PpsSlaveClock

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.7
Date	05.04.2024

Copyright Notice

Copyright © 2025 NetTimeLogic GmbH, Switzerland. All rights reserved.
Unauthorized duplication of this document, in whole or in part, by any means, is prohibited without the prior written permission of NetTimeLogic GmbH, Switzerland.
All referenced registered marks and trademarks are the property of their respective owners

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 PPS Slave Clock is a full hardware (FPGA) only implementation of a synchronization core able to synchronize to a Pulse per Second input.
The whole algorithms and calculations are implemented in the core, no CPU is required. This allows running PPS synchronization completely independent and standalone from the user application. The core can be configured either by signals or by an AXI4Lite-Slave Register interface.

Key Features:

- PPS Slave Clock
- Input signal filter
- PPS supervision
- PPS duty cycle analysis, width can be read via register
- Synchronization accuracy: +/- 25ns
- AXI4Lite register set or static configuration
- Timestamp resolution with 50 MHz system clock: 10ns
- Optional High Resolution Timestamping with 4ns resolution
- Optional TDC Timestamping with 1ns resolution
- Hardware PI Servo

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	28.12.2015	First draft
1.0	13.05.2016	First release
1.1	19.05.2016	Added structured types section
1.2	07.06.2016	Added polarity
1.3	20.12.2017	Status interface added
1.4	25.02.2020	HighResolution added
1.5	03.01.2023	Added Vivado upgrade version description
1.6	26.05.2023	Extended Cable delay range
1.7	05.04.2024	Added TDC

Table 1: Revision History

Content

1 INTRODUCTION 8

1.1 Context Overview 8

1.2 Function 8

1.3 Architecture 9

2 PPS BASICS 11

2.1 Interface 11

2.2 Delays 11

2.3 Accuracy 11

3 REGISTER SET 12

3.1 Register Overview 12

3.2 Register Descriptions 13

3.2.1 General 13

4 DESIGN DESCRIPTION 19

4.1 Top Level – Pps Slave 19

4.2 Design Parts 28

4.2.1 RX Processor 28

4.2.2 Registerset 33

4.3 Configuration example 36

4.3.1 Static Configuration 36

4.3.2 AXI Configuration 36

4.4 Clocking and Reset Concept 37

4.4.1 Clocking 37

4.4.2 Reset 37

5	**RESOURCE USAGE**	**39**
5.1	Intel/Altera (Cyclone V)	39
5.2	AMD/Xilinx (Artix 7)	39
6	**DELIVERY STRUCTURE**	**40**
7	**TESTBENCH**	**41**
7.1	Run Testbench	41
8	**REFERENCE DESIGNS**	**43**
8.1	Intel/Altera: Terasic SocKit	43
8.2	AMD/Xilinx: Digilent Arty	44
8.3	AMD/Xilinx: Vivado version	45

Definitions

Definitions	
PPS Slave Clock	A clock that can synchronize itself to a PPS input
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
PS	PPS Slave
TS	Timestamp
TDC	Time-to-Digital Converted
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 PPS Slave Clock is meant as a co-processor handling a PPS input.
It takes a PPS input, filters it, calculates offset and drift and corrects it on the Counter Clock. In parallel the duty cycle and state of the PPS is monitored which can be read via registers.
The PPS Slave Clock is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Lite slave for configuration and status supervision from a CPU, this is however not required since the PPS Slave Clock can also be configured statically via signals/constants directly from the FPGA.
It can be combined with a TOD Slave clock to synchronize for e.g. to a GPS receiver. Offset and drift are then corrected by the PPS Slave Clock to the next second and the TOD Slave Clock will correct the absolute time on seconds level.

