ClockSignal Generator

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.7
Date	05.08.2024

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 Generator is a full hardware (FPGA) only implementation of a Signal Generator. It allows to generate pulse width modulated (PWM) signals of configurable polarity aligned with the local clock. The Signal generator takes a start time, a pulse width and period as well as a repeat count as input and generates the signal accordingly. The settings can be configured either by signals or by an AXI4Lite-Slave Register interface.

Key Features:

- Pulse Width Modulated (PWM) signal generation
- 32 bit second and 32 bit nanosecond start time, pulse width and period
- Configurable polarity
- Output delay compensation
- Automatic disabling of generating on time jumps
- Repeat count
- AXI4Lite register set or static configuration
- Generator resolution with 50 MHz system clock: 20ns
- Optional High Resolution Generation with 4ns resolution
- Optional DTC Generation with 1ns resolution
- Interrupt on invalid reference
- 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	15.09.2017	Status register added
1.2	30.11.2017	Changed registerset, added cable delay
1.3	09.03.2018	Added Interrupt and shifted register addresses
1.4	09.03.2018	Added Driver
1.5	03.01.2023	Added Vivado upgrade version description
1.6	16.04.2024	Added DTC
1.7	05.08.2024	Fixed AXI Initialization example (Addresses)

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
DTC	Digital-to-Time Converted
CLK	Clock
CC	Counter Clock
TB	Testbench
LUT	Look Up Table
FF	Flip Flop
PWM	Pulse Width Modulation
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 Generator is meant as a co-processor handling Pulse Width Modulated (PWM) signal generation.
It takes a (synchronized) time input as reference and generates the PWM signal aligned with this clock (start time, pulse width and period) compensating the output delay.
The Signal Generator 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 Signal Generator can also be configured statically via signals/constants directly from the FPGA.

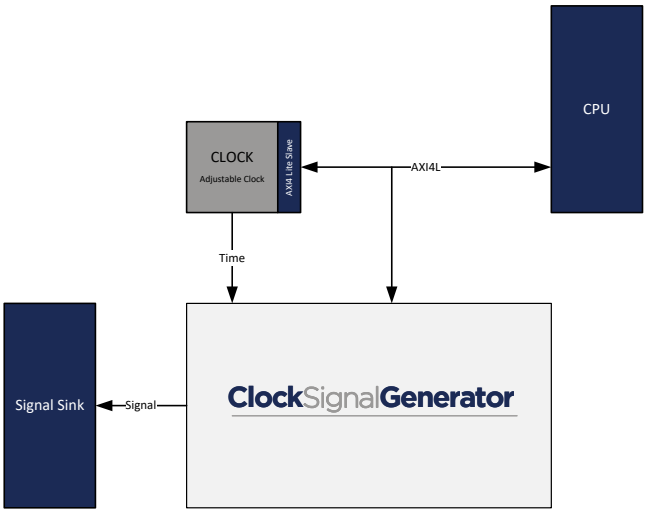

Figure 1: Context Block Diagram

1.2 Function

The Signal Generator is, as the name says already, a signal generator which generates a pulse width modulated signal aligned with a reference clock.
It uses several different inputs to generate the desired signal pattern: a start time which defines the first edge of the PWM signal, a pulse width which defines the duty cycle of the signal, a period which defines the period in case of a repeating signal and a pulse count which defines the number of pulses to be generated (or continuous).

When the Signal Generator is enabled and the new input values are set, it registers the values and calculates the next start time (beginning of pulse) and stop time (end of pulse). The start time is calculated the following way: input start time minus the output delay. The stop time is calculated the following way: calculated start time plus the pulse width.
When the start time is reached (equal or bigger) the pulse is asserted the configured polarity and a new start time is calculated by adding to the current start time the period. When the stop time is reached (also equal or bigger) the pulse is asserted to the inverse of the configured polarity, the stop time calculated by also adding the period and a pulse counter incremented.
This start/stop procedure is repeated until either the pulse count is reached or continuously repeated when the pulse count is set to zero.
When a time jump happens the pulse generation is immediately disabled and the output signal set to the idle level of the polarity. This is required because of the start/stop time calculations. This also means that a start time must be always in the future when the Signal Generator is configured. Disabling of the pulse generation can also enforced via the register set, the behavior will be the same as with a time jump.

