ClockCounterClock

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.9
Date	03.01.2023

Copyright Notice

Disclaimer

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

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

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

Overview

NetTimeLogic's Clock (CLK) Counter Clock is a full hardware (FPGA) only implementation of an adjustable counter clock with PI servo loop and spread adjustment. The whole protocol algorithms and calculations are implemented in the core, no CPU is required. This allows running synchronization completely independent and standalone from the user application. The Adjustable Counter Clock has multiple selectable input adjustments: PTP, PPS, TOD, IRIG, RTC and REG. The time and settings can be configured either by signals or by an AXI4Lite-Slave Register interface.

Key Features:

- 32 bit second and 32 bit nanosecond counter clock with fractional extension
- Provides time for all other cores
- 1 millisecond pulse generator aligned with the counter clock
- Allows non-integer clock periods (fractions)
- Multiplexing of multiple adjustment inputs
- 6 different adjustment sources: PTP, PPS, TOD, IRIG, RTC and REG
- Evenly spread offset and drift correction over time (offset might be set hard in case of large offsets)
- Hard setting of time possible
- Individual hardware only PI servo loops for offset and drift correction (PI parameters individually configurable)
- Runtime changeable PI parameters
- Offset correction: min 1 ns/s, max 0.5 s/s
- Drift correction: min 1 ns/s, max 0.05 s/s
- Conversion of fractional adjustments into even spread clock adjustments
- AXI4Lite register set or static configuration
- Get time and set time as well as adjusting of offset and drift possible via registers as well.
- External clock source selection
- Correction logging for Offset and Drift
- Linux Driver (PHC)

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	28.12.2015	First draft
1.0	13.05.2016	First release
1.1	07.06.2016	Added structured types section
1.2	24.02.2017	Changed SNTP to RTC, added status interface and ExtSelect and CoreInfo type
1.3	20.12.2017	Added Fractional multiplication
1.4	09.01.2018	Added Holdover
1.5	09.03.2018	Added Driver
1.6	12.06.2018	Added Dynmic PI Servo Paramters and Loging registers
1.7	27.08.2020	Updated Clock Select Register
1.8	22.04.2022	Added DCF and NTP
1.9	03.01.2023	Added Vivado upgrade version description

Table 1: Revision History

Content

Definitions

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

Table 2: Definitions

Abbreviations

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

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

The Adjustable Counter Clock is meant as a co-processor handling clock adjustments and a time generator.
It is designed to work with all synchronization cores from NetTimeLogic. All correction inputs are multiplexed and selectable. For each correction input it allows drift, offset and time corrections. The offset and drift values are filtered via a PI servo loop and evenly spread over time to allow smooth corrections.
It is the source of time for all other cores from NetTimelogic, not only the synchronization cores which will in the end adjust the Counter Clock but also for the Signal Timestampers and Signal Generators which work aligned with this Counter Clock.
It contains an AXI4Lite slave for configuration, time setting, getting and adjusting from a CPU, this is however not required since the Adjustable Counter Clock can also be configured statically via signals/constants directly from within the FPGA.

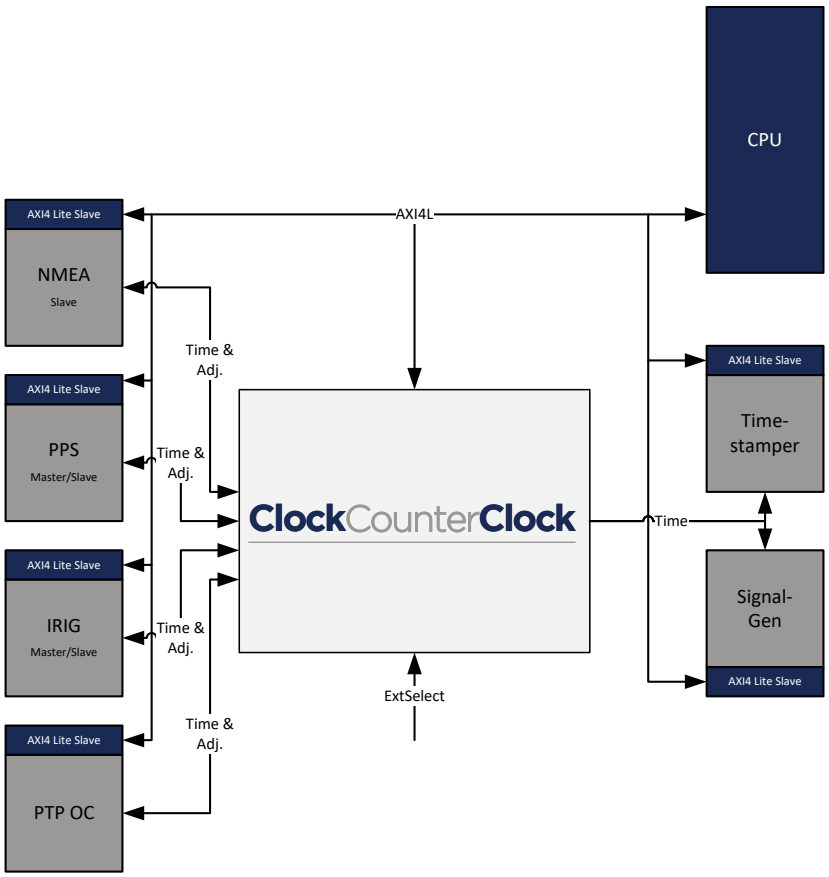


Figure 1: Context Block Diagram

1.2 Function

The adjustable counter clock is, as the name says already, a counter clock in the second and nanosecond format that can be frequency and phase adjusted.
It adds the clock period it runs on in nanoseconds every clock cycle. If the clock period is a non-integer value e.g. for 66MHz => 15.1515... ns the fractional part of the nanoseconds can be entered as a fraction e.g. for 66MHz => 10/66 => numerator = 10 and denumerator = 66. Every clock cycle a fraction counter is adding the numerator with an overflow at the denumerator value. At each overflow of the fraction counter it will add one additional nanosecond to the nanoseconds counter. This way no error is introduced because of non-integer periods. The nanosecond counter has an overflow at 1000000000 nanoseconds. At each overflow of the nanosecond counter it will add an additional second to the seconds counter.
For adjustments additional nanoseconds can be added or subtracted from the standard period to adjust frequency and phase and the time can be overwritten to hard set the clock to a new time when small offset adjustments are not suitable e.g. for the startup phase where a jump from the 1.1.1970 to the present is required.
To minimize the counter widths in other cores e.g. for timeouts or other time periods which don't need nanosecond resolution a timer is used which generates a counter clock aligned timer event every millisecond e.g. for a period count of 1 second only 10 bits instead of 30 bits are needed.
An adjustment block takes the offset, drift and time adjustments as inputs and converts them into a combined adjustment which is then applied to the counter clock. The offset and drift is converted into evenly spread adjustments which allows smooth corrections on the clock without time jumps. E.g. a drift of 1ppm is adjusted as 1 ns every 5000000 clock cycles at 50MHz. Again the adjustment is made with fractional counter for minimal computational error introduction (max computational error 1ns/s). In parallel to the correction a quality computation done to mark the in-sync state of the clock if corrections are below a certain threshold for at least 5 corrections.
Before the adjustments are converted for the counter clock, the drift and offset adjustments are passing a PI servo loop for filtering and smoother adjustments. The PI servo loop parameter for the drift and offset adjustments can be chosen individually, since frequency changes might happen quite slowly where offset adjustments probably shall be done much faster. The PI servo loops results are feed back to the other cores for correcting of the next adjustments.
Since the Adjustable Counter Clock is the heart of a synchronization solution it can take several adjustment inputs from different cores as input. Only one adjustment

input is taken as source for corrections at the time. From the Registerset the multiplexer gets the selection of the current input or if in external mode a selection from outside can be done.
The Registerset allows reading the in-sync state, taking a snapshot of the time, overwriting the current time for e.g. the startup phase and also allows adjusting the drift and offset as an additional input to the multiplexer.

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.

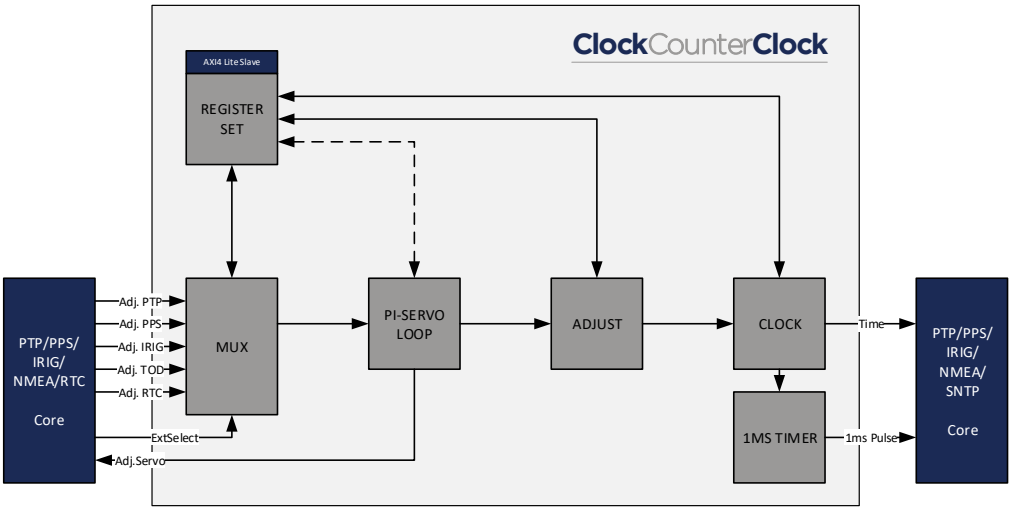


Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

Mux

This block is a multiplexer which forwards only one of the many possible adjustment inputs.