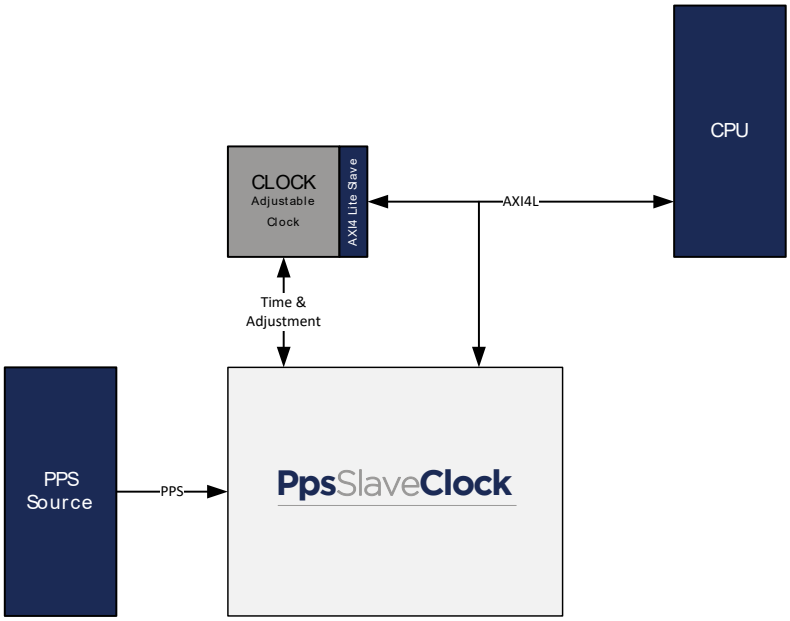

Figure 1: Context Block Diagram

1.2 Function

The PPS Slave Clock takes a PPS of configurable polarity and filters the input for minimum stable phases before next edge and changes the polarity to high active. This filtered signal is analyzed by the supervision and pulse width analyzer. This

block checks the PPS signal for correct patterns and extracts the duty cycle of the PPS and provides it to the register set for further processing by a CPU. In parallel to the analysis of the filtered PPS signal a timestamp is taken at the rising edge of the filtered PPS and compensated for the input processing time. After 2 consecutive correct PPS timestamps are taken the calculation of offset and drift is started. For offset correction an additional cable delay is added to the calculation to compensate the propagation delay between the master and slave. The offset is corrected to the next best second boundary. Offset and drift corrections are feed to the PI servo loops of the Adjustable Counter Clock Core and the output of the Servo loop used for the next calculations. In case of an error the correction is stopped until two PPS edges were correct again after the error flag was de-asserted.

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.

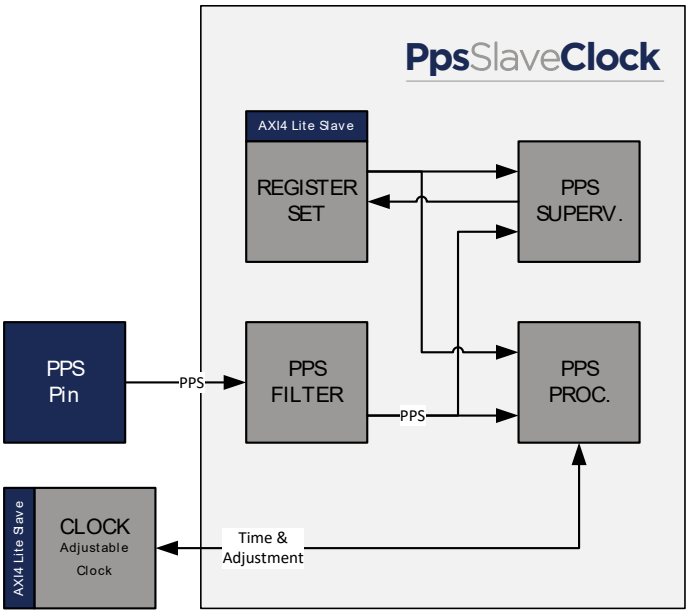

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

PPS Filter

This filters the input signals and outputs a regenerated signal which can be used by the other blocks. It is a look ahead filter which means it will immediately react on a signal change if the signal was stable for a long enough period. Afterwards signal changes will not have an effect until the signal was stable long enough. This is done this way to get maximum accuracy on the first edge of a timestamp. However this will not filter out glitches. A glitch will be detected and cause that the drift and offset calculation will wait for the 2 contiguous PPS cycles without glitches. A slow rising edge (around 80ns) however will not give any issues, this will be filtered out.

PPS Supervision

This block analyses the already filtered PPS. It will check the signal for the duty cycle and period and signal an error if one of them is out of bounds. In parallel to supervision it also extracts the duty cycle of the PPS.

PPS Processor

This block takes a timestamp of the rising edge of the PPS, takes the propagation delay of the cable into account, calculates the offset and the drift and adjusts the local clock.

