psi_multi_stream_daq

Documentation

Content

Table of Contents

Figures

1 Introduction

The purpose of the *psi_multi_stream_daq* is to implement one data acquisition component that fulfills all common requirements for recording data of multiple streams in an external memory.

The bus-interfaces are properly separated, so the same logic can be used for different bus systems. Currently the component is implemented for AXI4 (standard) and tosca2 (IOXOS specific bus system).

1.1 Feature List

- Separate clock domain for each stream
- Separate clock domains for configuration interface and memory interface
- Trigger signal handled per stream
- Up to 32 windows per stream (i.e. up to 32 trigger events can be recorded without overwriting existing recordings)
- Buffer configuration at runtime through software (address of the buffer, buffer size, etc.)
- Optional protection of recorded data (data is only overwritten after software acknowledged that the data was read)
- Buffers can be used linearly or as ring-buffers
- 64-bit internal datapath, bandwidth is 8-byte per clock cycle
- Each stream is configurable separately
    - Width (16, 32 or 64 bits)
    - Input buffer depth
    - Priority (3 different levels)
- Configurable burst size for memory access
    - Maximum burst size (to not exceed limitations of the bus-system or memory controller)
    - Minimum burst size (to not waste memory performance by doing many small transfers)
- 32-bit address range for external memory (4 GB)

1.2 Memory Organization

All data recorded is written into the external memory. There is a separate memory region reserved for each stream (called *buffer*). Each buffer is split into multiple *windows* that hold data for individual trigger events.

Buffer and window sizes are configurable per stream. The same applies for the buffer base-address but not to the window base-addresses (windows are placed back to back in the memory).

The buffer organization is shown in the figure below.

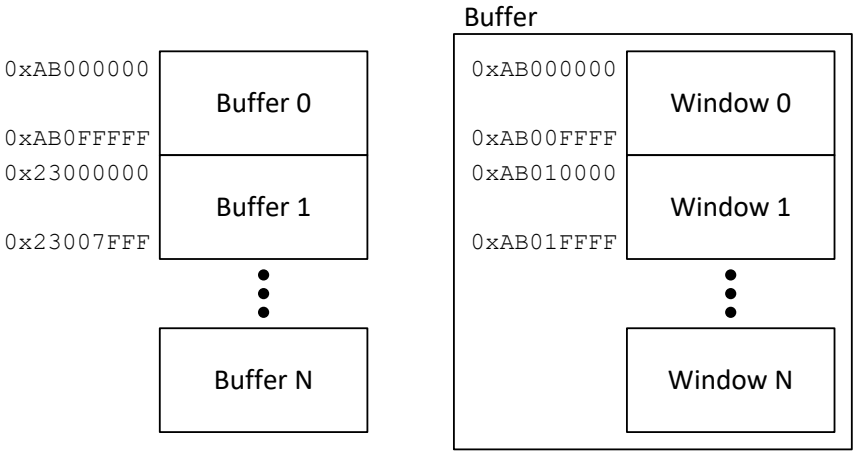

Figure 1: Memory organization

Each window can be regarded as ring-buffer into which data is written continuously. After a trigger, the amount of post-trigger data is written and afterwards the next window (the next ring-buffer) is selected and the same procedure starts again.

Since every window is a ring-buffer, the first sample may not be stored at the first address of the window but the data may be wrapped as shown in the figure below. The software is responsible for unwrapping the data according to the address of the last sample, which can be read via the register bank. The software is also responsible for detecting which sample is the *trigger-sample* based on the last sample address and the configured amount of post-trigger data to record.

Note that each window must have the size of an integer number of samples.

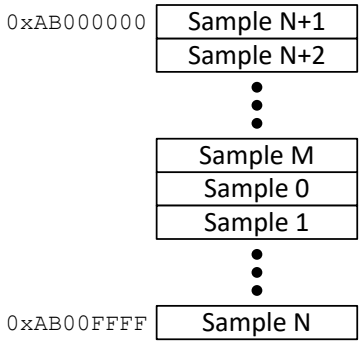

Figure 2: Data in window buffers may be wrapped

Optionally it is possible to disable the ring-buffer behavior and store data linearly. In this case, the recording for one window is finished and the next window is selected if either a trigger occurs or the window is full.

2 Architecture

2.1 Overview

The figure below gives a rough overview over the architecture.

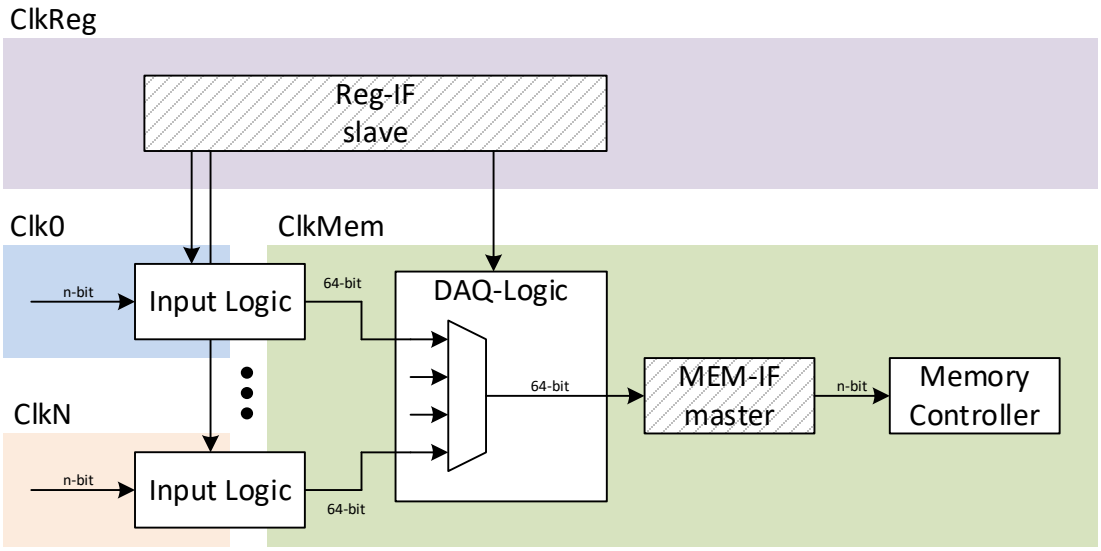

Figure 3: Architecture overview

For each stream there is some separate input logic required that does width conversion, trigger handling, clock-crossing and buffering of the data.

The acquisition logic is responsible for writing the data received to the correct memory addresses in an efficient way.

The memory interface master is implemented in a separate entity to allow interfacing with different bus standards such as Tosca-SMEM or AXI4.

To sustain high bandwidths, all logic must be designed in a way that the memory interface can be kept busy all the time if required. This means that data transfers have to be executed in parallel to the decisions about the next data transfers.