PI Servo Loop

This block contains two PI Servo Loops, one for the Offset and one for the Drift correction. The calculated values after the PI Servo are provided to the cores for the next calculations.

Adjustment

This block converts the adjustments in ratios or plain numbers to actual adjustments on the clock, smoothly spread over time. It also combines Drift and Offset adjustments and checks whether the clock is in sync.

Clock

This block is an adjustable counter clock with fractions. It provides the time used by all other cores.

1ms Timer

This block is one millisecond timer aligned with the clock. It generates a pulse every milliseconds. This is used by other cores to keep their counters small if only relative time is needed and millisecond resolution is sufficient.

2 Clock 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 quantitation of the clock period (this is what this core implements).

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.

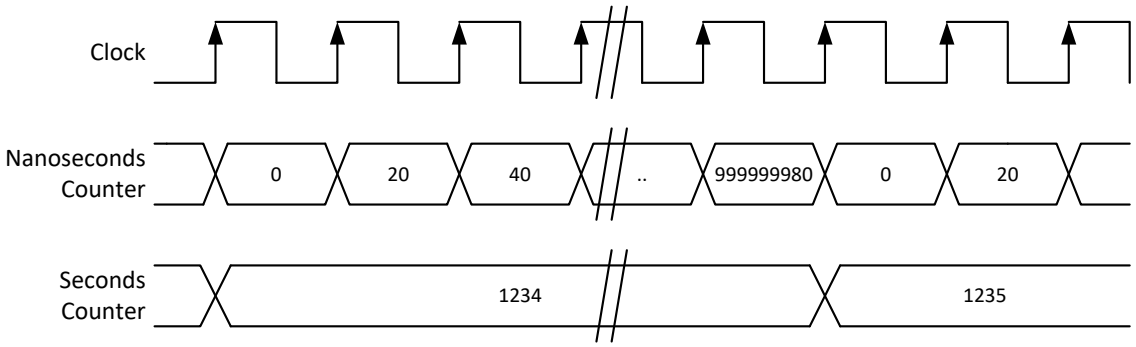


Figure 3: Counter Clock

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.

There are basically two possibilities how to adjust a digital counter clock. One is to keep the counter increment and adjust the clock frequency with e.g. load on a crystal oscillator where the load can be adjusted via a DAC. The second and the one used for this core, 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. Where the first solution implicitly generates a smooth adjustment, for the second this has to be calculated and explicitly done. 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, and this also applies for the solution where the frequency of the clock is adjusted.
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 this 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.

3 Register Set

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

3.1 Register Overview

Registerset Overview			
Name	Description	Offset	Access
Clk ClockControl Reg	Clock Read/Write Valid and Enabled Control Register	0x00000000	RW
Clk ClockStatus Reg	Clock Status Register	0x00000004	RO
Clk ClockSelect Reg	Clock Adjustment Multiplexer selection Register	0x00000008	RW
Clk ClockVersion Reg	Clock Version Register	0x0000000C	RO
Clk ClockTimeValueL Reg	Clock current Time Nanosecond Register	0x00000010	RO
Clk ClockTimeValueH Reg	Clock current Time Second Register	0x00000014	RO
Clk ClockTimeAdjValueL Reg	Clock adjust Time Nanosecond Register	0x00000020	RW
Clk ClockTimeAdjValueH Reg	Clock adjust Time Second Register	0x00000024	RW
Clk ClockOffsetAdjValue Reg	Clock adjust Offset Nanosecond Register	0x00000030	RW
Clk ClockOffsetAdjInterval Reg	Clock adjust Offset Interval Nanosecond Register	0x00000034	RW
Clk ClockDirftAdjValue Reg	Clock adjust Drift Nanosecond Register	0x00000040	RW
Clk ClockDriftAdjInterval Reg	Clock adjust Drift Interval Nanosecond Register	0x00000044	RW
Clk ClockInSyncThreshold Reg	Clock adjust Drift Interval Nanosecond Register	0x00000050	RW
Clk_ClockServoOffsetFactorP_Reg	Clock Offset Servo P Factor Register	0x00000060	RW
Clk_ClockServoOffsetFactorI_Reg	Clock Offset Servo I Factor Register	0x00000064	RW

Clk_ClockServoDriftFactorP_Reg	Clock Drift Servo P Factor Register	0x00000068	RW
Clk_ClockServoDriftFactorI_Reg	Clock Drift Servo I Factor Register	0x0000006C	RW
Clk_ClockStatusOffset_Reg	Clock corrected Offset Register	0x00000070	RO
Clk_ClockStatusDrift_Reg	Clock corrected Drift Register	0x00000074	RO

3.2 Register Descriptions

3.2.1 General

3.2.1.1 CLK Clock Control Register

Used for general control over the Adjustable Counter Clock. To get a new time snapshot the time read flag has to be set and the read done flag is asserted. Since most adjustment values are multi register values, set flags are available to mark validity of the whole value.

Clk ClockControl 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_READ_DONE	TIME_READ																						SERVO_VAL					DRIFT_VAL	OFFSET_VAL	TIME_VAL	ENABLE
RO	RW	RO																										RW	RW	RW	RW
Reset: 0x00000000																															
Offset: 0x0000																															

Name	Description	Bits	Access
TIME_READ_DONE	Time Read done (autocleared)	Bit: 31	RO

TIME_READ	Time Read (autocleared)	Bit: 30	RW
-	Reserved, read 0	Bit: 29:9	RO
SERVO_VAL	Servo Parameters Valid (autocleared), shall only be set when the clock is disabled	Bit: 8	RW
-	Reserved, read 0	Bit: 7:4	RO
DRIFT_VAL	Drift Ajustment Valid (autocleared)	Bit: 3	RW
OFFSET_VAL	Offset Ajustment Valid (autocleared)	Bit: 2	RW
TIME_VAL	Time Ajustment Valid (autocleared)	Bit: 1	RW
ENABLE	Enable	Bit: 0	RW

3.2.1.2 CLK Clock Status Register

Marks if the clock is in sync. In sync means 5 consecutive offset where below the threshold.

Clk ClockStatus 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_HOLDOVER	IN_SYNC
RO																														RO	RO
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:2	RO
IN_HOLDOVER	In holdover, after in sync for N seconds no new adjustment	Bit: 1	RO
IN_SYNC	In sync; Last 5 corrections below threshold	Bit: 0	RO

3.2.1.3 CLK Clock Select Register

Which input adjustment shall be active. If external select is active the CLK_SELECTED field shows the actual selection

Clk ClockSelect Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-								CLK_SELECTED								-								CLK_SELECT							
RO								RO								RO								RW							
Reset: 0x00000000																															
Offset: 0x0008																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:24	RO

CLK_SELECTED	Selected Source for Clock Adjustments: None = 0 Tod = 1 Irig = 2 Pps = 3 Ptp = 4 Rtc = 5 Dcf = 6 Ntp = 7 Regs = 254 Ext = 255	Bit: 23:16	RO
-	Reserved, read 0	Bit: 15:8	RO
CLK_SELECT	Source for Clock Adjustments: None = 0 Tod = 1 Irig = 2 Pps = 3 Ptp = 4 Rtc = 5 Dcf = 6 Ntp = 7 Regs = 254 Ext = 255	Bit: 7:0	RW

3.2.1.4 CLK Clock 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 ClockVersion 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 Clock Time Value Low Register

Time snapshot value nanosecond part.

Clk ClockTimeValueL 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_NS																															
RO																															
Reset: 0x00000000																															
Offset: 0x0010																															

Name	Description	Bits	Access
TIME_NS	Snapshoted Time Nanosecond	Bit: 31:0	RO

3.2.1.6 CLK Clock Time Value High Register

Time snapshot value second part.

Clk ClockTimeValueH 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_S																															
RO																															
Reset: 0x00000000																															
Offset: 0x0014																															

Name	Description	Bits	Access
TIME_S	Snapshoted Time Second	Bit: 31:0	RO

3.2.1.7 CLK Clock Time Adjustment Value Low Register

Time adjustment nanoseconds part value. Will overwrite the clock.

Clk ClockTimeAdjValueL 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_ADJ_NS																													
RW																													
Reset: 0x00000000																													
Offset: 0x0020																													

Name	Description	Bits	Access
TIME_ADJ_NS	OverwriteTime Nanosecond	Bit: 31:0	RW

3.2.1.8 CLK Clock Time Adjustment Value High Register

Time adjustment seconds part value. Will overwrite the clock.

