35.3 Functional Description

This section describes the HSSL protocol and it's implementation.

35.3.1 HSSL Protocol Definition

This section describes the high level protocol definition of the HSSL interface.

The HSSL communication normally consists of two stage transactions, sending a command and an receiving a response. It involves two participants:

- an initiator, which starts a transaction by sending a command
- a target, which responds to the command

35.3.1.1 List of Abbreviations, Acronyms, and Term Definitions

ACK	- Acknowledge Frame
NACK	- Not Acknowledge Frame (Target Error Frame)
CRC	- Cyclic Redundancy Check
HSSL	- High Speed Serial Link
TO	- Time Out
TT	- Transaction Tag
TTERR	- Transaction Tag Error
SPB	- Serial Peripheral Bus
SRI	- Shared Resource Interconnection
SoC	- System on Chip
Initiator	- Device that starts a HSSL transaction by sending a command frame
Target	- Device that responds to a command frame sent by an initiator

35.3.1.2 Frame Types

The HSSL module generates a serial frame by:

- taking the payload delivered by a CPU or a DMA
- first wrapping it in a HSSL frame by adding header and CRC fields, and then
- wrapping the HSSL frame in a HSCT frame by adding Sync, Header and End bit fields (see **Figure 357**)

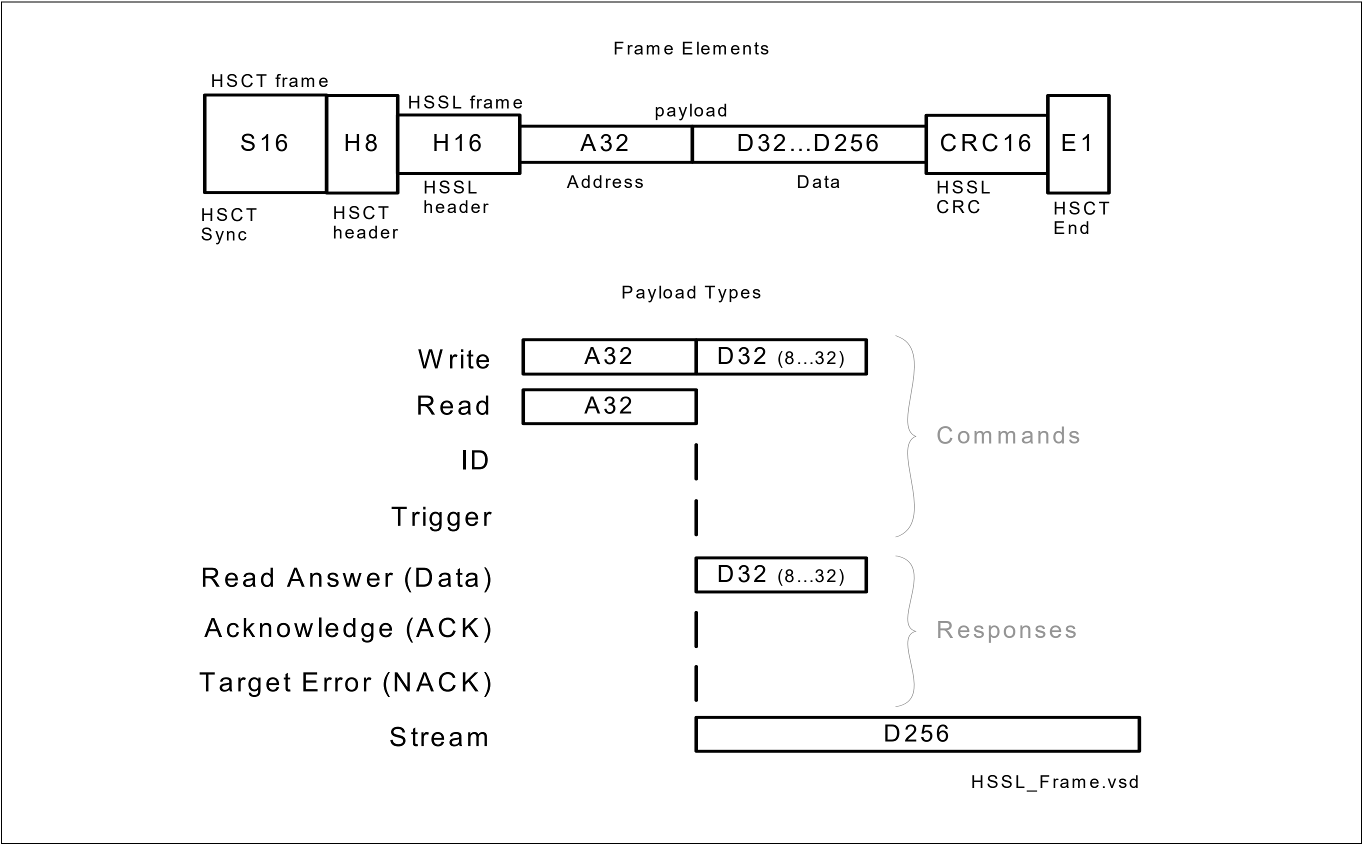


Figure 357 HSSL Frame Types

There are several types of frames:

Command Frames

Command frames are sent by the initiator to request an action by the target. There are four types of command frames:

- **Write Frame**
  - Used by the initiator to request the target controller to write the data to the address, both provided by the initiator.
- **Read Frame**
  - Used by the initiator to request the target controller to read and deliver a content from an address from its address space.
- **Read ID Frame**
  - Used by the initiator to request the target controller to read and deliver a content uniquely defining the target device. The answer is a standard Read Data Answer Frame.
- **Trigger Frame**
  - Used by the initiator to trigger the trigger interrupt in the target module.

In this document, the command frames are sometimes referred to as WRT frames.

Response Frames

Response frames are sent by the target either to deliver the requested data or to report a successful or unsuccessful completion of a request. There are three types of response frames:

- **Read Data Answer Frame**

– Used by the target to deliver the data, requested by the initiator with a read command frame. It is frequently referred to as a Data frame.

- **Acknowledge Frame**
  – Used by the target to confirm the completion of a write or trigger command. It is frequently referred to as an ACK frame.
- **Target Error Frame**
  – Used by the target to report the unsuccessful attempt to complete a write or read command. It is frequently referred to as a NACK frame.

Stream Frame

Used by the initiator controller for sending data streams. It behaves similar to a write command frame which contains long 256-bit data block, is acknowledged in the same way, but the acknowledge uses sliding window with depth of 2 (instead of 1 as for the simple command frames).

35.3.1.2.1 Frame and Payload Lengths

Each type of frame has a single data length associated with it. The length of the frame depends on the length of the payload. Different frame types have different payload lengths, but the payload length for one type is constant for all variations of the frame type.

For example all data frames have the same length, although they can carry 8, 16 or 32 bit data. This is due to the fact that the data field in the frame is always 32 bit long, and if the data itself is shorter, it is copied several times to fill in the whole field. The same behavior is valid for the read answer frames.

The rest of the frame, consisting of sync and header fields, has always the same length (see **Figure 361**).

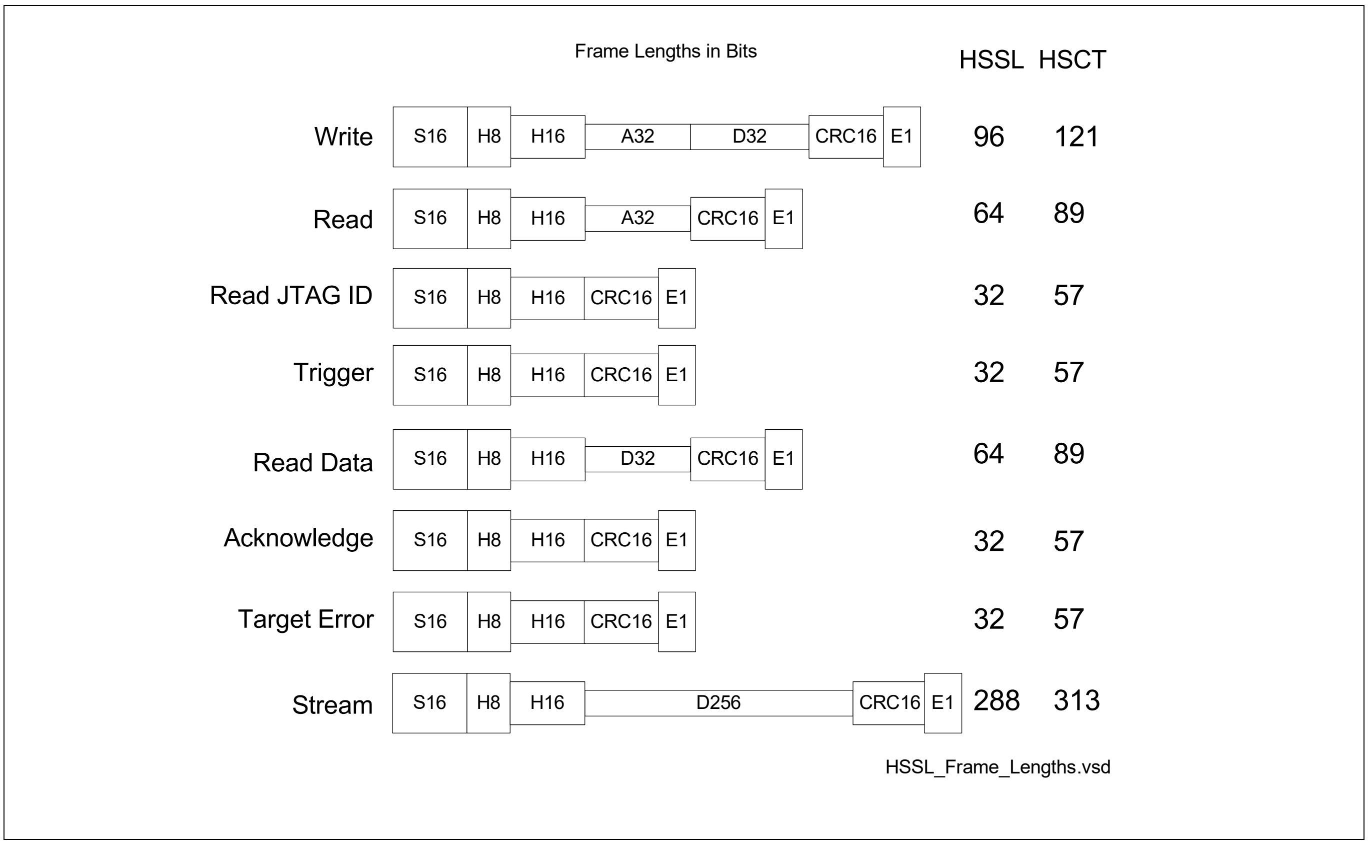


Figure 358 Frame Lengths

35.3.1.2.2 Data Types

There are four types of HSSL data:

- 8-bit
- 16-bit
- 32-bit
- 256-bit stream block

The in case of 8/16/32 bit data types, the transmitted number of bits is always 32, where the shorter data types are copied several times to fill up this length (see **Figure 359**).

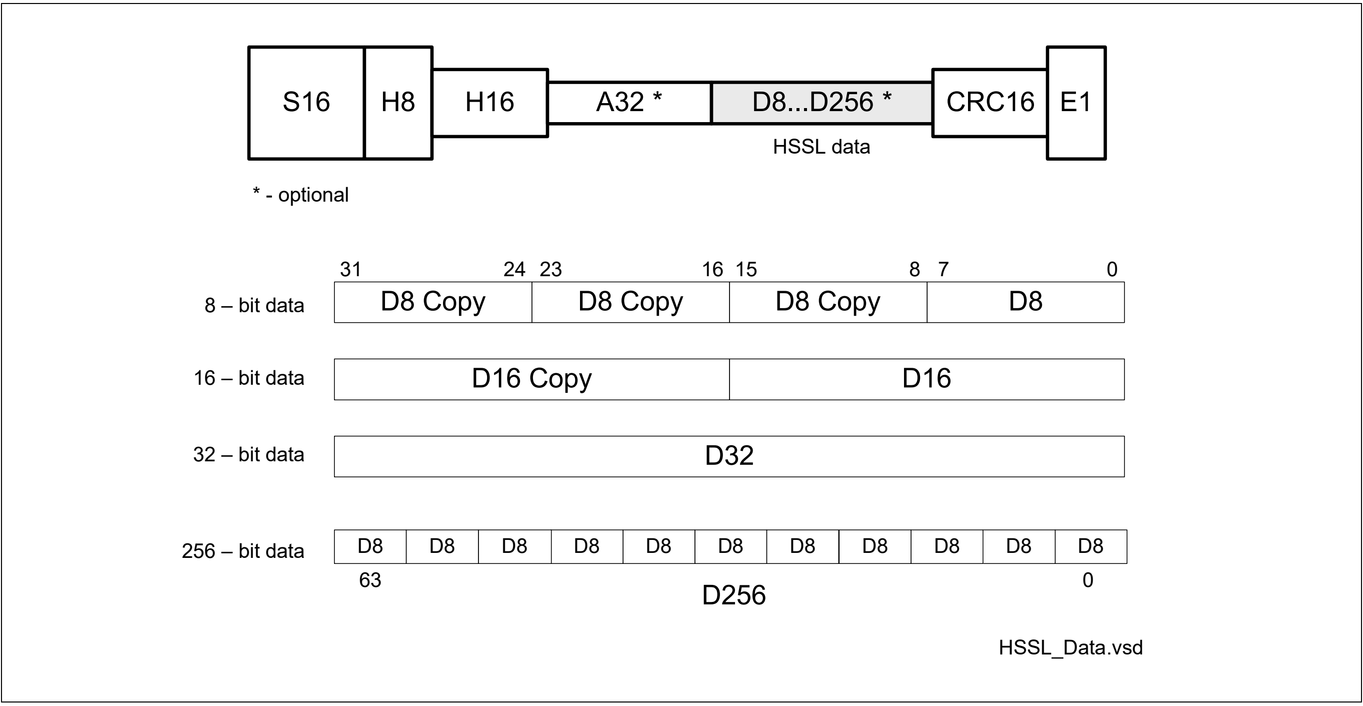


Figure 359 Data Types

35.3.1.2.3 Cyclic Redundancy Check Field - CRC

The HSSL protocol uses the CRC-16-CCITT polynomial: $x^{16} + x^{12} + x^5 + 1$ with the following standard properties:

- seed: 0xFFFF
- calculation direction: MSB first (header msb to lsb, then payload msb to lsb)
- CRC result direction: MSB first

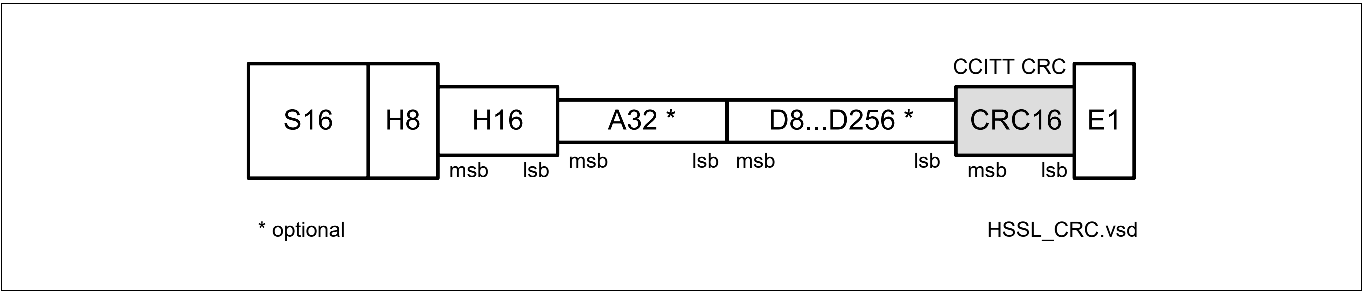


Figure 360 CRC Field

35.3.1.2.4 Header Structure

The HSSL header contains the protocol information that is required additionally to the raw payload data. The HSSL header describes the payload data regarding the length and the type, and carries additional information like channel number code and transaction tag. It also carries stand-alone information in case of frames without payload, like acknowledge and trigger frames.

