AN3411
Application note

IEEE 1588 precision time protocol demonstration for STM32F107 connectivity line microcontroller

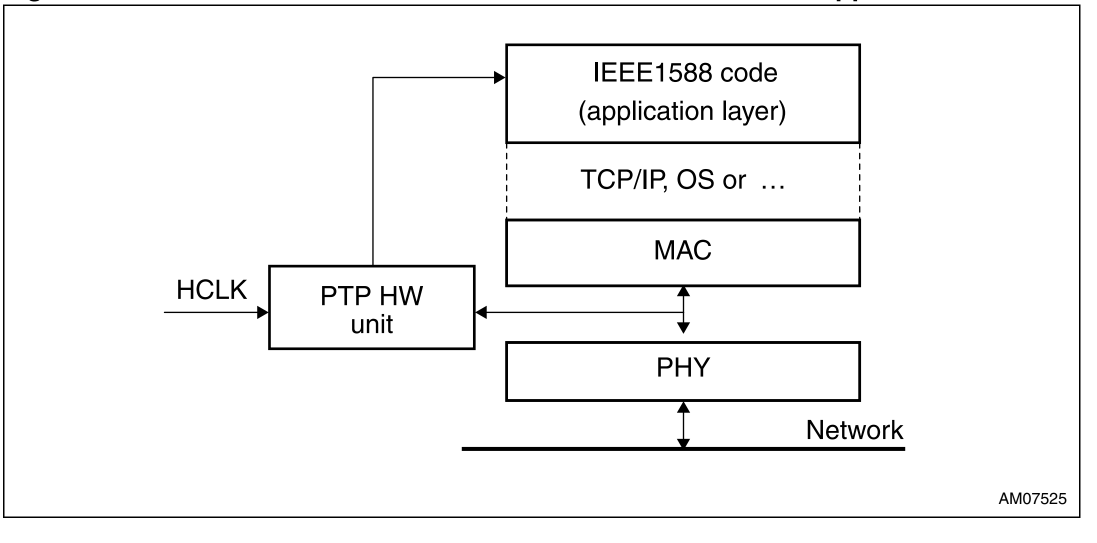
1 Introduction

To synchronize Ethernet network devices, an option to use the IEEE1588 (“Precision Time Protocol” - PTP) synchronization protocol is available. Many embedded MCUs and Ethernet PHYs available today in the market are equipped with the PTP HW time stamping unit. The PTP hardware time stamping unit allows very precise time synchronization compared to the SW solution. The hardware solution allows typically sub-microseconds time synchronization precision, the SW solution typically “only” sub-milliseconds range precision. IEEE1588 hardware unit itself is a must for precise synchronization results. In order to meet the IEEE1588 standard requirements, there must be a SW protocol stack running in the microcontroller on top of the HW.

One of the advanced features of the STM32F107's Ethernet MAC controller is the time stamping of the incoming and the outgoing packets by hardware. In this application note, you can find a real application that uses this feature: IEEE1588 PTP HW unit, *Figure 1*. The objective of this application note is to present a demonstration package built on top of the free lwIP TCP/IP stack and the free PTP stack - PTPd. Support for two hardware platforms is presented, *Figure 2*.

This software package content is:
- An implementation of IEEE 1588-2002 commonly named PTP v1 over IPv4/UDP using end-to-end delay mechanism.
- An implementation of IEEE 1588-2008 commonly named PTP v2 over IPv4/UDP using both end-to-end and Peer-to-Peer delay mechanisms.
- A target time example, which generates external trigger events at precise time.

Figure 1. STM32F107 PTP HW unit and its interaction with the application software



Contents

List of tables

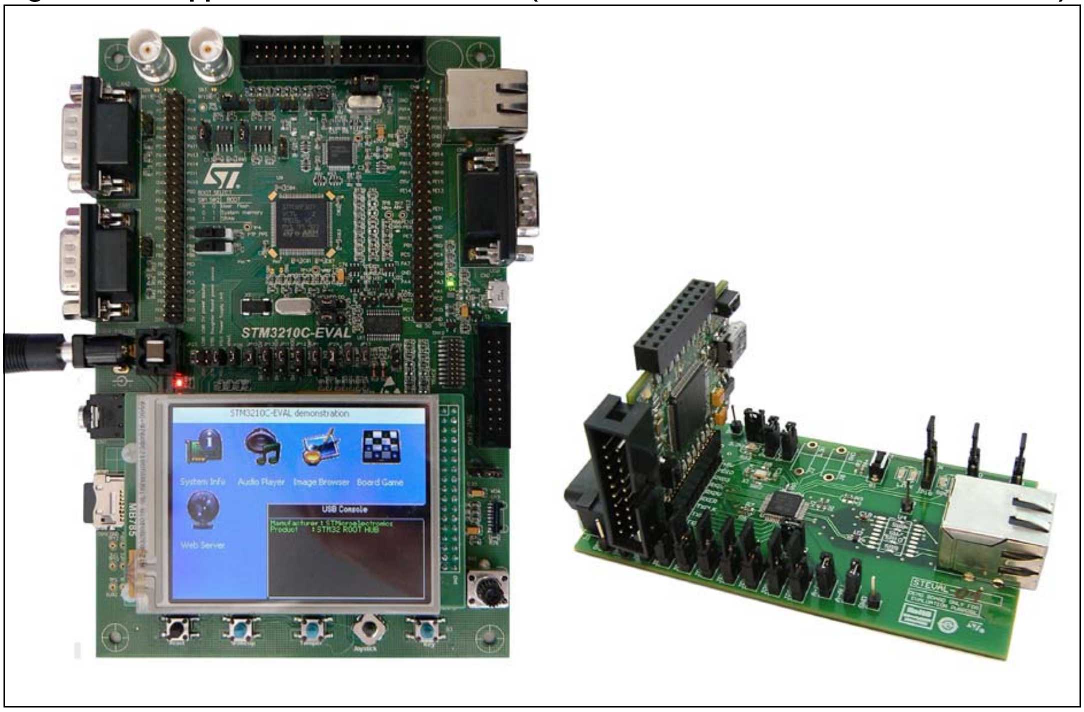
List of figures

2 Description

This application note presents implementation of the IEEE1588-2008 PTP protocol for STM32F107 microcontroller. IEEE1588 – 2008 is not backward compatible to the older IEEE1588 - 2002 version of this specification. IEEE1588-2002 implementation example is also available in source codes for STM32F107, but it is not described in this application note. Industrializations focus of the customers today is IEEE1588-2008.

The PTP daemon (PTPd) implements the precision time protocol (PTP) as defined in the IEEE1588 specification. The PTPd Version 1 implements IEEE 1588-2002 compliant functionality, and the PTPd Version 2 implements newer IEEE 1588-2008 specification. PTPd was developed to provide a precise time coordination of LAN connected computers. PTPd can run on most 32-bit and 64-bit processors. It does not require any FPU, therefore it is by definition easy to be used in small embedded processors. The PTPd originally runs on Linux, µClinux™, FreeBSD®, and NetBSD operating systems. It is also easy to port it to other platforms. The PTPd time stamping unit is originally software based and therefore for the STM32 use it has been adapted in order to benefit from the STM32 PTP hardware unit.

Figure 2. Supported evaluation boards (STM3210C-EVAL and STEVAL-PCC010V1)



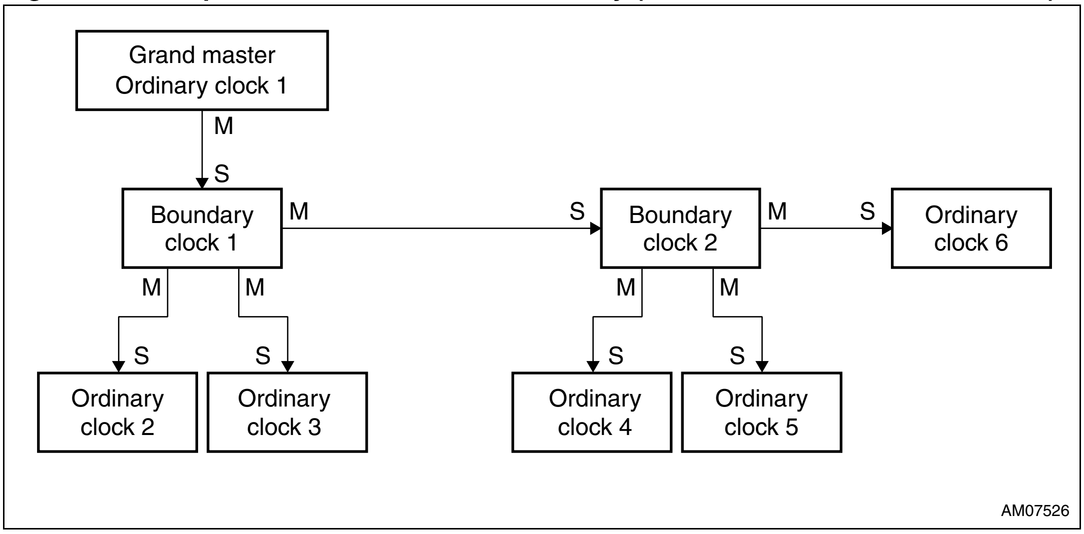
The SW implementation of PTPd is based on STMicroelectronics™ application note AN3102 (lwIP TCP/IP stack demonstration for STM32F107xx connectivity line microcontrollers) as available from ST website www.st.com/stm32. The AN3102 source codes of the project have been modified for operation with the PTPd protocol stack.

3 STM32F107 PTP stack implementation software resources

3.1 Precision time protocol (PTP)

The IEEE 1588 standard defines a protocol that allows precise clock synchronization in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The protocol applies to systems that communicate by local area networks supporting multicast messaging, including Ethernet. This protocol is used to synchronize systems that include clocks of different precision, resolution and stability. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with a minimum network and local clock computing resources. The message-based protocol, known as the precision time protocol (PTP), is transported over UDP/IP. The system or network (example in *Figure 3*) is classified into master and slave nodes for distributing the timing/clock information.

Figure 3. Simple master - slave clock hierarchy (M - master clock, S - slave clock)



The precision time protocol uses multicast messaging over UDP/IP. The underlying TCP/IP stack should have multicast support functionality or at least has to pass multicast messages. There are at least two free implementations of the TCP/IP stacks - lwIP (Light weight IP) and µIP. The lwIP TCP/IP stack has been chosen for the PTPd demonstration because of its support for multicast and IGMP messages. The µIP TCP/IP stack can also work, but it does not have support for IGMP protocol so it is not suitable for networks where switches with IGMP snooping are used. The IEEE1588 protocol itself has at least one free implementation with many derivates. Its name is PTPd and it is designed for Linux and FreeBSD systems.

3.1.1 lwIP stack overview

The lwIP TCP/IP stack is a free TCP/IP stack developed by Adam Dunkels at the Swedish Institute of Computer Science (SICS) and is licensed under the BSD license. The source code can be downloaded from http://savannah.nongnu.org/projects/lwip/. The lwIP TCP/IP stack supports the following protocols: IPv4, IPv6, UDP, TCP, ICMP, IGMP, SNMP, ARP and PPP.

