ClockSignal Timestamper

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.6
Date	03.01.2023

Copyright Notice

Disclaimer

The information available to you in this document/code may contain errors and is subject to periods of interruption. While NetTimeLogic GmbH does its best to maintain the information it offers in the document/code, it cannot be held responsible for any errors, defects, lost profits, or other consequential damages arising from the use of this document/code.

NETTIMELOGIC GMBH PROVIDES THE INFORMATION, SERVICES AND PRODUCTS AVAILABLE IN THIS DOCUMENT/CODE "AS IS," WITH NO WARRANTIES WHATSOEVER. ALL EXPRESS WARRANTIES AND ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS ARE HEREBY DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT SHALL NETTIMELOGIC GMBH BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL AND EXEMPLARY DAMAGES, OR ANY DAMAGES WHATSOEVER, ARISING FROM THE USE OR PERFORMANCE OF THIS DOCUMENT/CODE OR FROM ANY INFORMATION, SERVICES OR PRODUCTS PROVIDED THROUGH THIS DOCUMENT/CODE, EVEN IF NETTIMELOGIC GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF YOU ARE DISSATISFIED WITH THIS DOCUMENT/CODE, OR ANY PORTION THEREOF, YOUR EXCLUSIVE REMEDY SHALL BE TO CEASE USING THE DOCUMENT/CODE.

Overview

NetTimeLogic's Signal Timestamper is a full hardware (FPGA) only implementation of a Signal Timestamper. It allows to timestamp data aligned with an event signal of configurable polarity. Timestamps are taken on the configured edge of the signal and optional interrupts are generated. The Signal Timestamper is intended to be connected to a CPU or any other AXI master that can read out the timestamps. Optionally a timestamp buffer can be enabled to handle burst situations.
The settings can be configured either by signals or by an AXI4Lite-Slave Register interface.

Key Features:

- Signal edge timestamping
- 32 bit second and 32 bit nanosecond timestamp
- Configurable polarity
- Input delay compensation
- Data snapshot for timestamp alignment
- Optional timestamp and Data buffering for burst handling
- Interrupt generation
- Interrupt masking
- Edge counter for event detection
- Maximum event rate depends on CPU and AXI bus load
- AXI4Lite register set or static configuration
- Timestamp resolution with 50 MHz system clock: 10ns
- Optional High Resolution Timestamping with 4ns resolution
- Linux Driver

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	16.11.2016	First draft
1.0	18.11.2016	First release
1.1	12.09.2017	Timestamp buffer added
1.2	15.09.2017	Status register added
1.3	30.11.2017	Changed registerset, added cable delay
1.4	09.03.2018	Added Driver
1.5	25.02.2020	HighResolution added
1.6	03.01.2023	Added Vivado upgrade version description

Table 1: Revision History

Content

Definitions

Definitions	
Counter Clock	A counter based clock that count in the period of its frequency in nanoseconds
PI Servo Loop	Proportional–Integral servo loop, allows for smooth corrections
Offset	Phase difference between clocks
Drift	Frequency difference between clocks

Table 2: Definitions

Abbreviations

Abbreviations	
AXI	AMBA4 Specification (Stream and Memory Mapped)
IRQ	Interrupt, Signaling to e.g. a CPU
PPS	Pulse Per Second
TS	Timestamp
CLK	Clock
CC	Counter Clock
TB	Testbench
LUT	Look Up Table
FF	Flip Flop
RAM	Random Access Memory
ROM	Read Only Memory
FPGA	Field Programmable Gate Array
VHDL	Hardware description Language for FPGA's

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

The Signal Timestamper is meant as a co-processor handling event and timestamping them.
It takes a (synchronized) time input as reference, an event signal and optional data to be timestamped and generates reference clock timestamps and data snapshots on the configurable edge (polarity) of the event signal compensating the input delay. A timestamp event will also cause an interrupt to signal to the CPU that an event occurred and a timestamp is ready to be read. Whenever an edge is detected it will increase an internal counter which allows to detect missed events, since the timestamper disables itself until the CPU has cleared the interrupt.
The Signal Timestamperis designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Lite slave for configuration, status supervision and timestamp readout from a CPU.

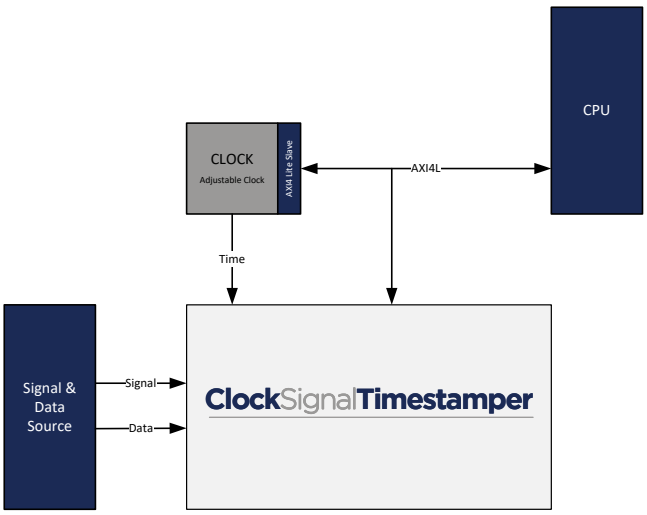

Figure 1: Context Block Diagram

1.2 Function

The Signal Timestamper is, as the name says already, a signal timestamper which takes a snapshot of a reference clock and optional data when an edge of configured polarity is detected.
It uses two inputs: an event signal which defines when an event to timestamp happens and optional a data array that shall be snapshot aligned with the event.

The data is used to correlate the timestamp, so the user application can make actual use of the timestamp. For example this can be measurement values from a sensor, e.g. when a vibration sensor is detecting an abnormal behavior it generates an alarm, this alarm signal is provided to the timestamper together with the detected g-force value. The Signal Timestamper then takes a snapshot of both the time and the g-force value so the user can check when the event happened and what the g-force value was at that point in time. Also a counter together with the g-force value can be provided to detect which occurance of this abnormal event it was.
Internally the Signal Timestamper has an event counter which is incremented whenever an event was detected. This is usefull because the snapshot logic disables itself after an event and waits until it gets enabled by the CPU again by acknowledging the interrupt.
As just stated out, interrupt clearing is reenabling the snapshot logic, so for every detected event an interrupt is created, which shall trigger the CPU to read a timestamp (and data) and acknowledge the completion of the readout. For burst handling, a buffer was introduced which can store timestamps until the CPU has read all timestamps.

