

# DcfSlaveClock

## Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.1
Date	03.01.2023

## Copyright Notice

Copyright © 2023 NetTimeLogic GmbH, Switzerland. All rights reserved.

Unauthorized duplication of this document, in whole or in part, by any means, is prohibited without the prior written permission of NetTimeLogic GmbH, Switzerland.

All referenced registered marks and trademarks are the property of their respective owners

## 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 DCF Slave Clock is a full hardware (FPGA) only implementation of a synchronization core able to synchronize to a DCF-77 input.

The whole time frame decoding, algorithms and calculations are implemented in the core, no CPU is required. This allows running DCF synchronization completely independent and standalone from the user application. The core can be configured either by signals or by an AXI4Lite-Slave Register interface.

## Key Features:

- DCF Slave Clock
- Supports DCF-77 format
- Optional Signal Filter
- DCF decoding and time format conversion
- DCF supervision
- Input delay compensation
- Air propagation delay compensation
- Additional seconds correction to convert between UTC and TAI time (or any other time base)
- Synchronization accuracy: +/- 10 ms
- AXI4Lite register set or static configuration
- Timestamp resolution with 50 MHz system clock: 20ns
- Hardware PI Servo

## Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	27.12.2017	First draft
1.0	10.09.2018	First release
1.1	03.01.2023	Added Vivado upgrade version description

Table 1: Revision History

# Content

<b>1</b>	<b>INTRODUCTION</b>	<b>9</b>
1.1	Context Overview	9
1.2	Function	9
1.3	Architecture	10
<b>2</b>	<b>DCF BASICS</b>	<b>12</b>
2.1	Interface	12
2.2	Delays	13
2.3	UTC vs TAI time bases	13
<b>3</b>	<b>REGISTER SET</b>	<b>15</b>
3.1	Register Overview	15
3.2	Register Descriptions	16
3.2.1	General	16
<b>4</b>	<b>DESIGN DESCRIPTION</b>	<b>22</b>
4.1	Top Level – Dcf Slave	22
4.2	Design Parts	30
4.2.1	RX Processor	30
4.2.2	Registerset	34
4.3	Configuration example	37
4.3.1	Static Configuration	37
4.3.2	AXI Configuration	37
4.4	Clocking and Reset Concept	38
4.4.1	Clocking	38
4.4.2	Reset	38

---

<b>5</b>	<b>RESOURCE USAGE</b>	<b>40</b>
5.1	Intel/Altera (Cyclone V)	40
5.2	AMD/Xilinx (Artix 7)	40
<b>6</b>	<b>DELIVERY STRUCTURE</b>	<b>41</b>
<b>7</b>	<b>TESTBENCH</b>	<b>42</b>
7.1	Run Testbench	42
<b>8</b>	<b>REFERENCE DESIGNS</b>	<b>44</b>
8.1	Intel/Altera: Terasic SocKit	44
8.2	AMD/Xilinx: Digilent Arty	45
8.3	AMD/Xilinx: Vivado version	46

## Definitions

Definitions	
DCF-77	German longwave time signal and standard-frequency radio station. The DCF77 station signal carries an amplitude-modulated, pulse-width coded 1-bit/s data signal. The transmitted data signal is repeated every minute.
DCF Slave Clock	A clock that can synchronize itself to a DCF input
PI Servo Loop	Proportional-integral servo loop, allows for smooth corrections
Offset	Phase difference between clocks
Drift	Frequency difference between clocks

Table 2: Definitions

## Abbreviations

Abbreviations	
AXI	AMBA4 Specification (Stream and Memory Mapped)
BCD	Binary Coded Decimal
PWM	Pulse Width Modulation
PPM	Pulse Per Minute
PPS	Pulse Per Second
IRQ	Interrupt, Signaling to e.g. a CPU
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 DCF Slave Clock is meant as a co-processor handling a DCF input. It takes a DCF input, converts the pulse width modulated (PWM) data stream into a binary format, decodes the BCD encoded time of day and converts it to a second/nanosecond format. In parallel it takes a timestamp when the reference marker is detected. It then calculates the offset and drift of the local clock against the reference clock from the DCF sender and corrects it.

The DCF Slave Clock is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Lite slave for configuration and status supervision from a CPU, this is however not required since the DCF Slave Clock can also be configured statically via signals/constants directly from the FPGA.