The lwIP offers three types of API (“Application Programming Interface”):

- A raw API: it is the native API used by the lwIP stack itself to interface with the different protocols.
- A NETCONN API: it is a sequential API with a higher level of abstraction than the raw API.
- A socket API: it is a Berkeley-like API

The API used to build the PTPd demonstration with STM32F107 is the raw API. The raw API selection has been made because of the standalone implementation of the lwIP TCP/IP stack example. Nevertheless the achievable time synchronization precision should be the same for all three APIs, but neither Netconn nor socket API has been used and changed to reflect the needs of timestamps in this PTPd implementation example. Only the raw API has been modified to work with the PTPd software stack. Both, the Netconn and the Socket APIs need an operating system. More information about the lwIP protocol version for the STM32F107 microcontroller can be found in the application note AN3102 available from the STMicroelectronics website http://www.st.com/stm32.

3.1.2 PTPd stack overview

The PTP daemon (PTPd) is a free implementation of the precision time protocol (PTP) as defined by the IEEE 1588 (2002/2008) standards. PTPd is complete implementation of the IEEE 1588 specification for standard (non-boundary) clock. The source code for PTPd is freely available under a BSD-style license. The source code can be downloaded from www.ptpd.sourceforge.net. The PTPd has two versions. PTP Version 1 implements IEEE 1588-2002 specification, and PTP Version 2 implements IEEE 1588-2008 specification. As mentioned, the PTPd protocol has to be adapted to work with the PTP HW time stamping unit of the STM32F107 microcontroller.

4 STM32F107 PTP hardware unit set-up

This chapter describes in details the STM32F107 PTP hardware unit initial set-up, programming steps for the fine correction method and resources for the PTP information triggering in the customer application.

4.1 Initialization of the STM32F107 hardware time stamping unit

The first step is the initialization of the time stamping unit of the embedded Ethernet MAC interface of STM32F107. The startup sequence is prepared in ETH_PTPStart function. This enables the time stamping ability of MAC controller, then it set ups default values of the time stamping registers, namely the addend and the increment registers. Finally it sets the current time to 0 s.

1. Mask the time stamp trigger interrupt by setting bit 9 in the MACIMR register.
2. Program time stamp register bit 0 to enable time stamping.
3. Program the sub-second increment register based on the PTP clock frequency.
4. Program the time stamp addend register and set time stamp control register bit 5 (addend register update).
5. Poll the time stamp control register until bit 5 is cleared.
6. To select the fine correction method program time stamp control register bit 1.
7. Program the time stamp high update and time stamp low update registers with the appropriate time value. (can be zero)
8. Set time stamp control register bit 2 (time stamp init).
9. The time stamp counter starts operation as soon as it is initialized with the value written in the time stamp update register.
10. Enable the MAC receiver and transmitter for proper time stamping.

4.2 Correction methods for the local clock

There are two possible methods of the clock correction supported in the STM32F107 IEEE1588 time stamping unit: fine and coarse correction methods. In the here described implementation example the fine correction method is used because it allows more precise synchronization results in comparison with the coarse correction method. The correction method should be selected in the initialization step of the hardware time stamping in function ETH_PTPStart. After that only the appropriate functions should be used for the clock correction.

If coarse method is used, only the ETH_PTPTime_UpdateOffset function can be used to perform local time corrections. In contrast if the fine correction method is used, only the ETH_PTPTime_SetTime and ETH_PTPTime_AdjFreq can be used.

ETH_PTPTime_UpdateOffset updates the current clock by the relative difference. The function call argument is added to or subtracted from the current time.

ETH_PTPTime_SetTime sets the absolute time. The function call argument is set as the current time.

ETH_PTPTime_AdjFreq adjusts the addend register value. The argument is the relative change of the default clock frequency in ppb (parts per 10^9 - billion). If the crystal used in final application is for example 5 ppm off, then setting this value to the 5000 will compensate the error.

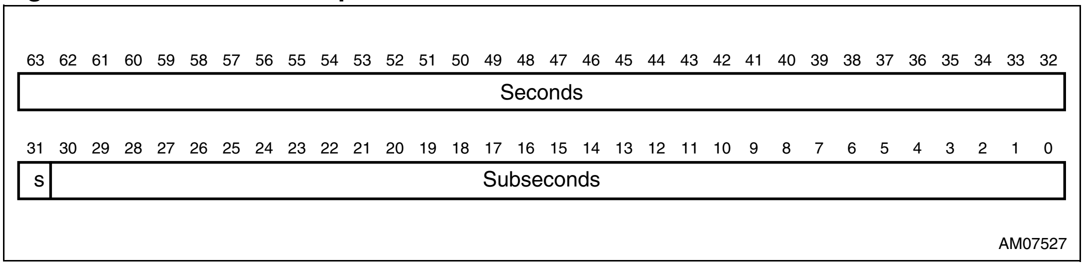
Following steps are used to perform the update of the addend register in function AdjFreq:

1. Calculate addend register value.
2. Update the time stamp addend register “ETH_PTPTSAR” with calculated value.
3. Enable the time stamp addend register by setting bit TSARU in ETH_PTPTSCR.

4.3 Data format of the time stamp

The registers holding the time stamps are using specific 64-bit format. The highest 32-bit register is unsigned integer holding number of seconds. Lowest 31 bits in the second 32-bit register are used for the fractional part of second and the 32nd bit is a negative sign.

Figure 4. PTP time stamp data format



In order to use the registers value in PTP stack it is necessary to convert these values to another format. Structure with signed, both seconds and nanoseconds, is used. So it is also necessary to convert the subseconds to nanoseconds and vice versa. For this purpose functions ETH_PTPSubSecond2NanoSecond and ETH_PTPNanoSecond2SubSecond are implemented.

This 64-bit data format is used in all time stamp related registers (ETH_PTPTSHR, ETH_PTPTSLR), time stamp update registers (ETH_PTPTSHUR, ETH_PTPTSLUR), Target time registers (ETH_PTPTTHR, ETH_PTPTTLR) and also in the DMA descriptors.

4.4 Computing the default values of the time stamp unit registers

If using the Fine correction method, the default values of the addend and the increment registers can be computed as follow.

Equation 1

$$\text{tick} = \frac{\text{Increment} \cdot 10^{9}}{2^{31}}$$

$$\text{Adden}(d \cdot \text{Increment}) = \frac{2^{63}}{\text{SysClk}}$$

For example, if *SysClk* is 72 MHz, we can chose tick approximately 20 ns.

Equation 2

$$\text{Increment} = \frac{20 \cdot 2^{31}}{10^{9}} = 42.94 \cong 43(0x2B)$$

$$\text{tick} = 43 \cdot \frac{10^{9}}{2^{31}} = 20.023\text{ns}$$

We can see that tick is not precisely 20 ns as we choose because of rounding increment value. Using the next equation, we can compute default value of the addend register.

Equation 3

$$\text{Addend} = \frac{2^{63}}{\text{SysClk} \cdot \text{Increment}}$$

$$\text{Addend} = \frac{2^{63}}{72\text{M} \cdot 43} = 2979125334.90$$

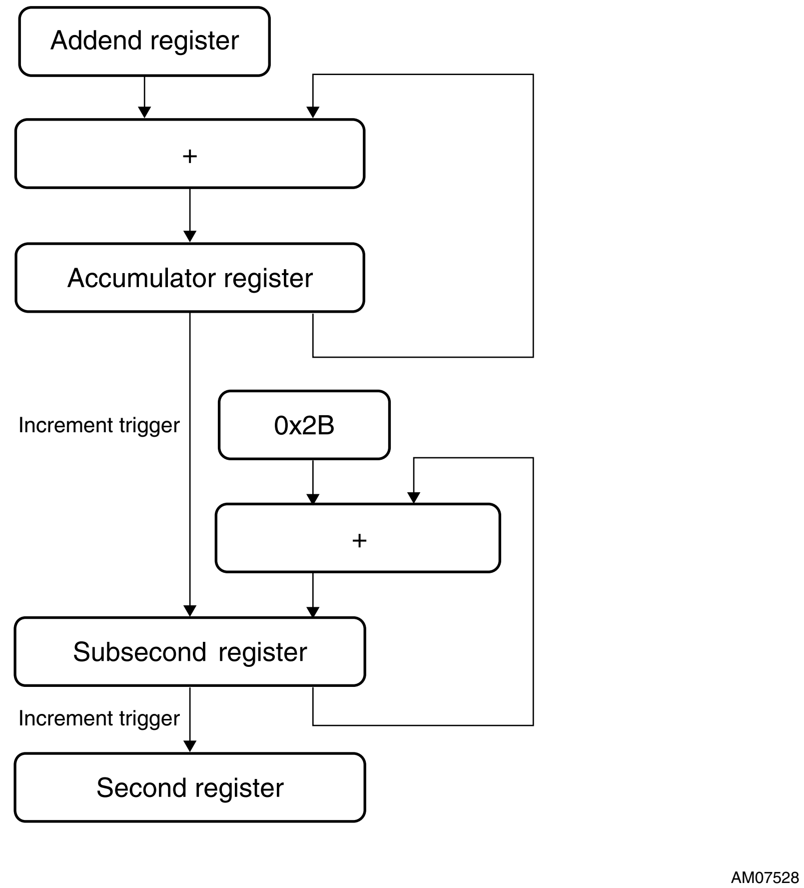
$$\text{Addend}_{\text{default}} = 2979125335(0xb191d857)$$

Value of the tick can be selected differently but it is necessary to validate the range of the increment and addend registers. The increment register is of data type unsigned char (8-bit) and the addend register is of data type unsigned long (32-bit). It is also necessary to validate the regulation range of the addend register.

Table 1. Examples of different default addend register values vs. increment register value for SysClk = 72 MHz

Tick	Increment	Addend
119 ns	255	0x1DF170C7
100 ns	215	0x238391AA
50 ns	107	0x475C1B20
20 ns	43	0xB191D856
14 ns	30	0xFE843E9E

Figure 5. Fine correction method



4.5 Generating trigger events

In the example delivered with this application note we have used an easy way to generate external trigger events. To enable this feature timer TIM2 should be properly configured following these steps.

1. Remap ITR1 input of TIM2 to the output of target time event by resetting bit TIM2ITR1_IREMAP of register AFIO_MAPR.
2. Set the prescaler, period and counter mode of TIM2.
3. Configure appropriate timer output to PWM1 mode.
4. Enable fast output compare state.
5. Select one pulse mode of TIM2.
6. Select ITR1 as input trigger for TIM2.
7. Select slave mode for TIM2.

If the timer TIM2 is configured to generate the target time events, interrupts can be enabled by unmasking interrupt bit TSTIM in register ETH_MACIMR.

Scheduling of the trigger event can be done by the following steps.

1. Set the target time registers ETH_PTPTTLR and ETH_PTPTTHR (the time which will occur later from the current - it is the time when you want to perform the scheduled event).
2. Enable bit TSITE in register ETH_PTPTSCR.

