	r
1	AREA2_VALID	0	This bit validates the area 2. So the SWB can use the area 2 to store data. Writing a '1' validates the area 2. **Writing a '0' has no effect.** This bit is automatically reset when the area 2 is full. **In test mode, writing a '0' will reset this bit.**	r/w
0	AREA1_VALID	0	This bit validates the area 1. So the SWB can use the area 1 to store data. Writing a '1' validates the area 1. **Writing a '0' has no effect.** This bit is automatically reset when the area 1 is full. **In test mode, writing a '0' will reset this bit.**	r/w

Tableau 2

Warning 1: To let unchanged the validity of the AREA1 and AREA2, the user must write '0' in bit 0 and bit 1 each time a write access is performed in this register.

Warning 2: To prevent accidental transfer abortion, the user must write '0' in bit 4 each time a write access is performed in this register.

Interrupt Register: address 04H

Bits	Name	Reset Value	Function	r/w
16	PACKET_REC	0	This bit is asserted when a complete packet has been received. *Nominal Interrupt*	r
15	LINK NOT ENABLED	0	This bit is asserted when a NULL character is received but the link is not enabled (LINK_DISABLED On or LINK_START Off and AUTOSTART Off) *Nominal Interrupt*	r
14	EXCEED_MEM	0	When the SWB cannot store a packet entirely because the host memory area is full, this bit is asserted. *Nominal Interrupt*	r
13	TICKOUT	0	This bit is asserted when a right time code has been received. *Nominal Interrupt*	r
12	END_LIST	0	In TX AHB master mode, when the SWB reaches the end of the linked list of packets, this bit is asserted. *Nominal Interrupt*	r
11	NO_AREA_VALID	0	When data have been received but any host memory area is available, this bit is asserted. *Nominal Interrupt*	r
10	WRONG_MODE	0	Writing in the descriptor register (24H) while in TX AHB slave mode or writing in the TX AHB slave while in TX AHB master mode will assert the WRONG_MODE bit. *Error Interrupt*	r
9	RD_ACCESS_ERROR	0	A read access to the TX AHB slave will assert this bit. A AHB error response will be generated. *Error Interrupt*	r
8	AMBA_ERROR	0	This bit is asserted when the SWB receives a AHB error response. *Error Interrupt*	r
7	LINK_NOT_READY	0	This bit is asserted when the TX AHB slave is selected and the link is not connected. *Error Interrupt*	
6	EEP_REC	0	This bit is asserted when a EEP has been received. *Error Interrupt*	r
5	CREDIT_ERR	0	This bit is asserted when the SWB receives a FCT but the increment of the credit counter will exceed 56. *Error Interrupt*	r
4	OUTSTAND_ERR	0	This bit is asserted when the SWB receives a data but is not waiting for any one (outstanding counter at 0).	r

Bits	Name	Reset Value	Function	r/w
			Error Interrupt	
3	CHAR_SEQ_ERR	0	This bit is asserted when a character sequence error is detected. *Error Interrupt*	r
2	DISCONNECT_ERR	0	This bit is asserted when a link disconnection is detected. *Error Interrupt*	r
1	PARITY_ERR	0	This bit is asserted when a parity error is detected. *Error Interrupt*	r
0	ESC_ERR	0	This bit is asserted when a received ESC character is followed by neither a FCT nor a data. *Error Interrupt*	r

Tableau 3

The active value of an interrupt is '1'.

Current Buffer End Register: address 08H

Bits	**Name**	**Reset Value**	**Function**	**r/w**
31-0	CUR_BUF_END	0	This address pointer indicates the current end of the used host memory area. All addresses between the start address value and the address pointer value (address pointer value not included) contain valid data.	r

Tableau 4

Start Address 1 Register: address 0CH (see note 1)

Bits	**Name**	**Reset Value**	**Function**	**r/w**
31-0	START_AREA1	0	This address pointer indicates the start address of the host memory area 1.	r/w

Tableau 5

Start Address 2 Register: address 10H (see note 1)

Bits	**Name**	**Reset Value**	**Function**	**r/w**
31-0	START_AREA2	0	This address pointer indicates the start address of the host memory area 2.	r/w

Tableau 6

Middle Address 1 Register: address 14H (see note 1)

Bits	Name	Reset Value	Function	r/w
31-0	END_PAC1	0	This address pointer indicates the middle address of the host memory area 1.	r/w

Tableau 7

Middle Address 2 Register: address 18H (see note 1)

Bits	Name	Reset Value	Function	r/w
31-0	END_PAC2	0	This address pointer indicates the middle address of the host memory area 2.	r/w

Tableau 8

End Address 1 Register: address 1CH (see note 1)

Bits	Name	Reset Value	Function	r/w
31-0	END_AREA1	0	This address pointer indicates the end address of the host memory area 1.	r/w

Tableau 9

End Address 2 Register: address 20H (see note 1)

Bits	Name	Reset Value	Function	r/w
31-0	END_AREA2	0	This address pointer indicates the end address of the host memory area 2.	r/w

Tableau 10

Descriptor Register: address 24H

Bits	Name	Reset Value	Function	r/w
31-0	DESC_ADDR	0	By writing in this register, the host gives the first element address of the linked list of packets. For a nominal operation, the host **must not** write again in this register before the END_LIST interrupt activation. By reading in this register, the host can monitor the progression in the linked list.	r/w

Tableau 11

Time-Out Register: address 28H (see note 2)

Bits	Name	Reset Value	Function	r/w
23-16	DIS_CNT_LIM	FF	This is the time out for the link disconnection detection. DIS_CNT_LIM = 1 means time out = 1 system clock period The DIS_CNT_LIM will take a definite value so that the time-out is set to 850 ns.	r/w
15-8	RESERVED		Not used for the moment. These bits will be used if the DELAYWIDTH constant becomes higher than 8 (Refer to RD22)	
7-0	DELAY_6_4	FF	This is the 6.4 µs time out used in the link initialization protocol. The user will give a value so that this time out is about 6.4 µs. DELAY_6_4 = 1 means time out = 1 system clock period	r/w

Tableau 12

Interrupt Mask Register: address 2CH

Bits	Name	Reset Value	Function	r/w
15-0	ITMASKREG	1	This register masks the interrupts. bit at '0' : the corresponding interrupt is masked bit at '1' : the corresponding interrupt is not masked	r/w

Tableau 13

Interrupt Reset: address 30H

Bits	Name	Reset Value	Function	r/w
15-0	IT_RESET	-	The host can reset the interrupts by writing at this address. Writing a '0' : the corresponding interrupt is reset Writing a '1' : **no effect** on the corresponding interrupt	w

Tableau 14

Interrupt Set: address 34H

Bits	Name	Reset Value	Function	r/w
15-0	IT_SET	-	The host can set the interrupts by writing at this address. Writing a '0' : **no effect** on the corresponding interrupt Writing a '1' : the corresponding interrupt is set	w

Tableau 15

Time Code Register: address 38H

Bits	Name	Reset Value	Function	r/w
15-8	TIMESEND_REG	0	The host can initiate the time code value to send.	r/w
7-0	TIMEREC_REG	0	This register contains the received time code value.	r

Tableau 16

<u>Note 1:</u> For this register, the value must be an address aligned with a 32-bit data.

Additional Status Register: address 3CH

Bits	Name	Reset Value	Function	r/w
17	FIFO_FULL	0	In AHB slave mode, this flag indicates the state of the 32-bit AHB FIFO. ➢ 0: not full ➢ 1: full	
16	GATED_TX_CLOCK	-	Indicates the TX clock configuration: ➢ 0: Not-gated TX clock configuration ➢ 1: Gated TX clock configuration	
15-14	UNUSED			
13-8	OUTSTANDING_CNT	0	Outstanding counter value.	r
7-6	UNUSED			
5-0	CREDIT_CNT	0	Credit counter value.	r

Tableau 17

3.5 INTERFACE DESCRIPTION

3.5.1 Clocks, test and reset

Signal name	I/O	Description	Active value
clk_txin	I	Max transmission clock. Lower TX frequency is a division of this clock.	-
clk_sw	I	system clock	-
test_mode_hard	I	asynchronous signal to activate the test mode	1
resetn	I	asynchronous reset	0

3.5.2 APB interface

Signal name	I/O	Description	Active value
apb_slv_in.PWDATA(31-0) apb_slv_in.PSEL apb_slv_in.PENABLE apb_slv_in.PADDR(31-0)	I	AMBA APB bus in. (Sampled on the clk_sw rising edge)	-
apb_slv_out.PRDATA(31-0)	O	AMBA APB bus out. (generated on the clk_sw rising edge)	-

3.5.3 TX AHB master interface

Signal name	I/O	Description	Active value
tx_ahb_mst_in.HGRANT tx_ahb_mst_in.HREADY tx_ahb_mst_in.HRESP(1-0) tx_ahb_mst_in.HRDATA(31-0) tx_ahb_mst_in.HCACHE	I	AMBA AHB master bus in for the TX host interface. (Sampled on the clk_sw rising edge)	-
tx_ahb_mst_out.HBUSREQ tx_ahb_mst_out.HTRANS tx_ahb_mst_out.HADDR(31-0) tx_ahb_mst_out.HWRITE	O	AMBA AHB master bus out for the TX host interface. (generated on the clk_sw rising edge)	-

tx_ahb_mst_out.HSIZE(2-0) tx_ahb_mst_out.HBURST(2-0) tx_ahb_mst_out.HPROT(3-0) tx_ahb_mst_out.HWDATA(31-0)			