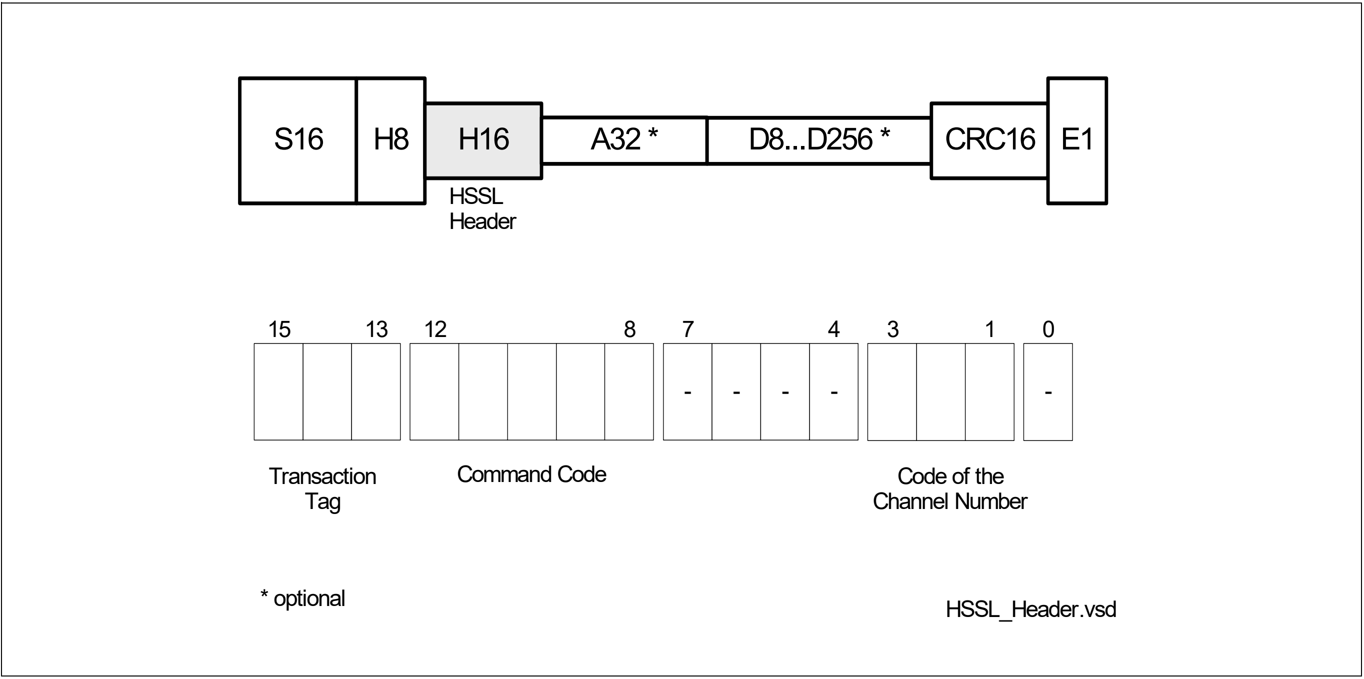


Figure 361 Header Types

Note: The unused bits in the header are padded with 0.

Table 306 Mapping of HSSL to HSCT channel codes

HSSL Channel Number	HSSL Channel Number, Binary Code	HSSL Channel Number, Special Code	HSCT Channel	HSCT Channel Code
0	000	100	A	0100
1	001	101	B	0101
2	010	110	C	0110
3	011	111	D	0111
4 (reserved)	100	000	E	1000
5 (reserved)	101	001	F	1001
6 (reserved)	110	010	G	1010
7 (reserved)	111	011	H	1011

Note: The HSSL module supports four channels, 0 to 3.

Table 307 List of Command Codes

Command Code	Command Description	HSSL Frame Size
00000	Read 8 bit	64
00001	Read 16 bit	64
00010	Read 32 bit	64
00011	RFU (Reserved for Future Use)	-
00100	Write 8 bit with ACK (Acknowledge)	96
00101	Write 16 bit with ACK	96
00110	Write 32 bit with ACK	96
00111	RFU	-
01000	ACK OK	32
01001	ACK Target Error	32
01010	Read Answer OK	64
01011	RFU	-
01100	Trigger with ACK	32
01101	RFU	-
01110	RFU	-
01111	RFU	-
10000	RFU	-
10001	RFU	-
10010	Read JTAG ID 32-bit	32
10011	RFU	-
10100	RFU	-
10101	RFU	-
10110	RFU	-
10111	Stream Data with ACK (32 Byte Data)	288

Table 307 List of Command Codes (cont'd)

Command Code	Command Description	HSSL Frame Size
11000	RFU	-
11001	RFU	-
11010	RFU	-
11011	RFU	-
11100	RFU	-
11101	RFU	-
11110	RFU	-
11111	RFU	-

35.3.1.3 Single and Block Transfers

Single frame transfers are always performed expecting an acknowledge.

35.3.1.4 Streaming Interface

Stream frame transfers are always performed with acknowledge.

There are only write streams possible. Read streams are not possible.

35.3.1.5 Sliding Window Protocol

For flow control the HSSL module implements two variants of the "Sliding Window" protocol regarding the sliding window depth:

- depth of one used for command frames
  - The depth of one means that a single timer tracks the arrival of the corresponding response frame. A new command can be sent only after the response to the previous command has been received, or a timeout occurred
- depth of two used for streaming frames
  - the depth of two means that there are two timers tracking the acknowledge times for the two latest streaming frames. In this case, a second stream frame can be sent although an acknowledge / timeout to the previous stream frame has not been received yet. After the second stream frame, no third stream frame can be sent before the first one has been acknowledged

An example of communication using sliding window with depth of one protocol is shown in **Figure 362**. Here, the initiator sends commands originating from one and the same channel, for example channel 0 (labeled with the capital letter "A"), and the corresponding acknowledge frames from the target are labeled with the small letter "a".

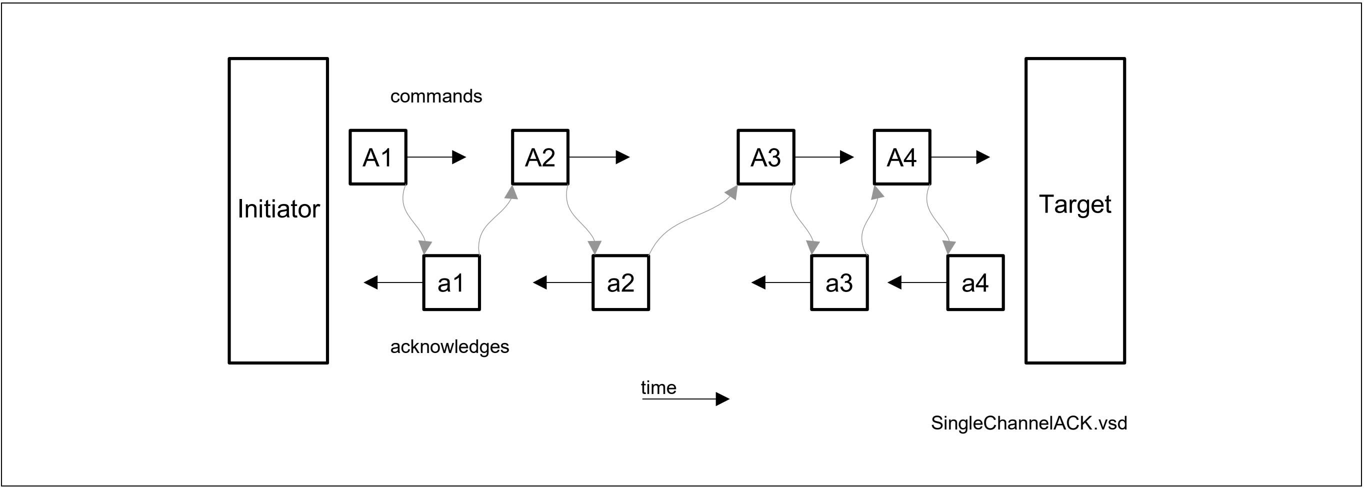


Figure 362 Flow Control, Single Channel, Sliding Window With Depth of One

The sliding window protocol requires capability of sending acknowledge frames in the target and timeout timers in the initiator. Every time the target receives a frame without detecting an error, it sends a response / acknowledge to the initiator. If the target detects a frame with a CRC error, it does not send an acknowledge. The missing acknowledge causes a timeout event in the initiator channel.

If the command frames are longer than the acknowledge frames plus the reaction time, like for example in case of streaming, back to back transfers are easily possible, see **Figure 363**.

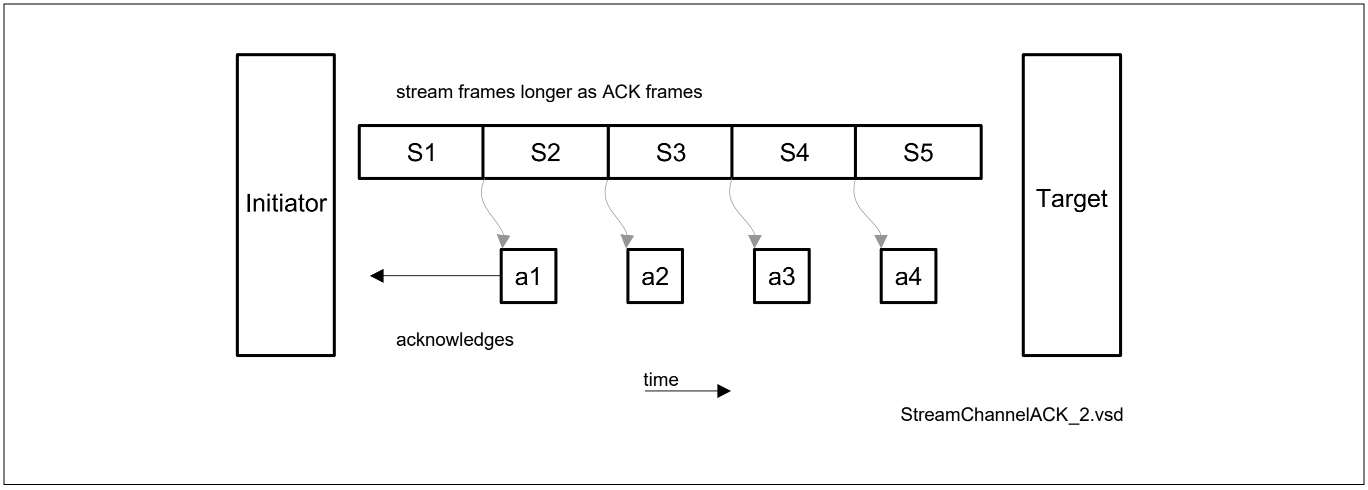


Figure 363 Flow Control, Streaming Frames

35.3.1.6 Error Management

The HSSL protocol defines four types of errors:

- Time Out
- Transaction Tag Error
- CRC Error
- Target Error

Time Out Error

A time out error is detected at the initiator side, if the expected ACK frame was not received within the expected time window. This can occur if a frame had been sent by the initiator, and the target detected an CRC error and did not answer with an acknowledge, or the connection between the initiator and the target is physically damaged in one or the other direction.

Transaction Tag Error

A transaction tag error occurs at the initiator side, if instead the expected ACK frame with the expected TAG number, an acknowledge with an unexpected transaction tag was received. This would indicate a missing frame or missing acknowledge. Transaction tag error generate frames that pass the CRC checking stage.

CRC Error

A CRC Error can occur:

- at the target side, in which case:
  - the CRC error flag is set
  - the received command frame with a CRC error is discarded
  - no acknowledge frame is sent and
  - a channel unspecific EXI error interrupt is generated, if enabled.
- at the initiator side, in which case:
  - the CRC error flag is set
  - the received response frame with a CRC error is discarded
  - a channel unspecific EXI error interrupt is generated, if enabled

Both scenarios lead to a time out at the initiator side.

In both cases the CRC error flag is set at the side receiving the erroneous frame (either initiator or target) and an interrupt is generated, if enabled.

For the purpose of CRC error injection, a bit field with an XOR mask used for toggling the bits of the calculated CRC is provided, see **CRC**.XORMASK. In order to switch on this feature, the E (Endinit) protected bit must be set by the application software. The error injection only influences the generation of the transmitted CRC.

Target Error

A Target Error can occur at the target side, when accessing the target memory a bus error or memory protection error occurs. In such a case, the target responds with an NACK frame indicating the error.

35.3.1.7 Shift Direction

In general, the HSSL interface makes a copy of a memory location or block from the address space of one microcontroller into the address space of the other microcontroller.

The HSSL module delivers 32-bit parallel data.

The transfer over HSCT serial link is done by using MSB first shifting at 32-bit level.

The overview of the path from the memory to the serial bus is shown in the **Figure 364**.

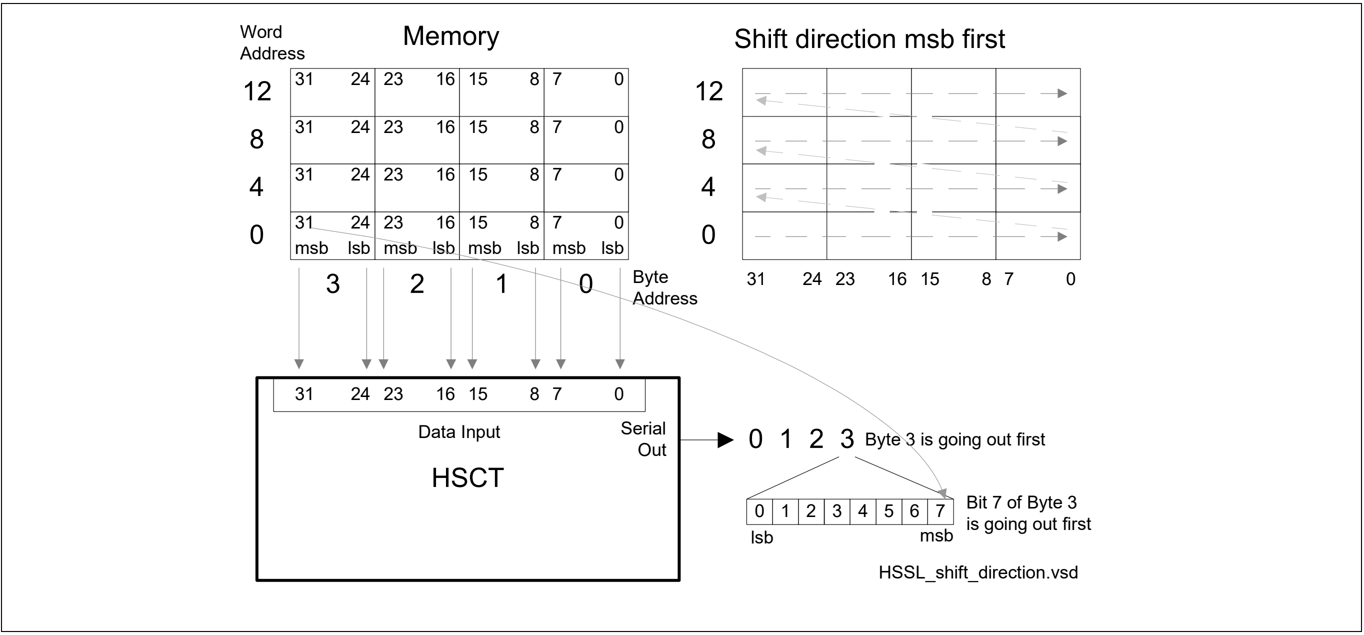


Figure 364 Shift Direction in Case of HSCT Layer

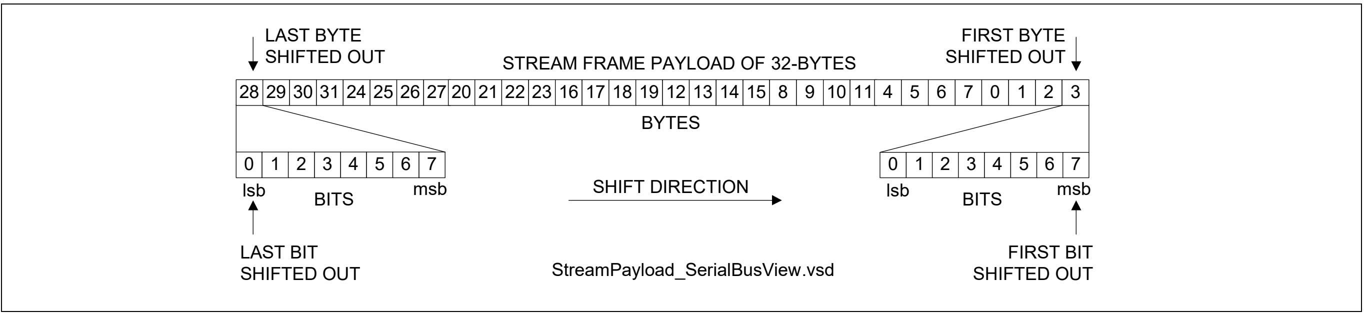


Figure 365 Stream Frame in Case of HSCT Layer

35.3.2 HSSL Implementation

This section describes the implementation of the HSSL protocol.

35.3.3 Overview

The HSSL module is connected to the SRI bus using a master interface and to the SPB bus using a BPI slave interface.

Additionally, it is connected to the SPB bus using a master interface.

The SRI master interface is capable of performing single and burst reads and writes on the SRI bus. Additionally, the HSSL kernel provides logic for performing streaming communication. The write and read accesses are always performed in User Mode.

The SPB BPI slave interface is used by an SPB master (DMA, CPU, HSSL SPB Master) for writing the HSSL registers, thus configuring the module and performing single read and write operations.

The SPB master interface is capable of performing single read operations, write operations, and streaming communication using BTR4 burst accesses on the SPB bus. The SPB master accesses are always performed in User Mode.

The HSSL module to use the SRI or the SPB master depending on the address of the access. Accesses to the SPB address space are automatically routed to the SPB master, accesses to the SRI address space are automatically routed to the SRI master. The address window F000 0000_H - F7FF $FFFF_H$ is used for the SPB bus, otherwise SRI.

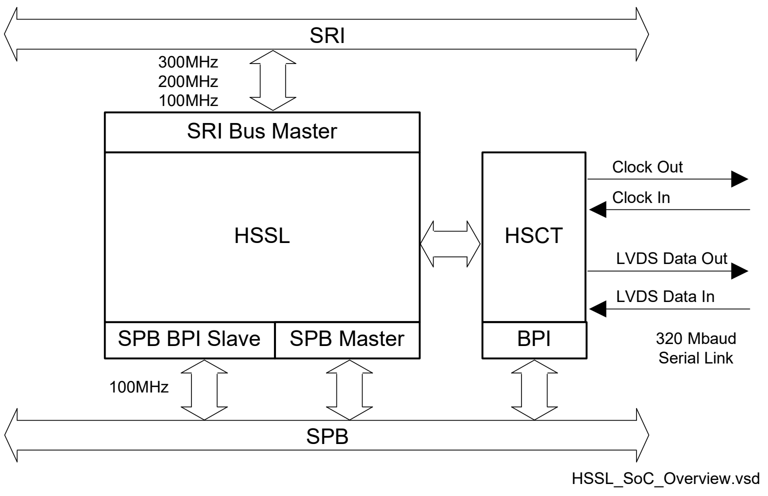


Figure 366 HSSL Subsystem Overview

35.3.3.1 HSSL Module Operation

The HSSL module consists of separate transmitter and receiver, each containing four channels.

