PpsClkToPps

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.0
Date	03.01.2023

Copyright Notice

Disclaimer

Overview

NetTimeLogic's PPS Clock to PPS core is a full hardware (FPGA) only implementation of a PPS generator out of a clock of configurable frequency, it is intended to be connected to a PPS Slave core able to syntonize to a Pulse per Second.
The core also checks if the input clock is in the configured range and only if so will generate a PPS. The core can be configured either by signals or by an AXI4Lite-Slave Register interface.
This core is intended to be used with either external clocks or also SyncE clocks when the frequency shall be adjusted numerically rather than a clock switch.

Key Features:

- Configurable input frequency from 100Hz to 100MHz
- Input frequency supervision
- PPS duty cycle configurable in ms steps
- PPS Generation runs directly on Input Clock (minimal Jitter)
- AXI4Lite register set or static configuration

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	06.02.2024	First draft
1.0	09.02.2024	First releas

Table 1: Revision History

Content

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)
CLK	Clock
IRQ	Interrupt, Signaling to e.g. a CPU
PPS	Pulse Per Second
PS	PPS Slave
TS	Timestamp
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 Clock to PPS core is meant as a co-processor to convert a clock input to a Pulse Per Second (PPS) which can be fed to a PPS Slave.
It takes a clock input of configurable frequency and generates a Pulse Per Second (1Hz signal) of configurable duty cycle width.
The PPS Clock to PPS core is designed to work in cooperation with the PPS Slave Clock and 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 Clock to PPS core can also be configured statically via signals/constants directly from the FPGA.

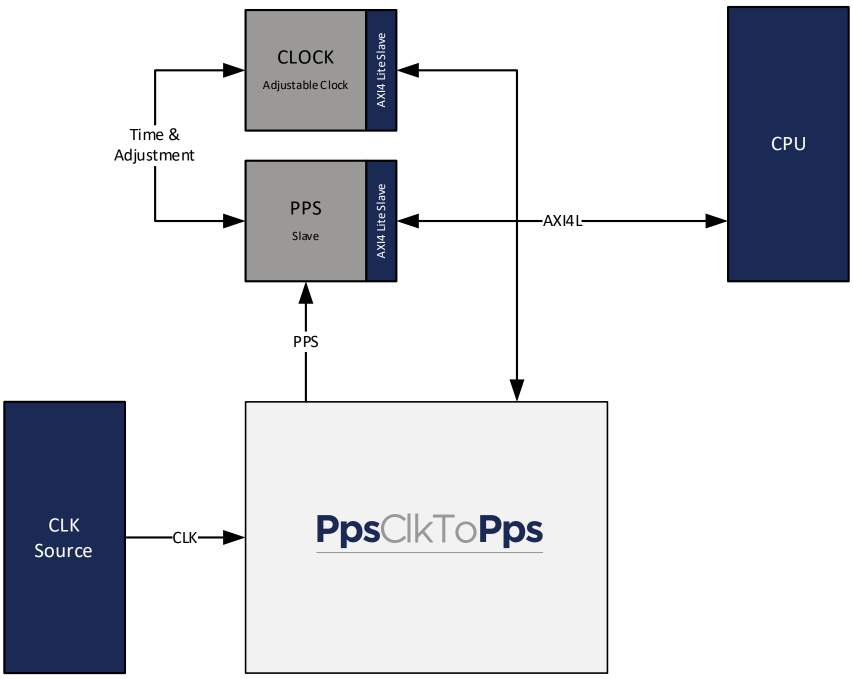

Figure 1: Context Block Diagram

1.2 Function

The PPS Clock to PPS core takes a Clock of configurable frequency and count the number of Clock cycles to generate a Pulse Per Second of configurable polarity and duty cycle. In addition, it checks if the input frequency is the range of the configured and only if it is in range generates a PPS.

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.

