ClockFrequency GeneratorSine

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	Thomas Schaub
Version	0.2
Date	01.09.2025

Overview

NetTimeLogic's Frequency Generator is a full hardware (FPGA only) implementation of a Frequency Generator for Sine Waves via a DAC. It allows to generate a signal of configurable frequency and polarity aligned with the local clock as a Sine Wave. The Frequency Generator takes a frequency in Hertz as input and generates the Since Wave samples, accordingly, based on a sampling frequency. The settings can be configured either by signals or by an AXI4Lite-Slave Register interface. During synthesis time the duty cycle of the generated signal can be set as either a single pulse or 50%.

Key Features:

- Configurable frequency signal generation (1-200kHz (depends on DAC sampling rate))
- Configurable polarity (positive or negative zero crossing and PPS boundary)
- Output delay compensation
- Alignment of the frequency generator to an input time (frequency and phase)
- Automatic realigning of the frequency generator on time jumps and frequency changes
- Continuous generation until disabled
- Configurable DAC Sample width
- Configurable DAC Sampling rate
- Optional DAC Sample scaling
- Optional DAC Sample offset
- SPI DAC Controller
- AXI4Lite register set or static configuration

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	27.08.2025	First draft
0.2	01.09.2024	Added mode to ignore phase

Table 1: Revision History

Content

Definitions

Definitions	
Counter Clock	A counter-based clock that counts 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)
DAC	Digital Analog Converter
IRQ	Interrupt, Signaling to e.g. a CPU
PPS	Pulse Per Second
TS	Timestamp
CLK	Clock
CC	Counter Clock
TB	Testbench
LUT	Look Up Table
FF	Flip Flop
PPS	Pulser Per Second
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 Frequency Generator is meant as a co-processor handling sine wave generation of configurable frequency.
It takes a (synchronized) time input as reference and generates a frequency aligned with this clock (given the input frequency and polarity) compensating the output delay and converting it to a sine wave sample with configurable sampling rate .This sine wave sample shall then be written to a DAC, for this an optional SPI DAC module exists.
The Frequency 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 Frequency Generator can also be configured statically via signals/constants directly from the FPGA.

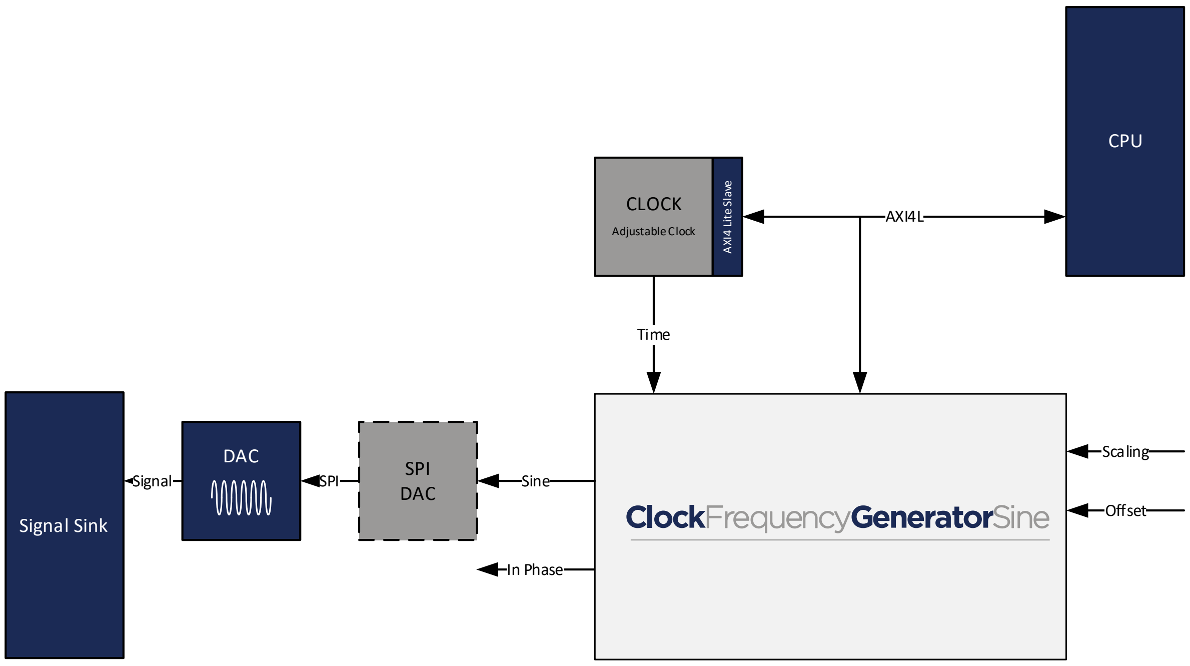

Figure 1: Context Block Diagram

1.2 Function

The Frequency Generator is a standalone core which generates a sine wave of configurable frequency aligned with a reference clock.