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.

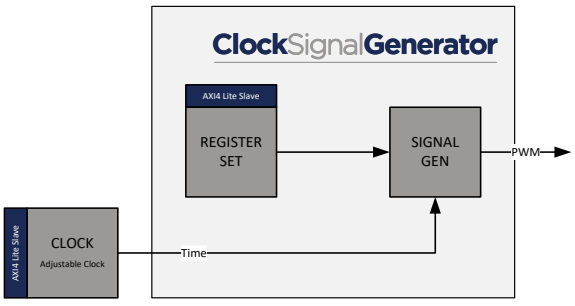

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

Signal Generator

This block is the actual generator. It takes the reference time and creates the PWM signal based on the start time, pulse width and period aligned with the clock.

2 Signal Generation 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.

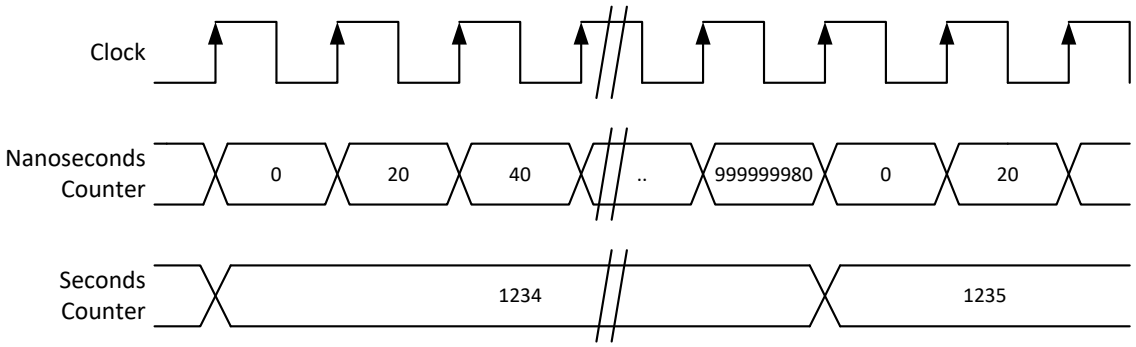

Figure 3: Counter Clock

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 Generation

For generating a Pulse Width Modulated signal, the following values are needed: a start time when the first edge of the signal shall be generated, a pulse width which defines the duty cycle and a period. When generating a signal one delay has to be taken into account which is the output delay. The signal generator has to generate the signal earlier to compensate for the output delay.
Also the frequency and therefore quantization of the clock is important. It in the end limits the resolution and therefore accuracy of the generated signal. When an edge shall be generated, 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 generation 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 signal generation can be used and fractions taken into account to correct the signal.

Figure 4: shows exactly the delay which is occurring when generating the signal. You can see that the internal signal is generated earlier so the first rising edge is exactly at the second's boundary.