1.3 Architecture

The core is split up into different functional blocks for reduction of the complexity, modularity and maximum reuse of blocks. The interfaces between the functional blocks are kept as small as possible for easier understanding of the core.

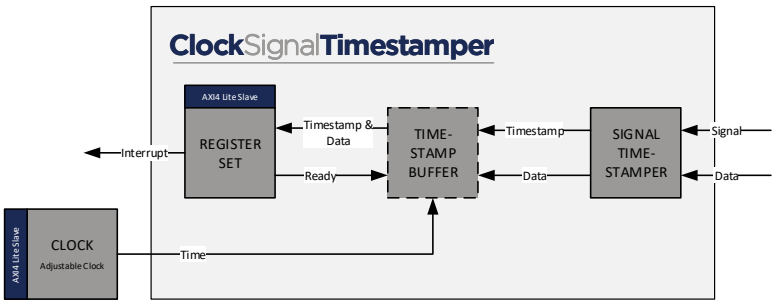

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values, the timestamp and data snapshot and writing configuration. On events generated by the Signal Timestamper block it takes a snapshot of the data

Signal Timestamper

This block is the actual timestamper. It does the edge detection, event generation and time snapshotting of the reference time.

Timestamp Buffer

This optional block buffers timestamps and data coming from the Signal Timestamper to handle bursts.

2 Signal Timestamping Basics

2.1 Digital Counter Clock

A digital counter clock is the most commonly used type of absolute time sources for digital systems. Its functionality is simple: every counter cycle it adds the period of the counter cycle to a counter. Optimally the counter period is an integer number which makes things easier. Normally such a counter clock is split into two counter parts, a sub seconds part and a seconds part, depending on the required resolution the sub second part is in nanoseconds, microseconds or milliseconds or even tens or hundreds of milliseconds. Once the sub seconds counter overflows e.g. 10^9 nanoseconds are reached, the seconds counter is incremented by one and the sub seconds counter is reset to the remainder if there is any.
The highest resolution can be achieved when the counter period is equal the clock period where the counter is run on, this is then normally a nanoseconds resolution, however with a quantization of the clock period.

Figure 3: shows a typical high resolution counter clock with nanosecond resolution and a counter period equal the clock period and a clock of 50MHz which equals to a 20ns clock period.

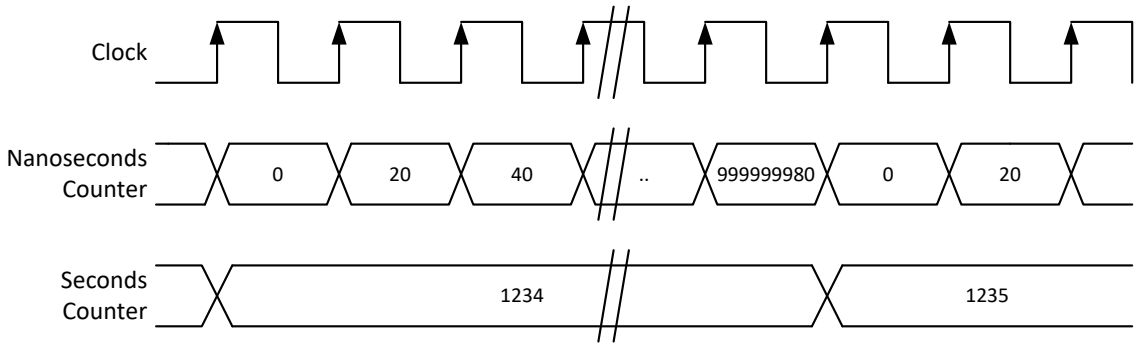

Figure 3: Counter Clock

2.2 Drift and Offset adjustments

When a digital counter clock shall be synchronized there are two things that have to be adjusted which is frequency differences aka drift and phase differences aka offset. Normally the phase difference is only considered the phase within a second. But for absolute time also the correct second is important.

Adjusting a counter clock in a simple way is to keep the clock frequency and adjust the counter increment. This has the advantages that it normally has a much higher resolution e.g. 1ns/s and it doesn't require or relies on external hardware. To adjust drift or offset additional nanoseconds are added or subtracted from the standard increment of the period.
E.g. for a 50 MHz counter clock an offset of +100 ns could be adjusted from one clock cycle to the next: 20 => 140=>160 => … (including 20 ns for the next clock cycle) or it could for example be spread over the next 100 clock cycles: 20 => 41 => 62 =>73 =>… which is a much smoother adjustment. The same applies to the drift which can also be set once in a period or evenly spread over time.
But why is a smooth adjustment important? If for example a PWM signal is generated from the counter clock then you don't want a time jump since the PWM would not be correct anymore, and this is exactly what would happen if the time is not corrected smoothly. The same applies for short time period measurements, these would measure wrong periods because of the adjustments.
However it is not always possible to adjust the time smoothly, e.g. at startup of a system the clock has to be adjusted by thousands of seconds to get to the time of day (TAI start with second 0 at midnight 1.1.1970) or if the adjustment is larger than the possible adjustment in a given period. This cannot be done smoothly in a reasonable time, therefore the time is then set with a time jump.
Also important is that the clock doesn't count backwards during adjustments. Data acquisition and measurement applications require for example a strongly monolithically increasing time. This requirement basically limits the maximal adjustment so the clock is still counting. E.g. at 50 MHz a norm period is 20 ns, the maximum adjustment is therefore +/-19ns per clock period so the clock would still count with 1ns per clock period.
All these mechanisms are implemented in NetTimeLogic's Adjustable Counter Clock core.

When using the counter clock for signal timestamping or signal generation the quantization fault is still the clock period but with an accurate nanosecond resolution.

2.3 Signal Timestamping

Timestamping means taking a snapshot of the time when an event occurred. In this case a rising edge of a signal.
When timestamping, two delays must be considered, one is the delay from the input (or source) to the core and the other is the delay the time it takes to detect an edge. The sum of the two delays must be subtracted from the current time to get back to the moment where the event happened.
Also the frequency and therefore quantization of the clock is important. It in the end limits the resolution and therefore accuracy of the timestamp. When an event happens, the best assumption is to correct to the middle of a period, this will limit the maximum error to half a clock period. To achieve higher accuracy the signal detection can work on both edges of the clock, this will again increase the accuracy by a factor of two. And of course higher frequencies for event detection can be used and fractions taken into account to correct the timestamp.

