Nutaq® INNOVATION TODAY FOR TOMORROW®

PPS Sync

Programmer's Reference Guide

February 2016

Revision history

Revision	Date	Comments
0.1	January 2014	First draft.
0.2	February 2014	Add User logic clock interface block diagram. Add related functions from the API as references to Building a host application section
1.0	February 2014	Layout Up to date for Software Tools Release 6.6
1.1	October 2015	New glossary
1.2	October 2015	Up to date for Software Tools Release 7.0 Added PPSClockDisciplining application description
1.3	February 2016	Up to date 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 1.3

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

Congratulations on the purchase of PPSSync module for Nutaq's Perseus AMC.

This document contains all the information necessary to understand and use the PPSSync module of your product. It should be read carefully before using the card 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.
Base design	Empty design or template that is incapable of data processing and is not instantiated in the custom logic of the board FPGA.
Board software development kit	Abbreviated BSDK, this kit gives users the possibility to quickly become fully functional developing C/C++ or assembly code for the DSP 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.
cPCI	Short for CompactPCI, refers to a 3U or 6U Eurocard-based industrial computer where the all boards are connected through a passive PCI backplane.
Default design	Design loaded by default on Nutaq boards used for FPGA design.
Digital signal processing	Digital signal processing is the study of signals in a digital representation and the processing methods of these signals. The algorithms required for DSP are sometimes performed using specialized devices that use specialized microprocessors called digital signal processors (DSP).
Digital signal processor (DSP)	A digital signal processor is a specialized microprocessor designed specifically for digital signal processing, generally in real time.
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 the CPU board of the cPCI 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.
Peer	A host peer is an associated host running RTDEx on either Linux or Windows. An FPGA peer is an associated FPGA device.
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, DSP 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

The PPSSync core uses to discipline the on-board voltage-controlled crystal oscillator (VCXO) in Radio 420x and ADAC250 FMC cards with aid of pulse per second (PPS) signal coming from an external Global Positioning System (GPS) module. The PPSSync core is designed to work with the Nutaq's Perseus AMC carrier.

2.1 Outstanding Features

Fast lock time

With a precise and stable GPS PPS signal, the PPSYNC core is able to lock the VCXO output to a PPS signal within 100 part-per-billion (ppb) in less than 60 seconds from cold start.

Individual resets

The core can be fully reset independently from the register-mapped interface.

3 PPSSync FPGA Core Description

The PPSSync core is accessible with register-mapped interface on the user's side. Input PPS signal is filtered against unexpected glitches.

3.1 Core Ports

Port	Direction	Description
PPSSYNC_CLOCK		
i_SysClk_p	IN STD_LOGIC	System (Microblaze) clock
i_FmcClk_p	IN STD_LOGIC	Target clock under GPS disciplined
I_Pps_p	IN STD_LOGIC	PPS signal input
PPSSYNC_STATUS		
ov32_CoreIdVers_p	OUT STD_LOGIC_VECTOR(31 downto 0)	PPSSync core ID and version
ov32_PPsCnt_p	OUT STD_LOGIC_VECTOR(31 downto 0)	Number of PPS count
ov32_DiffAcc_p	OUT STD_LOGIC_VECTOR(31 downto 0)	Accumulate ticks error between actual GPS disciplined clock frequency and target clock frequency
PPSSYNC_CONTROL		
i_CoreReset_p	IN STD_LOGIC	Global reset
i_CoreEnable_p	IN STD_LOGIC	Global enable
i_RstUponRead_p	IN STD_LOGIC	Enable automatically clear accumulate ticks error register after read
i_DiffAccRead_p	IN STD_LOGIC	AXI bus read acknowledge uses for automatic clearing accumulate ticks error register after reading
i_RstAcc_p	IN STD_LOGIC	Clear accumulate error register
i_MovingSumEna_p	IN STD_LOGIC	Enable accumulate error integration over a specific amount of time (in second)
i_MovingSumRst_p	IN STD_LOGIC	Clear accumulate error integration register
iv32_ProgDelay_p	IN STD_LOGIC_VECTOR(31 downto 0)	Programmable delay register (in second)
iv32_FmcClkVal_p	IN STD_LOGIC_VECTOR(31 downto 0)	Target clock frequency value under GPS disciplined (in Hertz)
PPSSYNC_STATUS		
ov32_CoreIdVers_p	OUT STD_LOGIC_VECTOR(31 downto 0)	PPSSync core ID and version
ov32_PPsCnt_p	OUT STD_LOGIC_VECTOR(31 downto 0)	Number of PPS count
ov32_DiffAcc_p	OUT STD_LOGIC_VECTOR(31 downto 0)	Accumulate error between actual GPS disciplined clock frequency and target clock frequency (in tick)