2 PPS Basics

2.1 Interface

The Pulse per Second is a very simple interface and can be electrical or optical. It can be a single ended, differential, open drain, open collector and therefore also high or low active signal. The signal has a frequency of 1Hz as the name says.
The reference point is the edge to the active level; this shall be at the second overflow of the reference clock. This edge shall be very accurate compared to the edge to the idle level (drive active level, tristate idle level).
For high accuracy synchronization delays have to be compensated for, and the duty cycle of the PPS can be used for accuracy encoding.
A PPS network is normally a one-to-many configuration: one PPS master synchronizes multiple PPS slaves of different distance from the master.

2.2 Delays

There are two kinds of delays in a PPS Network. One is the input delay of the PPS to the core; this shall be constant and is compensated for. The second delay is the propagation delay of the signal from the master to the slave. This is dependent on the cable length and medium: 15cm of copper cable are equal to roughly 1ns of propagation delay. This delay can be set in the core, e.g. for compensation of the cable length to a time server on the building's roof.

2.3 Accuracy

Some PPS Masters are capable of encoding its synchronization accuracy to the duty cycle of the PPS signal. Often a logarithmic scale is used to encode the accuracy to a primary reference. E.g. 100ms of duty cycle = 10ns, 200ms = 100ns, 300ms = 1000ns, 400ms = 10000ns … However this is not standardized. This core is capable of measuring the duty cycle with millisecond resolution (+/- 1ms). Interpretation of the duty cycle is up to the user.

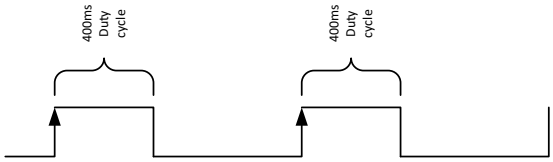

Figure 3: PPS Waveform

3 Register Set

This is the register set of the PPS Slave Clock. 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
Pps SlaveControl Reg	PPS Slave Enable Control Register	0x00000000	RW
Pps SlaveStatus Reg	PPS Slave Error Status Register	0x00000004	WC
Pps SlavePolarity Reg	PPS Slave Polarity Register	0x00000008	RW
Pps SlaveVersion Reg	PPS Slave Version Register	0x0000000C	RO
Pps SlavePulseWidth Reg	PPS Slave Pulse Width Register	0x00000010	RO
Pps SlaveCableDelay Reg	PPS Slave Cable Delay Register	0x00000020	RW

Table 4: Register Set Overview

3.2 Register Descriptions

3.2.1 General

3.2.1.1 PPS Slave Control Register

Used for general control over the PPS Slave Clock, all configurations on the core shall only be done when disabled.

PPS SlaveControl 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 PPS Slave Status Register

Shows the current status of the PPS Slave Clock.

Pps SlaveStatus 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
																														SUPERVISION_ERROR	FILTER_ERROR
RO																														WC	WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:2	RO
SUPERVISION_ERROR	PPS Supervision Error (sticky)	Bit: 1	WC
FILTER_ERROR	PPS Filter Error (sticky)	Bit: 0	WC

3.2.1.3 PPS Slave Polarity Register

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

PPS SlavePolarity 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 PPS Slave 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.

Pps SlaveVersion 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																															
0xXXXXXXXX																															
Offset: 0x000C																															

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

3.2.1.5 PPS Slave Pulse Width Register

Shows the current pulse width in milliseconds of the PPS input generated by the PPS source. This can be useful if the Master supports accuracy encoding on the PPS duty cycle (as NetTimeLogic's Master is capable of)

Pps SlavePulseWidth 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
-																						PULSE_WIDTH									
RO																						RO									
Reset: 0x000003FF																															
Offset: 0x0010																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:10	RO
PULSE_WIDTH	Observed pulse width of PPS in milliseconds (0x3FF means no pulse length known or pulse wrong (< 100ms or > 1000ms))	Bit: 9:0	RO

3.2.1.6 PPS Slave Cable Delay Register

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

Pps SlaveCableDelay 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
SIGN	-	CABLE_DELAY																													
RW	RO	RW																													
Reset: 0x00000000																															
Offset: 0x0020																															

Name	Description	Bits	Access
SIGN	Sign of the cable delay, default positive (0)	Bit: 31	RW
-	Reserved, read 0	Bit: 30	RO
CABLE_DELAY	Cable delay of cable to master in nanoseconds (15cm is around 1ns)	Bit: 29:0	RW

