IrigSlaveClock

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	1.2
Date	26.06.2018

Copyright Notice

Disclaimer

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

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

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

Overview

NetTimeLogic's IRIG Slave Clock is a full hardware (FPGA) only implementation of a synchronization core able to synchronize to a IRIG-B input.
The whole time frame decoding, algorithms and calculations are implemented in the core, no CPU is required. This allows running IRIG synchronization completely independent and standalone from the user application. The core can be configured either by signals or by an AXI4Light-Slave Register interface.

Key Features:

- IRIG Slave Clock
- Supports IRIG-B006 format (compatible with B004, B005, B006 and B007 IRIG-B Masters) with PWM encoded DCLS
- Optionally supports IRIG-B126 format (compatible with B124, B125, B126 and B127 IRIG-B Masters) with AM encoded AC (requires an ADC)
- IRIG decoding and time format conversion
- IRIG supervision
- Optional support Control Bits for IRIG-Bxx0/Bxx1/Bxx4/Bxx5
- Input delay compensation
- Cable delay compensation
- Additional seconds correction to convert between UTC and TAI time (or any other time base)
- Synchronization accuracy: +/- 25ns
- AXI4 Light 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	20.10.2016	First draft
1.0	28.10.2016	First release
1.1	20.12.2017	Status interface added
1.2	26.06.2018	Added Control Bits

Table 1: Revision History

Content

Definitions

Definitions	
IRIG Slave Clock	A clock that can synchronize itself to an IRIG 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
DCLS	DC Level Shift
IRQ	Interrupt, Signaling to e.g. a CPU
IRIG	Inter Range Instrumentation Group Timecode
IS	IRIG Slave
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 IRIG Slave Clock is meant as a co-processor handling an IRIG input.
It takes an IRIG 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 IRIG master and corrects it. It also extracts the Control Bits for processing by the user.
The IRIG Slave Clock is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Light slave for configuration and status supervision from a CPU, this is however not required since the IRIG Slave Clock can also be configured statically via signals/constants directly from the FPGA.

Optionally the IRIG Slave Clock can also handle AC/AM encoded IRIG-B with an external ADC and an AC/AM to DCLS converter. Contact us for details

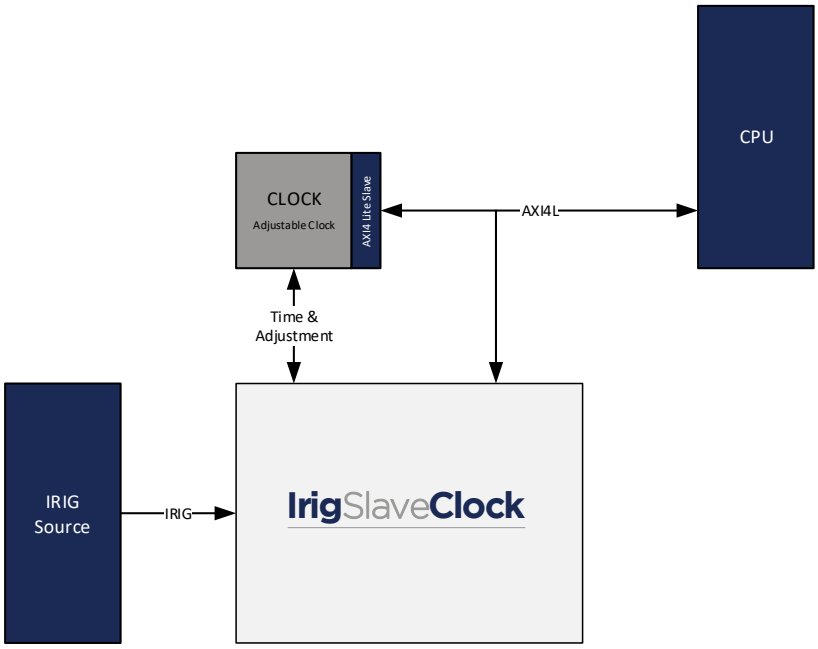

Figure 1: Context Block Diagram

1.2 Function

The IRIG Slave Clock takes an IRIG signal, decodes the IRIG input data stream and converts the signal from a DCLS encoded 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. 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. After 2 consecutive correct PPS timestamps are taken and 2 consecutive IRIG time frames were decoded, the calculation of offset and drift is started. For offset correction an additional cable 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 IRIG time frames were correctly decoded again after the error flag was deasserted.