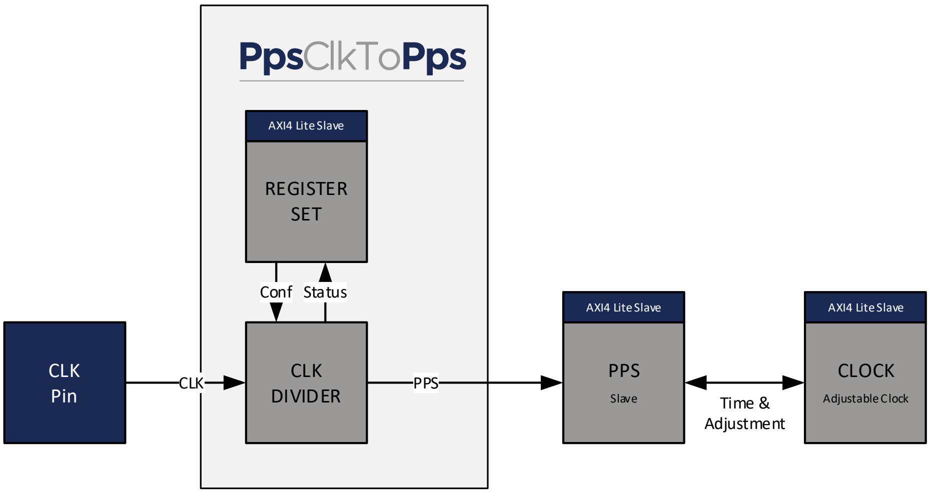

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

Clock Divider

This divides the input clock into a 1Hz aka Pulse Per Second signal of configurable polarity and duty cycle. In addition it checks if the input frequency is within 12.5% of the configured frequency. Only if the frequency is in range it will generate the PPS otherwise it will signal an error.

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. Since in this case the input is now only a frequency, we don't have any phase information and we only generate a 1Hz signal, a PPS Slave in this case can only do syntonization and not complete synchronization.

2.2 Accuracy

Some PPS Sources 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 can configure the duty cycle with millisecond resolution (+/- 1ms). Interpretation of the duty cycle is up to the user.

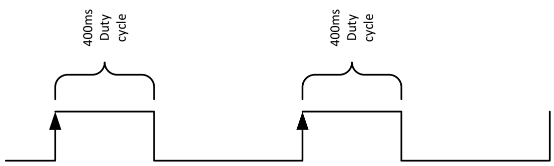

Figure 3: PPS Waveform

3 Register Set

This is the register set of the PPS Clock to PPS core. 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 ClkToPpsControl Reg	Pps Clk To Pps Enable Control Register	0x00000000	RW
Pps ClkToPpsStatus Reg	Pps Clk To Pps Error Status Register	0x00000004	WC
Pps ClkToPpsPolarity Reg	Pps Clk To Pps Polarity Register	0x00000008	RW
Pps ClkToPpsVersion Reg	Pps Clk To Pps Version Register	0x0000000C	RO
Pps ClkToPpsPulseWidth Reg	Pps Clk To Pps Pulse Width Register	0x00000010	RW
Pps ClkToPpsClkFrequency Reg	Pps Clk To Pps Clock Frequency Register	0x00000020	RW

Table 4: Register Set Overview

3.2 Register Descriptions

3.2.1 General

3.2.1.1 PPS Clock to PPS Control Register

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

<table>
<tr><th colspan="32">PPS ClockToPpsControl Reg</th></tr>
<tr><th colspan="32">Reg Description</th></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="31">-</td><td>ENABLE</td></tr>
<tr><td colspan="31">RO</td><td>RW</td></tr>
<tr><td colspan="32">Reset: 0x00000000</td></tr>
<tr><td colspan="32">Offset: 0x0000</td></tr>
</table>

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

3.2.1.2 PPS Clock to PPS Status Register

Shows the current status of the PPS Clock to PPS core.

<table>
<tr><th colspan="32">PPS ClockToPpsStatus Reg</th></tr>
<tr><td colspan="32">Reg Description</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="31"></td><td>INPUT_CLK_ERROR</td></tr>
<tr><td colspan="31">RO</td><td>WC</td></tr>
<tr><td colspan="32">Reset: 0x00000000</td></tr>
<tr><td colspan="32">Offset: 0x0004</td></tr>
</table>

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:1	RO
INPUT_CLK_ERROR	Input Frequency Error, either not running or not within 12.5% of the configured frequency (sticky)	Bit: 0	WC

3.2.1.3 PPS Clock to PPS Polarity Register

Used for setting the output polarity of the PPS, shall only be done when disabled. Default value is set by the OutputPolarity_Gen generic.

PPS ClockToPpsPolarity 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 Clock to PPS 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 ClockToPpsVersion 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 Clock to PPS Pulse Width Register

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

<table>
<tr><th colspan="32">PPS ClockToPpsPulseWidth Reg</th></tr>
<tr><td colspan="32">Reg Description</td></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td colspan="22">-</td><td colspan="10">PULSE_WIDTH</td></tr>
<tr><td colspan="22">RO</td><td colspan="10">RW</td></tr>
<tr><td colspan="32">Reset: OutputPulseWidthMillsecond_Gen</td></tr>
<tr><td colspan="32">Offset: 0x0010</td></tr>
</table>

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:10	RO
PULSE_WIDTH	Generated pulse width of PPS in milliseconds	Bit: 9:0	RW

3.2.1.6 PPS Clock to PPS Input Frequency Register

This register allows to configure the Input Clock Frequency.