4 Design Description

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

4.1 Top Level – Pps Slave

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
PulseWidthDynamic Support_Gen	boolean	1	Support for Pulse width analysis: true = pulse width is available to read, false = pulse width is ignored
StaticConfig_Gen	boolean	1	If Static Configuration or AXI is used
FilterDelay Millisecond_Gen	natural	1	Min stable time of input PPS before next edge (also min duty cycle required of the PPS)
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
InputDelay Nanosecond_Gen	natural	1	Input delay of the PPS from the connector to the input signal.
InputPolarity_Gen	boolean	1	true = high active, false = low active
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
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
TdcSupport_Gen	boolean	1	If TDC is supported
TdcCarryDelay Femtosecond_Gen	natural	1	Delay of a Carry element
TdcInputDelay Picoseconds_Gen	natural	1	Delay from the IO Pin until the first Carry
TdcFixPosition_Gen	boolean	1	If the position of the TDC shall be fixed in the design (Xilinx only)
TdcXPosition_Gen	natural	1	TDC Start position Slice X position. Area is (X-1) - (X+1)
TdcYPosition_Gen	natural	1	TDC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
Sim_Gen	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis

Table 5: Parameters

4.1.1.2 Structured Types

4.1.1.2.1 Clk_Time_Type

Defined in Clk_Package.vhd 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 6: Clk_Time_Type

4.1.1.2.2 Clk_TimeAdjustment_Type

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

Field Name	Type	Size	Description
TimeAdjustment	Clk_Time_Type	1	Time to adjust
Interval	std_logic_vector	32	Adjustment interval, for the drift correction this is the denumerator of the rate in nanoseconds (TimeAdjust-ment every Interval = drift rate), for offset correction this is the period in which the time shall be correct-ed(TimeAdjustment in Inter-val), for setting the time this has no mining.
Valid	std_logic	1	Whether the Adjustment is valid or not

Table 7: Clk_TimeAdjustment_Type

4.1.1.2.3 Pps_SlaveStaticConfig_Type

Defined in Pps_SlaveAddrPackage.vhd of library PpsLib
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	30	Compensation value for the cable delay between master and slave in Nanoseconds: 1ns = 15cm

Table 8: Pps_SlaveStaticConfig_Type

4.1.1.2.4 Pps_SlaveStaticConfigVal_Type

Defined in Pps_SlaveAddrPackage.vhd of library PpsLib
This is the type used for valid flags of the static configuration.

Field Name	Type	Size	Description
Enable_Val	std_logic	1	Enables the PPS Slave

Table 9: Pps_SlaveStaticConfigVal_Type

4.1.1.2.5 Pps_SlaveStaticStatus_Type

Defined in Pps_SlaveAddrPackage.vhd of library PpsLib
This is the type used for static status supervision.

Field Name	Type	Size	Description
CoreInfo	Clk_CoreInfo_ Type	1	Infor about the Cores state

Table 10: Pps_SlaveStaticConfig_Type

4.1.1.2.6 Pps_SlaveStaticStatusVal_Type

Defined in Pps_SlaveAddrPackage.vhd of library PpsLib
This is the type used for valid flags of the static status supervision.

Field Name	Type	Size	Description
CoreInfo_Val	std_logic	1	Core Info valid

Table 11: Pps_SlaveStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

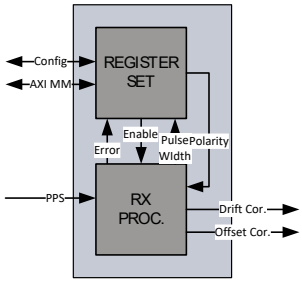

Figure 4: PPS Slave Clock

4.1.1.4 Entity Description

Rx Processor

This module handles the incoming PPS signal. It filters and timestamps the rising edge of the PPS input with the local clock. In parallel it supervises the input signal for abnormalities and for the duty cycle which it provides to the Registerset. From the timestamp of the rising edge of the PPS it calculates drift and offset and adjusts the local clock.
See 4.2.1 for more details.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows configuring the PPS Slave Clock. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the registers is done via signals and can be easily done from within the FPGA without CPU. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU
See 4.2.2 for more details.