3.5.4 TX AHB slave interface

Signal name	I/O	Description	Active value
tx_ahb_slv_in.HSEL tx_ahb_slv_in.HWRITE tx_ahb_slv_in.HADDR(31-0) tx_ahb_slv_in.HTRANS(1-0) tx_ahb_slv_in.HWDATA(31-0) tx_ahb_slv_in.HREADY tx_ahb_slv_in.HSIZE(2-0) tx_ahb_slv_in.HMASTER(3-0) tx_ahb_slv_in.HMASTLOCK tx_ahb_slv_in.HBURST(2-0) tx_ahb_slv_in.HPROT(3-0)	I	AMBA AHB slave bus in for the TX host interface. (Sampled on the clk_sw rising edge)	-
tx_ahb_slv_out.HREADY tx_ahb_slv_out.HRESP(1-0) tx_ahb_slv_out.HRDATA(31-0) tx_ahb_slv_out.HSPLIT(15-0)	O	AMBA AHB slave bus out for the TX host interface. (generated on the clk_sw rising edge)	-

3.5.5 RX AHB master interface

Signal name	I/O	Description	Active value
rx_ahb_mst_in	I	AMBA AHB master bus in for the RX host interface. (Sampled on the clk_sw rising edge)	-
rx_ahb_mst_out	O	AMBA AHB master bus out for the RX host interface. (generated on the clk_sw rising edge)	-

3.5.6 Link interface

Signal name	I/O	Description	Active value
d_in	I	asynchronous input data signal	-
s_in	I	asynchronous input strobe signal	-
d_out	O	output data signal. (Generated on the internal TX clock rising edge)	-
s_out	O	output strobe signal. (Generated on the internal TX clock rising edge)	-

3.5.7 Time interface

Signal name	I/O	Description	Active value
tickin_ctm	I	signal to send time code. (Sampled on the clk_sw rising edge)	1
tickout_ctm	O	right time code received. (Generated on the clk_sw rising edge)	1

3.5.8 Interrupt interface

Signal name	I/O	Description	Active value
err_int	O	output error interrupt. (Generated on the clk_sw rising edge)	1
nom_int	O	output error interrupt. (Generated on the clk_sw rising edge)	1

4 PERFORMANCE

The TX frequency can be up to 4 times the system clock frequency.

The RX rate can be up to 4 times the system clock frequency.

The Time Code transmission order is taken into account 1 system clock period after its generation. But the real Time Code transmission depends on the length of the current transmitted character.

When data have to be transmitted, no Null character is transmitted between 2 data transmissions. But when the TX master mode is used and the packet size is too small, Null character can be transmitted because of the time lost to retrieve the packet size and the data address. If any bus request is immediately granted, the minimum packet size will be 6 in order to avoid Null character transmission between 2 data transmissions (at maximum TX frequency).

In TX slave mode, to ensure that the TX FIFO does not become empty, the host sends one data word at least at each 10 system clock periods. So, it prevents the Null character transmission between 2 data transmissions.

If the AHB bus is immediately granted, it will take at the most 9 system clock cycles to build a 32-bit word with 4 received 8-bit data and to store it into the host memory.

5 ARCHITECTURE DESCRIPTION

The figure below gives an overview of the SWB architecture.

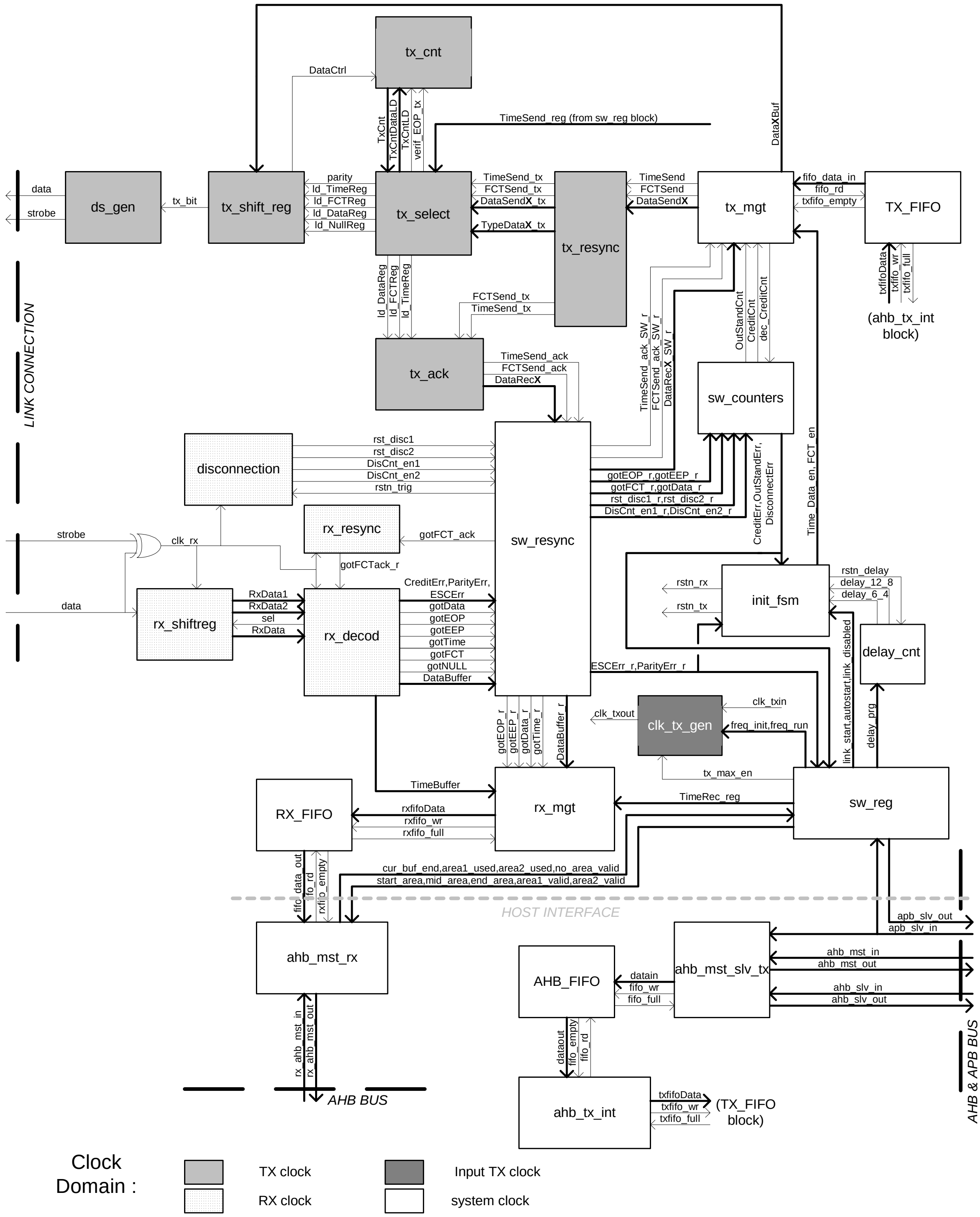


Figure 3.5.8-1 global architecture

The above schema also describes the clock trees. The TX clock used for the transmission is made from the input TX clock. The RX clock is made from the data and strobe input signals.

<u>Top input signals description:</u>

clk_sw	:	SpaceWire clock
clk_txin	:	Max Tx clock
resetn	:	asynchronous reset
tickin_ctm	:	time code to send
d_in	:	data input
s_in	:	strobe input
apb_slv_in	:	APB slave
tx_ahb_slv_in	:	AHB SLAVE
tx_ahb_mst_in	:	AHB MASTER
rx_ahb_mst_in	:	AHB MASTER
test_mode_hard	:	test mode asserted by hardware

<u>Top output signals description:</u>

clk_txout	:	Tx clock for test (disabled for timing performance)
tickout_ctm	:	right time code received
d_out	:	data output
s_out	:	strobe output
apb_slv_out	:	APB slave
tx_ahb_slv_out	:	TX AHB SLAVE
tx_ahb_mst_out	:	TX AHB MASTER
rx_ahb_mst_out	:	RX AHB MASTER
err_int	:	Error interrupt
nom_int	:	Nominal interrupt

5.1 DESCRIPTION OF THE RESET TREES

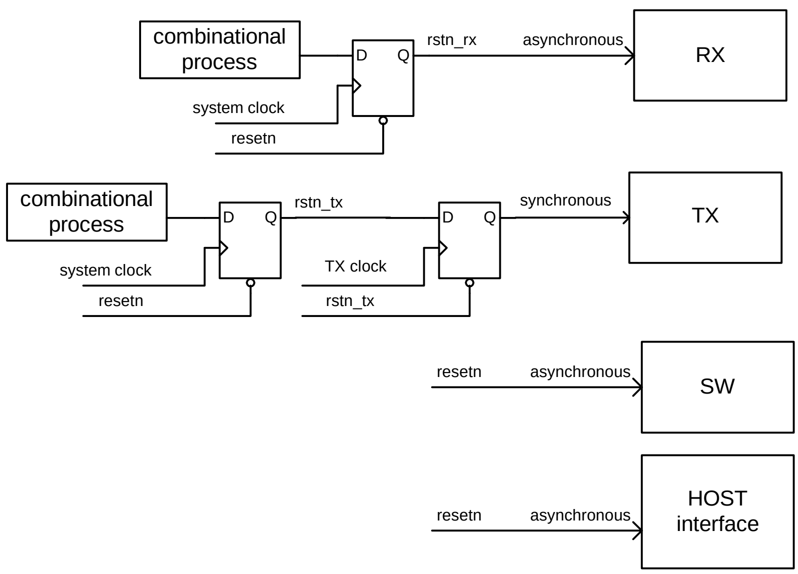


Figure 3.5.8-1 Reset trees

The RX block includes all blocks working at RX clock.

The TX block includes all blocks working at TX clock.

The SW and host interface blocks include all blocks working at system clock.

5.2 BLOCKS WORKING AT TX CLOCK

5.2.1 CLK_TX_GEN block

There are two different architectures for the CLK_TX_GEN block:

- The first one is a gated TX clock. The generated TX clock has a various frequency following the 2(n+1) frequency divider.
- The second one is a not-gated TX clock. The generated TX clock has a constant and equal frequency to the input TX clock. The use of an enable signal (clk_tx_en) allows the TX frequency variation. A (n+1) frequency divider is used.