Fehler! Verweisquelle konnte nicht gefunden werden.Figure 3: shows exactly the delays which are occurring when timestamping.

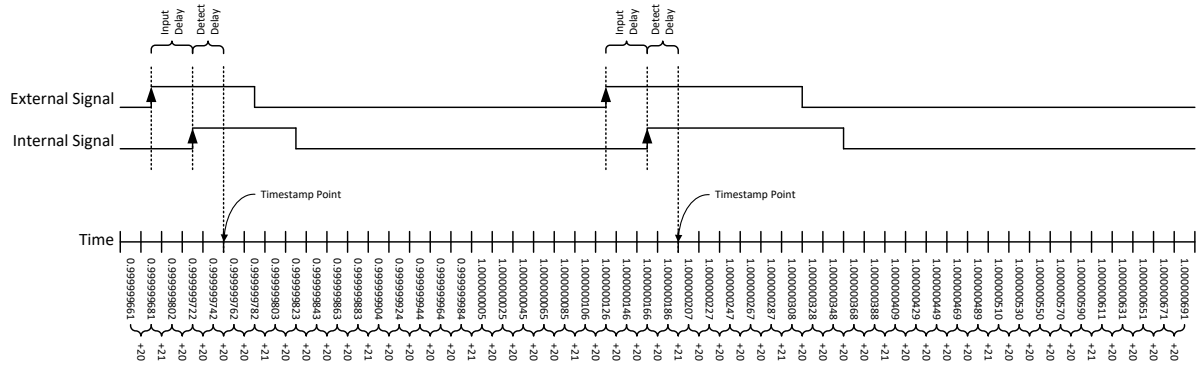

Figure 4: Time Stamping

3 Register Set

This is the register set of the Signal Timestamper. It is accessible via AXI4Lite Memory Mapped. All registers are 32bit wide, no burst access, no unaligned access, no byte enables, no timeouts are supported. Register address space is not contiguous. Register addresses are only offsets in the memory area where the core is mapped in the AXI inter connects. Non existing register access in the mapped memory area is answered with a slave decoding error.

3.1 Register Overview

Registerset Overview			
Name	Description	Offset	Access
Clk StControl Reg	Clock Signal timestamping Read/Write Valid and Enabled Control Register	0x00000000	RW
Clk StStatus Reg	Clock Signal timestamping Status Register	0x00000004	WC
Clk StPolarity Reg	Clock Signal timestamping Polarity Register	0x00000008	RW
Clk StVersion Reg	Clock Signal timestamping Version Register	0x0000000C	RO
Clk StCableDelay Reg	Clock Signal timestamping Cable Delay Register	0x00000020	RW
Clk StIrq Reg	Clock Signal timestamping Interrupt Register	0x00000030	WC
Clk StIrqMask Reg	Clock Signal timestamping Interrupt Mask Register	0x00000034	RW
Clk StEvtCount Reg	Clock Signal timestamping Event Count Register	0x00000038	RO
Clk StCount Reg	Clock Signal timestamping Count Register	0x00000040	RO
Clk StTimeValueL Reg	Clock Signal timestamping Timestamp Nanosecond Register	0x00000044	RO
Clk StTimeValueH Reg	Clock Signal timestamping Timestamp Second Register	0x00000048	RO
Clk StDataWidth Reg	Clock Signal timestamping Data Width Register	0x0000004C	RO
Clk StData Reg	Clock Signal timestamping Data Register(s)	0x00000050+	RO

3.2 Register Descriptions

3.2.1 General

3.2.1.1 CLK Signal Timestamper Control Register

Used for general control over the Signal Timestamper. Since most adjustment values are multi register values, set flags are available to mark validity of the whole value.

Clk StControl Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															ENABLE
RO																															RW
Reset: 0x00000000																															
Offset: 0x0000																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:1	RO
ENABLE	Enable	Bit: 0	RW

3.2.1.2 CLK Signal Timestamper Status Register

Used for status supervision. Shows if a timestamp was dropped in the buffer

Clk StStatus Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															DROP
RO																															WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:1	RO
DROP	Drop	Bit: 0	WC

3.2.1.3 CLK Signal Timestamper Polarity Register

Used for setting the signal input polarity, shall only be done when disabled. Default value is set by the InputPolarity_Gen generic.

Clk StPolarity Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															POLARITY
RO																															RW
Reset: 0x0000000X																															
Offset: 0x0008																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
POLARITY	Signal Polarity (1 active high, 0 active low)	Bit: 0	RW

3.2.1.4 CLK Signal Timestamper Version Register

Version of the IP core, even though is seen as a 32bit value, bits 31 down to 24 represent the major, bits 23 down to 16 the minor and bits 15 down to 0 the build numbers.

Clk StVersion Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VERSION																															
RO																															
Reset: 0xXXXXXXXX																															
Offset: 0x000C																															

Name	Description	Bits	Access
VERSION	Version of the core	Bit: 31:0	RO

3.2.1.5 CLK Signal Timestamper Cable Delay Register

This register allows to compensate for the propagation delay of the cable between the source and sink. To calculate the delay a rule of thumb says around 1ns per 15cm of cable.

Clk StCableDelay Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																CABLE_DELAY															
RO																RW															
Reset: 0x00000000																															
Offset: 0x0020																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:16	RO
CABLE_DELAY	Cable delay in nanoseconds (15cm is around 1ns)	Bit: 15:0	RW

3.2.1.6 CLK Signal Timestamper Interrupt Register

Shows the interrupt state. 1 means interrupt asserted. As long as the interrupt is asserted timestamping is disabled until cleared.

Clk StIrq Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																														IRQ
RO																														WC
Reset: 0x00000000																														
Offset: 0x0030																														

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
IRQ	Interrupt, clearing re-enables timestamping	Bit: 0	WC

3.2.1.7 CLK Signal Timestamper Interrupt Mask Register

Enabled and disable the interrupt generation. 1 means interrupt enabled. As long as the interrupt is disabled timestamping is disabled.

Clk StIrqMask Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															IRQ_MASK
RO																															RW
Reset: 0x00000000																															
Offset: 0x0034																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
IRQ	Interrupt Mask, also enable	Bit: 0	RW

3.2.1.8 CLK Signal Timestamper Timestamp Event Count Register

There is an internal counter which counts the number of timestamp events. This counts also if the interrupt is not cleared.