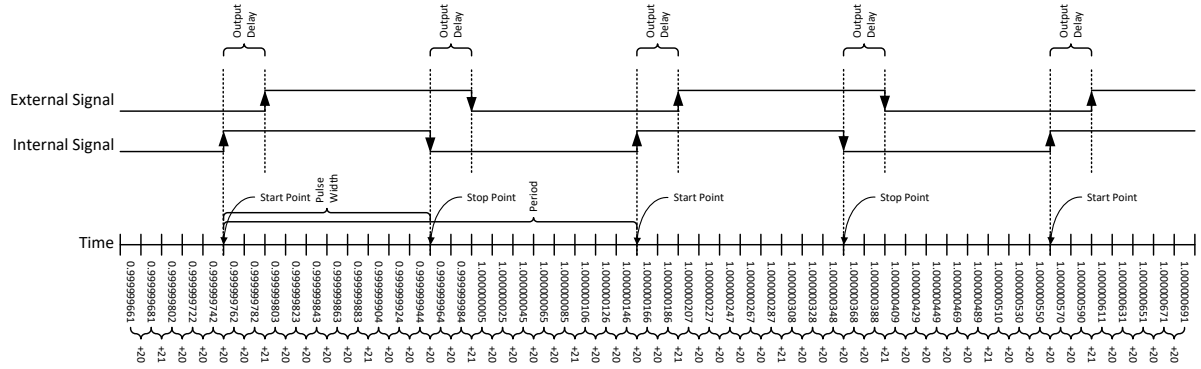

Figure 4: Time Stamping

3 Register Set

This is the register set of the Signal Generator. 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 SgControl Reg	Clock Signal generation Valid and Enabled Control Register	0x00000000	RW
Clk SgStatus Reg	Clock Signal generation Status Register	0x00000004	WC
Clk SgPolarity Reg	Clock Signal generation Polarity Register	0x00000008	RW
Clk SgVersion Reg	Clock Signal generation Version Register	0x0000000C	RO
Clk SgCableDelay Reg	Clock Signal generation Cable Delay Register	0x00000020	RW
Clk SgIrq Reg	Clock Signal generation Interrupt Register	0x00000030	WC
Clk SgIrqMask Reg	Clock Signal generation Interrupt Mask Register	0x00000034	RW
Clk SgStartTimeValueL Reg	Clock Signal generation Start Time Nanosecond Register	0x00000040	RW
Clk SgStartTimeValueH Reg	Clock Signal generation Start Time Second Register	0x00000044	RW
Clk SgPulseWidthValueL Reg	Clock Signal generation Pulse Width Time Nanosecond Register	0x00000048	RW
Clk SgPulseWidthValueH Reg	Clock Signal generation Pulse Width Time Second Register	0x0000004C	RW
Clk SgPeriodValueL Reg	Clock Signal generation Period Time Nanosecond Register	0x00000050	RW
Clk SgPeriodValueH Reg	Clock Signal generation Period Time Second Register	0x00000054	RW
Clk SgRepeatCount Reg	Clock Signal generation Repeat Counter Register	0x00000058	RW

3.2 Register Descriptions

3.2.1 General

3.2.1.1 CLK Signal Generator Control Register

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

Clk SgControl 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
RO																														RW	RW
Reset: 0x00000000																															
Offset: 0x0000																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:2	RO
SIGNAL_VAL	Signal generation values valid	Bit: 1	RW

ENABLE	Enable	Bit: 0	RW

3.2.1.2 CLK Signal Generator Status Register

Used for status supervision if a time jump has happened and/or if a generator error occurred. If a generator error occurred generation is immediately stopped and the generation has to be started again with a new start time

Clk SgStatus 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
																														TIME_JUMP	ERROR
RO																														WC	WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:2	RO
TIME_JUMP	A time jump occurred	Bit: 1	WC
ERROR	A generator error occurred (disable during gen or time jump or new start time not in the future)	Bit: 0	WC

3.2.1.3 CLK Signal Generator Polarity Register

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

Clk SgPolarity 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 Generator 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 SgVersion 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 Generator 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 SgCableDelay 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 Generator Interrupt Register

Shows the interrupt state. 1 means interrupt asserted. When the interrupt was asserted, signal generation is disabled.

Clk SgIrq 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, Signal generation got disabled	Bit: 0	WC

3.2.1.7 CLK Signal Generator Interrupt Mask Register

Enabled and disable the interrupt generation. 1 means interrupt enabled.

Clk SgIrqMask 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	Bit: 0	RW

3.2.1.8 CLK Signal Generator Start Time Value Low Register

Start time value nanosecond part.

Clk SgStartTimeValueL 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
START_TIME_NS																															
RW																															
Reset: 0x00000000																															
Offset: 0x0040																															

Name	Description	Bits	Access
START_TIME_NS	Start Time in Nanosecond (must be larger than current time)	Bit: 31:0	RW

3.2.1.9 CLK Signal Generator Start Time Value High Register

Start time value second part.

Clk SgTimeValueH 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
START_TIME_S																															
RW																															
Reset: 0x00000000																															
Offset: 0x0044																															

Name	Description	Bits	Access
START_TIME_S	Start Time in Second (must be larger than current time)	Bit: 31:0	RW

3.2.1.10 CLK Signal Generator Pulse Width Value Low Register

Pulse width nanoseconds part value.