The GATED_TX_CLK parameter selects the CLK_TX_GEN block architecture:

- GATED_TX_CLK = True : gated TX clock is used.
- GATED_TX_CLK = False : not-gated TX clock is used.

5.2.2 DS_GEN block

Input signals description

clk_tx : clock
rstn_tx : reset
DataIn : data in

Output signals description

D : data signal out
S : strobe signal out

The goal is to generate the data and strobe signals according to the AD11 specification.

5.2.3 TX_SHIFT_REG block

Input signals description

rstn_tx : reset
clk_tx : clock
LD_TimeReg : load time code register
LD_FCTReg : load FCT register
LD_DataReg : load data register
LD_NULLReg : load NULL register
TypeData1 : type data or EOP
TypeData2 : type data or EOP
TypeData3 : type data or EOP
TypeData4 : type data or EOP
parity : parity bit
tx_mux(2:0) : mux control. Selection of the character to be serialized
TimeDataLD(7:0): time code to load

Data1LD(8:0) : data to load from buffer1
Data2LD(8:0) : data to load from buffer2
Data3LD(8:0) : data to load from buffer3
Data4LD(8:0) : data to load from buffer4

Output signals description

DataSend_sel(1:0): data buffer select
DataCtrl : data control flag to differentiate data from EOP or EEP
tx_bit : bit to transmit

This block receives orders to load the shift registers then transmits the serial TX_BIT signal to the DS_GEN block.

As there are 4 data buffers (Data1LD(8:0), Data2LD(8:0), Data3LD(8:0) and Data4LD(8:0)), the block swaps from one to another each time a data is loaded. The DataSend_sel signal indicates which data is selected.

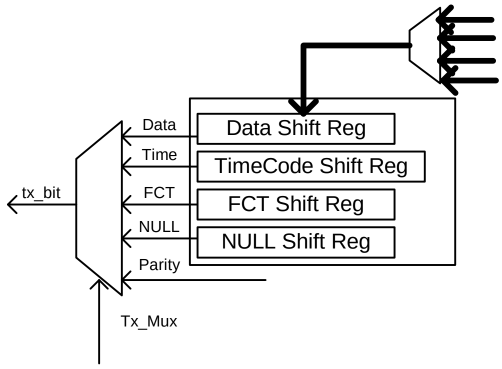


Figure 5.2.3-1 TX shift registers

5.2.4 TX_SELECT block

Input signals description

rstn_tx : reset
TimeSend_tx : time code to send
FCTSend_tx : FCT to send
DataSend1_tx : data from buffer1 to send
DataSend2_tx : data from buffer2 to send
DataSend3_tx : data from buffer3 to send
DataSend4_tx : data from buffer4 to send
DataSend_sel(1:0): data buffer select
TypeData1 : type data or EOP
TypeData2 : type data or EOP
TypeData3 : type data or EOP
TypeData4 : type data or EOP
TxCnt(3:0) : position of the current transmitted character
tx_bit : TX data bit
TimeSend_ack : TimeSend acknowledge
FCTSend_ack : FCTSend acknowledge

Output signals description

tx_mux(2:0) : character select

TxCntDataLD(3:0) : data to load
LD_TimeReg : load time code
LD_FCTReg : load FCT
LD_DataReg : load data
LD_NULLReg : load NULL
verif_EOP_tx : check if data or EOP
parity : i_parity bit
TxCntLD : load txcnt

This block manages the character transmission requests from the TX_MGT block. Following the priority order (time code > FCT > data > NULL), the TX_SELECT block generates the appropriate load signal to the TX_SHIFT_REG block.

The TX_SELECT block also activates the data or EOP/EEP check performed by the TX_CNT. Then this block manages the TX_CNT load.

The parity bit is computed in this block.

5.2.5 TX_CNT block

<u>Input signal description</u>

rstn_tx : reset
TxCntLD : Load command
DataCtrl : character control bit (data or EOP/EEP)
verif_EOP_tx : check if data or EOP to update the counter
TxCntDataLD(3:0): data to load

<u>Output signal description</u>

CntOut(3:0) : counter value

This 4-bit counter is used to count the characters length. Thus, the TX_SLECT block can generate the load signals in appropriate time.

This counter loads the TxCntDataLD value when the TxCntLD signal is high. The counter is corrected when an EOP/EEP is checked.

5.2.6 TX_ACK block

<u>Input signals description</u>

rstn_tx : reset
DataSend_sel(1:0): data buffer select
FCTSend_tx : FCT to send
LD_FCTReg : load FCT register
LD_DataReg : load Data register
TimeSend_tx : time code to send
LD_TimeReg : load time code register

<u>Output signals description</u>

DataRec1 : DataSend acknowledge for buffer1
DataRec2 : DataSend acknowledge for buffer2
DataRec3 : DataSend acknowledge for buffer3

DataRec4 : DataSend acknowledge for buffer4
TimeSend_ack : TimeSend acknowledge
FCTSend_ack : FCTSend acknowledge

This block generates acknowledgement signals for the time code, FCT and data requests. The acknowledgement is activated when the corresponding shift register from the TX_SELECT block is loaded.

The DataRec1/DataRec2/DataRec3/DataRec4 signals is activated for 4 TX clock cycles, then is automatically off after this time period.

After the activation of the TimeSend_ack/ FCTSend_ack signal, the deactivation is performed only when the TimeSend_tx/ FCTSend_tx signal is low.

5.2.7 TX_RESYNC block

Input signals description

rstn_tx : reset
DataSend1 : data from buffer1 to send
DataSend2 : data from buffer2 to send
DataSend3 : data from buffer3 to send
DataSend4 : data from buffer4 to send
FCTSend : FCT to send
TimeSend : Time Code to send
TypeData1 : Data or EOP
TypeData2 : Data or EOP
TypeData3 : Data or EOP
TypeData4 : Data or EOP

Output signals description

TypeData1_tx : resynchronised signal
TypeData2_tx : resynchronised signal
TypeData3_tx : resynchronised signal
TypeData4_tx : resynchronised signal
DataSend1_tx : resynchronised signal
DataSend2_tx : resynchronised signal
DataSend3_tx : resynchronised signal
DataSend4_tx : resynchronised signal
FCTSend_tx : resynchronised signal
rstn_tx_r : resynchronised signal
TimeSend_tx : resynchronised signal

This block performs the resynchronisation of the signals from blocks working at the system clock following the below architecture.

combinational process
D Q signal
D Q
D Q signal_r
clock 1
asynchronous reset
clock 2
asynchronous reset

5.3 BLOCKS WORKING AT RX CLOCK

The RX clock is built from the DATA and STROBE signals as shown hereafter:

Data
Strobe
RX clock

Figure 5.2.7-1 RX clock generation

5.3.1 RX_SHIFTREG block

<u>Input signals description</u>

rstn_rx	:	asynchronous resetn
d	:	data in
sel	:	select RxData1 or RxData2

<u>Output signals description</u>

RxData1(9:0)	:	data with first bit detected on falling edge
RxData2(9:0)	:	data with first bit detected on rising edge
RxData(9:0)	:	RxData1 or RxData2, depending on the NULL detection

This block memorizes the input serial data on the rising and falling edge of the RX clock.

The RX_SHIFTREG block contains 2 shift registers. The one works on rising edge, the other on the falling edge.

The character can be received with its first bit sampled on falling or rising edge. So, RxData1(9:0) and RxData2(9:0) are used to determine on which edge the first bit is sampled.

This detection is only performed for the first NULL character. The first bit of the following characters is sampled on the same edge.

The RxData(9:0) word is either RxData1(9:0) or RxData2(9:0) following the SEL signal value which depends on the first bit detection on rising/falling edge.

The SEL signal is determined when the first NULL is detected and will remain unchanged as long as the link is running.

So, to detect the first NULL character, RxData1(9:0) and RxData2(9:0) are used. Then to detect the following characters, RxData(9:0) word is used.

The architecture of RX_SHIFTREG is shown hereafter:

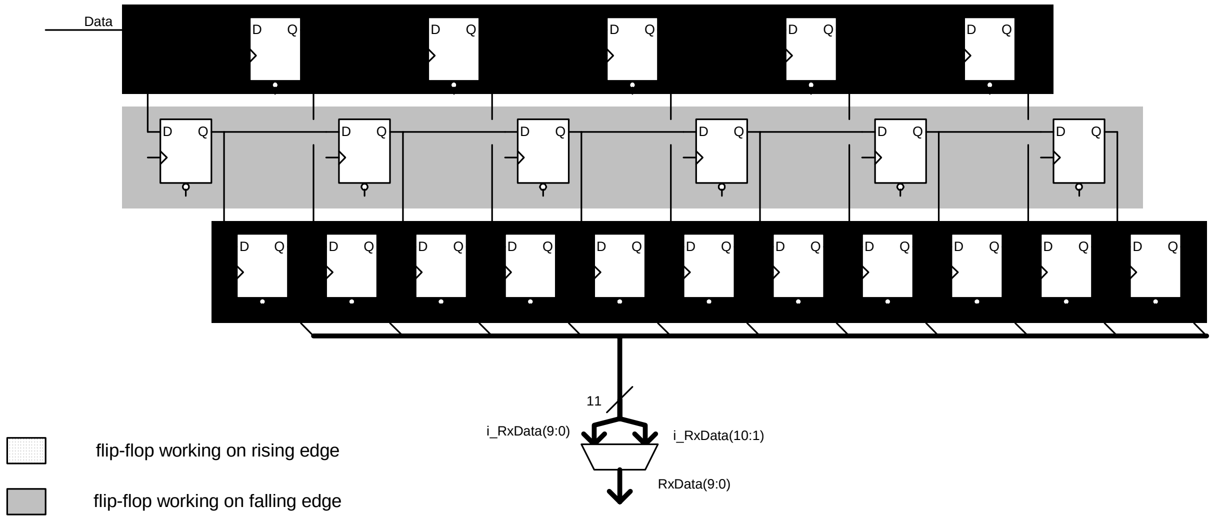


Figure 5.3.1-1 RX shift registers

5.3.2 RX_DECOD block

Input signals description

rstn_rx	:	asynchronous reset
RxData1(9:0)	:	data with first bit detected on falling edge
RxData2(9:0)	:	data with first bit detected on rising edge
RxData(9:0)	:	RxData1 or RxData2, depending on the NULL detection
gotFCT_ack_r	:	gotFCT acknowledge

Output signals description

sel	:	select RxData1 or RxData2
gotData	:	data received in Databuffer
gotEOP	:	EOP received
gotEEP	:	EEP received
gotTime	:	time code received in TimeBuffer
TimeBuffer(7:0)	:	time code
DataBuffer(7:0)	:	data
ParityErr	:	parity error
CreditErr	:	credit error
ESCErr	:	ESC error
gotNULL	:	got first NULL
gotFCT	:	FCT received

The RX_DECOD block contains a 3-bit counter to note the number of FCT received.

