Nutaq® INNOVATION TODAY FOR TOMORROW®

Timestamp

Programmer's Reference Guide

February 2016

Revision history

Revision	Date	Comments
1.0	March 2015	Updated for release 6.6.1
1.1	October 2015	Added Host Applications Using the Timestamp Library Updated for release 7.0
1.2	February 2016	Updated for PicoSDR official release

To ensure the accuracy of the information contained herein, particular attention was given to usage in preparing this document. It corresponds to the product version manufactured prior to the date appearing on the title page. There may be differences between the document and the product, if the product was modified after the production of the document.

Nutaq reserves itself the right to make changes and improvements to the product described in this document at any time and without notice.

Version 1.2

Trademarks

WARNING

Do not use Nutaq products in conjunction with life-monitoring or life-critical equipment. Failure to observe this warning relieves Nutaq of any and all responsibility.

FCC WARNING

This equipment is intended for use in a controlled environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of personal computers and peripherals pursuant to subpart J of part 15 of the FCC rules. These rules are designed to provide reasonable protection against radio frequency interference. Operating this equipment in other environments may cause interference with radio communications, in which case the user must, at his/her expense, take whatever measures are required to correct this interference.

Table of Contents

List of Figures and Tables

1 Introduction

This document contains all the information necessary to understand and use the Timestamp module with your Perseus product. It should be read carefully before using the module and stored in a handy location for future reference.

1.1 Conventions

In a procedure containing several steps, the operations that the user has to execute are numbered (1, 2, 3…). The diamond (♦) is used to indicate a procedure containing only one step, or secondary steps. Lowercase letters (a, b, c…) can also be used to indicate secondary steps in a complex procedure.

The abbreviation NC is used to indicate no connection.

Capitals are used to identify any term marked as is on an instrument, such as the names of connectors, buttons, indicator lights, etc. Capitals are also used to identify key names of the computer keyboard.

All terms used in software, such as the names of menus, commands, dialog boxes, text boxes, and options, are presented in bold font style.

The abbreviation N/A is used to indicate something that is not applicable or not available at the time of press.

Note:
The screen captures in this document are taken from the software version available at the time of press. For this reason, they may differ slightly from what appears on your screen, depending on the software version that you are using. Furthermore, the screen captures may differ from what appears on your screen if you use different appearance settings.

1.2 Glossary

This section presents a list of terms used throughout this document and their definition.

Term	Definition
Application programming interface (API)	An application programming interface is the interface that a computer system, library, or application provides to allow requests for services to be made of it by other computer programs or to allow data to be exchanged between them.
Board software development kit	Abbreviated BSDK, this kit gives users the possibility to quickly become fully functional developing C/C++ for the host computer and HDL code for the FPGA through an understanding of all Nutaq boards major interfaces.
Chassis	Refers to the rigid framework onto which the CPU board, Nutaq development platforms, and other equipment are mounted. It also supports the shell-like case—the housing that protects all the vital internal equipment from dust, moisture, and tampering.
Default design	Design loaded by default on Nutaq boards used for FPGA design.
Example	Refers to examples used to demonstrate functions or applications supplied with the board software development kit. For this reason, examples come in two flavors: application examples and functional examples.
HDL	Stands for hardware description language.
Host	A host is defined as the device that configures and controls a Nutaq board. The host may be a standard computer or an embedded CPU board in the same chassis system where the Nutaq board is installed. You can develop applications on the host for Nutaq boards through the use of an application programming interface (API) that comprises protocols and functions necessary to build software applications. These API are supplied with the Nutaq board.
Model-based design	Refers to all the Nutaq board-specific tools and software used for development with the boards in MATLAB and Simulink and the Nutaq model-based design kits.
NTP	Network Time Protocol. NTP is a protocol to synchronize the computer time over a network.
Peer	A host peer is an associated host running RTDEx on either Linux or Windows. An FPGA peer is an associated FPGA device.
PPS	Pulse per second. Event to indicate the start of a new second.
Reception	Any data received by the referent is a reception. Abbreviated RX.
Reference design	Blueprint of an FPGA system implanted on Nutaq boards. It is intended for others to copy and contains the essential elements of a working system (in other words, it is capable of data processing), but third parties may enhance or modify the design as necessary.
Software development	Refers to development performed with and for the board with a software development kit. Software development for a board comes in three flavors: host software development and FPGA software development.
Transmission	Any data transmitted by the referent is a transmission. Abbreviated TX.
VHDL	Stands for VHSIC hardware description language.