Back-pressure is handled through the whole chain, so the READY handshaking at the stream input goes low if not all data can be written to the memory. This is crucial since not handling back-pressure could lead to undetected loss of data.

The diagram also shows the clock-domains. All recording logic is running on clock domain of the memory interface to avoid unnecessary overhead due to clock-crossings in the main datapath. Since this is the main clock domain, it is also called “acquisition clock domain” in this document.

2.2 Input Logic

The input logic is implemented per stream. Its main responsibility is to buffer the input data and make it available to the acquisition logic in the correct format (64-bit wide, acquisition clock domain).

It also does the framing of the data. After the configured number of post-trigger samples was recorded, it ends the recording frame by asserting TLAST (end of frame signal according to AXI-S specification). Additionally the input logic ends a frame when no samples arrive for a configured timeout to ensure all data is written to the memory eventually. The reason for ending a frame (timeout or trigger) is sent to the acquisition logic together with the frame data.

For each trigger input, the timestamp is sampled ad stored in a FIFO. The sampling of the timestamp happens in the source clock domain to ensure jitter-free operation. A FIFO is required since data for multiple trigger events may be stored in the data FIFO. The resource usage of this concept is acceptable because the timestamp FIFO is shallow and can therefore be realized in distributed RAMs.

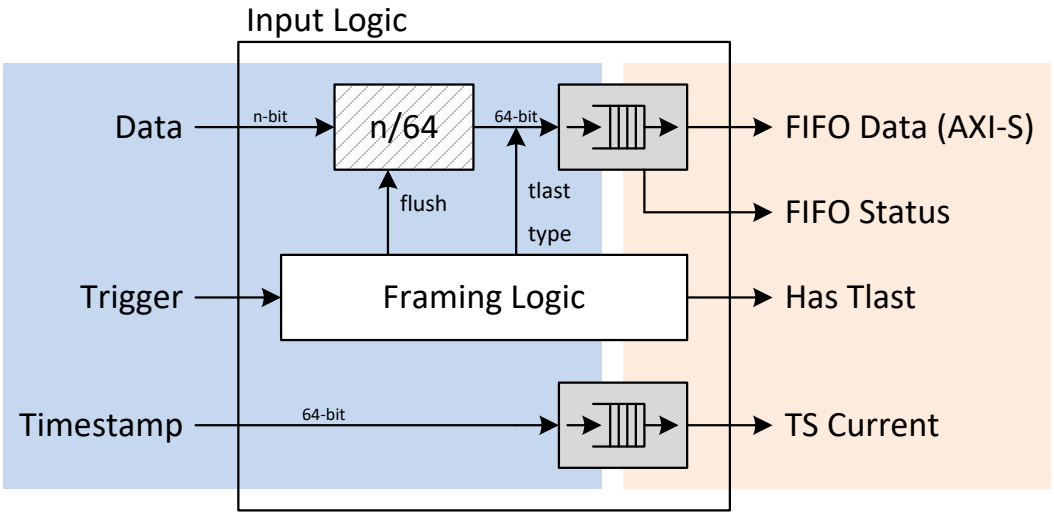

Figure 4: Input logic

Since the end of the frame may not be aligned to a 64-bit word, some data may be stuck in the width conversion. To prevent this, the width conversion is flushed at the end of a recording. This leads to not all data in the last 64-bit word being valid. The information about what parts of a 64-bit word are valid is transferred to the acquisition logic together with the data.

The FIFO status (fill level) is made available to the acquisition logic, so it can transfer the data to the external memory whenever enough data for an efficient burst is available.

Additionally the acquisition logic can see if there is a TLAST (end of frame) in the FIFO. In this case the data must be written to the external memory independently of the fill level to make it available to the software that reads it. It is possible that triggers arrive quickly one after the other, so there may be multiple TLAST signals in the FIFO. Therefore the framing logic counts how many TLAST signals are in the FIFO and only de-asserts “Has Tlast” only after all frames are read from the FIFO.

Note that triggers arriving during the recording of post-trigger data of the last trigger event are ignored.

If a trigger pulse arrives between two samples, the sample after the trigger pulse is regarded as “trigger sample”.

2.3 Acquisition Logic

The main goal of the acquisition logic is to write data to the external memory as efficiently as possible. The bandwidth shall only be limited by Tosca, so data transfers must happen back-to-back and no time must be lost when taking the decision what data to transfer next. This requirement is accounted for by separating the control state machine that decides about what data to transfer next from the DMA logic that actually executes the data transfers.

With this setup, the control state machine can be kept simple because it is allowed to take a few clock cycles to decide about the next data transfer. During that time, the DMA logic stays running and data keeps flowing.