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.

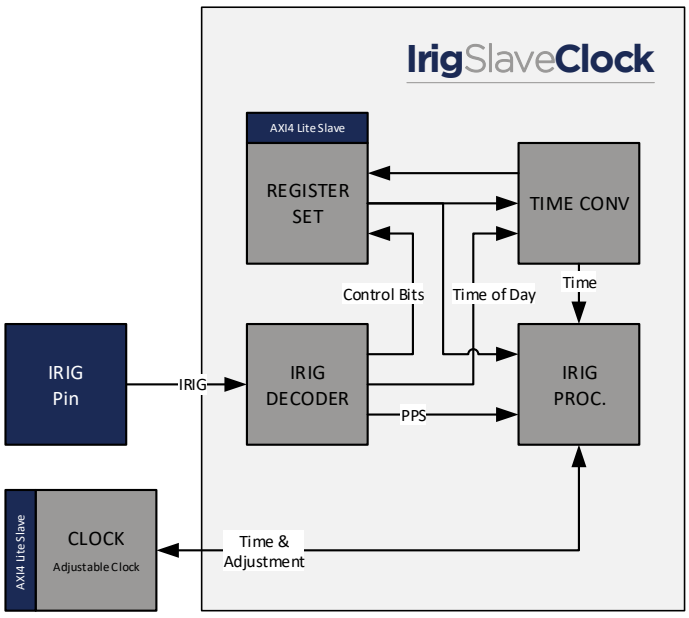

Figure 2: Architecture Block Diagram

Register Set

This block allows reading status values and writing configuration.

IRIG Decoder

This block decodes the IRIG input data stream and converts the signal from a DCLS encoded 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 and extracts the Control Bits.

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

IRIG Processor

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

2 IRIG Basics

2.1 Interface

IRIG is a very simple interface and can be either DC Level shift or Amplitude modulated on a base frequency and has Pulse Width modulated symbols. 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 the reference mark; this shall be at the second 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 can be repeated from multiple times a second to once every multiple second, depending on the IRIG code used. Also the number of bits in a time frame varies from 60 to 100 bits and the time frame content is depending on the IRIG code. This core supports the IRIG-B B006/B126 code which generates a time frame of 100 bits once every second.

An IRIG-B time frame in B007/B127 format contains the following:

- Second (BCD encoded)
- Minute (BCD encoded)
- Hour (BCD encoded)
- Day of Year (BCD encoded)
- Year (BCD encoded)
- Seconds of Day (Straight Binary encoded) - unused