Clk SgPulseWidthValueL 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_NS																															
RW																															
Reset: 0x00000000																															
Offset: 0x0048																															

Name	Description	Bits	Access
PULSE_NS	Pulse width in Nanosecond	Bit: 31:0	RW

3.2.1.11 CLK Signal Generator Pulse Width Value High Register

Pulse width seconds part value.

Clk SgPulseWidthValueH 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_S																															
RW																															
Reset: 0x00000000																															
Offset: 0x004C																															

Name	Description	Bits	Access
PULSE_S	Pulse width in Second	Bit: 31:0	RW

3.2.1.12 CLK Signal Generator Period Value Low Register

Pulse period nanoseconds part value.

Clk SgPeriodValueL 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
PERIOD_NS																															
RW																															
Reset: 0x00000000																															
Offset: 0x0050																															

Name	Description	Bits	Access
PERIOD_NS	Period in Nanosecond	Bit: 31:0	RW

3.2.1.13CLK Signal Generator Period Value High Register

Pulse period seconds part value.

Clk SgPeriodValueH 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
PERIOD_S																															
RW																															
Reset: 0x00000000																															
Offset: 0x0054																															

Name	Description	Bits	Access
PERIOD_S	Period in Second	Bit: 31:0	RW

3.2.1.14 CLK Signal Generator Repeat Count Register

Repeat count of the pulse generation.

Clk SgRepeatCount 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
REPEAT_COUNT																															
RW																															
Reset: 0x00000000																															
Offset: 0x0058																															

Name	Description	Bits	Access
REPEAT_COUNT	How often the signal pattern shall be repeated (0 continous)	Bit: 31:0	RW

4 Design Description

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

4.1 Top Level – Clk SignalGenerator

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
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
CableDelay_Gen	boolean	1	If a cable delay shall be configurable (only needed when connected externaly)
OutputDelay Nanosecond_Gen	natural	1	Output delay of the signal from the output signal to the connector.
OutputPolarity_Gen	boolean	1	true = high active, false = low active
HighResSupport_Gen	boolean	1	If a high-resolution clock shall be used.
HighResFreqMulti-ply_Gen	Natural range [4-10]	1	The high-resolution clock frequency is a multiple of the system clock's frequency. Default is 5.
DtcSupport_Gen	boolean	1	If DTC is supported
DtcCarryDelay Femtosecond_Gen	natural	1	Delay of a Carry element
DtcOutputDelay	natural	1	Delay from the Carry to the IO

Picoseconds_Gen			Pin
DtcFixPosition_Gen	boolean	1	If the position of the DTC shall be fixed in the design (Xilinx only)
DtcXPosition_Gen	natural	1	DTC Start position Slice X position. Area is (X-1) - (X+1)
DtcYPosition_Gen	natural	1	DTC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
AxiAddressRange Low_Gen	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	std_logic_vector	32	AXI Base Address plus Regis-terset 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_SignalGeneratorStaticConfig_Type

Defined in Clk_SignalGeneratorAddrPackage.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
StartTime	Clk_Time_Type	1	Time when the signal generation shall be started
PulseWidth	Clk_Time_Type	1	Duty cycle time
Period	Clk_Time_Type	1	Period of the signal generation
RepeatCount	std_logic_vector	32	How often the pulse pattern shall be generated

Table 6: Clk_SignalGeneratorStaticConfig_Type

4.1.1.2.3 Clk_SignalGeneratorStaticConfigVal_Type

Defined in Clk_SignalGeneratorAddrPackage.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 Clock
Signal_Val	std_logic	1	Enables the generating with the values from the config

Table 7: Clk_SignalGeneratorStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

