RtcMasterClock

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.1
Date	20.12.2017

Copyright Notice

Disclaimer

Overview

NetTimeLogic's Real Time Clock (RTC) Master Clock is a full hardware (FPGA) only implementation of a synchronization core able to synchronize an RTC or gets synchronized by an RTC. In addition, the time can be read and written aligned with the RTC PPS event. The RTC Master Clock core is designed to allow fast synchronization on startup, run in holdover mode when no other synchronization source is available and keep the Time of Day when powered down. The whole algorithms and calculations are implemented in the core, no CPU is required. This allows running RTC synchronization completely independent and standalone from the user application. The core can be configured/supervised either by signals or by an AXI4Light-Slave Register interface.

Key Features:

- Real Time Clock (RTC) Master Clock
- Supports DS1307 and MCP7941x (and compatible) Real Time Clocks (RTC) and self-configuration at startup
- Allows to synchronize the internal clock to the RTC and vise versa
- Built-in I2C controller with configurable baudrate
- Aligned writing and reading of time with the RTC PPS
- Hardware time conversion from Time of Day format (hh:mm:ss dd:mm:yyyy) into seconds since midnight 1.1.1970 (Linux, TAI, PTP)
- PI Servo Loop in hardware
- In combination with a Adjustable Counter Clock from NetTimeLogic: synchronization accuracy: +/- 100ns
- AXI4 Light register set or static configuration

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	04.01.2017	First draft
1.0	24.02.2017	First release and added CoreInfo type
1.1	20.12.2017	Status interface added

Table 1: Revision History

Content

Definitions

Definitions	
RTC Master Clock	A clock that can synchronize or be synchronized by an RTC
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
RTC	Real Time Clock
RM	RTC Master
TS	Timestamp
TB	Testbench
LUT	Look Up Table
FF	Flip Flop
RAM	Random Access Memory
ROM	Read Only Memory
FPGA	Field Programmable Gate Array
VHDL	Hardware description Language for FPGA's

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

The RTC Master Clock is meant as a co-processor handling an Real Time Clock (RTC). It can read and write the RTC registers via I2C and aligns these reads and writes aligned to the 1Hz square wave output (RTC PPS) of the RTC. For that, it self-configures the RTC to the correct mode at startup and enables the clock. The RTC maintains its own time reference also when the FPGA is powered down due to its battery backed oscillator. This is especially useful for Systems where the clock shall immediately run on a reference at startup and to have absolute time when no other synchronization source is available.
For synchronization, the core takes a snapshot of the local Counter Clock whenever a RTC PPS event happens, reads the seconds part of the time via I2C, converts the time from BCD encoded time of day format to binary seconds, calculates the offset and drift and adjusts the counter clock.
The RTC Master Clock is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement).
In addition to synchronization the core allows to read and write the RTC time via registers or signals. For a read it maintains also a nanoseconds part which represents the time until the time was read and converted since the PPS happens. This is useful when the read time is used for comparison, since at the time the valid flag is set the whole time (including nanoseconds) is accurate.

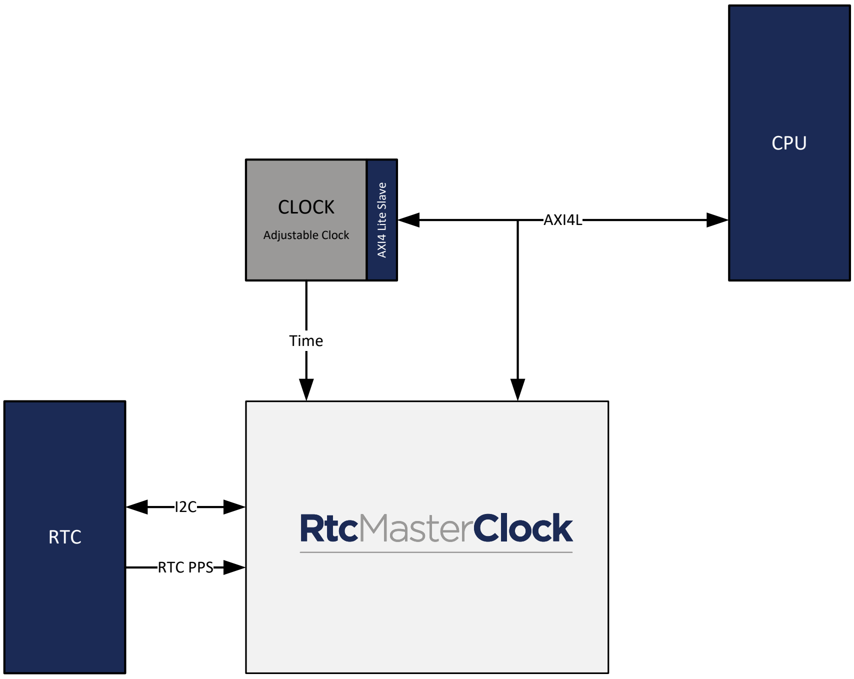

Figure 1: Context Block Diagram

1.2 Function

The RTC Master Clock takes a snapshot of the local Counter Clock whenever a RTC PPS event of configurable polarity happens, reads the seconds part of the time via I2C, converts the time from BCD encoded time of day format to binary seconds, calculates the offset and drift and adjusts the counter clock.
In addition to synchronization the core allows to read and write the RTC time via registers or signals. For a write the binary time in seconds/nanoseconds format is converted into BCD encoded time of day format and written to the corresponding time registers of the RTC via I2C right after the RTC PPS. For a read the corresponding time registers of the RTC are read via I2C right after the RTC PPS and decoded from the BCD encoded time of day format into binary time ins seconds/nanoseconds format. The read processor resets the nanosecond counter when a RTC PPS event occurs which runs during the reading and encoding, this gives nanosecond accuracy when the valid flag of the time is set.
Alignment with the RTC PPS is done to ensure the time value will not change during the I2C access. The RTC updates its time registers at the moment the RTC PPS happens. Also the time is always written/read in the order year, month, day, hour, minute, seconds. Reading and writing to the RTC can therefore take up to one second since it waits for the next RTC PPS to happen before accessing the RTC.