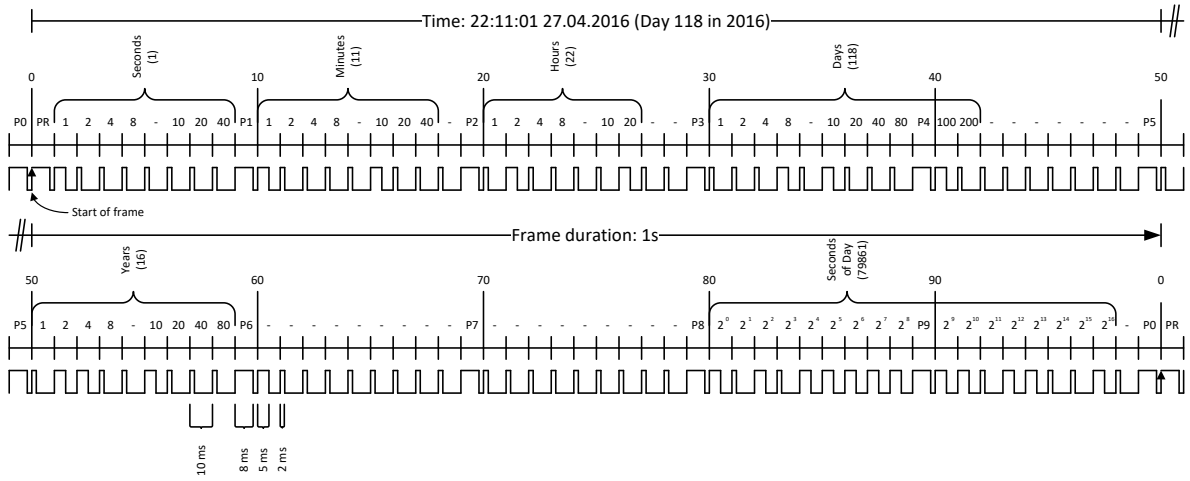

Figure 3: IRIG-B B007/B127 time frame

The B007/B127 format is in principle compatible with B004/B124, B005/B125, B006/B126 and B007/B127 IRIG-B Slaves which only check the time fields.

An IRIG-B time frame in B004/B124 format contains the following:

- Second (BCD encoded)
- Minute (BCD encoded)
- Hour (BCD encoded)
- Day of Year (BCD encoded)
- Year (BCD encoded) (Control Bits 0:8)
- Control Bits 9:17
- Control Bits 18:26
- Seconds of Day (Straight Binary encoded) - unused

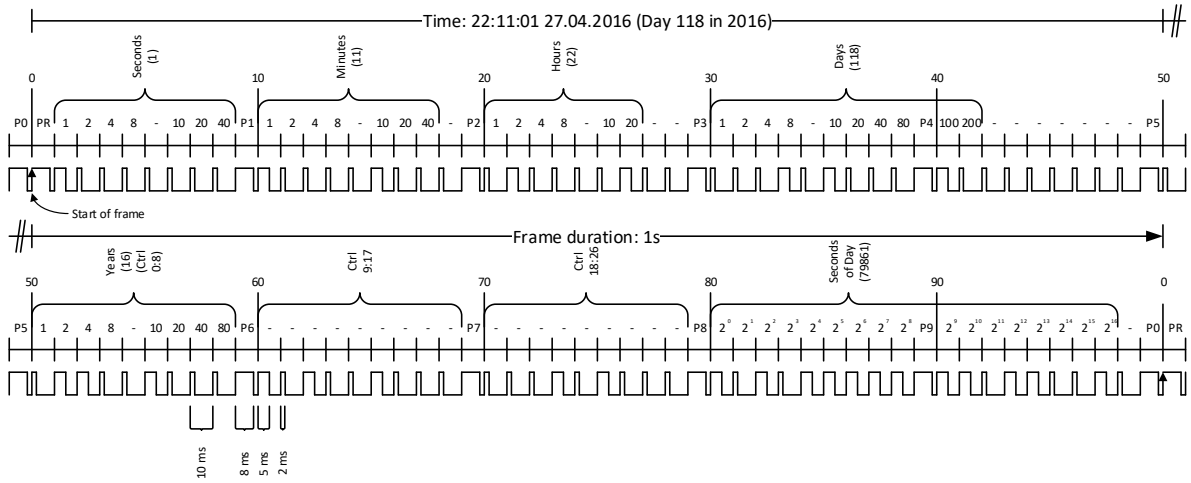

Figure 4: IRIG-B B004/B124 time frame

So, in addition to IRIG-B007/B127 Control Bits are transferred which can be used to send user data or IEEE1344 or C37.118 Information to the Slave.

For high accuracy synchronization delays have to be compensated for.
An IRIG network is normally a one-to-many configuration: one IRIG master synchronizes multiple IRIG slaves of different distance from the master.

2.2 Delays

There are two kinds of delays in an IRIG Network. One is the input delay of the IRIG signal from the connector 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 cable length and medium: 15cm of copper cable are equal to roughly 1ns of propagation delay. This delay can be set in the core, e.g. for compensation of the cable length to a time server on the building's roof.