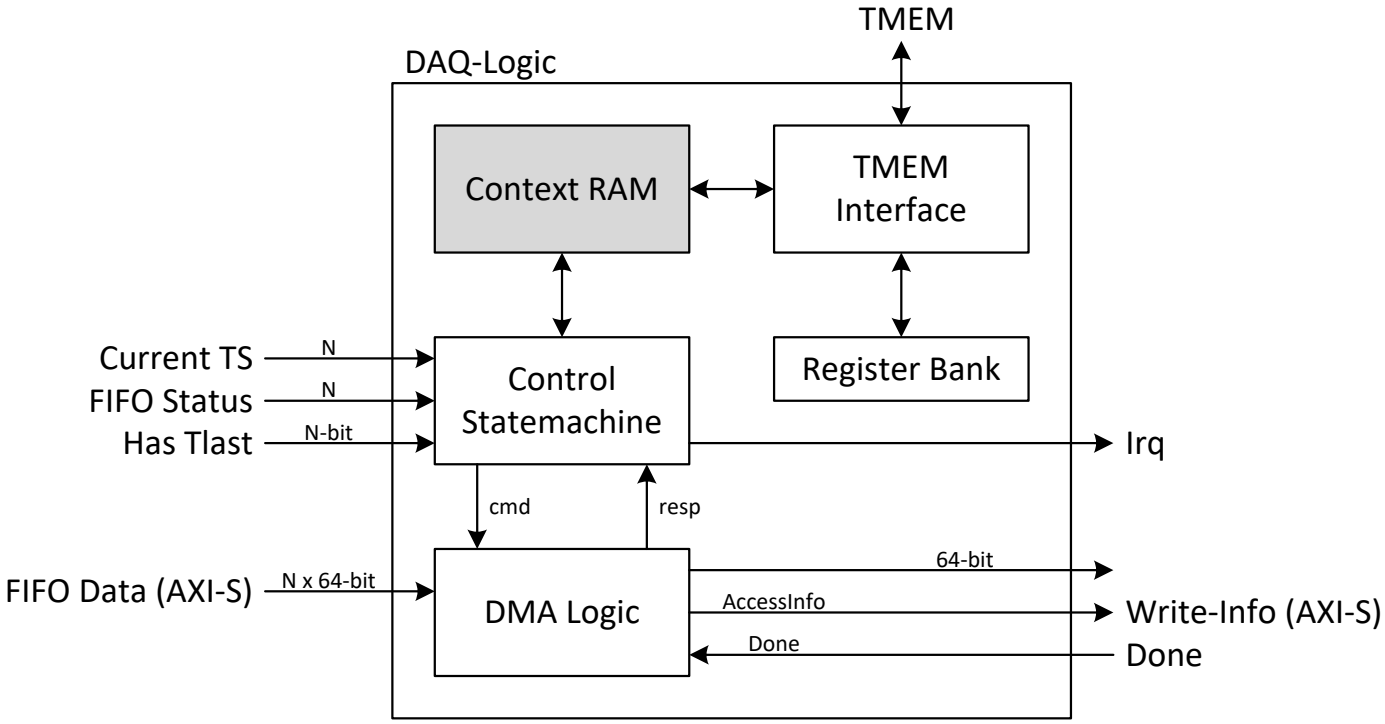

Figure 5: Acquisition logic

2.3.1 Context RAM

For every data stream the buffer configuration and status must be stored. This requires many bits and it would therefore not be efficient to implement this storage in registers. Not only because it just requires a lot of flip-flops, but also because many LUTs would be wasted for multiplexing all these registers. Therefore all that information is stored in a context RAM. This is a dual-port RAM. One port is accessed by the control state machine to obtain the current write pointer and the configuration of the buffer to use for the next transfer. The other port is accessible over the register bank to configure the buffers and read information such as the location of the last sample of a frame in the buffer.

2.3.2 Control State Machine

The figure below shows the control state-machine. Note that the figure only depicts the concept but may not contain all substates that are required for implementation reasons.

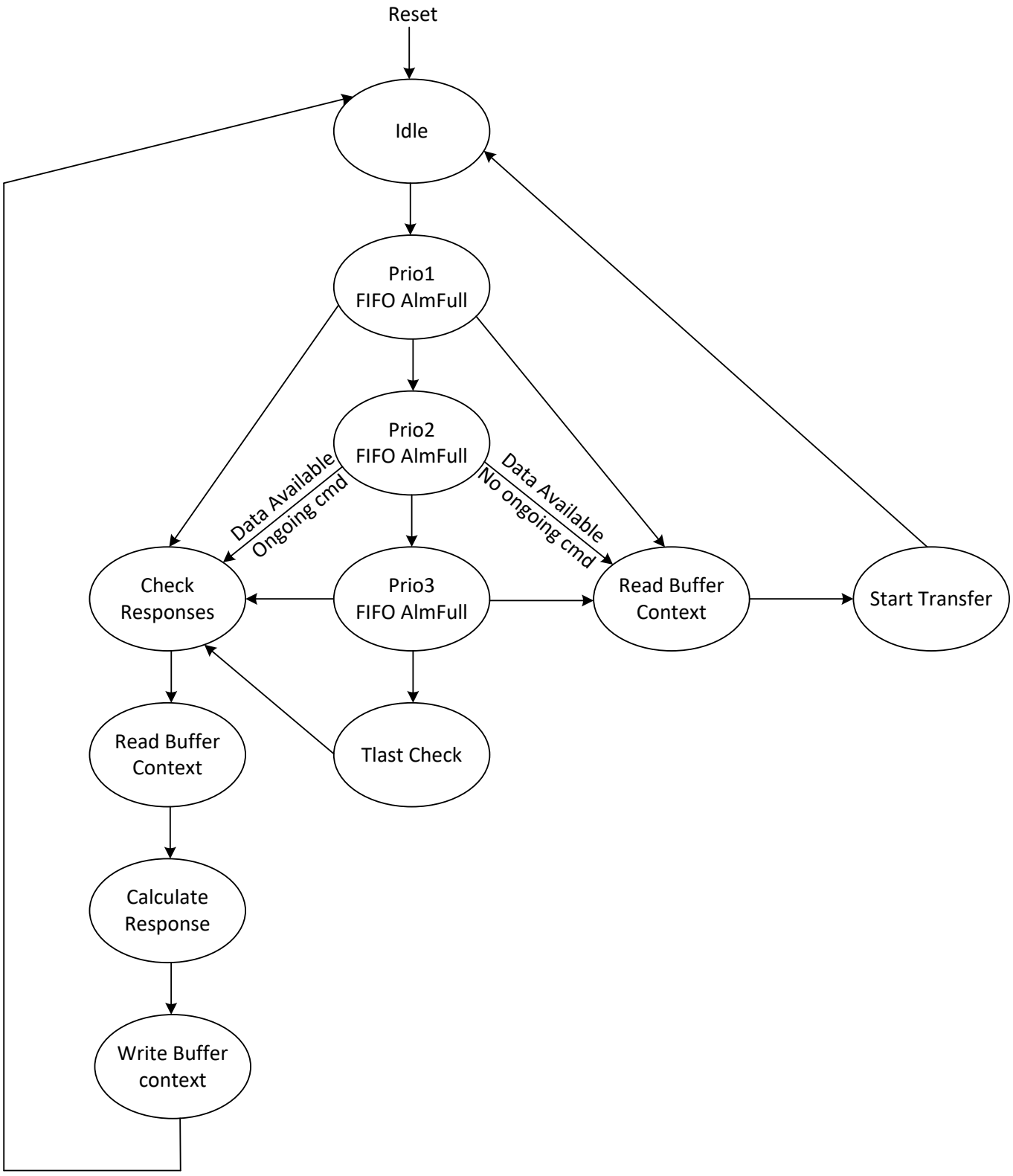

Figure 6: DAQ control state machine

The state machine checks whether enough data for the minimum configured burst size is available from any FIFO. This check is done for all three priorities, one after the other. This way it is ensured that the streams with highest priority get the access first if data is available.

The state machine ensures that only one DMA command per stream is started by masking streams with ongoing commands before arbitrating. Since the state machine runs faster than the commands are executed, this also leads to each stream getting a fair share of the bandwidth: If data on all streams of a given priority is available, for each stream one command is scheduled before waiting for the responses.

If data on a higher priority stream is available, lower priority streams are not checked, even if no command can be issued on the high priority stream. This is required since the state machine would otherwise usually just always schedule one command per stream, independently of the priority since the state machine operates way faster than the commands can be executed.

If no buffer has enough data for an efficient burst, the state machine checks if any stream completed a recording (FIFO contains TLAST).

The state machine is also responsible for ensuring that all data transfers are aligned correctly. This includes not requesting transfers that go over 4k boundaries or window boundaries.