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.

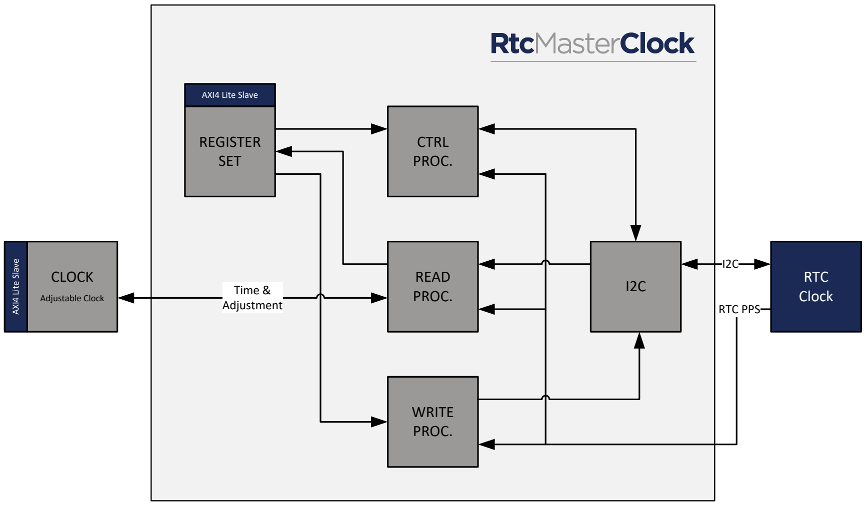

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration. It also enables a CPU to read and write the RTC time values.

I2C

This is the I2C controller converting the internal parallel register access to the serial I2C access. There is arbitration between the Read-, Write- and Control-Processor which can access the I2C bus.

Control Processor

This configures the RTC clock at startup to run in the correct mode and to put out the 1Hz square wave output. Once ready it releases the Read- and Write-Processor

Read Processor

This reads the time from the RTC via I2C at the RTC PPS event, converts it to binary time, calculates offset and drift and adjusts the clock.

Write Processor

This converts the time to BCD time of day format and writes the time to the RTC via I2C at the RTC PPS event.

2 RTC Basics

2.1 RTC Internals

A Real Time Clock (RTC) is a battery backed clock running on an local oscillator providing the time over register access and square wave output to the user. The oscillator frequency is divided to achieve a 1Hz internal signal which is the used for incrementing the time. To get sub second accuracy the 1Hz signal has to be mapped to the square wave output where the IP core can synchronize itself to by adjusting subsecond phase and the clock frequency.

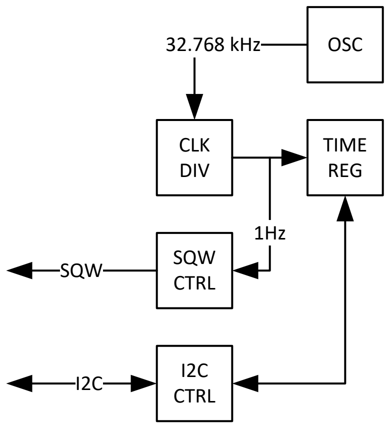

Figure 3: I2C Waveform

2.2 Interface

An RTC is commonly connected via an I2C interface. The I2C interface is a simple two wire bus containing a data line (SDA) and a clock line (SCL) allowing up to 127 slaves running at 100 or 400 kHz. The bus is open drain allowing all slaves to pull low using pull-ups to get the signals to high.
There are several access mechanisms defined but the RTC Master Clock described here only uses the byte wise access to the registers.

An I2C access consists always of the following parts (in this order):

- A Start condition (SDA goes from high to low while SC is high, also for repeated Start)
- 8 Data bits (MSB first)
- An Acknowledge (low for ACK, high for NACK, on write driven by the I2C slave, on read by the I2C Master)
- A Stop condition (SDA goes from low to high while SC is high)

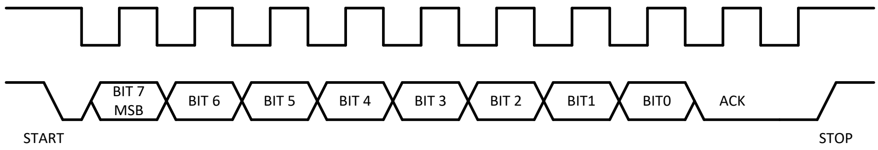

Figure 4: I2C Waveform

There are three different types of cycles in a byte wise I2C access:

- A device address cycle
- A register address cycle
- A data cycle

For a write the I2C access looks as following:

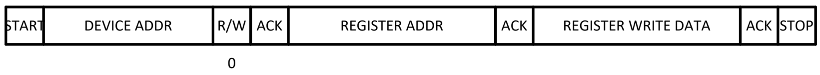

Figure 5: I2C Write Access

For a read the I2C access looks as following:

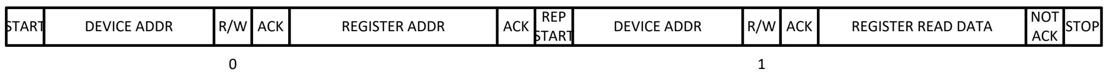

Figure 6: I2C Read Access

For more information on I2C check this link:
http://www.nxp.com/documents/user_manual/UM10204.pdf

In addition to the I2C interface an RTC normally consists of a square wave output which allow to generate different frequencies from the oscillator driving the RTC. The frequencies can be set normally to 32.768kHz, 8.192kHz, 4.096kHz and 1Hz. For this RTC Master Clock core the square wave output frequency must be set to 1Hz (PPS). Setting the frequency is done by the core via I2C during startup. The duty cycle of the square wave output is normally set to 50% and the polarity is active low. On the falling edge, the internal clock counter values are updated.