The transmitter contains an arbiter block and a wrapper block, which are common for all channels. The arbiter provides fixed arbitration between the channels which are ready to send when the serial link is available, with channel 0 having the highest priority, and the channel 3 having the lowest priority. The wrapper block generates the CRC.

The receiver unwraps the HSSL frame, discards the frames with a CRC error, and distributes the error free commands and acknowledges to the channels.

A common prescaler generates the time basis for all channels, in particular for their timeout timers.

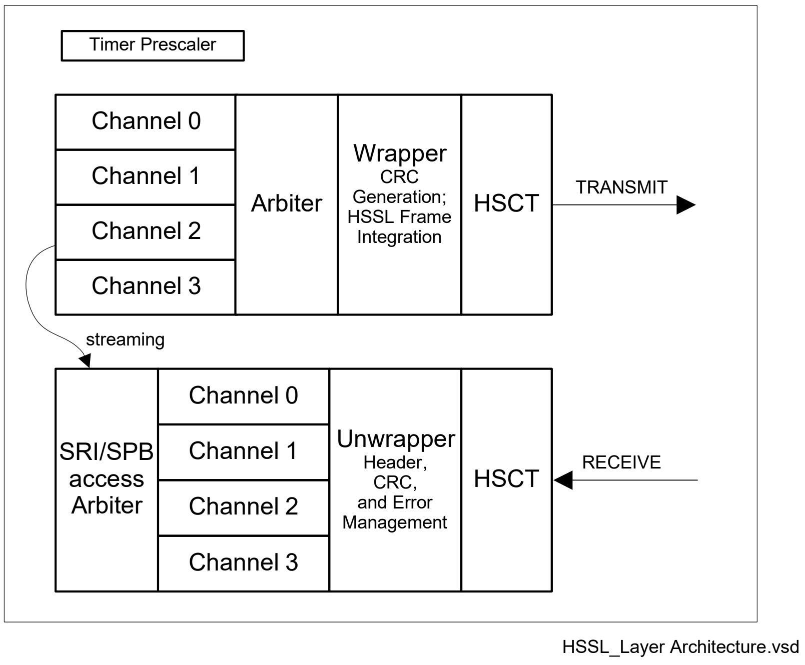


Figure 367 HSSL Layer Architecture

35.3.3.1.1 Frame Transmission Priorisation

When more than one frames are pending, an arbitration is performed between them.

Each channel activates two flags: initiator command flag and response flag: channel 0 I0 and R0, channel 1 I1 and R1, … .

Between the response and command frame request types, the response request type has higher priority. Among the requests of a same type, lower channel number has higher priority, see **Figure 368**. A pending frame is indicated with a corresponding request flag in the register **QFLAGS**.

The SRI/SPB bus master sets the response requests (“R” flags).

The software write accesses via the SPB bus set the initiator requests (“I” flags).

The transmit arbiter clears the “I” and “R” flags of the arbitration winner.

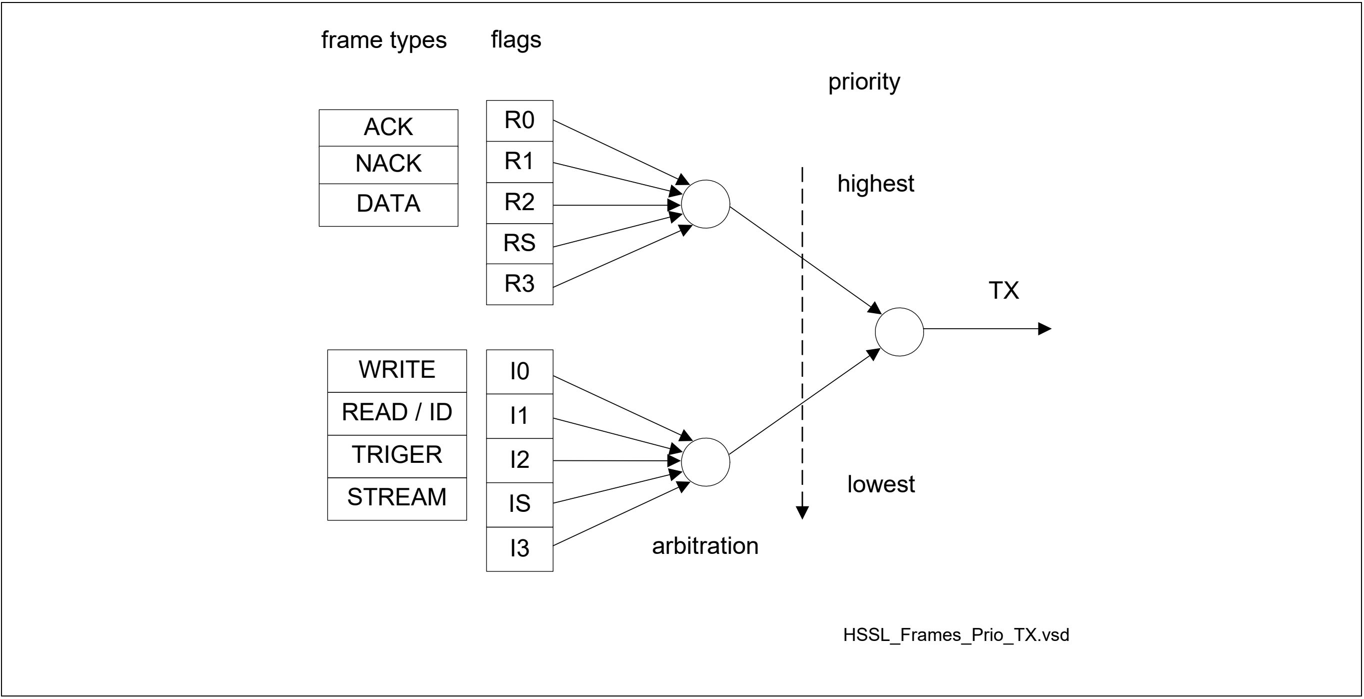


Figure 368 Frame Priorities in the HSSL module

35.3.3.1.2 Received Frame Management and Command Execution

After a frame passes the CRC check, it is checked for its type. A received command frame sets a request flag for the SRI/SPB master, a received response frame resets the expect flag and stops the appropriate channel timeout, see **Figure 369**.

The target flags "T" are set by the frame distributor, and cleared by the arbiter of the SRI/SPB master.

The expect flags "E" are set by the Tx arbiter, and reset by the Rx distributor.

The ISB and ISF flags operate together to control the streaming process. The ISB flag (Initiator Stream Block flag) is set by the software to enable the streaming. The ISF flag (Initiator stream FIFO flag) is set/cleared by the TXFIFO according to its filling level.

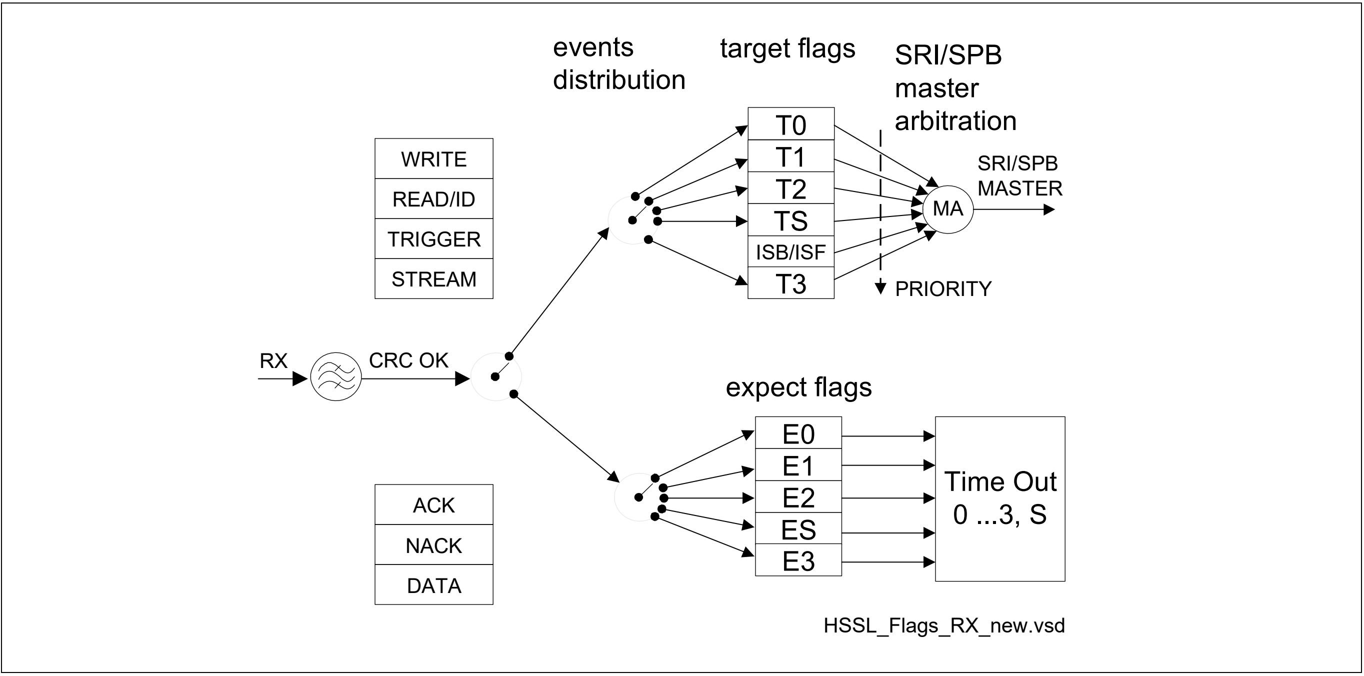


Figure 369 Received Frames Management

The CRC and Transaction Tag error flags are located in the **MFLAGS** register.

An overview of a full communication cycle over all sections of a channel (initiator, target or receive, transmit) is shown in **Figure 370**.

See also the **QFLAGS** register description.

If the channel does not have a command pending and does not expect a response, it is ready.

Note: In case of at trigger command frame, the target command distributor sets the corresponding Rx flag, and not the corresponding Tx flag, due to the fact that the service request is triggered directly, and not by the SRI/SPB master,

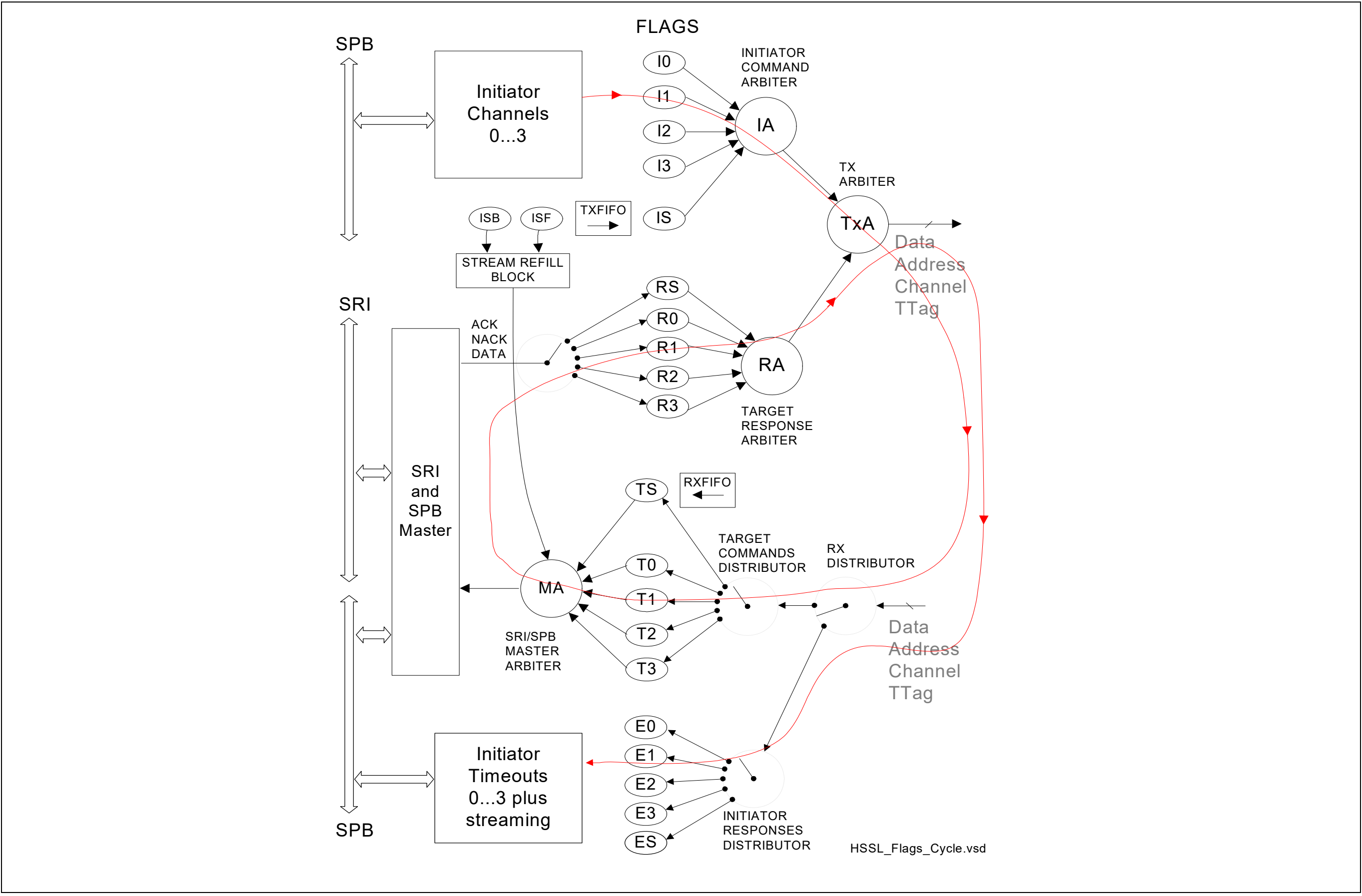


Figure 370 Request Flags Overview and a Communication Cycle Example

Table 308 Set and Clear of Request Flags

	I	**R**	**T**	**E**	**ISB**	**ISF**
SPB Software Write	set	-	-	-	set	-
Tx Arbiter	clear	clear	-	set	-	-
Rx Distributor	-	-[1)]	set	clear	-	-
SRI / SPB Master	-	set	clear	-	-	-
Stream Refill Block	-	-	-	-	clear	clear
TXFIFO	-	-	-	-	-	set

1) In case of a trigger command, an R flag is set by the Rx Distributor instead of a T Flag, because a trigger command is not executed by the SRI/SPB master.

35.3.3.2 HSSL Channel Architecture

The HSSL module contains four channels. The functionality of each channel can be subdivided in two ways:

- transmitter and receiver functionality
  - The transmit functionality generates an appropriate header for each frame
  - The receiver generates an appropriate acknowledge header with the tag and channel number code. The read and writes are performed by the SRI/SPB master. Therefore, the module can send back to the transmitter a target error frame if an SRI/SPB bus error occurs and the transfer on the bus can not be executed.
  - The timeout monitoring on one hand and the acknowledge management on the other hand involve both the transmitter and receiver channels
- initiator and target functionality
  - The initiator functionality transmits write, read or trigger command frames in random sequence, after the previous command has been responded to.
  - Generation/comparison of a Transaction Tag per channel is an initiator functionality
  - The timeout management is a pure initiator functionality.
  - The target functionality is to receive commands, executes them, and generates and transmits responses.

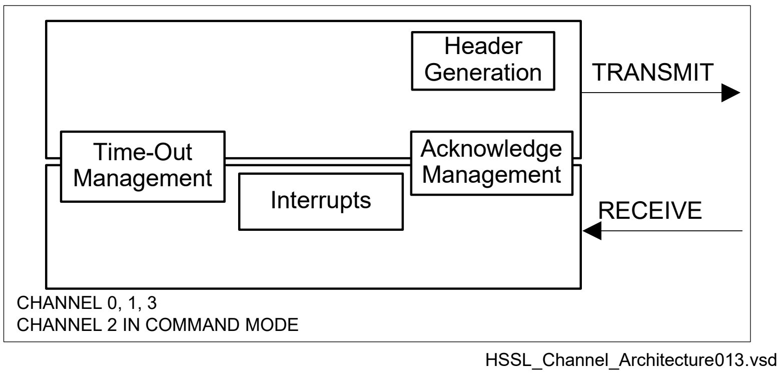


Figure 371 Architecture of Channels 0,1, 3 and Channel 2 in Command Mode

Channel 2 can be switched between command and streaming mode of operation. In single mode a single data register is used and in streaming mode a FIFO is used.

Channel 2 contains one initiator and one target address counter which are used to fill a memory range with data. Each counter contains two start address registers and one stream frame count register.
At reaching the predefined frame count, a wrap around is automatically executed.

The granularity of the start address is 256-bit.