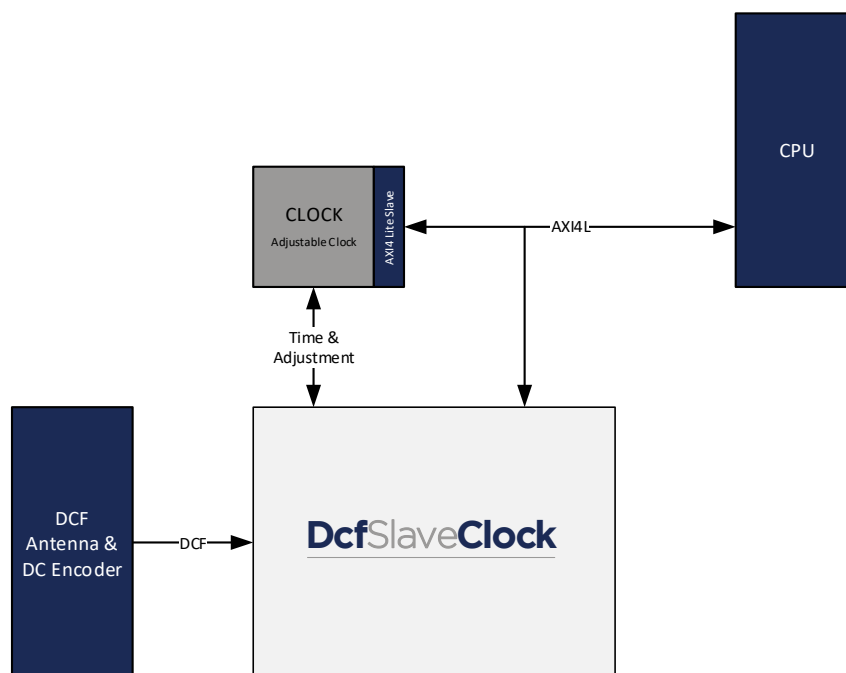


Figure 1: Context Block Diagram

## 1.2 Function

The DCF Slave Clock takes a DCF signal, decodes the DCF input data stream and converts the signal from a PWM signal to a BCD encoded time and converts it then into UTC in time of day format. In parallel it detects the start of minute condition

and generates a PPM. Then it converts the UTC time in time of day format into TAI format in seconds since 1.1.1970 without leap seconds. It also takes a timestamp of the rising edge of the PPM. After 2 consecutive correct PPM timestamps are taken and 2 consecutive DCF time frames were decoded, the calculation of offset and drift is started. For offset correction an additional air propagation delay and time base correction value is added to the calculation to compensate the propagation delay between the master and slave and the offset between the different time-domains (UTC, TAI etc.). Offset and drift corrections are feed to the PI servo loops of the Adjustable Counter Clock Core and the output of the Servo loop used for the next calculations. In case of an error the correction is stopped until two DCF time frames were correctly decoded again after the error flag was de-asserted.

## 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.

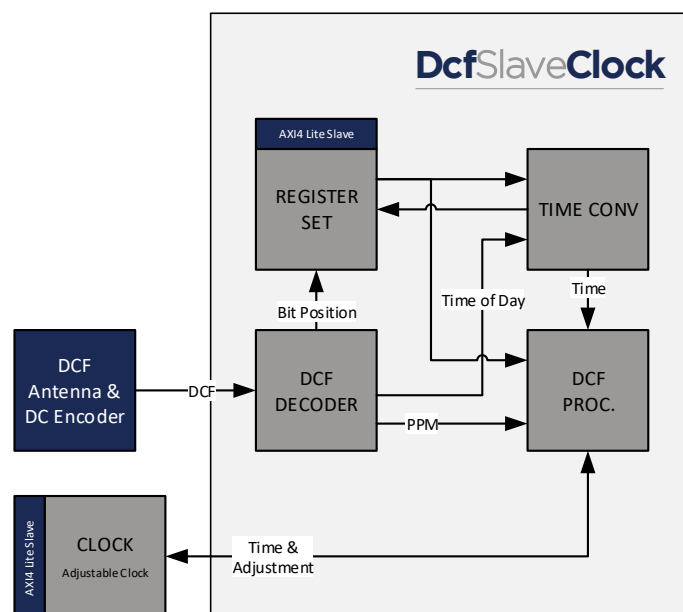


Figure 2: Architecture Block Diagram

### Register Set

This block allows reading status values and writing configuration.

### DCF Decoder

This block decodes the DCF input data stream and converts the signal from a PWM signal to a BCD encoded time and converts it then into UTC in time of day format. In parallel it detects the start of second condition and generates a PPS.

### **Time Converter**

This block converts the UTC time in time of day format into TAI format in seconds since 1.1.1970 without leap seconds

### **DCF Processor**

This block takes a timestamp of the rising edge of the DCF marker, takes the propagation delay of the cable into account, calculates the offset and the drift and adjusts the local clock.