Another 3-bit counter is used to determine the time that the character remains in the shift register (RX_SHIFTREG block).

The RX_DECOD block identifies the character type and verifies the parity.

When the parity is checked, the valid received character is flagged by a signal (gotData, gotEOP, gotEEP, gotTime, gotFCT or gotNULL). The gotData, gotEOP, gotEEP or gotTime is asserted for 2 RX clock cycles each time the corresponding character is received. The gotNULL is always asserted after the first NULL character reception.

As long as the 3-bit FCT counter value is not null, the gotFCT signal is generated. This signal uses a handshake protocol. Each time the gotFCT acknowledgement is received, the FCT counter is decremented and the gotFCT signal deasserted.

If the received character is a time code or a data, the value will be stored into the TimeBuffer or DataBuffer.

The block also generates 3 error signals. When the parity is false, the ParityErr signal is produced. The ESCErr indicates that a ESC is not followed by a FCT or a Data. Here, the CreditErr is asserted when the number of received FCTs is out of limit (>7). The SW_COUNTERS block also generates a CreditErr signal which depends on the number of data to be transmitted.

5.3.3 RX_RESYNC block

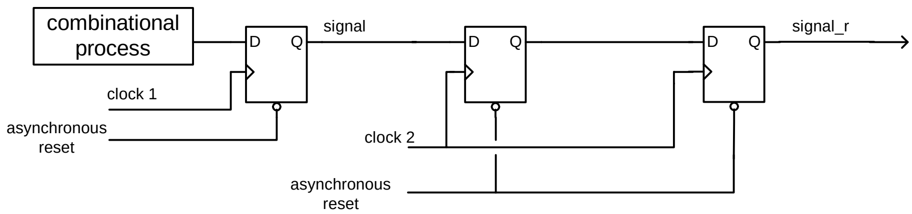
Input signals description

rstn_rx : asynchronous reset
gotFCT_ack : gotFCT acknowledge

Output signal description

gotFCT_ack_r : resynchronized signal

This block resynchronizes the signals from blocks working at system clock following the below architecture.



5.3.3.1 DISCONNECTION block

Input signals description

rstn_rx: Rx asynchronous reset
rstn_trig: specific asynchronous reset

Output signals description

rst_disc1: reset when link disconnected (on rising edge)
rst_disc2: reset when link disconnected (on falling edge)
DisCnt_en1: DisCnt counter enable
DisCnt_en2: DisCnt counter enable

The DISCONNECTION block produces the reset and enables signals for the counter used to detect the link disconnection.

After the RX reset, the counter is enabled on the first edge of the RX clock. So, there are 2 enable signals (DisCnt_en1 and DisCnt_en2). The one is asserted on the rising edge of the RX clock, the other on the falling edge. Once they are asserted, the DisCnt_en1 and DisCnt_en2 signals remain activated.

Each time an edge of the RX clock occurs, the rst_disc1 or rst_disc2 signal is asserted to reset the counter. Then they are de-asserted once the counter is reset.

5.4 BLOCKS WORKING AT SYSTEM CLOCK

5.4.1 INIT_FSM block

Input signals description

resetn : asynchronous reset
delay_6_4 : delay of 6.4 μs
delay_12_8 : delay of 12.8 μs
CreditErr_rx_r : Credit error from Rx
CreditErr : Credit error from CreditCnt
OutstandErr : Credit error from OutstandCnt
ParityErr_r : parity error
ESCErr_r : ESC error
DisconnectErr : link disconnection
gotData_r : got data in buffer
gotEOP_r : got EOP
gotEEP_r : got EEP
gotTime_r : got time code in buffer
gotFCT_r : got FCT
gotNULL_r : got first NULL
link_disabled : link disabled
link_start : link start
autostart : link auto start

Output signals description

st_trans(2:0) : state transition for test
sel : selects the init frequency or the run frequency for the TX clock
FCT_en : FCT enabled
Time_Data_en : Time code and Data enabled
CharSeqErr : character sequence error
add_EEP : Add EEP to Rx FIFO when error occurs
rstn_tx : synchronous Tx reset
rstn_rx : synchronous Rx reset
rstn_delay : reset the delay counter

This block contains the FSM described in the AD11. This FSM manages the link initialisation protocol.

It also includes some additional outputs.

The ST_TRAN(2:0) is used to monitor the link initialisation progression:

ST_TRAN	STATE
000	ErrorReset
001	ErrorReset
010	Ready
011	Started

100	Connecting
101	Run

Tableau 5.4.1-1 State signification

5.4.2 DELAY_CNT block

Input signals description

resetn : asynchronous reset
rstn_delay : synchronous reset
cntmax(7:0) : number of system clock periods to reach 6.4 µs

Output signals description

delay_6_4 : delay of 6.4 µs achieved
delay_12_8 : delay of 12.8 µs achieved

The DELAY_CNT block contains an 8-bit counter to compute the 6.4 µs and 12.8 µs delays used in the INIT_FSM block.

After reset (resetn or rstn_delay), the DELAY_6_4 signal goes high when the counter reaches the input CNTMAX(7:0) value once.

After reset (resetn or rstn_delay), the DELAY_12_8 signal goes high when the counter reaches the input CNTMAX(7:0) value twice.

5.4.3 RX_MGT block

Input signals description

resetn : asynchronous reset
rstn_rx : Rx asynchronous reset
RxFifo_full : Rx FIFO full
gotEOP_r : got EOP resynchronised once
gotEOP_r_r : got EOP resynchronised twice
gotEEP_r : got EEP resynchronised once
gotEEP_r_r : got EEP resynchronised twice
gotData_r : gotData1 resynchronised once
gotData_r_r : gotdata1 resynchronised twice
gotTime_r : gotTime1 resynchronised once
gotTime_r_r : gotTime1 resynchronised twice
DataBuffer(7:0) : data from RX_DECOD block
TimeBuffer(7:0) : time code from RX_DECOD block
add_EEP : Add EEP to Rx FIFO when error occurs
TimeRec_reg(7:0): time code register received

Output signals description

tickout : good time code received
EEPRec : EEP received
RxFifoData(8:0) : data to store in rx FIFO
RxFifo_wr : Rx FIFO write

The main purpose of this block is to store the RX data into the RX FIFO. The block receives a 8-bit data and stores it into the RX FIFO, adding a control flag bit. It also stores EOP/EEP when gotEOP/gotEEP is asserted. The format is described in 3.3.4.

If the block receives an EOP/EEP and another EOP/EEP later (without any data between the 2 EOP/EEP), only the first EOP/EEP will be written into the RX FIFO, the second one will not be taken into account.

The detection of time code, data, EOP or EEP is done on the rising edge of gotTime_r, gotData_r, gotEOP_r or gotEEP_r.

An interrupt (EEPRec signal) is generated when an EEP is received.

When a new time code is received, the block compares the new time code value (TimeBuffer) with the last stored time code value (TimeRec_reg). The TICKOUT signal is asserted when TimeBuffer=TimeRec_reg+1.

5.4.4 RX_FIFO block

Input signals description

rstn	: asynchronous reset
datain(8:0)	: data in
fifo_rd	: FIFO read
fifo_wr	: FIFO write

Output signals description

fifo_full	: FIFO full
fifo_empty	: FIFO empty
dataout(8:0)	: data out

The synchronous RX FIFO can contain 64 9-bit words. This FIFO stores the RX data.

5.4.5 TX_FIFO block

Input signals description

rstn	: asynchronous reset
datain(8:0)	: data in
fifo_rd	: FIFO read
fifo_wr	: FIFO write

Output signals description

fifo_full	: FIFO full
fifo_empty	: FIFO empty
dataout(8:0)	: data out

The synchronous TX FIFO can contain 8 9-bit words. This FIFO stores the TX data.

5.4.6 AHB_FIFO block

Input signals description

rstn	: asynchronous reset
datain(31:0)	: data in

fifo_rd : FIFO read
fifo_wr : FIFO write

Output signals description

fifo_full : FIFO full
fifo_empty : FIFO empty
dataout(31:0) : data out

The synchronous AHB FIFO can contain 4 32-bit words. This FIFO stores the 32-bit data from the AHB bus.