2.3 UTC vs TAI time bases

IRIG 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 IRIG Slave 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
Irig SlaveControl Reg	IRIG Slave Enable Control Register	0x00000000	RW
Irig SlaveStatus Reg	IRIG Slave Error Status Register	0x00000004	WC
Irig SlaveVersion Reg	IRIG Slave Version Register	0x0000000C	RO
Irig SlaveCorrection Reg	IRIG Slave Second Corrections Register	0x00000010	RW
Irig SlaveCableDelay Reg	IRIG Slave Cable Delay Register	0x00000020	RW
Irig SlaveControlBits Reg	IRIG Slave Control Bits Register	0x00000014	RO

Table 4: Register Set Overview

3.2 Register Descriptions

3.2.1 General

3.2.1.1 IRIG Slave Control Register

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

IRIG 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 IRIG Slave Status Register

Shows the current status of the IRIG Slave Clock.

Irig 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	IRIG Error (sticky)	Bit: 0	WC

3.2.1.3 IRIG 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.

Irig 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
VERSION																															
RO																															
0xXXXXXXXX																															
Offset: 0x000C																															

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

3.2.1.4 IRIG Slave Correction Register

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

Irig 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	COR_S																														
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 IRIG => used to convert between TAI, UTC and GPS (leap seconds)	Bit: 30:0	RW

3.2.1.5 IRIG Slave Cable Delay Register

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

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

Name	Description	Bits	Access
-	Reserved, read 0	Bit: 31:16	RO
CABLE_DELAY	Cable delay of cable to master in nanoseconds (15cm is around 1ns)	Bit: 15:0	RW

3.2.1.6 IRIG Slave Control Bits Register

Received Control bits. Representing Bit50-Bit58 & Bit60-Bit68 & Bit70-Bit78 of the IRIG frame: Bit50 => Bit0, Bit78 => Bit26 of the register. When IRIG-B004/B005 is used only Bit60-Bit68 & Bit70-Bit78 are representing control bits in the register, Bit50-Bit58 are representing the year information.

Irig SlaveControlBits 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
					CONTROL_BITS																										
RO					RO																										
0x00000000																													
Offset: 0x0014																													

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:27	RO
CONTROL_BITS	Received Control bits for IRIG-B004/B005	Bit: 26:0	RO

4 Design Description

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

4.1 Top Level – Irig 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 IRIG from the connector to the input signal.
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 denumerator of the rate in nanoseconds (TimeAdjust-ment every Interval = drift rate), for offset correction this is the period in which the time shall be correct-ed(TimeAdjustment in Inter-val), for setting the time this has no mining.
Valid	std_logic	1	Whether the Adjustment is valid or not

Table 7: Clk_TimeAdjustment_Type

4.1.1.2.3 Irig_SlaveStaticConfig_Type

Defined in Irig_SlaveAddrPackage.vhd of library IrigLib
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.
CableDelay	std_logic_vector	16	Compensation value for the cable delay between master and slave in Nanoseconds: 1ns = 15cm

Table 8: Irig_SlaveStaticConfig_Type

4.1.1.2.4 Irig_SlaveStaticConfigVal_Type

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

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

Table 9: Irig_SlaveStaticConfigVal_Type

4.1.1.2.5 Irig_SlaveStaticStatus_Type

Defined in Irig_SlaveAddrPackage.vhd of library IrigLib
This is the type used for static status supervision.

Field Name	Type	Size	Description
CoreInfo	Clk_CoreInfo_ Type	1	Info about the Cores state
ControlBits	std_logic_vector	27	Received Control Bits

Table 10: Irig_SlaveStaticStatus_Type

4.1.1.2.6 Irig_SlaveStaticStatusVal_Type

Defined in Irig_SlaveAddrPackage.vhd of library IrigLib

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: Irig_SlaveStaticStatusVal_Type

4.1.1.3 Entity Block Diagram

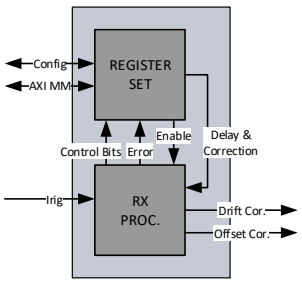

Figure 5: IRIG Slave Clock

4.1.1.4 Entity Description

Rx Processor

This module handles the incoming IRIG signal. It decodes the IRIG input data stream and converts the signal from a DCLS encoded 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. 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 AXI Light Memory Mapped Slave. It provides access to all registers and allows configuring the IRIG 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 IRIG from the connector to the input signal.
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	Irig_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Irig_Slave StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Irig_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Irig_Slave 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 Response
Pulse Per Second Input				
Irig_DatIn	in	std_logic	1	IRIG input from an IRIG 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: IRIG Slave Clock