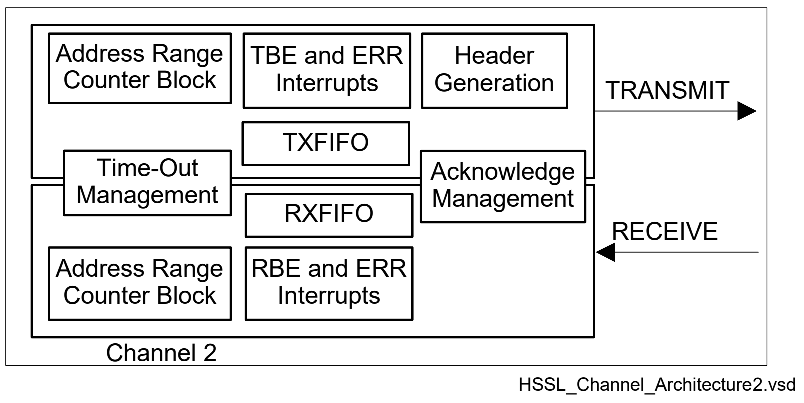
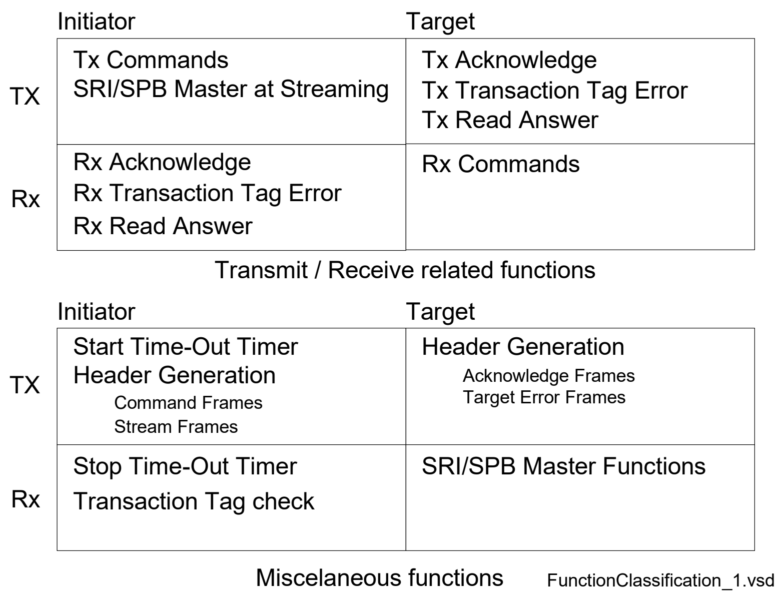


Figure 372 HSSL Architecture of Channel 2

A classification of the various channel functions is shown in **Figure 373**.



	Initiator	Target
TX	Tx Commands SRI/SPB Master at Streaming	Tx Acknowledge Tx Transaction Tag Error Tx Read Answer
Rx	Rx Acknowledge Rx Transaction Tag Error Rx Read Answer	Rx Commands

Transmit / Receive related functions

	Initiator	Target
TX	Start Time-Out Timer Header Generation Command Frames Stream Frames	Header Generation Acknowledge Frames Target Error Frames
Rx	Stop Time-Out Timer Transaction Tag check	SRI/SPB Master Functions

Miscelaneous functions FunctionClassification_1.vsd

Figure 373 Overview of Channel Functions

35.3.3.3 Acknowledge Responses

A read or write command is executed by the SRI / SPB master. The SRI / SPB master reads or writes to the SRI / SPB bus and signals either a successful transaction or an error. The are the following cases:

- When the SRI master writes or reads an SRI memory, both cases are acknowledged so that the HSSL module receives a feedback in any case.
- When the SPB master writes or reads an SPB module or memory, both cases are acknowledged so that the HSSL module receives a feedback in any case.

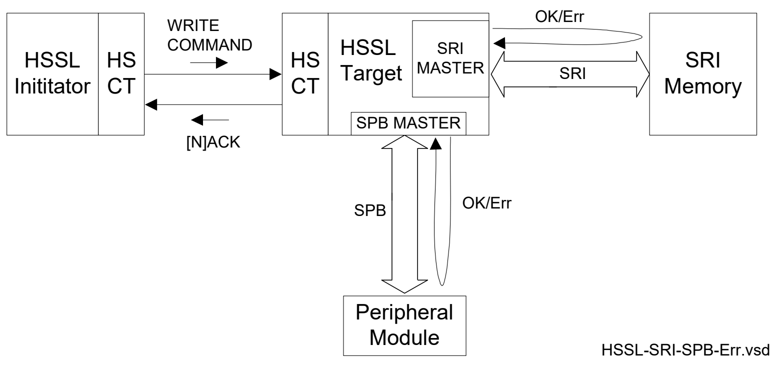


Figure 374 Bus Error Feedback Paths

35.3.3.4 Cross Dependencies Between the Frame Types

For a channel in a target role, there are cross-dependencies between command frame arrival and triggering of an ACK or NACK frame for a channel. Each command frame received with correct CRC of types Write, Stream, or Trigger Frame will result in either ACK or NACK (target error) response frame. Each Read Frame received with correct CRC will result either with Data Frame (Read Response Frame) or with NACK (Target Error Frame), see **Figure 375**.

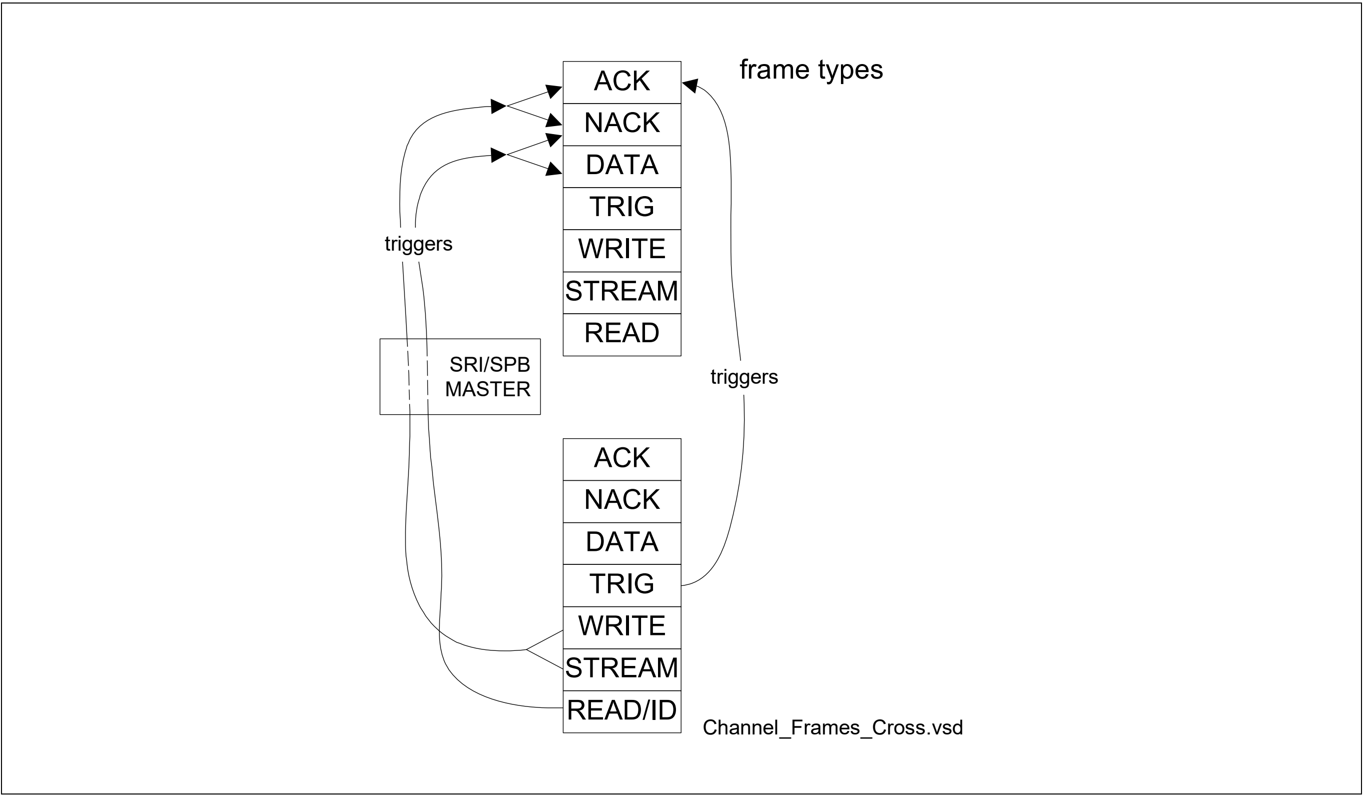


Figure 375 Channel 2 Cross Dependencies Between Stream and Command Frames and their Responses in a Target Role

Additionally, generating a response frame involves copying of the transaction tag of the command frame into the response frame. The response frame is triggered by an SRI/SPB master signalling either a successful SRI/SPB bus transaction or an SRI/SPB bus error. Only a Trigger Command Frame triggers immediate acknowledge, because it results in an interrupt flag being set, and not in a bus transaction.

On the initiator side, there is a forward looking cross dependency between a command frame and the expected response, see **Figure 376**. Setting a transmission request flag for a certain command is accompanied in parallel with:

- capturing the current transaction tag in the bit field **ICONx (x=0-3)**.CETT.
- generating the next transaction tag by incrementing a three bit counter per channel, the **ICONx (x=0-3)**.ITTAG.
- starting the timeout counter at the moment when the command wins the arbitration
- setting an expect tag indicating that the timeout is running and a response frame is expected. See **MFLAGS**.Ex bit fields.

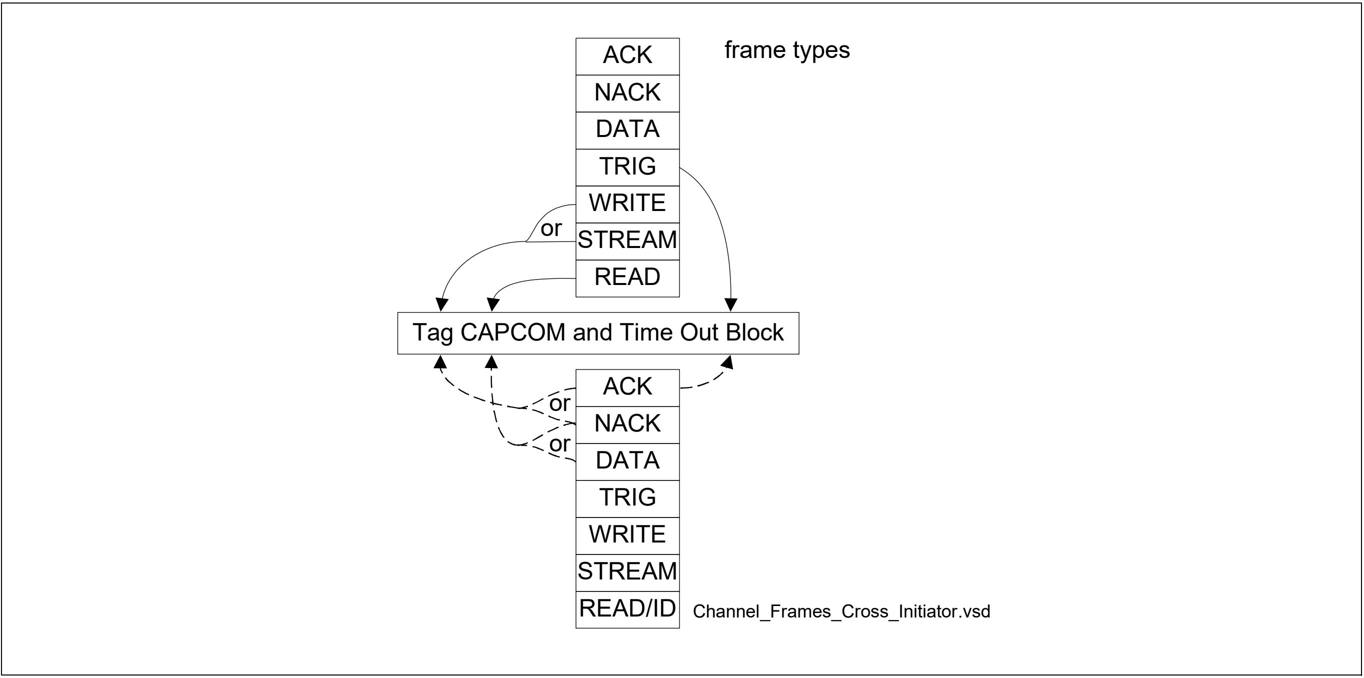


Figure 376 Channel 2 Cross Dependencies Between Stream and Command Frames and their Responses in an Initiator Role

35.3.3.5 Command Timeout

A timeout timer is started when a frame wins the arbitration and is delivered by the channel for transmission. The timeout timer is stopped if an appropriate response frame has arrived in time (without a CRC error). If an appropriate response has not arrived, and the timeout timer reaches zero, the timer is stopped, a timeout error is triggered and the expect flag is cleared.

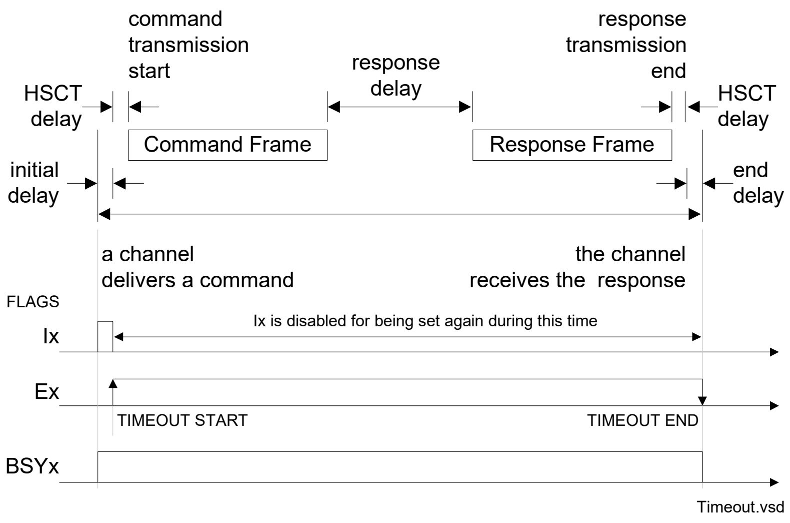

command transmission start
response delay
response transmission end
HSCT delay
HSCT delay
Command Frame
Response Frame
initial delay
end delay
a channel delivers a command
the channel receives the response
FLAGS
Ix
Ix is disabled for being set again during this time
Ex
TIMEOUT START
TIMEOUT END
BSYx
Timeout.vsd

Figure 377 Command Timeout Measurement

35.3.3.5.1 Command Timeout Operation

The HSSL module contains one prescaler common to all channels, and one channel specific timer block per channel. The timer block generates the transaction tag for the channel by incrementing a three bit counter, captures the current transaction tag and compares it with the arrived response. In case of an error sequence CRC and timeout error, the timeout block generates a timeout signal. The timeout timer is reloaded and starts down-counting when the command transmission process starts.

The current timeout timer current value is indicated in the bit field **ICONx (x=0-3)**.TOCV, and the timeout reload value is configured in the bit field **ICONx (x=0-3)**.TOREL.

The currently expected transaction tag is indicated in the bit field **ICONx (x=0-3)**.CETT, the captured value of the latest erroneous transaction tag is indicated in the bit field **ICONx (x=0-3)**.LETT.