5.4.7 TX_MGT block

Input signals description

rstn_tx : synchronous reset
rstn_tx_r : resynchronised signal
rstn_tx_r_r : resynchronised signal
resetn : asynchronous reset
FCT_en : FCTSend enable
Time_Data_en : Time code and Data enabled
DataRec1_SW_r: DataSend1 acknowledge
DataRec2_SW_r: DataSend2 acknowledge
DataRec1_SW_r_r: DataSend1 acknowledge
DataRec2_SW_r_r: DataSend2 acknowledge
DataRec3_SW_r: DataSend3 acknowledge
DataRec4_SW_r: DataSend4 acknowledge
DataRec3_SW_r_r: DataSend3 acknowledge
DataRec4_SW_r_r: DataSend4 acknowledge
tickin : tick in
tickin_r : tick in
fifo_data_in(8:0): data from FIFO
FifoECnt(6:0) : Rx FIFO empty slots number
FCTSend_ack_SW_r: FCTSend Acknowledge
TxFifo_empty : Tx FIFO empty
OutstandCnt(5:0): Outstanding data counter
CreditCnt(5:0) : Credit data counter
TimeSend_ack_r: TimeSend acknowledge

Output signals description

FCTSend : FCT to send
fifo_rd : read fifo
Data1Buf(8:0) : TX data buffer 1
Data2Buf(8:0) : TX data buffer 2
Data3Buf(8:0) : TX data buffer 3
Data4Buf(8:0) : TX data buffer 4
dec_CreditCnt : decrement Credit counter
TimeSend : Time Code to send
DataSend1 : Data from buffer1 to send
DataSend2 : Data from buffer2 to send
DataSend3 : Data from buffer3 to send
DataSend4 : Data from buffer4 to send

The TX_MGT block retrieves data from the TX FIFO and generates character transmission requests to the TX_SELECT block.

There are 4 data buffers (Data1Buf, Data2Buf, Data3Buf and Data4Buf) to keep the maximum data transfer rate. The DataSend1 request corresponds to the buffer 1, the DataSend2 request corresponds to the buffer2 and so on...

When more than one DataSend is asserted, the TX_SELECT block knows which one has priority because it takes it in turns.

When the TX reset (rstn_tx signal) rising edge is detected, the TX_MGT block flushes the TX FIFO until an EOP/EEP to delete the current data packet.

The following schema describes how the FCTSend signal is generated:

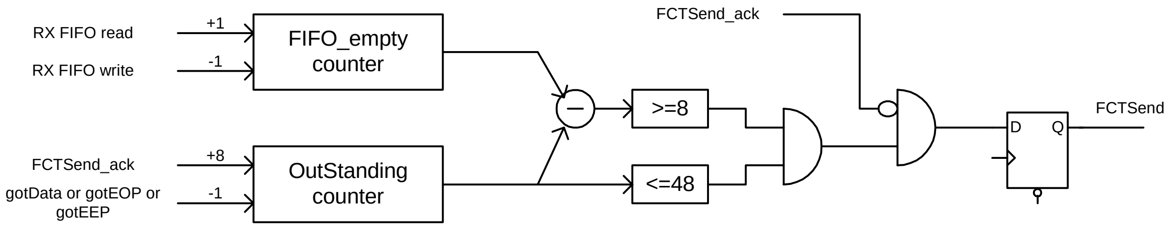


Figure 5.4.7-1 FCT send function

The TimeSend request is activated after the TICKIN rising edge detection.

5.4.8 SW_COUNTERS block

Input signals description

resetn	:	asynchronous reset
rstn_rx	:	asynchronous reset
RxFifo_empty	:	Fifo empty flag
FCTSend	:	FCT to send
FCTSend_ack_SW:		FCTSend acknowledge
RxFifo_rd	:	fifo read
RxFifo_wr	:	fifo write
gotEOP_r	:	got EOP
gotEEP_r	:	got EEP
gotData_r	:	got Data
dec_CreditCnt	:	decrement Credit counter
gotEOP_r_r	:	got EOP
gotEEP_r_r	:	got EEP
gotData_r_r	:	got Data
gotFCT_SW_r	:	gotFCT
gotFCT_ack	:	gotFCT acknowledge
DisCntLim(7:0)	:	disconnect time limit
rst_disc1	:	reset when link disconnected
rst_disc2	:	reset when link disconnected
DisCnt_en1	:	DisCnt counter enable
DisCnt_en2	:	DisCnt counter enable

Output signals description

rstn_trig : specific asynchronous reset
DisconnectErr : link disconnection detected
OutstandErr : Outstanding Error
OutstandCnt(5:0): Outstanding data counter
CreditCnt(5:0) : Credit data counter
CreditErr : Credit Error
FifoECnt(6:0) : Number of free space in the RX FIFO

This block contains 4 counters:

- The FifoECnt 7-bit counter is used to note the number of free space in the RX FIFO. This counter is incremented when a read is performed; it is decremented when a write is done. Its reset value is 64.
- The CreditCnt 6-bit counter is used to store the number of data that can be transmitted. Its reset value is 0. It is incremented by 8 when a FCT is received and is decremented when a data is transmitted.
- The OutStandCnt 6-bit counter is used to store the number of data that is expected to be received. Its reset value is 0. It is incremented by 8 when a FCT is transmitted and is decremented when a data is received.
- The DisCnt 8-bit counter is used to count the delay beyond which one the link disconnection error is activated. When the DISCONNECTION block enables this counter, it is incremented at each system clock period. Its reset value is 0. The reset is done at each edge of the RX clock.

The SW_COUNTERS block generates the Credit Error when its value is out of 56.

The OutStandErr signal is asserted when a data is received while no one is expected.

The DisConnectErr signal is asserted when the disconnection time out is reached.

5.4.9 SW_RESYNC block

Input signals description

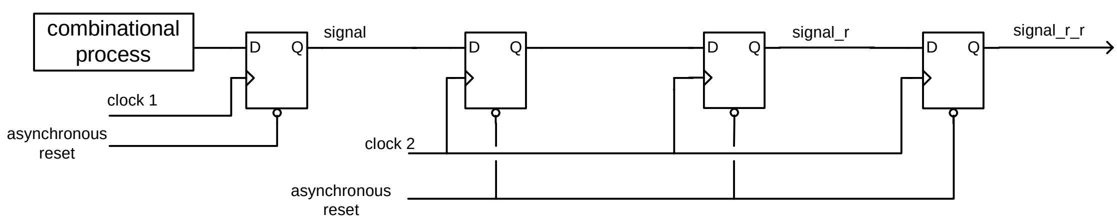
resetn : asynchronous reset
test_mode_hard : test mode asserted by hardware
CreditErr_rx : credit error from Rx FCT counter
tick_req : tick request
ESCErr : ESC error
ParityErr : Parity Error
gotData : got data in buffer1
gotTime : got Time Code in buffer1
gotEOP : got EOP
gotEEP : got EEP
gotFCT : got FCT
rstn_tx : reset Tx
FCTSend_ack : FCTSend acknowledge
TimeSend_ack : TimeSend acknowledge
DataRec1 : DataSend1 acknowledge
DataRec2 : DataSend2 acknowledge
DataRec3 : DataSend1 acknowledge

DataRec4 : DataSend2 acknowledge
gotNULL : got NULL character
rst_disc1 : reset when link disconnected
rst_disc2 : reset when link disconnected
DisCnt_en1 : DisCnt counter enable
DisCnt_en2 : DisCnt counter enable

Output signals description

rst_disc1_r : resynchronised on rising edge
rst_disc2_r : resynchronised on rising edge
DisCnt_en1_r : resynchronised on rising edge
DisCnt_en2_r : resynchronised on rising edge
tick_req_r : resynchronised on rising edge
CreditErr_rx_r : resynchronised on rising edge
ParityErr_r : resynchronised on rising edge
ESCErr_r : resynchronised on rising edge
gotData_r : resynchronised on rising edge
gotData_r_r : resynchronised on rising edge
gotEOP_r : resynchronised on rising edge
gotEOP_r_r : resynchronised on rising edge
gotEEP_r : resynchronised on rising edge
gotEEP_r_r : resynchronised on rising edge
gotTime_r : resynchronised on rising edge
gotTime_r_r : resynchronised on rising edge
gotFCT_r : resynchronised on rising edge
gotFCT_r_r : resynchronised on rising edge
FCTSend_ack_r : resynchronised on rising edge
test_mode_hard_r: test mode asserted by hardware
TimeSend_ack_r: resynchronised on rising edge
gotNULL_r : resynchronised on rising edge
rstn_tx_r : resynchronised on rising edge
rstn_tx_r_r : resynchronised on rising edge
DataRec1_r : resynchronised on rising edge
DataRec2_r : resynchronised on rising edge
DataRec1_r_r : resynchronised on rising edge
DataRec2_r_r : resynchronised on rising edge
DataRec3_r : resynchronised on rising edge
DataRec4_r : resynchronised on rising edge
DataRec3_r_r : resynchronised on rising edge
DataRec4_r_r : resynchronised on rising edge

This block resynchronised the signals from blocks working at RX or TX clock following the below architecture.



5.4.10 SW_REG block

Input signals description

resetn : asynchronous reset
rstn_tx : synchronous reset
st_trans(2:0) : state transition for test
apb_slv_in : APB input signals
rd_access_error : read access error - bad address
wrong_mode : AHB slave write access in wrong mode
end_list : end of linked list in tx AHB master mode
clear_area1_valid: clear the area 1 validity
clear_area2_valid: clear the area 2 validity
amba_error : AMBA error
no_area_valid : no valid memory area detected
cur_buf_end(31:0): current buffer end
area1_used : area1 is used to store data
area2_used : area2 is used to store data
exceed_mem : Host Memory full
clear_abort : clear the abort packet signal
desc_addr(31:0) : descriptor address
tick_req : tickin
tick_req_r : resynchronised signal
ESCErr_r : ESC Error
ParityErr_r: Parity Error
DisconnectErr : Disconnect Error
CharSeqErr : Character sequence error
OutstandErr : outstanding error
CreditErr : credit error
CreditErr_rx_r : credit error
EEPRec : EEP received
gotTime_r : got time code in buffer
gotTime_r_r : resynchronised on rising edge
TimeBuffer(7:0) : time code
gotNULL_r : got NULL
tickout : a right time code has been received
test_mode_hard: test mode asserted by hardware

Output signals description

autostart : link auto start
link_start : link start
link_disabled : link disabled
tickin : time code to send
TimeSend_reg(7:0): time code register to send
freq_init(7:0) : frequency in initialisation state
freq_run(7:0) : frequency in run state
tx_max_en : enables the TX max frequency in run state
err_int : Error Interrupt
nom_int : Nominal Interrupt
ahb_mode_tx : TX AHB master or slave
new_list : new linked list descriptor is available
TimeRec_reg(7:0): time code register received
delay_prg(7:0) : delay of 6.4 µs
DisCntLim(7:0) : Disconnect time limit
area1_valid : host memory area 1 is valid
area2_valid : host memory area 2 is valid

start_area1(31:0) : area1 start address for area1
start_area2(31:0) : area2 start address for area2
end_pac1(31:0) : packet end address for area1
end_pac2(31:0) : packet end address for area2
end_area1(31:0) : area1 end address for area1
end_area2(31:0) : area2 end address for area2
abort_packet : abort the packet transfer
apb_slv_out : APB output signals

This block contains all the registers described in the paragraph 3.4, except for the DESC_ADDR register which is implemented in the AHB_MST_SLV_TX block. The read and write accesses to these registers through the APB interface are managed in the SW_REG block.