The frequency is provided in Hertz as input, along with the signal polarity and the cable delay of the output signal. When the Frequency Generator is enabled and the new input values are set, it registers the values and starts generating the signal of the configured frequency. At the beginning of the generation and until the beginning of the new second of the reference clock, the generated signal will have aligned frequency to the input time, but it will be out of phase. The phase will be aligned when the next new second of the input timer clock is reached. When a time jump happens the frequency generation will continue with the previous phase, until the first new second is reached. Then, the phase will also realign to the new time. Due to a phase realignment, the frequency generator might corrupt the sine wave. When a new cycle of the signal begins the pulse is asserted to the configured polarity. At a configurable sampling interval, it calculates the angle of the sine wave and uses a Cordic calculation to get the sine of the angle. This sine value is then converted into a DAC sample which can be optionally scaled and offset. The frequency generation is repeated continuously, until the core is disabled via the register set.
In addition, there is a generic SPI DAC controller which allows to feed the sine wave samples to a DAC.
.

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.

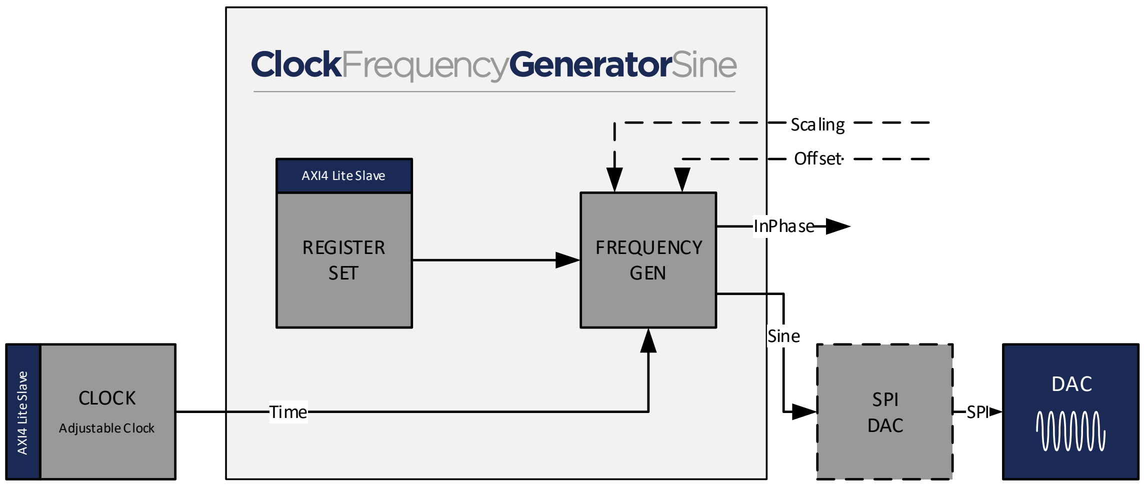

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

Frequency Generator

This block is the actual generator. It takes the reference time and creates the sine wave samples based on the configured frequency aligned with the clock.

SPI DAC

This block is a generic SPI controller which allows you to feed the samples to a DAC via SPI. It is optional since a parallel DAC could also be used.

2 Frequency Generation Basics

2.1 Digital Counter Clock

A digital counter clock is the most used type of absolute time source for digital systems. Its functionality is simple: every counter cycle it adds the period of the counter cycle to a counter value. 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 second part and a second 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.

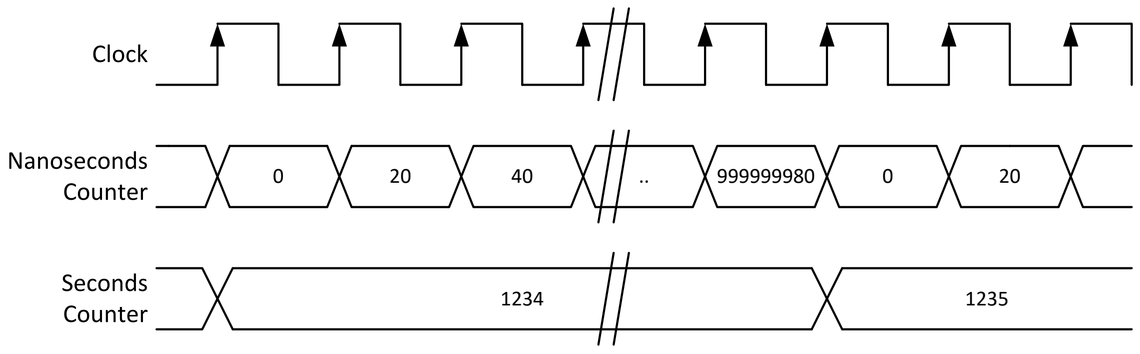

Figure 3: Counter Clock

2.2 Drift and Offset adjustments

When a digital counter clock shall be synchronized there are two things that have to be adjusted which are 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 advantage that it normally has a much higher resolution e.g. 1ns/s and it does not 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 do not want a time jump since the PWM would not be correct anymore, and this is exactly what would happen if the time were 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 must 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 does not count backwards during adjustments. Data acquisition and measurement applications require for example a strongly monolithically increasing time. This requirement 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 frequency generation the quantization fault is still the clock period but with an accurate nanosecond resolution.

2.3 Frequency Generation

For the frequency generation the following values are needed: the frequency in Hertz and the signal polarity. When generating a signal, the output delay must be considered. The frequency generator must generate the output 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. To achieve higher precision, the frequency generator can fine-tune the assertion and de-assertion of the generated signal by using a high-resolution clock that has a frequency of an integer multiple of the system clock and can also be combined with a DTC to achieve 1ns accuracy.

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. Also, the phase of the generated frequency is realigned at the arrival of the new second (by slightly truncating or extending the period of the last generated cycle).