When the time is greater than the target time, event will be generated and the appropriate output pin will generate pulse. Every time the next trigger event would be scheduled these two steps should be repeated.

5 Software porting of the PTPd stack for STM32F107

The application note AN3102 describes management of all basic needs for the PTP protocol - the lwIP TCP/IP stack implementation for STM32F107, but it has not been originally considered that it will be ever extended with the PTP functionality. Some minor changes to the original lwIP code has been done. The same is valid for the original Ethernet MAC low level driver. Changes which have been done to the original code as presented in AN3102 are briefly described in the following sections.

5.1 Modifications of the STM32F107 Ethernet MAC low level driver

Low level function for transmission of the packets is slower than the original one (without time stamping) because it must wait until the timestamp is known. The time stamp is measured at the beginning of the packet transmission (when the physical raw frame preamble is present at the MII, RMII interface).

A set of functions to convert hardware internal subsecond value to PTP nanosecond value were added (ETH_PTPSubSecond2NanoSecond, ETH_PTPNanoSecond2SubSecond). Functions to get and to set the precise time were added (ETH_PTPTimeStampGetTime, ETH_PTPTimeStampSetTime). Functions to adjust time base in fine correction method mode (ETH_PTPTimeStampAdjFreq) and in the Coarse correction method mode (ETH_PTPTimeStampUpdateOffset) were also added.

Function ETH_PTPTimeStampAdjFreq is the equivalent of adjtimex, ntp_adjtime or SetSystemTimeAdjustment known from operating systems. Its argument is the time correction in units of ppb (parts per 10^9).

It is also essential to enable multicast frames on the interface. To do that the Ethernet controller has to be initialized with ETH_MulticastFramesFilter = ETH_MulticastFramesFilter_None; This disables the multicast filter completely. All multicast packets are passed through.

5.2 Modifications of the lwIP stack

The official release of lwIP does not allow passing time stamps from the Ethernet interface to the user application. Additional fields to the packet structure (pbuf) have been added (seconds and nanoseconds fields). It was also necessary to modify the source of UDP packet handling (Utilities\lwip-1.3.1\src\core\udp.c) to ensure relaying of timestamps of the transmitted packets.

The other modification is related to the handling of the packet timestamps (interaction of the low level MAC driver with the lwIP stack). All functions in the ethernetif.c file (under Utilities\lwip-1.3.1\src\netif) have now their equivalents with PTP prefix, namely ETH_PTPTxPkt_ChainMode and ETH_PTPRxPkt_ChainMode which are the core functions for handling of the packet timestamps.

5.3 Modifications of the PTPd stack

The official release of the PTPd does not provide any porting to any microcontroller. The PTPd however comes with two parts. The PTP stack itself and OS (“Operation System”) dependent functions. These functions could be rewritten to support a specified architecture. These OS dependent functions have been rewritten to work without the OS using only interrupts and the lwIP stack. Two separate packet queues have been created for event and general messages. This is done because executing whole PTP stack takes long time and it is not efficient to execute it at interrupt level.

PTPd expects time stamping of packets only for incoming messages. Outgoing messages are transferred through the internal loopback of the Ethernet interface. This method is used in the original PTPd stack code because time stamping is not considered to be precise (SW time stamping). The modified stack for STM32F107 uses both time stamps for incoming and outgoing frames captured by the HW because they are both very precise. Additional changes have been done in protocol.c.

5.4 Periodic PTPd tasks

Only minimal part of handling incoming messages is done at interrupt level. Packet is added to the corresponding queue and the program control is returned back to the lwIP stack. Whole PTP stack is executed periodically in ptp_Periodic_Handle function. This function polls both packet queues and also checks all running internal timers.

If there is some packet in the queue it is handled. If some timer has expired, an appropriate event is executed. If the queue is full, the next packets are discarded until the PTP stack processes the packet in the queue and frees its space.

5.5 lwIP configuration

The lwIP can be tuned to suit the application's requirements. The default parameters of the stack can be found in the opt.h file, located under the lwIP directory at src\include\lwIP\.

To modify these settings a new file is defined, lwipopts.h, based on the opt.h file, and located under the lwIP directory at port\. It contains the lwIP configuration for the STM32F107 project. A new configuration constant “LWIP_PTP” is introduced. Enabling this directive will cause the lwIP stack to handle timestamps. This directive is enabled in our example by default. SYS_LIGHTWEIGHT_PROT has been enabled and new functions sys_arch_protect and sys_arch_unprotect have been implemented.

5.6 PTPd configuration

Parameters of the PTPd stack can be changed at compile time by setting default values in file constants.h, located under the PTPd directory at src\.

These settings can be overridden in file ptpd.c, located under the PTPd directory at src\.

Some of these parameters can be change at runtime by PTP management messages. This is implemented only in v1 implementation of the PTPd.

Important parameters, that can be modified are

- Sync interval (v1, v2)
- Announce interval (v2)
- Delay mechanism (v2)
- Clock priority
  - ClockPreferred (v1)
  - ClockStratum (v1)
  - SlaveOnly (v1, v2)
  - Priority1, priority2 (v2)
  - ClockQuality (v2)
- Display stats (v1, v2)
- Domain name (v1), number (v2)

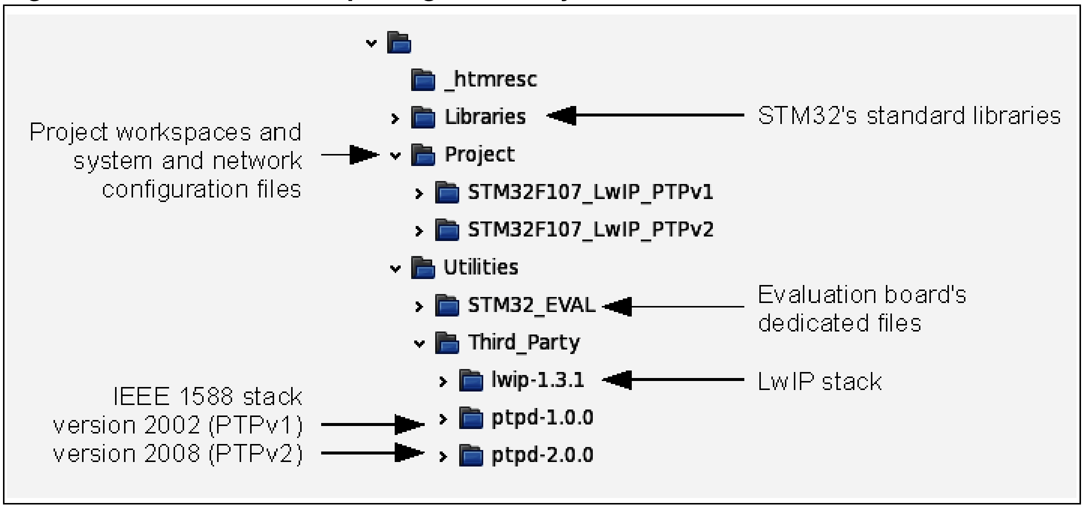
6 Getting started with the demonstration software

This section describes the necessary steps which have to be followed in order to run successfully the PTPd demonstration software on the STM32F107 platforms. Description of the package directories, STM32F107 boards configuration, software project configurations / compilation and application HW set-up is described in details.

6.1 Package directories

The software implementation of the PTPd is based on STMicroelectronics application note AN3102 (lwIP TCP/IP stack demonstration for STM32F107xx connectivity line microcontrollers) as available from ST website www.st.com/stm32. The AN3102 source codes of the project have been modified for operation with the PTPd protocol stack. The software package has the same structure as the AN3102 project structure, shown in *Figure 6*, with support files and directories added for the PTPd stack operation.

Figure 6. PTPd software package directory structure

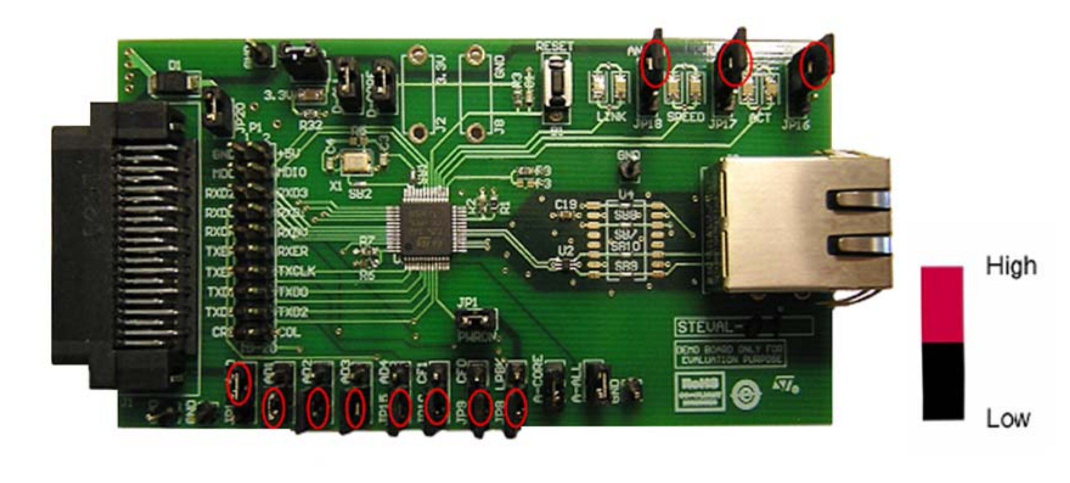


6.2 Configuration of the demonstration boards

In this chapter, the hardware configuration of the demonstration boards is presented. The PTPd STM32f107 project has been tested on STM3210C-EVAL and STEVAL-PCC010V1 demonstration boards. Nevertheless support for Keil™ MCBSTM32C evaluation board is also available, hardware set-up description is not described in this chapter, please refer to the appropriate documentation.

6.2.1 STEVAL-PCC010V1 hardware configuration

Figure 7. STEVAL-PCC010V1 - ST802RT1 board configuration



MII configuration settings:

JP11 - High

JP12, JP13, JP14, JP15, JP10, JP9, JP8, JP16, JP17, JP18 - Low

Validate the soldering position of solder bridges SB1,SB2 and SB3 on the STM32F107 controller board as described in *Table 2*. MII configuration has to be selected.

For more information regarding the possible HW set-up scenario (e.g. RMII mode use), please refer to UM0819 - Getting started with STEVAL-PCC010V1, ST802RT1 TX mode Ethernet PHY demonstration kit.