If a data transfer has to be executed, the context for the corresponding stream is read from the context RAM and the data transfer is started by sending the corresponding stream to the DMA logic. The command contains the following information:

- Start address of the transfer
- Maximum transfer size in bytes (limited by maximum burst length or end-address of the buffer)
- Stream number

At the end of a transfer, the DMA logic passes its response back to the control state machine. The response contains the following information:

- Stream number
- Actual transfer size in bytes
- Information about whether a the recording of this window was completed in the transfer (TLAST + type=end of recording)

If there are pending responses from the DMA logic, the state machine updates the context memory accordingly before it starts the next transfer. Since it is crucial that the context memory is updated before the next transfer on the same stream is started, the state machine can only start transfers on different streams simultaneously. As a result, the full memory bandwidth cannot be achieved for single-stream systems.

The state machine also fires interrupts whenever the recording of one window is completed and all related data is written to the memory. For implementation reasons, the order of the DMA response and the “Done” signal for the memory interface is not known. Therefore some synchronization logic is implemented that ensures that IRQs are only fired if the response from both sides is received.

2.3.3 DMA Logic

The DMA Logic is responsible for transferring the data for a given stream to the bus interface that will write it into the memory. It contains a small command FIFO that can store multiple commands to allow executing transfers back-to-back. Another small FIFO contains the responses. Both FIFOs support one entry per stream, so they are small and can be implemented in distributed memory.

The control logic counts the exact number of bytes transferred because this information is required for the response.

The actual length of the transfer is detected by the DMA logic since the transfer must be stopped whenever the stream FIFO is empty or a TLAST arrives (in this case the next sample is potentially transferred to another window buffer).

The state machine may request transfers that are not 64-bit aligned. This can for example happen if a timeout occurred that cause all data to be written to the memory. If the size of “all data” is not a multiple of 64-bits, the next write is not 64-bit aligned. As a result data alignment logic is required.

Another case that increases the complexity of the DMA logic is that the size of the full transfer is not necessarily a multiple of 64-bits. This situation usually occurs at window boundaries, either because the boundaries are not 64-bit aligned or because the transfers went out of alignment because of timeouts. In this situation some bytes of the last QWORD must be saved for the next transfer. For this purpose the remaining data is stored in the *Remaining RAM*.

To allow for frame-based operation the timeout in the input stage can be configured to be active only between the start of a frame transmission and the corresponding TLAST. After a TLAST, the timeout counter is cleared and turned off until the next data sample is received. Furthermore, the timeout can be disabled completely.

One important corner case occurs when the last few bytes of a window finished by a trigger are stored in the *Remaining RAM.* In this case the input logic does not see the TLAST anymore (it was already extracted from the FIFO) but the data is also not yet in the memory. To handle this case, the DMA logic has an output that says if any data related to a TLAST is stored in the remaining RAM, so the state machine can schedule another transfer quickly.

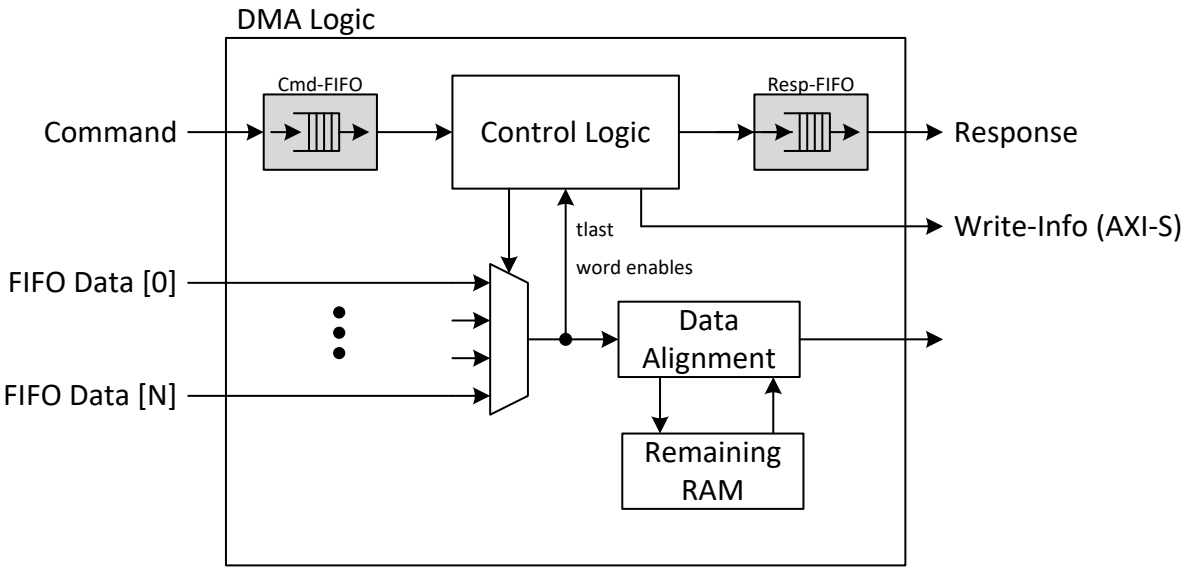

Figure 7: DMA logic

2.4 Memory Interface Master

The memory interface master implements write transactions only because no read operations are required for the data-recorder.

Two interface types are implemented:

- Tosca2 (IOXOS specific)
- AXI-4 (Industry Standard)

The figure below shows the architecture of the memory interface master.

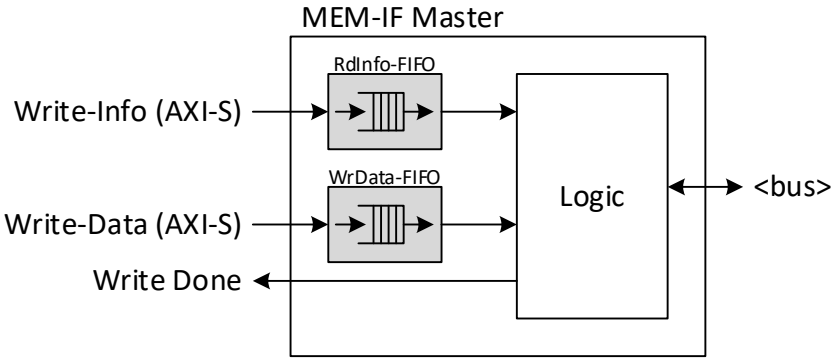

Figure 8: Memory interface master

Write commands are stored in a small FIFO with one entry per stream, so operations of all streams can be pending at the same time.

For tosca SMEM: Since Tosca SMEM does not implement any throttling, write transfers are only started when all data of the full transfer is in the write data FIFO. Only in this situation it is clear that data can be sent to Tosca in one single burst. For AXI-4, the implementation is the same for simplicity reasons, even if throttling would be available for AXI-4.

Completion of a transfer is signaled using the done port. In the context of the *psi_multi_stream_daq* this information will be used to trigger interrupts.