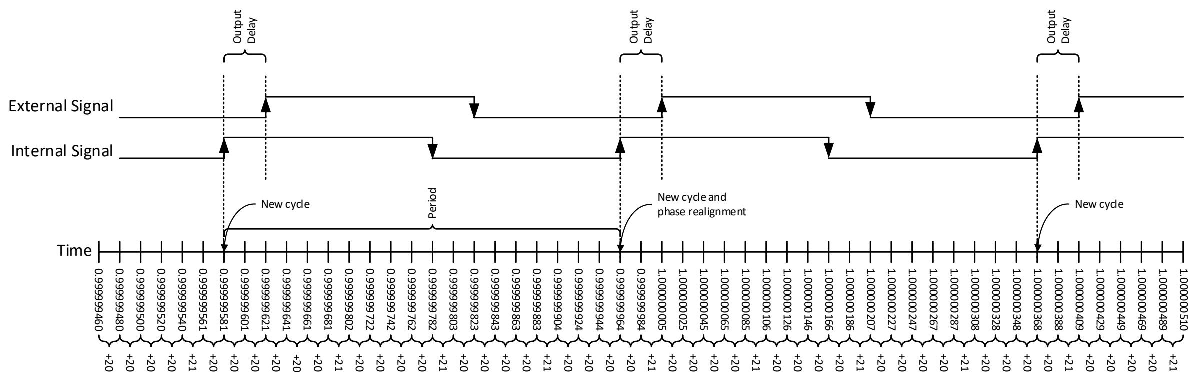

Figure 4: Frequency Generation

2.4 Sine Wave Generation

For the sine wave generation, the intermediate angle of the frequency generation is used. Between two rising edges the value range for the angle is 0-2π. There is an integrator which divides the clock period into a range of 2π. For different frequencies only the increment rate of the integrator matters since this will define the sine wave level. In this core the CORDIC algorithm with 16 increments and 20bit precision for the angle is used to calculate the sine value (-1.0 - +1.0) out of an angle in RAD. The sine value can then be converted into a DAC value which can be fed to a DAC e.g. via SPI.

Figure 5: shows exactly the angle integration and the corresponding sine wave

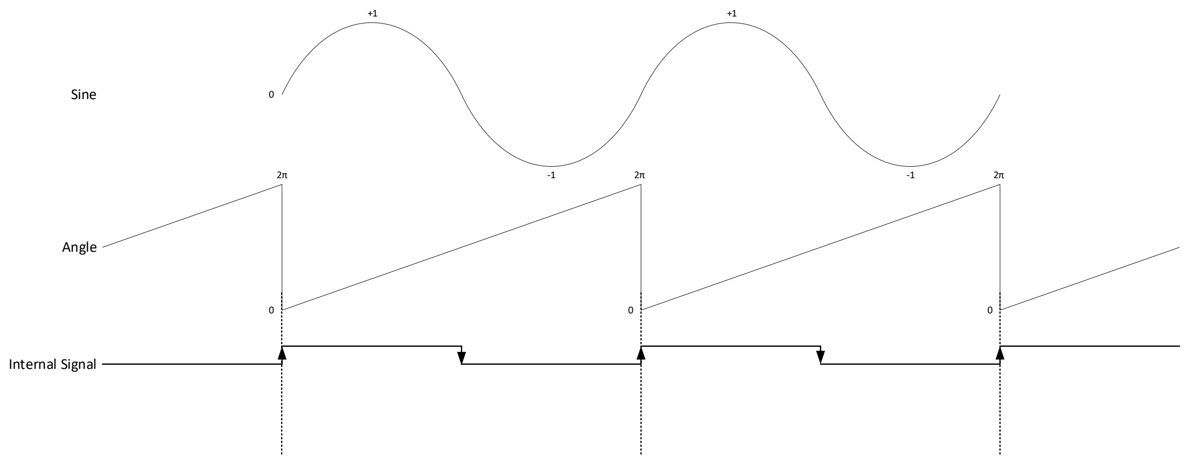

Figure 5: Sine Wave Generation

3 Register Set

This is the register set of the Frequency 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 interconnects. 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 FgControl Reg	Clock Frequency generation Valid and Enabled Control Register	0x00000000	RW
Clk FgStatus Reg	Clock Frequency generation Status Register	0x00000004	RW
Clk FgPolarity Reg	Clock Frequency generation Polarity Register	0x00000008	RW
Clk FgVersion Reg	Clock Frequency generation Version Register	0x0000000C	RO
Clk FgCableDelay Reg	Clock Frequency generation Cable Delay Register	0x00000020	RW
Clk FgFrequency Reg	Clock Frequency generation Frequency value Register	0x00000030	RW
Clk FgCyclesPerSecond Reg	Clock Frequency generation Cycles Per Second Register	0x00000034	RW

3.2 Register Descriptions

3.2.1 General

3.2.1.1 CLK Frequency Generator Control Register

Used for general control over the Frequency Generator. Set flags are available to mark validity of the configuration.