Figure 8. STEVAL-PCC010V1 - STM32F107 demonstration board set-up

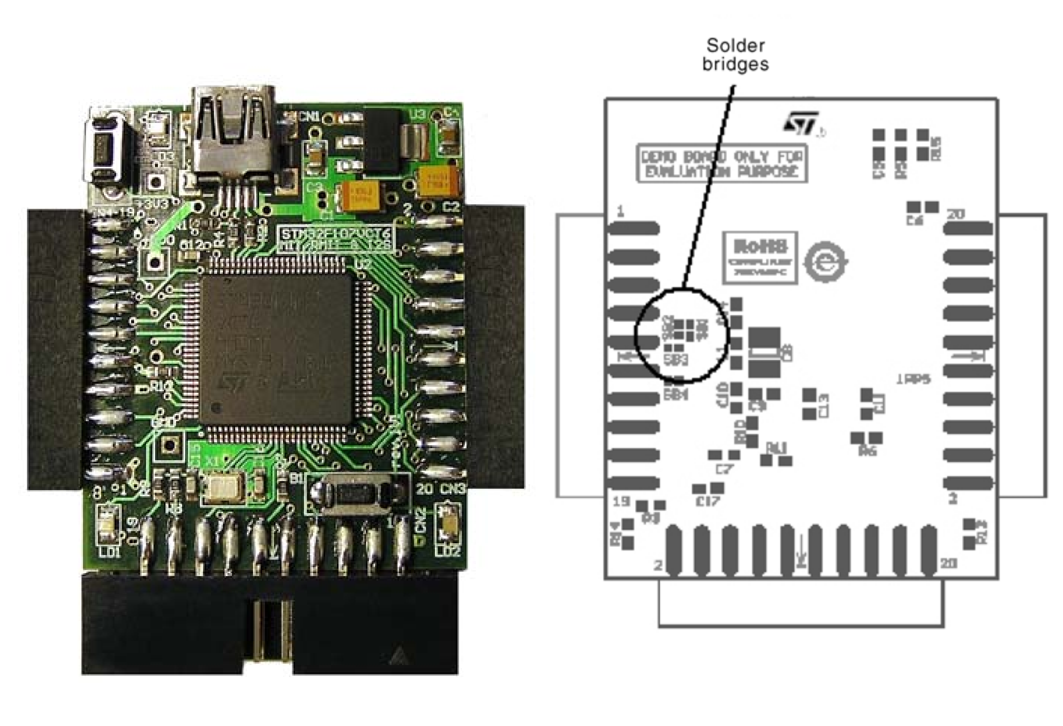


Table 2. STEVAL-PCC010V1 MII/RMII interface STM32F107 add-on board selection by solder bridges SB1, SB2 and SB3

	SB1	SB2	SB3
MII	Remove	Remove	Fit
RMII	Fit	Fit	Remove

6.2.2 STM3210C-EVAL hardware configuration

Figure 9. STM3210C-EVAL configuration for MII functionality in PTPd application example

STM3210C-EVAL board HW set-up:

- JP3, JP10, JP11, JP12, JP13 are in 2<->3 position.
- JP4, JP14 are in 1<->2 position.
- JP2 is open. (NC)

For more information regarding the possible HW set-up scenario, please refer to UM0600 - STM3210C-EVAL evaluation board.

6.3 Configuration of the PTPd SW project

There are two options prepared in terms of software development tools support. Everybody can decide to use either the RIDE7 or Keil µVision environment. In this chapter, a description of how to configure the project properly is described for RIDE7. In order to run the PTPd demonstration software, you need to customize several constants in the project file structure. All necessary and few optional steps are described in the following chapters.

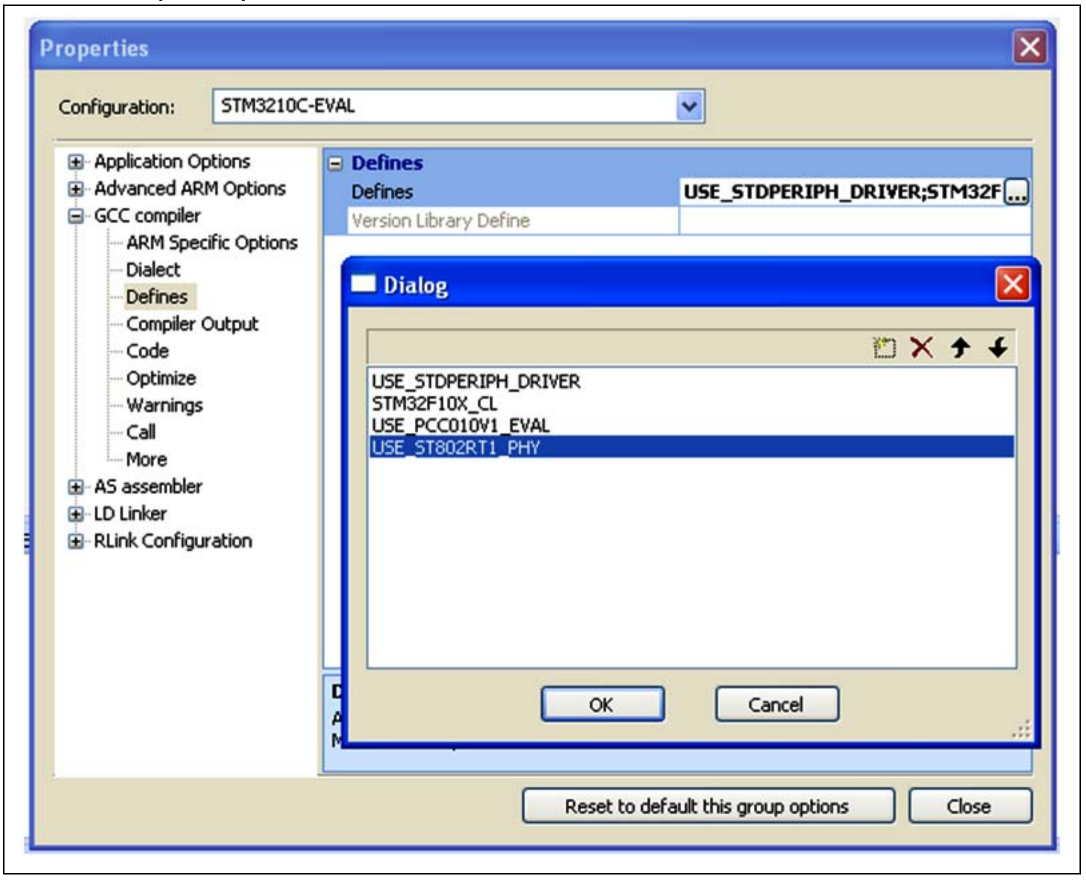
6.3.1 HW platform selection

This example is designed to work with two STM32F107 target platforms. In the project you will find support for three Ethernet PHY chips, which can be selected using USE_ST802RT1_PHY, USE_LAN8700_PHY or USE_DP83848_PHY directive.

Depending on the hardware configuration, RMII or MII mode should be selected by directives RMII_MODE and MII_MODE in stm32f107.c file, located under the project directory in src\. It is strongly recommended to use MII interface in order to limit additional latency and non-determinism in the system.

In the project file there are also prepared configurations for two development boards, namely USE_STM3210C_EVAL and USE_PCC010V1_EVAL.

Figure 10. Change the project compilation defines according to the board used (RIDE7)



With respect to the hardware platform selected, the project must be modified to follow the HW features. Use the project properties (Ctrl+Alt+Enter) to change the compilation options (*Figure 10*) for the demo board used (GCC compiler / defines):

USE_PCC010V1_EVAL (MII/RMII)

USE_STM3210C_EVAL (MII/RMII)

USE_MCBSTM32C_EVAL (only RMII supported) - configuration and implementation details are not described in this application note

Ethernet PHY selection:

USE_ST802RT1_PHY (STEVAL-PCC010V1)

USE_DP83848_PHY (STM3210C-EVAL)

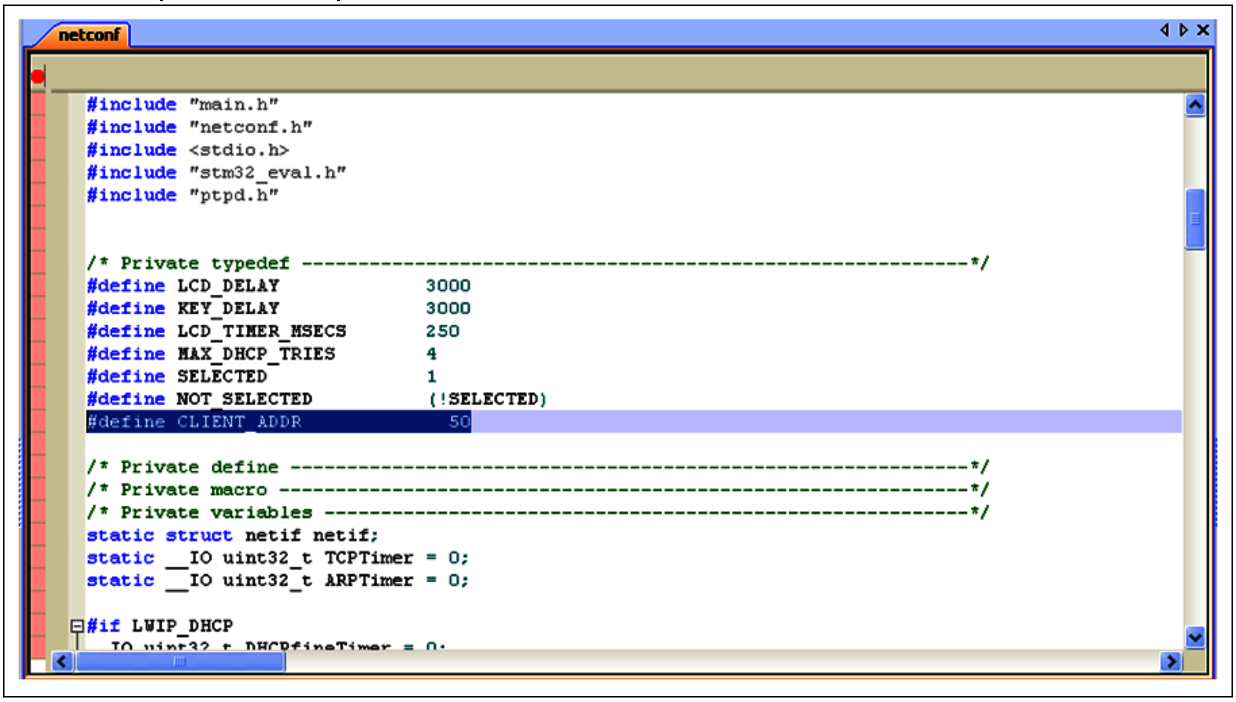
USE_ DP83848_PHY (MCBSTM32C)

6.3.2 MAC and IP address settings

Both addresses, the MAC and IP are derived from CLIENT_ADDR constant. Every device in the network must have an unique MAC address. If it is not true, this implementation of PTP will not work. The IP address is static 192.168.0.xx where the last number is the CLIENT_ADDR. The CLIENT_ADDR is also the last number of MAC address, *Figure 10*.

The UUID parameter is evaluated by the best master clock (BMC). UUID is derived from the MAC address so it is also modified by the CLIENT_ADDR constant. In other words the higher number of CLIENT_ADDR will cause lower priority of the device in BMC algorithm. For example using two evaluation kits with default parameters: one board with CLIENT_ADDR=1 will be the master clock if the other board has CLIENT_ADDR>1. If the other board has also CLIENT_ADDR=1 synchronization will not work. It is necessary to compile the PTPd project specifically for each board in the system.