3.2 Core Registers

Register name	Address Offset	Direction	Description
PPS_SYNC_COREID	0×00	R	Core ID and version identifier
PPS_SYNC_FMCCLK	0x04	R/W	FMC clock frequency value
PPS_SYNC_CTRL	0x08	R/W	Core control register
PPS_SYNC_PPSCNT	0×0C	R/W	PPS count register
PPS_SYNC_DIFFACC	0×10	R/W	Accumulator error register
PPS_SYNC_PROGDELAY	0x14	R/W	Programmable delay register

Table 2 PPSYNC register map

3.2.1 Register Definitions

Offset 0x00 — PPS_SYNC_COREID

This register indicates the PPSYNC core ID and version identifier.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:16	R	C	0xCC01	PPSSYNC_CoreID	ID of the PPSSync Core
15:0	R	C	0x0200	PPSSYNC _VersionID	Version of the PPSSync Core

Table 3 PPS_SYNC_COREID register

Offset 0x04 — PPS_SYNC_FMCCLK

This register indicates the target FMC clock frequency value.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:0	RW	I	0x0000	PPSSYNC_FmcClkVal	Target FMC clock frequency value

Table 4 PPS_SYNC_FMCCLK register

Offset 0x08 — PPS_SYNC_CTRL

This register gives acces to the controls of the PPSSync core.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:6	R	C	0	rsvd	Reserved
5	RW	I	0	ResetProgDelay	Clear content of the programmable delay logic
4	RW	I	0	MovSumEnable	Enable/disable accumulate error integration logic
3	RW	I	0	CoreEnable	Enable/disable control bit to start/stop the clock discipline. The Enable bit should be set only after PPS is detected to avoid accumulating wrong value in the accumulator register
2	RW	I	0	ResetAccReg	Force clear the accumulator register
1	RW	I	0	ResetUponRead	Select if the accumulator register will be auto-clearing upon read.
0	RW	I	0	CoreReset	Global resets core logic

Table 5 PPS_SYNC_FMCCLK register

Offset 0x0C — PPS_SYNC_PPSCNT

This register holds PPS count value.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:0	R	O	0x0000	PPSSYNC_PpsCnt	PPS count value

Table 6 PPS_SYNC_PPSCNT register

Offset 0x10 — PPS_SYNC_DIFFACC

This register holds accumulate ticks error.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:0	R	O	0x0000	PPSSYNC_DiffAcc	Accumulate ticks error

Table 7 PPS_SYNC_DIFFACC register

Offset 0x14 — PPS_SYNC_PROGDELAY

This register holds number of delay sample.

Bits	R,W,RW	I,O,C	Initial Value	Name	Description
31:0	RW	I	0x0000	PPSSYNC_ProgDelay	Amount of delay in the programmable delay logic

Table 8 PPS_SYNC_PROGDELAY register

3.3 Interfacing the PPSSync Core to the User Logic

The PPSSync core can be easily inserted in an XPS project. Figure 1 shows how the User logic clock interfaces with FMC clock driven by the VCXO within the FMC card. The FMC card is either FMC Radio 420 or FMC ADAC250. In this figure, the SPI controller core is a part of FMC Radio 420 or FMC ADAC250 logic, user application uses PPSSync EAPI to control the PPSSync core via “axi_perseus6010_regs” interface (one can refer to the Perseus601x User’s Guide for using this core). The disciplined FMC clock is used to drive the User logic.