Clk FgControl Reg				
Reg Description				
31 … 4	3	2	1	0
-	IGNORE_PHASE	-	FREQUENCY_VAL	ENABLE
RO	RW	RO	RW	RW
Reset: 0x00000000				
Offset: 0x0000				

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:4	RO
IGNORE_PHASE	No phase alignment	Bit: 3	RW
-	Reserved, read 0	Bit:2	RO

FREQUENCY_VAL	Frequency generation values valid	Bit: 1	RW
ENABLE	Enable frequency generation	Bit: 0	RW

3.2.1.2 CLK Frequency Generator Status Register

Used for status supervision if the phase of the generated frequency is aligned to the input counter clock.

Clk FgStatus 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
-																							IN_PHASE_ERROR	-						SKIP_PULSE	IN_PHASE
RO																							RW	RO						RW	RW
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:9	RO
IN_PHASE_ERROR	Sticky bit of in phase error until it is cleared by the master	Bit: 8	RW
-	Reserved, read 0	Bit:7:2	RO

SKIP_PULSE	If a Waveform generation was skipped, currently Reserved, read 0.	Bit: 1	RW
IN_PHASE	The frequency generation is in phase with the received time (e.g. from the Adjustable Clock). In case of time jump the generation will be out of phase to the counter clock until a new second.	Bit: 0	RW

3.2.1.3 CLK Frequency 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 FgPolarity 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 positive waveform, 0 negative waveform)	Bit: 0	RW

3.2.1.4 CLK Frequency Generator Version Register

Version of the IP core, even though it 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 FgVersion 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 Frequency 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 FgCableDelay 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 Frequency Generator Frequency Register

The frequency to be generated in Hertz. The range is [0-16,777,215] Hz

Clk FgFrequency 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
-								FREQUENCY																							
RO								RW																							
Reset: 0x00000000																															
Offset: 0x0030																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:24	RO
FREQUENCY	Value of the frequency to be generated	Bit: 23:0	RW

3.2.1.7 CLK Frequency Generator Cycles Per Second

The register provides the number of cycles generated by the core over the last second.

Clk FgCyclesPerSecond 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
-								CYLCES_PER_SECOND																							
RO								RW																							
Reset: 0x00000000																															
Offset: 0x0034																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:24	RO
FREQUENCY	Number of cycles generated over the last second	Bit: 23:0	RW

4 Design Description

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

4.1 Top Level – Clk FrequencyGeneratorSine

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
ResetBuffer_Gen	boolean	1	If a reset buffer is used to provide reset synchronous to the system clock true = use reset buffer false = do not use reset buffer
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 con-figurable (only needed when connected externally)
OutputDelay Nanosecond_Gen	natural	1	Output delay of the signal from the output signal to the connector.
OutputPolarity_Gen	boolean	1	Polarity of the generated signal true = high active, false = low active
ScalingSupport_Gen	boolean	1	If the value shall be scaled
OffsetSupport_Gen	boolean	1	If a DC Offset can be added, Requires Scaling to be active
SampleFreqHz_Gen	natural	1	Sample Frequency in Hz from

			500kHz to 2MHz
Sample DataWidth_Gen	natural	1	Sample Data Width 8 – 32bit
SampleSigned_Gen	boolean	1	If the sample represents a signed value or unsigned value
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_FrequencyGeneratorStaticConfig_Type

Defined in Clk_FrequencyGeneratorAddrPackage.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
EmbeddedPps	std_logic	1	'1' = embedded PPS, '0' = no PPS (not used)
IgnorePhase	std_logic	1	'1' = no Phase alignment, '0' = Phase Alignment with second
CableDelay	std_logic_vector	16	Cable Delay in Nanoseconds
Frequency	Std_logic_vector	24	Frequency to be generated in Hertz

Table 6: Clk_FrequencyGeneratorStaticConfig_Type

4.1.1.2.3 Clk_FrequencyGeneratorStaticConfigVal_Type

Defined in Clk_FrequencyGeneratorAddrPackage.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 generation
Frequency_Val	std_logic	1	Validates the values from the configuration

Table 7: Clk_FrequencyGeneratorStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

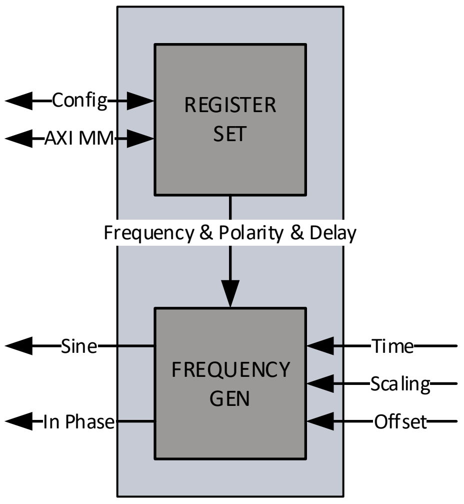

Figure 6: Frequency Generator

4.1.1.4 Entity Description

Frequency Generator

This module generates the signal with the configured frequency. Frequency generation is aligned with the reference time. It receives the configuration from the Registerset module.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows us to configure the Frequency Generator. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration is done via signals and can be easily done from within the FPGA without a CPU. If in

AXI mode, an AXI Master must 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				
ResetBuffer_Gen	-	boolean	1	If a reset buffer is used to provide reset synchronous to the system clock
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 externally)
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
ScalingSupport_Gen	-	boolean	1	If the value shall be scaled
OffsetSupport_Gen	-	boolean	1	If a DC Offset can be added, Requires Scaling to be active
SampleFreqHz_Gen	-	natural	1	Sample Frequency in Hz from 500kHz to 2MHz
Sample	-	natural	1	Sample Data Width