Figure 11. Both addresses, MAC and IP, are derived from constant CLIENT_ADDR (/src/netconf.c)

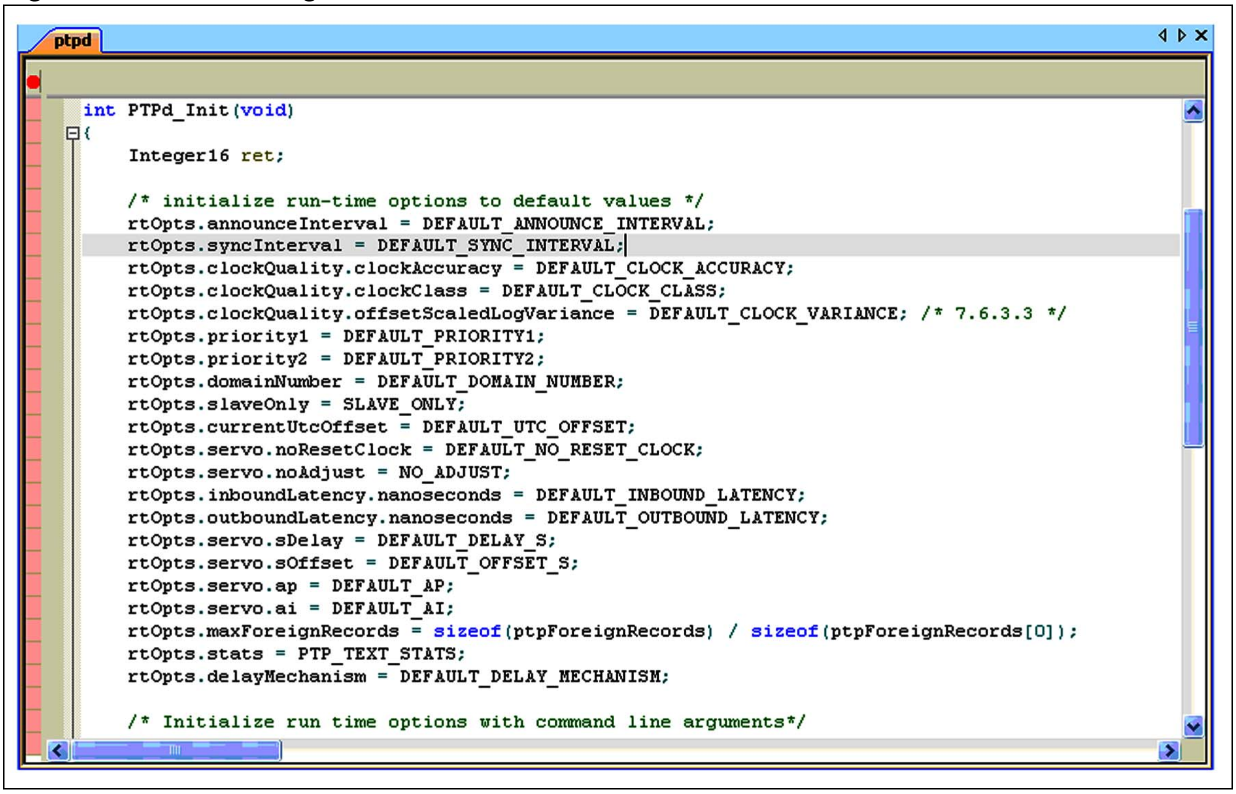


6.3.3 PTPd settings

In order to achieve the correct functionality of the software example; PTP version, sync interval, announce interval, domain and delay mechanism should be equal for all nodes that are designed to communicate together.

Using another synchronization interval is possible in order to achieve a better synchronization result. The default synchronization interval (DEFAULT_SYNC_INTERVAL, *Figure 12*) in the project is 1 s (one second), file ptpd.c. The sync interval can be any value of power of 2. For example the sync interval 0.125 s is 2-3, the correct setting would be rtOpts.syncInterval = -3 in the ptpd.c file, *Figure 12*.

Figure 12. PTPd settings



For the STM3210C-EVAL board it is also possible to switch on/off the on board LCD that displays the PTPd information. In ptpd-2.0.0/src/ptpd.c there is an option for it:

rtOpts.stats = PTP_TEXT_STATS;

or

rtOpts.stats = PTP_NO_STATS;

6.3.4 Compiling the project and flashing the HW platform

There two IDE project options prepared in the PTPd project hierarchy located in directory /Project/STM32F107_LwIP_PTPv2; here you can find RIDE7 and RVMDK (Keil µVision) project files at your convenience. Open the appropriate project file, configure the project as described in the previous chapters, compile it for each board and flash it.

6.4 Application boards connections

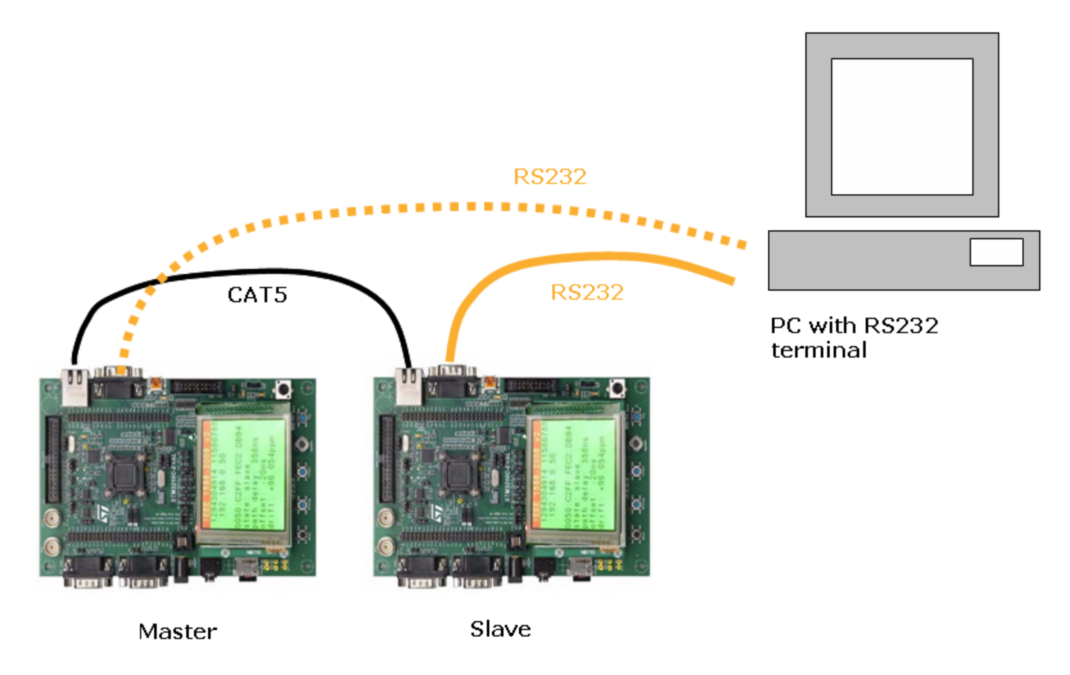
This section shows several different application scenarios, connection of the hardware platforms, with short description of the application hints. In order to use the demonstration firmware, at least two hardware platforms are required. The first device is a board with the PTPd firmware (STM3210C-EVAL, STEVAL-PCC010V1) and the second platform can be another board or other PTP capable device, e.g. Linux PC with PTPd daemon. In case of using two boards with the PTPd firmware, each board should have its own unique MAC address and all other PTP parameters should be kept the same, see *Section 6.2* and *Section 6.3*.

6.4.1 Back-to-back connection of two boards

When connecting the boards directly, back-to-back, one of them will become a master and the other one a slave (if all parameters but the MAC address are the same, the master will be device with lower MAC address). After few PTP synchronization cycles the boards are synchronized. Both boards (in case of using STM3210C-EVAL platforms) should show its state (one should show master and the other should show slave). Slave board should also show measured transfer time of message from master to slave (mean path delay), current expected offset from master and measured crystal oscillator frequency relative difference from the master in ppm (pulse per million).

For each of the two boards you have to configure the SW project as described in chapters 7.2 and 7.3. The RS232 cable connection is optional and available only for STM3210C-EVAL board, *Figure 13*. In order to use it with a PC COM terminal, it is required to configure the ptpd-2.0.0/src/dep/constants_dep.h file, to uncomment PTPD_DBGV directive before the compilation. It is also possible to combine different HW platforms i.e. one STM3210C-EVAL and one STEVAL-PCC010V1 board.

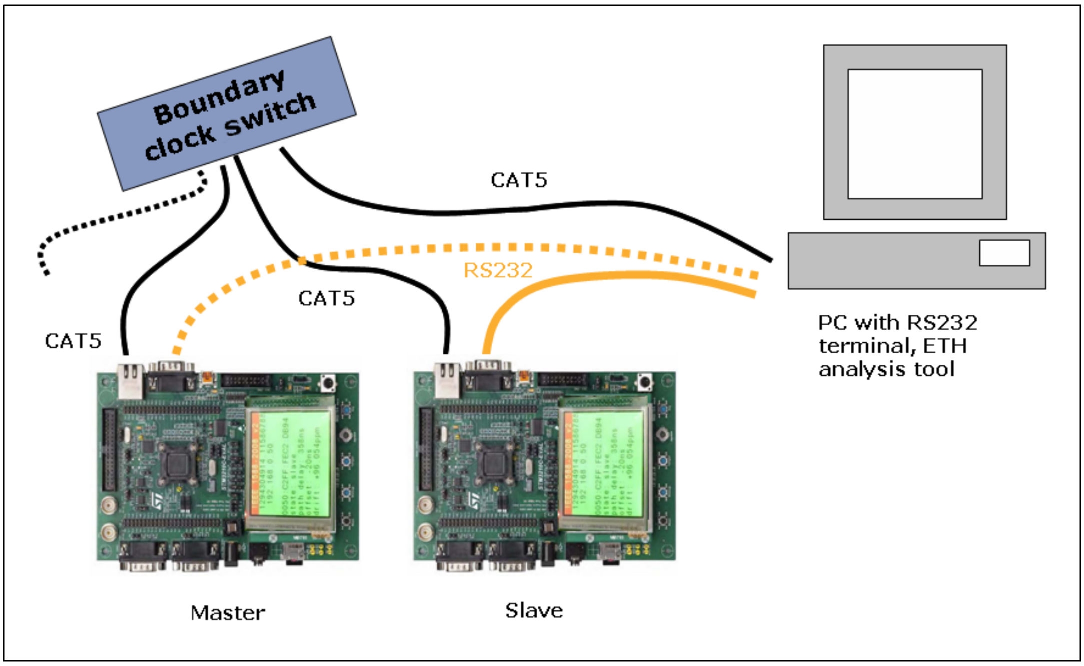
Figure 13. Back-to-back connection of the two boards



6.4.2 Boundary clock switch option

In comparison with the back-to-back connection option, this configuration allows to connect two boards and more. You can also involve e.g. a packet analysis tool running in your PC for packet sniffing (e.g. Wireshark). This configuration requires an Ethernet switch with boundary clock functionality. Using standard Ethernet switch brings uncertainty (jitter) in the system and causes degradation of the best achievable synchronization results.