4.2 Design Parts

The IRIG 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

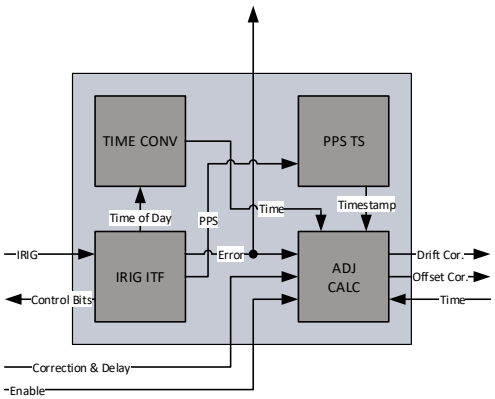

Figure 6: RX Processor

4.2.1.2 Entity Description

IRIG Interface Adapter

This module decodes the IRIG input data stream and converts the signal from a DCLS encoded 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. It also extracts the Control Bits (including the Year part)

PPS Timestamper

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

Time Converter

This module converts the time from the Time of Day format: hh:mm:ss ddd:yyyy into seconds since midnight 1.1.1970. It loops over the years 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 second. 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 cable 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 IRIG 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
IRIG Error Output				
Irig_ErrOut	out	std_logic_vector	1	Indicates an error in the decoding

Enable Input				
Enable_EnaIn	in	std_logic	1	Enables the correction and supervision
IRIG Cable Delay Input				
IrigCableDelay_DatIn	in	std_logic_vector	16	Propagation delay of the IRIG signal from the master source to the connector.
IRIG Correction Input				
IrigCorrection_DatIn	in	Clk_Time_Type	1	Additional correction to convert from UTC to a different time format with an offset
IRIG Control Bits Output				
IrigControlBits_DatOut	out	std_logic_vector	27	Control Bits received for IRIG-B000/B001/B004/B005
IRIG Input				
Irig_DatIn	in	std_logic	1	IRIG input from an IRIG 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
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 13: RX Processor

4.2.2 Registerset

4.2.2.1 Entity Block Diagram

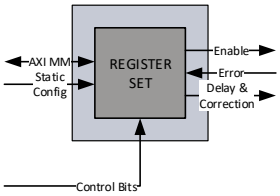

Figure 7: Registerset

4.2.2.2 Entity Description

Register Set

This module is an AXI Light Memory Mapped Slave. It provides access to all registers and allows configuring the IRIG Slave 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 cable 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
AxiAddressRangeLow_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRangeHigh_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset

				Size
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Irig_Slave StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Irig_Slave StaticConfigVal _Type	1	Static Configuration valid
Status				
StaticStatus_DatOut	out	Irig_Slave StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Irig_Slave 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 Response
IRIG Correction Output				
IrigCorrection_DatOut	out	Clk_Time_Type	1	Additional correction to the received UTC time
IRIG Error Input				
Irig_ErrIn	in	std_logic_vector	1	Indicates an error either in the filter or because of missing IRIG
IRIG Cable Delay Output				
IrigCable Delay_DatOut	in	std_logic_vector	16	Propagation delay of the IRIG signal from the master source to the connector.
IRIG Control Bits Input				
IrigControlBits_DatIn	in	std_logic_vector	27	Control Bits received for IRIG-B000/B001/B004/B005
Enable Output				
IrigSlave Enable_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 IrigStaticConfigSlave_Con : Irig_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
  CableDelay           => (others => '0')
);