The management of the interrupts is also done here.

5.4.11 AHB_TX_INT block

Input signals description

resetn : asynchronous reset
ahb_fifo_dataout(31:0) : ahb fifo data out
ahb_fifo_empty : ahb fifo empty flag
abort_packet : abort packet transfer
tx_fifo_full : tx fifo full flag

Output signals description

tx_fifo_datain(8:0) : tx fifo data in
ahb_fifo_rd : ahb fifo read
tx_fifo_wr : tx fifo write
clear_abort : reset the abort signal

The goal of this block is to retrieve the 32-bit data from the AHB FIFO, to split the 32-bit data into 8-bit data, and to fill the TX FIFO with 9-bit data.

For this purpose, the block contains a FSM and a 16-bit counter.

The counter is used to count the number of data in order to push an EOP into the TX FIFO at the end of the packet.

Description of the FSM

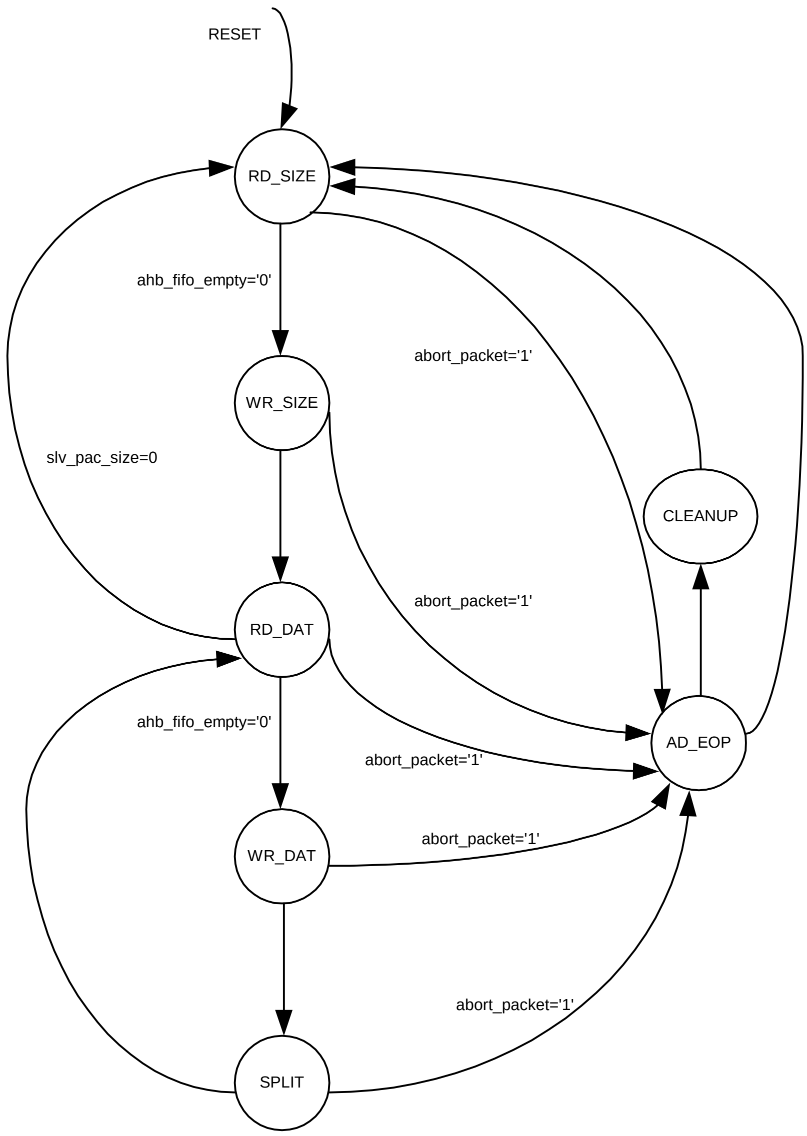


Figure 5.4.11-1 AHB_TX_INT FSM

RD_SIZE: reads the AHB FIFO

If abort_packet='1', goes to AD_EOP state.

If the AHB FIFO is not empty, activates the read then goes to WR_SIZE state.

WR_SIZE: memorizes the packet size

If abort_packet='1', goes to AD_EOP state.

Loads the slv_pac_size counter with the AHB FIFO value then goes to RD_DAT state.

RD_DAT: reads the AHB FIFO

If abort_packet='1', goes to AD_EOP state.

If the packet size is null, goes to RD_SIZE state to retrieve another packet size.

Otherwise if the AHB FIFO is not empty, activates the read, decrements the slv_pac_size and goes to WR_DAT state.

WR_DAT: memorizes the 32-bit data from the AHB FIFO

If abort_packet='1', goes to AD_EOP state.

Otherwise load the 32-bit data then goes to SPLIT state.

SPLIT: split and storage

If abort_packet='1', goes to AD_EOP state.

Otherwise, splits the 32-bit data into 8-bit data, adds the control flag '0' then writes the 9-bit data into the TX FIFO.

If end of the packet, goes to AD_EOP state. Otherwise, returns to the RD_DAT state.

AD_EOP: adds an EOP/EEP

if abort_packet=1, adds an EEP then goes to CLEANUP state. Otherwise, adds an EOP then returns to RD_SIZE state.

CLEANUP: cleanup

Flushes the AHB FIFO, clears the abort_packet signal then goes to RD_SIZE state.

5.4.12 AHB_MST_SLV_TX block

Input signals description

resetn : asynchronous reset
abort_packet : abort packet transfer
ahb_mode_tx : ahb mode: master or slave
new_list : new linked list received
ahb_fifo_full : AHB fifo full flag
ahb_slv_in : AHB slave in
ahb_mst_in : AHB master in
apb_slv_in : APB input signals

Output signals description

desc_addr(31:0) : descriptor address
ahb_mode_error: mode error
end_list : end of linked list
amba_error : AMBA error
rd_access_error : read access error - bad address
ahb_fifo_wr : AHB fifo write
ahb_fifo_datain(31:0): tx fifo data input
ahb_slv_out : AHB slave out
ahb_mst_out : AHB master out

This block includes functionalities of the TX in DMA mode (AHB master) and in slave mode (AHB slave).

The following schema shows the TX data flow:

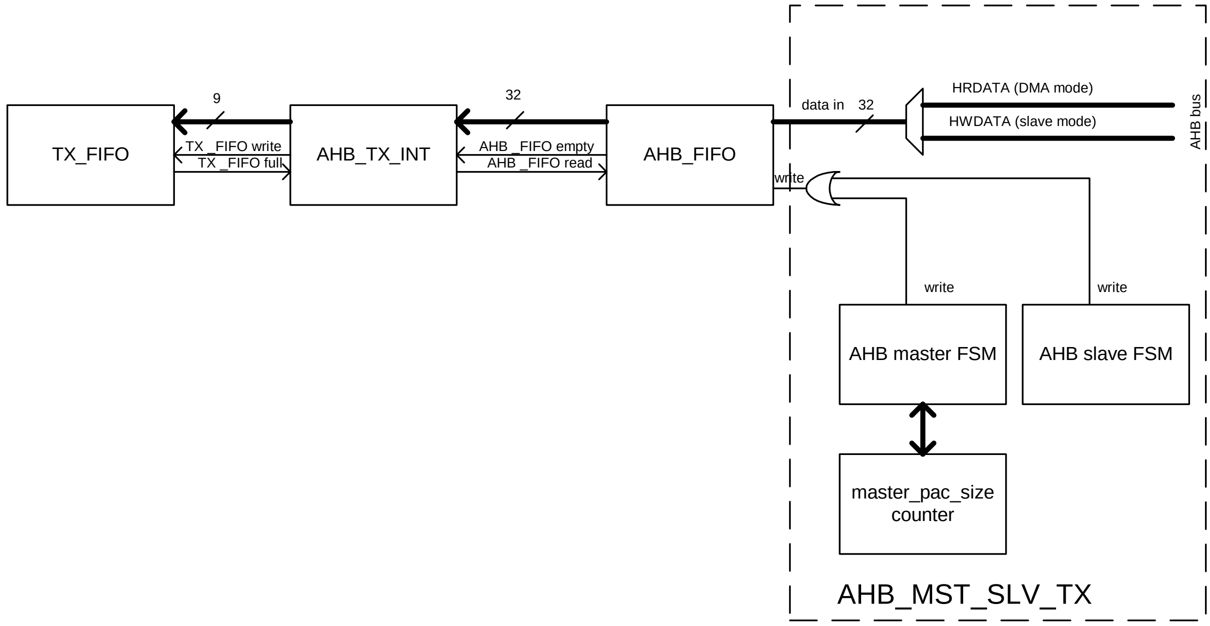


Figure 5.4.12-1 TX Host Interface

The block also contains a 16-bit counter (master_pac_size) to monitor the packet size in the DMA mode.