Table 1 Glossary

1.3 Technical Support

Nutaq is firmly committed to providing the highest level of customer service and product support. If you experience any difficulties using our products or if it fails to operate as described, first refer to the documentation accompanying the product. If you find yourself still in need of assistance, visit the technical support page in the Support section of our Web site at www.nutaq.com.

2 Product Description

Nutaq's Timestamp core provides the user design with a free running counter that is incremented at every design clock cycle. The user logic can monitor the counter value to see the time elapsed between two events as a number of clock ticks.

The counter value can be initialized with a value provided by the user. This initial value is applied to the counter when the next rising-edge is detected at the input PPS port of the Timestamp core. If the precise moment the PPS rising-edge event occurs is known, all future reading of the counter value can be expressed in an absolute time format.

The counter is 64-bit wide. It would take more than 2000 years to wrap if the design clock is running at a frequency of 250 MHz. If the initial counter value is set relatively low, the counter value should not wrap around. For example, setting the counter value based on the number of clock periods elapsed since January 1st 1970 should not cause any problem.

3 Timestamp FPGA Cores Description

3.1 Timestamp FPGA Core Overview

The Timestamp core has been developed to provide the user logic with a free-running counter. The initial value of the counter is easily configurable by the host computer and is applied when the next rising-edge event is detected at the input PPS port of the core.

The Timestamp core does not have its own dedicated AXI decoder logic but instead uses the platform register core AXI decoder to instantiate configurable core registers.

The Figure 1 shows how the Timestamp module is configurable through the Platform Register AXI interface. It also shows that the counter is initialized and increments based on the PPS and design clock from the user logic. The Timestamp module provides the time to the user logic through a 64-bit user port.

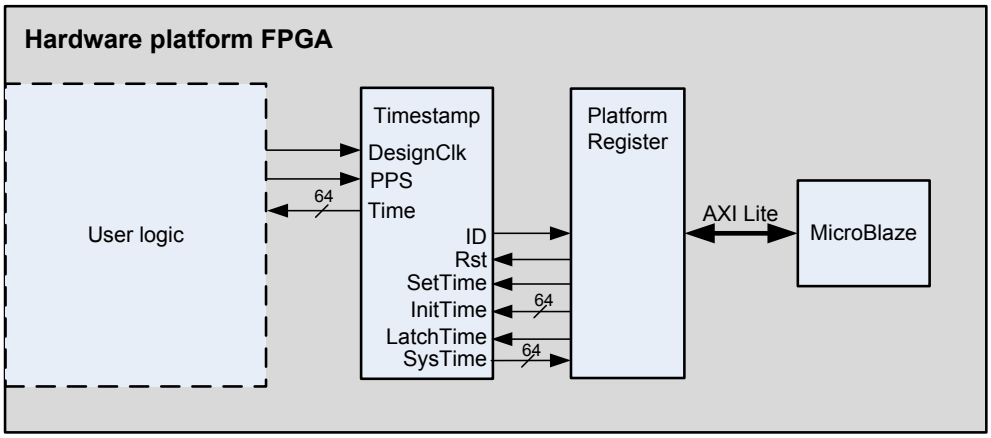

Figure 1 RTDExSync interaction with RTDEx

3.2 Cores Ports

Port	Direction	Description
Configuration and status ports		
i_SysClk_p	IN STD_LOGIC	System clock. This clock must be the same clock used by the input configuration port signals of this core. This clock is usually the AXI Lite clock used by the Platform Register core.
ov32_CoreIdVers_p	OUT STD_LOGIC_VECTOR(31 downto 0)	Timestamp core ID and version. The Timestamp core ID is 0xCC03. *This port must be connected to the related input port of the platform register core.*