constant IrigStaticConfigValSlave_Con : Irig_SlaveStaticConfigVal_Type := (
  Enable_Val                  => '1'
);
```

Figure 8: Static Configuration

The cable 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 IRIG Slave Clock is 0x10000000.

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

Figure 9: AXI Configuration

In the example the Cable 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 IRIG 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
System		
System Clock	50MHz (Default)	System clock where the PS runs on as well as the counter clock etc.
IRIG Interface		
IRIG	100 Hz	No clock, asynchronous data signal.
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 15: 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 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 Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static configuration)	1356	5315	0	0
Maximal (AXI configuration)	1356	5315	0	0

Table 17: Resource Usage Altera

5.2 Xilinx (Artix 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (Static configuration)	1377	5005	0	0
Maximal (AXI configuration)	1419	5080	0	0

Table 18: 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

IRIG                                -- IRIG library folder
|-Core                              -- IRIG library cores
 |-Doc                              -- IRIG library cores documentations
 |-Library                          -- IRIG library component sources
 |-Package                          -- IRIG library package sources
 |-Refdesign                        -- IRIG library cores reference designs
 |-Testbench                        -- IRIG 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 Irig Slave testbench consist of 3 parse/port types: AXI, CLK and IRIG.
The IRIG TX port takes the CLK port time as reference and generates the IRIG stream aligned with the time from the CLK. The IRIG RX port takes the time of the Clock instance as reference and the IRIG stream from the IRIG 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.

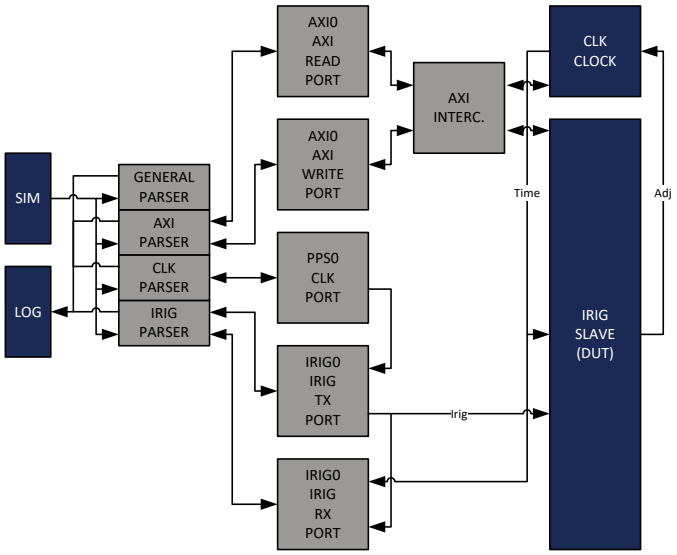

Figure 10: Testbench Framework

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

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

7.1 Run Testbench

1. Run the general script first

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

2. Start the testbench with all test cases