4.1.1.5 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
PulseWidthDynamicSupport_Gen	-	boolean	1	Support for Pulse width analysis
StaticConfig_Gen	-	boolean	1	If Static Configura-

				tion or AXI is used
FilterDelay Millisecond_Gen	-	natural	1	Min stable time of input PPS before next edge
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the connector to the input signal.
InputPolarity_Gen	-	boolean	1	True: High active, False: Low active
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
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
TdcSupport_Gen	-	boolean	1	If TDC is supported
TdcCarryDelay Femtosecond_Gen	-	natural	1	Delay of a Carry element
TdcInputDelay Picoseconds_Gen	-	natural	1	Delay from the IO Pin until the first Carry
TdcFixPosition_Gen	-	boolean	1	If the position of the TDC shall be fixed in the design (Xilinx only)
TdcXPosition_Gen	-	natural	1	TDC Start position Slice X position. Area is (X-1) - (X+1)

TdcYPosition_Gen	-	natural	1	TDC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
MaxOffset_Gen	-	natural	1	If Offset is larger than this change into Uncalibrated state if Slave
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	Pps_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Pps_Slave StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Pps_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Pps_Slave StaticStatusVal _Type	1	Static Status 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
Pulse Per Second Input				
Pps_EvtIn	in	std_logic	1	PPS input from a PPS Master
Time Adjustment Output				
TimeAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Time to set hard (unused)
TimeAdjustment _ValOut	out	std_logic	1	Time valid (unused)

Offset Adjustment Output				
OffsetAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Offset between Master and Slave
OffsetAdjustment _ValOut	out	std_logic;	1	Calculated new Offset valid
Drift Adjustment Output				
DriftAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Drift between Master and Slave
DriftAdjustment _ValOut	out	std_logic;	1	Calculated new Drift valid
Offset Adjustment Input				
OffsetAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Offset after the PI Servo loop
OffsetAdjustment _ValIn	in	std_logic;	1	Calculated new Offset after the PI Servo loop valid
Drift Adjustment Input				
DriftAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Drift after the PI Servo loop
DriftAdjustment _ValIn	in	std_logic	1	Calculated new Drift after the PI Servo loop valid

Table 12: PPS Slave Clock

4.2 Design Parts

The PPS Slave Clock 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 RX Processor

4.2.1.1 Entity Block Diagram

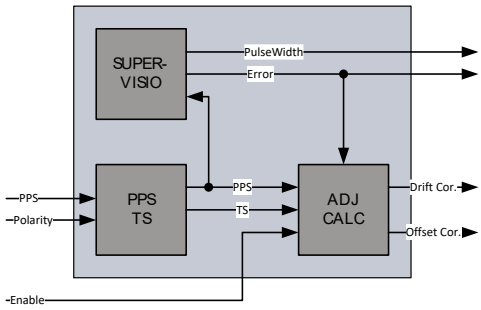

Figure 5: RX Processor

4.2.1.2 Entity Description

PPS Timestamper

This module filters the incoming PPS and takes a timestamp. of the Counter Clock at the detection of the rising edge of the PPS. Filtering is done that way that a certain time before an edge the signal had to be stable. Therefore a small spike will still cause a pulse but a slow rise time or bouncing at the start of a pulse will be filtered out. This is done this way to get maximum accuracy for the first edge of a pulse but ignoring an “ugly” signal edge. In the same step of filtering the filtered output PPS is created which is always high active. This high active PPS is then the input to the timestamper which compensates the input and preprocessing delay of the signal.

Supervision

This module checks the waveform of the filtered PPS and extracts the duty cycle. If no PPS is detected or wrong duty cycles are detected an error is signaled.
The duty cycle is provided as PulseWidth to the Registerset and gives information about the PPS Master’s accuracy (if supported)

Adjustment Calculation

This module calculates the drift and offset of the local clock and corrects it. After enabling or error detection the calculation waits for two consecutive PPS before adjusting the clock again. Initially the offset is corrected to the next best second by either slowing down or accelerating the local clock.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
RX Processor				
PulseWidthDynamic Support_Gen	-	boolean	1	Support for Pulse width analysis
FilterDelay Millisecond_Gen	-	natural	1	Min stable time of input PPS before next edge
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the con-nector to the input signal.
InputPolarity_Gen	-	boolean	1	True: High active, False: Low cctive
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
TdcSupport_Gen	-	boolean	1	If TDC is supported
TdcCarryDelay Femtosecond_Gen	-	natural	1	Delay of a Carry element
TdcInputDelay Picoseconds_Gen	-	natural	1	Delay from the IO Pin until the first

				Carry
TdcFixPosition_Gen	-	boolean	1	If the position of the TDC shall be fixed in the design (Xilinx only)
TdcXPosition_Gen	-	natural	1	TDC Start position Slice X position. Area is (X-1) - (X+1)
TdcYPosition_Gen	-	natural	1	TDC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
MaxOffset_Gen	-	natural	1	If Offset is larger than this change into Uncalibrated state if Slave
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
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
Pulse Per Second Polarity				
PpsPolarity_DatIn	in	std_logic	10	'1': High active, '0': Low active
Pulse Per Second Error Output				
Pps_ErrOut	out	std_logic_vector	2	Indicates an error either in the filter or because of missing