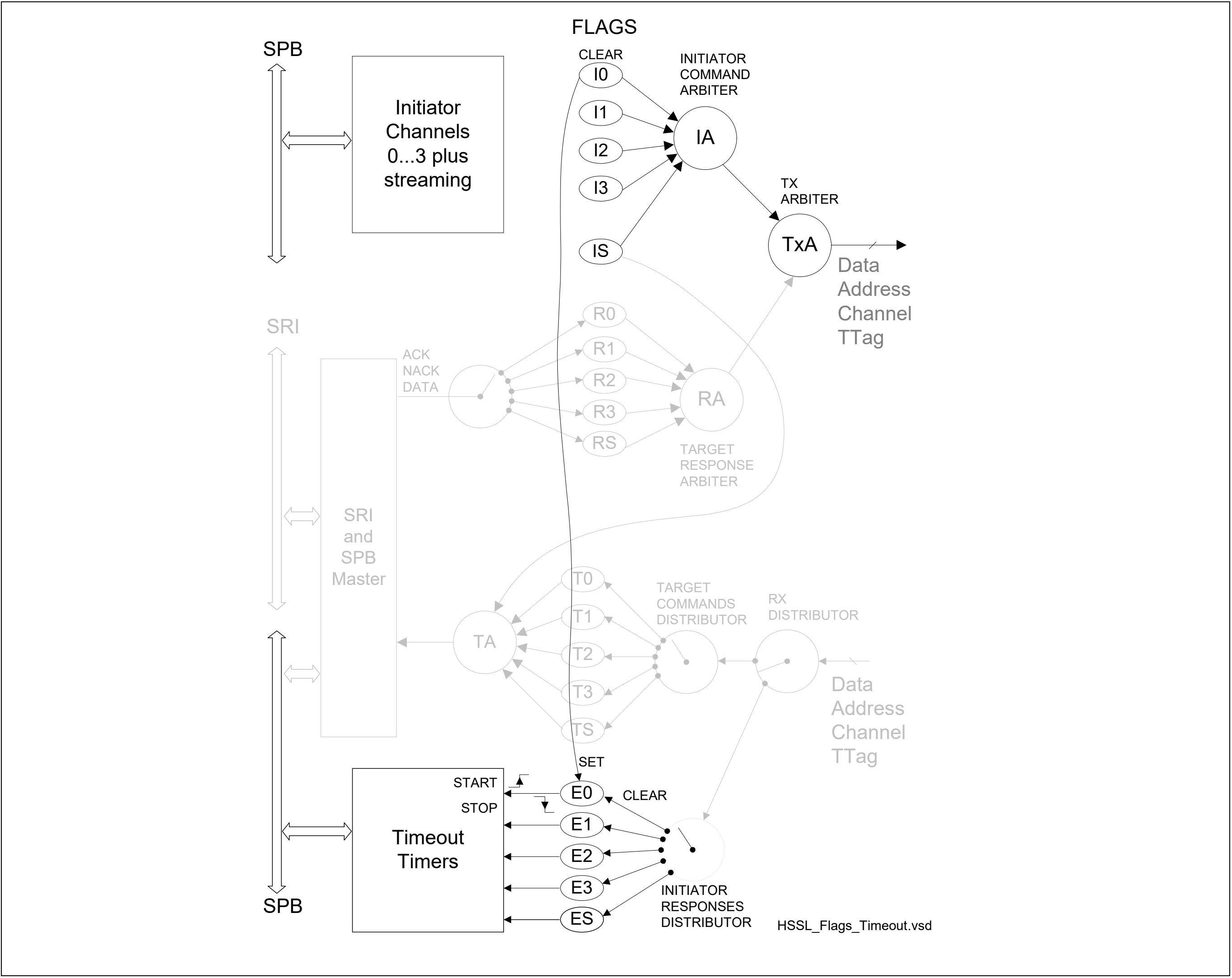


Figure 378 Request Flags Overview and a Communication Cycle Example

From the point of view of the flags, the timeout starts with the rising edge of the **QFLAGS**.Ex flag and ends with its falling edge, see **Figure 376**. The Ex flag is set by the falling edge of the Ix flag, which is reset when its command request wins the transmit arbitration, see **Figure 378**.

35.3.3.6 Stream Timeout

A free timeout timer block is started when a frame wins an arbitration round and is delivered by the channel for transmission. The timeout timer block is reset to zero and stopped if an appropriate response frame has arrived in time (without a CRC error). If an appropriate response has not arrived, a timeout error is triggered and the timeout timer block is reset.

The stream initiator keeps track of the expected acknowledges using a virtual expect fifo having a virtual filling level EXFL. Each new stream frame transmission increment the EXFL, receiving a correct acknowledge decreases the EXFL. As long as the expect fifo is full (filling level 2) no new transmit request can be issued. In order a new stream frame request to be issued, which means the flag IS to be set by the module, three conditions must be met:

- EXFL<2 - the expect level must be below two,
- TXFL>0 - the TXFIFO must be not-empty and
- ISB = 1 - the bit ISB bit must have been set by the software.

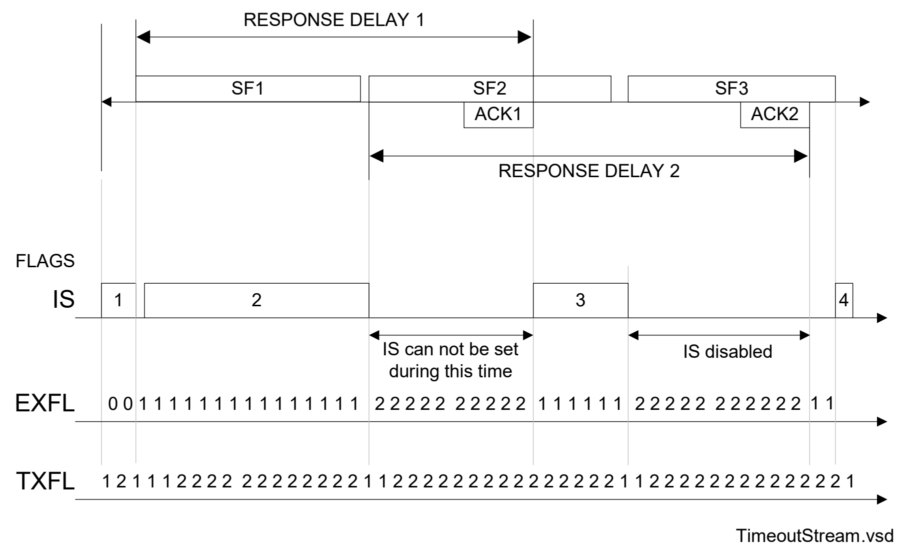


Figure 379 Stream Timeout Measurement

35.3.3.6.1 Stream Timeout Operation

The stream channel monitors the arrivals of the stream frame responses by using two timeout timer blocks. A single timer block is shown in **Figure 380**.

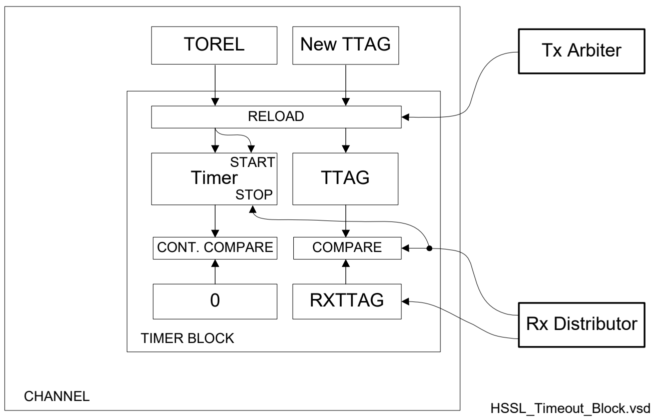


Figure 380 Single Timeout Block

The new initiator TTAG value which will be used for the next frame is visible in the bit field **ICONx (x=0-3)**.ITTAG. The **ICONx (x=0-3)** register also contains the TOREL and the TOCV bit fields.

The streaming functionality uses two timers sharing the same reload value, as defined by the **ICONx (x=0-3)**.TOREL, i=2. The timeout timer blocks simulate the behavior of a FIFO (named Expect FIFO), by having additionally a “write” and “read” pointer:

- the “write” or transmit pointer points to the timer block where the next command frame will trigger the writing of the timer reload value and the transaction tag,
- the “read” or receive pointer points to the block which performs the transaction tag comparison with the currently arrived stream acknowledge TTAG.
- The EXFL or virtual filling level indicating how many timers are active 0, 1 or 2

The EXFL is visible in the register **SFSFLAGS**.

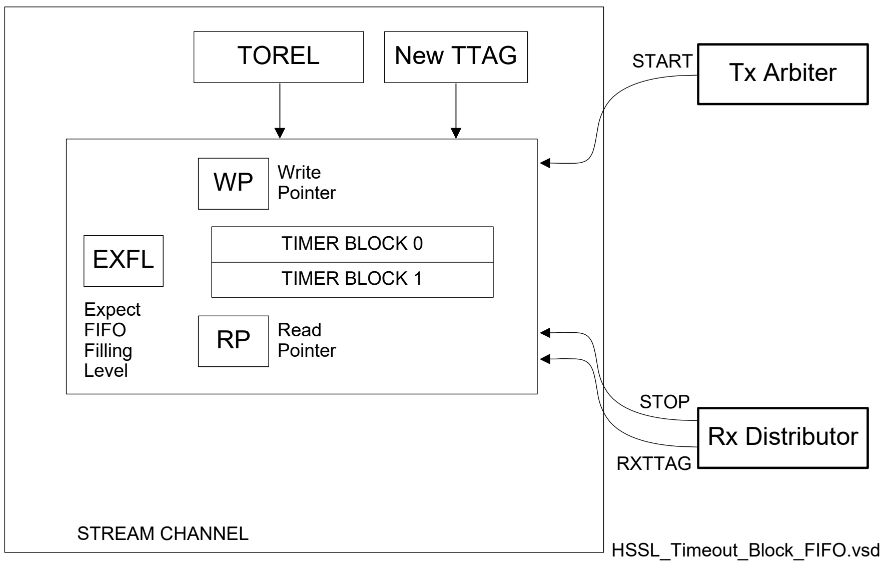


Figure 381 Stream Timeout Block

35.3.3.7 Data FIFOs of the Streaming Channel 2

Each stream frame delivers 256 bit data. This data is transferred by the SRI/SPB bus master in a memory by using BTR4 bursts. The start address and the end address is aligned at 256-bit block border, see **ISSAx (x=0-1)**, **ISFC**, **TSSAx (x=0-1)**, **TSFC**.

The channel expects one full streaming frame of 256 bit to be delivered before triggering either the SRI/SPB master or the HSCT module for fetching the data. The streaming is full duplex capable, which means one FIFO for each direction is available, and also Priorisation for the requests to the SRI/SPB master.

Emptying the RXFIFO has higher priority than filling the TXFIFO.

A service request to the SRI/SPB master is generated when the complete payload of a streaming frame is available in a FIFO.

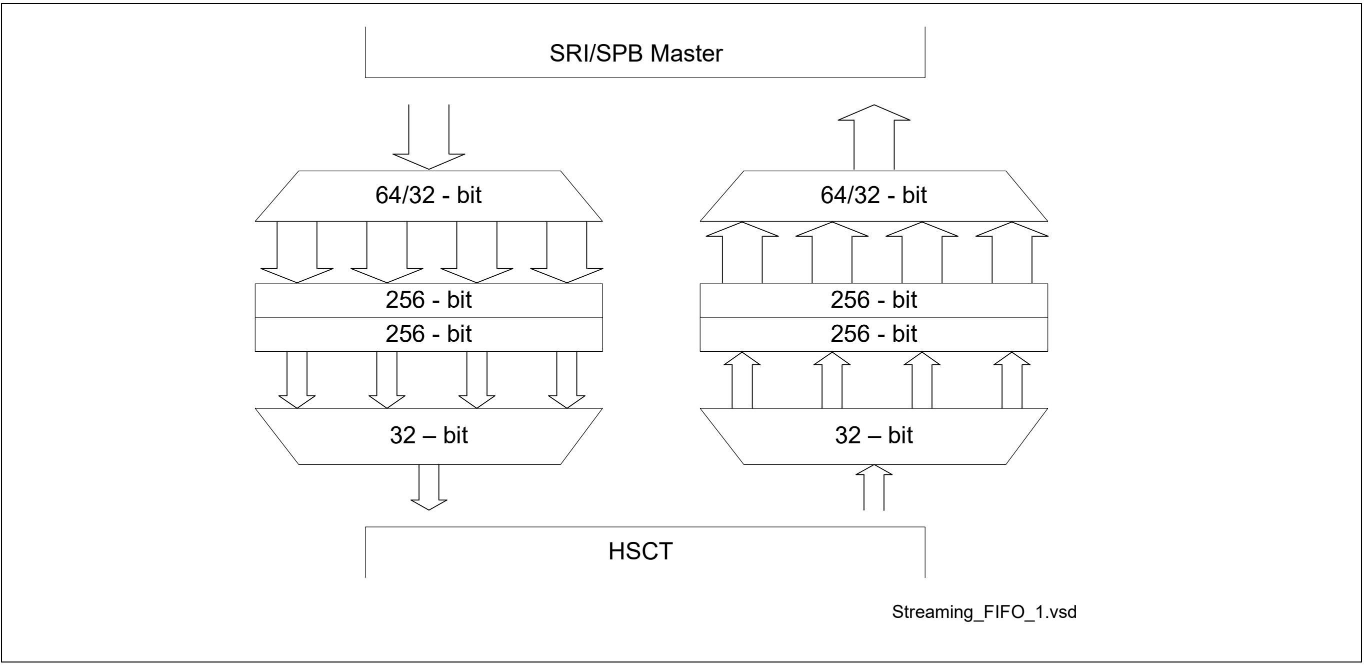


Figure 382 Streaming FIFO

35.3.4 Modes of Operation

The HSSL module has the following modes:

• Disabled mode	(software controlled via DISR bit)
• Initialize mode	(software via INI bit, sleep & suspend signals)
• Run mode	(INI bit, sleep and suspend signals)
• OCDS soft suspend mode	(OCDS suspend signal)
• Sleep mode	(sleep signal)

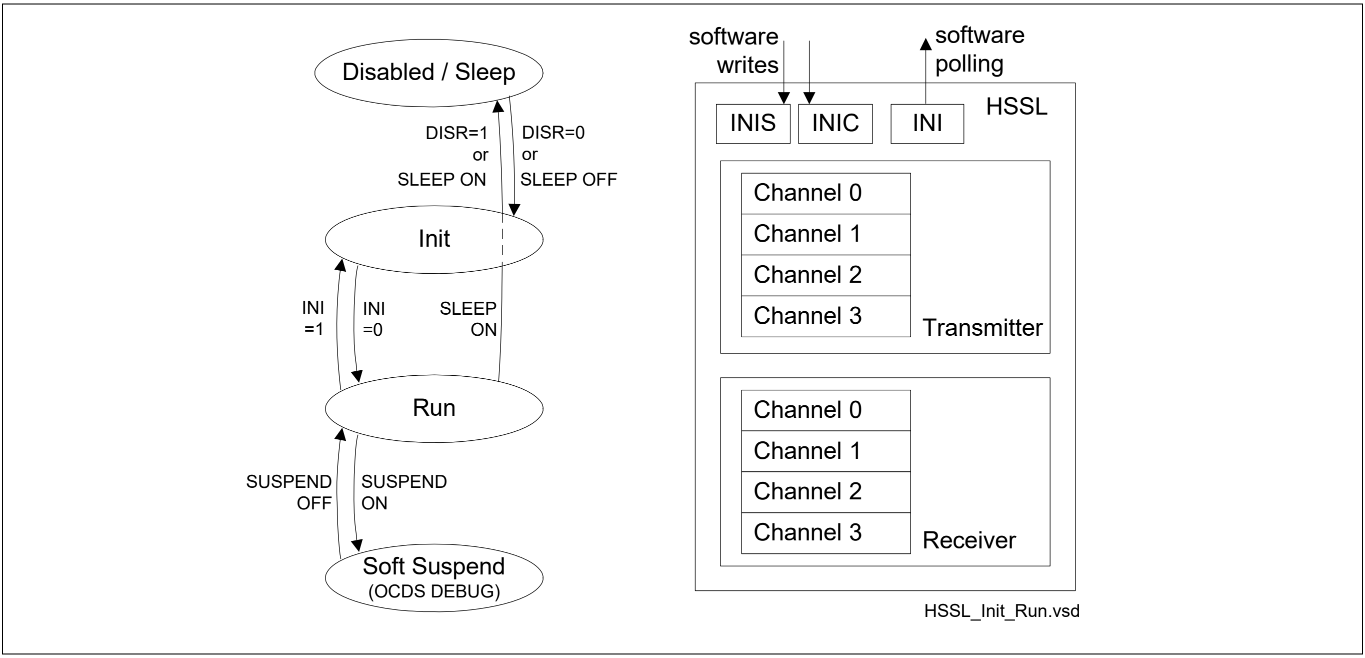


Figure 383 Modes of operation

After INIT state has been requested by using the **MFLAGSSET**.INIS bit, the HSSL module immediately stops starting new SRI/SPB transactions and new command, response and stream frame transmissions by resetting all request bits (I, T and R) in the **QFLAGS** register and disabling these bits for being set again. The already started transactions/transmissions will be finished. Afterwards, it waits for all pending response frames to arrive or the corresponding timeout events to be raised. This condition is achieved when all expect bits in the **QFLAGS** register has been cleared by the hardware, and there are no ongoing transmission and SRI/SPB master activities. Subsequently the module enters the Init state and the **MFLAGS**.INI flag is set. The CTS signal is deactivated in order to inform the peer module about the Init state.

During the transition from run to sleep state, triggered by the sleep signal, the **MFLAGS**.INI bit is automatically set by the hardware (behaves as INI bit was set by software). When the sleep period is over and the sleep signal is deactivated, the module goes into the Init state and must be set to Run state by software, by writing **MFLAGSCL**.INIC=1.

At the state transition of Init -> Run the timeout timers and all the flags are reset. The RUN state starts with all channels inactive.

At the state transition SoftSuspend -> Run the timers are not reset but continue to run and the flags states are preserved, because the module continues operating from the point at which it was suspended.

When leaving the Run state towards Sleep the CTS signal is deactivated and the transmission of the pending commands is stopped.

When a soft suspend is requested, the module stops the transmission of the pending command and stream frames, waits until all expect flags are cleared, then deactivates the CTS signal and at the end acknowledges the soft suspend request and sets the bit **OCS**.SUSSTA. Incoming target frames are served until the CTS signal stops the other side from sending new command and stream frames.

Hard Suspend request causes immediate switching off of the module clocks. After ending the Hard Suspend state both HSSL and HSCT modules must be reset by the application software and communication restarted from reset state.

Note: Reading and writing of registers is possible but will enable the kernel clock f_{CLC} for a few cycles. Attention: register accesses with clocking in Hard Suspend Mode can have unintended side effects like signals becoming and staying active. This can affect also other modules, so a HSSL kernel reset might not be sufficient to bring the system into a defined state.