Because of timeouts, it can happen that the write data is not 64-bit aligned but the most bus standards require all access to be aligned (in this case to 64-bits). Therefore the memory interface master ensures 64-bit alignment by shifting the write data accordingly if the bus requires alignment.

3 Interfaces

3.1 Generics

3.1.1 Common Generics

Streams_g	**integer**	Number of data streams
StreamWidth_g	**integer array**	Width of each stream (must be 64, 32 or 16)
StreamPrio_g	**integer array**	Priority of each stream (1..3, 1 is highest)
StreamBuffer_g	**integer array**	Buffer depth per stream (in samples at input width).
StreamTimeout_g	**real array**	Timeout for each stream (see 2.2) in seconds
StreamClkFreq_g	**real array**	Clock frequency for each stream in Hz
StreamTsFifoDepth_g	**integer array**	Depth of the timestamp FIFO per stream Should be set to the maximum number of triggers expected during *StreamBuffer_g[N]* input samples.
StreamUseTs_g	**boolean array**	For each stream the timestamp logic can be enabled or disabled in order to save resources. If disabled, the timestamp is always 0xFF…F
MaxWindows_g	**integer**	Maximum number of windows per stream (applies for all streams, range is 0..32)
MinBurstSize_g	**integer**	Minimum burst size on the Memory interface (in words of the memory interface, i.e. 64-bit words for Tosca, *AxiDataWidth_g*-words for AXI). For good bandwidth results, this value should be >= 64
MaxBurstSize_g	**integer**	Maximum burst size on the memory interface (in words of the memory interface, i.e. 64-bit words for Tosca, *AxiDataWidth_g*-words for AXI). For Tosca this value must be <= 512

3.1.2 Generics for AXI-4 Version only

AxiDataWidth_g	**integer**	Width of the AXI master interface to the memory in bits
AxiMaxBurstBeats_g	**integer**	Maximum beats in one AXI burst (usually 256, 16 for AXI-3)
AxiMaxOpenTransactions_g	**integer**	Maximum pending transactions on the AXI bus
AxiFifoDepth_g	**integer**	Depth of the AXI-side buffer for the memory interface (in AXI words)

3.1.3 Generics for Tosca2 Version only

There are no tosca2 specific generics.

3.1.4 Generic Configuration Guide

Check the list of considerations below to choose generics properly:

- If *MinBurstSize_g* is chosen to the same value as *MaxBurstSize_g*, the core always writes bursts of the same size except for flushing the data out after a trigger. This results in bursts being 4kB aligned after the first crossing of a 4kB boundary. If *MinBurstSize_g/MaxBurstSize_g* are chosen to 4kB (or an integer part of it), no bursts have to be split at 4kB boundaries, so bandwidth is optimally utilized.
- Setting *MinBurstSize_g* and *MaxBurstSize_g* to different values generally leads to more transfers begin unaligned to 4kB boundaries and hence transfers have to be split into multiple parts more often. This results in slightly more overhead on the memory bus.
- Setting *MinBurstSize_g* low can lead to high overhead since many small transfers are executed over the memory bus.
- The *psi_multi_stream_daq* only writes data to the memory if either *MinBurstSize_g* words are buffered in the input buffer or a trigger arrives. As a result, the *StreamBuffer_g* setting must be at least *MinBurstSize_g*. Otherwise the core will never write data because even a full buffer contains too little data for the smallest allowed burst.
- Choosing *MaxBurstSize_g* higher than the maximum AXI burst allowed by *AxiMaxBurstBeats_g* does not help much with bandwidth. It also does not harm but there is not much benefit from it.
- *AxiFifoDepth_g* should be chosen at least 2x *AxiMaxBurstBeats_g* in order to have space available in the FIFO while a burst of maximal size is pending but access to the bus is not yet possible.
- When using different *StreamPrio_g*, be aware that no low-prio stream can write any data to the memory until all pending data of high priority streams is written. If *MinBurstSize_g* is chosen low, high-priority streams may have data pending all the time.

3.2 Ports

3.2.1 Common Ports

Signal	Direction	Width	Description
Data Streams			
Str_Clk	Input	Streams_g	Clock (one clock per stream)
Str_Data	Input	Streams_g x 64	Stream data. For streams with less than 64 bit width, only the bits [W-1:0] are used and all other bits can be left unconnected. Handshaking via *StrVld* and *StrRdy*
Str_Ts	Input	Streams_g x 32	Timestamp input (sampled on *Str_Trig* = '1')
Str_Vld	Input	Streams_g	AXI-S handshaking signal (per stream)
Str_Rdy	Input	Streams_g	AXI-S handshaking signal (per stream)
Str_Trig	Input	Streams_g	Trigger signal (per stream). The trigger does not necessarily have to be aligned with *StrData* (i.e. it can occur independently of the handshaking). In this case the next sample arriving after the trigger is regarded as "trigger sample".
Miscellaneous			
Irq	Output	1	Interrupt output (leven sensitive, high active) Synchronous to *Tmem_Clk*

3.2.2 Ports for AXI-4 Version only

Register Interface			
S_Axi_Aclk	Input	1	Register interface clock
S_Axi_Aresetn	Input	1	Register interface reset (low active)
S_Axi_*	*	*	AXI signals, see AXI specification
Memory Interface			
M_Axi_Aclk	Input	1	Memory interface clock
M_Axi_Aresetn	Input	1	Memory interface reset (low active)
M_Axi_*	*	*	AXI signals, see AXI specification

3.2.3 Ports for Tosca2 Version only

Register Interface			
Tmem_Clk	Input	1	TMEM interface clock
Tmem_Rst	Input	1	TMEM interface reset
AcqTmem.PIPE*	Output	2	TMEM read latency indicator
AcqTmem.BUSY*	Output	1	TMEM throttling Not working according to Patric Bucher, so it will not be used.
TmemAcq.ADD*	Input	24	TMEM byte address (64-bit aligned)
TmemAcq.DATW*	Input	64	TMEM write data
AcqTmem.DATR*	Output	64	TMEM read data
TmemAcq.ENA*	Input	1	TMEM enable
TmemAcq.WE*	Input	8	TMEM byte enable
Memory Interface			
Smem_Clk	Input	1	SMEM interface clock
Smem_Rst	Input	1	SMEM interface reset
AcqSmem.WREQ*	Output	2	SMEM write request
SmemAcq.WACK*	Input	2	SMEM write acknowledge
AcqSmem.WSIZ*	Output	10	SMEM write transfer size in bytes
AcqSmem.WADD*	Output	32	SMEM write address (byte address)
AcqSmem.WDAT*	Output	64	SMEM write data
AcqSmem.WBE*	Output	8	SMEM write byte enable
AcqSmem.WCCMD*	Output	2	SMEM write cache control
AcqSmem.WCTAG*	Output	32	SMEM write cache tag
AcqSmem.RREQ*	Output	2	SMEM read request
SmemAcq.RACK*	Input	2	SMEM read acknowledge
AcqSmem.RSIZ*	Output	10	SMEM read size in bytes
AcqSmem.RADD*	Output	32	SMEM read address (byte address)
SmemAcq.RDAT*	Input	64	SMEM read data

* This is not an individual port but a signal within a record

3.3 Address Map

The address map is preliminary and may change during development.