Clk StCount Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_EVT_COUNT																															
RO																															
Reset: 0x00000000																															
Offset: 0x0038																															

Name	Description	Bits	Access
TS_EVT_COUNT	Timestamp counter	Bit: 31:0	RO

3.2.1.9 CLK Signal Timestamper Timestamp Count Register

There is an internal counter which counts the number of timestamps. This counts up monotonic for each timestamp. If the number is not incremented by one, timestamps were missed (the number of missed is new – old -1).

Clk StCount Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_COUNT																															
RO																															
Reset: 0x00000000																															
Offset: 0x0040																															

Name	Description	Bits	Access
TS_COUNT	Timestamp number	Bit: 31:0	RO

3.2.1.10 CLK Signal Timestamper Time Value Low Register

Timestamp of first event nanoseconds part value.

Clk StTimeValueL Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_TIME_NS																															
RO																															
Reset: 0x00000000																															
Offset: 0x0044																															

Name	Description	Bits	Access
TS_TIME_NS	Timestmap in Nanosecond	Bit: 31:0	RO

3.2.1.11 CLK Signal Timestamper Time Value High Register

Timestamp of first event seconds part value.

Clk StTimeValueH Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_TIME_S																															
RO																															
Reset: 0x00000000																															
Offset: 0x0048																															

Name	Description	Bits	Access
TS_TIME_S	Timestamp in Second	Bit: 31:0	RO

3.2.1.12 CLK Signal Timestamper Data Width Registers

Data width in bits of the Data to be snapshot with the event.

Clk StDataWidth Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_DATA_WIDTH																															
RO																															
Reset: 0x00000000																															
Offset: 0x004C																															

Name	Description	Bits	Access
TS_DATA_WIDTH	Timestamp Data Width in bits	Bit: 31:0	RO

3.2.1.13CLK Signal Timestamper Data Registers

Data snapshot with the timestamp. This can be used for aligning the timestamp with data. The data width to be snapshot can be passed via DataWidth_Gen and will create the corresponding registers. Data storage is least significant word (LSW) first and if the data width is not 32bit aligned the higher bits are padded width 0.

Clk StData Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TS_DATA																															
RO																															
Reset: 0x00000000																															
Offset: 0x0050 (and further)																															

Name	Description	Bits	Access
TS_DATA	Timestamp Data for alignment	Bit: 31:0	RO

4 Design Description

The following chapters describe the internals of the Signal Timestamper: starting with the Top Level, which is a collection of subcores, followed by the description of all subcores.

4.1 Top Level – Clk SignalTimestamper

4.1.1.1 Parameters

The core must be parametrized at synthesis time. There are a couple of parameters which define the final behavior and resource usage of the core.

Name	Type	Size	Description
StaticConfig_Gen	boolean	1	If Static Configuration or AXI is used: true = Static, false = AXI
Buffered_Gen	boolean	1	Whether the Timestamp buffer shall be instantiated or not
BufferDepth_Gen	natural	1	How deep the buffer shall be in number of timestamps
DataWidth_Gen	natural	1	How width in bits the data that shall snapshot aligned with the timestamp
CableDelay_Gen	boolean	1	If a cable delay shall be configurable (only needed when connected externaly)
InputDelay Nanosecond_Gen	natural	1	Input delay of the signal from the connector to the input signal.
InputPolarity_Gen	boolean	1	true = high active, false = low active
DoubleEdge Support_Gen	boolean	1	If timestamp detection shall be done on both edges of the clock
HighResSupport_Gen	boolean	1	If a high-resolution clock

			SysClkNx with alignment to SysClk is used
HighResFreq Multiply_Gen	natural	1	Multiplication factor of the high-resolution clock compared to SysClk
AxiAddressRange Low_Gen	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	std_logic_vector	32	AXI Base Address plus Registerset Size Default plus 0xFFFF
Sim_Gen	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis

Table 4: Parameters

4.1.1.2 Structured Types

4.1.1.2.1 Clk_Time_Type

Defined in Clk_Package.h of library ClkLib
Type represents the time used everywhere. For this type overloaded operators + and – with different parameters exist.

Field Name	Type	Size	Description
Second	std_logic_vector	32	Seconds of time
Nanosecond	std_logic_vector	32	Nanoseconds of time
Fraction	std_logic_vector	2	Fraction numerator (mostly not used)
Sign	std_logic	1	Positive or negative time, 1 = negative, 0 = positive.
TimeJump	std_logic	1	Marks when the clock makes a time jump (mostly not used)

Table 5: Clk_Time_Type

4.1.1.2.2 Clk_SignalTimestamperStaticConfig_Type

Defined in Clk_SignalTimestamperAddrPackage.h of library ClkLib

This is the type used for static configuration.

Field Name	Type	Size	Description
Polarity	std_logic	1	'1' = high active, '0' = low active
CableDelay	std_logic_vector	16	Cable Delay in Nanoseconds

Table 6: Clk_SignalTimestamperStaticConfig_Type

4.1.1.2.3 Clk_SignalTimestamperStaticConfigVal_Type

Defined in Clk_SignalTimestamperAddrPackage.h of library ClkLib
This is the type used for valid flags of the static configuration.

Field Name	Type	Size	Description
Enable_Val	std_logic	1	Enables the Timestamper

Table 7: Clk_SignalTimestamperStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

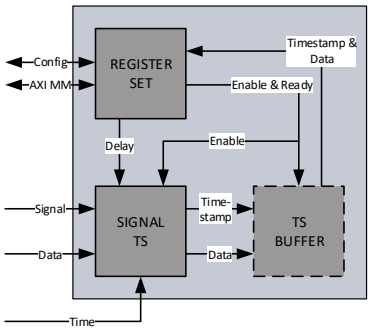

Figure 5: Signal Timestamper

4.1.1.4 Entity Description

Signal Timestamper

This module handles the incoming signal. It detects the configured edge (via the polarity) of the input and generates a timestamp event which is the input for the Register set module to also snapshot the data. When the edge is detected it takes a timestamp of the reference clock and compensates it for the input and detection delay. It also increments a counter which is incremented at every rising edge.
See 4.2.1 for more details.

Timestamp Buffer

This optional module buffers timestamps and data in a FIFO manner to overcome burst situations. It stores and potentially drops timestamps if not eventually read by the CPU via the Register Set. The Registerset signals to this module whenever it is ready for a new timestamp.