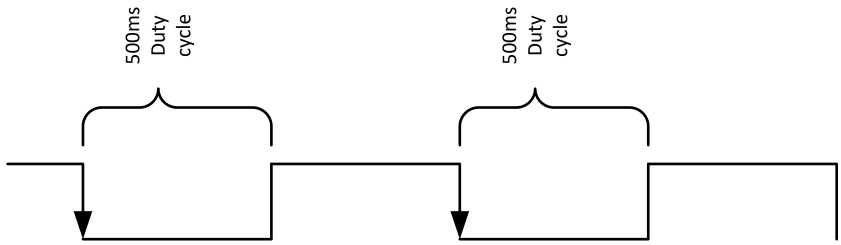

Figure 7: RTC PPS Waveform

2.3 RTC Registers

Depending on the RTC used the register set off the RTC differs slightly. What they all have in common is the time registers as binary encoded decimal value (BCD) in time of year format hh:mm:ss dd:mm:yy format and a control register for the squarewave output. To make use of the time in the FPGA and the other cores, the time must be converted from time of year in BCD to binary time since midnight 1.1.1970, taking leap years and into account.

The registersets of the two supported RTCs look as following:

DS1307

	MSB 7	6	5	4	3	2	1	LSB 0
Seconds 0x00	CH	SECONDS TEN			SECONDS ONE			
Minutes 0x01	-	MINUTES TEN			MINUTES ONE			
Hours 0x02	-	12/24	HOURS TEN: AM/PM	HOURS TEN: HOUR TEN	HOURS ONE			
Week Day 0x03	unused							
Day 0x04	-		DAY TEN		DAY ONE			
Day 0x05	-			MONTH TEN	MONTH ONE			
Seconds 0x06	YEAR TEN				YEAR ONE			
Control 0x07	OUT	-		SQWEN	-		RS1	RS0

1Hz
RS1 = 0
RS0 = 0

MCP7941X

	MSB 7	6	5	4	3	2	1	LSB 0
Seconds 0x00	ST	SECONDS TEN			SECONDS ONE			
Minutes 0x01	-	MINUTES TEN			MINUTES ONE			
Hours 0x02	-	12/24	HOURS TEN: AM/PM	HOURS TEN: HOUR TEN	HOURS ONE			
Week Day 0x03	-		OSC RUN	unused				
Day 0x04	-		DAY TEN		DAY ONE			
Day 0x05	-		LEAP YEAR	MONTH TEN	MONTH ONE			
Seconds 0x06	YEAR TEN				YEAR ONE			
Control 0x07	OUT	-	SQWEN	-	-		RS1	RS0

1Hz
RS1 = 0
RS0 = 0

Figure 8: RTC Registersets

2.4 UTC vs TAI time bases

The RTC clock contains the time of day on TAI base. UTC has an offset to TAI which is the time base normally used for the Counter Clock. This time offset has to be handled by the user so the local clock can still run on a TAI base. UTC in comparison to TAI or GPS time has so called leap seconds. A leap second is an additional second which is either added or subtracted from the current time to adjust for the earth rotation variation over time. Until 2016 UTC had additional 36 leap seconds, therefore TAI time is currently 36 seconds ahead of UTC. The issue with UTC time is, that the time makes jumps with the leap seconds which may cause that synchronized nodes go out of sync for a couple of seconds. Leap seconds are normally introduced at midnight of either the 30 of June or 31 of December. For an additional leap second the seconds counter of the UTC time will count to 60 before wrapping around to zero, for one fewer leap second the UTC second counter will wrap directly from 58 to 0 by skipping 59 (this has not happened yet).

3 Register Set

This is the register set of the RTC Master Clock. It is accessible via AXI4 Light 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
Rtc MasterControl Reg	Rtc Master Enable Control Register	0x00000000	RW
Rtc MasterStatus Reg	Rtc Master Error Status Register	0x00000004	WC
Rtc MasterPolarity Reg	Rtc Master Polarity Register	0x00000008	RW
Rtc MasterVersion Reg	Rtc Master Version Register	0x0000000C	RO
Rtc MasterTimeReadValueL Reg	RTC write Time Nanosecond Register	0x00000010	RO
Rtc MasterTimeReadValueH Reg	RTC write Time Second Register	0x00000014	RO
Rtc MasterTimeWriteValueL Reg	RTC write Time Nanosecond Register	0x00000020	RW
Rtc MasterTimeWriteValueH Reg	RTC write Time Second Register	0x00000024	RW

Table 4: Register Set Overview

3.2 Register Descriptions

3.2.1 General

3.2.1.1 RTC Master Control Register

Used for general control over the RTC Master Clock. To get a new time snapshot the time read flag has to be set and the read done flag is asserted as soon as the time is read from the RTC. Since the time values are multi register values, a set flag is available to mark validity of the whole value.

Rtc MasterControl Reg				
Reg Description				
31	30	29 ... 2	1	0
TIME_READ_DONE	TIME_READ		TIME_WRITE_VAL	ENABLE
RO	RW	RO	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: 29:2	RO
TIME_WRITE_VAL	Time Write Valid (autocleared)	Bit: 1	RW
ENABLE	Enable	Bit: 0	RW

3.2.1.2 RTC Master Status Register

Shows the current status of the RTC Master Clock. Each controller (Read, Write, Ctrl, I2C) has its own error flag.

Rtc MasterStatus 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
-																												I2C_ERROR	WRITE_ERROR	READ_ERROR	CTRL_ERROR
RO																												WC	WC	WC	WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 21:4	RO
I2C_ERROR	Error (sticky)	Bit: 3	WC
WRITE_ERRO	Error (sticky)	Bit: 2	WC
READ_ERROR	Error (sticky)	Bit: 1	WC
CTRL_ERROR	Error (sticky)	Bit: 0	WC

3.2.1.3 RTC Master Polarity Register

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

Rtc MasterPolarity 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	RTC PSS Signal Polarity (1 active high, 0 active low)	Bit: 0	RW

3.2.1.4 RTC Master 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.

Rtc MasterVersion Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VERSION																															
RO																															
0xXXXXXXXX																															
Offset: 0x000C																															

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

3.2.1.5 RTC Master Time Read Value Low Register

Time snapshot value nanosecond part. This represents the number of nanoseconds after the seconds pulse of the RTC when the time was converted. This is an internally generated value and not from the RTC.