Clk ClockTimeAdjValueH 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_ADJ_S																													
RW																													
Reset: 0x00000000																													
Offset: 0x0024																													

Name	Description	Bits	Access
TIME_ADJ_S	OverwriteTime Second	Bit: 31:0	RW

3.2.1.9 CLK Clock Offset Adjustment Value Register

Offset adjustment absolute value. A negative sign will slow down the clock, a positive sign therefore accelerates the clock to adjust the phase.

Clk ClockOffsetAdjValue 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
OFFSET_SIGN	OFFSET_ADJ_NS																														
RW																															
Reset: 0x00000000																															
Offset: 0x0030																															

Name	Description	Bits	Access
OFFSET_SIGN	Sign of the Adjustment 0 positive, 1 negative	Bit: 31	RW
OFFSET_ADJ_NS	Offset Adjustment in Nanosecond	Bit: 30:0	RW

3.2.1.10 CLK Clock Offset Adjustment Interval Register

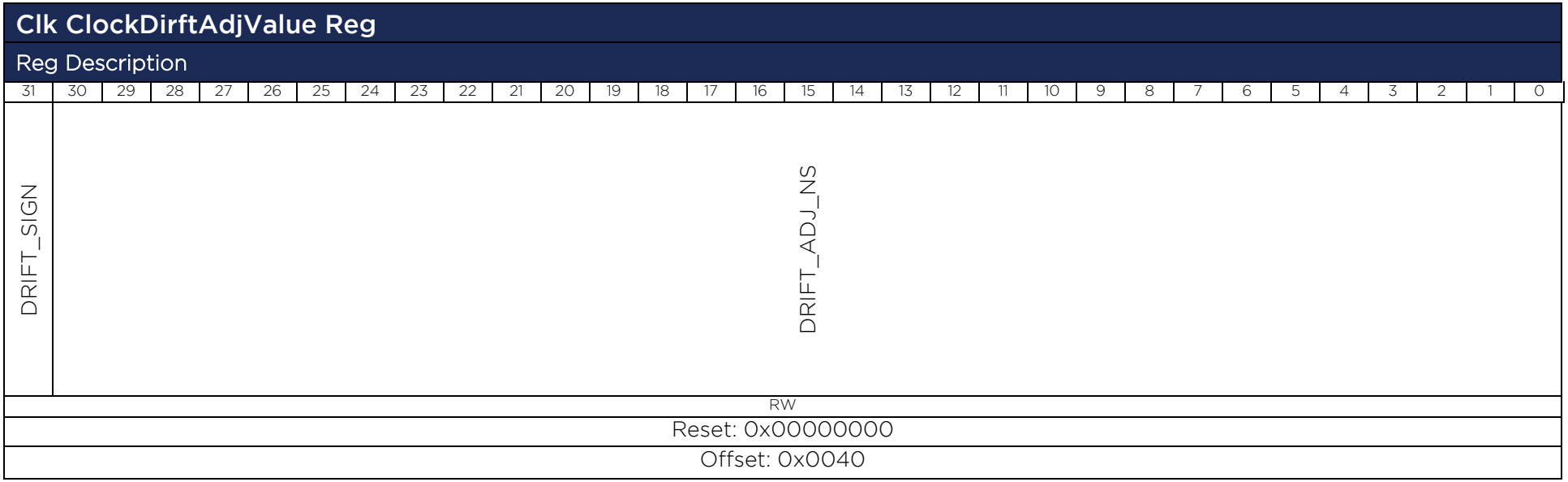
Offset adjustment interval. It defines the interval in which the offset shall be corrected. If the offset value is larger or equal the interval, the time will be set directly.

Clk ClockOffsetAdjInterval 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
OFFSET_ADJ_INTV_NS																													
RW																													
Reset: 0x00000000																													
Offset: 0x0034																													

Name	Description	Bits	Access
OFFSET_ADJ_INTV_NS	Offset Adjustment Interval in Nanosecond	Bit: 31:0	RW

3.2.1.11 CLK Clock Drift Adjustment Value Register

Drift adjustment value. Together with the drift interval this indicates a ratio of correction in the format ns per ns which means how many nanoseconds shall be corrected in the interval. A negative sign will slow down the clock, a positive sign therefore accelerates the clock.



Name	Description	Bits	Access
DRIFT_SIGN	Sign of the Adjustment 0 positice, 1 negative	Bit: 31	RW
DRIFT_ADJ_NS	Drift Adjustment in Nanosecond	Bit: 30:0	RW

3.2.1.12 CLK Clock Drift Adjustment Interval Register

Drift adjustment interval. Together with the drift value this indicates a ratio of correction in the format ns per ns which means how many nanoseconds shall be corrected in the interval specified by this register.

Clk ClockDriftAdjInterval 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
DRIFT_ADJ_INTV_NS																															
RW																															
Reset: 0x00000000																															
Offset: 0x0044																															

Name	Description	Bits	Access
DRIFT_ADJ_INTV_NS	Drift Adjustment Interval in Nanosecond	Bit: 31:0	RW

3.2.1.13CLK Clock In Sync Threshold Register

Threshold of the in sync indication flag. After consecutive offset adjustments below the threshold the in sync flag is asserted. The threshold is in nanosecond and the default value is set via generic.

Clk ClockInSyncThreshold 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
THRESHOLD_NS																															
RW																															
Reset: 0xXXXXXXXX																															
Offset: 0x0050																															

Name	Description	Bits	Access
THRESHOLD_NS	Threshold for InSync flag in Nanoseconds	Bit: 31:0	RW

3.2.1.14 CLK Clock Servo Offset Factor P Register

Proportional factor for the fractional offset multiplier. The Value is calculated the following way:

$(MulP * 2^{16})/DivP$

After reset the value is defined by the generics.

This register is only available if the generic BypassServo_Gen is false and the generics FractionalMultiply_Gen and DynamicServo-Parameters_Gen are true.

Clk ClockServoOffsetFactorP 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
OFFSET_FACTOR_P																															
RW																															
Reset: 0xXXXXXXXX																															
Offset: 0x0060																															

Name	Description	Bits	Access
OFFSET_FACTOR_P	Fractional multiplication Factor Offset P: ((Mul*2^16)/Div)	Bit: 31:0	RW

3.2.1.15 CLK Clock Servo Offset Factor I Register

Integral factor for the fractional offset multiplier. The Value is calculated the following way:

$(MulI * 2^{16})/DivI$

After reset the value is defined by the generics.

This register is only available if the generic BypassServo_Gen is false and the generics FractionalMultiply_Gen and DynamicServo-Parameters_Gen are true.

Clk ClockServoOffsetFactorIReg

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
OFFSET_FACTOR_I																															
RW																															
Reset: 0xXXXXXXXX																															
Offset: 0x0064																															

Name	Description	Bits	Access
OFFSET_FACTOR_I	Fractional multiplication Factor Offset I: ((Mul*2^16)/Div)	Bit: 31:0	RW

3.2.1.16CLK Clock Servo Drift Factor P Register

Proportional factor for the fractional drift multiplier. The Value is calculated the following way:

$(MulP * 2^{16})/DivP$

After reset the value is defined by the generics.

This register is only available if the generic BypassServo_Gen is false and the generics FractionalMultiply_Gen and DynamicServo-Parameters_Gen are true.

Clk ClockServoDriftFactorP 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
DRIFT_FACTOR_P																															
RW																															
Reset: 0xXXXXXXXX																															
Offset: 0x0060																															

Name	Description	Bits	Access
DRIFT_FACTOR_P	Fractional multiplication Factor Drift P: ((Mul*2^16)/Div)	Bit: 31:0	RW

3.2.1.17 CLK Clock Servo Drift Factor I Register

Integral factor for the fractional drift multiplier. The Value is calculated the following way:

$(MulI * 2^{16})/DivI$

After reset the value is defined by the generics.

This register is only available if the generic BypassServo_Gen is false and the generics FractionalMultiply_Gen and DynamicServo-Parameters_Gen are true.