Port	Direction	Description
i_CoreReset_p	IN STD_LOGIC	Active high core reset. Resets all the internal registers and channel states. After a reset, the counter will start at 0 and will increment its value at every i_UserClk_p clock cycle. *This port must be connected to the related input port of the platform register core.*
i_SetTime_p	IN STD_LOGIC	On the rising-edge of this signal, arm the counter to set the iv64_TimeStamp_p on the next i_Pps_p rising-edge. After the rising-edge of this signal, the counter value will be hold at 0 until the next PPS rising-edge event happens. *This port must be connected to the related input port of the platform register core.*
i_LatchTime_p	IN STD_LOGIC	On the rising-edge of this signal, the current timestamp value is latch and output to ov64_SysTimeStamp_p port. This is done since the timestamp counter is not on the same clock domain than the system clock. *This port must be connected to the related input port of the platform register core.*
iv64_TimeStamp_p	IN STD_LOGIC_VECTOR(63 downto 0)	Value to initialize the timestamp counter.
ov64_SysTimeStamp_p	OUT STD_LOGIC_VECTOR(63 downto 0)	Current counter value latched at i_LatchTime_p rising-edge.
User ports		
i_UserClk_p	OUT STD_LOGIC	User clock. This clock is used to increment the counter value at every clock cycle and is used to detect the i_Pps_p rising-edge event.
i_Pps_p	IN STD_LOGIC	PPS signal. Once a i_SetTime_p rising-edge has been performed, the initial time will be applied to the counter value at the rising-edge of this port. All subsequent PPS signal will be ignored until the next i_SetTime_p rising-edge.
ov64_UserTimeStamp_p	OUT STD_LOGIC_VECTOR(63 downto 0)	The current counter value synchronous to the i_UserClk_p clock. It increments its value at every i_UserClk_p clock cycle after a reset or when the time is set and a PPS rising-edge event has been detected.

Table 2 Timestamp core ports

3.3 Core Registers

3.3.1 Registers Map

The core registers are accessible through the platform's register core ports. The base address of these registers depends on the Platform Register AXI Lite base address and on the offset address of the Timestamp register inside the Platform Register core. The Timestamp base address can be found in the user's guide of the platform you are using.

Register name	Address	Direction	Description
TIMESTAMP_COREID	0x0	R	Timestamp core ID and version
TIMESTAMP_CTRL	0x4	R/W	Timestamp control register
TIMESTAMP_MSW	0x8	R/W	Timestamp value MSW

Register name	Address	Direction	Description
TIMESTAMP_LSW	0xC	R	Timestamp value LSW

Table 3 Timestamp registers

3.3.2 Registers Description

Offset 0x0 — TIMESTAMP_COREID

This register contains the Timestamp core identification and version number.

Bit	31 to 16	15 to 0
Name	CoreID	Version
Direction	R	R
Reset value	0xCC03	0x0100

Table 4 TIMESTAMP_COREID register

Name	Description	Configuration
CoreID	This is the Timestamp core unique ID number.	0xCC03
Version	This is the Timestamp core version number.	0x0100 = Version 1.0

Table 5 TIMESTAMP_COREID register fields description

Offset 0x4 — TIMESTAMP_CTRL

This register controls the inner logic of the Timestamp core.

Bit	31 to 29	2	1	0
Name	Reserved		SetTime	CoreReset
Direction	R	P	p	p
Reset value	0	0	0	0

Table 6 TIMESTAMP_CTRL register

Name	Description	Configuration
CoreReset	Reset all the internal registers of the FPGA core to their default value. After a reset, the counter will start at 0 and will increment its value at every user clock cycle.	1 = Reset core registers Self-clearing bit.
SetTime	Set the specified time on the next PPS rising-edge. After the time is set, the counter value will be hold at 0 until the next PPS rising-edge event happens.	1 = Set the initial time and arm the PPS rising-edge detection process Self-clearing bit.
LatchTime	Latch the current timestamp value. This is done before reading the timestamp register value since the timestamp counter is not on the same clock domain as the system clock.	1 = Latch the timestamp value Self-clearing bit.

Table 7 TIMESTAMP_CTRL register fields description

Offset 0x8 — TIMESTAMP_MSW

This register gives access to the most significant word (upper 32 bits) of the timestamp value.

When writing this register, it changes the timestamp value that will be used when the initial time has to be set.

When reading this register, it gives access to the current time value latched.

Bit	31 to 0
Name	Timestamp_MSW
Direction	R/W
Reset value	0

Table 8 TIMESTAMP_MSW register

Offset 0xC — TIMESTAMP_LSW

This register gives access to the least significant word (lower 32 bits) of the timestamp value.

When writing this register, it changes the timestamp value that will be used when the initial time has to be set.

When reading this register, it gives access to the current time value latched.