Pps ClkToPpsClkFrequency 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
CLK_FREQUENCY																															
RW																															
Reset: InputFrequencyHz_Gen																															
Offset: 0x0020																															

Name	Description	Bits	Access
CLK_FREQUENCY	Nominal Clock Frequency of the Input Clock in Hz	Bit: 31:0	RW

4 Design Description

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

4.1 Top Level – PPS Clock To Pps

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
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
OutputPolarity_Gen	boolean	1	true = high active, false = low active
PulseWidthDynamic Support_Gen	boolean	1	If the PPS duty cycle can be changed dynamically
OutputPulseWidth Millsecond_Gen	natural	1	PPS Pulse width in Milliseconds
FrequencyDynamic Support_Gen	boolean	1	If the input Frequency can be configured dynamically
InputFrequency Hz_Gen	natural	1	Input Clock Frequency in Hz
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 5: Parameters

4.1.1.2 Structured Types

4.1.1.2.1 Pps_ClockToPpsStaticConfig_Type

Defined in Pps_ClockToPpsAddrPackage.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
PulseWidth	std_logic_vector	10	PPS Pulse width in Milliseconds
ClkFrequency	std_logic_vector	32	Input Clock Frequency in Hz

Table 6: Pps_ClockToPpsStaticConfig_Type

4.1.1.2.2 Pps_ClockToPpsStaticConfigVal_Type

Defined in Pps_ClockToPpsAddrPackage.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 7: Pps_ClockToPpsStaticConfigVal_Type

4.1.1.2.3 Pps_ClockToPpsStaticStatus_Type

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

Field Name	Type	Size	Description
CoreInfo	Clk_CoreInfo_Type	1	Info about the Cores state

UtcInfo	Clk_UtcInfo_ Type	1	UTC Info about the Cores state

Table 8: Pps_ClockToPpsStaticConfig_Type

4.1.1.2.4 Pps_ClockToPpsStaticStatusVal_Type

Defined in Pps_ClockToPpsAddrPackage.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
UtcInfo_Val	std_logic	1	UTC Info valid

Table 9: Pps_ClockToPpsStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

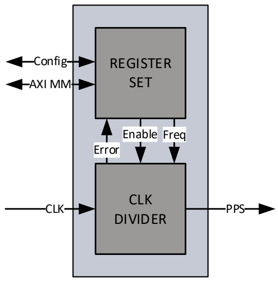

Figure 4: PPS Clock to PPS core

4.1.1.4 Entity Description

Clock Divider

This module handles the incoming clock signal and generates a PPS. parallel it supervises the input signal for the correct frequency.
See **Fehler! Verweisquelle konnte nicht gefunden werden.** for more details.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows configuring the PPS Clock to PPS core. 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				
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
OutputPolarity_Gen	-	boolean	1	True: High active, False: Low active

PulseWidthDynamic Support_Gen	-	boolean	1	If the PPS duty cycle can be changed dynamically
OutputPulseWidth Millsecond_Gen	-	natural	1	PPS Pulse width in Milliseconds
FrequencyDynamic Support_Gen	-	boolean	1	If the input Frequency can be configured dynamically
InputFrequency Hz_Gen	-	natural	1	Input Clock Frequency in Hz
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
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Pps_ClockToPps StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Pps_ClockToPps StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Pps_ClockToPps StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Pps_ClockToPps StaticStatusVal _Type	1	Static Status valid
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock

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
Clock Input				
Clk_ClkIn	in	std_logic	1	External Clock Input
Pulse Per Second Output				
Pps_EvtOut	out	std_logic	1	PPS Output

Table 10: PPS Clock to PPS core

4.2 Design Parts

The PPS Clock to PPS 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 Clock Divider