Rtc MasterTimeReadValueL 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	Read Time Nanosecond	Bit: 31:0	RO

3.2.1.6 RTC Master Time Read Value High Register

Time snapshot value second part. This is the RTC time converted from time of day into seconds when the PPS has happened. Time is always read when the RTC PPS has ocured

Rtc MasterTimeReadValueH 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	Read Time Second	Bit: 31:0	RO

3.2.1.7 RTC Master Time Write Value Low Register

Write time nanoseconds part value. This is currently unused.

Rtc MasterTimeWriteValueL Reg
Reg Description
Bits 31–0: TIME_WRITE_NS
RW
Reset: 0x00000000
Offset: 0x0020

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

3.2.1.8 RTC Master Time Write Value High Register

Write time seconds part value. The time will be written to the Clock at the next RTC PPS event.

Rtc MasterTimeWriteValueH 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_WRITE_S																															
RW																															
Reset: 0x00000000																															
Offset: 0x0024																															

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

4 Design Description

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

4.1 Top Level – Rtc Master

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
RtcClockType_Gen	Rtc_ClockType_ Type	1	DS1307_E or MCP7941x_E are supported to define the type of RTC connected
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
I2cClkPeriod Nanosecond_Gen	natural	1	I2C clock Period in Nanosec-ond: Default for 100 kHz = 10000 ns
I2cAdress_Gen	natural	1	I2C 7 bit address of the RTC
InputDelay Nanosecond_Gen	natural	1	Input delay of the PPS from the output signal to the con-nector.
InputPolarity_Gen	boolean	1	true = high active, false = low active
AxiAddressRange Low_Gen	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	std_logic_vector	32	AXI Base Address plus Regis-terset Size Default plus 0xFFFF
Sim_Gen	boolean	1	If in Testbench simulation mode:

			true = Simulation, false = Synthesis

Table 5: Parameters

4.1.1.2 Structured Types

4.1.1.2.1 Clk_Time_Type

Defined in Clk_Package.vhd of library ClkLib

Type represents the time used everywhere. For this type overloaded operators + and – with different parameters exist.

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

Table 6: Clk_Time_Type

4.1.1.2.2 Clk_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

4.1.1.2.3 Rtc_MasterStaticConfig_Type

Defined in Rtc_MasterAddrPackage.vhd of library RtcLib
This is the type used for static configuration.

Field Name	Type	Size	Description
Polarity	std_logic	1	'1' = high active, '0' = low active
WriteTime	Clk_Time_Type	1	Time to write

Table 8: Rtc_MasterStaticConfig_Type

4.1.1.2.4 Rtc_MasterStaticConfigVal_Type

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

Field Name	Type	Size	Description
Enable_Val	std_logic	1	Enables the PPS Master
WriteTime_Val	std_logic	1	Writes the time

Table 9: Rtc_MasterStaticConfigVal_Type

4.1.1.2.5 Rtc_MasterStaticStatus_Type

Defined in Rtc_MasterAddrPackage.vhd of library RtcLib
This is the type used for static status supervision.

Field Name	Type	Size	Description
CoreInfo	Clk_CoreInfo_Type	1	Infor about the Cores state
ReadTime	Clk_Time_Type	1	Read time

Table 10: Rtc_MasterStaticConfig_Type

4.1.1.2.6 Rtc_MasterStaticStatusVal_Type

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

Field Name	Type	Size	Description
CoreInfo_Val	std_logic	1	Core Info valid
ReadTime_Val	std_logic	1	Read time valid

Table 11: Rtc_MasterStaticConfigVal_Type

4.1.1.3 Entity Block Diagram

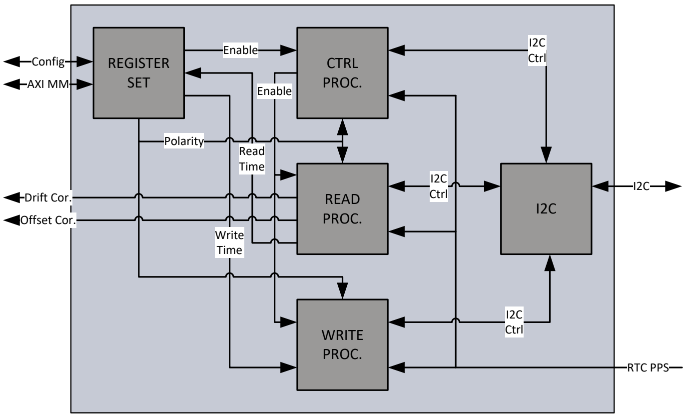

Figure 9: RTC Master Clock

4.1.1.4 Entity Description

I2C

This module is the I2C controller converting the internal parallel register access to the serial I2C access. There is arbitration between the Read-, Write- and Control-Processor which can access the I2C bus.
See 4.2.1 for more details.

Control Processor

This module configures the RTC clock at startup to run in the correct mode and to put out the 1Hz square wave output. Once ready it releases the Read- and Write-Processor
See 4.2.2 for more details.

Read Processor

This module reads the time from the RTC via I2C at the RTC PPS event, converts it to binary time, calculates offset and drift and adjusts the clock.
See 4.2.3 for more details.

Write Processor

This module converts the time to BCD time of day format and writes the time to the RTC via I2C at the RTC PPS event.
See 4.2.4 for more details.