DataWidth_Gen				8 – 32bit
SampleSigned_Gen	-	boolean	1	If the sample repre-sents a signed value or unsigned value
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysClkNx_ClkIn	in	std_logic	1	High Resolution Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Clk_FrequencyGenerator StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_FrequencyGenerator 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
Scaling Input				
ScalingFactor_DatIn	in	std_logic_vector	9	Scaling as fractional value 1Q8 1.0 = 0x100, 0.5 = 0x080
Offset Input				
Offset Percentage_DatIn	in	std_logic_vector		DC offset in per-centage of the whole Sample range: -100 to + 100

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 Re-sponse
In Phase Output				
InPhase_DatOut	out	std_logic	1	If '1', the frequency generator is in phase to the time input
Frequency Output				
FrequencyGenera-tor_DatOut	out	std_logic_vector	Sample Data Width_	Since Wave sample

			Gen	for the DAC
FrequencyGenerator_ValOut	out	std_logic	1	Sine Wave sample valid

Table 8: Frequency Generator

4.2 Design Parts

The Frequency Generator core consists of a couple of subcores. Each of the sub-cores itself consists again of smaller function blocks. The following chapters describe these subcores and their functionality.

4.2.1 Frequency Generator

4.2.1.1 Entity Block Diagram

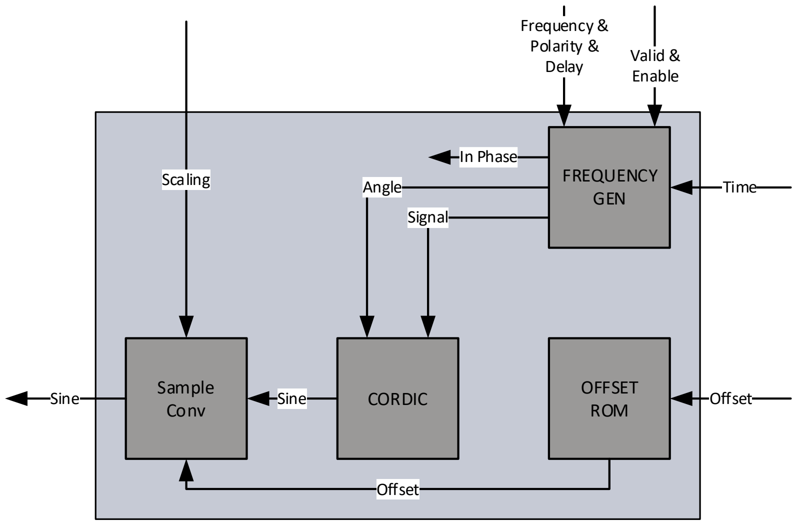

Figure 7: Frequency Generator

4.2.1.2 Entity Description

Frequency Generator

This process generates a signal with the configured frequency. When the Frequency Generator is enabled and the new input values are set, it registers the values and starts generating the signal of the configured frequency. At the beginning of the generation and until the beginning of the new second of the reference clock, the generated signal will have aligned frequency to the input time, but it will be out of phase. The phase will be aligned when the next new second of the input timer clock is reached. When a time jump happens the frequency generation will continue with the previous phase, until the first new second is reached. Then, the phase will also realign to the new time. Due to a phase realignment, the sine wave might be corrupted..

When a new cycle of the wave begins a pulse is asserted. This pulse will trigger the calculation of a new angle value based on the integrator value (angle). After this the sample frequency will define when the next angle value shall be calculated out of the integrator value.

CORDIC

This module calculates the sine value of an angle in the format 0 -2π by using the CORDIC algorithm with 16 iterations.

Offset ROM

This module stores the offsets in the sample format for each percentage so they can be directly added by the sample converter.

Sample Converter

This process converts the sine value into a DAC sample that can be feed to a DAC taking the data width and optional scaling and offsets correction into account.

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 externally)
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
ScalingSupport_Gen	-	boolean	1	If the value shall be scaled
OffsetSupport_Gen	-	boolean	1	If a DC Offset can be added, Requires Scaling to be active

SampleFreqHz_Gen	-	natural	1	Sample Frequency in Hz from 500kHz to 2MHz
Sample DataWidth_Gen	-	natural	1	Sample Data Width 8 – 32bit
SampleSigned_Gen	-	boolean	1	If the sample represents a signed value or unsigned value
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
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
Scaling Input				
ScalingFactor_DatIn	in	std_logic_vector	9	Scaling as fractional value 1Q8 1.0 = 0x100, 0.5 = 0x080
Offset Input				
Offset Percentage_DatIn	in	std_logic_vector		DC offset in percentage of the whole Sample range: -100 to + 100
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				
FrequencyValue_DatIn	in	std_logic_vector	24	Frequency to be generated in Hertz
FrequencyPolari-	in	std_logic	1	'1': High active, '0':

ty_DatIn				Low active
FreqeuncyCableDelay_DatIn	in	std_logic_vector	16	Delay in Nanosec-onds
FrequencyIgnorePhase_DatIn	in	std_logic	1	'1' = no Phase align-ment, '0' = Phase Alignment with second
Frequency_ValIn	in	std_logic	1	Configuration values are valid
Generation Monitor Output				
InPhase_DatOut	out	std_logic	1	If '1', the frequency generator is in phase to the time input
CyclesOverSec-ond_DatOut	out	std_logic_vector	24	The number of cycles generated during the previous second
Generation-Skip_DatOut	out	std_logic	1	If '1', the frequency generator skipped part of the last cycle due to phase align-ment (unused right now)
Frequency Output				
FrequencyGenera-tor_DatOut	out	std_logic_vector	SampleDataWidth_Gen	Since Wave sample for the DAC
FrequencyGenera-tor_ValOut	out	std_logic	1	Sine Wave sample valid