## 2 DCF Basics

### 2.1 Interface

DCF is a very simple wireless interface amplitude modulated on a base frequency of 77,5 kHz. The sender is a 50kW sender near Frankfurt, Germany and reaches a distance of around 2000km. A DCF antenna and encoder which has a very narrowband receiver at 77,5kHz create a Pulse Width Modulated (PWM) symbol. The period of a symbol is 1000 ms. A duty cycle of 100 ms marks a “Zero”, a duty cycle of 200 ms marks a “One” and no duty cycle at all marks a “Mark” symbol. It is a continuous data stream of “One”, “Zero” and “Mark” symbols with symbol patterns to mark the beginning of a time frame. The reference point is the edge to the active level of bit 0; this shall be at the minute overflow of the reference clock. This edge shall be very accurate compared to the other edges of the symbols of a time frame. A time frame is repeated once every minute, therefore the number of symbols in a time frame is fixed to 60.

A DCF-77 time frame contains the following:

- Weather Info
- Summer/Winter Time announcement (A1)
- Summer/Winter Time (Z1 & Z2)
- Leap Second announcement (A2).
- Time Marker (S)
- Minute (BCD encoded)
- Even Parity Minute (P1)
- Hour (BCD encoded)
- Even Parity Hour (P2)
- Day (BCD encoded)
- Day of Week (BCD encoded, unused)
- Minute (BCD encoded)
- Year (BCD encoded)
- Even Parity Date (P3)
- Minute Marker

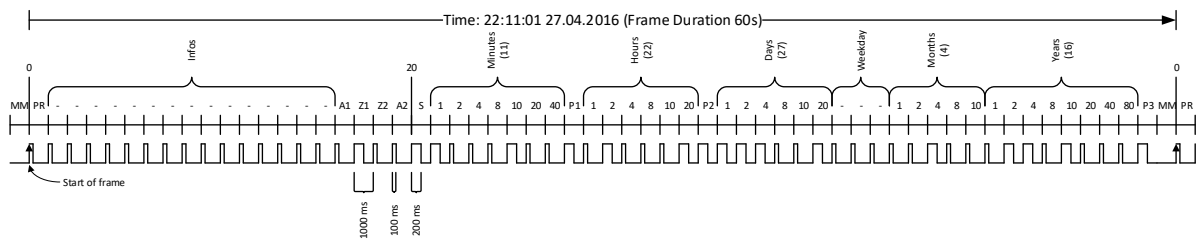


Figure 3: DCF-77 time frame

A DCF network is a one-to-many configuration: one DCF sender synchronizes multiple DCF slaves of different distance from the master.

DCF can also be generated on a wire similar to IRIG and transferred as PWM encoded DCLS signal

## 2.2 Delays

There are two kinds of delays in a DCF System. One is the input delay of the DCF signal from the antenna to the core; this shall be constant and is compensated for. The second delay is the propagation delay of the signal from the master to the slave. This is dependent on the distance from the sender in Mainflingen, Germany or the wire length in case of wired DCF: 15cm are equal to roughly 1ns of propagation delay. This delay can be set in the core.

## 2.3 UTC vs TAI time bases

DCF time frames contain the time of day on UTC base. UTC has an offset to TAI which is the time base normally used for the Counter Clock. This time offset can be set in the core 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).

Be aware that this core takes no additional precautions to handle leap seconds, so it will make a time jump at a UTC leap second and will lose synchronization since it thinks that it has an offset of one second at tries to readjust this offset. A way to avoid this is to disable the adjustment at the two dates right before midnight (e.g. one minute earlier), wait for the leap second to happen, fetch some time server to get the new offset between TAI and UTC, set this offset to the core and enable the core again. This way the local clock on TAI base makes no jump since the new offset is already taken into account. The only issue with this is that for the time around midnight the clock is free running without a reference.

## 3 Register Set

This is the register set of the DCF Slave 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
Dcf SlaveControl Reg	DCF Slave Enable Control Register	0x00000000	RW
Dcf SlaveStatus Reg	DCF Slave Error Status Register	0x00000004	WC
Dcf SlaveVersion Reg	DCF Slave Version Register	0x0000000C	RO
Dcf SlaveCorrection Reg	DCF Slave Second Corrections Register	0x00000010	RW
Dcf SlaveAirDelay Reg	DCF Slave Air propagation delay Register	0x00000020	RW
Dcf SlaveBitPosition Reg	DCF Slave Bit Position Register	0x00000030	RO

Table 4: Register Set Overview

## 3.2 Register Descriptions

### 3.2.1 General

#### 3.2.1.1 DCF Slave Control Register

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

Dcf SlaveControl 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
																															ENABLE
RO																															RW
Reset: 0x00000000																															
Offset: 0x0000																															

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



### 3.2.1.2 DCF Slave Status Register

Shows the current status of the DCF Slave Clock.