See 4.2.2 for more details.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows to configure the Signal Timestamper. It also contains the data snapshot and interrupt logic. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration is done via signals and can be easily done from within the FPGA without a CPU. If in AXI mode, an AXI Master has to configure the core with AXI writes to the registers, which is typically done by a CPU. The Signal Timestamper is only usefull when a CPU or another AXI master with interrupt input is connected.
See 4.2.3 for more details.

4.1.1.5 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
Buffered_Gen	-	boolean	1	Whether the Timestamp buffer shall be instantiated or not
BufferDepth_Gen	-	natural	1	How deep the buffer shall be in number of timestamps
DataWidth_Gen	-	natural	1	How width in bits the data that shall snapshot aligned with the timestamp
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the signal from the connector to the input signal.
InputPolarity_Gen	-	boolean	1	True: High active,

				False: Low active
HighResSupport_Gen	-	boolean	1	If a high-resolution clock SysClkNx with alignment to SysClk is used
HighResFreq Multiply_Gen	-	natural	1	Multiplication factor of the high-resolution clock compared to SysClk
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysClkNx_ClkIn	in	std_logic	1	High-resolution clock (multiple of Sys Clock)
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Clk_SignalTime stamperStatic Config_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_SignalTime stamperStatic ConfigVal_Type	1	Static Configuration valid
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid

AXI4 Lite Slave				
AxiWriteAddrValid_ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady_RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress_AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt_DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid_ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady_RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData_DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe_DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid_ValOut	out	std_logic	1	Write Response Valid
AxiWriteRespReady_RdyIn	in	std_logic	1	Write Response Ready
AxiWriteResp Response_DatOut	out	std_logic_vector	2	Write Response
AxiReadAddrValid_ValIn	in	std_logic	1	Read Address Valid
AxiReadAddrReady_RdyOut	out	std_logic	1	Read Address Ready
AxiReadAddrAddress_AdrIn	in	std_logic_vector	32	Read Address
AxiReadAddrProt_DatIn	in	std_logic_vector	3	Read Address Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadData Response_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response
Interrupt Output				
Irq_EvtOut	out	Clk_Clock StaticConfig_Type	1	Active high level interrupt
Data Input				
Data_DatIn	in	std_logic_vector	Data Width_ Gen	Data to snapshot
Signal Input				
SignalTimestamper_EvtIn	in	std_logic	1	Input signal for

				event detection

Table 8: Signal Timestamper

4.2 Design Parts

The Signal Timestamper core consists of a couple of subcores. Each of the subcores itself consist again of smaller function block. The following chapters describe these subcores and their functionality.

4.2.1 Signal Timestamper

4.2.1.1 Entity Block Diagram

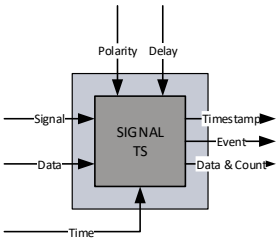

Figure 6: Signal Timestamper

4.2.1.2 Entity Description

Signal Timestamper

This module detects events and takes timestamps. When the Signal Timestamper is enabled it detects edges on the input signal and generates based on the polarity an event on the rising or falling edge. In parallel it takes a snapshot of the reference time and compensates the timestamp for the input- and detection- delay. This timestamp and event is the passed to the Registerset module for further processing.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriodNanosecond_Gen	-	natural	1	Integer Clock Period
CableDelay_Gen	-	boolean	1	If a cable delay shall be configurable (only needed when connected externaly)

InputDelay Nanosecond_Gen	-	natural	1	Input delay of the signal from the connector to the input signal
InputPolarity_Gen	-	boolean	1	True: High active, False: Low active
DataWidth_Gen	-	natural	1	How width in bits the data that shall snapshot aligned with the timestamp
DoubleEdge Support_Gen	-	boolean	1	If timestamping shall be done on both edges
HighResSupport_Gen	-	boolean	1	If a high-resolution clock SysClkNx with alignment to SysClk is used
HighResFreq Multiply_Gen	-	natural	1	Multiplication factor of the high-resolution clock compared to SysClk
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysClkNx_ClkIn	in	std_logic	1	High-resolution clock (multiple of Sys Clock)
SysRstN_RstIn	in	std_logic	1	System Reset
Enable Input				
Enable_EnaIn	in	std_logic	1	If the core is ena-bled or not
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
Signal Values Input				
SignalPolarity_DatIn	in	std_logic	1	'1': High active, '0': Low active

SignalCableDelay_DatIn	in	std_logic_vector	16	Delay in Nanoseconds
Timestamp Signal Input				
ClockTimestamp_EvtIn	in	std_logic	1	Timestamp event
Data Input				
Data_DatIn	in	std_logic_vector	Data Width_ Gen	Data to snapshot
Timestamp Output				
ClockTimestamp_DatOut	out	Clk_Time_Type	1	Corrected Timestamp
ClockTimestamp_ValOut	out	std_logic	1	Corrected Timestamp valid
Data Output				
Data_DatOut	out	std_logic_vector	Data Width_ Gen	Data snapshot
Count_CntOut	out	std_logic_vector	32	Timestamp Count

Table 9: Signal Timestamper

4.2.2 Timestamp Buffer

This is an optional module which can be enabled via the Buffered_Gen generic.

4.2.2.1 Entity Block Diagram

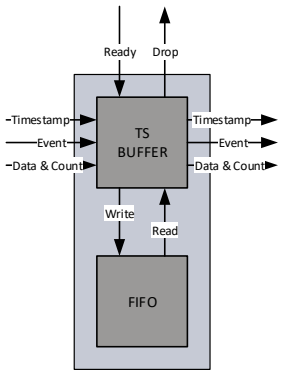

Figure 7: Timestamp Buffer

4.2.2.2 Entity Description

Timestamp Buffer

This module buffers timestamp in a FIFO, it writes the timestamp, count and data in 32bit blocks into the FIFO and reads it the same way from the FIFO. This allows burst handling of quite high frequencies (low MHz range). It waits for propagating the timestamps to the Registerset until this module is ready.

FIFO

This module is the actual storage of the timestamps. It acts as a FIFO with 32bit width.