				PPS
Pulse Per Second Width Output				
PpsPulse-Width_DatOut	out	std_logic_vector	10	0-999 in millisecond marks the duty cycle of the incom-ing PPS
Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the correc-tion and supervision
Pulse Per Second Cable Delay Input				
PpsCableDelay_DatIn	in	Clk_Time_Type	1	Propagation delay of the PPS signal from the master source to the con-nector.
Pulse Per Second Input				
Pps_EvtIn	in	std_logic	1	PPS input from a PPS Master
Offset Adjustment Output				
OffsetAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Offset between Master and Slave
OffsetAdjustment _ValOut	out	std_logic;	1	Calculated new Offset valid
Drift Adjustment Output				
DriftAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Drift between Master and Slave
DriftAdjustment _ValOut	out	std_logic;	1	Calculated new Drift valid
Offset Adjustment Input				
OffsetAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Offset after the PI Servo loop
OffsetAdjustment _ValIn	in	std_logic;	1	Calculated new Offset after the PI Servo loop valid
Drift Adjustment Input				
DriftAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Drift after the PI Servo

				loop
DriftAdjustment _ValIn	in	std_logic	1	Calculated new Drift after the PI Servo loop valid

Table 13: RX Processor

4.2.2 Registerset

4.2.2.1 Entity Block Diagram

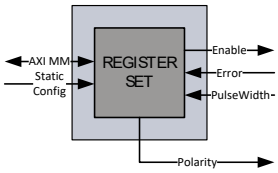

Figure 6: Registerset

4.2.2.2 Entity Description

Register Set

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows configuring the PPS Slave Clock. 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 of the registers 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 configuration parameters the clock has to be disabled and enabled again, the cable delay value can be changed at runtime. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.

4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
PulseWidthDynamicSupport_Gen	-	boolean	1	Support for Pulse width analysis
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
Register Set				
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used

AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_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	Pps_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Pps_Slave StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Pps_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Pps_Slave StaticStatusVal _Type	1	Static Status 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	out	std_logic_vector	2	Write Response

Response_DatOut				
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
Pulse Per Second Polarity				
PpsPolarity_DatOut	out	std_logic	10	'1': High active, '0': Low active
Pulse Per Second Error Input				
Pps_ErrIn	in	std_logic_vector	2	Indicates an error either in the filter or because of missing PPS
Pulse Per Second Width Output				
PpsPulseWidth_DatIn	in	std_logic_vector	10	0-999 in millisecond marks the duty cycle of the incoming PPS
Pulse Per Second Cable Delay Output				
PpsCable Delay_DatOut	in	Clk_Time_Type	1	Propagation delay of the PPS signal from the master source to the connector.
Enable Output				
PpsSlave Enable_DatOut	out	std_logic	1	Enables the correction and supervision

Table 14: 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 PpsStaticConfigSlave_Con : Pps_SlaveStaticConfig_Type := (
  Polarity                    => '1',
  CableDelay                  => std_logic_vector(to_unsigned(128, 30)) - 128 ns
);

constant PpsStaticConfigValSlave_Con : Pps_SlaveStaticConfigVal_Type := (
  Enable_Val                  => '1'
);
```

Figure 7: Static Configuration

The cable delay can be changed at runtime. It is always valid.

4.3.2 AXI Configuration

The following code is a simplified pseudocode from the testbench: The base address of the PPS Slave Clock is 0x10000000.