Bit	31 to 0
Name	Timestamp_LSW
Direction	R/W
Reset value	0

Table 9 TIMESTAMP_LSW register

4 Timestamp Host Programming

This chapter provides the functions of the Timestamp library as well as the procedure to build a host application.

4.1 Timestamp Library

Function	Description
adp_result_t Timestamp_Open(Timestamp_Handle_t *phTimestamp, connection_state state, uint32_t u32ClkFreq);	Open a Timestamp instance. A pointer to a timestamp handle and a valid connection with the platform must be provided. **u32ClkFreq** is the frequency of the clock connected to the timestamp module in Hz.
void Timestamp_Close(Timestamp_Handle_t hTimestamp);	Close the Timestamp handle.
adp_result_t Timestamp_Reset(Timestamp_Handle_t hTimestamp);	Reset the counter value to 0 and set the counter in free-running mode
adp_result_t Timestamp_GetClkFreq(Timestamp_Handle_t hTimestamp, uint32_t * pu32ClkFreq);	Get the current clock frequency of the timestamp.
adp_result_t Timestamp_ApplyHostTimeToDevice(Timestamp_Handle_t hTimestamp);	The host time is sent to the platform to set the counter value at the next PPS rising-edge event. It sends the value when the host time is not near a second transition (±100 ms) in order to be sure to set the time corresponding to the right PPS event. If the host time and PPS signal are minimally synchronized, this should assure that the absolute time will be correctly set inside the platform at the next PPS event.
adp_result_t Timestamp_VerifyTime(Timestamp_Handle_t hTimestamp);	Verify that the platform time matches the host computer time. If a PPS event has not occurred since the time setting inside the Open function, after a timeout of 1.5 second, an error will be returned indicating that the PPS signal is absent. If the time read from the platform does not fit the host time within ±100 milliseconds, an out of sync error will be returned. The Timestamp library does not make an extra effort to synchronize the platform time if the host time is not within ±100 milliseconds of the PPS time. It is the user responsibility to make sure the host time and the PPS signal that goes to the platform are synchronized. To do so, use a GPS receiver to generate the PPS event and synchronize the host computer time to an NTP server.
adp_result_t Timestamp_SetDeviceTickTime(Timestamp_Handle_t hTimestamp, uint64_t u64Tick);	Set the current device time in clock tick (number of clock cycles). This value will be applied when the next rising-edge is detected at the PPS input port.
adp_result_t Timestamp_SetDevicePosixTime(Timestamp_Handle_t hTimestamp, uint32_t u32sec, uint32_t u32nsec);	Set the current device time in POSIX format (UTC POSIX time and offset in nanosecond from the UTC POSIX time). This value will be applied when the next rising-edge is detected at the PPS input port.

Function	Description
adp_result_t Timestamp_SetDeviceReadableLocalTime(Timestamp_Handle_t hTimestamp, char * pcTime);	Set the current device time readable local time format. This value will be applied when the next rising-edge is detected at the PPS input port. Ex.: 2014-11-20 10:22:06.123456789 where the last 9 digit after the '.' represent an offset in nanosecond
adp_result_t Timestamp_GetHostPosixTime(uint32_t * pu32sec, uint32_t * pu32nsec);	Get the current host time. This function should not be used to have precise time but to have a good idea of the host time.
adp_result_t Timestamp_GetHostReadableLocalTime(char * pcTime, uint32_t u32Length);	Get the current host time. This function should not be used to have precise nanoseconds time but to display in a readable format the computer local time. The **u32Length** argument is the **pcTime** maximal length. **pcTime** length should at least be 30 bytes.
adp_result_t Timestamp_GetDeviceTickTime(Timestamp_Handle_t hTimestamp, uint64_t * pu64Tick);	Get the platform time in clock tick. This function should not be used to have precise time but to have a good idea of the platform time. The communication latency is not compensated and several milliseconds can easily be required in order for the time to be available in the application.
adp_result_t Timestamp_GetDevicePosixTime(Timestamp_Handle_t hTimestamp, uint32_t * pu32sec, uint32_t * pu32nsec);	Get the platform time. This function should not be used to have precise time but to have a good idea of the platform time. The communication latency is not compensated and several milliseconds can easily be required in order for the time to be available in the application.
adp_result_t Timestamp_GetDeviceReadableLocalTime(Timestamp_Handle_t hTimestamp, char * pcTime, uint32_t u32Length);	Get the platform time in a readable format and in the host computer local time format. This function should not be used to have precise time but to have a good idea of the platform time. The communication latency is not compensated and several milliseconds can easily be required in order for the time to be available in the application. The **u32Length** argument is the **pcTime** maximal length. **pcTime** length should at least be 30 bytes.
adp_result_t Timestamp_PosixToReadableLocalTime(uint32_t u32sec, uint32_t u32nsec, char *pcString, uint32_t u32Length);	Convert a POSIX time with nanosecond offset in a readable format and in the host computer local time format. The **u32Length** argument is the **pcTime** maximal length. **pcTime** length should at least be 30 bytes.