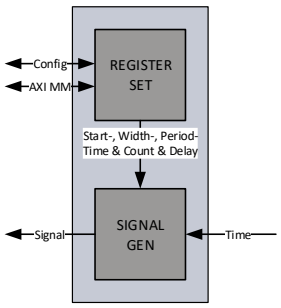

Figure 5: Signal Generator

4.1.1.4 Entity Description

Signal Generator

This module generates the signal pattern at the configured start time with the configured duty cycle and period for the configured number of cycles. Signal generation is aligned with the reference time. It receives the pattern configuration from the Registerset module.
See 4.2.1 for more details.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows to configure the Signal Generator. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the signal pattern 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 signal pattern 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
CableDelay_Gen	-	boolean	1	If a cable delay shall be configurable (only needed when connected externaly)
OutputDelay Nanosecond_Gen	-	natural	1	Output delay of the signal from the output signal to the connector
OutputPolarity_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
DtcSupport_Gen	-	boolean	1	If DTC is supported
DtcCarryDelay Femtosecond_Gen	-	natural	1	Delay of a Carry element
DtcOutputDelay Picoseconds_Gen	-	natural	1	Delay from the Carry to the IO Pin
DtcFixPosition_Gen	-	boolean	1	If the position of the DTC shall be fixed in the design (Xilinx

				only)
DtcXPosition_Gen	-	natural	1	DTC Start position Slice X position. Area is (X-1) - (X+1)
DtcYPosition_Gen	-	natural	1	DTC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
AxiAddressRangeLow_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRangeHigh_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	Clk_SignalGenerator StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_SignalGenerator StaticConfigVal_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	in	std_logic_vector	32	Write Address

_AdrIn				
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 Re-sponse
Interrupt Output				
Irq_EvtOut	out	Clk_Clock StaticConfig_Type	1	Active high level interrupt
Signal Output				
SignalGenerator _EvtOut	out	std_logic	1	Output signal for PWM generation

Table 8: Signal Generator

4.2 Design Parts

The Signal Generator 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 Generator

4.2.1.1 Entity Block Diagram

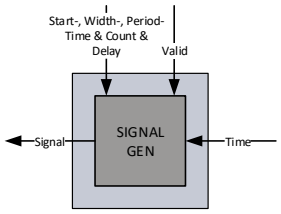

Figure 6: Signal Generator

4.2.1.2 Entity Description

Mux Selector

This module generates the PWM signal. When the Signal Generator is enabled and the new input values are set, it registers the values and calculates the next start time (beginning of pulse) and stop time (end of pulse). The start time is calculated the following way: input start time minus the output delay. The stop time is calculated the following way: calculated start time plus the pulse width.
When the start time is reached (equal or bigger) the pulse is asserted the configured polarity and a new start time is calculated by adding to the current start time the period. When the stop time is reached (also equal or bigger) the pulse is asserted to the inverse of the configured polarity, the stop time calculated by also adding the period and a pulse counter incremented.
This start/stop procedure is repeated until either the pulse count is reached or continuously repeated when the pulse count is set to zero.
When a time jump happens the pulse generation is immediately disabled and the output signal set to the idle level of the polarity. This is required because of the start/stop time calculations. This also means that a start time must be always in the future when the Signal Generator is configured. Disabling of the pulse generation can also enforced via the register set, the behavior will be the same as with a time jump.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
CableDelay_Gen	-	boolean	1	If a cable delay shall be configurable (only needed when connected exter-naly)
OutputDelay Nanosecond_Gen	-	natural	1	Output delay of the signal from the output signal to the connector
OutputPolarity_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
DtcSupport_Gen	-	boolean	1	If DTC is supported
DtcCarryDelay Femtosecond_Gen	-	natural	1	Delay of a Carry element
DtcOutputDelay Picoseconds_Gen	-	natural	1	Delay from the Carry to the IO Pin
DtcFixPosition_Gen	-	boolean	1	If the position of the DTC shall be fixed in the design (Xilinx only)
DtcXPosition_Gen	-	natural	1	DTC Start position Slice X position. Area is (X-1) - (X+1)