Register Set

This module is an AXI Light Memory Mapped Slave. It provides access to all registers and allows configuring the RTC Master Clock. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the registers is done via signals and can be easily done from within the FPGA without CPU. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU. 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				
RtcClockType	-	Rtc_ClockType_Type	1	DS1307_E or MCP7941x_E are supported to define the type of RTC connected
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
I2cClkPeriod Nanosecond_Gen	-	natural	1	I2C clock Period in Nanosecond: Default for 100 kHz = 10000 ns
I2cAdress_Gen	-	natural	1	I2C 7 bit address of the RTC
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the connector to the input signal
InputPolarity_Gen	-	boolean	1	True: High active, False: Low active
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address

AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
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	Rtc_Master StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Rtc_Master StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Rtc_Master StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Rtc_Master StaticStatusVal _Type	1	Static Status valid
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
AXI4 Light 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
I2C Interface				
I2cClk_ClkOut	in	std_logic	1	I2C clock output
I2cSda_DatInOut	in-out	std_logic	1	I2C data input and output
RTC Pulse Per Second Input				
Rtc_EvtIn	in	std_logic	1	PPS input
Time Adjustment Output				
TimeAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Time to set hard (unused)
TimeAdjustment _ValOut	out	std_logic	1	Time valid (unused)
Offset Adjustment Output				
OffsetAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Offset between Master and Slave
OffsetAdjustment _ValOut	out	std_logic;	1	Calculated new Offset valid

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

Table 12: RTC Master Clock

4.2 Design Parts

The RTC Master 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 I2C

4.2.1.1 Entity Block Diagram

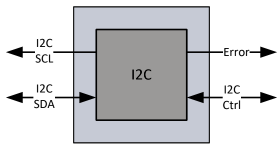

Figure 10: I2C

4.2.1.2 Entity Description

I2C

This module handles the I2C interface it uses the control signals from the parallel register interface and converts it into an I2C access. Via the register control interface the device address, the register address, read or write data and a read/write flag can be set.

The I2C module then creates the following I2C access cycles:

Figure 11: I2C Write Access

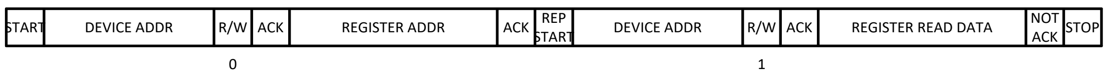

Figure 12: I2C Read Access

The module also supervises and creates the ACK signals and state and signals an error if not correct.
Once the access is completed it signals this to the Control-, Read- or Write-Processor, which will then start another register access until all bytes are handled.

The I2C controller only supports single byte access, no continuous read or block read or write. This is not the highest performance mechanism but the simplest and least resource consuming mechanism, which is perfectly fine since no high performance or throughput is required.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
I2C				
I2cClkPeriod Nanosecond_Gen	-	natural	1	I2C clock Period in Nanosecond: Default for 100 kHz = 10000 ns
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
I2C Interface				
Clk_ClkOut	out	std_logic	1	I2C Clock Output
Sda_DatIn	in	std_logic	1	I2C Data Input
Sda_DatOut	out	std_logic	1	I2C Data Output
SdaOe_ValOut	out	std_logic	1	I2C Data Output enable
I2C Control Interface				
ReadWrite_DatIn	in	std_logic	1	'1': Write, '0': Read
WriteData_DatIn	in	std_logic_vector	8	Write Data
ReadData_DatOut	out	std_logic_vector	8	Read Data
ChipAddress_AdrIn	in	std_logic_vector	7	I2C 7 bit RTC address
RegAddress_AdrIn	in	std_logic_vector	8	Register address
Access_ErrOut	out	std_logic	1	I2C error
Access_ValIn	in	std_logic	1	Do Access

Access_ValOut	out	std_logic	1	Access done

Table 13: I2C

4.2.2 Control Processor

4.2.2.1 Entity Block Diagram

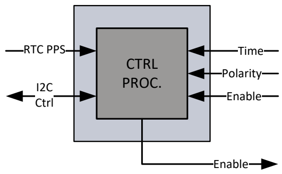

Figure 13: Control Processor

4.2.2.2 Entity Description

Control Processor

This module configures and enables the RTC on startup. It sets the correct frequency of 1Hz on the square wave output and puts the RTC in a mode where it continuously runs (also in battery holdover mode). Once enabled it waits for a second and one RTC PPS event before releasing the Read- and Write-Processor. This is done this way to ensure the Read- and Write Processor start at a second boundary. This procedure repeats when the core is disabled and reenabled. Once enabled the Control Processor stays silent.

4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
RtcClockType	-	Rtc_ClockType_Type	1	DS1307_E or MCP7941x_E are supported to define the type of RTC connected
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Control Processor				
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the con-

				nector to the input signal
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
RTC Pulse Per Second Polarity				
RtcPolarity_DatIn	in	std_logic	10	'1': High active, '0': Low active
RTC Pulse Per Second Input				
Rtc_EvtIn	in	std_logic	1	PPS input
RTC Error Output				
Rtc_ErrOut	out	std_logic	1	Indicates an access error
I2C Control Interface				
ReadWrite_DatOut	out	std_logic	1	'1': Write, '0': Read
WriteData_DatOut	out	std_logic_vector	8	Write Data
ReadData_DatIn	in	std_logic_vector	8	Read Data
ChipAddress_AdrOut	out	std_logic_vector	7	I2C 7 bit RTC address
RegAddress_AdrOut	out	std_logic_vector	8	Register address
AccessReq_ValOut	out	std_logic	1	Request I2C access
AccessAck_ValIn	in	std_logic	1	I2C Access granted
Access_ErrIn	in	std_logic	1	I2C error
Access_ValOut	out	std_logic	1	Do Access
Access_ValIn	in	std_logic	1	Access done
Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the configu-ration of the RTC
Enable Output				

Enable_EnaOut	out	std_logic	1	Enables the other processors

Table 14: Control Processor

4.2.3 Read Processor

4.2.3.1 Entity Block Diagram

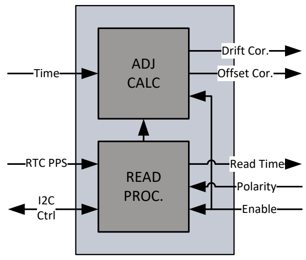

Figure 14: Read Processor

4.2.3.2 Entity Description

Read Processor

This module reads the time when the RTC PPS event is detected. A timestamp of the local clock is taken at this event and a nanosecond counter reset, both taking input delays into account. At this event it starts to read the years part first and reads all registers one by one until the seconds part. Then it decodes the BCD encoded register values into straight binary values. After that it converts the straight binary values of the time in time of day format into time in seconds since midnight 1.1.1970 (TAI) taking leap years into account. Last the nanosecond counter, which was reset at the RTC PPS event, is merged with the just decoded second part and the Read Time is validated. Before the nanosecond counter is merged with the decoded seconds value the time is passed to the Adjustment Calculator module together with the timestamp taken at the RTC PPS event for offset and drift calculation. The two values passed, represent the two times of the two clocks at the same moment in time and can therefore be used for comparison.
This mechanism guarantees that the time is always valid since the RTC only updates its time once a second and the read of individual registers hall not happen during an update cycle, but on the other hand this also only allows to update the time once a second.