Clk ClockServoOffsetFactorIReg

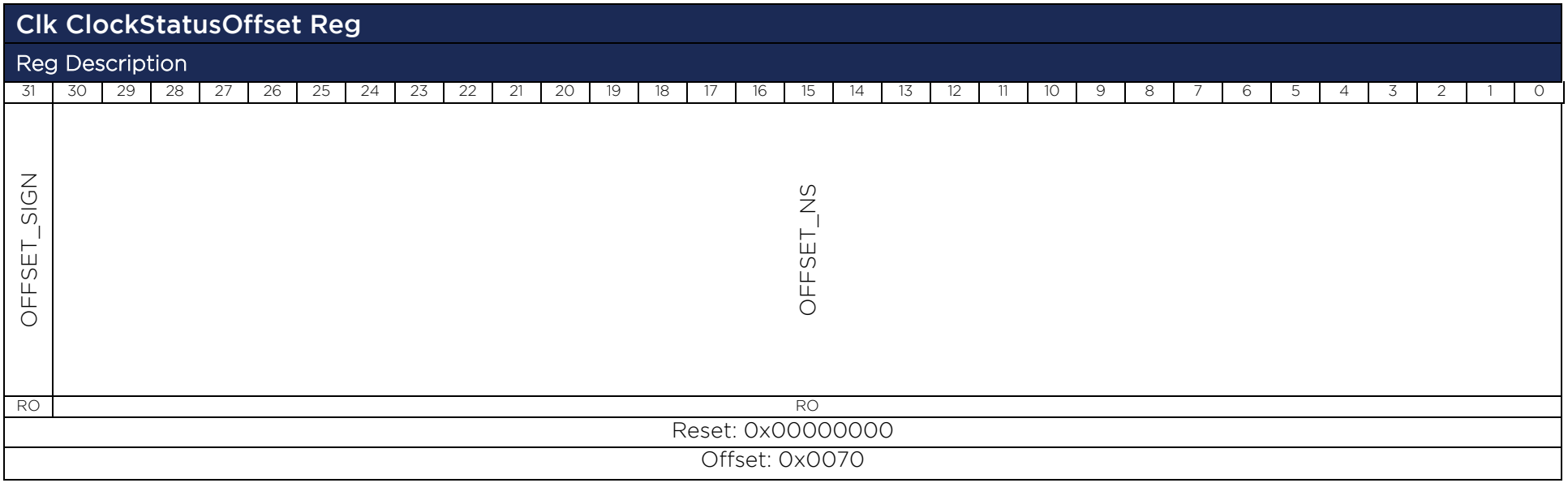
Reg Description

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
DRIFT_FACTOR_I																															
RW																															
Reset: 0xXXXXXXXX																															
Offset: 0x0064																															

Name	Description	Bits	Access
DRIFT_FACTOR_I	Fractional multiplication Factor Drift I: ((Mul*2^16)/Div)	Bit: 31:0	RW

3.2.1.18CLK Clock Status Offset Value Register

Last corrected offset in nanoseconds. This register is only available if the generic LogCorrections_Gen is true.



Name	Description	Bits	Access
OFFSET_SIGN	Sign bit: 0 = positive, 1 = negative	Bit: 31	RO
OFFSET_NS	Last corrected Offset value in Nanoseconds	Bit: 30:0	RO

3.2.1.19CLK Clock Status Drift Value Register

Last corrected drift in nanosecond. This register is only available if the generic LogCorrections_Gen is true.

Clk ClockStatusDrift 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
DEIFT_SIGN	DRIFT_NS																														
RO	RO																														

Reset: 0x00000000

Offset: 0x0074

Name	Description	Bits	Access
DRIFT_SIGN	Sign bit: 0 = positive, 1 = negative	Bit: 31	RO
DRIFT_NS	Last corrected Drift value in Nanoseconds/Second (ppb)	Bit: 30:0	RO

4 Design Description

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

4.1 Top Level – Clk Clock

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
ExtSelect_Gen	boolean	1	If external selection of correction shall be done or not: true = external selection is possible, false only selection via reg
ClockClkPeriodNanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns, if non-integer only the nanoseconds part here
ClockClkPeriodFractNum_Gen	natural	1	Fractional Clock Period Numerator (0 if integer)
ClockClkPeriodFractDeNum_Gen	natural	1	Fractional Clock Period Denumerator (0 if integer)
ClockInSyncNanosecond_Gen	natural	1	Default value for the threshold when the clock is considered in sync in Nanoseconds Default 500 ns
ClockInHoldover TimeoutSecond_Gen	natural	1	Value after how many seconds after in Sync without

			new correction values it goes in holdover Default 3 s
LogCorrections_Gen	boolean	1	Log Correction values to registers
BypassServo_Gen	boolean	1	Bypass PI Servo loops, the input will directly be taken for calculations
FractionalMultiply_Gen_Gen	boolean	1	Use a fractional multiplication (uses DSP slices) instead of binary multiplication and divisions (default)
DynamicServo Parameters_Gen	boolean	1	Allow to change the PI Servo parameters at runtime
DriftMulP_Gen	natural	1	Drift Proportional part ratio Numerator
DriftDivP_Gen	natural	1	Drift Proportional part ratio Denumerator
DriftMulI_Gen	natural	1	Drift Integral part ratio Numerator
DriftDivI_Gen	natural	1	Drift Integral part ratio Denumerator
OffsetMulP_Gen	natural	1	Offset Proportional part ratio Numerator
OffsetDivP_Gen	natural	1	Offset Proportional part ratio Denumerator
OffsetMulI_Gen	natural	1	Offset Integral part ratio Numerator
OffsetDivI_Gen	natural	1	Offset Integral part ratio Denumerator
AxiAddressRange Low_Gen	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	std_logic_vector	32	AXI Base Address plus Registerset Size Default plus 0xFFFF
Sim_Gen	boolean	1	If in Testbench simulation mode:

			true = Simulation, false = Synthesis

Table 4: Parameters

If an integer clock period is used all fraction parameters have to be 0.

4.1.1.2 Structured Types

4.1.1.2.1 Clk_Time_Type

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

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

Table 5: Clk_Time_Type

4.1.1.2.2 Clk_TimeAdjustment_Type

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

Field Name	Type	Size	Description
TimeAdjustment	Clk_Time_Type	1	Time to adjust
Interval	std_logic_vector	32	Adjustment interval, for the drift correction this is the denumerator of the rate in nanoseconds (TimeAdjustment

			every Interval = drift rate), for offset correction this is the period in which the time shall be corrected(TimeAdjustment in Interval), for setting the time this has no mining.

Table 6: Clk_TimeAdjustment_Type

4.1.1.2.3 Clk_CoreInfo_Type

Defined in Clk_Package.vhd of library ClkLib
This is the type used for getting info about the cores state status.

Field Name	Type	Size	Description
State	Clk_CoreState_Type	1	State of the core: Unknown_E, Slave_E or Master_E
Accuracy	std_logic_vector	8	Accuracy of the core, higher is better
Enabled	std_logic	1	If the core is enabled
InSync	std_logic	1	If the core is synchronized
Error	std_logic	1	If the core has an error

Table 7: Clk_CoreInfo_Type

4.1.1.2.4 Clk_ClockStaticConfig_Type

Defined in Clk_ClockAddrPackage.vhd of library ClkLib
This is the type used for static configuration.

Field Name	Type	Size	Description
TimeAdjustment	Clk_TimeAdjustment_Type	1	Set the time to this value
OffsetAdjustment	Clk_TimeAdjustment_Type	1	Do an offset adjustment with this value
DriftAdjustment	Clk_TimeAdjustment_Type	1	Do an drift adjustment with this value
ClockSelect	std_logic_vector	32	Which adjustment input shall be used:

			0 => None 1 => Tod 2 => Irig 3 => Pps 4 => Ptp 5 => Rtc 6 => Regs 7 => Ext
InSyncThreshold	std_logic_vector	32	Threshold of the in sync flag indication in nanoseconds

Table 8: Clk_ClockStaticConfig_Type

4.1.1.2.5 Clk_ClockStaticConfigVal_Type

Defined in Clk_ClockAddrPackage.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
TimeAdjustment_Val	std_logic	1	Set the time
OffsetAdjustment_Val	std_logic	1	Start an offset adjustment
DriftAdjustment_Val	std_logic	1	Start a drift adjustment

Table 9: Clk_ClockStaticConfigVal_Type

4.1.1.2.6 Clk_ClockStaticStatus_Type

Defined in Clk_ClockAddrPackage.vhd of library ClkLib
This is the type used for static status.

Field Name	Type	Size	Description
CoreInfo	Clk_CoreInfo_ Type	1	Infor about the Cores state
ClockTime	Clk_Time_Type	1	Time of the Clock

Table 10: Clk_ClockStaticStatus_Type

4.1.1.2.7 Clk_ClockStaticStatusVal_Type

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

Field Name	Type	Size	Description
ClockTime_Val	std_logic	1	Set the time

Table 11: Clk_ClockStaticStatusVal_Type

4.1.1.3 Entity Block Diagram

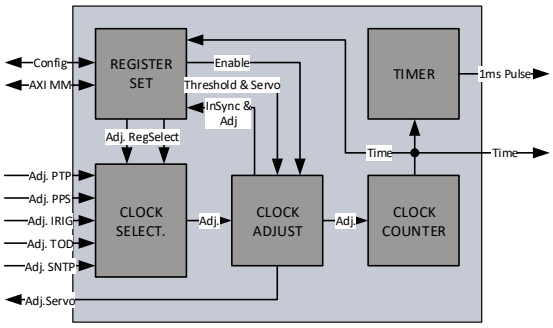


Figure 4: Adjustable Counter Clock

4.1.1.4 Entity Description

Clock Selector

This module multiplexes multiple adjustment inputs. Only one adjustment is active at the time. This can be selected by a configuration register. PTP, PPS, TOD, IRIG, RTC, REG and EXT (where the selection is feed from externally) are potential sources. The multiplexer can also be set to disable any adjustment input.
See 4.2.1 for more details.

Clock Adjuster

This module contains the PI Servo loops for the Drift and Offset and the adjustment conversion. It takes the adjustment inputs and converts them into periodic small adjustments which are then set to the Counter Clock. E.g. a drift of 1ns per 1000ns is converted to a 1ns adjustment every 50 clock cycles at a frequency of 50MHz. For offset it works similar. E.g. 50ns shall be corrected in 2000ns which will be converted into 1ns every second clock cycle at 50MHz. The conversion of course also works for non-integer corrections with minimal error (1/- 1ns per correction cycle).