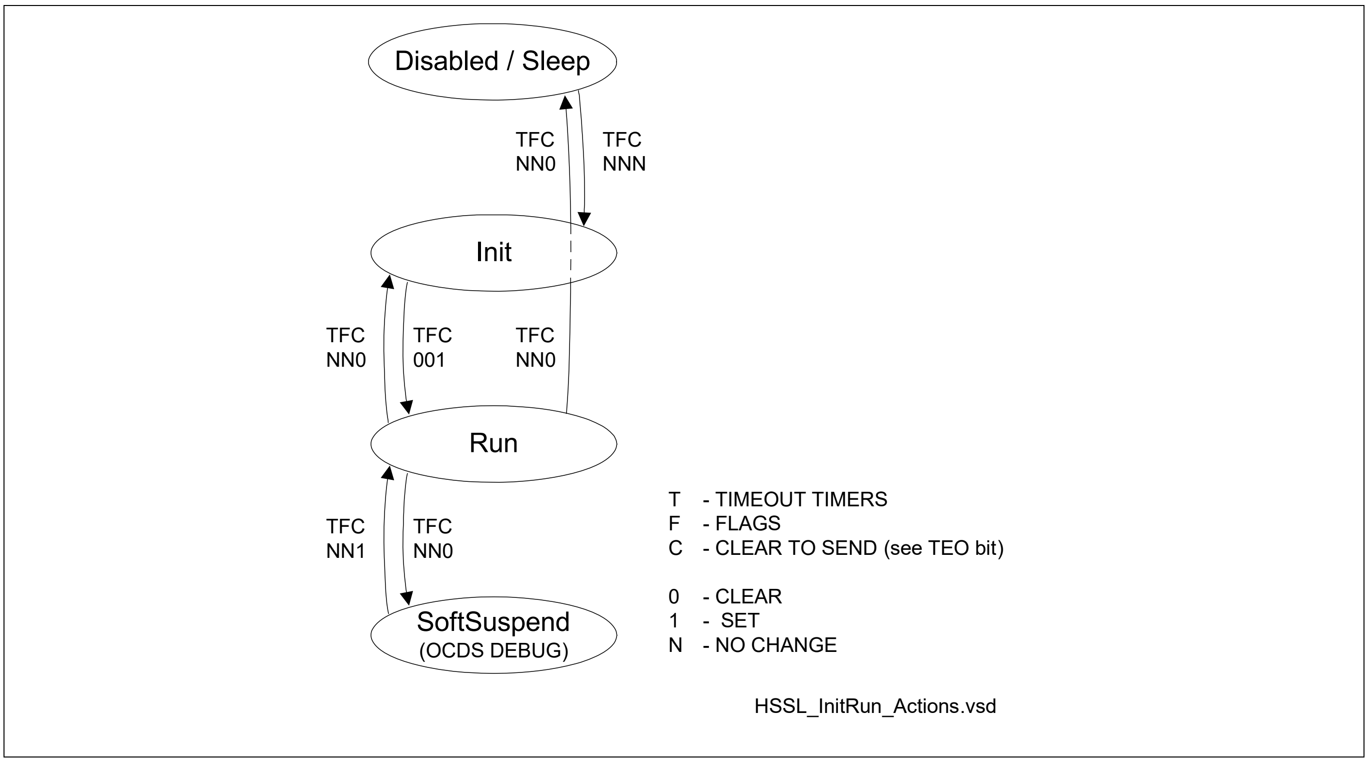


Figure 384 Actions on State Transitions

35.3.5 Interrupts

The HSSL module generates four types of interrupts per channel and one channel unspecific module interrupt.

Command Channel Interrupts

Each HSSL command channel generates four interrupts:

- Command OK interrupt COK
- Read Data interrupt RDI
- Error interrupt ERR
- Trigger interrupt, triggered by a trigger command frame TRG

The names of the interrupt signals are COK_INT, RDI_INT, ERR_INT and TRG_INT correspondingly.

An arrival of an error free frame response (ACK frame) triggers a COK interrupt, otherwise an ERR interrupt is triggered. An arrival of a read response frame triggers at the initiator side, additionally to the COK interrupt, an RDI interrupt.

An arrival of a trigger frame at the target side triggers a TRG interrupt there.

ERR interrupt is disjunct to COK and RDI the , meaning that for one frame, either ERR or COK and optionally RDI can occur. After an ERR interrupt, normal transmission must be resumed by software, because an optional DMA would remain not retriggered and would wait for COK indefinitely.

Stream Channel Interrupts

The HSSL stream transmit sub-channel generates three interrupts:

- TBE - Transmit Block End interrupt, shared with the COK interrupt of the command mode of the channel 2
- error interrupt ERR, shared with the ERR interrupt of the command mode of the channel 2
- RBE - Receive Block End interrupt, shared with the TRG interrupt of the command mode of the channel 2.

Exception Interrupt EXI

If the receive stage of the HSSL module detects a CRC error or any inconsistency in the received data the global EXI Interrupt is activated, which is not channel specific.

Note: The HSCT module uses channels A to D for the HSSL channels 0 to 3. So, HSCT channel 0100_B corresponds to 000_B (binary code) or 100_B (special code) in the HSSL header.

The EXI interrupt is also raised on an edge of the TEI signal (HSCT CTS output).

In total, the HSSL module provides 4*4 + 1 = 17 interrupt lines.

35.3.6 Operating a Command Channel

All HSSL command channel provide identical functionality.

35.3.6.1 Initiating a Single Write Command

In order to initiate a write command, the software must

- configure the data width and the type of the request which will follow (read or write)
- provide the new data (most frequently, but not necessary)
- provide new address

Write to the address register requests the predefined type of the transfer

35.3.6.2 Initiating a Single Read Command

In order to initiate a read command, the software must

- configure the data width and the type of the request which will follow (read or write)
- skip providing of the data step, or provide dummy data
- provide new address

Write to the address register requests the predefined type of the transfer

35.3.6.3 Initiating a Single Trigger Command

In order to initiate a trigger command, the software can either

- request a trigger type of frame and provide dummy data and address
- directly request a trigger frame by writing **ICONx (x=0-3)**.TQ

35.3.6.4 DMA Operated Command Queues

It is possible to use DMA to initiate lists of commands, by moving memory blocks containing the command lists to the HSSL module. In order to support such a way of operation, each channel provides three registers arranged in a particular way: first **IWDx (x=0-3)**, followed by **ICONx (x=0-3)** and **IRWAx (x=0-3)** at the end. For executing a list of write commands all three registers must be written for each command; for executing a list of read commands the first register **IWDx (x=0-3)** is optional (dummy write possible) but the last two **ICONx (x=0-3)** and **IRWAx (x=0-3)** are mandatory; for executing a trigger command only writing **ICONx (x=0-3)**.TQ is mandatory, but optionally even all three registers can be written: dummy write to **IWDx (x=0-3)**, write to **ICONx (x=0-3)**.RWT and write to **IRWAx (x=0-3)** to issue the request.

The answer data to a read command is available in the **IRDx (x=0-3)** register.

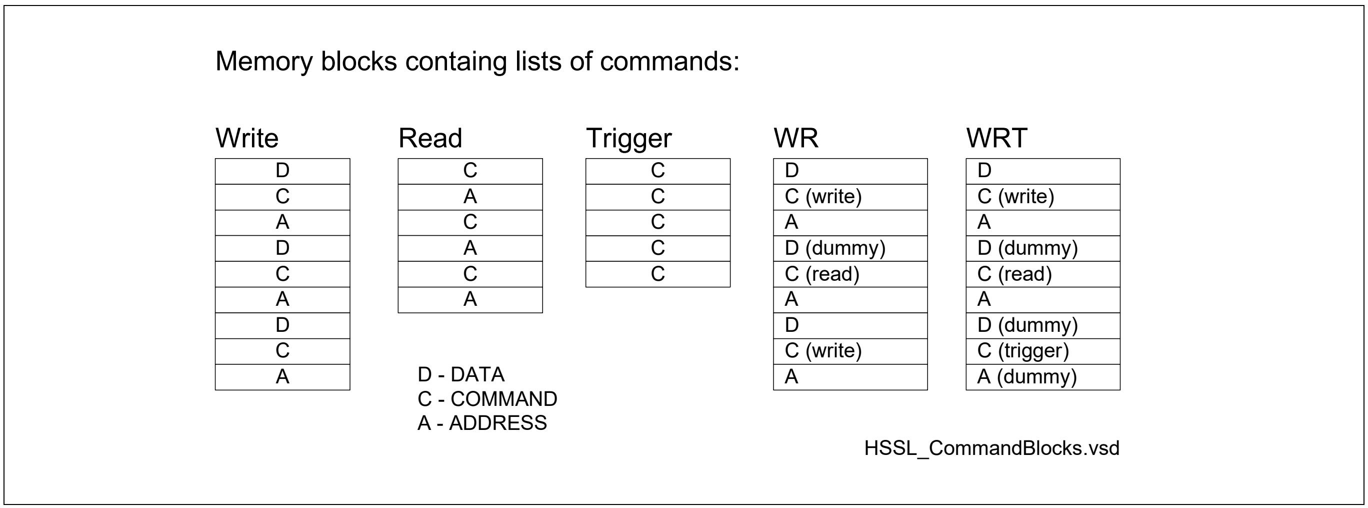


Figure 385 Command Queues

35.3.6.5 Receiver Error Handling

The HSSL receiver generates three error signals, two channel specific and one general:

- channel specific timeout and transaction tag errors and
- channel unspecific CRC error
- command frames, which pass the CRC check, containing an RFU command code are ignored
- Command frames, which pass the CRC check, containing invalid (greater than 3) channel number are ignored.

- If a command frame is received and the acknowledge of the previous command has not been sent yet, the new command frame is discarded.
- If a response frame comes, and the channel does not expect any more a response (the E flag is 0), than an incoming response is discarded without any error interrupts, additional to the timeout interrupt. This scenario can occur if a response is received after a timeout, and before the next command has been sent.
- If a channel in command mode receives a stream frame, the stream frame will be ignored.
- if a channel in streaming mode receives a command frame, the command frame will be ignored.

35.3.6.5.1 Timeout Error

Error triggered by the initiator.

35.3.6.5.2 Transaction Tag Error

Error triggered by the initiator.

35.3.6.6 Global Error

Global, channel not specific error is triggered if a CRC error, inconsistency between the HSCT and HSSL header, and SRI/SPB error occurs.

CRC error is triggered:

- by the target: when erroneous command frames has been received and
- by the initiator for erroneous response frames.

There is a dedicated interrupt named EXI which signals the CRC error. There is no indication if the CRC error was caused by a command or response frame. The CRC error caused by a response frame will be followed by a timeout interrupt. The received data is discarded; it is not available for the software in any register(s).

Additionally, the CRC error interrupt signals an inconsistency between the data length and channel number code delivered by the HSCT module and the information contained in the HSSL header. The differentiation between the two types of error can be done by reading the **MFLAGS**.CRCE, PIE1 and PIE2 flags. All three error types can occur and are detected independently of each other, so that more than one flag can be set at the same time.

An SRI error interrupt (EXI) is triggered if there was a transaction ID error, ECC error or SRI error acknowledge on the SRI bus. The flag **MFLAGS**.SRIE indicates this type of error. In case of an SRI read data ECC error, the READ_PH_ERR_ALARM signal reports this event to the SMU module.

An SPB error also triggers the global error interrupt (EXI) and sets the **MFLAGS**.SRIE flag.

35.3.7 Memory Block Transfer Modes of the Stream Channel

HSSL streaming channel is capable of transmitting one stream and receiving one stream in parallel. Streaming operates either in single block mode or continuous mode.

In single block mode, after triggering/enabling the streaming by using the **MFLAGS**.ISB/TSE bit, the SRI/SPB master of the HSSL module transmits/receives the preconfigured memory block, generates an interrupt signal and waits for the next trigger.

In continuous mode, after triggering/enabling the streaming by using the **MFLAGS**.ISB/TSE bit, the SRI/SPB master of the HSSL module transmits/receives two memory blocks of the same size in a round robin fashion indefinitely, or after being stopped by the software.

The streaming mode of the initiator/transmitter is configured by using the **CFG**.SMT bit. The streaming mode of the target/receiver is configured by using the **CFG**.SMR bit.

Triggering/enabling a block or continuous stream is done by using the Stream Block Request bit **MFLAGS**.ISB/TSE bit. In single block mode, the hardware resets this bit after performing a block transfer. In continuous mode, the transfer is ongoing continuously (**MFLAGS**.ISB/TSE remains set) and a stop of the transfer must be requested by writing one to **MFLAGSCL**.ISBC/TSEC bit.

On the initiator side, after completing the transfer of the current memory block that is currently read from the memory, the transmitter will be stopped and the **MFLAGS**.ISB will be cleared.

On the target side, after completing the transfer of the current memory block that is currently written to the memory, the receiver will be stopped and the **MFLAGS**.TSE will be cleared.

SRI/SPB error on the target triggers an error interrupt and the hardware stops. The TSE bit (and the RXFIFO) will be cleared by the hardware and the incoming frames are ignored.

Any error on the initiator side (like SRI/SPB error, target error, timeout, transaction tag error) triggers an error interrupt and the hardware stops streaming. The bit ISB (and the TXFIFO) will be cleared by hardware. All the incoming frames will be ignored, including the acknowledge frames of the previous frames.

Error can also occur at the beginning of a new memory block, while the last frames of the previous memory block are still in the TXFIFO. These frames are also lost, and the previous memory block remains incompletely received.

After an error while streaming in one direction the software must restart the streaming in that direction at both sides. The streaming will restart with a new block.

An error while streaming in most cases leaves the target waiting in the middle of a block. The target receiver can be stopped immediately (and afterwards restarted) by clearing the **MFLAGS**.TSE only when **TSFC**.CURCOUNT equals **TSFC**.RELCOUNT. Normally, CURCOUNT equals RELCOUNT before the start of a transfer of a block or between block transfers. Otherwise, while waiting in the middle of a block transfer, RELCOUNT can be made equal to CURCOUNT by writing the CURCOUNT value to RELCOUNT. This can be done either locally, by the target itself, or remotely, by the initiator using HSSL command frames.

The two initiator start addresses are configured in **ISSAx (x=0-1)**. The single block size is configured in the **ISFC** register. The current address being transferred is indicated in the read only (rh) **ISCA** register. The bit **MFLAGS**.IMB selects the corresponding **ISSAx (x=0-1)** register. This bit is not modified by the hardware in single block mode, and is toggled by the hardware in the continuous mode. At the start of the transfer, the selected start address **ISSAx (x=0-1)** is copied to the **ISCA** register.

The two target start addresses are configured in **TSSAx (x=0-1)**. The single block size is configured in the **TSFC** register. The current address being transferred is indicated in the read only (rh) **TSCA** register.The bit **MFLAGS**.TMB selects the corresponding **TSSAx (x=0-1)** register. This bit is not modified by the hardware in single block mode, and is toggled by the hardware in the continuous mode. At the start of the transfer, the selected start address **TSSAx (x=0-1)** is copied to the **TSCA** register.

The HSSL stream transmitter generates one dedicated interrupt and several error events:

- transmit block end interrupt, generated after the acknowledge of the last frame has been received.
- error events that can trigger a channel error interrupt (target, timeout, transaction tag, unexpected error)
- a global SRI/SPB error interrupt (check if the ISB bit is cleared by hardware)

The HSSL stream receiver generates one dedicated interrupt and several error events:

- receive block end interrupt event, generated after the last stream frame in a block has been written to the memory and the SRI/SPB master has received a confirmation (an acknowledge, but not an error).
- error event (target error that triggers an NACK frame)
- a SRI/SPB global error interrupt (check if the TSE bit is cleared by hardware)

As long as the ISB bit is low, the TXFIFO and the expect FIFO are cleared and kept in the empty state.

As long as the TSE bit is low, the RXFIFO is cleared and kept in the empty state.