```
src XXX/IRIG/Testbench/Core/IrigSlave/Script/run_Irig_Slave_Tb.tcl
```

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

8 Reference Designs

The IRIG Slave reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the IRIG 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 IRIG Master. The Phase and Frequency is corrected via the IRIG 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 cable delay can be entered in 5ns steps.
All generics can be adapted to the specific needs.

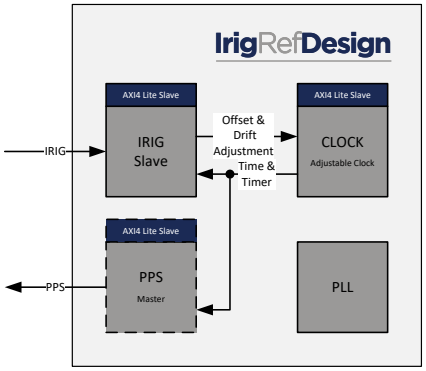

Figure 11: Reference Design

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 /IRIG/Refdesign/Altera/SocKit/IrigSlave/IrigSlave.qpf
3. If the optional core PPS Master Clock is available add the files from the corresponding folders (PPS/Core, PPS/Library and PPS/Package)
4. Change the generics (PpsMasterAvailable_Gen) in Quartus (in the settings menu, not in VHDL) to true for the optional cores that are available.

5. Rerun implementation
6. Download to FPGA via JTAG

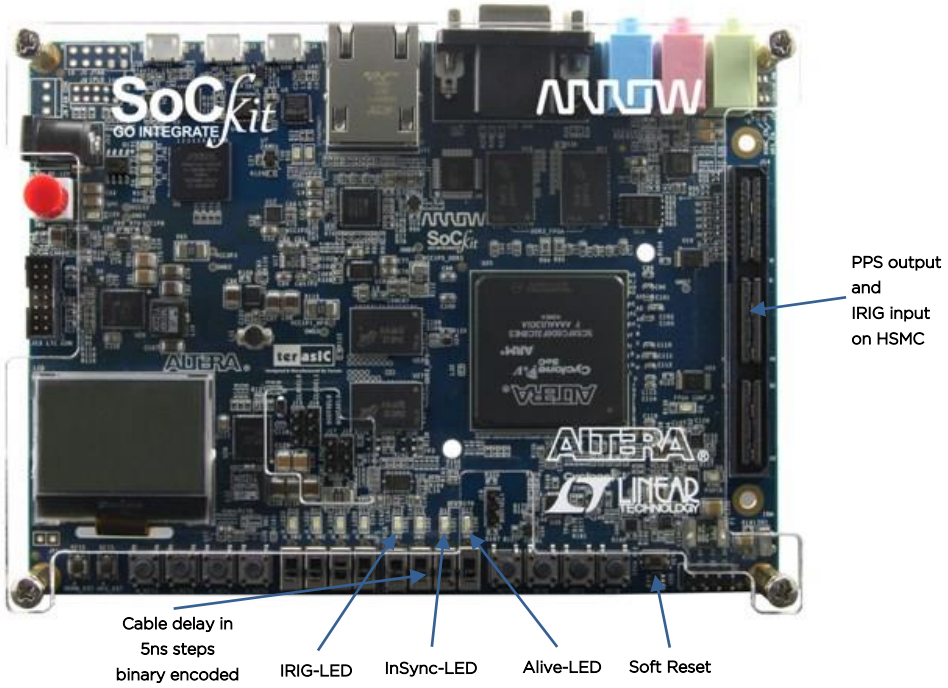

Figure 12: SocKit (source Terasic Inc)

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

8.2 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 /IRIG/Refdesign/Xilinx/Arty/IrigSlave/IrigSlave.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

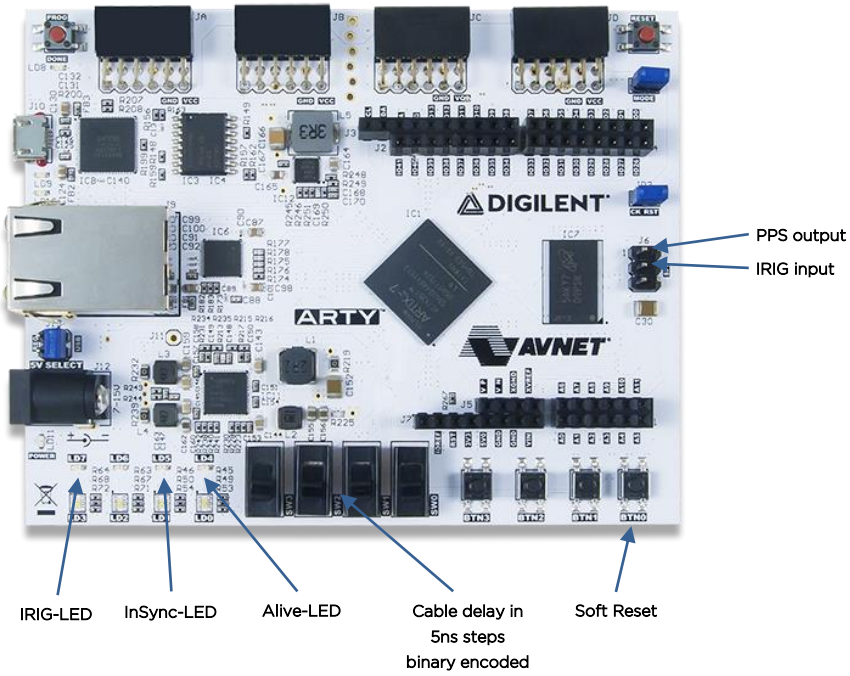

Figure 13: Arty (source Digilent Inc)

A List of tables

B List of figures