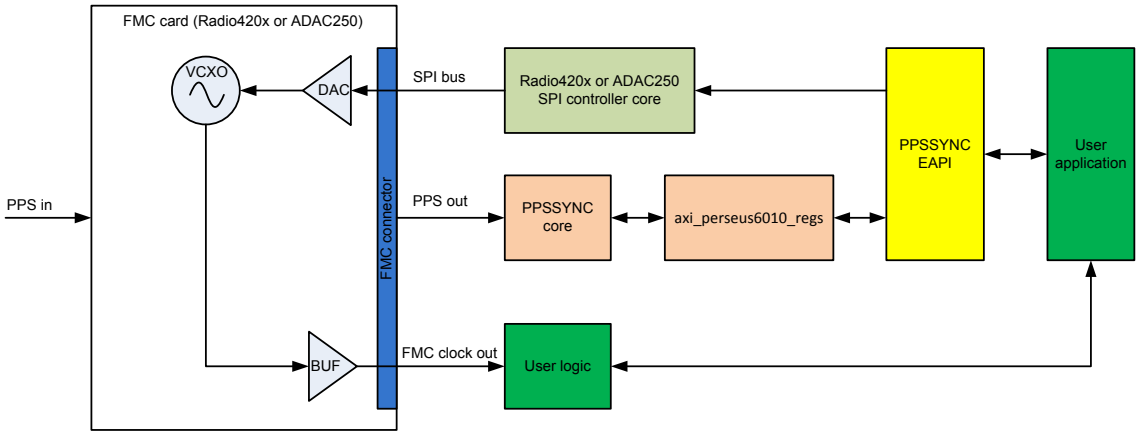

Figure 1 Interfacing the PPSSync core with user logic

4 Low-Level and Host Programming

This chapter provides the functions of the PPSSync driver, the EAPI library, the CLI commands as well as the procedure to build a host application.

4.1 PPSSync Driver

The table below presents the functions and use of the PPSSync driver.

Function	Use
PPSSync_Open	Instanciates PPS struct and resets core
PPSSync_Presence	Verify PPS Sync logic presence and returns core ID and version
PPSSync_Reset	Reset PPS-SYNC core
PPSSync_DacInit	Initialize DAC
PPSSync_DacLimite	Limit reference DAC value
PPSSync_Start	Start the sync, first verifies PPS presence, then starts algorithm
PPSSync_GetInfo	Return all information
PPSSync_Stop	Stop the sync
PPSSync_Close	Close the core
PPSSync_SaveRefDACVal	Write the ref. DAC value to file system
PPSSync_ReadRefDACVal	Read a ref. DAC value to file system
PPSSync_SavePIProfile	Store proportional and integral gains to file system
PPSSync_ReadPIProfile	Read proportional and integral gains stored in file system

Table 9 PPSSync driver functions

4.2 PPSSync EAPI Library

The table below presents the functions and use of the PPSSync EAPI library.

Function	Use
PPSSync_Presense_send	Verify PPS Sync logic presence and returns core ID and version
PPSSync_DacInit_send	Initialize DAC with a value
PPSSync_DacLimiter_send	Limit reference DAC value
PPSSync_Start_send	Start the sync, first verifies PPS presence, then starts algorithm
PPSSync_GetInfo_send	Return all info
PPSSync_Stop_send	Stop the sync
PPSSync_SaveRefDACVal_send	Write the ref. DAC value to file system
PPSSync_ReadRefDACVal_send	Read a ref. DAC value to file system
PPSSync_SavePIProfile_send	Store proportional and integral gains to file system

PPSSync_ReadPIProfile_send	Read proportional and integral gains stored in file system
PPSSync_WaitVCXOLocked_send	Wait for VCXO locked and returning lock status

Table 10 PPSSync EAPI library functions

4.3 Command Line Interface (CLI) Commands

The table below presents the functions and use of the command line interface.

Function	Use
ppssync_presence	Verify PPS Sync logic presence and returns core ID and version
ppssync_ref_dac_init	Initialize DAC with a value
ppssync_ref_dac_limit	Limit reference DAC value
ppssync_start	Start the sync, first verifies PPS presence, then starts algorithm
ppssync_get_info	Return all information
ppssync_stop	Stop the sync
ppssync_save_ref_dac	Write the ref. DAC value to file system
ppssync_read_ref_dac	Read a ref. DAC value to file system
ppssync_save_pi_profile	Store proportional and integral gains to file system
ppssync_read_pi_profile	Read proportional and integral gains stored in file system
ppssync_wait_vcxo_locked	Wait for VCXO locked and returning lock status

Table 11 PPSSync CLI functions

4.4 Building a Host Application Using the PPSSync

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

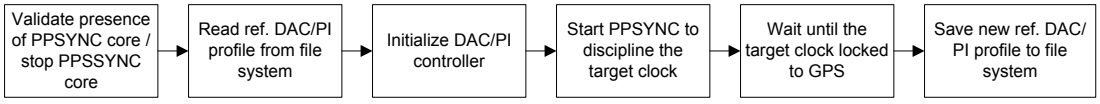

Figure 2 PPSSync application creation flow

The first step taken by a programmer building a PPSSync application should be to detect the PPSSync core (PPSSync_Presense_send). If the PPSSync core has already run, user must stop the core (PPSSync_Stop_send) This step ensures a known state for all the components of the PPSSync at startup.