4.2.1.1 Entity Block Diagram

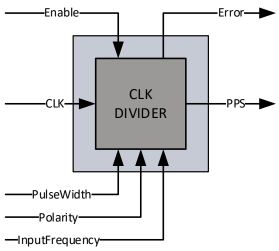

Figure 5: Clock Divider

4.2.1.2 Entity Description

Clock Divider

This module divides the input clock into a 1Hz aka Pulse Per Second signal of configurable polarity and duty cycle. In addition it checks if the input frequency is within 12.5% of the configured frequency. Only if the frequency is in range it will generate the PPS otherwise it will signal an error.
The duty cycle is provided as PulseWidth from the Registerset and allows to configure an accuracy encoding on the output PPS.
The Input Frequency in Hz also comes from the Registerset.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
CLK Divider				
OutputPolarity_Gen	-	boolean	1	True: High active, False: Low active

PulseWidthDynamic Support_Gen	-	boolean	1	If the PPS duty cycle can be changed dynamically
OutputPulseWidth Millsecond_Gen	-	natural	1	PPS Pulse width in Milliseconds
FrequencyDynamic Support_Gen	-	boolean	1	If the input Frequency can be configured dynamically
InputFrequency Hz_Gen	-	natural	1	Input Clock Frequency in Hz
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
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 Polarity				
PpsPolarity_DatIn	in	std_logic	1	'1': High active, '0': Low active
Pulse Per Second Width Input				
PpsPulseWidth_DatIn	in	std_logic_vector	10	0-999 in millisecond marks the duty cycle of the incoming PPS
Frequency Input				
PpsClkFrequency _DatIn	in	std_logic_vector	32	Expected Frequency of the input Clock in Hz
Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the correction and supervision

Pulse Per Second Output				
Pps_EvtOut	out	std_logic	1	PPS output

Table 11: Clock DIvider

4.2.2 Registerset

4.2.2.1 Entity Block Diagram

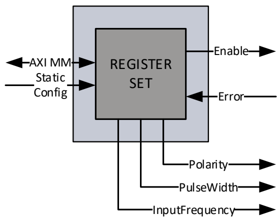

Figure 6: Registerset

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 Clock to PPS core. 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				
Clk To Pps				
OutputPolarity_Gen	-	boolean	1	True: High active, False: Low active
PulseWidthDynamic Support_Gen	-	boolean	1	If the PPS duty cycle can be changed dynamically

OutputPulseWidth Millsecond_Gen	-	natural	1	PPS Pulse width in Milliseconds
FrequencyDynamic Support_Gen	-	boolean	1	If the input Frequency can be configured dynamically
InputFrequency Hz_Gen	-	natural	1	Input Clock Frequency in Hz
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_ClockToPps StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Pps_ClockToPps StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Pps_ClockToPps StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Pps_ClockToPps 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 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 Error Input				
Pps_ErrIn	in	std_logic_vector	2	Indicates an error of the Input Frequency
Pulse Per Second Polarity				
PpsPolarity_DatOut	out	std_logic	10	'1': High active, '0': Low active
Pulse Per Second Width Output				
PpsPulseWidth_DatOut	out	std_logic_vector	10	0-999 in millisecond marks the duty cycle of the generated PPS
Pulse Per Second Cable Delay Output				
PpsClkFrequency_DatOut	out	std_logic_vector	32	Input Clock Frequency in Hz
Enable Output				

PpsClkToPps Enable_DatOut	out	std_logic	1	Enables the PPS generation

Table 12: 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_ClockToPpsStaticConfig_Type := (
  Polarity                     => '1',
  PulseWidth                   => std_logic_vector(to_unsigned(100, 10)) --100 ms
  CableDelay                   => std_logic_vector(to_unsigned(10000000, 32)) --10 MHz
);