Dcf SlaveStatus 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
																															ERROR
RO																															WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:1	RO
ERROR	DCF Error (sticky)	Bit: 0	WC

### 3.2.1.3 DCF Slave 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.

Dcf SlaveVersion 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
<div>VERSION</div>																															
RO																															
0xFFFFFFFF																															
Offset: 0x000C																															

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

### 3.2.1.4 DCF Slave Correction Register

Correction register to compensate for leap seconds between the different time domains. DCF is UTC time, all other time in the system is TAI, this leads to a correction of 36 seconds by 2016.

Dcf SlaveCorrection 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
COR_SIGN	<div>COR_S</div>																														
RW	RW																														
Reset: 0x00000000																															
Offset: 0x0010																															

Name	Description	Bits	Access
COR_SIGN	Correction sign	Bit: 31	RW
COR_S	Correction in seconds to the time extracted from the DCF => used to convert between TAI, UTC and GPS (leap seconds)	Bit: 30:0	RW

### 3.2.1.5 DCF Slave Air propagation delay Register

This register allows to compensate for the propagation delay of the wave in air between the DCF master and the DCF slave. To calculate the delay a rule of thumb says around 1ns per 15cm.

Dcf SlaveAirDelay Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		CABLE_DELAY																													
RO		RW																													
Reset: 0x00000000																															
Offset: 0x0020																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:30	RO
CABLE_DELAY	Air propagation delay from the Sender in Mainflingen (near Frankfurt, Germany) to the current position (15cm is around 1ns)	Bit: 29:0	RW

### 3.2.1.6 DCF Slave Bit Position Register

Shows the current bit position of the DCF decoder, which bit is received at the moment. When the bit position is at 59 it will wrap-around through 0. When the bit position changes from anything other than 58 to 50 a wrong symbol was detected (or any other error condition)

Dcf SlaveBitPosition 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						
																										BIT_POSITION											
RO																										RO											
Reset: 0x00000000																																					
Offset: 0x0030																																					

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:6	RO
BIT_POSITION	Bit Position which bit is received by the DCF decoder (0-59)	Bit: 5:0	RO

## 4 Design Description

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

### 4.1 Top Level – Dcf Slave

#### 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
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
InputDelay Nanosecond_Gen	natural	1	Input delay of the DCF from the connector to the input signal.
DcfFilter_Gen	boolean	1	If a filter shall be used that required minum 60ms pulses to set a one (recommended)
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\_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 denominator 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 meaning.
Valid	std_logic	1	Whether the Adjustment is valid or not

Table 7: Clk\_TimeAdjustment\_Type

#### 4.1.1.2.3 Dcf\_SlaveStaticConfig\_Type

Defined in Dcf\_SlaveAddrPackage.vhd of library DcfLib

This is the type used for static configuration.

Field Name	Type	Size	Description
Correction	Clk_Time_Type	1	Time to correct the parsed time to correct UTC to TAI or another base.
AirDelay	std_logic_vector	16	Compensation value for the Air propagation delay between master and slave in Nanoseconds: 1ns = 15cm

Table 8: Dcf\_SlaveStaticConfig\_Type

#### 4.1.1.2.4 Dcf\_SlaveStaticConfigVal\_Type

Defined in Dcf\_SlaveAddrPackage.vhd of library DcfLib

This is the type used for valid flags of the static configuration.

Field Name	Type	Size	Description
Enable_Val	std_logic	1	Enables the DCF Slave

Table 9: Dcf\_SlaveStaticConfigVal\_Type

#### 4.1.1.2.5 Dcf\_SlaveStaticStatus\_Type

Defined in Dcf\_SlaveAddrPackage.vhd of library DcfLib

This is the type used for static status supervision.

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

Table 10: Dcf\_SlaveStaticStatus\_Type

#### 4.1.1.2.6 Dcf\_SlaveStaticStatusVal\_Type



Defined in Dcf\_SlaveAddrPackage.vhd of library DcfLib

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

Table 11: Dcf\_SlaveStaticStatusVal\_Type

### 4.1.1.3 Entity Block Diagram

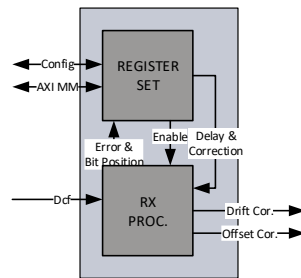


Figure 4: DCF Slave Clock

### 4.1.1.4 Entity Description

#### Rx Processor

This module handles the incoming DCF signal. It decodes the DCF input data stream and converts the signal from a PWM signal to a BCD encoded time and converts it then into UTC in time of day format. In parallel it detects the start of minute condition and generates a PPM. Then it converts the UTC time in time of day format into TAI format in seconds since 1.1.1970 without leap seconds. It also takes a timestamp of the rising edge of the PPS, takes the propagation delay of the cable into account, calculates the offset and the drift against the encoded time and adjusts the local clock.