Adjustment Calculator

This module calculates the drift and offset of the local clock against the RTC time and corrects it. After enabling or error detection the calculation waits for two consecutive time reads of the Read Processor before adjusting the clock again.

4.2.3.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
RtcClockType	-	Rtc_ClockType_Type	1	DS1307_E or MCP7941x_E are supported to define the type of RTC connected
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Read Processor				
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the con-nector to the input signal
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
RTC Pulse Per Second Polarity				
RtcPolarity_DatIn	in	std_logic	10	'1': High active, '0': Low active
RTC Pulse Per Second Input				
Rtc_EvtIn	in	std_logic	1	PPS input
RTC Error Output				
Rtc_ErrOut	out	std_logic	1	Indicates an access error

RTC Read Time Output				
ReadTime_DatOut	out	Clk_Time_Type	1	RTC Read Time
ReadTime_ValOut	out	std_logic	1	RTC Read Time valid
Offset Adjustment Output				
OffsetAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Offset between Master and Slave
OffsetAdjustment _ValOut	out	std_logic;	1	Calculated new Offset valid
Drift Adjustment Output				
DriftAdjustment _DatOut	out	Clk_TimeAdjustment _Type	1	Calculated new Drift between Master and Slave
DriftAdjustment _ValOut	out	std_logic;	1	Calculated new Drift valid
Offset Adjustment Input				
OffsetAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Offset after the PI Servo loop
OffsetAdjustment _ValIn	in	std_logic;	1	Calculated new Offset after the PI Servo loop valid
Drift Adjustment Input				
DriftAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Calculated new Drift after the PI Servo loop
DriftAdjustment _ValIn	in	std_logic	1	Calculated new Drift after the PI Servo loop valid
I2C Control Interface				
ReadWrite_DatOut	out	std_logic	1	'1': Write, '0': Read
WriteData_DatOut	out	std_logic_vector	8	Write Data
ReadData_DatIn	in	std_logic_vector	8	Read Data
ChipAddress_AdrOut	out	std_logic_vector	7	I2C 7 bit RTC address
RegAddress_AdrOut	out	std_logic_vector	8	Register address
AccessReq_ValOut	out	std_logic	1	Request I2C access

AccessAck_ValIn	in	std_logic	1	I2C Access granted
Access_ErrIn	in	std_logic	1	I2C error
Access_ValOut	out	std_logic	1	Do Access
Access_ValIn	in	std_logic	1	Access done
Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the reading of the RTC and Clock adjustment

Table 15: Read Processor

4.2.4 Write Processor

4.2.4.1 Entity Block Diagram

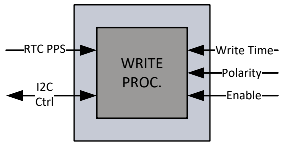

Figure 15: Write Processor

4.2.4.2 Entity Description

Write Processor

This module writes the time when the RTC PPS event is detected and a write time request is pending. At this event it starts to convert the time in seconds since midnight 1.1.1970 (TAI) into straight binary values in the time of day format. Then it encodes the straight binary values into BCD encoded values. After that it writes the BCD encoded values to the RTC by writing the years part first and writing all registers one by one until the seconds part.

This mechanism guarantees that the time is always valid when written and the RTC updates after the write, but on the other hand this also only allows to update the time once a second. When the time shall be written, an extra second shall be added to the time and the write command shall be also set before and aligned with the next RTC PPS event, this guarantees that the right second is written since RTC also updates its only once a second and a write to the individual registers shall not happen during an update cycle.

4.2.4.3 Entity Declaration

Name	Dir	Type	Size	Description
		Generics		
General				
RtcClockType	-	Rtc_ClockType_Type	1	DS1307_E or MCP7941x_E are supported to define the type of RTC connected
ClockClkPeriod	-	natural	1	Clock Period in

Nanosecond_Gen				Nanosecond
Sim_Gen	-	boolean	1	If in Testbench simulation mode
Write Processor				
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the PPS from the connector to the input signal
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Time Input				
ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
RTC Pulse Per Second Polarity				
RtcPolarity_DatIn	in	std_logic	10	'1': High active, '0': Low active
RTC Pulse Per Second Input				
Rtc_EvtIn	in	std_logic	1	PPS input
RTC Error Output				
Rtc_ErrOut	out	std_logic	1	Indicates an access error
RTC Write Time Input				
WriteTime_DatIn	in	Clk_Time_Type	1	RTC Write Time
WriteTime_ValIn	in	std_logic	1	RTC Write Time valid
RTC Time Adjustment Input				
TimeAdjustment _DatIn	in	Clk_TimeAdjustment _Type	1	Time to set hard
TimeAdjustment _ValIn	in	std_logic	1	Time valid
I2C Control Interface				
ReadWrite_DatOut	out	std_logic	1	'1': Write, '0': Read

WriteData_DatOut	out	std_logic_vector	8	Write Data
ReadData_DatIn	in	std_logic_vector	8	Read Data
ChipAddress_AdrOut	out	std_logic_vector	7	I2C 7 bit RTC address
RegAddress_AdrOut	out	std_logic_vector	8	Register address
AccessReq_ValOut	out	std_logic	1	Request I2C access
AccessAck_ValIn	in	std_logic	1	I2C Access granted
Access_ErrIn	in	std_logic	1	I2C error
Access_ValOut	out	std_logic	1	Do Access
Access_ValIn	in	std_logic	1	Access done
Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the writing of the RTC

Table 16: Write Processor

4.2.5 Registerset

4.2.5.1 Entity Block Diagram

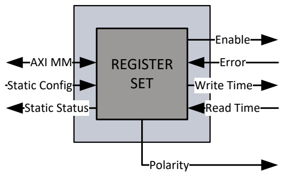

Figure 16: Registerset

4.2.5.2 Entity Description

Register Set

This module is an AXI Light Memory Mapped Slave. It provides access to all registers and allows configuring the RTC Master Clock. AXI4 Light only supports 32 bit wide data access, no byte enables, no burst, no simultaneous read and writes and no unaligned access. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the registers is done via signals and can be easily done from within the FPGA without CPU. For each parameter a valid signal is available, the enable signal shall be set last (or simultaneously). To change configuration parameters the clock has to be disabled and enabled again, the time can be changed at runtime. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.