constant PpsStaticConfigValSlave_Con : Pps_ClockToPpsStaticConfigVal_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 Clock to PPS core is 0x10000000.

```
-- PPS CLK TO PPS
-- Config
-- Set polarity to high active
AXI WRITE 10000008 00000001
-- Set Duty Cycle to 100ms
AXI WRITE 10000020 00000064
-- Set frequency to 10MHz (10000000Hz)
AXI WRITE 10000020 00989680
-- enable PPS CLK TO PPS
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 Clock to PPS core, 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 Clock to PPS runs on as well as the counter clock etc.
CLK Interface		
Input Clock	100 Hz - 100 MHz	External Input Clock
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 13: 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 14: 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 and frequency support)	158	187	0	0
Maximal (Dynamic pulse width and frequency support)	258	481	0	0

Table 15: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (No Dynamic pulse width and frequency support)	155	194	0	0
Maximal (Dynamic pulse width and frequency support)	280	376	0	0

Table 16: 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 Clock to PPS 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 CLK port as reference and the signal from the DUT. 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.

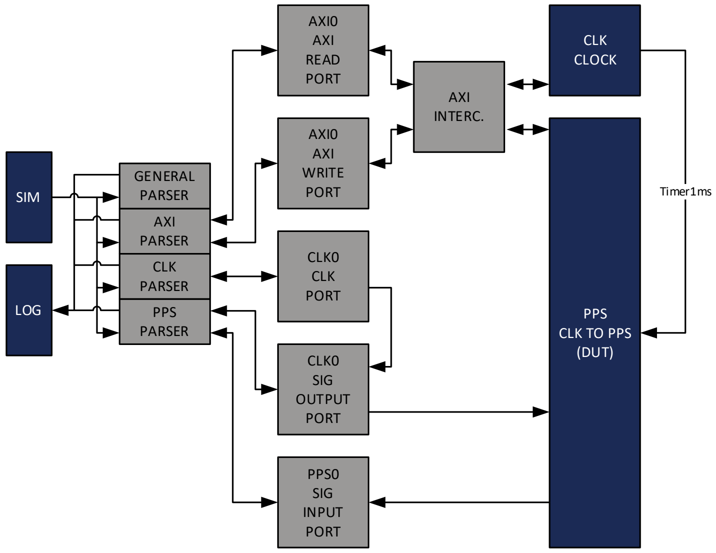

Figure 9: Testbench Framework

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/PpsClockToPps/Script/run_Pps_ClockToPps_Tb.tcl
```

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

8 Reference Designs

The PPS Clock to PPS reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the PPS Clock to PPS IP core, PPS Slave IP core (needs to be purchased separately) and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately).
The Reference Design is intended to be connected to any Clock Source. The Phase and Frequency is corrected via the PPS Clock to PPS core and the internal generate PPS fed to the PPS Slave core. An uncompensated PPS is directly generated out of the MSB of the Time.

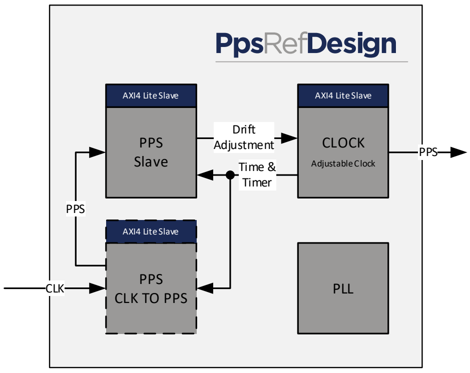

Figure 10: Reference Design

8.1 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 8.2.
2. Run TCL script
   /PPS/Refdesign/Xilinx/ArtyA7/PpsClockToPps/PpsClockToPps.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. Download to FPGA via JTAG

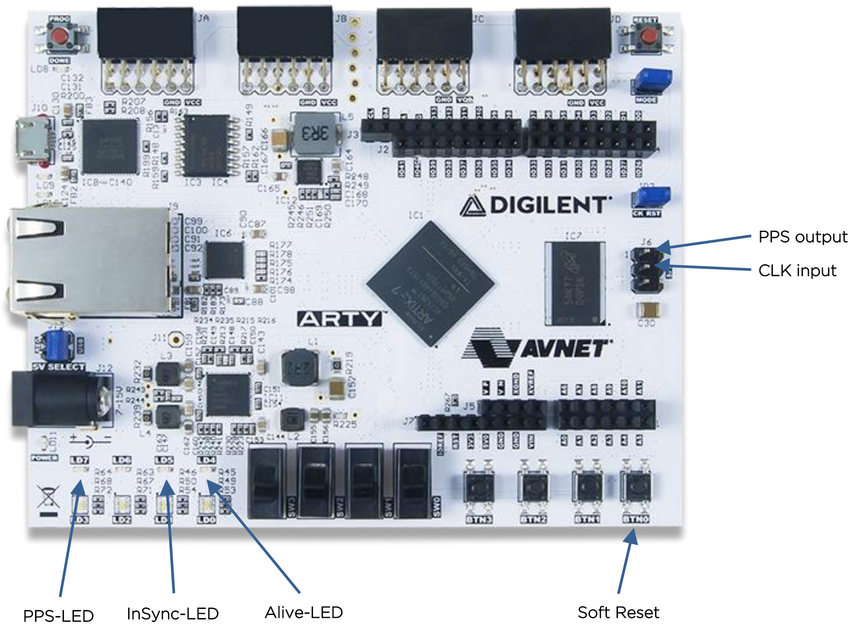

Figure 11: Arty (source Digilent Inc)

8.2 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