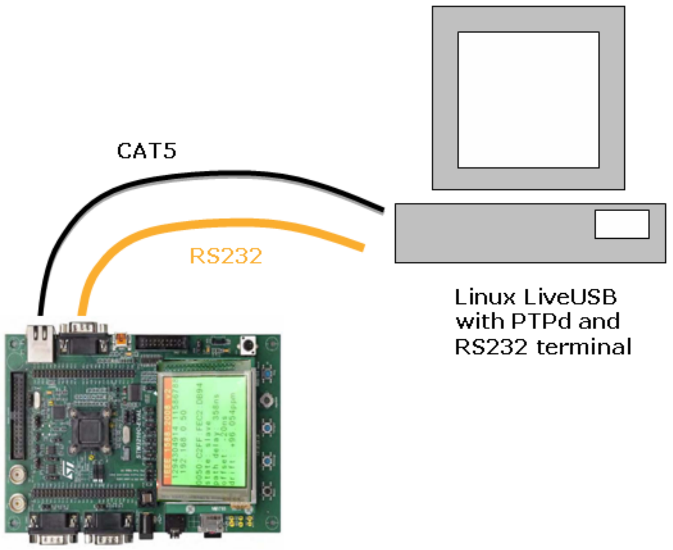
Figure 14. Two boards connection through boundary clock switch, packet sniffing in PC



6.4.3 Linux LiveUSB

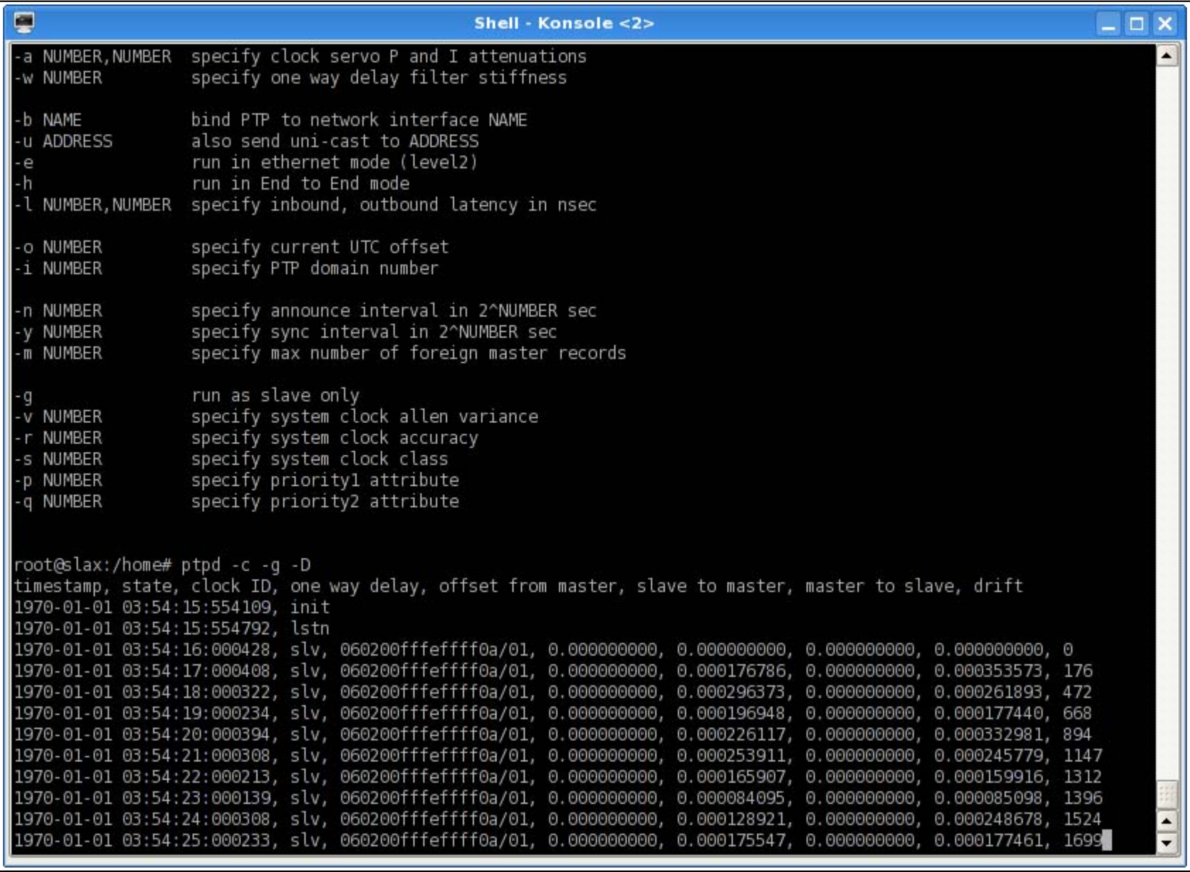
Many embedded microcontrollers (MCUs) and Ethernet physical layer chips (PHYs) available today in the market are equipped with a PTP HW time stamping unit allowing achieve time synchronization in sub-microsecond range. Pure SW solutions (with no dedicated PTP HW) allow achieve synchronization “only” typically in microseconds. An alternative to the previous two hardware options is therefore to use the Linux LiveUSB when not having at least two boards for demonstration. The PTPd software, according to the IEEE1588-2008 standard, can be include as object code in the scope of supply and thus enables a quick and easy introduction to the IEEE 1588 technology. The Linux LiveUSB distributions are widely used. An extensive list of the live distributions is presented at www.livecdlist.com website. The PTPd source codes can be compiled for use with the desired Linux LiveUSB distribution.

Figure 15. STM3210C-EVAL connected to a PC running Linux LiveUSB distribution with software PTPd daemon



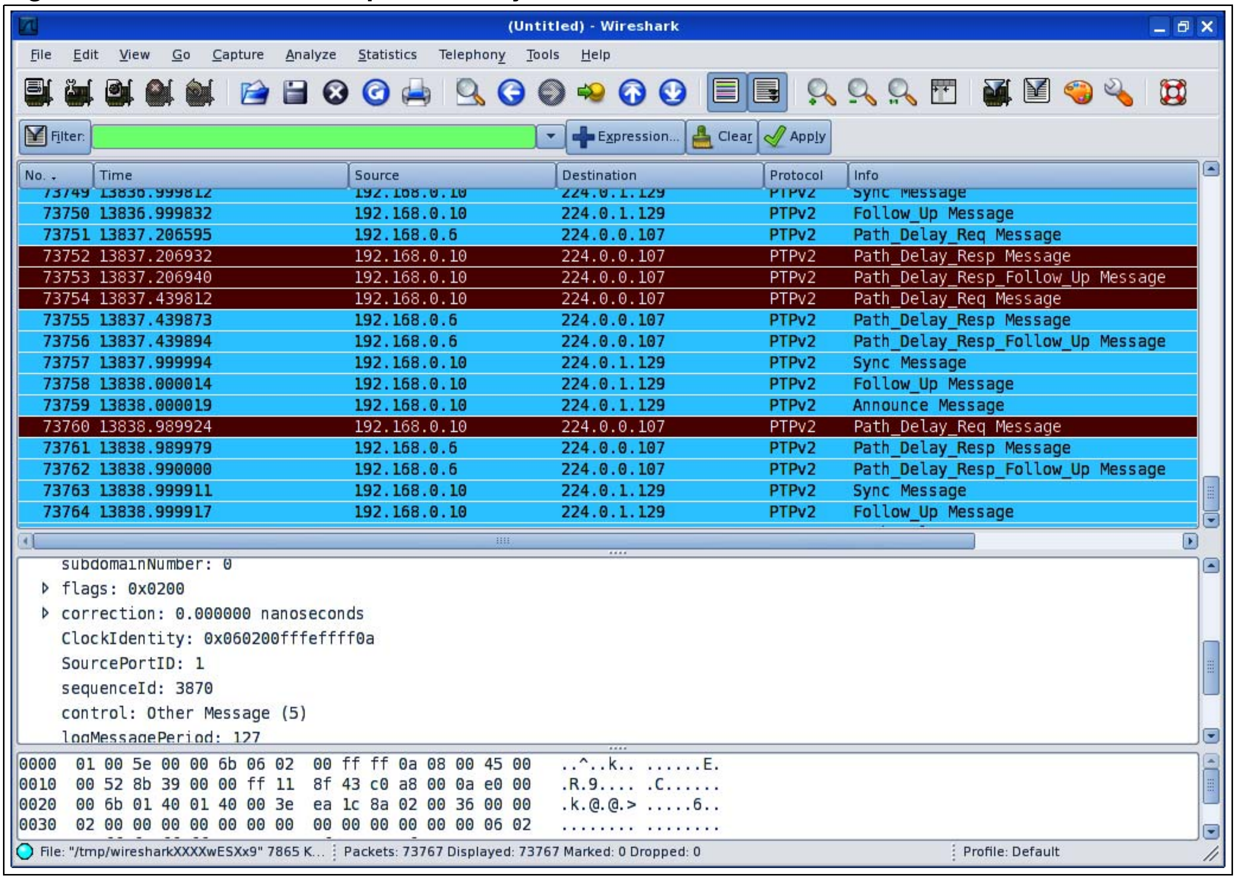
The PTPd use in the Linux LiveUSB distribution is very easy. You can run the PTPd daemon from the console window, *Figure 16*.

Figure 16. PTPd running in the Linux console window



You can kill the ptpd process if needed by using system/performance monitor. You can see the PTP message traffic in the Wireshark (www.wireshark.com) packet analyzer if you add it in your LiveUSB customization, *Figure 17*.

Figure 17. Wireshark - PTP packet analysis



PTPd daemon in Linux LiveUSB

This chapter gives a short overview and tips about how to use and configure the PTPd application in the Linux LiveUSB environment. It is considered that you run PTPd from a Linux console. PTPv2d runs on UDP/IP and by default in P2P (Peer-to-Peer) mode. PTPd is using communication ports 319, 320 therefore it is required to run in it in super user mode.

Customization of the ptpd application run:

-b eth2	bind PTP to network interface name (e.g. eth2)
-c	runs ptpd in command line (non-daemon) mode
-f FILE	sends its output to the desired file
-h	runs ptpd in E2E delay mechanism
-g	runs the ptpd in slave mode only (PC can never become a master)
-t	do not adjust the system clock (valuable when you want to observe the slave to master time drift)
-l	specify inbound and outbound latency startup value in nanoseconds

-n PTP specify announce interval

-y specify sync interval as power of 2 (-1 is equal to 0.5 seconds, etc.)