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				
General				
InputPolarity_Gen	-	boolean	1	True: High active, False: Low active
Register Set				
StaticConfig_Gen	-	boolean	1	If Static Configura-

				tion or AXI is used
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Rtc_Master StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Rtc_Master StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Rtc_Master StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Rtc_Master StaticStatusVal _Type	1	Static Status valid
AXI4 Light 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
RTC Pulse Per Second Polarity				
RtcPolarity_DatOut	out	std_logic	10	'1': High active, '0': Low active
Pulse Per Second Error Input				
Rtc_ErrIn	in	std_logic_vector	4	Indicates a time jump
Read Time Input				
ReadTime_DatIn	in	Clk_Time_Type	1	RTC Read Time
ReadTime_ValIn	in	std_logic	1	RTC Read Time valid
Write Time Output				
WriteTime_DatOut	out	Clk_Time_Type	1	RTC Write Time
WriteTime_ValOut	out	std_logic	1	RTC Write Time valid
Enable Output				
RtcMaster Enable_DatOut	out	std_logic	1	Enables the RTC core

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 RtcStaticConfigMaster_Con : Rtc_MasterStaticConfig_Type := (
  Polarity                      => '1',
  WriteTime            => (
    Second              => x"12345678", -- seconds
    Nanosecond          => (others => '0'), -- no nanoseconds
    Fraction            => (others => '0'), -- no fractions
    Sign                => '0', -- UTC correct in positive
    TimeJump            => '0'),
);

constant RtcStaticConfigValMaster_Con : Rtc_MasterStaticConfigVal_Type := (
  Enable_Val                    => '1'
  WriteTime_Val                    => '1'
);
```

Figure 17: Static Configuration

The time can be written at runtime.

4.3.2 AXI Configuration

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

```
-- RTC MASTER
-- Config
-- enable RTC Master
AXI WRITE 10000000 00000001
-- write time
AXI AXI0 WRITE 10000024 12345678
AXI AXI0 WRITE 10000000 00000003
```

Figure 18: AXI Configuration

In the example the core is enabled and the time written.

4.4 Clocking and Reset Concept

4.4.1 Clocking

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

Clock	Frequency	Description
System		
System Clock	50MHz (Default)	System clock where the RTC Master runs on as well as the counter clock etc.
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock
I2C Interface		
I2C Clock	100kHz (Default)	I2C clock generated out of 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 Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static config)	1830	5831	0	0
Maximal (AXI config)	1972	5943	0	0

Table 20: Resource Usage Altera

5.2 Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static config)	1789	6552	0	0
Maximal (AXI config)	1927	6713	0	0

Table 21: Resource Usage Xilinx

6 Delivery Structure

```
AXI                                 -- AXI library folder
 |-Library                          -- AXI library component sources
 |-Package                          -- AXI library package sources

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

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

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

SIM                                 -- SIM library folder
 |-Doc                              -- SIM library command documentation
 |-Package                          -- SIM library package sources
 |-Testbench                        -- SIM library testbench template sources
 |-Tools                            -- SIM simulation tools
```

7 Testbench

The Tod Slave testbench consist of 3 parse/port types: AXI, CLK and RTC.
The RTC Slave port takes the time of the CLK port instance as initial reference, this can be overwritten by the ip core. The RTC Slave has an internal RTC shadow register set which the core can read and write via I2C. The RTC Slave port runs an internal clock which runs with the frequency of the CLK port and generates the RTC PPS event if enabled.
In addition for configuration and result checks an AXI read and write port is used in the testbench and for accessing more than one AXI slave also an AXI interconnect is required.

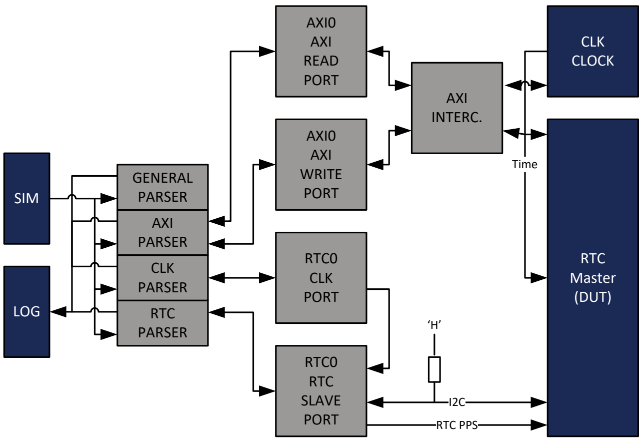

Figure 19: Testbench Framework

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/RTC/Testbench/Core/RtcMaster/Script/run_Rtc_Master_Tb.tcl
```

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

8 Reference Designs

The RTC Master reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the RTC Master Clock IP core and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately). The Reference Design is intended to be connected to a DS1307 or MCP7941x (or compatible) Real Time Clocks (RTC) which has an I2C interface, a 1Hz square wave output and a compatible register set of one of the RTC types supported. 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.