DtcYPosition_Gen	-	natural	1	DTC Start position Slice Y position. Area is (Y-1) - (Y+NrOfCarries)
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable the Generator
Error Output				
Generate_ErrOut	out	std_logic	1	Generator expected an error
Signal Values Input				
SignalStartTime_DatIn	in	Clk_Time_Type	1	Signal pattern first start time
SignalPulseWidth_DatIn	in	Clk_Time_Type	1	Signal pattern duty cycle
SignalPeriod_DatIn	in	Clk_Time_Type	1	Signal pattern period
SignalRepeatCount_DatIn	in	std_logic_vector	32	Number of cycles that the pattern shall be generated. 0 = infinite
SignalPolarity_DatIn	in	std_logic	1	'1': High active, '0': Low active
SignalCableDelay_DatIn	in	std_logic_vector	16	Delay in Nanoseconds
Signal_ValIn	in	std_logic	1	Pattern values are valid
Signal Output				
SignalGenerator_EvtOut	out	std_logic	1	Output signal for PWM generation

Table 9: Signal Generator

4.2.2 Registerset

4.2.2.1 Entity Block Diagram

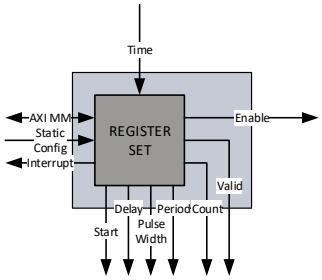

Figure 7: Registerset

4.2.2.2 Entity Description

Register Set

This module is an AXI4Lite Memory Mapped Slave. It provides access to the signal pattern registers and allows configuring the Signal Generator. 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 signal pattern 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 core has to be disabled and enabled again. If in AXI mode, an AXI Master has to configure the signal pattern with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.
It can also generate a timestamp whenever the reference time gets invalid, meaning the time made a jump or is invalidated. This is required since the generator stops generating when the reference time gets invalid.

4.2.2.3 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 exter-naly)
OutputPolarity_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
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Clk_SignalGenerator StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_SignalGenerator StaticConfigVal _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
Error Input				
Generate_ErrIn	in	std_logic	1	Generator expected an error
Signal Values Output				
SignalStart Time_DatOut	out	Clk_Time_Type	1	Signal pattern first start time
SignalPulseWidth_DatOut	out	Clk_Time_Type	1	Signal pattern duty cycle
SignalPeriod_DatOut	out	Clk_Time_Type	1	Signal pattern period
SignalRepeatCount_DatOut	out	std_logic_vector	32	Number of cycles that the pattern shall be generated. 0 = infinite
SignalPolarity_DatOut	out	std_logic	1	'1': High active, '0': Low active
SignalCableDelay_DatOut	out	std_logic_vector	16	Delay in Nanoseconds
Signal_ValOut	out	std_logic	1	Pattern values are

				valid
Interrupt Output				
Irq_EvtOut	out	Clk_Clock StaticConfig_Type	1	Active high level interrupt
Enable Output				
GenerateEnable _DatOut	out	std_logic	1	Enable Signal Generator

Table 10: 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 ClkStaticConfigSignalGenerator_Con : Clk_SignalGeneratorStaticConfig_Type := (
    Polarity                        => '1',
    StartTime                       => (
      Second                        => std_logic_vector(to_unsigned(5, 32)),
      Nanosecond                    => std_logic_vector(to_unsigned(300, 32)),
      Fraction                      => (others => '0'),
      Sign                          => '0',
      TimeJump                      => '0'),
    PulseWidth                      => (
      Second                        => std_logic_vector(to_unsigned(1, 32)),
      Nanosecond                    => std_logic_vector(to_unsigned(0, 32)),
      Fraction                      => (others => '0'),
      Sign                          => '0',
      TimeJump                      => '0'),
    Period                          => (
      Second                        => std_logic_vector(to_unsigned(2, 32)),
      Nanosecond                    => std_logic_vector(to_unsigned(0, 32)),
      Fraction                      => (others => '0'),
      Sign                          => '0',
      TimeJump                      => '0'),
    RepeatCount                     => std_logic_vector(to_unsigned(0, 32)) -- infinite
  );

  constant ClkStaticConfigValSignalGenerator_Con : Clk_SignalGeneratorStaticConfigVal_Type :=
(
    Enable_Val                      => '1',
    Signal_Val                      => '1'
  );
```

Figure 8: Static Configuration

The signal generation pattern values can be changed while Signal_Val is set to '0'.

4.3.2 AXI Configuration

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