Table 9: Frequency Generator

4.2.2 Registerset

4.2.2.1 Entity Block Diagram

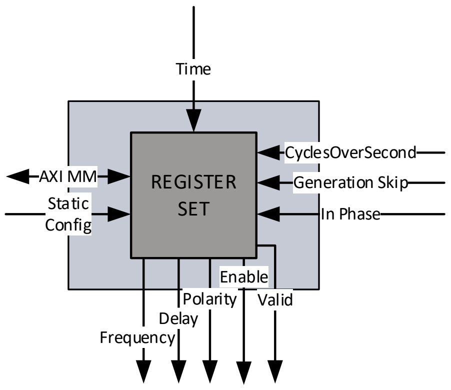

Figure 8: Registerset

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 Frequency 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 must be disabled and enabled again. If in AXI mode, an AXI Master must 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.

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
EmbeddedPps Support_Gen	-	boolean	1	Support to embed a PPS into the Fre-quency a duty cycle modulation
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_FrequencyGener ator StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_FrequencyGener ator StaticConfigVal _Type	1	Static Configuration 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 Re-sponse
Signal Values Output				
Frequen-cyValue_Datout	out	std_logic_vector	24	Frequency to be generated in Hertz
FrequencyPolari-ty_DatOut	out	std_logic	1	'1': High active, '0': Low active
FreqeuncyCableDelay _DatOut	out	std_logic_vector	16	Delay in Nanosec-onds
FrequencyIgnore Phase_DatOut	out	std_logic	1	'1' = no Phase align-ment, '0' = Phase Alignment with second
Frequency_ValOut	out	std_logic	1	Configuration values are valid
Generation Monitor Input				
InPhase_DatIn	in	std_logic	1	If '1', the frequency generator is in phase to the time input
CyclesOverSec-	in	std_logic_vector	24	The number of

ond_DatIn				cycles generated during the previous second
GenerationSkip_DatIn	in	std_logic	1	If '1', the frequency generator skipped part of the last cycle due to phase align-ment
Enable Output				
GenerateEnable _DatOut	out	std_logic	1	Enable Frequency 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 ClkStaticConfigFrequencyGenerator_Con : Clk_FrequencyGeneratorStaticConfig_Type := (
    Polarity                        => '1',
    CableDelay                      => std_logic_vector(to_unsigned(20, 16)),
    EmbeddedPps                     => '0', -- unused
    IgnorePhase                     => '0', -- phase aligned
    Frequency                       => std_logic_vector(to_unsigned(72000, 24)) - 72kHz
  );

constant ClkStaticConfigValFrequencyGenerator_Con : Clk_FrequencyGeneratorStaticConfigVal_Type
:= (
    Enable_Val                      => '1',
    Frequency_Val                   => '1'
  );
```

Figure 9: 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 ad-dress of the Clock is 0x10000000.