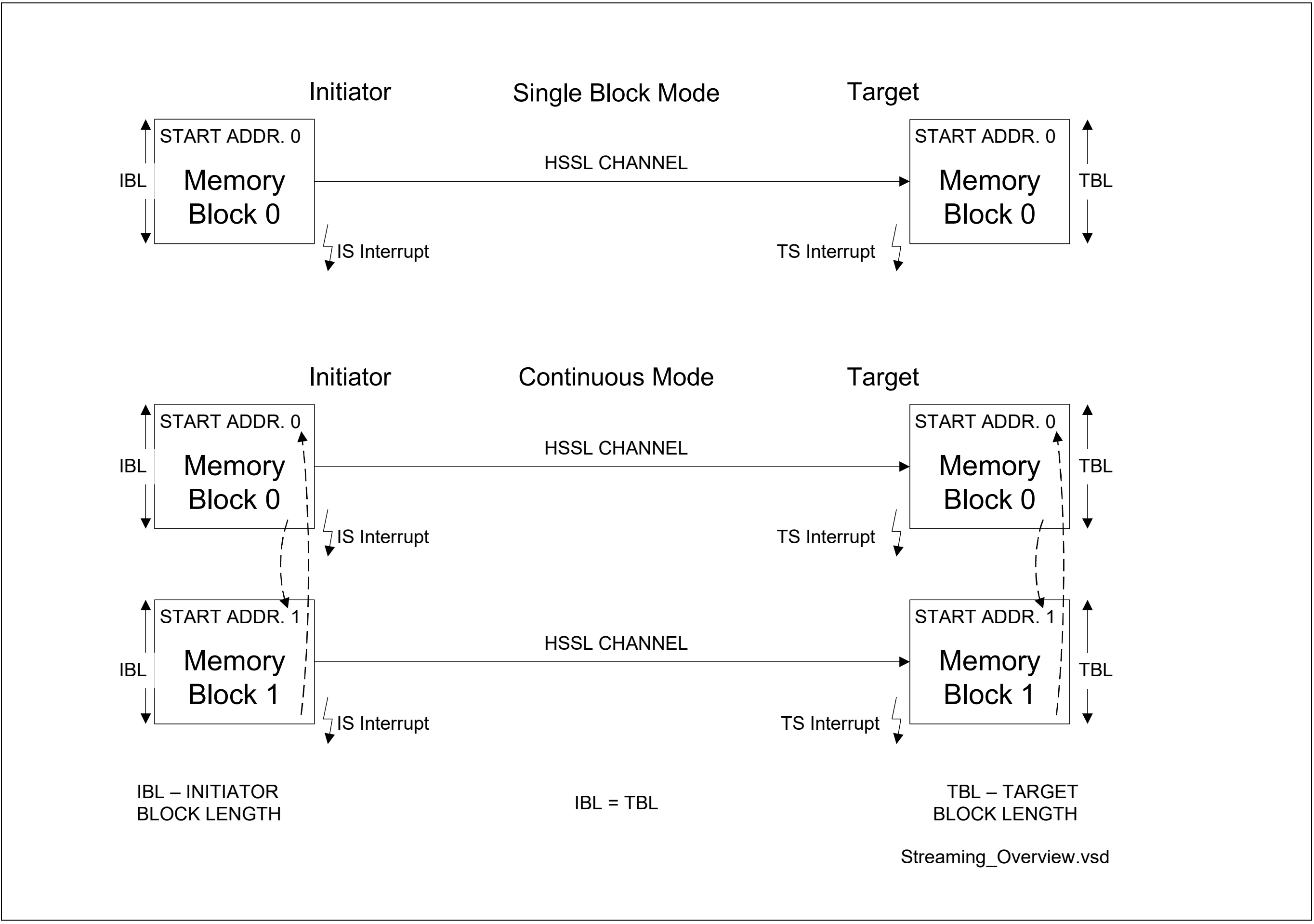


Figure 386 Stream Modes of Operation

35.3.8 HSSL Reset

Both HSCT and the HSSL modules must be reset together.

35.3.9 OCDS SRI / SPB Master Suspend

The SRI /SPB master access on the bus can be suspended by using the OCDS suspend request. When the master reaches idle state the suspend acknowledge signal is activated.

35.3.10 OCDS Trigger Sets

In order to support the debugging activities, the HSSL module provides a set of internal signals to the on-chip debug system OCDS. An overview of this feature is shown in **Figure 387**. Its configuration is done by using the bits **OCS**.TGS and TGB.

An edge on any module internal signal belonging to the selected set going out on one of the two OTGB busses triggers an action of the OCDS system.

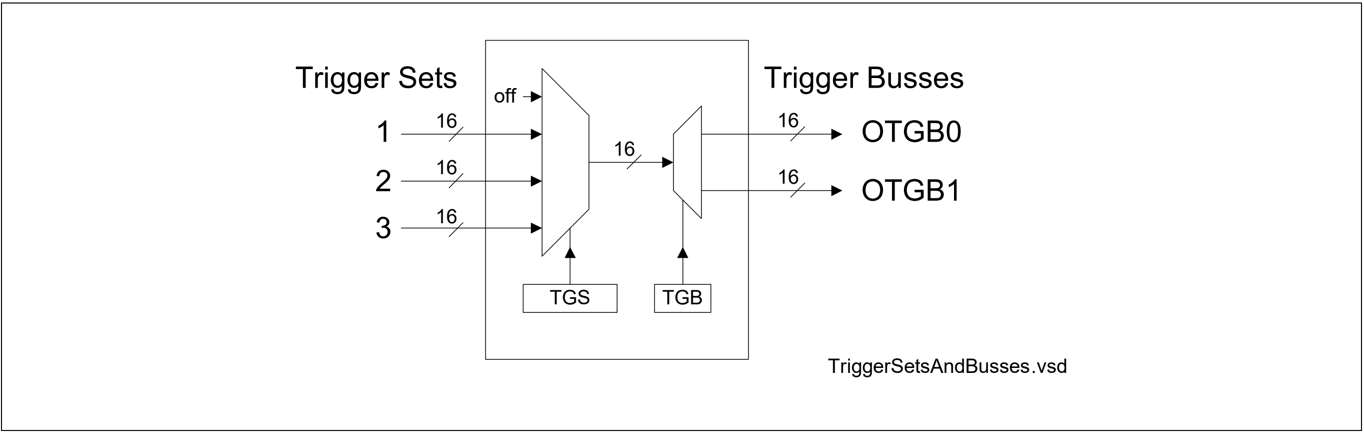


Figure 387 Overview of the Trigger Sets and Busses

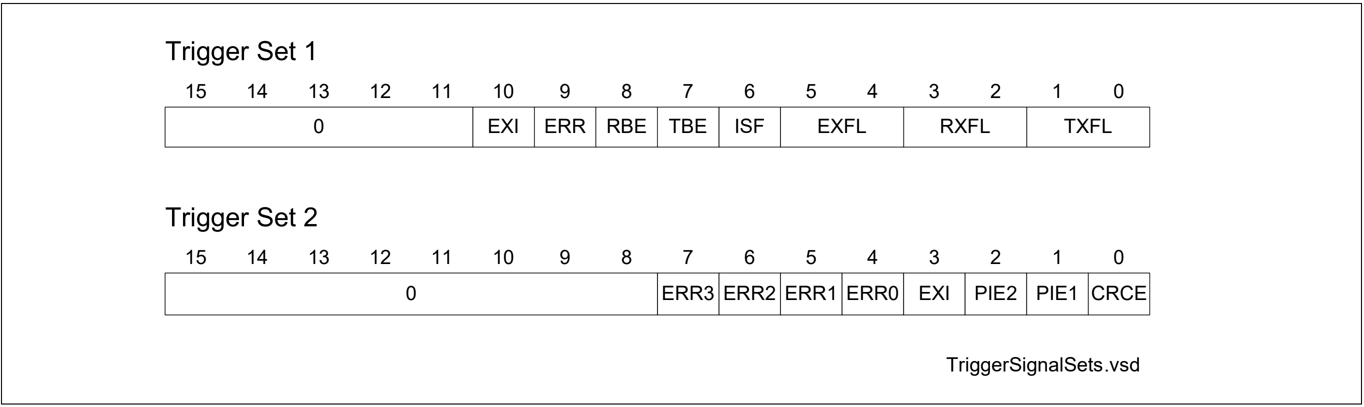


Figure 388 Overview of the Trigger Signal Sets

Table 309 HSSL Trigger Sets

Name	Description
TS16_STR	Streaming Channel Trigger Set (Trigger Set 1)
TS16_ERR	Error Trigger Set (Trigger Set 2)

The trigger sets 1 and 2 are used, the set 3 is not used (padded with 0).

Set 1, streaming channel debugging:

- 2 bits: TXFIFO filling level **SFSFLAGS**.TXFL
- 2 bits: RXFIFO filling level **SFSFLAGS**.RXFL
- 2 bits: Expect FIFO filling level **SFSFLAGS**.EXFL
- 1 bit: Initiator Stream Frame Request **SFSFLAGS**.ISF
- 4 bits: streaming channel interrupt signals TBE, RBE, ERR, EXI, see **Interrupts**.

- others: not used

Set 2, timeout errors caused by unspecific errors debugging:

- 1 bit: CRC error **MFLAGS**.CRCE
- 1 bit: Phy Inconsistency error, inconsistent channel number code, **MFLAGS**.PIE1
- 1 bit: Phy Inconsistency error, inconsistent data length, **MFLAGS**.PIE2
- 5 bits: error interrupt signals 1 x EXI and 4 x ERR, see **Interrupts**.
- others: not used

Note: The signal lists from above are mapped to the trigger sets in the following way: first item from the list to the trigger signal 0,… to 15.

35.3.11 Access Protection

The HSSL module provides memory access protection in form of four memory access windows. Each window can be located anywhere within the address space, having an arbitrary size with the granularity of 256 bytes.

Each window defines a memory range where an access is allowed. All four windows create a sort of an access filter that protects the memory from external writing or reading. A window can be a read only, write only or read and write window.

Each window is defined with:

- window start address, see register **AWSTARTi (i=0-3)**
- window end address, see register **AWENDi (i=0-3)** and
- access rule: read only, write only or read and write, see register **AR**

If two access windows overlap, one read only and one write only, the common address range is read and write accessible. If no access window overlaps with r, w or rw window, the r, w or rw window wins. No access window means window disabled.

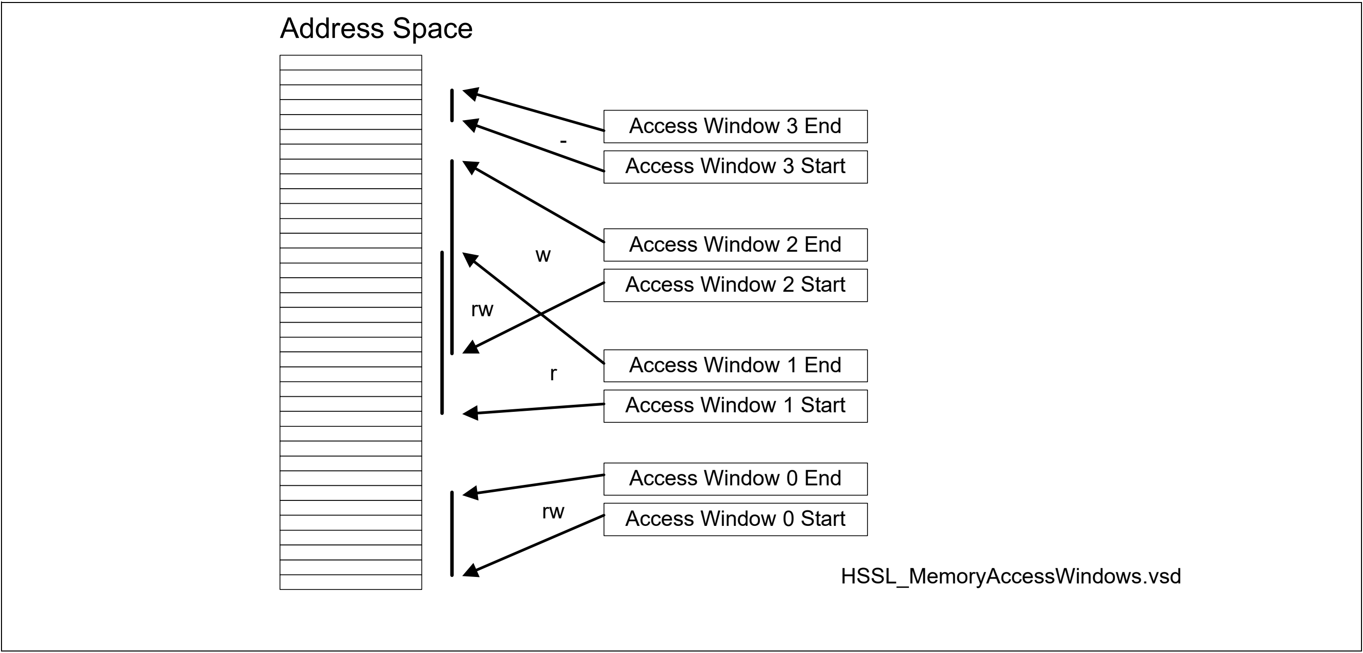


Figure 389 HSSL Access Windows

Normally, it is expected that the End Address is higher then or equal to the Start Address. Otherwise, the window is invalid and no commands can pass through the up-side-down range.

The **AWSTARTi (i=0-3)** registers define the first address of the first 256-byte block of an address window. The **AWENDi (i=0-3)** registers define the first address of the last 256-byte block of an address window. The following diagram shows the details of the address window definition.

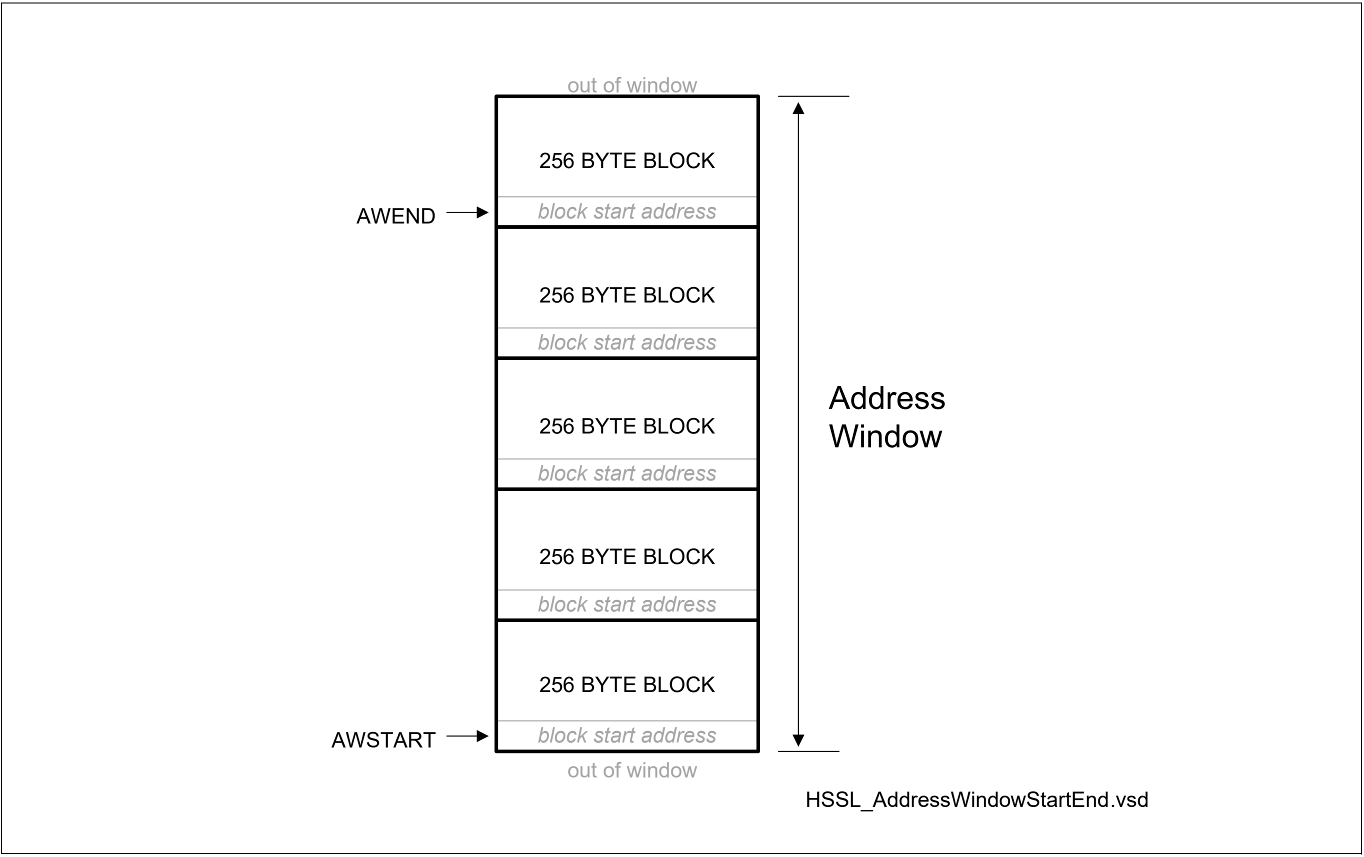


Figure 390 Access Windows Definition

These command frames are subject to filtering:

- Register Read
- Register Write and
- Stream Frame

These command always pass through:

- Read Answer
- ACK
- NACK
- Trigger
- ID Request

Command Frames Access Protection

If a HSSL target receives a read or write command frame with an address which passes through one of the windows, the command is executed and a response frame is sent back.

If a HSSL target receives a command frame with an address which does not pass through any windows, the command is not executed, a NACK response is sent back. The memory access violation flag **MFLAGS**.MAV is set, the access violating channel is captured in the bit field **AR**.MAVCH, and the EXI interrupt is triggered in the target.

Streaming Access Protection

In contrast to the read and write command frames, the stream frames do not contain an address information. There are two cases to distinguish, target side and initiator side.

At the target side, when the streaming starts, the initiator has already programmed the target addresses in the **TSSAx (x=0-1)** registers by using write frames, which pass through and carry the stream address in their data field. Therefore, the memory filtering uses the current access address, visible in the **TSCA** register, and if the current streaming address passes through the access protection filter, the read or write is executed, else a NACK frame is sent as a response.

If a HSSL target receives a stream frame and the current access address does not pass through any windows, the access is not executed, a NACK response is sent back. The TSE bit will be cleared by hardware and the incoming frames will be ignored. The memory access violation flag **MFLAGS**.MAV is set, the access violating channel is captured in the bit field **AR**.MAVCH, and the EXI interrupt is triggered in the target.

At the initiator side, the **ISSAx (x=0-1)** registers, which are externally accessible, define the memory range to be transmitted. Therefore, the access filtering at the initiator side covers the streaming read accesses by using the current access address, visible in the **ISCA** register.

At the initiator side, access violation stops the streaming, and the ISB bit is cleared by hardware. The memory access violation flag **MFLAGS**.MAV is set, the access violating channel is captured in the bit field **AR**.MAVCH, and the EXI interrupt is triggered.

Public and Private Registers

The HSSL module contains two types of registers:

- public registers, which can be both read and written by the HSSL module itself
- private registers, which can be only read by the HSSL module itself

This behavior of the HSSL module registers is implemented by using the User Mode (U) and Supervisor Mode (SV) of writing registers. The private registers are writable in SV mode, the public registers are writable in U and SV mode.