See 4.2.1 for more details.

#### Registerset

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows configuring the DCF Slave 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

See 4.2.2 for more details.

### 4.1.1.5 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				

StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
ClockClkPeriod Nanosecond_Gen	-	natural	1	Integer Clock Period
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the DCF from the connector to the input signal.
DcfFilter_Gen	-	boolean	1	If a very low pass filter shall be used (60ms min integral)
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
<b>Ports</b>				
<b>System</b>				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
<b>Config</b>				
StaticConfig_DatIn	in	Dcf_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Dcf_Slave StaticConfigVal_Type	1	Static Configuration valid
<b>Status</b>				
StaticStatus_DatOut	out	Dcf_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Dcf_Slave StaticStatusVal_Type	1	Static Status valid
<b>Timer</b>				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
<b>Time Input</b>				

ClockTime_DatIn	in	Clk_Time_Type	1	Adjusted PTP Clock Time
ClockTime_ValIn	in	std_logic	1	Adjusted PTP Clock Time valid
<b>AXI4 Lite Slave</b>				
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
AxiWriteRespResponse_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
AxiReadDataResponse_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response
<b>Pulse Per Second Input</b>				
Dcf_DatIn	in	std_logic	1	DCF input from a DCF Master

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: DCF Slave Clock

## 4.2 Design Parts

The DCF Slave 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 RX Processor

#### 4.2.1.1 Entity Block Diagram

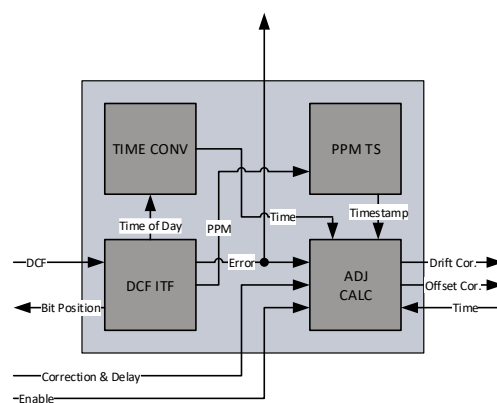


Figure 5: RX Processor

#### 4.2.1.2 Entity Description

##### DCF Interface Adapter

This module decodes the DCF input data stream and converts the signal from a PWM signal to a BCD encoded time and converts it then into UTC in time of day format. In parallel it detects the start of minute condition and generates a PPM.

##### PPM Timestamper

This module takes a timestamp. of the Counter Clock at the detection of the rising edge of the PPM and compensates the input delay. The input PPS is generated at the detection of the bit 0 after the Mark symbol by the DCF Interface Adapter

##### Time Converter

This module converts the time from the Time of Day format: hh:mm:ss dd:mm:yyyy into seconds since midnight 1.1.1970. It loops over the years, months and days taking the leap years into account and finally adds the seconds of the hours, minutes and seconds. After this conversion, a final correction is done to adjust the

second to the next minute. Then this time is passed to the adjustment calculation module.

### Adjustment Calculation

This module calculates the drift and offset of the local clock and corrects it. It takes the Air propagation delay and additional correction into account for the offset correction. After enabling or error detection the calculation waits for two consecutive calculations before adjusting the clock again.

#### 4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
ClockClkPeriod Nanosecond_Gen	-	natural	1	Clock Period in Nanosecond
Sim_Gen	-	boolean	1	If in Testbench simulation mode
RX Processor				
InputDelay Nanosecond_Gen	-	natural	1	Input delay of the DCF from the connector to the input signal.
DcfFilter_Gen	-	boolean	1	If a very low pass filter shall be used (60ms min integral)
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

DCF Error Output				
Dcf_ErrOut	out	std_logic_vector	1	Indicates an error in the decoding
DCF Bit Position Output				
BitPosition_DatOut	out	std_logic_vector	6	Which bit is received by the DCF decoder
Enable Input				
Enable_EnalIn	in	std_logic	1	Enables the correction and supervision
DCF Air propagation delay Input				
DcfAirDelay_DatIn	in	std_logic_vector	30	Propagation delay of the DCF signal from the master source to the connector.
DCF Correction Input				
DcfCorrection_DatIn	in	Clk_Time_Type	1	Additional correction to convert from UTC to a different time format with an offset
DCF Input				
Dcf_DatIn	in	std_logic	1	DCF input from a DCF Master
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
<b>Drift Adjustment Input</b>				
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 13: RX Processor

## 4.2.2 Registerset

### 4.2.2.1 Entity Block Diagram

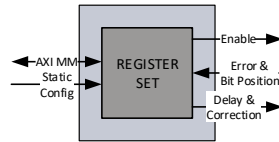


Figure 6: Registerset

### 4.2.2.2 Entity Description

#### Register Set

This module is an AXI4Lite Memory Mapped Slave. It provides access to all registers and allows configuring the DCF Slave 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 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 Air propagation delay value and correction can be changed at runtime. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.