The following conventions are used for register field descriptions:

R Read only (do not modify content!)
W Write only
RW Read/Write
RCW1 Read, clear by writing '1'
SPCL Special handling (other than mentioned above)

3.3.1 Overview

All registers are 32-bit wide.

Byte Address	Name	Description
0x000000	ACQCONF	Acquisition configuration registers
0x001000 + 0x20 * N	CTXMEMn	Context memory for stream N
0x004000 + 0x10 * W * N	WNDWn	Window memory for stream N

N = stream number
W = window number

3.3.2 ACQCONF – Acquisition Configuration

3.3.2.1 Overview

Byte Address Offset	Name	Description
0x000	GCFG	General configuration register
0x004	GSTAT	General status register
0x010	IRQVEC	Interrupt vector register
0x014	IRQENA	Interrupt enable register
0x020	STRENA	Data stream enable register
-	-	Reserved
0x200 + 0x10 * N	MAXLVLn	Maximum FIFO level for stream N
0x204 + 0x10 * N	POSTTRIGn	Post-trigger samples for stream N
0x208 + 0x10 * N	MODEn	Operation mode of stream N
0x20C + 0x10 * N	LASTWINn	Last window written to memory for stream N

N = stream number

3.3.2.2 GCFG – General Configuration Register (0x000)

Field	Bit(s)	Type	Reset	Description
ENA	0	RW	0	1 Data acquisition is enabled 0 Data acquisition is disabled
IRQENA	8	RW	0	1 Interrupts are enabled 0 Interrupts are disabled

If IRQENA is zero, interrupts are still detected (i.e. IRQVEC is set) but the interrupt output is not asserted.

3.3.2.3 GSTAT – General Status Register (0x004)

Field	Bit(s)	Type	Reset	Description
TBD	-	-	-	-

3.3.2.4 IRQVEC – Interrupt Vector Register (0x010)

Field	Bit(s)	Type	Reset	Description
IRQVECn	31:0	RCW1	0	Interrupt flag for each stream 1 Interrupts is pending for the related stream 0 No interrupts is pending for the related stream

The corresponding bit in the *IRQVEC* is set whenever the recording of a window is completed. Usually this is the case when a trigger occurred but depending on other settings, it can also happen if one window is full (*SCFG[RINGBUF] = '0'*).

3.3.2.5 IRQENA – Interrupt Enable Register (0x014)

Field	Bit(s)	Type	Reset	Description
IRQENAn	31:0	RW	0	Interrupt enable for each stream. 1 Interrupts are enabled for a given stream 0 Interrupts are disabled for a given stream

If IRQENAn is zero, interrupts are still detected (i.e. IRQVEC is set) but the interrupt output is not asserted.

3.3.2.6 STRENA – Data Stream Enable Register (0x020)

Field	Bit(s)	Type	Reset	Description
STRENAn	31:0	RW	0	Enable for each stream. 1 Data stream is enabled (data is recorded) 0 Data stream is disabled

If a data stream is disabled, the input FIFO is cleared automatically to ensure no old data is persisting in the FIFO after it is re-enabled.

Disabling a stream also disables the assertion of the related bit in *IRQVEC*. As a result, the last IRQ of a stream disabled in full operation may be lost. This is not critical since the arrival of this IRQ is a race condition anyway.

3.3.2.7 MAXLVLn – Maximum FIFO level for stream N (0x200 + 0x10*N)

Field	Bit(s)	Type	Reset	Description
LVL	31:0	SPCL	0	Maximum level of the input FIFO for the corresponding stream. The maximum level register can be cleared by writing to it.

The maximum FIFO level is obtained to allow users to check if there was potentially a loss of data (if the FIFO was full).

It also can be used to measure how much margin exists: If the maximum FIFO level is close to full, the margin is small and the buffer size may have to be changed to improve system stability. If the maximum FIFO level is close to empty, not the full buffer is used and the buffer size could possibly be reduced to save resources.

3.3.2.8 POSTTRIGn – Post-trigger samples for stream N (0x204 + 0x10*N)

Field	Bit(s)	Type	Reset	Description
POSTTRIG	31:0	RW	0	Number of post-trigger samples to record for the corresponding stream.

The trigger sample itself is not regarded as post-trigger sample. So if *POSTTRIG* is set to 7, the trigger sample plus 7 post trigger samples is recorded.

3.3.2.9 MODEn – Operation mode of stream N (0x208 + 0x10*N)

Field	Bit(s)	Type	Reset	Description
RECM	1:0	RW	0	Recording mode 0 Continuously record data 1 Trigger Mask Mode Pre-trigger is always recorded but a trigger event is detected only once after ARM is set. 2 Single Shot Mode Pre-trigger recording is started after ARM is set. Then one trigger event is detected and the recording is stopped to not use bandwidth for the continuous recording of pre-trigger 3 Manual Mode After ARM is set, “POST-TRIGGER+1” samples are recorded immediately and the recording is stopped. The recorder does not wait for the external trigger event.
ARM	8	RW	0	This bit is only used for single shot mode. After a trigger was received, this bit is automatically cleared. The ARM bit is high during pre-trigger recording only. 1 Trigger detection is armed 0 Trigger detection is not armed or trigger already occured
REC	16	R	0	This bit shows if the recorder is currently recording data. This is especially interesting in single-shot mode to see if a recording is still ongoing or already finished. The REC bit is high during pre- and post-trigger recording. 1 Data is being recorded 0 No data being recorded, waiting for arming
TODE	24	RW	0	Disable timeout in input stage
FRAMETO	25	RW	0	Enable frame based timeout

The figures below show the behavior in different recording modes.

Continuous (0)

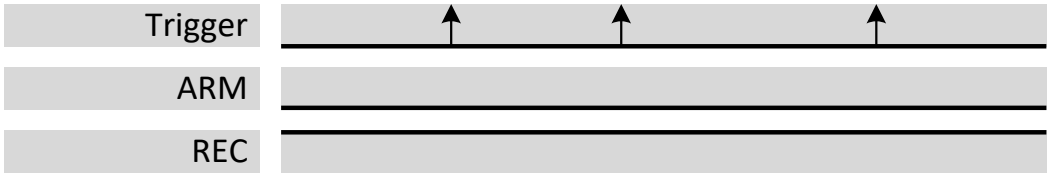

Figure 9: Continuous Recording Mode

In continuous recording mode, each trigger is detected (except it occurs during the post-trigger recording of the last trigger). The *ARM* signal is not used and therefore always zero, independently of what is written to it. Since pre-trigger data is always recording, the *REC* bit is always high.