In addition it supervises the adjustments ad sets an InSync flag if 4 consecutive offset corrections were below a configurable threshold.
See 4.2.2 for more details.

Clock Counter

This module is the heart of the core. It is a counter with nanosecond resolution in a 32 bit second and 32 bit nanosecond format. It can run with fractions to allow also non-integer clock periods. It normally adds the clock period to the nanoseconds counter every clock cycle but can add or subtract some extra nanosecond to do the clock adjustment. It also has an overwrite mode where the clock can be set.
See 4.2.3 for more details.

Timer

This module creates a 1 millisecond timer event aligned with the Counter Clock. If the clock makes a jump in time this may lead to a loss of an event or to two events very close to each other.
See 4.2.4 for more details.

Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows to configure the Adjustable Counter Clock. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the resgisters is done via signals and can be easily done from within the FPGA without CPU. If in AXI mode, an AXI Master has to configure the Datasets with AXI writes to the registers, which is typically done by a CPU. It also provides a status interface which allows similar to the static configuration to supervise the status via signals.
See 4.2.5 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

ExtSelect_Gen	-	boolean	1	If external selection of sync source is used
ClockClkPeriodNano-second_Gen	-	natural	1	Clock Period in Nanosecond
ClockClkPeriodFract-Num_Gen	-	natural	1	Fractional Clock Period Numerator (0 if integer)
ClockClkPeriod-FractDeNum_Gen	-	natural	1	Fractional Clock Period Denumerator (0 if integer)
ClockInSyncNanosec-ond_Gen	-	natural	1	Default value for the threshold when the clock is considered in sync
ClockInHoldover TimeoutSecond_Gen	-	natural	1	Value after how many seconds after in Sync without new correction values it goes in holdover Default 3 s
LogCorrections_Gen	-	boolean	1	Log corrections to registers
BypassServo_Gen	-	boolean	1	Bypass PI Servo loops
FractionalMultiply _Gen	-	boolean	1	Use a fractional multiplication (uses DSP slices) instead of binary multiplication and divisions (default)
DynamicServo Parameters_Gen	-	boolean	1	Allow to change the PI Servo parameters at runtime
DriftMulP_Gen	-	natural	1	Drift Proportional part ratio Numerator

DriftDivP_Gen	-	natural	1	Drift Proportional part ratio Denumerator
DriftMulI_Gen	-	natural	1	Drift Integral part ratio Numerator
DriftDivI_Gen	-	natural	1	Drift Integral part ratio Denumerator
OffsetMulP_Gen	-	natural	1	Offset Proportional part ratio Numerator
OffsetDivP_Gen	-	natural	1	Offset Proportional part ratio Denumerator
OffsetMulI_Gen	-	natural	1	Offset Integral part ratio Numerator
OffsetDivI_Gen	-	natural	1	Offset Integral part ratio Denumerator
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_Clock StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_Clock StaticConfigVal_Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Clk_Clock StaticStatus_Type	1	Static Status

StaticStatus_ValOut	out	Clk_Clock StaticStatusVal _Type	1	Static Status valid
Timer Output				
Timer1ms_EvtOut	out	std_logic	1	Single clock cycle event every millisec-ond aligned with the adjusted clock
AXI4 Lite Slave				
AxiWriteAddrValid _ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady _RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress _AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt _DatIn	in	std_logic_vector	3	Write Address Pro-tocol
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 Pro-tocol
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
Adjustment Selector Input				
ExtSelect_DatIn	in	Clk_Select_Type	1	Which core shall be the source for the correction when Select_DatIn is external, this is feed from the outside
InSync Output				
InSync_DatOut	out	std_logic	1	Clock Adjustments were below a certain level
InHoldover_DatOut	out	std_logic	1	No new Clock Adjustments after in Sync for a defined number of seconds
Time Output				
ClockTime_DatOut	out	Clk_Time_Type	1	Adjusted Clock Time
ClockTime_ValOut	out	std_logic	1	Adjusted Clock Time valid
Time Adjustment Input				
TimeAdjustmentTod_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from TOD core
TimeAdjustmentTod_ValIn	in	std_logic;	1	Calculated new Time from TOD core valid
TimeAdjustmentIrig_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from IRIG core
TimeAdjustmentIrig_ValIn	in	std_logic;	1	Calculated new Time from IRIG core valid
TimeAdjustmentPps_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from PPS core
TimeAdjustmentPps_ValIn	in	std_logic;	1	Calculated new Time from PPS core valid

TimeAdjustmentPtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from PTP core
TimeAdjustmentPtp_ValIn	in	std_logic;	1	Calculated new Time from PTP core valid
TimeAdjustmentRtc_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from RTC core
TimeAdjustmentRtc_ValIn	in	std_logic;	1	Calculated new Time from RTC core valid
TimeAdjustmentDcf_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from DCF core
TimeAdjustmentDcf_ValIn	in	std_logic;	1	Calculated new Time from DCF core valid
TimeAdjustmentNtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from NTP core
TimeAdjustmentNtp_ValIn	in	std_logic;	1	Calculated new Time from NTP core valid
Offset Adjustment Input				
OffsetAdjustmentTod_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from TOD core
OffsetAdjustmentTod_ValIn	in	std_logic;	1	Calculated new Off-set from TOD core valid
OffsetAdjustmentIrig_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from IRIG core
OffsetAdjustmentIrig_ValIn	in	std_logic;	1	Calculated new Off-set from IRIG core valid
OffsetAdjustmentPps_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from PPS core
OffsetAdjustmentPps_ValIn	in	std_logic;	1	Calculated new Off-set from PPS core valid
OffsetAdjustmentPtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from PTP core

OffsetAdjustmentPtp_ValIn	in	std_logic;	1	Calculated new Offset from PTP core valid
OffsetAdjustmentRtc_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from RTC core
OffsetAdjustmentRtc_ValIn	in	std_logic;	1	Calculated new Offset from RTC core valid
OffsetAdjustmentDcf_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from DCF core
OffsetAdjustmentDcf_ValIn	in	std_logic;	1	Calculated new Offset from DCF core valid
OffsetAdjustmentNtp_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from NTP core
OffsetAdjustmentNtp_ValIn	in	std_logic;	1	Calculated new Offset from NTP core valid
Offset Adjustment Output				
OffsetAdjustment_DatOut	out	Clk_TimeAdjustment_Type	1	Calculated new Offset after the PI Servo loop
OffsetAdjustment_ValOut	out	std_logic;	1	Calculated new Offset after the PI Servo loop valid
Drift Adjustment Input				
DriftAdjustmentTod_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Drift from TOD core
DriftAdjustmentTod_ValIn	in	std_logic	1	Calculated new Drift from TOD core valid
DriftAdjustmentIrig_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Drift from IRIG core
DriftAdjustmentIrig_ValIn	in	std_logic	1	Calculated new Drift from IRIG core valid
DriftAdjustmentPps_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Drift from PPS core
DriftAdjustmentPps_ValIn	in	std_logic	1	Calculated new Drift from PPS core valid

DriftAdjustmentPtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from PTP core
DriftAdjustmentPtp_ValIn	in	std_logic	1	Calculated new Drift from PTP core valid
DriftAdjustmentRtc_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from RTC core
DriftAdjustmentRtc_ValIn	in	std_logic	1	Calculated new Drift from RTC core valid
DriftAdjustmentDcf_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from DCF core
DriftAdjustmentDcf_ValIn	in	std_logic	1	Calculated new Drift from DCF core valid
DriftAdjustmentNtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from NTP core
DriftAdjustmentNtp_ValIn	in	std_logic	1	Calculated new Drift from NTP core valid
Drift Adjustment Output				
DriftAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Drift after the PI Servo loop
DriftAdjustment_ValOut	out	std_logic;	1	Calculated new Drift after the PI Servo loop valid

Table 12: Clock

4.2 Design Parts

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

4.2.1 Clock Selector

4.2.1.1 Entity Block Diagram

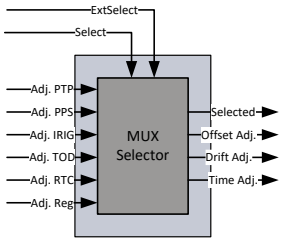


Figure 5: Clock Selector

4.2.1.2 Entity Description

Mux Selector

This module multiplexes multiple adjustment inputs. Only one adjustment is active at the time. This can be selected by a configuration register. PTP, PPS, TOD, IRIG, RTC, REG and EXT (where the selection is feed from externally) are potential sources. The multiplexer can also be set to disable any adjustment input.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ExtSelect_Gen	-	boolean	1	If external selection of sync source is used
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Adjustment Selector Input				