In TX slave mode, the block manages the split response.

Description of the TX AHB master FSM

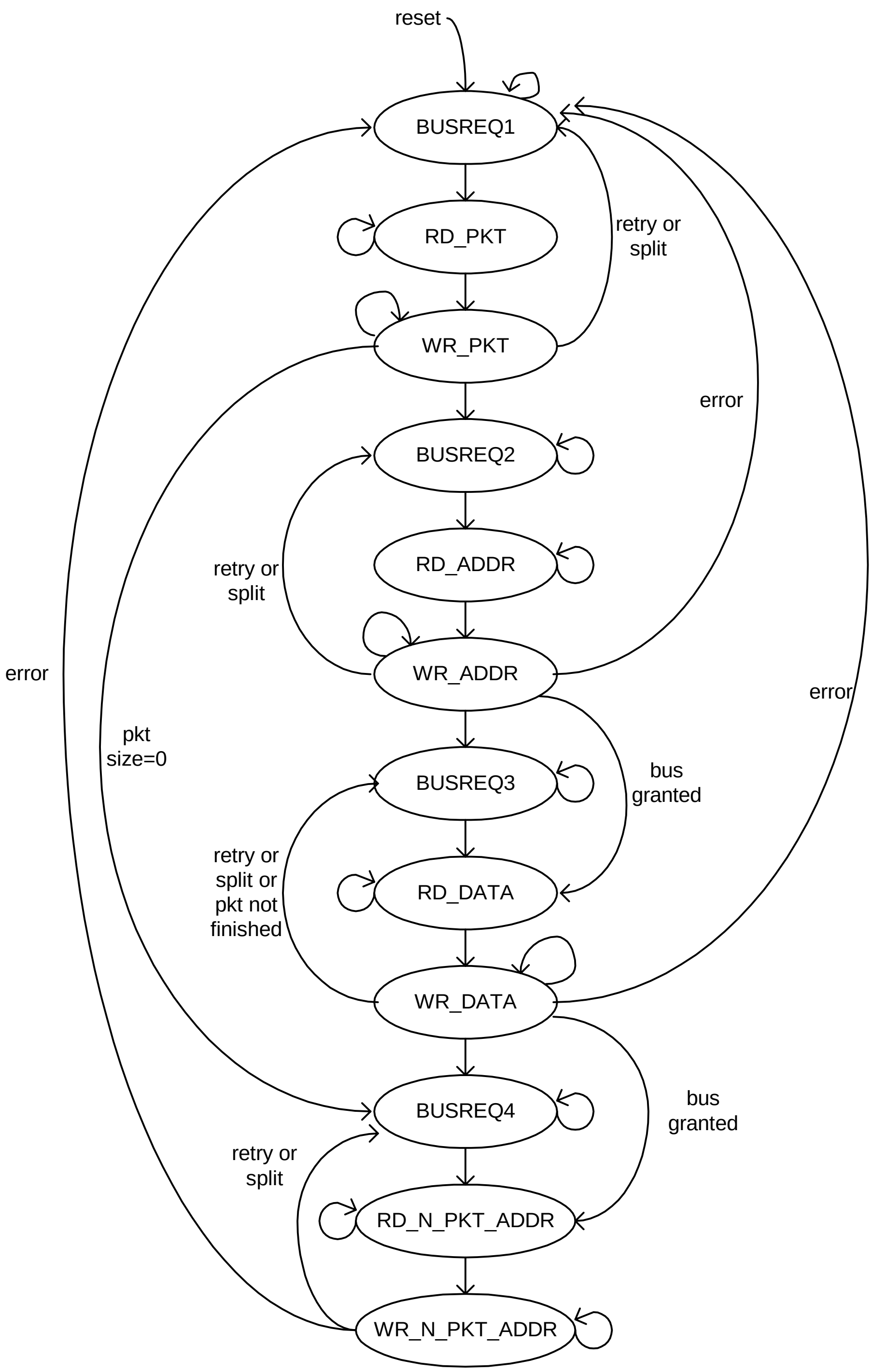


Figure 5.4.12-2 AHB master FSM

BUSREQ1: bus request

If the DMA is launched, the AHB bus is requested.

Goes to RD_PKT state when the bus is granted.

RD_PKT: reads the packet size

Performs a read access to the packet size.

If HREADY='1', goes to WR_PKT state.

WR_PKT: stores the packet size

If error response, goes to BUSREQ1 state.

If retry or split response, goes to BUSREQ1 state.

If the transfer is OKAY :

> if the packet size is null, goes to BUSREQ4 state. Otherwise, memorizes the packet size into the AHB FIFO then goes to BUSREQ2.

BUSREQ2: bus request

Bus requested. If bus granted, goes to RD_ADDR state.

RD_ADDR: reads the data address

Performs a read access to retrieve the first data address.

Goes to WR_ADDR state when HREADY='1'.

WR_ADDR: memorizes the data address

If error response, goes to BUSREQ1.

If retry or split response, goes to BUSREQ2.

If transfer is OKAY, memorizes the data address.

If the bus is always granted, goes to RD_DATA state, otherwise goes to BUSREQ3 state.

BUSREQ3: bus request

Bus requested. If bus granted, goes to RD_DATA state.

RD_DATA: reads the data value

Performs a read access to retrieve the data.

Goes to WR_DATA when HREADY='1'.

WR_DATA: stores the data into the AHB FIFO, manages the address pointers

If error response, goes to BUSREQ1.

If retry or split response, goes to BUSREQ3.

If transfer is OKAY, memorizes the data into the AHB FIFO.

If the packet is not finished, goes back to BUSREQ3 state.

Otherwise:

> if bus always granted, goes to RD_N_PKT_ADDR state. Otherwise goes to BUSREQ4 state.

BUSREQ4: bus request

Bus requested. If bus granted, goes to RD_N_PKT_ADDR state.

RD_N_PKT_ADDR: reads the next packet address

Performs a read access to retrieve the next packet address.

Goes to WR_N_PKT_ADDR state when HREADY='1'.

WR_N_PKT_ADDR: memorizes the next packet address

If error response, goes to BUSREQ1.

If retry or split response, goes to BUSREQ4.

If transfer is OKAY, memorizes the next packet address.

The DMA is stopped if the address is null.

Goes to BUSREQ1 state.

Description of the TX AHB slave FSM

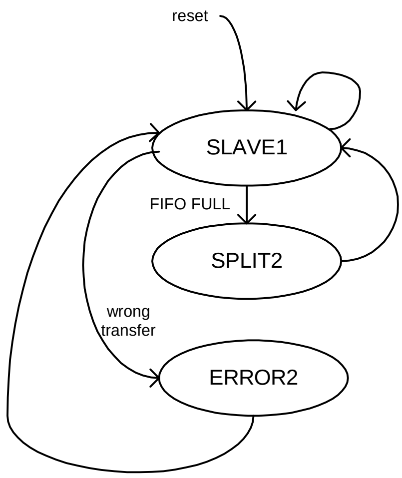


Figure 5.4.12-3 AHB slave FSM

SLAVE1: management of single and burst transfer

Generates the first cycle of split response then goes to SPLIT2 state when the AHB FIFO is full.

Generates the first ERROR cycle then goes to ERROR2 state when the transfer request is invalid (wrong mode, more than 1 master, read request,...).

Handles the split release.

Fills the AHB FIFO when the transfer is valid.

SPLIT2: split response

Generates the second cycle of split response. Then goes to SLAVE1 state.

ERROR2: error response

Generates the second cycle of error response. Then goes to SLAVE1 state.

5.4.13 AHB_MST_RX block

Input signals description

resetn	:	asynchronous reset
rx_fifo_dataout(8:0)	:	data from RX FIFO
rx_fifo_empty	:	fifo empty flag
area1_valid	:	area1 validity
area2_valid	:	area2 validity
start_area1(31:0)	:	area1 start address
start_area2(31:0)	:	area2 start address
mid_area1(31:0)	:	area1 middle address
mid_area2(31:0)	:	area2 middle address
end_area1(31:0)	:	area1 end address
end_area2(31:0)	:	area2 end address

Output signals description

exceed_mem	:	memory exceeded
rx_fifo_rd	:	fifo read
amba_error	:	AMBA error
clear_area1_valid	:	clear the validity
clear_area2_valid	:	clear the validity
no_area_valid	:	no valid area detected
cur_buf_end(31:0)	:	current buffer end
area1_used	:	area1 is used to store data
area2_used	:	area2 is used to store data
ahb_mst_in	:	AHB input signals
ahb_mst_out	:	AHB output signals

The global architecture of the AHB_MST_RX block is shown hereafter:

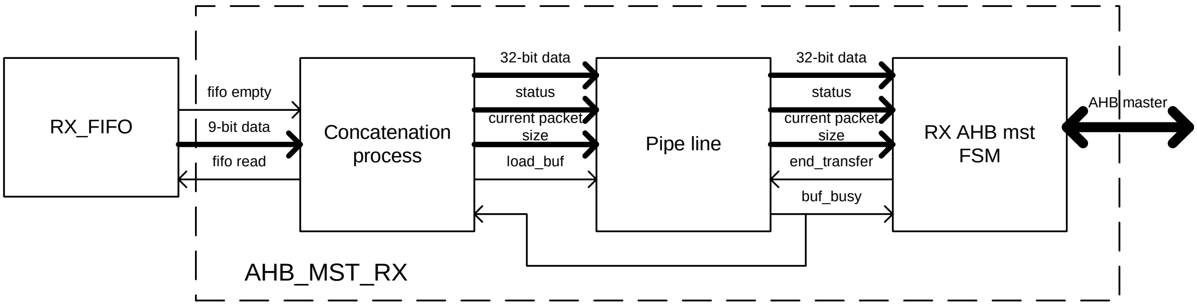


Figure 5.4.13-1 RX Host Interface

The Concatenation block contains a FSM to produce the 32-bit data word.