adp_result_t Timestamp_ReadableLocalTimeToPosix(
char *pcString,
uint32_t * pu32sec,
uint32_t * pu32nsec);
	Convert time expressed in a readable format and in the host computer local time format to POSIX time with nanosecond offset.
	The readable local time should have the following syntax:
	`"2014-11-20 10:22:06.123456789"`
	where the last 9 digit after the '.' represent an offset in nanosecond
	Careful, `"2014-11-20 10:22:06.5"` means 10:22:06 +5 ns (not +500 ms). To express +500 ms make sure remaining zeros are present to express the value in nanosecond:
	`"2014-11-20 10:22:06.500000000"`

adp_result_t Timestamp_PosixTimeToTick(Timestamp_Handle_t hTimestamp, uint32_t u32sec, uint32_t u32nsec, uint64_t * pu64Tick);	Convert a POSIX time + offset in nanosecond in number of clock ticks.
adp_result_t Timestamp_TickToPosixTime(Timestamp_Handle_t hTimestamp, uint64_t u64Tick, uint32_t * pu32sec, uint32_t * pu32nsec);	Convert Timestamp module clock tick to a POSIX time + offset in nanosecond.

Table 10 Timestamp library functions

4.2 Building a Host Application Using Timestamp

The following diagram illustrates the typical flow of a PPSYNC application.

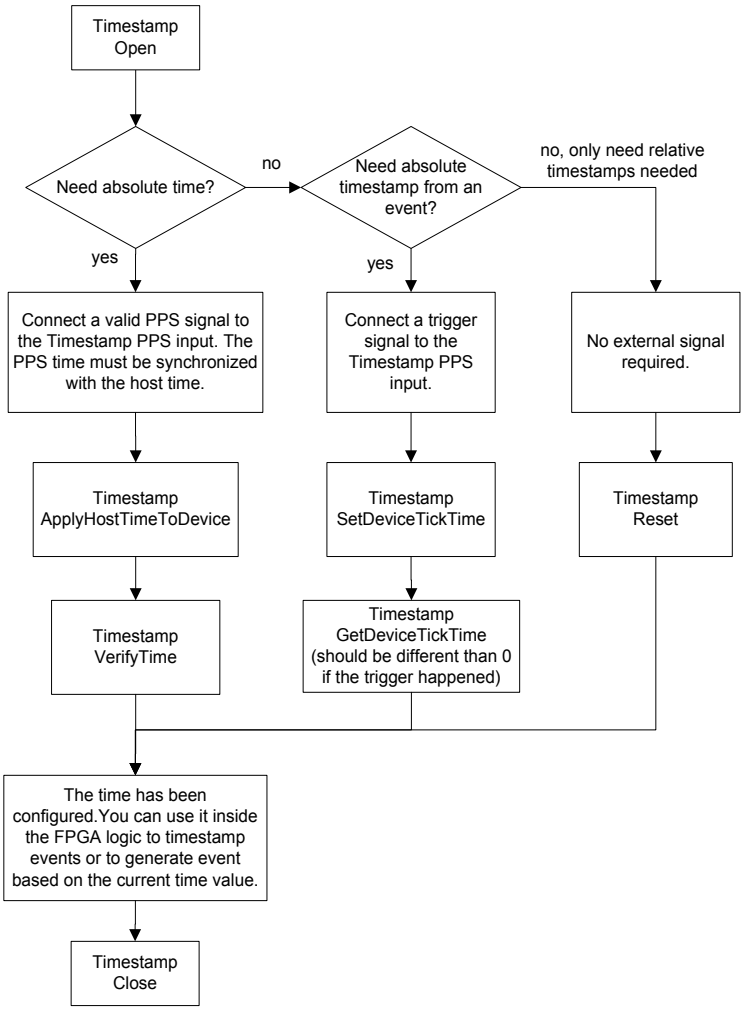

Figure 2 Timestamp application creation flow

The first step taken by a programmer building a Timestamp application should be to open a Timestamp instance. For this step to succeed, the Timestamp core must be present inside the FPGA design.