-d displays statistics in the console

-D statistics in .csv data format

Timestamp (year and time), state (master/slave), clock ID, one way delay, offset from master, slave to master delay, master to slave delay, drift

PTP clock settings (clock description according to IEEE1588 spec.)

-v specify system clock variance

-r specify system clock accuracy

-s specify system clock class

-p specify priority 1 attribute

-q specify priority 2 attribute

Servo settings:

-a specify clock servo P and I attenuations

-w specify one way delay filter stiffness

The basic implementations of PTP operate only as ordinary clock application running on top of the network protocol stack. Timestamps are generated at the Kernel level and transferred to SW. Protocol stack and HW reaction delays variations bring precision (jitter) errors in these timestamps. These errors can be in the microseconds to hundred microseconds range depending on the hardware and the operating system architecture. The negative effect of this delay fluctuation can be eliminated by suitable design of the clock servo mechanism.

Examples:

ptpd2 -c -g -D -b eth2

-g run as a slave only (not master capable)

-D display statistic in .csv format

ptpd2 -c -d -t -b eth2 -p 100

-p 100 increases priority of the PC to become master

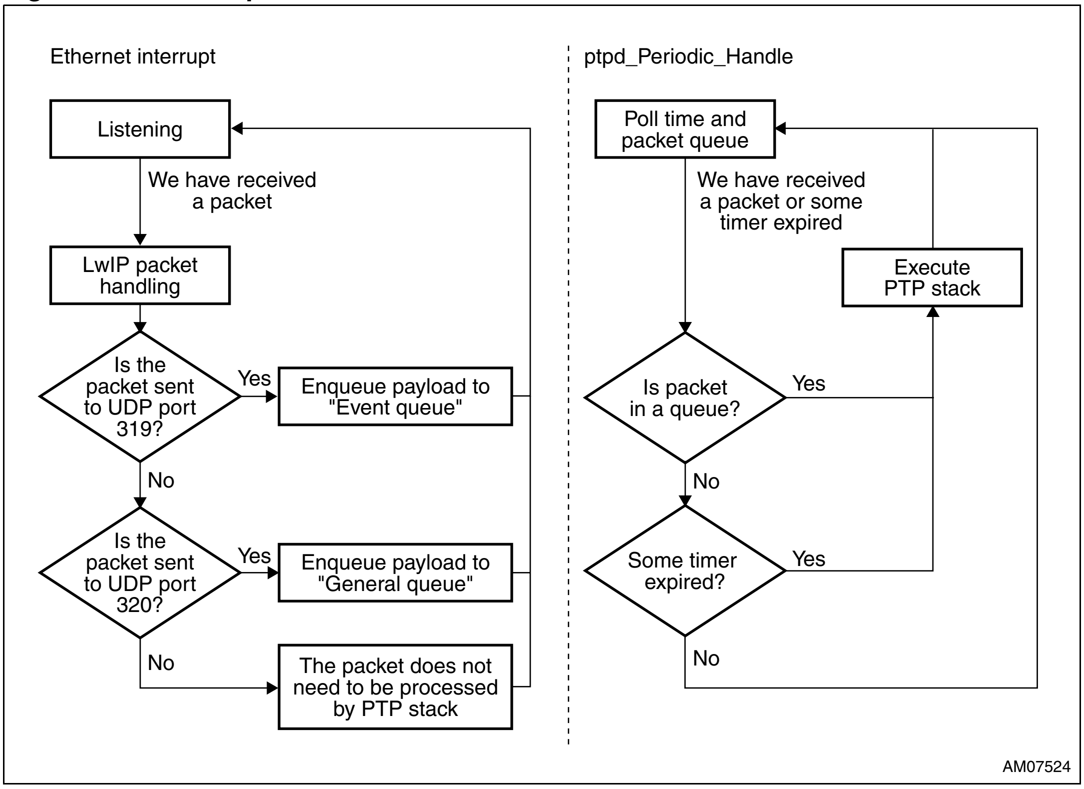
More information can be found at http://ptpd.sourceforge.net/ project website.

6.5 How to use the precise time information in the customer application

6.5.1 PTPd operation overview

When a frame is received, the Ethernet interface layer extracts the data and the timestamps and sends them to the PTP stack. This is ensured by the ethernetif.c file. The lwIP stack handles the packets at the interrupt level. If the packet is targeted to the PTP protocol stack, it is added to the appropriate (event or general) packet queue. If the packet is for another application, it is handled by that application and not processed by the PTP stack. Outside of the interrupt, the PTP packet queues are polled for new packets. If there are new packets in the queue, they are processed by the PTP stack and deleted from the queue. The PTP stack is executed also if some of its timers have expired.

Figure 18. PTPd operation overview



The whole PTP stack is executed in function ptpd_Periodic_Handle. If this function is not executed as often as needed, some incoming packets could be lost because of the full queue. If only few packets are lost, the protocol stack simply doesn't use them. The synchronization will be degraded but not lost.

6.5.2 Target time as external trigger example

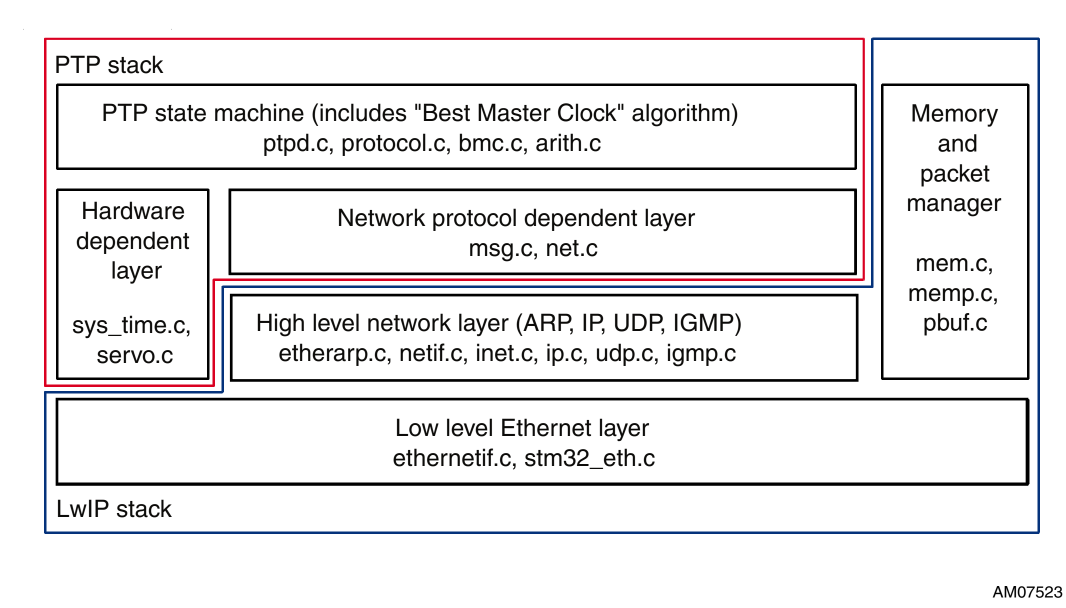
The timer TIM2 can be configured to be triggered by the target time event. Demonstration firmware is set-up to generate such event every second. This functionality is started by “TargetTime_Init function”. The functional behavior can be changed in function “ETH_IRQHandler” where the new trigger event is calculated and prepared. The output event is propagated to output of TIM2 depending on selected board. Using STM3210C-EVAL the output pin is “TIM2_CH1” (PA15).

The precision of the generated output signal depends only on the granularity of timestamps. All signals are routed in hardware with no intervention of the CPU therefore deterministic. To validate the synchronization accuracy there is an option to measure the “Pulse Per Second” (PPS) output (PB5) of each board. This signal is generated directly from the PPT hardware. The difference of the rising edges of these pulses should be near the value of measured offset from the master.

Target time can be configured to generate event at any time in the future. If the time is set to the past event is generated immediately after setting bit TSITE of register ETH_PTPTSCR.

6.6 PTPd project example structure

Figure 19. PTPd project example structure



6.7 Precision of the PTPd system

Demonstration firmware has been tested using the version 2 of the IEEE1588 PTP protocol with the “Point-to-Point” delay mechanism (P2P). The synchronization interval has been set to 1 second and in the other case to 0.125 ms. Non-equal send and receive path delays have been observed so configuration parameter of the PTP stack “OutboundLatency” has been set to 160 ns, because the mean offset has been 80 ns.

The first test has been done with the STM3210C-EVAL board connected with precise PTPv2 master clock device, *Figure 20*. The synchronization interval has been set to 1 second. In *Figure 21*, there is a histogram showing the measured offset from the master. The offset measured for this hardware set-up shows minimum and maximum offset from -600 ns to 200 ns. These variations depend on the default crystal quality used on the STM3210C-EVAL board. The second part of this test has been done with modified STM3210C-EVAL board using external low cost crystal oscillator (built from the original on-board crystal and 74HC04 inverters) connected to the X1 pin of the MCU. The on-board crystal has been removed. In *Figure 22* there is a histogram of the measured offset from the master. The offset measured for this hardware set-up shows the minimum and maximum offset from -70 ns to 70 ns. Because the oscillator drifts only by the temperature changes and the frequency does not change rapidly, the synchronization with master is better compared to the use of the default crystal and the built in oscillator.

Table 3. PTPd STM32F107 test set-up

Parameter	Value
Protocol version	IEEE 1588-2008
Sync interval	1 s, 0.125 s
Connection	Direct (back-to-back) connection without switch
Master clock	Meinberg LANTIME M600
Delay mechanism	P2P (Point-to-Point)
OutboundLatency correction	160 ns

Figure 20. PTPd STM32F107 test set-up

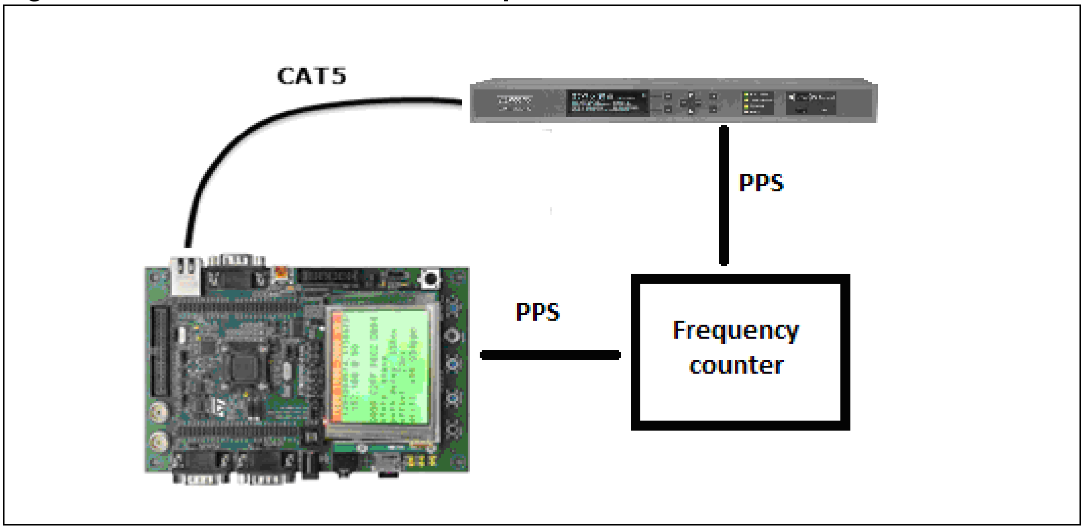


Figure 21. Precision reached using the default crystal and built in oscillator, the synchronization interval has been set to 1 second