```
-- PPS SLAVE
-- Config
-- Set polarity to high active
AXI WRITE 10000008 00000001
-- Set cable delay to 128 ns
AXI WRITE 10000020 00000080
-- enable PPS Slave
AXI WRITE 10000000 00000001
```

Figure 8: AXI Configuration

In the example the Cable delay is first set to 128ns then the core is enabled.

4.4 Clocking and Reset Concept

4.4.1 Clocking

To keep the design as robust and simple as possible, the whole PPS Slave 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 meta-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 PPS Slave runs on as well as the counter clock etc.
PPS Interface		
PPS	1 Hz	No clock, asynchronous data signal.
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 15: Clocks

4.4.2 Reset

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 16: 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 (No Dynamic pulse width support)	993	3668	0	0
Maximal (Dynamic pulse width support)	1003	3696	0	0

Table 17: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (No Dynamic pulse width support)	1022	3692	0	0
Maximal (Dynamic pulse width support)	1081	3790	0	0

Table 18: 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
 |-Library                          -- CLK library component sources
 |-Package                          -- CLK library package sources

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

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

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 Pps Slave testbench consist of 3 parse/port types: AXI, CLK and SIG.
The SIG output port takes the CLK port time as reference and sets the output signals aligned with the time from the CLK. The SIG input port takes the time of the Clock instance as reference and the signal from the SIG output port. Once the clock is synchronized the CLK port and Clock generated time should be phase and frequency aligned to a second. In addition for configuration and result checks an AXI read and write port is used in the testbench and for accessing more than one AXI slave also an AXI interconnect is required.

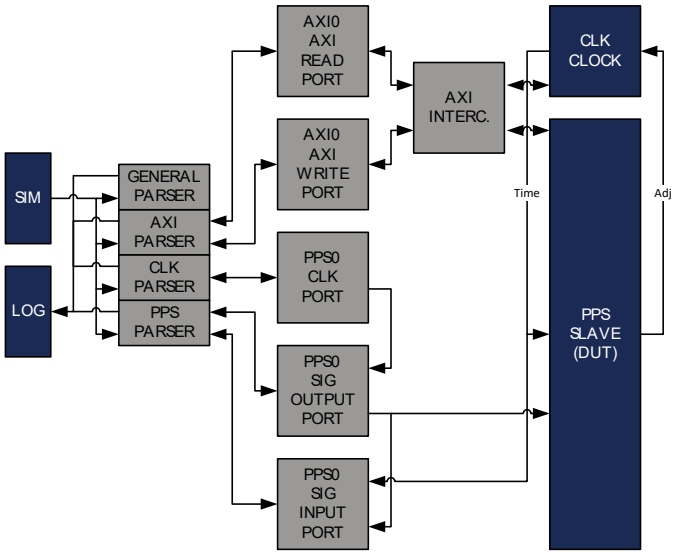

Figure 9: Testbench Framework

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/PPS/Testbench/Core/PpsSlave/Script/run_Pps_Slave_Tb.tcl
```

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

8 Reference Designs

The PPS Slave reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the PPS Slave Clock 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 be connected to any PPS Master. The Phase and Frequency is corrected via the PPS Slave Clock. 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. Via the dip switches the cable delay can be entered in 5ns steps.
All generics can be adapted to the specific needs.

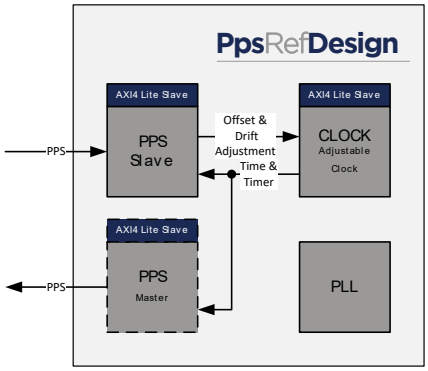

Figure 10: Reference Design

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 /PPS/Refdesign/Altera/SocKit/PpsSlave/PpsSlave.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