### 4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
Register Set				
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Ports				

System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Dcf_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Dcf_Slave StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Dcf_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Dcf_Slave StaticStatusVal _Type	1	Static Status valid
AXI4 Lite Slave				
AxiWriteAddrValid_ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady_RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress_AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt_DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid_ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady_RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData_DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe_DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid_ValOut	out	std_logic	1	Write Response Valid
AxiWriteRespReady_RdyIn	in	std_logic	1	Write Response Ready
AxiWriteRespResponse_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	in	std_logic_vector	3	Read Address

_DatIn				Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadDataResponse_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response
<b>DCF Error Input</b>				
Dcf_ErrIn	in	std_logic_vector	1	Indicates an error either in the filter or because of missing DCF
<b>DCF Bit Position Input</b>				
BitPosition_DatIn	in	std_logic_vector	6	Which bit is received by the DCF decoder
<b>DCF Correction Output</b>				
DcfCorrection_DatOut	out	Clk_Time_Type	1	Additional correction to the received UTC time
<b>DCF Air propagation delay Output</b>				
DcfAirDelay_DatOut	in	std_logic_vector	30	Propagation delay of the DCF signal from the master source to the connector.
<b>Enable Output</b>				
DcfSlaveEnable_DatOut	out	std_logic	1	Enables the correction and supervision

Table 14: 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 DcfStaticConfigSlave_Con : Dcf_SlaveStaticConfig_Type := (  
  Correction          => (  
    Second            => x"00000024", -- UTC 36 leap seconds  
    Nanosecond        => (others => '0'), -- no nanoseconds  
    Fraction          => (others => '0'), -- no fractions  
    Sign              => '0', -- UTC correct in positive  
    TimeJump          => '0'), -- no  
  AirDelay            => (others => '0')  
);  
  
constant DcfStaticConfigValSlave_Con : Dcf_SlaveStaticConfigVal_Type := (  
  Enable_Val          => '1'  
);
```

Figure 7: Static Configuration

The Air propagation delay can be changed at runtime. It is always valid.

### 4.3.2 AXI Configuration

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

```
-- DCF SLAVE  
-- Config  
-- correction of plus 36 second to convert UTC to TAI  
AXI WRITE 10000010 00000024  
-- Set Air propagation delay to 35 ns  
AXI WRITE 10000020 00000023  
-- enable DCF Slave  
AXI WRITE 10000000 00000001
```

Figure 8: AXI Configuration

In the example the Air propagation delay is first set to 35ns and the Correction is set to plus 36 seconds then the core is enabled.

## 4.4 Clocking and Reset Concept

### 4.4.1 Clocking

To keep the design as robust and simple as possible, the whole DCF Slave 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
<b>System</b>		
System Clock	50MHz (Default)	System clock where the PS runs on as well as the counter clock etc.
<b>DCF Interface</b>		
DCF	1 Hz	No clock, asynchronous data signal.
<b>AXI Interface</b>		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 15: Clocks

### 4.4.2 Reset

In connection with the clocks, there is a reset signal for each clock domain. All resets are active low. All resets can be asynchronously set and shall be synchronously released with the corresponding clock domain. All resets shall be asserted for the first couple (around 8) clock cycles. All resets shall be set simultaneously and released simultaneously to avoid overflow conditions in the core. See the reference designs top file for an example of how the reset shall be handled.

Reset	Polarity	Description
<b>System</b>		
System Reset	Active low	Asynchronous set, synchronous release with the system clock
<b>AXI Interface</b>		

AXI Reset	Active low	Asynchronous set, synchronous release with the AXI clock, which is the same as the system clock
-----------	------------	---

Table 16:     Resets

## 5 Resource Usage

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

### 5.1 Intel/Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static configuration, No Filter)	1298	4517	0	0
Maximal (AXI configuration, With Filter)	1368	4955	0	0

Table 17: Resource Usage Intel/Altera

### 5.2 AMD/Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static configuration, No Filter)	1322	4252	0	0
Maximal (AXI configuration, With Filter)	1390	4430	0	0

Table 18: Resource Usage AMD/Xilinx



## 6 Delivery Structure

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

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

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

DCF                                -- DCF library folder
|-Core                           -- DCF library cores
|-Doc                            -- DCF library cores documentations
|-Library                        -- DCF library component sources
|-Package                        -- DCF library package sources
|-Refdesign                       -- DCF library cores reference designs
|-Testbench                      -- DCF 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 Dcf Slave testbench consist of 3 parse/port types: AXI, CLK and DCF. The DCF TX port takes the CLK port time as reference and generates the DCF stream aligned with the time from the CLK. The DCF RX port takes the time of the Clock instance as reference and the DCF stream from the DCF TX port. Once the clock is synchronized the CLK port and Clock generated time should be phase and frequency aligned. In addition, for configuration and result checks an AXI read and write port is used in the testbench and for accessing more than one AXI slave also an AXI interconnect is required.

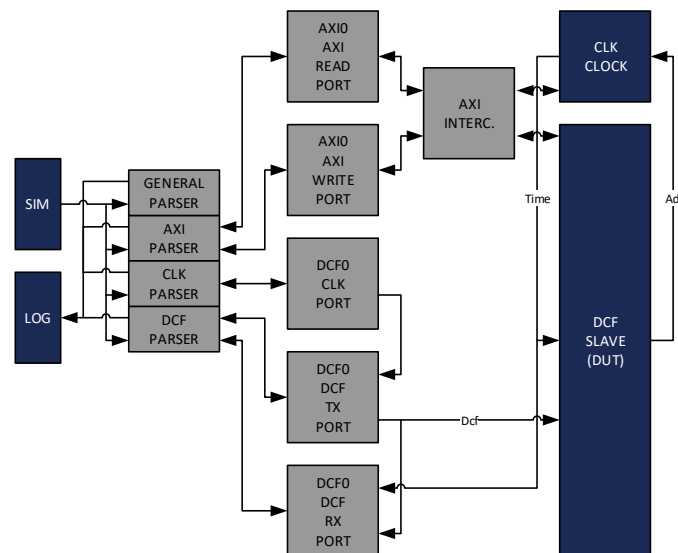


Figure 9: Testbench Framework

For more information on the testbench framework check the Sim\_ReferenceManual documentation.

With the Sim parameter set the time base for timeouts are divided by 1000 to 100000 to speed up simulation time.

### 7.1 Run Testbench

1. Run the general script first