ExtSelect_DatIn	in	Clk_Select_Type	1	Which core shall be the source for the correction when Select_DatIn is external, this is feed from the outside
Select_DatIn	in	Clk_Select_Type	1	Which core shall be the source for the correction
Adjustment Selector Output				
Selected_DatOut	out	Clk_Select_Type	1	Which core was selected as the source for the correction
Time Adjustment Input				
TimeAdjustmentTod_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from TOD core
TimeAdjustmentTod_ValIn	in	std_logic;	1	Calculated new Time from TOD core valid
TimeAdjustmentIrig_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from IRIG core
TimeAdjustmentIrig_ValIn	in	std_logic;	1	Calculated new Time from IRIG core valid
TimeAdjustmentPps_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from PPS core
TimeAdjustmentPps_ValIn	in	std_logic;	1	Calculated new Time from PPS core valid
TimeAdjustmentPtp_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from PTP core
TimeAdjustmentPtp_ValIn	in	std_logic;	1	Calculated new Time from PTP core valid
TimeAdjustmentRtc_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from RTC core

TimeAdjustmentRtc_ValIn	in	std_logic;	1	Calculated new Time from RTC core valid
TimeAdjustmentDcf_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from DCF core
TimeAdjustmentDcf_ValIn	in	std_logic;	1	Calculated new Time from DCF core valid
TimeAdjustmentNtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from NTP core
TimeAdjustmentNtp_ValIn	in	std_logic;	1	Calculated new Time from NTP core valid
TimeAdjustmentReg_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Time from Registers
TimeAdjustmentReg_ValIn	in	std_logic;	1	Calculated new Time from Registers valid
Time Adjustment Output				
TimeAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Time from the se-lected core
TimeAdjustment_ValOut	out	std_logic;	1	Calculated new Time from the se-lected core valid
Offset Adjustment Input				
OffsetAdjustmentTod_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from TOD core
OffsetAdjustmentTod_ValIn	in	std_logic;	1	Calculated new Off-set from TOD core valid
OffsetAdjustmentIrig_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from IRIG core
OffsetAdjustmentIrig_ValIn	in	std_logic;	1	Calculated new Off-set from IRIG core valid
OffsetAdjustmentPps_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set from PPS core

OffsetAdjustmentPps_ValIn	in	std_logic;	1	Calculated new Offset from PPS core valid
OffsetAdjustmentPtp_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from PTP core
OffsetAdjustmentPtp_ValIn	in	std_logic;	1	Calculated new Offset from PTP core valid
OffsetAdjustmentRtc_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from RTC core
OffsetAdjustmentRtc_ValIn	in	std_logic;	1	Calculated new Offset from RTC core valid
OffsetAdjustmentDcf_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from DCF core
OffsetAdjustmentDcf_ValIn	in	std_logic;	1	Calculated new Offset from DCF core valid
OffsetAdjustmentNtp_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from NTP core
OffsetAdjustmentNtp_ValIn	in	std_logic;	1	Calculated new Offset from NTP core valid
OffsetAdjustmentReg_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from Registers
OffsetAdjustmentReg_ValIn	in	std_logic;	1	Calculated new Offset from Registers valid
Offset Adjustment Output				
OffsetAdjustment_DatOut	out	Clk_TimeAdjustment_Type	1	Calculated new Offset from the selected core
OffsetAdjustment_ValOut	out	std_logic;	1	Calculated new Offset from the selected core valid
Drift Adjustment Input				
DriftAdjustmentTod_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Drift from TOD core

DriftAdjustmentTod_ValIn	in	std_logic	1	Calculated new Drift from TOD core valid
DriftAdjustmentIrig_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from IRIG core
DriftAdjustmentIrig_ValIn	in	std_logic	1	Calculated new Drift from IRIG core valid
DriftAdjustmentPps_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from PPS core
DriftAdjustmentPps_ValIn	in	std_logic	1	Calculated new Drift from PPS core valid
DriftAdjustmentPtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from PTP core
DriftAdjustmentPtp_ValIn	in	std_logic	1	Calculated new Drift from PTP core valid
DriftAdjustmentRtc_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from RTC core
DriftAdjustmentRtc_ValIn	in	std_logic	1	Calculated new Drift from RTC core valid
DriftAdjustmentDcf_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from DCF core
DriftAdjustmentDcf_ValIn	in	std_logic	1	Calculated new Drift from DCF core valid
DriftAdjustmentNtp_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from NTP core
DriftAdjustmentNtp_ValIn	in	std_logic	1	Calculated new Drift from NTP core valid
DriftAdjustmentReg_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from Registers
DriftAdjustmentReg_ValIn	in	std_logic	1	Calculated new Drift from Registers valid
Drift Adjustment Output				
DriftAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from the selected core
DriftAdjustment_ValOut	out	std_logic;	1	Calculated new Drift from the selected core valid

Table 13: Clock Selector

4.2.2 Clock Adjuster

4.2.2.1 Entity Block Diagram

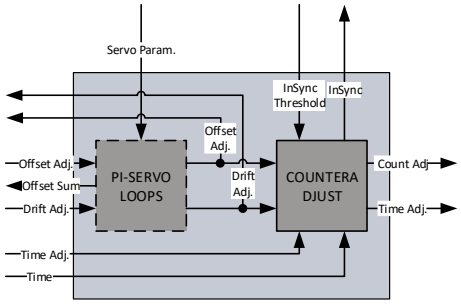


Figure 6: Clock Adjuster

4.2.2.2 Entity Description

PI Servo Loops

This module contains two individual PI Servo loops; one for the Drift and one for Offset correction. Both P and I can be configured individually for each servo loop. The drift is summed up since the corrected drift is only the change to the current frequency of the clock. All PI calculated values are also made available to the other IP cores to do the correct calculations based on the real adjustments.
The multiplications and divisions needed by the PI Servo are either binary multiplications and divisions or made with a fractional multiplier based on the FractionalMultiply_Gen generic.

$$OffsetOut = OffsetIn * \frac{OffsetMulP_{Gen}}{OffsetDivP_{Gen}} + \int OffsetIn * \frac{OffsetMulI_{Gen}}{OffsetDivI_{Gen}}$$

$$DriftOut = OldDrift + (DriftIn * \frac{DriftMulP_{Gen}}{DriftDivP_{Gen}} + \int DriftIn * \frac{DriftMulI_{Gen}}{DriftDivI_{Gen}})$$

The following servo loop parameters are used per default in the reference designs:

- Offset: 3/4 P, 3/16 I
- Drift: 3/4 P, 3/16 I

The servo loop parameters can be changed at runtime if the FractionalMultiply_Gen and DynamicServoParameters_Gen generic are true and no static config is done. In this case the PI Mul and Div generics are only used as the default values. The servo paramteres schall only be changed when the core is disabled.

Counter Adjuster

This module takes the adjustment inputs and converts them into periodic small adjustments which are then set to the Counter Clock. E.g. a drift of 1ns per 1000ns is converted to a 1ns adjustment every 50 clock cycles at a frequency of 50MHz. For offset it works similar. E.g. 50ns shall be corrected in 2000ns which will be converted into 1ns every second clock cycle at 50MHz. The conversion of course also works for non-integer corrections with minimal error (1/- 1ns per correction cycle). In addition it supervises the adjustments ad sets an InSync flag if 4 consecutive offset corrections were below a configurable threshold, a hard set on the clock will cause immediate InSync deasertion.

4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriodNanosecond_Gen	-	natural	1	Clock Period in Nanosecond
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Clock Adjuster				
ClockInHoldoverTimeoutSecond_Gen	-	natural	1	Value after how many seconds after in Sync without new correction values it goes in holdover Default 3 s
BypassServo_Gen	-	boolean	1	Bypass PI Servo loops
FractionalMultiply_Gen	-	boolean	1	Use a fractional multiplication (uses DSP slices) instead of binary multiplication and divisions (default)

DynamicServo Parameters_Gen	-	boolean	1	Allow to change the PI Servo parameters at runtime
DriftMulP_Gen	-	natural	1	Drift Proportional part ratio Numerator
DriftDivP_Gen	-	natural	1	Drift Proportional part ratio Denumerator
DriftMulI_Gen	-	natural	1	Drift Integral part ratio Numerator
DriftDivI_Gen	-	natural	1	Drift Integral part ratio Denumerator
OffsetMulP_Gen	-	natural	1	Offset Proportional part ratio Numerator
OffsetDivP_Gen	-	natural	1	Offset Proportional part ratio Denumerator
OffsetMulI_Gen	-	natural	1	Offset Integral part ratio Numerator
OffsetDivI_Gen	-	natural	1	Offset Integral part ratio Denumerator
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable Correction
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted Clock Time valid
Adjustment Selector Input				
Selected_DatIn	in	Clk_Select_Type	1	Which core was selected as the source for the correction
Servo Parameters Input				