```
-- CLK SIGNALGENERATOR
-- Polarity = 1
AXI WRITE 10000008 00000001
-- Start time nanoseconds = 300
AXI WRITE 10000040 0000012C
-- Start time seconds = 5
AXI WRITE 10000044 00000005
-- Pulse time nanoseconds = 0
AXI WRITE 10000048 00000000
-- Pulse time seconds = 1
AXI WRITE 1000004C 00000001
-- Period time nanoseconds = 0
AXI WRITE 10000050 00000000
-- Period time seconds = 2
AXI WRITE 10000054 00000002
-- Repeat Count = 0 (infinite)
AXI WRITE 10000058 00000000
-- Write values and enable signal generator
AXI WRITE 10000000 00000003
```

Figure 9: AXI Configuration

The Signal pattern values should be set before enabling but can also be changed when enabled. The valid bit is self clearing, but will have immediate effect.

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 frequency generation core runs on.
High Rsolution		
High Resolution Clock	250MHz (Default)	High resolution clock for more accurate assertion/deassertion of the output signal
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 11: 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 12: 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)	488	2528	0	0
Maximal (AXI)	531	2679	0	0

Table 13: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static Config)	488	2395	0	0
Maximal (AXI)	529	2567	0	0

Table 14: 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 Generator testbench consist of 3 parse/port types: AXI, CLK and SIG. The Signal Input Port is checking the generated output with the same clock reference from the Clock Port as the Signal Generator
For configuration and result checks an AXI read and write port is used.

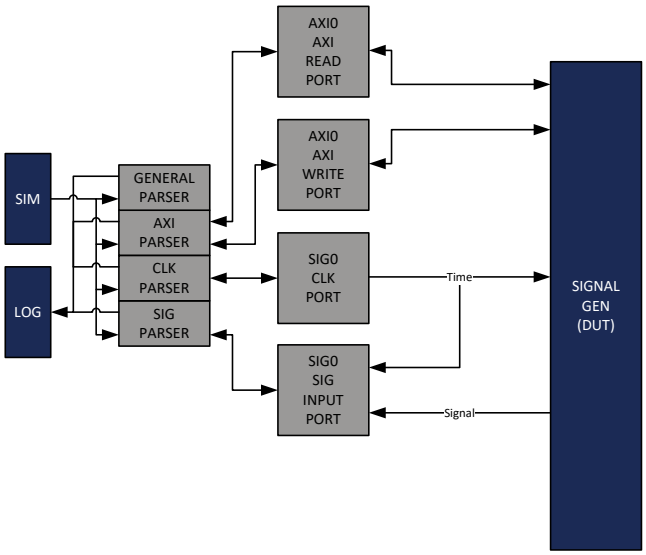

Figure 10: Testbench Framework

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/ClkSignalGenerator/Script/run_Clk_SignalGenerator_Tb.tcl
```

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

8 Reference Designs

The Signal Generator reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz) and an instance of the Signal Generator 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, show the instantiation and generate a signal output. 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.

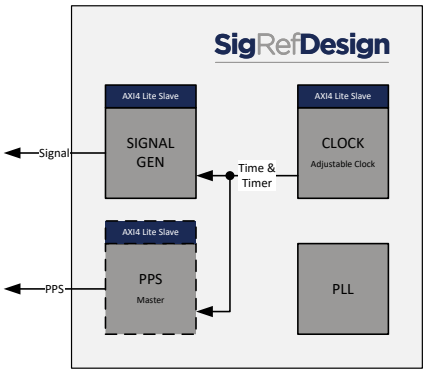

Figure 11: Reference Design

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/ClkSignalGenerator/ClkSignalGenerator.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