Once this is done, read reference DAC value (PPSSync_ReadRefDACVal_send) and proportional-integral controller (PI) profile (PPSSync_ReadPIProfile_send) previously stored in the file system and initialize the reference DAC and PI controller (PPSSync_DacInit_send). If the reference DAC value and the proportional-integral controller (PI) profile were not previously stored in file system, the programmer must manually provide initial values for the DAC and PI controller.

Once the initialization is done, start the target clock disciplining by issuing a start command (PPSSync_Start_send) to tell PPSSync to discipline the clock.

The programmer must wait until the target clock locks to the GPS signal (lock status returns by “PPSSync_GetInfo_send” function) and then store the new reference DAC value (PPSSync_SaveRefDACVal_send) and PI profile (PPSSync_SavePIProfile_send) to the file system. The clock disciplining is still running in background unless the programmer issues a stop request (PPSSync_Stop_send).

5 Host Applications Using the PPS Sync Library

The BAS software suite provides an example utility application, named *PPSClockDisciplining*, that the user can modify. This application sends commands to the CCE which performs the actual clock disciplining using a PPS signal.

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

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

These examples require the Radio420 FMC to be run and they reside in the *%BASROOT%\examples\radio420\host\scripts\rtdex_recplay* directory under the names Radio420_DDS.bat and Radio420_Passthrough.bat.

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 PPSClockDisciplining description

Usage: *PPSClockDisciplining <ip address> <target frequency>*

- Parameter 1 must be the carrier IP address, ex 192.168.0.101
- Parameter 2 must be the target frequency of the design clock. The disciplining algorithm will attempt to lock the design clock to this frequency.

This application enables clock disciplining using the PPSSync FPGA core and algorithms running on the embedded processor.

1. the application connects to the carrier CCE using `connect_cce()`, which gets the application a `connection_state` object. This object will be used for all other EAPI function calls.
2. It calls function `PPSClockDisciplining()` which does the actual work of sending the commands necessary for clock disciplining.
    - It verifies the PPSSync FPGA core presence using function `PPSSync_Presence_send()`.
    - It attempts to load a previously saved VCXO DAC value from the FMC EEPROM using function `PPSSync_ReadRefDacVal_send()`.
    - Using function `PPSSync_DacInit_send()`. It initializes the DAC using previously loaded value or the default value if none was found.
    - It attempts to load P and I gains from the FMC EEPROM using function `PPSSync_ReadPIProfile_send()`.
    - It then calls function `PPSSync_Start_send()` which requests the embedded processor to perform the clock disciplining.
    - The function then loops until the clock is locked to the target frequency.
    - It then finally saves the DAC value using function `PPSSync_SaveRefDacVal_send()` and the PI profile using function `PPSSync_SavePIProfile_send()`.

3. It disconnects from the CCE using function `disconnect_cce()`.

After application *PPSClockDisciplining* returns, the embedded processor keeps disciplining the clock by continuously running the disciplining algorithm.

5.2 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.

5.2.1 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\PPSClockDisciplining\prj_win* folder and select the *PPSClockDisciplining.sln* solution
4. Select the build configuration Release x64.
5. On the **Build** menu, click **Build Solution**.

5.2.2 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/PPSClockDisciplining/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 PPSSync 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

The PPSSync BSDK examples showcase the PPSSync design on the Nutaq's Perseus AMC carrier using Xilinx's Platform Studio, Microsoft's Visual Studio, and Nutaq's Command Line Interface.

The procedures for the PPSSync examples are presented in the following documents

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

6.2 MBDK Examples

The PPSSync MBDK examples showcase the PPSSync design on the Nutaq's Perseus AMC carrier.

The procedures for the PPSSync examples are presented in the following documents

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