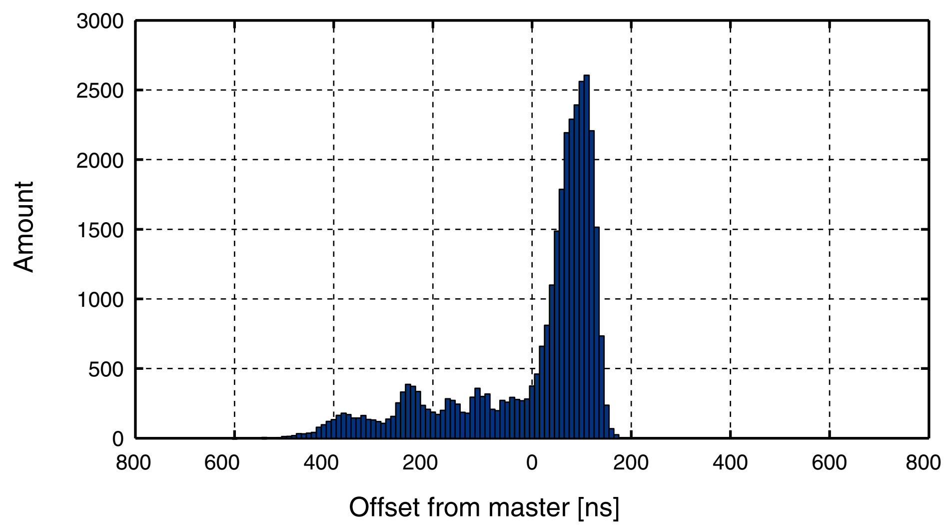
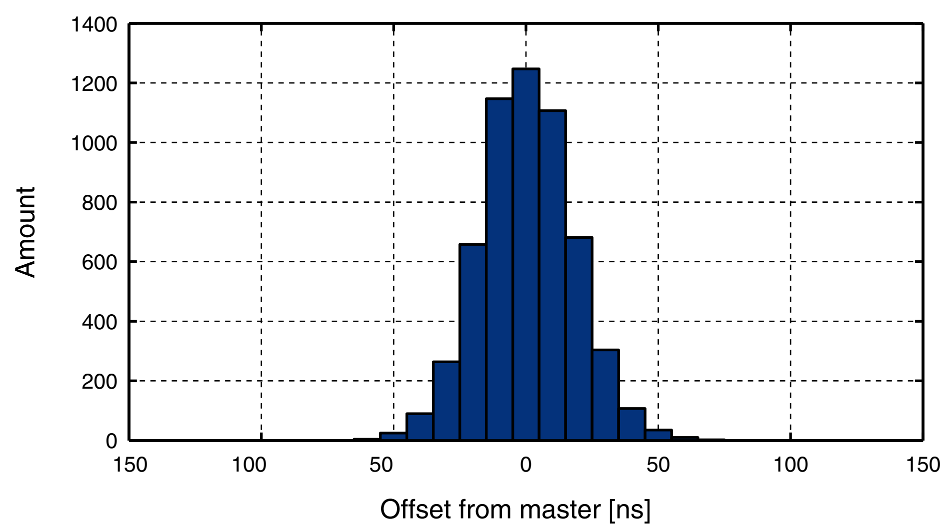


Figure 22. Precision reached using the external oscillator on the STM3210C-EVAL, the synchronization interval has been set to 1 second



The second test has been done again with the STM3210C-EVAL board connected with the precise PTPv2 master clock device, *Figure 20*. The synchronization interval has been set to 0.125 second. In *Figure 23*, there is a histogram showing the measured offset from the master. The offset measured for this hardware set-up shows minimum and maximum offset from -130 ns to 10 ns, average -17 ns. These variations depend again on the default crystal quality used on the STM3210C-EVAL board. The second part of this test has been done with modified STM3210C-EVAL board using external crystal oscillator connected to the X1 pin of the MCU. The on-board crystal has been removed. In *Figure 24* there is a histogram of the measured offset from the master. The offset measured for this hardware set-up shows the minimum and maximum offset from -40 ns to 20 ns, average -8 ns. Because the oscillator drifts only by the temperature changes and the frequency does not change rapidly, the synchronization with master is better compared to the use of the default crystal and the built in oscillator.

Figure 23. Precision reached using the default crystal and built in oscillator, the synchronization interval has been set to 0.125 second

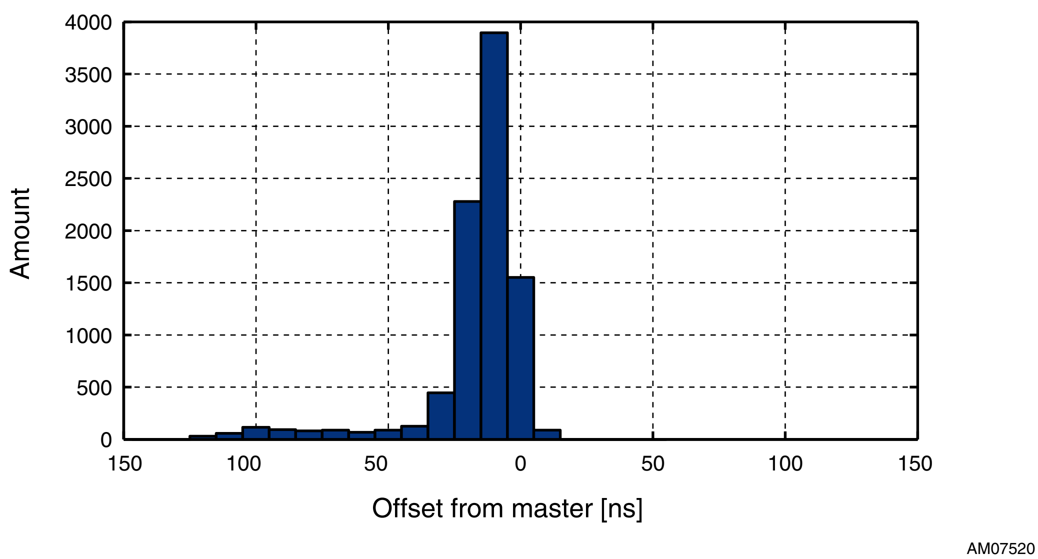
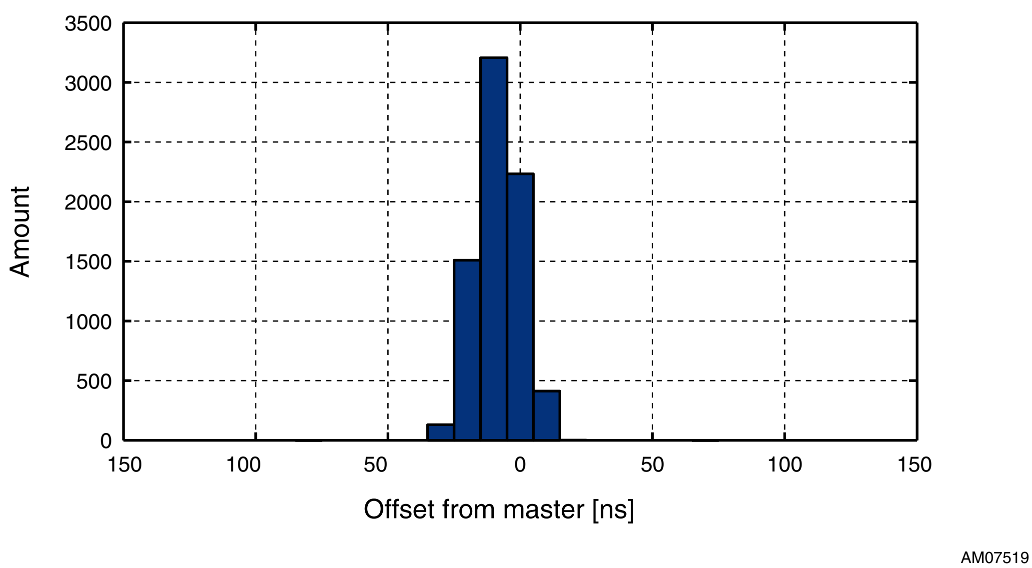


Figure 24. Precision reached using the external oscillator on the STM3210C-EVAL, the synchronization interval has been set to 0.125 second



7 Conclusion

This application note describes the STM32F107 SW project implementing the PTPd (precision time protocol) stack in connection with the lwIP TCP/IP stack implementation as described in AN3102. The application note describes the capability of the STM32F107 microcontroller to generate the time stamps of the incoming and outgoing packets and the capability to generate the precise trigger events. Step by step procedure is described making it easier to understand the necessary HW configuration set-up, changes which had to be applied to the original lwIP stack and to the original PTPd stack. The synchronization result is highly dependent on the application hardware. Selection of a deterministic Ethernet PHY device, the crystal or oscillator unit, are the key factors for the final synchronization accuracy of the slave device to the precise master. STM32 F-2 PTPd implementation is planned as a separate application note.

8 References

1. “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems,” IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002), vol., no., pp.c1-269, July 24 2008.
2. AN3102 - lwIP TCP/IP stack demonstration for STM32F107xx connectivity line microcontrollers, www.st.com.
3. RM0008 - Reference manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM-based 32-bit MCUs, www.st.com.
4. K. Correll and N. Barendt, “Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol“, in Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2006.
5. J. Breuer, “Hardware-Assisted IEEE 1588 Implementation Using the STM32 Connectivity Line Processor,” In POSTER 2010 - Proceedings of the 14th International Conference on Electrical Engineering [CD-ROM]. Prague: CTU, Faculty of Electrical Engineering, 2010, ISBN 978-80-01-04544-2.
6. www.sics.se/~adam/lwip - lwIP is a light-weight implementation of the TCP/IP protocol suite, http://savannah.nongnu.org/projects/lwip/- lwIP project download side
7. http://ptpd.sourceforge.net - The PTP daemon (PTPd) website.
8. www.slax.org - portable Linux operating system, www.pendrivelinux.com - run Linux from a flash memory drive.
9. www.meinberg.de/english - Innovative Solutions for Time and Frequency Synchronization.
10. www.livecdlist.com – Linux LiveCD/LiveUSB overview.
11. www.wireshark.com – Ethernet packet analyzer

Special thanks to Czech Technical University in Prague, Czech Republic, Faculty of Electrical Engineering, Department of Measurement, namely to the project co-authors Ing. Jan Breuer and doc. Ing. Jaroslav Roztocil, CSc.

9 Revision history

Table 4. Document revision history

Date	Revision	Changes
11-Jul-2011	1	Initial release.