Trigger Mask (1)

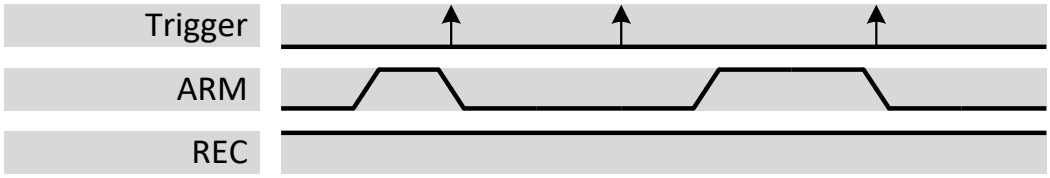

Figure 10: Trigger-Mask Recording Mode

In trigger mask mode only one trigger is detected after arming the recorder. The *ARM* bit shows whether the trigger already occurred or not (on trigger it is reset). Pre-trigger data is always recorded in order to be able to detect a trigger (and have all pre-trigger data recorded) after arming instantaneously. As a result, the *REC* flag is always set.

Single Shot (2)

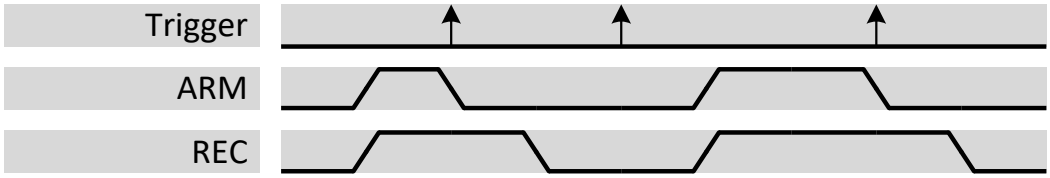

Figure 11: Single Shot Recording Mode

Single shot mode works similar to trigger mask mode described above. The only difference is that pre-trigger data recording is started on arming and recording is stopped after all post-trigger data is recorded. As a result, a stream does not use bandwidth if no recording is ongoing. The drawback is that no pre-trigger data is available if a trigger occurs directly after arming the recorder.

The *REC* flag shows whether data is currently recorded. Note that the flag corresponds to the state of the input engine. So a falling edge of *REC* does not mean that all data is already written to the memory but only that all data is recorded (but possibly still in some internal buffers).

Manual (3)

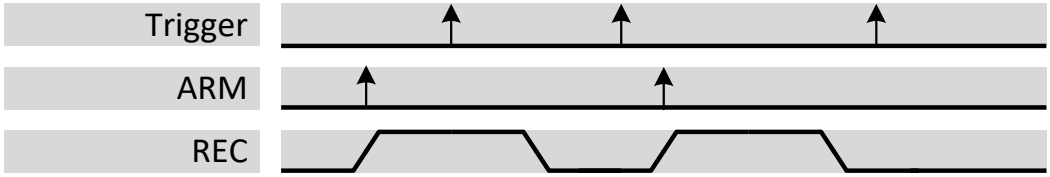

Figure 12: Manual Recording Mode

In manual mode, a recording is executed after writing a one to *ARM*. Data is only written during the time a recording is really ongoing. In this mode, the trigger input does not have any effect.

3.3.2.10 LASTWINn – Last window written to memory for stream N (0x20C + 0x10*N)

Field	Bit(s)	Type	Reset	Description
LASTWIN	4:0	R	0	Window number of the last window written to memory completely

The *CTXMEMn* values can be used to determine the current state of the recorder and to what window it is currently writing. However, this does not mean that the data of the last window was already written to memory completely. The write command is set-up but it could be delayed for example because the bus towards the memory is not yet accessible or because the memory is busy with some refresh operations.

The *LASTWINn* register allows finding out if the write of all data of a window fully completed. This register is only updated when the writing of the data to the memory is acknowledged.

3.3.3 CTXMEMn – Context Memory for Stream N

3.3.3.1 Overview

Byte Address Offset	Name	Description
0x000	SCFG	Stream configuration register
0x004	BUFSTART	Buffer start address
0x008	WINSIZE	Window size
0x00C	PTR	Next memory location to write
0x010	WINEND	End address of the currently active window
-	-	Reserved

3.3.3.2 SCFG – Stream Configuration Register (0x000)

Field	Bit(s)	Type	Reset	Description
RINGBUF	0	RW	0	1 Each window is handled as ring-buffer and only trigger events lead to switching to the next window. 0 Windows are handled as linear buffers. If a window is fully written, switching to the next window happens automatically (regardless of trigger events)
OVERWRITE	8	RW	0	1 Windows that contain data (WINCNTw != 0) are overwritten if new data arrives. 0 Windows that contain data (WINCNTw != 0) are not overwritten. If new data arrives but the next window to use contains data, recording is blocked until the window is cleared.
WINCNT	20:16	RW	0	Number of windows to use – 1 (0 → 1 Window, 13 → 14 Windows)
WINCUR	28:24	R	0	Window number that is currently in use

Modifying this register while the recording for the corresponding stream is running is not allowed.

If *OVERWRITE = '0'*, the field *WINCNTw* must be cleared by software whenever the data was read to mark the window as free for the next recording.

Note that this register contains the state of the recorder state machine that defines calculates the next write command. This does not mean that all pending commands are yet completed (i.e not all data may yet be written to memory). To find out if all data of a window was written to the memory, use the *LASTWINn* register.

3.3.3.3 BUFSTART – Buffer Start Address (0x004)

Field	Bit(s)	Type	Reset	Description
ADDR	31:0	RW	0	Start address of the buffer for the corresponding stream (byte address)

Modifying this register while the recording for the corresponding stream is running is not allowed.

3.3.3.4 Window Size (0x008)

Field	Bit(s)	Type	Reset	Description
SIZE	31:0	RW	0	Size of the windows for the corresponding stream (in bytes). The size of a window must always be an integer number of samples. Other values can lead to undefined behavior.

Modifying this register while the recording for the corresponding stream is running is not allowed.

3.3.3.5 PTR – Next Memory Location to write (0x00C)

Field	Bit(s)	Type	Reset	Description
PTR	31:0	R	0	Current address pointer

Note that this register contains the state of the recorder state machine that defines calculates the next write command. This does not mean that all pending commands are yet completed (i.e not all data may yet be written to memory).

3.3.3.6 WINEND – End address of the currently active window (0x010)

Field	Bit(s)	Type	Reset	Description
ADDR	31:0	R	0	Endaddress of the current window + 1

This register shall not be accessed by software. It does not have any meaning in terms of software. It only exists for firmware implementation reasons.