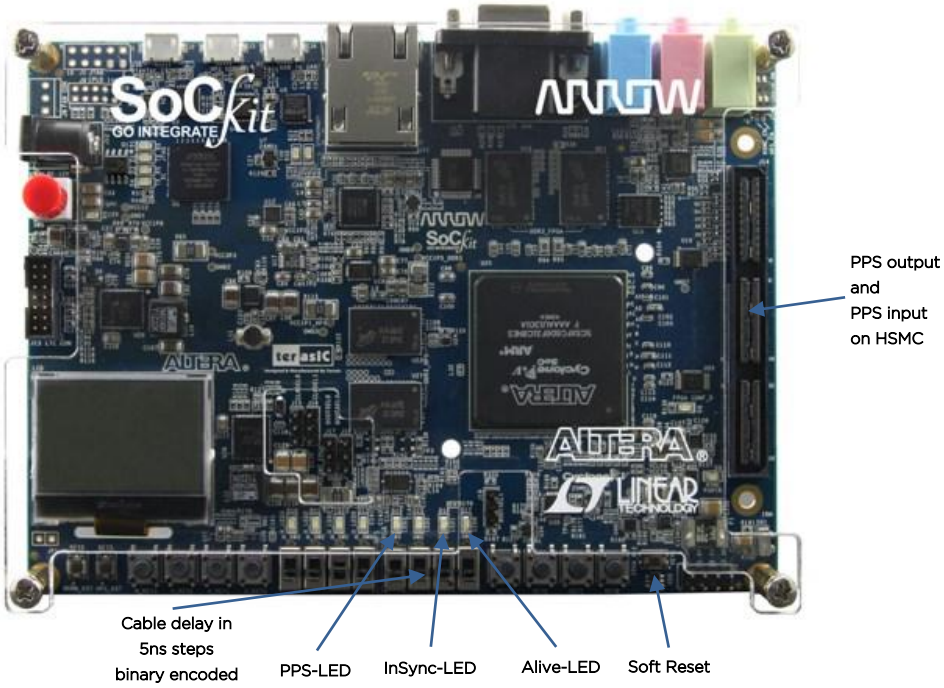

Figure 11: SocKit (source Terasic Inc)

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 /PPS/Refdesign/Xilinx/Arty/PpsSlave/PpsSlave.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

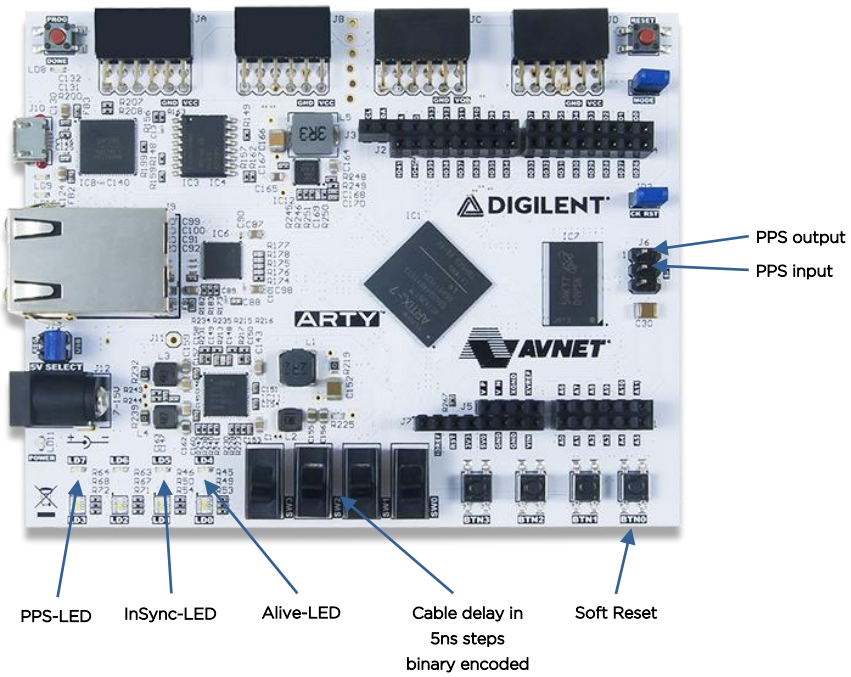

Figure 12: Arty (source Digilent Inc)

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

Table 1: Revision History 4
Table 2: Definitions 7
Table 3: Abbreviations 7
Table 4: Register Set Overview 12
Table 5: Parameters 20
Table 6: Clk_Time_Type 21
Table 7: Clk_TimeAdjustment_Type 21
Table 8: Pps_SlaveStaticConfig_Type 22
Table 9: Pps_SlaveStaticConfigVal_Type 22
Table 10: Pps_SlaveStaticConfig_Type 22
Table 11: Pps_SlaveStaticConfigVal_Type 22
Table 12: PPS Slave Clock 27
Table 13: RX Processor 32
Table 14: Registerset 35
Table 15: Clocks 37
Table 16: Resets 38
Table 17: Resource Usage Intel/Altera 39
Table 18: Resource Usage AMD/Xilinx 39

B List of figures

Figure 1: Context Block Diagram 8
Figure 2: Architecture Block Diagram 9
Figure 3: PPS Waveform 11
Figure 4: PPS Slave Clock 23
Figure 5: RX Processor 28
Figure 6: Registerset 33
Figure 7: Static Configuration 36
Figure 8: AXI Configuration 36
Figure 9: Testbench Framework 41
Figure 10: Reference Design 43
Figure 11: SocKit (source Terasic Inc) 44
Figure 12: Arty (source Digilent Inc) 45