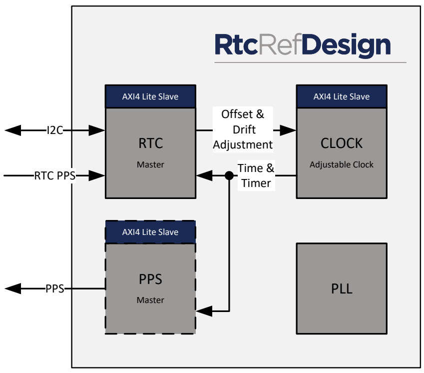

Figure 20: Reference Design

8.1 Altera: Terasic SocKit

The SocKit board is an FPGA board from Terasic Inc. with a Cyclone V SoC FPGA from 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 /RTC/Refdesign/Altera/SocKit/RtcMaster/RtcMaster.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

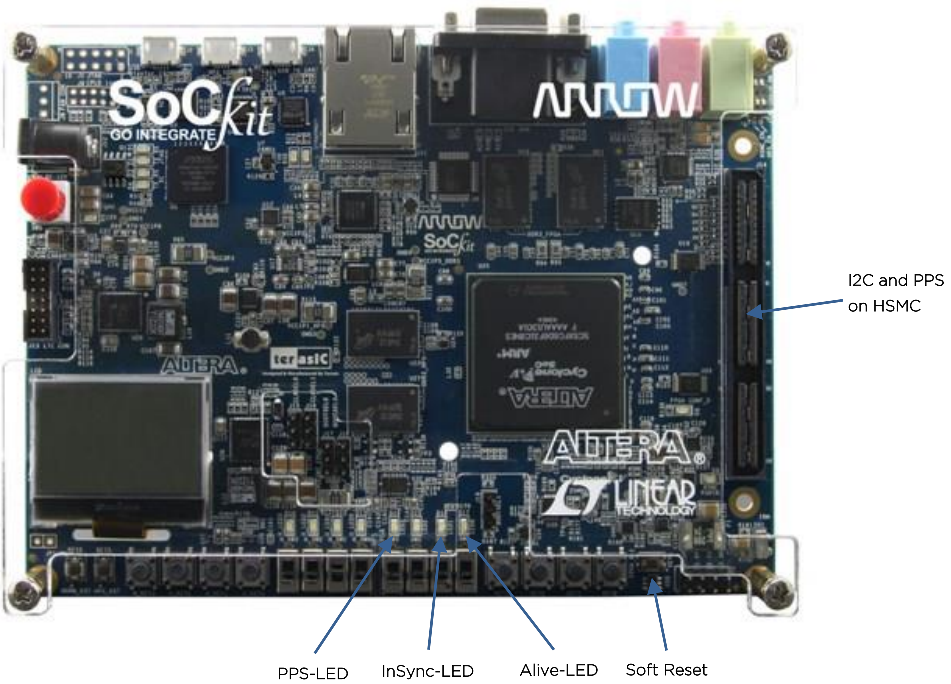

Figure 21: SocKit (source Terasic Inc)

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

8.2 Xilinx: Digilent Arty

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

1. Open Vivado 2017.4
2. Run TCL script /RTC/Refdesign/Xilinx/Arty/RtcMaster/RtcMaster.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

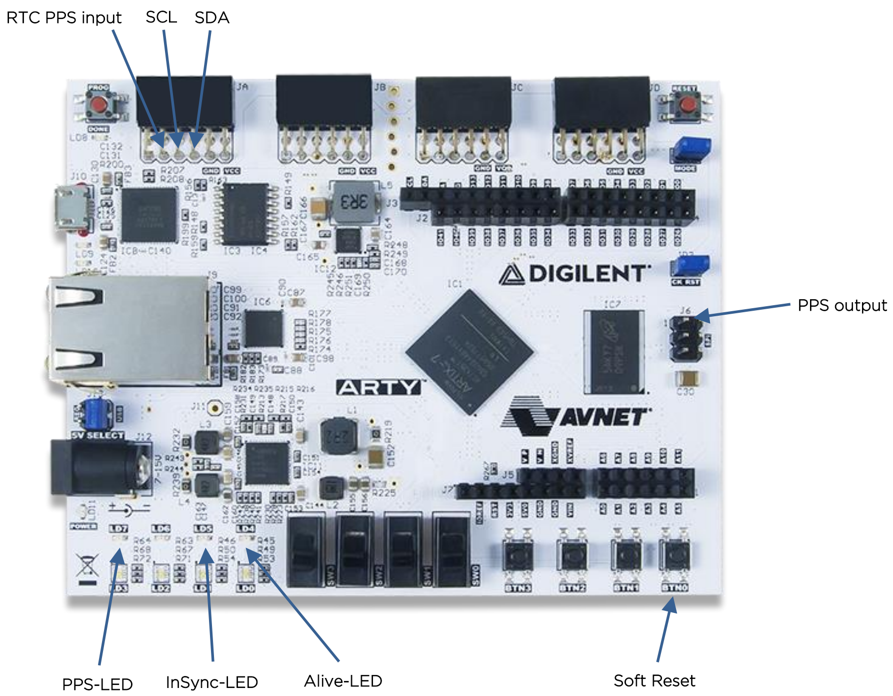

Figure 22: Arty (source Digilent Inc)

8.2.1 RTC CLock

The RTC Clock used in the reference design is a PMOD RTC from Digilent Inc. (http://store.digilentinc.com/pmod-rtcc-real-time-clock-calendar) which can be directly connected to the PMOD JA on the Arty. The RTC PPS has to be connected via a cable from the pinheader of the PMOD RTC to the pin on the upper row of PMOD JA next to the SCL (See Figure 22:)
The RTC needs an additional battery to be able to update the time when the board is powered down.

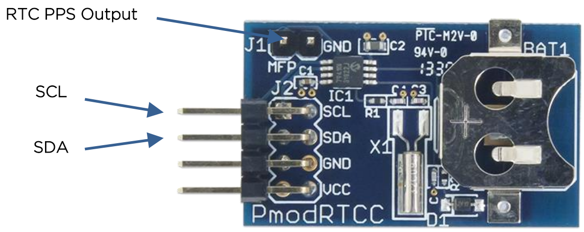

Figure 23: PMOD RTC (source Digilent Inc)

A List of tables

B List of figures