The Concatenation block reads the 9-bit data from the RX FIFO, then produces a 32-bit data word. This word is stored into the pipeline block when the buf_busy flag is low. This storage asserts the buf_busy flag.

When the buf_busy flag is asserted, the RX_AHB_mst FSM can read the 32-bit data. When the treatment is done, the end_transfer signal is asserted to clear the buf_busy flag. So the Concatenation block can load another 32-bit data word.

When the buf_busy flag is asserted, the Concatenation block can build the 32-bit data word as long as the RX FIFO is not empty but can't store it into the pipeline block.

Description of the Concatenation block FSM

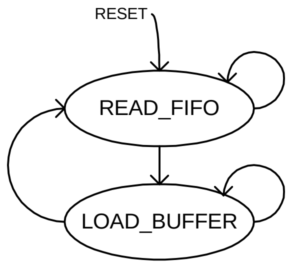


Figure 5.4.13-2 Concatenation FSM

READ_FIFO: reads the RX FIFO

When the RX FIFO is not empty, the FSM reads the FIFO to complete the 32-bit word.

If the retrieved 9-bit word from the FIFO is an EOP/EEP, the corresponding status is generated then the FSM goes to LOAD_BUFFER state.

The packet size is incremented when a data has been retrieved from the RX FIFO.

If the 32-bit word is completed, the FSM goes to LOAD_BUFFER state.

LOAD_BUFFER: loads the pipeline block

In this state, the FSM loads the pipeline with the 32-bit word, the current status and the current packet size when the buf_busy signal is low. Then it goes to READ_FIFO state.

Description of the RX AHB master FSM

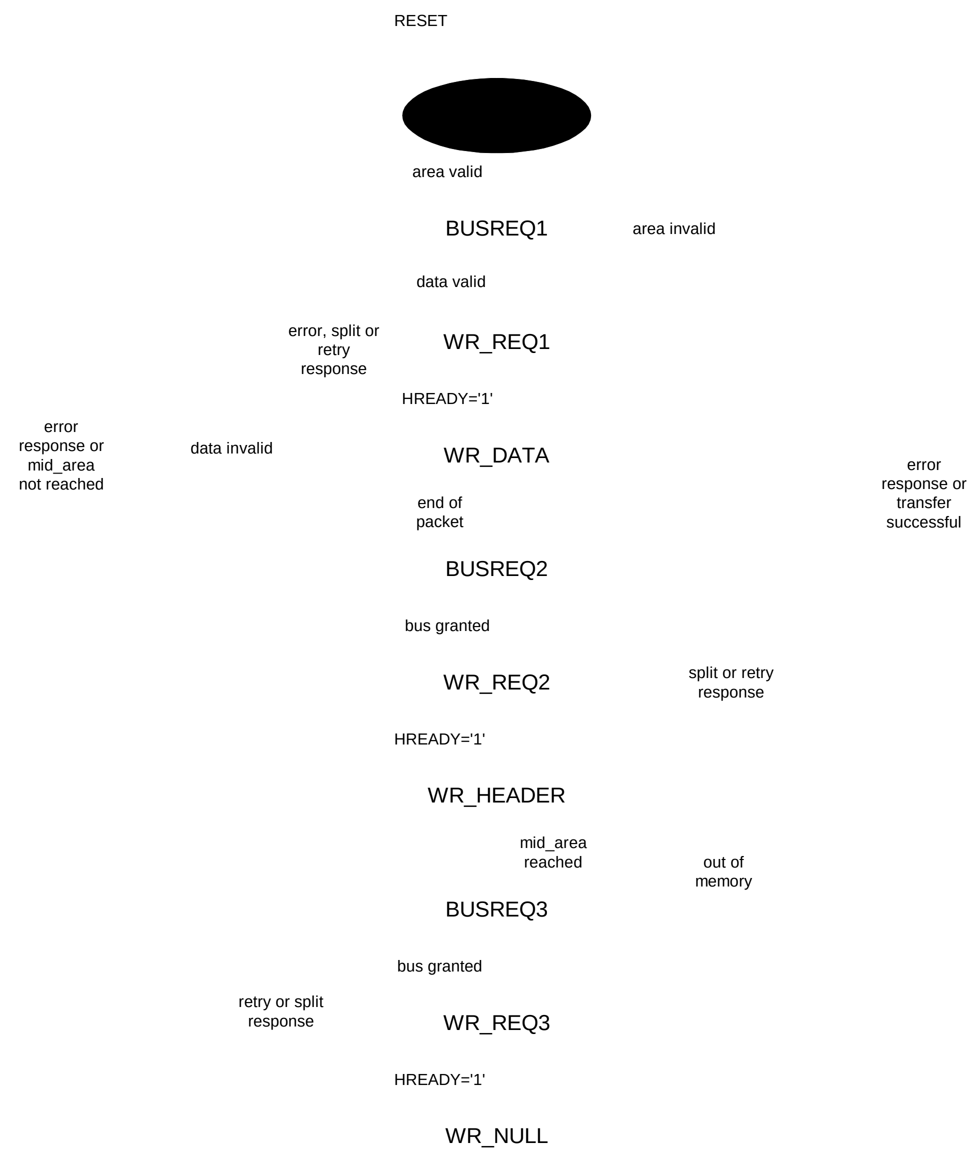


Figure 5.4.13-3 RX AHB master FSM

<u>INIT_MEM: waits for a valid memory area to load</u>

When area1 or area2 is valid, the FSM initializes the area then goes to the BUSREQ1 state.

Otherwise, the no_area_valid signal is asserted and the FSM stays in this state.

<u>BUSREQ1: bus request</u>

If the area is invalid, the FSM goes to INIT_MEM state.

If the transfer is valid (data valid, memory area valid), the AHB bus is requested.

To write a data, the FSM goes to WR_REQ1 state. To write a header, the FSM goes to WR_REQ2 state.

<u>WR_REQ1: write request</u>

Single transfer requested. Goes to WR_DATA state when HREADY='1'.

WR_DATA: outputs the data value

Waits for the transfer end.

If error, retry or split response, the FSM goes to BUSREQ1 state.

If the transfer is successful:

> If end of packet, the FSM goes to BUSREQ2 state.
>
> If end of memory area, the EXCEED_MEM interrupt is activated then the FSM goes to BUSREQ2 state.
>
> Otherwise, the FSM goes to BUSREQ1 state.

BUSREQ2: bus request

Goes to WR_REQ2 when the bus is granted.

WR_REQ2: write request

Single transfer requested to write the header.

Goes to WR_HEADER state when HREADY='1'.

WR_HEADER: outputs the header value

Waits the transfer end.

If error response, the FSM goes to BUSREQ1 state.

If retry or split response, the FSM goes to BUSREQ2 state.

If the transfer is okay, the data address is incremented and the current buffer end address is updated.

If the current data address is higher than the area packet end address (area_mid_addr), the FSM goes to BUSREQ3 state. Otherwise, the FSM goes to BUSREQ1 state.

BUSREQ3: bus request

If no space left, the FSM goes to INIT_MEM state.

Otherwise, the bus is requested to write the null header.

Goes to WR_REQ3 when the bus is granted.

WR_REQ3: write request

Single transfer requested to write a null header.

Goes to WR_NULL state when HREADY='1'.

WR_NULL: outputs the null header on the data bus

Waits for the transfer end.

If error response, the FSM generates the amba_error interrupt, clears the area validity and goes to INIT_MEM state.

If retry or split response, the FSM goes to BUSREQ3 state.

If the transfer is okay, The FSM clears the area validity then goes to the INIT_MEM state.

5.5 BLOCK WORKING AT INPUT TX CLOCK

5.5.1 CLK_TX_GEN block

Input signals description

resetn	:	asynchronous reset
sel	:	frequency selection between freq_init and freq_run
freq_init(7:0)	:	frequency at initialization state
freq_run(7:0)	:	frequency at run state
tx_max_en	:	TX max frequency enable

Output signal description

clk_txout	:	TX clock used for the transmission

The architecture of the block is shown hereafter:

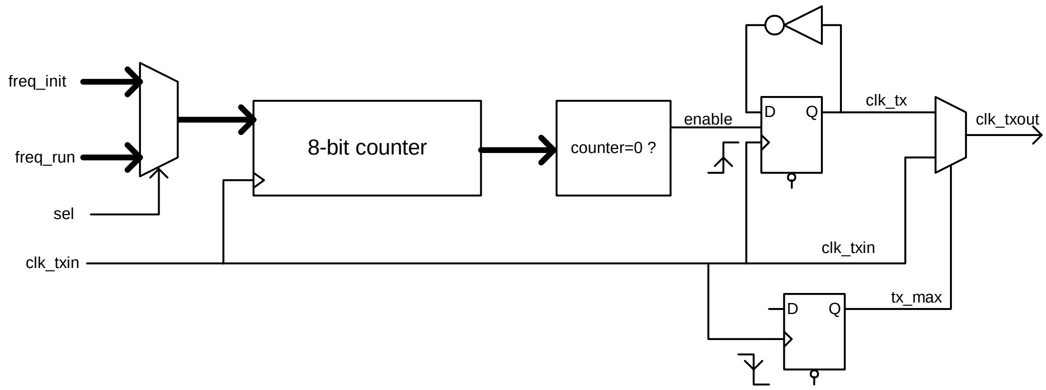


Figure 5.5.1-1 TX clock generation

The internal clk_tx signal changes its value each time the counter value is 0.

The clock selection between clk_tx and clk_txin is done by the tx_max signal.

To avoid any glitch on the clk_txout signal, the tx_max signal is updated on clock falling edge while the clk_tx signal is updated on clock rising edge.

---%---%---%---%---