4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
DataWidth_Gen	-	natural	1	How width in bits the data that shall snapshot aligned with the timestamp
BufferDepth_Gen	-	natural	1	How deep the buffer shall be in

				number of timestamps
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Enable Input				
Enable_EnaIn	in	std_logic	1	If the core is enabled or not, disabled will flush the buffer
Ready Input				
Ready_ValIn	in	std_logic	1	Signals if the Registerset is ready for the next timestamp
Drop Output				
Drop_ValOut	out	std_logic	1	Asserted when the FIFO had dropped/missed a timestamp
Timestamp Input				
ClockTimestamp_DatIn	in	Clk_Time_Type	1	Corrected Timestamp
ClockTimestamp_ValIn	in	std_logic	1	Corrected Timestamp valid
Data Input				
Data_DatIn	in	std_logic_vector	Data Width_ Gen	Data to snapshot
Count_CntIn	in	std_logic_vector	32	Timestamp Count
Timestamp Output				
ClockTimestamp_DatOut	out	Clk_Time_Type	1	Corrected Timestamp
ClockTimestamp_ValOut	out	std_logic	1	Corrected Timestamp valid
Data Output				
Data_DatOut	out	std_logic_vector	Data Width_ Gen	Data snapshot
Count_CntOut	out	std_logic_vector	32	Timestamp Count

Table 10: Timestamp Buffer

4.2.3 Registerset

4.2.3.1 Entity Block Diagram

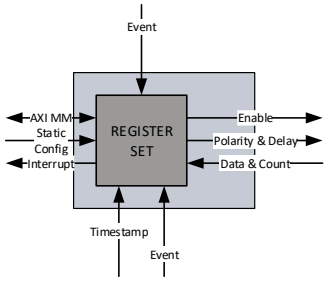

Figure 8: Registerset

4.2.3.2 Entity Description

Register Set

This module is an AXI4Lite Memory Mapped Slave. It provides access to the timestamp and data snapshot registers and allows configuring the Signal Timestamper. AXI4Lite only supports 32 bit wide data access, no byte enables, no burst, no simultaneous read and writes and no unaligned access. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration is done via signals and can be easily done from within the FPGA without CPU. For each parameter a valid signal is available, the enable signal shall be set last (or simultaneously). To change parameters the clock has to be disabled and enabled again. If in AXI mode, an AXI Master has to configure the Signal Timestamper with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.
This module also contains the snapshot of data logic. Timestamping and data snapshotting is automatically disabled after an event was detected and must be released via clearing of the interrupt. This implies that a CPU is reacting with the timestamper at all times to not lose a timestamp. For this situation, an internal counter is used which is incremented at every event independent of the interrupt state. This allows detecting lost events.
The number of timestamps possible to handle heavily depends on the interrupt latency, CPU speed and AXI bus load.

4.2.3.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
Register Set				
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
CableDelay_Gen	-	boolean	1	If a cable delay shall be configurable (only needed when connected externaly)
InputPolarity_Gen	-	boolean	1	True: High active, False: Low active
DataWidth_Gen	-	natural	1	How width in bits the data that shall snapshot aligned with the timestamp
AxiAddressRangeLow_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRangeHigh_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Clk_SignalTimestamperStaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_SignalTimestamperStaticConfigVal_Type	1	Static Configuration valid
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted Clock Time valid

AXI4 Lite Slave				
AxiWriteAddrValid_ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady_RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress_AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt_DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid_ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady_RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData_DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe_DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid_ValOut	out	std_logic	1	Write Response Valid
AxiWriteRespReady_RdyIn	in	std_logic	1	Write Response Ready
AxiWriteResp Response_DatOut	out	std_logic_vector	2	Write Response
AxiReadAddrValid_ValIn	in	std_logic	1	Read Address Valid
AxiReadAddrReady_RdyOut	out	std_logic	1	Read Address Ready
AxiReadAddrAddress_AdrIn	in	std_logic_vector	32	Read Address
AxiReadAddrProt_DatIn	in	std_logic_vector	3	Read Address Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadData Response_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response
Timestamp Signal Input				
ClockTimestamp_EvtIn	in	std_logic	1	Timestamp event
Timestamp Input				
ClockTimestamp_DatIn	in	Clk_Time_Type	1	Corrected Timestamp
ClockTimestamp_ValIn	in	std_logic	1	Corrected

				Timestamp valid
Data Input				
Data_DatIn	in	std_logic_vector	Data Width_ Gen	Data to snapshot
Count_CntIn	in	std_logic_vector	32	Timestamp Count
Drop Input				
Drop_ValIn	in	std_logic	1	Asserted when the FIFO had dropped/missed a timestamp
Signal Values Output				
SignalPolarity_DatOut	out	std_logic	1	'1': High active, '0': Low active
SignalCableDelay _DatOut	out	std_logic_vector	16	Delay in Nanosec-onds
Interrupt Output				
Irq_EvtOut	out	Clk_Clock StaticConfig_Type	1	Active high level interrupt
Enable Output				
TimestampEna-ble_DatOut	out	std_logic	1	Enable Signal Timestamper

Table 11: Registerset

4.3 Configuration example

In both cases the enabling of the core shall be done last, after or together with the configuration.

4.3.1 Static Configuration

```
  constant ClkStaticConfigSignalTimestamper_Con : Clk_SignalTimestamperStaticConfig_Type := (
    Polarity                        => '1'
  );

  constant ClkStaticConfigValSignalTimestamper_Con : Clk_SignalTimestamperStaticConfigVal_Type
:= (
    Enable_Val                      => '1'
  );
```

Figure 9: Static Configuration

4.3.2 AXI Configuration

The following code is a simplified pseudocode from the testbench: The base address of the Signal Timestamper is 0x10000000.

```
-- CLK SIGNALTIMESTAMPER
-- Config
-- Polarity = 1
AXI WRITE 10000008 00000001
-- enable IRQ
AXI WRITE 10000014 00000001
-- enable signal timestamper
AXI WRITE 10000000 00000001
```

Figure 10: AXI Configuration

4.4 Clocking and Reset Concept

4.4.1 Clocking

To keep the design as robust and simple as possible, the whole Ordinary Clock, including the Counter Clock and all other cores from NetTimeLogic are run in one clock domain. This is considered to be the system clock. Per default this clock is 50MHz. Where possible also the interfaces are run synchronous to this clock. For clock domain crossing asynchronous fifos with gray counters or message patterns with meat stability flip-flops are used. Clock domain crossings for the AXI interface is moved from the AXI slave to the AXI interconnect.