ServoOffset FactorP_DatIn	in	std_logic_vector	32	Fractional Multiplier Offset P
ServoOffset FactorI_DatIn	in	std_logic_vector	32	Fractional Multiplier Offset I
ServoDrift FactorP_DatIn	in	std_logic_vector	32	Fractional Multiplier Drift P
ServoDrift FactorI_DatIn	in	std_logic_vector	32	Fractional Multiplier Drift I
Servo_ValIn	in	std_logic	1	New Parameters valid
InSync Threshold Input				
InSync Threshold_DatIn	in	std_logic_vector	32	When the clock is considered InSync in Nanoseconds
InSync Output				
InSync_DatOut	out	std_logic	1	Clock Adjustments were below the threshold for at least 4 adjustment cycles
InHoldover_DatOut	out	std_logic	1	No new Clock Adjustments after in Sync for a defined number of seconds
Time Adjustment Input				
TimeAdjustment _DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Time from selected core
TimeAdjustment _ValIn	in	std_logic;	1	Calculated new Time from selected core valid
Offset Adjustment Input				
OffsetAdjustment _DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from selected core
OffsetAdjustment _ValIn	in	std_logic;	1	Calculated new Offset from selected core valid
Offset Adjustment Output				

OffsetAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Off-set after the PI Servo loop
OffsetAdjustment_ValOut	out	std_logic;	1	Calculated new Off-set after the PI Servo loop valid
Drift Adjustment Input				
DriftAdjustment_DatIn	in	Clk_TimeAdjust-ment_Type	1	Calculated new Drift from selected core
DriftAdjustment_ValIn	in	std_logic	1	Calculated new Drift from selected core valid
Drift Adjustment Output				
DriftAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Drift after the PI Servo loop
DriftAdjustment_ValOut	out	std_logic;	1	Calculated new Drift after the PI Servo loop valid
Drift Adjustment Sum Output				
DriftAdjustmentSum_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Drift after the PI Servo loop as sum with old
DriftAdjustmentSum_ValOut	out	std_logic;	1	Calculated new Drift after the PI Servo loop as sum with old
Time Adjustment Output				
TimeAdjustment_DatOut	out	Clk_TimeAdjust-ment_Type	1	Calculated new Time from selected core to clock
TimeAdjustment_ValOut	out	std_logic;	1	Calculated new Time from selected core to clockvalid
Counter Adjustment Output				
CountAdjust-ment_DatOut	out	Clk_Count Adjustment_Type	1	Calculated new counter adjustment

				after PI servo and conversion to clock
CountAdjust-ment_ValOut	out	std_logic;	1	Calculated new counter adjustment after PI servo and conversion to clock valid

Table 14: Clock Adjuster

4.2.3 Clock Counter

4.2.3.1 Entity Block Diagram

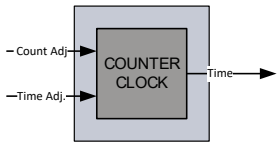


Figure 7: Clock Counter

4.2.3.2 Entity Description

Counter Clock

This is the adjustable counter clock with nanosecond resolution in a 32 bit second and 32 bit nanosecond format. It can take any input frequency, also non integer values with fractions. It normally adds the clock period to the nanoseconds counter every clock cycle but can add or subtract some extra nanosecond to do the clock adjustment. It also has an overwrite mode where the clock can be set. When the nanosecond counter overflows to the next second the second counter is incremented and the increment is added to the nanosecond part and a billion subtracted to keep the remainder. If fractions are used, every clock cycle the fraction numerator is summed up and when reaching the denominator value an extra nanosecond is added. The time domain is TAI, so after every reset the clock starts from 1.1.1970 at midnight which represents second 0.

4.2.3.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
Clock COunter				
ClockClkPeriodNano-second_Gen	-	natural	1	Clock Period in Nanosecond
ClockClkPeriodFract-Num_Gen	-	natural	1	Fractional Clock Period Numerator (0 if integer)
ClockClkPeriod-FractDeNum_Gen	-	natural	1	Fractional Clock Period Denumerator (0 if integer)
Ports				

System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Time Output				
ClockTime_DatOut	out	Clk_Time_Type	1	Adjusted Clock Time
ClockTime_ValOut	out	std_logic	1	Adjusted Clock Time valid
Time Adjustment Input				
TimeAdjustment_Dat	in	Clk_TimeAdjust-ment_Type	1	Overwrite Time
TimeAdjustment_Val	in	std_logic;	1	Overwrite Time valid
Counter Adjustment Input				
CountAdjust-ment_DatIn	in	Clk_Count Adjustment_Type	1	Calculated new counter adjustment after PI servo and conversion to clock
CountAdjust-ment_ValIn	in	std_logic;	1	Calculated new counter adjustment after PI servo and conversion to clock valid, make correc-tion

Table 15: Clock Counter

4.2.4 Clock Timer

4.2.4.1 Entity Block Diagram

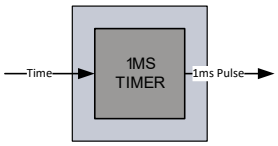


Figure 8: Clock Timer

4.2.4.2 Entity Description

1 Millisecond Timer

This module creates a single clock cycle pulse every millisecond aligned with the adjusted counter clock. This pulse is used by all NetTimeLogic's IP cores to calculate timeouts etc. If the counter clock makes a jump in time this can lead to a loss of a pulse or two pulses next to each other.

4.2.4.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriodNanosecond_Gen	-	natural	1	Clock Period in Nanosecond
Clock Adjuster				
TimerIntervalNanosecond_Gen	-	natural	1	Timer Interval between two events in Nanosecond
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 Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted Clock Time valid
Timer Output				

Timer_EvtOut	out	std_logic	1	Single clock cycle event every interval aligned with the adjusted clock

Table 16: Timer

4.2.5 Registerset

4.2.5.1 Entity Block Diagram

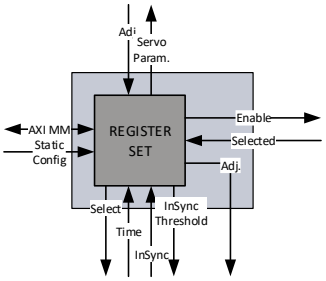


Figure 9: Registerset

4.2.5.2 Entity Description

Register Set

This module is an AXI4Lite Memory Mapped Slave. It provides access to all Registers and allows configuring the Adjustable Counter Clock. AXI4Lite only supports 32 bit wide data access, no byte enables, no burst, no simultaneous read and writes and no unaligned access. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the Datasets is done via signals and can be easily done from within the FPGA without CPU. For each parameter a valid signal is available, the enable signal shall be set last (or simultaneously). To change parameters the clock has to be disabled and enabled again. If in AXI mode, an AXI Master has to configure the Datasets with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime. For status supervision, similar to the static configuration the static status signals are available which allows to use the values also directly from within the FPGA without CPU.

4.2.5.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
Register Set				
ClockInSyncNanosecond_Gen	-	natural	1	Default value for the threshold when the

				clock is considered in sync
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
ExtSelect_Gen	-	boolean	1	If external selection of sync source is used
LogCorrections_Gen	-	boolean	1	Log corrections to registers
BypassServo_Gen	-	boolean	1	Bypass PI Servo loops
FractionalMultiply_Gen	-	boolean	1	Use a fractional multiplication (uses DSP slices) instead of binary multiplication and divisions (default)
DynamicServo Parameters_Gen	-	boolean	1	Allow to change the PI Servo parameters at runtime
DriftMulP_Gen	-	natural	1	Drift Proportional part ratio Numerator
DriftDivP_Gen	-	natural	1	Drift Proportional part ratio Denumerator
DriftMulI_Gen	-	natural	1	Drift Integral part ratio Numerator
DriftDivI_Gen	-	natural	1	Drift Integral part ratio Denumerator
OffsetMulP_Gen	-	natural	1	Offset Proportional part ratio Numerator
OffsetDivP_Gen	-	natural	1	Offset Proportional part ratio Denumerator
OffsetMulI_Gen	-	natural	1	Offset Integral part ratio Numerator

OffsetDivI_Gen	-	natural	1	Offset Integral part ratio Denumerator
AxiAddressRangeLow_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRangeHigh_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Clk_Clock StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Clk_Clock StaticConfigVal_Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Clk_Clock StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Clk_Clock StaticStatusVal_Type	1	Static Status valid
In Sync Input				
InSync_DatIn	in	std_logic	1	InSync flag
InHoldover_DatIn	in	std_logic	1	InHoldover flag
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 Pro-tocol

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
InSync Threshold Output				
InSync Threshold_DatOut	out	std_logic_vector	32	When the clock is considered InSync in Nanoseconds
Adjustment Selector Output				
Select_DatOut	out	Clk_Select_Type	1	Which core shall be the source for the correction
Servo Parameters Output				
ServoOffset FactorP_DatOut	out	std_logic_vector	32	Fractional Multiplier Offset P
ServoOffset FactorI_DatOut	out	std_logic_vector	32	Fractional Multiplier Offset I
ServoDrift FactorP_DatOut	out	std_logic_vector	32	Fractional Multiplier Drift P