```
-- CLK FREQUENCY GENERATOR
-- Polarity = 1
AXI WRITE 10000008 00000001
-- Write value of Frequency 10000 Hz
AXI AXI0 WRITE 10000030 00002710
-- Write value of output cable delay 100 ns
AXI AXI0 WRITE 10000020 00000064
-- Write values and enable FrequencyGenerator and phase aligned
AXI AXI0 WRITE 10000000 00000003
```

Figure 10: AXI Configuration

The 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 Frequency Generator, like all other cores from NetTimeLogic, runs in one main clock domain. This is 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 metastability flip-flops are used. Clock domain crossings for the AXI interface are moved from the AXI slave to the AXI interconnect.

If a higher SPI clock rate is required (> ¼ System Clock) then an additional clock is used. Its frequency shall be a multiple of the system clock frequency. By default, the clock shall be 4-5 times faster than system clock and shall have a fixed relationship (generated from the same

Clock	Frequency	Description
System		
System Clock	50MHz (Default)	System clock where the frequency generation core runs on.
High Resolution		
High Resolution Clock	250MHz (Default)	SPI Module runs on a higher frequency
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 11: 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 12: Resets

5 Resource Usage

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

5.1 Intel/Altera (Cyclone 10)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static Config, No scaling and offset, disable cable delay, no SPI)	490	1520	0	5
Maximal (AXI, scaling and offset, cable delay, SPI)	670	1760	1	6

Table 13: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static Config, No HighResSupport, disable cable delay)	395	1230	0	5
Maximal (AXI, HighResSupport, enable cable delay)	570	1410	1	6

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 Frequency Generator testbench consists 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 Frequency Generator
For configuration and result checks an AXI read and write port is used.

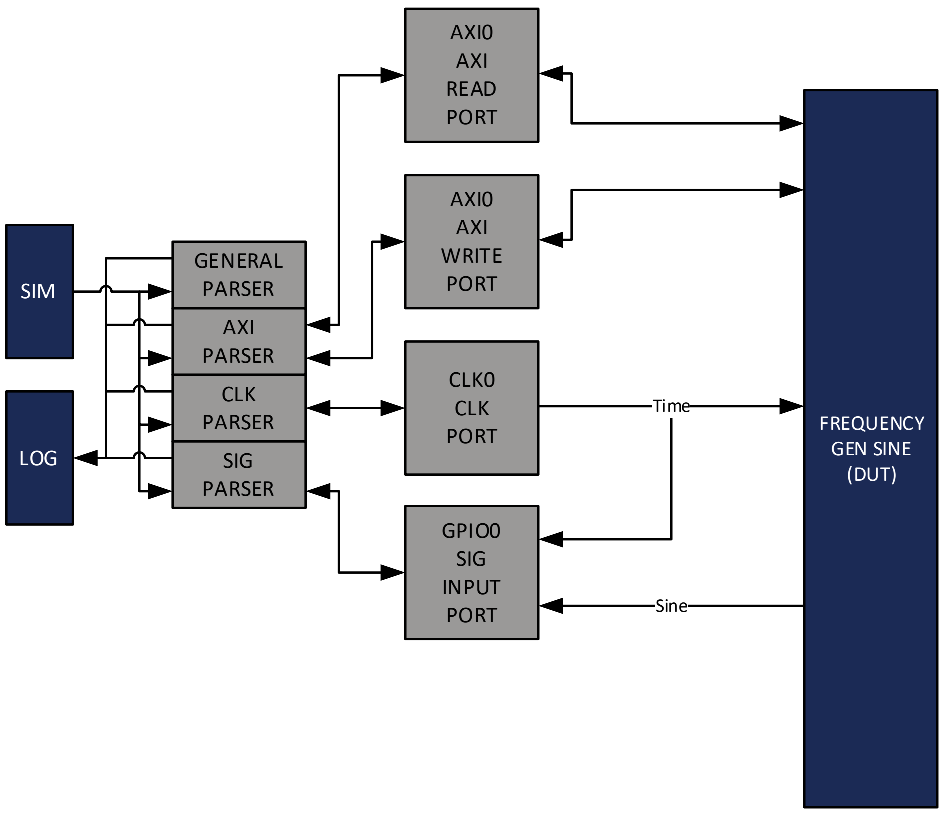

Figure 11: Testbench Framework

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

With the Sim parameter set the time base for timeouts are divided by 1000 to 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/ClkFrequencyGeneratorSine/Script/run_Clk_FrequencyGeneratorSine_Tb.tcl
```

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

8 Reference Designs

The Frequency Generator reference design contains a PLL to generate all necessary clocks (cores run at 50 MHz and 200MHz) and an instance of the Frequency Generator Sine IP core and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately). It also contains an SPI DAC interface to communicate with a PMOD DA2 DAC module. Optionally it also contains an instance of a PPS Master Clock IP core (must 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.

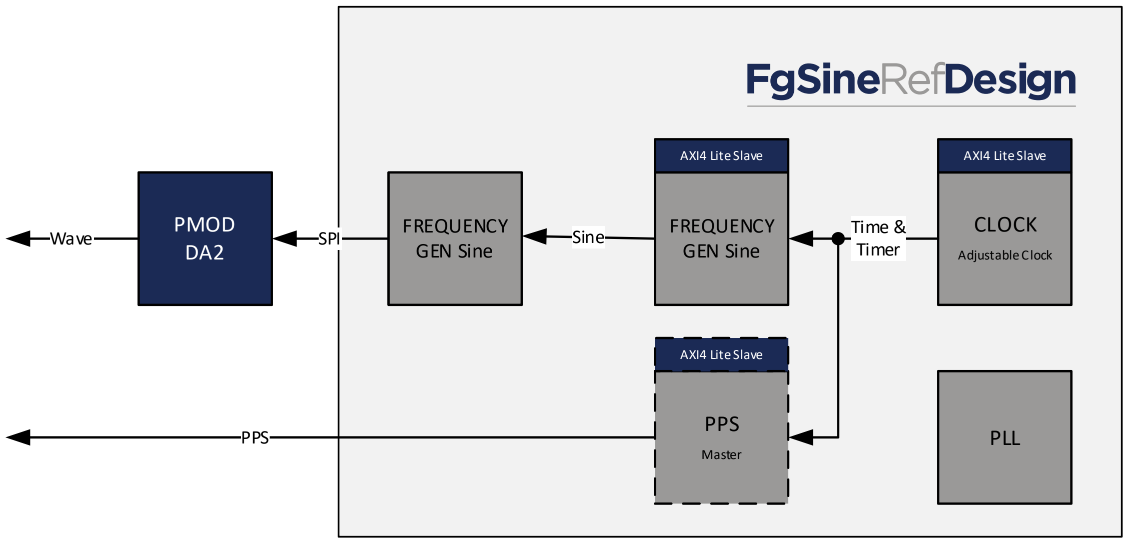

Figure 12: Reference Design

8.1 Intel/Altera: Cyclone 10 LP RefKit

The Cyclone 10 LP RefKit 10CL055 Development Board is an FPGA board from Arrow Electronics and Trenz Electronic GmbH with a Cyclone 10 FPGA from Intel/Altera. (https://shop.trenz-electronic.de/en/TEI0009-02-055-8CA-Cyclone-10-LP-RefKit-10CL055-Development-Board-32-MByte-SDRAM-16-MByte-Flash)