Resetting the timestamp module is all that is required if only relative timestamps are needed inside the FPGA design. This will restart the Timestamp internal counter to 0 and then increment the counter at every clock cycle. By reading the value of the current time when events occur, the time difference between them can easily be retrieved.

If absolute time from a specific event is needed inside the FPGA design, the Timestamp module needs to be configured with the “SetDeviceTime” functions. These functions set the specified time in the Timestamp counter on the next detection of a rising-edge at the PPS input port. Once the initial time has been set, the timestamp value can be retrieved and converted to the desired format by using the conversion functions, “TickToPosixTime” and “PosixToReadableLocalTime”.

If absolute time timestamps are needed inside the FPGA design, the Timestamp module needs to be configured with the “ApplyHostTimeToDevice” function. This function sets the host time in the Timestamp

counter on the next detection of a rising-edge at the PPS input port. The counter time is expressed as the number of clock cycles elapsed since January 1st 1970. This function requires that a valid PPS signal is connected to the platform. The PPS time needs to be synchronized with the host computer time within ±100 milliseconds. Once the initial time has been set, the timestamp value can be retrieved and converted to the desired format by using the conversion functions, “TickToPosixTime” and “PosixToReadableLocalTime”.

5 Host Applications Using the Timestamp Library

The BAS software suite provides an example utility application, named *TimestampUtil*, that the user can modify. This application performs configurations, readings, or other actions on the Timestamp FPGA core.

This application comes precompiled in the %BASROOT%\tools\bin directory. The project folder is also made available in the %BASROOT%\tools\apps\core\TimestampUtil directory.

Examples showcase how this application is used and are available in your BAS software suite.

- One of them features the ADAC250 FMC. It is available in the %BASROOT%\examples\adac250\host\scripts directory under the name ADAC250_Record. Please refer to document **ADAC250 Examples Guide.pdf** in folder %BASROOT%\doc\fmc\ADAC250 for details on how to run this example.
- The other one features the Radio420 FMC. It is available in the %BASROOT%\examples\radio420\host\scripts\rtdex_recplay directory, under the name Radio420_Record. Please refer to document **Radio420 - Rtdex RecordPlayback Examples Guide.pdf** in folder %BASROOT%\doc\fmc\Radio420 for details on how to run this example.

5.1 TimestampUtil description

Usage: *TimestampUtil <ip address> <configuration file name>*

- Parameter 1 must be the carrier IP address, ex 192.168.0.101
- Parameter 2 must be the configuration file name, ex Timestamp_Init.ini

This application can perform different actions based on parameters specified in the configuration file. The principal function of application TimestampUtil is `TimestampUtil()`. The function is basically a sequence of `if, else` statements surrounding tasks selected with parameter `action` in the configuration file. Please see section 5.2 for a detailed description of how to configure the operation of TimestampUtil using configuration files.

5.2 TimestampUtil configuration file

Instead of getting its parameters through command line arguments, application *TimestampUtil* uses a separate configuration file to gather them. The path to this file is passed as an argument to the application. Details on how Nutaq's configuration files are built are available in the application note "*App Note - Parsing and Creating Configuration Files*" residing in the %BASROOT%*doc**app_notes* repository.

To get a feel of what the *TimestampUtil* configuration file looks like, the reader is encouraged to open file *Timestamp_Init.ini* in %BASROOT%*examples**radio420**host**scripts* with a text editor. It is provided with the Radio420 applicative example. The file is similar to this example:

```
#Timestamp module configuration
[Timestamp]
type=timestamp
clock_frequency=30720000
```

```
action=3

# Contextual parameters
initial_value=0
trigger_timeout=2000
wait_until_time=2015-06-01_16:30:00.0,0
nanosec_offset=0
lsw_custom_register_id=3
msw_custom_register_id=4
```

Configuration files for using TimestampUtil must contain all TimestampUtil parameters.