ServoDrift FactorI_DatOut	out	std_logic_vector	32	Fractional Multiplier Drift I
Servo_ValOut	out	std_logic	1	New Parameters valid
Adjustment Selector Input				
Selected_DatIn	in	Clk_Select_Type	1	Which core was selected as the source for the correction
Offset Adjustment Input				
OffsetAdjustment_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Offset from selected core
OffsetAdjustment_ValIn	in	std_logic;	1	Calculated new Offset from selected core valid
Drift Adjustment Input				
DriftAdjustment_DatIn	in	Clk_TimeAdjustment_Type	1	Calculated new Drift from selected core
DriftAdjustment_ValIn	in	std_logic	1	Calculated new Drift from selected core valid
Time Adjustment Output				
TimeAdjustment_DatOut	out	Clk_TimeAdjustment_Type	1	New Time to set
TimeAdjustment_ValOut	out	std_logic;	1	New Time to set valid
Offset Adjustment Output				
OffsetAdjustment_DatOut	out	Clk_TimeAdjustment_Type	1	New Offset
OffsetAdjustment_ValOut	out	std_logic;	1	New Offset valid
Drift Adjustment Output				
DriftAdjustment_DatOut	out	Clk_TimeAdjustment_Type	1	New Drift
DriftAdjustment_ValOut	out	std_logic;	1	New Drift valid
Enable Output				
ClockEnable_DatOut	out	std_logic	1	Enable Adjustable Counter Clock

Table 17: 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 ClkStaticConfig_Con : Clk_ClockStaticConfig_Type := (
  TimeAdjustment                => Clk_TimeAdjustment_Type_Rst_Con,
  OffsetAdjustment              => Clk_TimeAdjustment_Type_Rst_Con,
  DriftAdjustment               => Clk_TimeAdjustment_Type_Rst_Con,
  ClockSelect                   => Clk_Select_Tod_Con,
  InSyncThreshold               => x"000001F4" - +/- 500 ns
);

constant ClkStaticConfigVal_Con : Clk_ClockStaticConfigVal_Type := (
  Enable_Val                    => '1',
  TimeAdjustment_Val            => '0',
  OffsetAdjustment_Val          => '0',
  DriftAdjustment_Val           => '0'
);
```

Figure 10: Static Configuration

The ClockSelect and InSyncThreshold should be set before enabling, but can also be changed when enabled. TimeAdjustment, DriftAdjustment and OffsetAdjustment can only be set once the clock is enabled and only have an effect if ClockSelect is set to REG.

4.3.2 AXI Configuration

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

```
-- CLOCK
-- Config
-- select: REG
AXI WRITE 10000008 00000006
-- enable adjustable counter clock
AXI WRITE 10000000 00000001
-- set time: 970000000 nanoseconds
AXI WRITE 10000020 39D10680
-- set time: 2 seconds
AXI WRITE 10000024 00000002
-- set valid bits
AXI WRITE 10000000 00000003
-- select: TOD
AXI WRITE 10000008 00000001
-- in sync threshold: 500 ns
AXI WRITE 10000050 000001F4
```

Figure 11: AXI Configuration

The Clock Select and In Sync Threshold registers should be set before enabling but can also be changed when enabled. Time, Drift and Offset can only be set once the clock is enabled and only have an effect if Clock Select is set to REG.
In the example the time is set first 2.97 seconds via register and then the TOD core selected for further adjustments.

4.4 Clocking and Reset Concept

4.4.1 Clocking

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

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

Table 18: 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 19: Resets

5 Resource Usage

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

5.1 Intel/Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (No PI Servo)	1868	3846	0	0
Maximal (With PI Servo)	3093	6484	0	0

Table 20: Resource Usage Intel/Altera

5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (No PI Servo)	1872	3878	0	0
Maximal (With PI Servo)	3094	6744	0	0

Table 21: 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 Clock testbench consist of 1 parse/port type: AXI.
For configuration and result checks an AXI read and write port is used. The time can be written and read back.

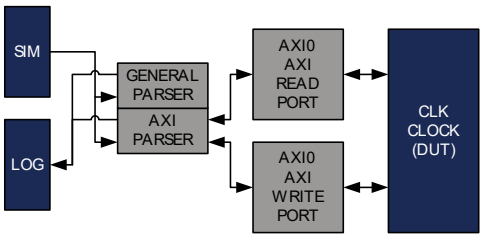


Figure 12: Testbench Framework

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/ClkClock/Script/run_Clk_Clock_Tb.tcl
```

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

8 Reference Designs

The Adjustable Counter Clock reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz) and an instance of the Adjustable Counter Clock IP core. 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 PPS 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.

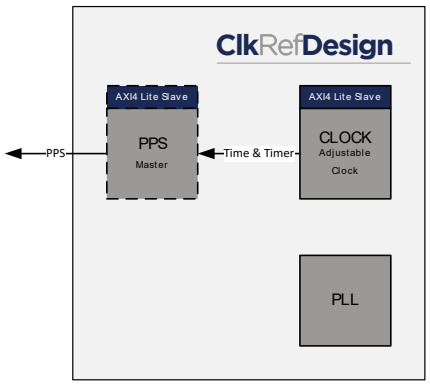


Figure 13: Reference Design

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/ClkClock/ClkClock.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

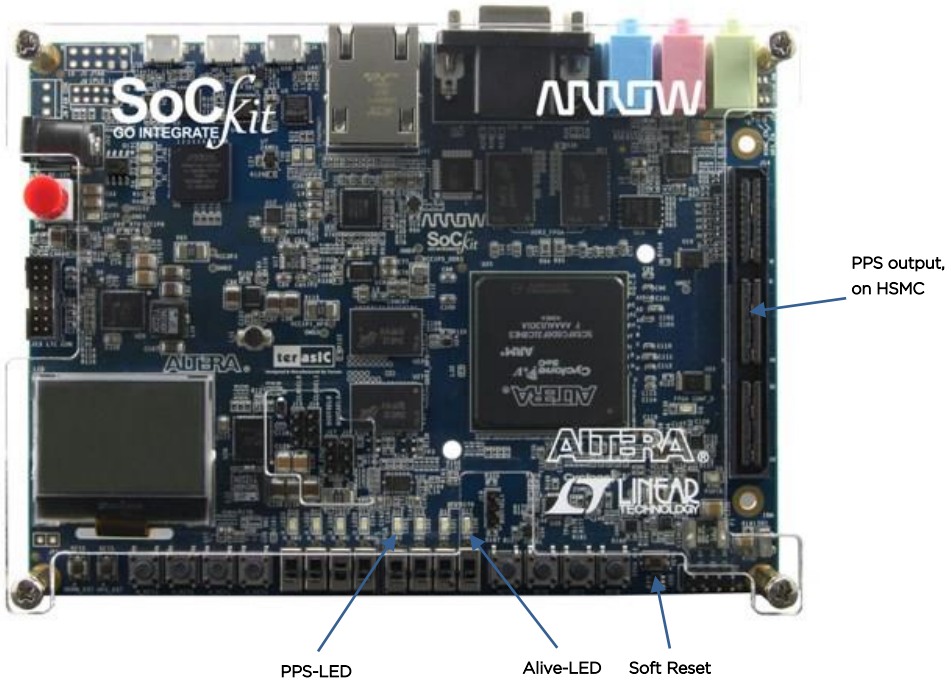


Figure 14: SocKit (source Terasic Inc)

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

8.2 AMD/Xilinx: Digilent Arty

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

1. Open Vivado 2019.1.
   Note: If a different Vivado version is used, see chapter 8.3.
2. Run TCL script /CLK/Refdesign/Xilinx/Arty/ClkClock/ClkClock.tcl
   a. This has to be run only the first time and will create a new Vivado Project
3. If the project has been created before open the project and do not rerun the project TCL
4. If the optional core PPS Master Clock is available add the files from the corresponding folders (PPS/Core, PPS/Library and PPS/Package) to the corresponding Library (PpsLib).

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

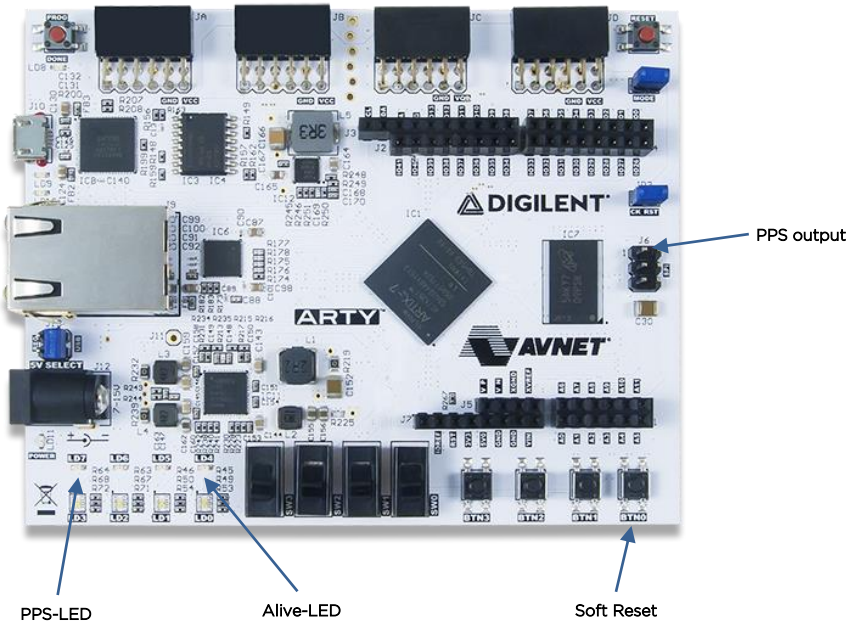


Figure 15: Arty (source Digilent Inc)

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