1. Open Quartus 18.x

2. Open Project /CLK/Refdesign/Altera/C10LpRefKit /ClkFrequencyGeneratorSine/ClkFrequencyGeneratorSine.qpf
3. If the optional core PPS Master Clock is available add the files from the corresponding folders (PPS/Core, PPS/Library and PPS/Package)
4. Change the generics (PpsMasterAvailable_Gen) in Quartus (in the settings menu, not in VHDL) to true for the optional cores that are available.
5. Rerun implementation
6. Download to FPGA via JTAG

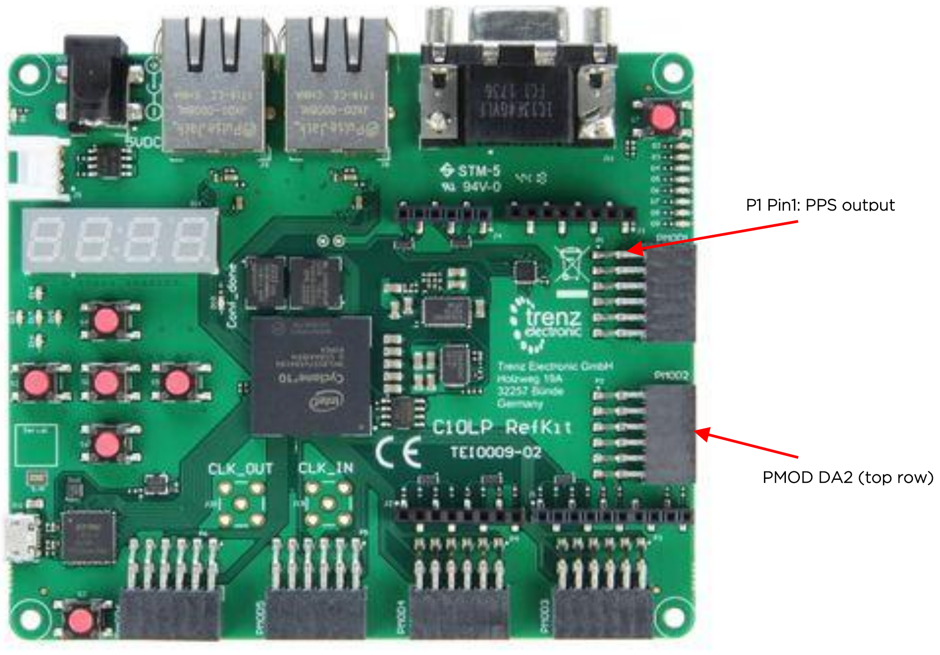

Figure 13: Cyclone 10 LP RefKit (source Trenz Electronic GmbH)

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/ ClkFrequencyGeneratorSine/ ClkFrequencyGeneratorSine.tcl
   a. This must be run only the first time and will create a new Vivado Project

4. If the project has been created before opening 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

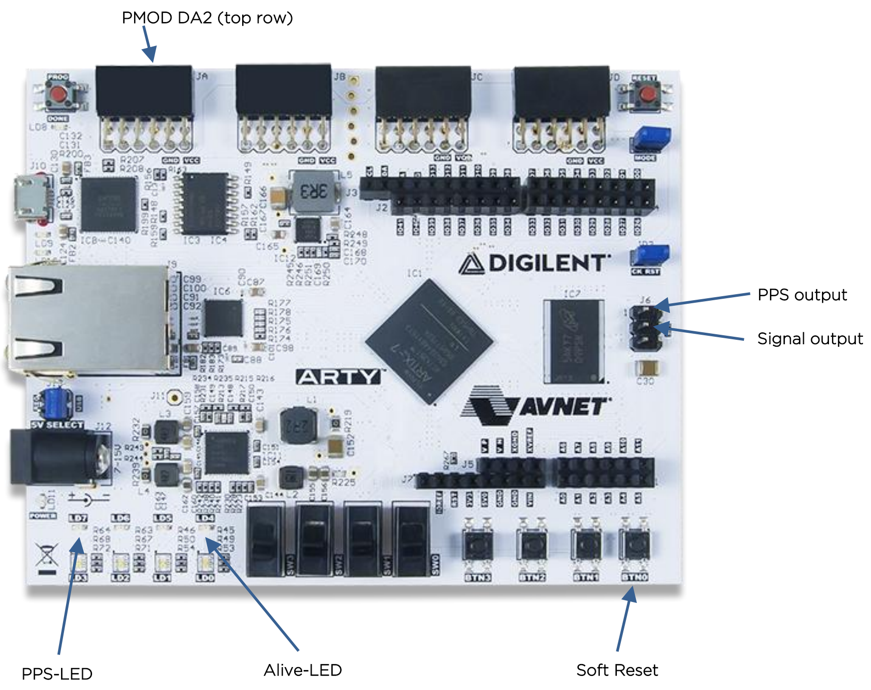

Figure 14: Arty (source Digilent Inc)

8.3 AMD/Xilinx: Vivado Version

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

- Before executing the project creation TCL script, the script's references of Vivado 2019 should be manually replaced with 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