- `type` (an instance of that parameter must be equal to timestamp)
- `clock_frequency`. This tells the application the frequency of the clock used by the Timestamp FPGA core. This information is needed by the application if it does clock ticks to actual time conversions.
- `action`. This is the actual task the TimestampUtil application will perform.
    - `0`: The application retrieves the time stored in the Timestamp FPGA core, in number of clock ticks, and prints it.
    - 1: The application retrieves the time stored in the Timestamp FPGA core, in number of clock ticks. Then, it converts it to POSIX time and prints it.
    - 2: The application retrieves the time stored in the Timestamp FPGA core, in number of clock ticks. Then, it converts it to human readable date and time and prints it.
    - 3: The application resets the Timestamp core. It will then start counting clock ticks in free running mode.
    - 4: The application configures the timestamp module with the value specified with parameter `initial_value`. The value will be latched at the next trigger event and the counter will start couting in free running mode from that point on.
      This action uses parameters
        - `initial_value`
        - `trigger_timeout`
    - 5: Configure the timestamp module with the absolute time based on the host time and on the PPS signal. A valid PPS signal must be connected to the board and the PPS signal must be synchronized with the host time within ±100 ms.
    - 6: Only wait until time specified with parameter `wait_until_time` and return. This application only checks the host time, which is usually accurate to 50 ms compared with the GPS time. This action may be useful when batching other applications and tasks in a script file.
      This action uses parameters
        - `wait_until_time`
        - `nanosec_offset`
    - 7: Retrieve time from custom registers `lsw_custom_register_id` and `msw_custom_register_id`, in ticks, and print it. Time value fetched from this action is

only meaningful if the Timestamp FPGA core was configured with absolute time.
This action uses parameters

- `lsw_custom_register_id`
- `msw_custom_register_id`

- 8: Retrieve time from custom registers `lsw_custom_register_id` and `msw_custom_register_id`, convert it to POSIX time, and print it. Time value fetched from this action is only meaningful if the Timestamp FPGA core was configured with absolute time.
  This action uses parameters
  - `lsw_custom_register_id`
  - `msw_custom_register_id`
- 9: Retrieve time from custom registers `lsw_custom_register_id` and `msw_custom_register_id`, convert it to readable local time, and print it. Time value fetched from this action is only meaningful if the Timestamp FPGA core was configured with absolute time.
  This action uses parameters
  - `lsw_custom_register_id`
  - `msw_custom_register_id`

5.3 Exploring the Application Source Code and Modifying/Rebuilding it

A project folder, allowing the user to modify the source code of the application are also available. It resides in *%BASROOT%\tools\apps\core*.

If rebuilding is needed, follow the steps described in the section appropriate to the operating system you are using.

5.3.1.1 Windows

A folder named *prj_win* which contains the Visual Studio 2012 solution associated with the application is present in the folders listed above.

1. Start Microsoft Visual Studio.
2. On the **File** menu, point to **Open** and click **Project/Solution**.
3. Browse to the *%BASROOT%\tools\apps\core\TimestampUtil\prj_win* folder and select the *TimestampUtil.sln* solution
4. Select the build configuration Release x64.
5. On the **Build** menu, click **Build Solution**.

5.3.1.2 Linux

A folder named *prj_linux* which contains a shell script and a makefile associated with the application is present in the folder listed above. Source files are available in the *inc* and *src* folders

1. Open a terminal and use the *cd* command to browse to the *%BASROOT%\tools\apps\core\TimestampUtil\prj_linux* repository in the installation.
2. To build the application, run the following command in the Linux terminal.

   sudo ./build_demo.sh

6 Functional Examples

Functional examples for the Timestamp have been implemented for Nutaq's Perseus AMC carrier. This chapter presents the examples and offers links to the example documentation.

6.1 BSDK Examples

A few BSDK examples showcase the Timestamp module on Nutaq's Perseus AMC carrier platform using Xilinx's Platform Studio, and Microsoft Visual Studio. The Timestamp module is used in the Radio420 and ADAC250 record examples. It is also used indirectly in the RTDExSync example. In this example, it is used through the RTDExTs library which is built on top of the Timestamp library.

The procedures for the Radio420 and ADAC250 record timestamp examples are presented in the following documents

- *%BASROOT%\doc\fmc\Radio420\Radio420 - Rtdex RecordPlayback Examples Guide.pdf*
- *%BASROOT%\\doc\fmc\ADAC250\ADAC250 - Examples Guide.pdf*