Clock	Frequency	Description
System		
System Clock	50MHz (Default)	System clock where the CC runs on as well as the counter clock etc.
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 12: Clocks

4.4.2Reset

In connection with the clocks, there is a reset signal for each clock domain. All resets are active low. All resets can be asynchronously set and shall be synchronously released with the corresponding clock domain. All resets shall be asserted for the first couple (around 8) clock cycles. All resets shall be set simultaneously and released simultaneously to avoid overflow conditions in the core. See the reference designs top file for an example of how the reset shall be handled.

Reset	Polarity	Description
System		
System Reset	Active low	Asynchronous set, synchronous release with the system clock
AXI Interface		
AXI Reset	Active low	Asynchronous set, synchronous release with the AXI clock, which is the same as

		the system clock

Table 13: Resets

5 Resource Usage

Since the FPGA Architecture between vendors and FPGA families differ there is a split up into the two major FPGA vendors.

5.1 Intel/Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static Config, 32bit data)	-	-	-	-
Maximal (AXI, 32bit data)	311	883	0	0

Table 14: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static Config, 32bit data)	-	-	-	-
Maximal (AXI, 32bit data)	311	632	0	0

Table 15: Resource Usage AMD/Xilinx

6 Delivery Structure

```
AXI                                 -- AXI library folder
 |-Library                          -- AXI library component sources
 |-Package                          -- AXI library package sources

CLK                                 -- CLK library folder
 |-Core                             -- CLK library cores
 |-Doc                              -- CLK library cores documentations
 |-Driver                           -- CLK library driver
 |-Library                          -- CLK library component sources
 |-Package                          -- CLK library package sources
 |-Refdesign                        -- CLK library cores reference designs
 |-Testbench                        -- CLK library cores testbench sources and sim/log

COMMON                              -- COMMON library folder
 |-Library                          -- COMMON library component sources
 |-Package                          -- COMMON library package sources

PPS                                 -- PPS library folder
|-Package                           -- PPS library package sources

SIM                                 -- SIM library folder
 |-Doc                              -- SIM library command documentation
 |-Package                          -- SIM library package sources
 |-Testbench                        -- SIM library testbench template sources
 |-Tools                            -- SIM simulation tools
```

7 Testbench

The Signal Timestamper testbench consist of 3 parse/port types: AXI, CLK and SIG. The Signal Input Port can check the generated signal from the Signal Output Port which uses the same clock reference from the Clock Port as the Signal Timestamper. For configuration and result checks an AXI read and write port is used. Also interrupt handling can be done with this.

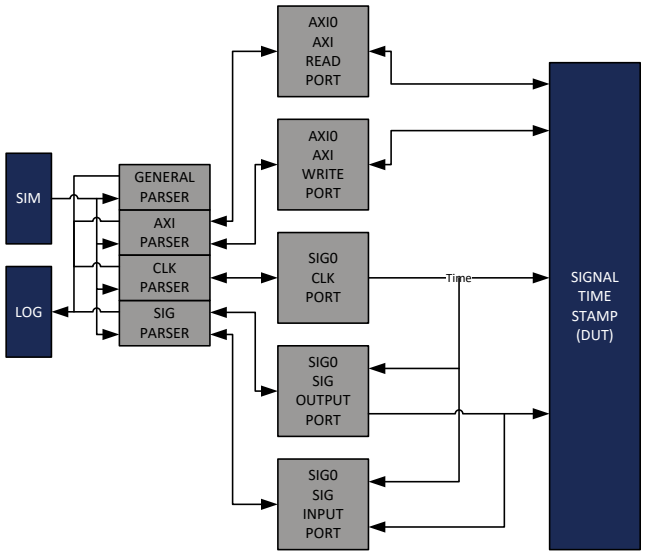

Figure 11: Testbench Framework

For more information on the testbench framework check the Sim_ReferenceManual documentation.

With the Sim parameter set the time base for timeouts are divided by 1000 to 100000 to speed up simulation time.

7.1 Run Testbench

1. Run the general script first

```
source XXX/SIM/Tools/source_with_args.tcl
```

2. Start the testbench with all test cases

```
src XXX/CLK/Testbench/Core/ClkSignalTimestamper/Script/run_Clk_SignalTimestamper_Tb.tcl
```

3. Check the log file LogFile1.txt in the XXX/CLK/Testbench/Core/ClkSignalTimestamper/Log/ folder for simulation results.

8 Reference Designs

The Signal Timestamper reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz) and an instance of the Signal Timestamper IP core and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately). Optionally it also contains an instance of a PPS Master Clock IP core (has to be purchased separately). To instantiate the optional IP core, change the corresponding generic (PpsMasterAvailable_Gen) to true via the tool specific wizards.
The Reference Design is intended to run just standalone and show the instantiation.Since now CPU is connected in the reference Design the timestamper will not create any IRQs or snapshots The PPS Master Clock is used to create a PPS output which is compensated for the output delay and has a configurable duty cycle, if not available an uncompensated PPS is directly generated out of the MSB of the Time. All generics can be adapted to the specific needs.

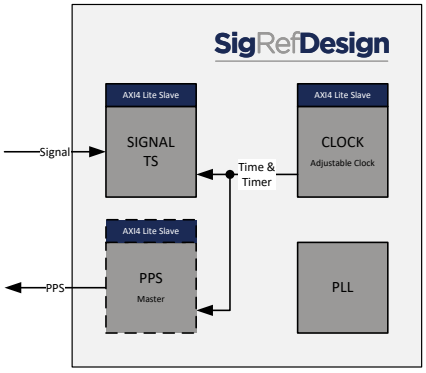

Figure 12: Reference Design

8.1 Intel/Altera: Terasic SocKit

The SocKit board is an FPGA board from Terasic Inc. with a Cyclone V SoC FPGA from Intel/Altera. (http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=816)

1. Open Quartus 16.x
2. Open Project /CLK/Refdesign/Altera/SocKit/ClkSignalTimestamper/ClkSignalTimestamper.qpf
3. If the optional core PPS Master Clock is available add the files from the corresponding folders (PPS/Core, PPS/Library and PPS/Package)

4. Change the generics (PpsMasterAvailable_Gen) in Quartus (in the settings menu, not in VHDL) to true for the optional cores that are available.
5. Rerun implementation
6. Download to FPGA via JTAG