```
source XXX/SIM/Tools/source_with_args.tcl
```

2. Start the testbench with all test cases

```
src XXX/DCF/Testbench/Core/DcfSlave/Script/run_Dcf_Slave_Tb.tcl
```

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

## 8 Reference Designs

The DCF Slave reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the DCF Slave Clock IP core and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately). Optionally it also contains an instance of an 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 be connected to any DCF Master. The Phase and Frequency is corrected via the DCF Slave Clock. 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. Via the dip switches the Air propagation delay can be entered in 25000ns steps (50km).

All generics can be adapted to the specific needs.

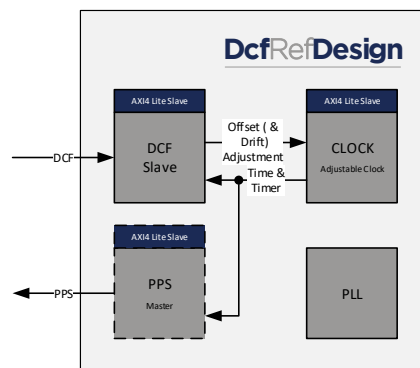


Figure 10: Reference Design

### 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 /DCF/Refdesign/Altera/SocKit/DcfSlave/DcfSlave.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

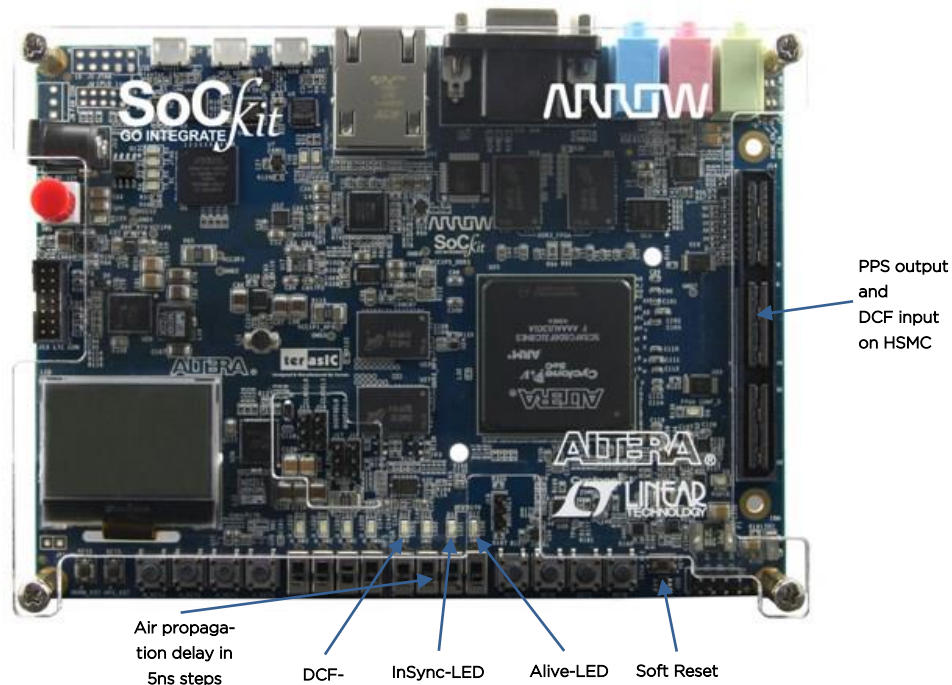


Figure 11: SockKit (source Terasic Inc)

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/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 /DCF/Refdesign/Xilinx/Arty/DcfSlave/DcfSlave.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