3.3.4 WNDWn – Window Memory for Stream N

3.3.4.1 Overview

Byte Address Offset	Name	Description
0x000 + SO*N + 0x10*W	WINCNTw	Number of samples in window W of stream N
0x004 + SO*N + 0x10*W	WINLASTw	Address of the last sample written to window W of stream N
0x008 + SO*N + 0x10*W	WINTSLOw	Lower 32-bits of the timestamp of the trigger for window W of Stream N
0x00C + SO*N + 0x10*W	WINTSHIw	Higher 32-bits of the timestamp of the trigger for window W of Stream N

W = window number
SO = Stream offset, see below

$$SO = 2^{ceil\left(log2(WindowsPerStream)\right)} * 0x10$$

The window memory is separated from the rest of the context memory since its size can strongly vary depending on the stream count and number of windows per stream. To foresee it in the address map of the

general context RAM could lead to big gaps in the memory map of the context RAM and therefore result in very inefficient resource usage, especially for low window counts.

3.3.4.2 WINCNTw – Number of samples in window W

Field	Bit(s)	Type	Reset	Description
CNT	30:0	RW	0	Number of valid data samples (not bytes) in Window
ISTRIG	31	R	0	This flag says whether a window was finished by a trigger or not. 1 Window finished by trigger 0 Window exited because full

To acknowledge that the data of a window was read by the software, the WINCNTw register bust be set to zero.

3.3.4.3 WINLASTw – Address of the last sample written to window W

Field	Bit(s)	Type	Reset	Description
LAST	31:0	R	0	Address of the last sample that was written into window W. This address is required to unwrap data in for SCFG[RINGBUF]=1.

3.3.4.4 WINTSLOw – Timestamp of the trigger for window W [31:0]

Field	Bit(s)	Type	Reset	Description
TSLO	31:0	R	0	Bits [31:0] of the timestamp of the trigger that belongs to window W. In case of overflows of the timestamp FIFO, the timestamp 0xFF..F is used. The same applies for stream that do not have the timestamping enabled.

3.3.4.5 WINTSHIw – Timestamp of the trigger for window W [63:32]

Field	Bit(s)	Type	Reset	Description
TSHI	31:0	R	0	Bits [63:32] of the timestamp of the trigger that belongs to window W. In case of overflows of the timestamp FIFO, the timestamp 0xFF..F is used. The same applies for stream that do not have the timestamping enabled.

4 Implementation Details

4.1 Clock Crossings

The figure below roughly shows the setup of the clock domains and the clock domain crossings (*CDC*) in between them.

All configuration and status data that must be passed between the register interface clock domain and the clock domains of individual data streams is clock-crossed inside the *input logic* entity. As a result, the clocks of the individual streams are only required inside this entity which eases understanding as well as physical implementation.

For the same reason the clock crossing of the main data stream to the memory interface clock domain is done inside the input logic.

The maximum level of the input FIFO is detected inside the register interface. This detection is done on the source clock domain (memory interface) to avoid peaks being missed because of CDC effects.

The context memory is a dual-port RAM with ports running on register interface and memory interface clocks, so it serves as clock-crossing on itself.

Configuration data for the acquisition state machine (enable signals for individual streams) and the IRQ are clock-crossed inside the register interface.

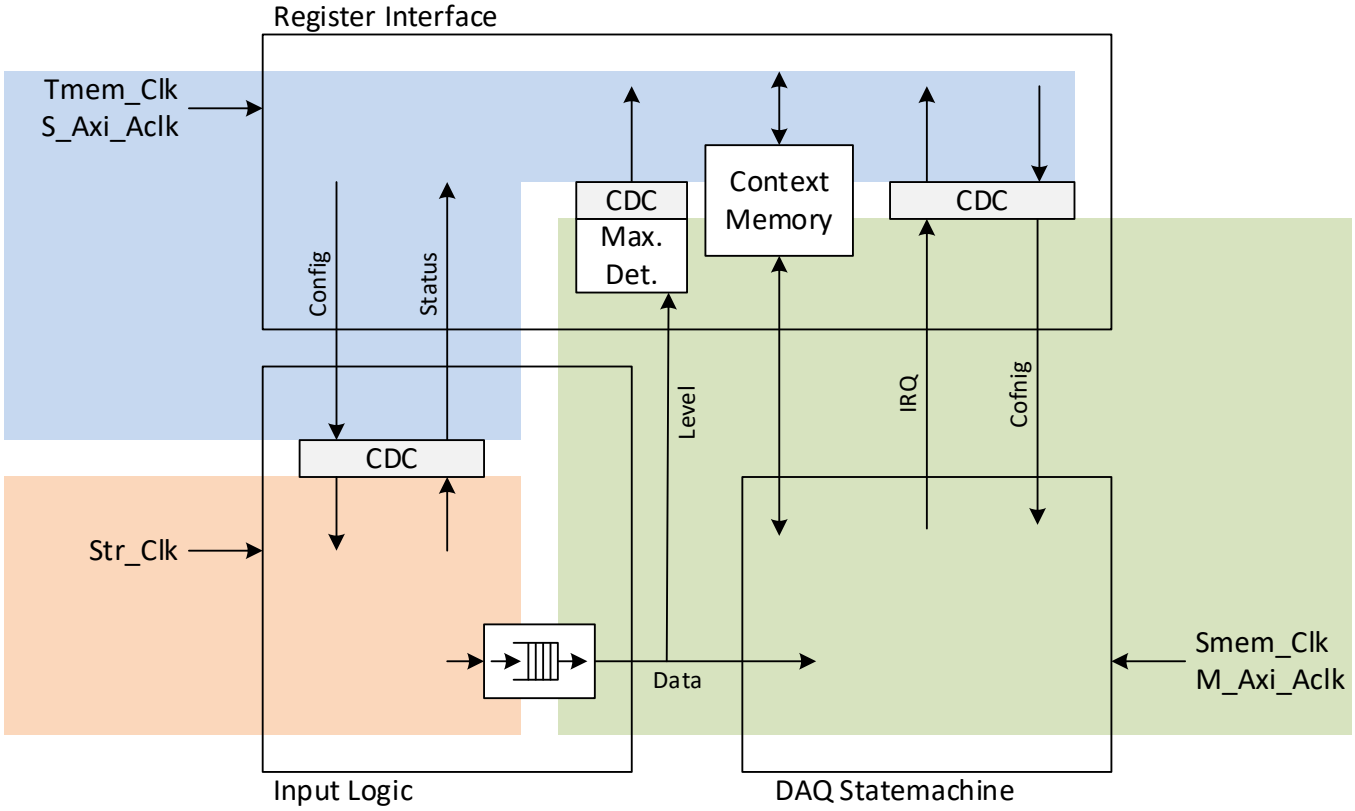

Figure 13: Clock Crossings