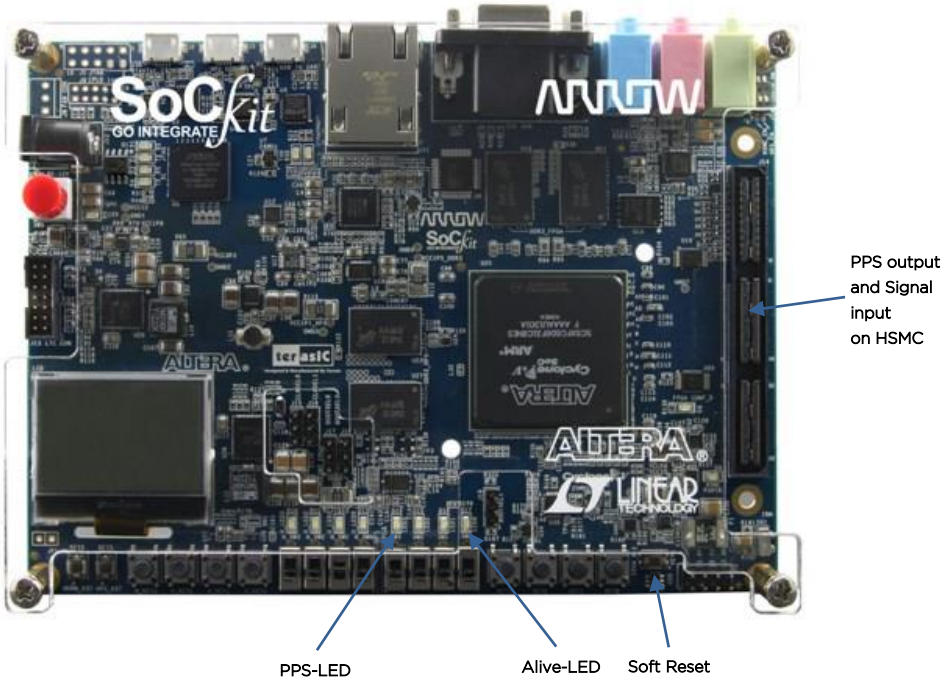

Figure 13: SocKit (source Terasic Inc)

For the ports on the HSMC connector the GPIO to HSMC adapter from Terasic Inc. was used.

8.2 AMD/Xilinx: Digilent Arty

The Arty board is an FPGA board from Digilent Inc. with an Artix7 FPGA from AMD/Xilinx. (http://store.digilentinc.com/arty-board-artix-7-fpga-development-board-for-makers-and-hobbyists/

1. Open Vivado 2019.1.
   Note: If a different Vivado version is used, see chapter 8.3.
2. Run TCL script
   /CLK/Refdesign/Xilinx/Arty/ClkSignalTimestamper/ClkSignalTimestamper.tcl
   a. This has to be run only the first time and will create a new Vivado Project

3. If the project has been created before open the project and do not rerun the project TCL
4. If the optional core PPS Master Clock is available add the files from the corresponding folders (PPS/Core, PPS/Library and PPS/Package) to the corresponding Library (PpsLib).
5. Change the generics (PpsMasterAvailable_Gen) in Vivado (in the settings menu, not in VHDL) to true for the optional cores that are available.
6. Rerun implementation
7. Download to FPGA via JTAG

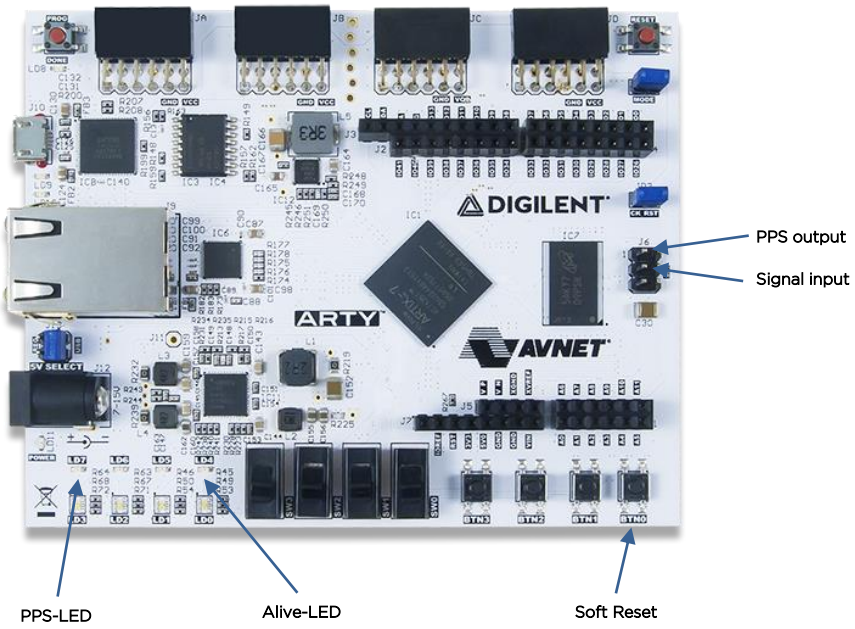

Figure 14: Arty (source Digilent Inc)

8.3 AMD/Xilinx: Vivado Version

The provided TCL script for creation of the reference-design project is targeting AMD/Xilinx Vivado 2019.1.
If a lower Vivado version is used, it is recommended to upgrade to Vivado 2019.1 or higher.
If a higher Vivado version is used, the following steps are recommended:

- Before executing the project creation TCL script, the script's references of Vivado 2019 should be manually replaced to the current Vivado version. For example, if version Vivado 2022 is used, then:
  - The statement occurrences:
    ```
    set_property flow "Vivado Synthesis 2019" $obj
    ```
    shall be replaced by:
    ```
    set_property flow "Vivado Synthesis 2022 $obj
    ```
  - The statement occurrences:
    ```
    set_property flow "Vivado Implementation 2019" $obj
    ```
    shall be replaced by:
    ```
    set_property flow "Vivado Implementation 2022" $obj
    ```
- After executing the project creation TCL script, the AMD/Xilinx IP cores, such as the Clocking Wizard core, might be locked and a version upgrade might be required. To do so:
  1. At "Reports" menu, select "Report IP Status".
  2. At the opened "IP Status" window, select "Upgrade Selected". The tool will upgrade the version of the selected IP cores.

A List of tables

B List of figures