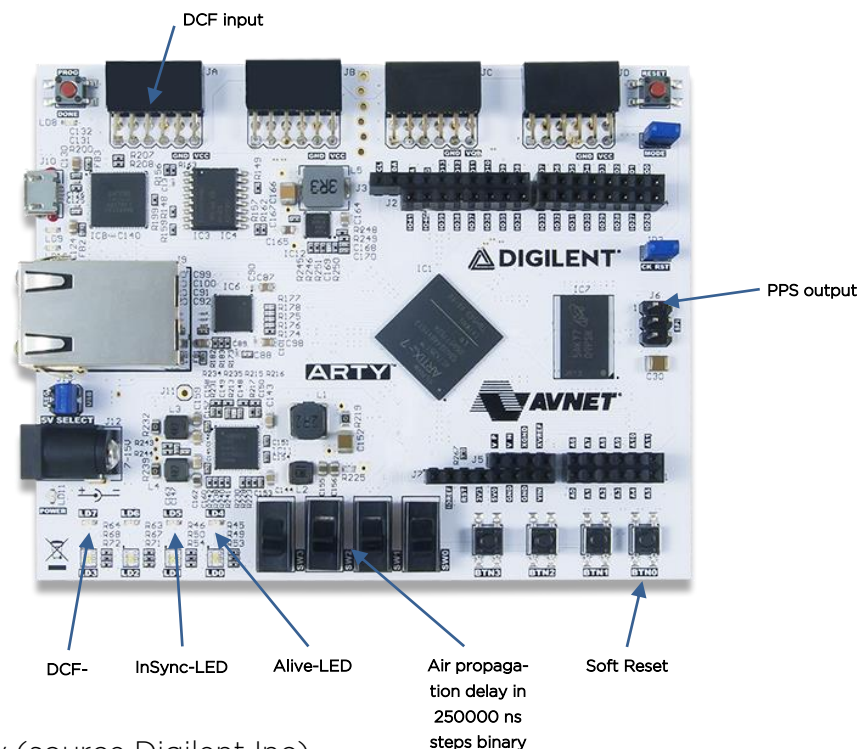


Figure 12: Arty (source Digilent Inc)

## 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

Table 1:	Revision History.....	4
Table 2:	Definitions.....	7
Table 3:	Abbreviations .....	8
Table 4:	Register Set Overview .....	15
Table 5:	Parameters .....	22
Table 6:	Clk_Time_Type .....	23
Table 7:	Clk_TimeAdjustment_Type.....	23
Table 8:	Dcf_SlaveStaticConfig_Type.....	24
Table 9:	Dcf_SlaveStaticConfigVal_Type .....	24
Table 10:	Dcf_SlaveStaticStatus_Type.....	24
Table 11:	Dcf_SlaveStaticStatusVal_Type.....	25
Table 12:	DCF Slave Clock .....	29
Table 13:	RX Processor.....	33
Table 14:	Registerset .....	36
Table 15:	Clocks .....	38
Table 16:	Resets .....	39
Table 17:	Resource Usage Intel/Altera.....	40
Table 18:	Resource Usage AMD/Xilinx .....	40

## B List of figures

Figure 1:	Context Block Diagram .....	9
Figure 2:	Architecture Block Diagram.....	10
Figure 3:	DCF-77 time frame .....	13
Figure 4:	DCF Slave Clock .....	26
Figure 5:	RX Processor.....	30
Figure 6:	Registerset .....	34
Figure 7:	Static Configuration .....	37
Figure 8:	AXI Configuration.....	37
Figure 9:	Testbench Framework .....	42
Figure 10:	Reference Design.....	44
Figure 11:	SockIt (source Terasic Inc).....	45
Figure 12:	Arty (source Digilent Inc) .....	46