The memory access protection registers **AWSTARTi (i=0-3)**, **AWENDi (i=0-3)** and **AR** including the **TIDADD** register are all writable only in Supervisor Mode, and are consequently private. The HSSL itself makes only User Mode accesses, so that it can not write these registers.

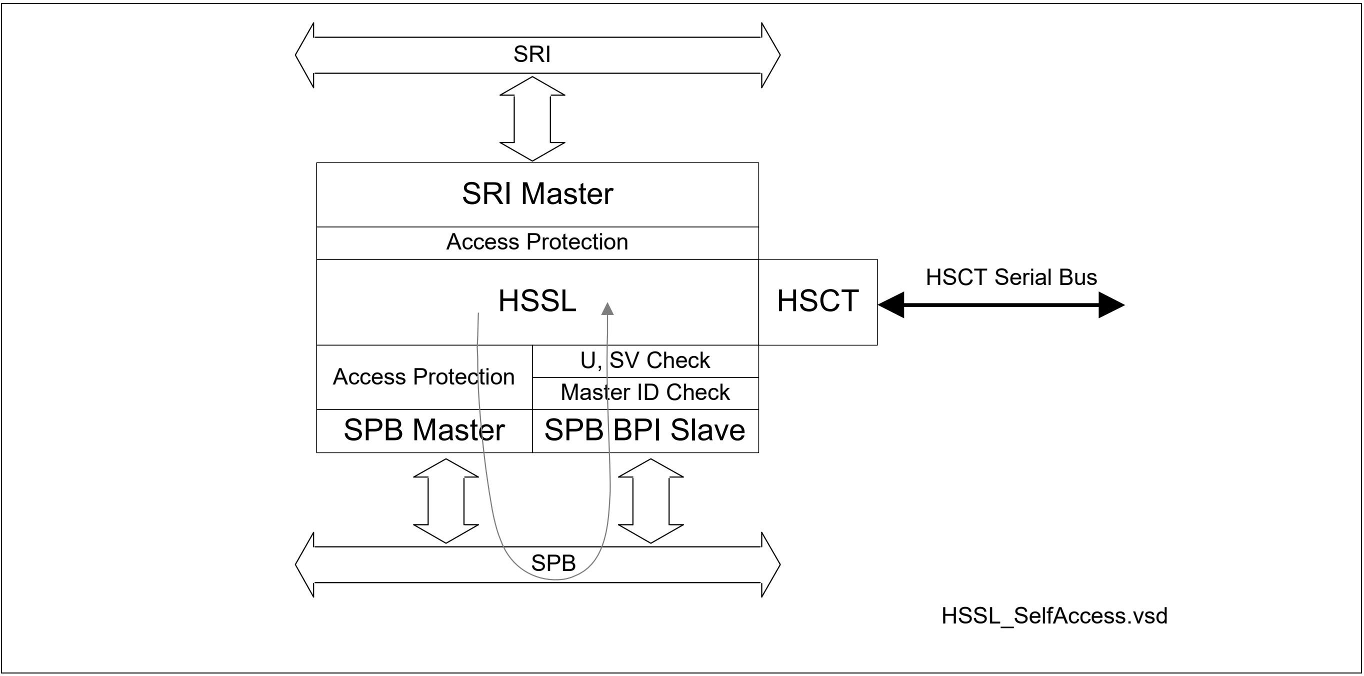


Figure 391 HSSL access to its own registers

35.3.12 Multi-Slave Operation

In multi slave use-cases, one HSSL master can communicate with up to three slaves. The master generates the reference clock from the HSCT module point of view and selects the active counterpart for the connection. The slaves receive the reference clock and are being selected by the master one at a time.

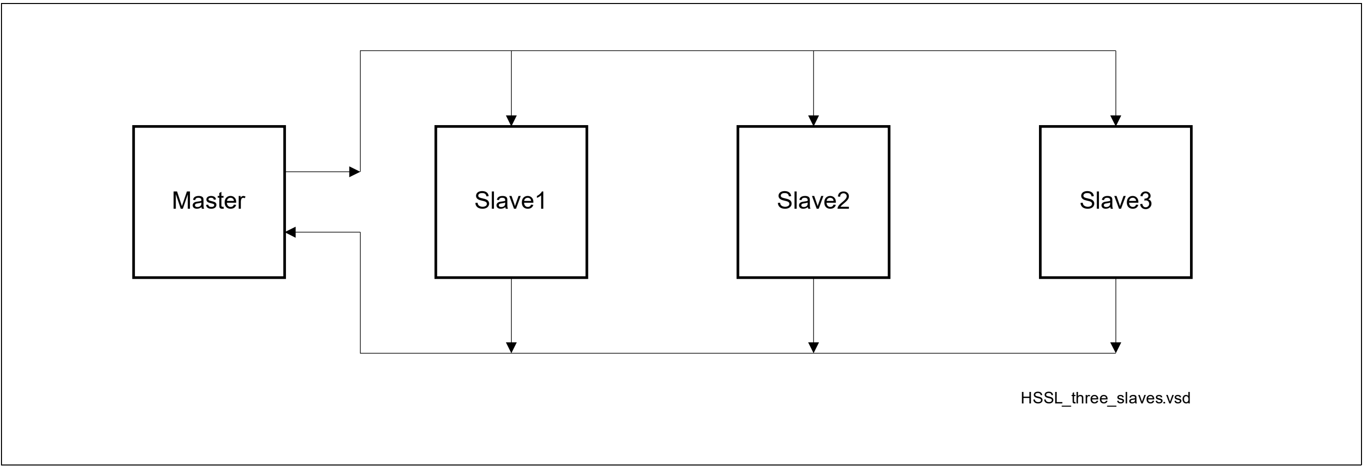


Figure 392 Three Slave HSSL Connection

The communication is connection based - at any point in time there can be only one active connection (or zero) between the master and one slave. There are no broadcast or multicast messages on the HSSL level. There is a defined procedure for terminating a connection with one slave and activating a connection with another slave. During an active connection both master and slave can be initiators and targets.

After power-on reset, the transmit outputs of the slaves and the transmit output of the master are high impedance. Up to three slaves can be enabled for reception by software.

35.3.12.1 Slave Tag and Slave Control

The multi slave operation uses the identical set of command and stream frames as the standard HSSL two-device operation. The slaves are selected with two bits in the HSSL header. These bits are used to filter the frames which are relevant for a slave by comparing them with the slave internal two bit tags, see **MSCR**.SLAVETAG.

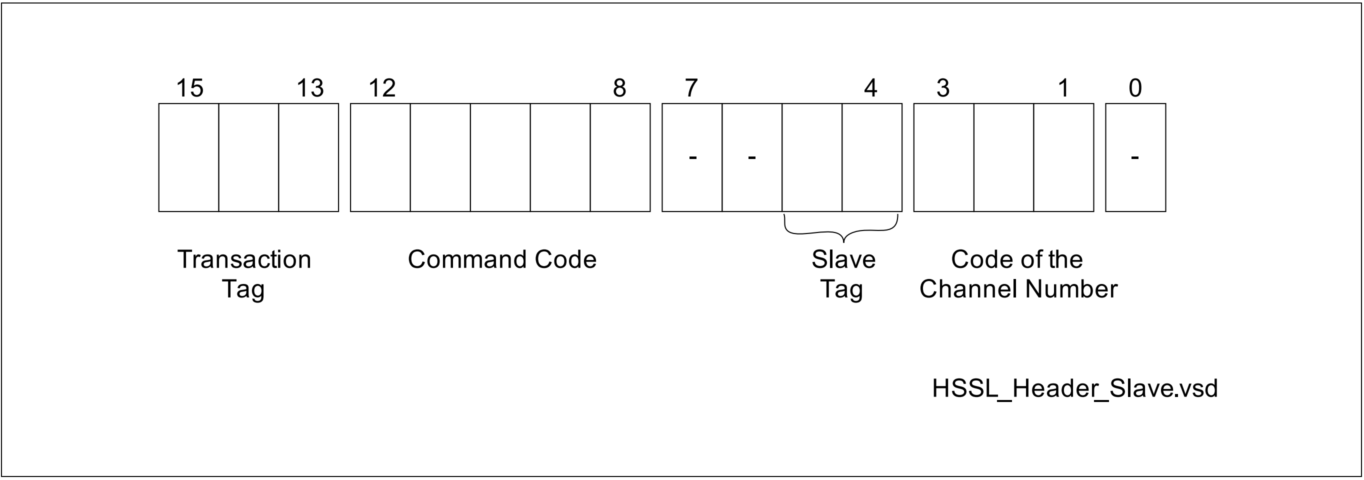


Figure 393 HSSL Header with Slave Tag

35.3.12.2 Slave Tag Frame Filter and Translator

The filter is located after the CRC block. In receive direction, if there is a match, the frame is passed through and handled in a standard HSSL two-device-connection way. If no match, the frames are discarded. In transmit direction, the headers of transmitted frames get the appropriate tag inserted. In case of a CRC error in a frame sent by the master, all slaves may detect the CRC error. The CRC checker discards the frames with CRC error, and even the addressed slave does not get the frame with a CRC error delivered. Frames with PIE1 and PIE2 error are also discarded. The corresponding error interrupts can be triggered if enabled.

The slaves have their tags fixed in the initialization phase and they never change. The software of the master changes its slave tag when establishing connection with different slave. First the HSCT link must be established. The master must have its **MSCR**.SLAVETAG configured before starting with sending HSSL commands or stream frames to the appropriate slave.

The **MSCR**.EN and **MSCR**.SLAVETAG bit fields are propagated to the HSCT module in order to ensure consistent operation of the whole communication channel.

If the multi slave operation is disabled (**MSCR**.EN=0), then both the injection of the slave tag in transmit direction and the filtering of the slave frames in the receive direction are not performed. The header remains unchanged and all the incoming frames pass through.

It is not intended to set the **MSCR**.ITXSTOP bit in the master.

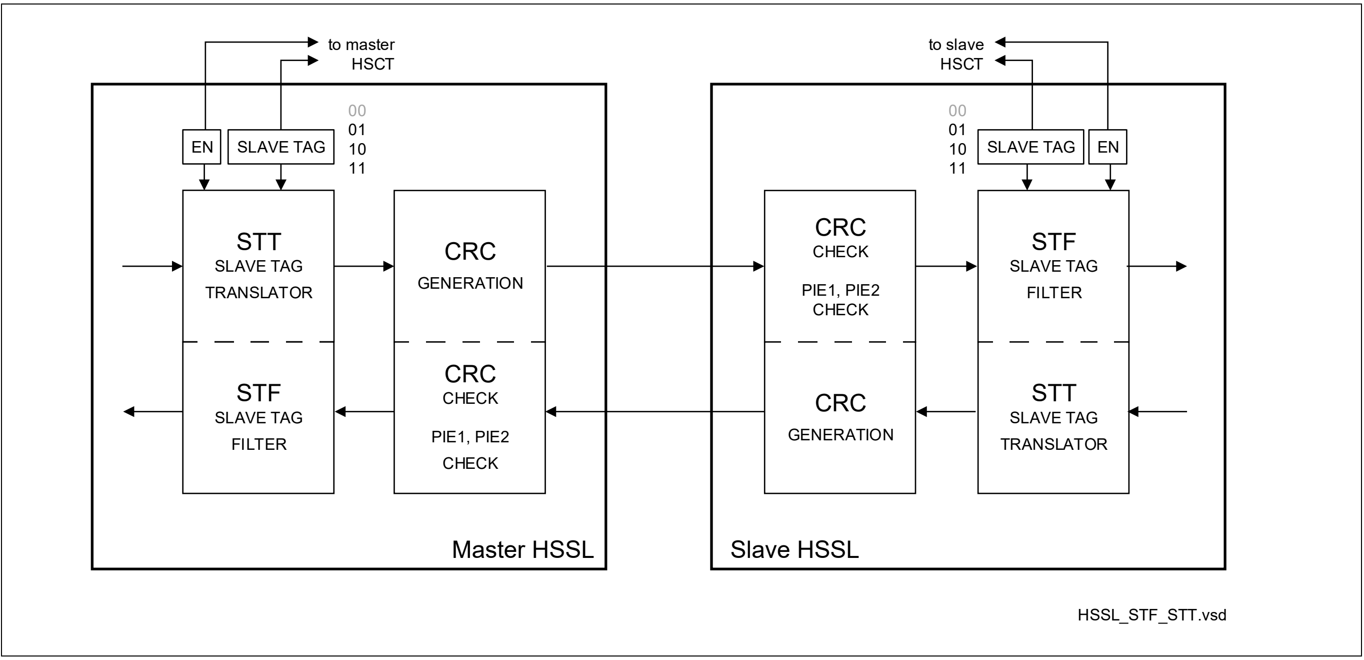


Figure 394 Slave Frame Filter

35.3.12.3 Activating a Slave

Activating a slave is done in two steps:

- The master enables the LVDS transmitter pads of one slave by using slave specific HSCT command. The Tx enable HSCT commands for the slaves 1, 2 and 3 have payloads 51_H, 52_H and 53_H correspondingly.
- The second step is either soft or hard HSSL activation:
  - Soft activation - the master notifies the slave by using standard HSSL commands, according to a pre-defined procedure. For example, master can issue a write command to a certain location, send an interrupt trigger command frame, and a slave having a CPU can start transmitting using the HSSL module.

  - Hard activation - the master clears the **MSCR**.ITXSTOP bit by using standard HSSL 16-bit write command frame (an appropriate address window for the HSSL registers must have been already set up). In the next step, the master notifies the slave, for example with a trigger command, or enables the operation of the slave by setting an appropriate bit(s). The slave continues the operation from the point its initiator transmission has been stopped.

35.3.12.4 Deactivating a Slave

Master can deactivate a slave in two ways, soft and hard.

Soft Deactivation

Soft deactivation procedure uses software handshake procedure to bring the slave in a state where all ongoing slave initiator transactions has been completed, including streams, no new slave transactions are initiated and the slave notifies the master that the safe deactivation state has been reached. Subsequently, the master disables the LVDS pad.

The soft deactivation sequence consists of:

- Soft HSSL deactivation - using software handshake with standard HSSL commands, like writing to memory location and triggering interrupts
- Switching off the LVDS Tx pads with slave unspecific (broadcast) HSCT command with payload 32_H

Hard Deactivation

Hard deactivation is intended to provide faster deactivation with bounded reaction time, primarily for slaves without CPU and use-cases without streaming.

The hard deactivation sequence consists of:

- Hard HSSL deactivation
  - the master sets the **MSCR**.ITXSTOP bit by using standard HSSL 16-bit write command frame (an appropriate address window for the HSSL registers must have been already set up)
  - The master must assure that slave initiator commands have been processed by itself and that the acknowledge frames have been sent to the slave. For example, the master polls the slave E (expect) flags by using standard HSSL 16-bit read commands until it reads back 0.
  - Additionally, the master must assure that its command frames have been processed and acknowledged by the slave. For example, the master can poll its own expect flags until it reads 0.
- Switching off the LVDS Tx pads with slave unspecific (broadcast) HSCT command with payload 32_H

The slave HSSL module initiator transmit path remains inactive and later, after being re-activated by the master, continues the operation from the point where it has been stopped. The master is not allowed to send commands to a deactivated slave.

35.3.12.5 MSCR HSSL to HSCT Connections

The HSSL module provides the **MSCR**.EN and **MSCR**.SLAVETAG bit fields at the module border as outputs. These signals are connected to the HSCT module and provide single point of configuring the behavior of both HSSL and HSCT regarding multi slave behavior.

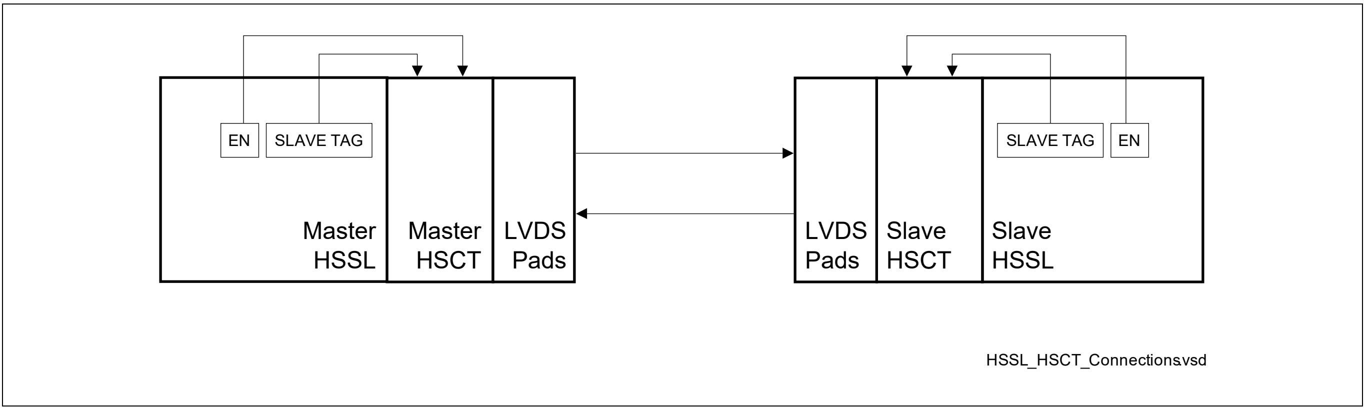


Figure 395 HSSL to HSCT Connections