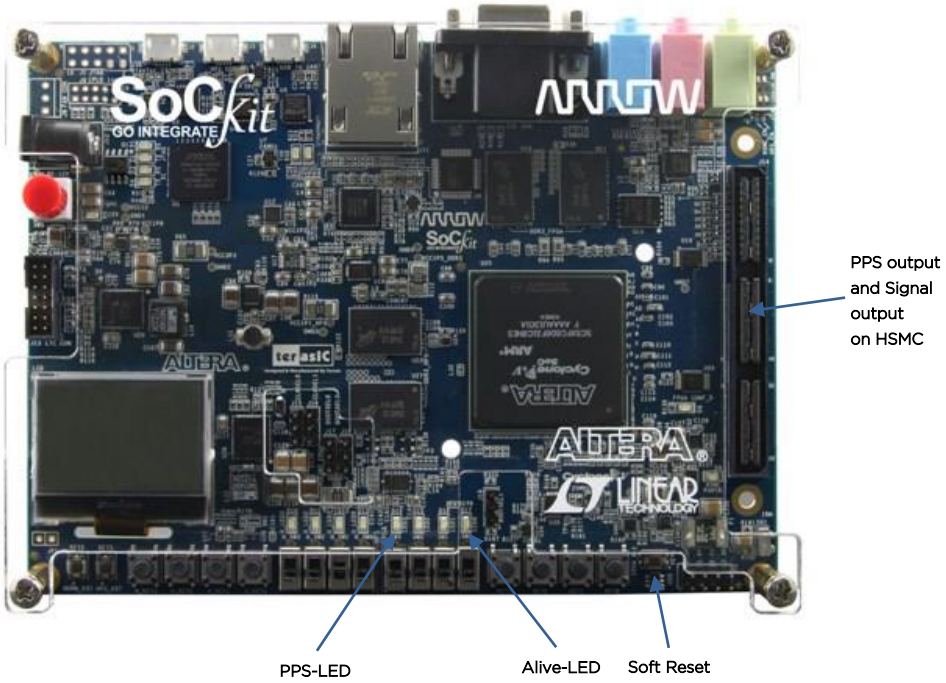

Figure 12: SocKit (source Terasic Inc)

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.
2. Note: If a different Vivado version is used, see chapter 8.3.
3. Run TCL script /CLK/Refdesign/Xilinx/Arty/ClkSignalGenerator/ClkSignalGenerator.tcl
    a. This has to be run only the first time and will create a new Vivado Project
4. If the project has been created before open the project and do not rerun the project TCL

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

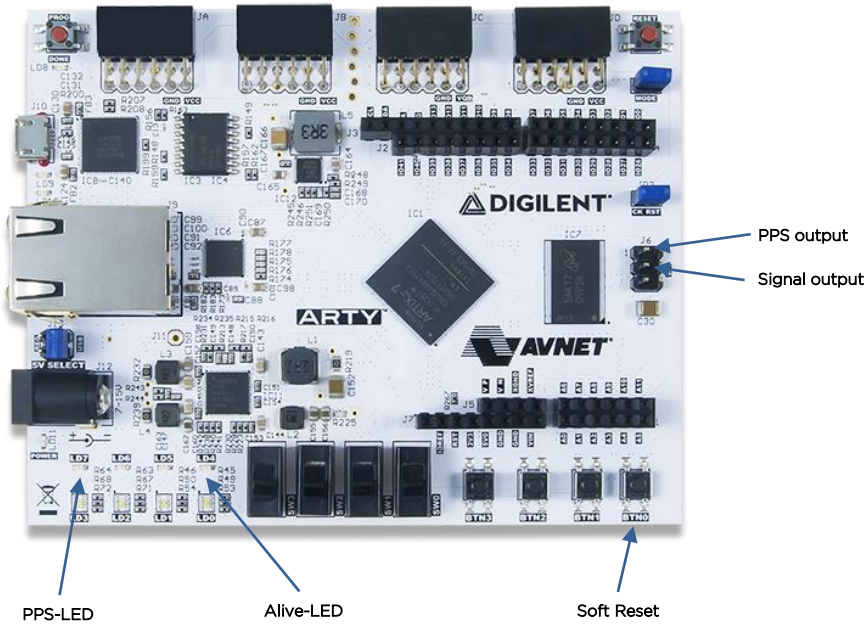

Figure 13: Arty (source Digilent Inc)

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