Intel® Ethernet Network Connection

Application Programming Interface (API)

for
Intel® Ethernet Network Connection GPY211 (GPY211B1VC)
Intel® Ethernet Network Connection GPY212 (GPY212B1VC)
Intel® Ethernet Network Connection GPY215 (GPY215B1VI)
Intel® Ethernet Network Connection GPY115 (GPY115B1VI)

Programmer’s Guide

Intel Confidential

Revision 3.0, 2020-04-24
Reference ID 617809

Legal Notice

Revision History

Current: **Revision 3.0, 2020-04-24**
Previous: **Revision 2.0, 2019-12-09**

Page	Major changes since previous revision
All	Updated GPY API version to V2.5.1.
5	**Chapter 1**, **Introduction**: Overview updated.
11	**Chapter 2.4**, **Media Access Control Security (MACsec)**: Added MACsec feature and description.
18	**Chapter 2.5**, **Precision Timing Protocol**: Added PTP feature and description.
23	**Chapter 2.6**, **Synchronous Ethernet**: Added SyncE feature and description.
26	**Chapter 2.8**, **Thermal Management**: Added thermal management description.

Table of Contents

1 Introduction

This document provides an introduction and functional description of the Intel® Ethernet Network Connection Application Programming Interface (GPY API) for the following devices:

- Intel® Ethernet Network Connection GPY211 (GPY211B1VC)
- Intel® Ethernet Network Connection GPY212 (GPY212B1VC)
- Intel® Ethernet Network Connection GPY215 (GPY215B1VI)
- Intel® Ethernet Network Connection GPY115 (GPY115B1VI)

The devices listed above are Ethernet PHYs supporting the 10BASE-T, 100BASE-T, 1000BASE-T and 2.5GBASE-T modes of the IEEE802.3 standard [6].

The GPY API code is executed on the MAC Host processor connected to the Ethernet PHY. Ethernet PHYs are controlled over the MDIO interface and the IEEE 802.3 standard compliant functionality is usually controlled by the network stack and the MDIO driver of the operating system executed on the Host processor.

The GPY API allows to make use of additional functionality, which is not provided by the operating system. The GPY API is provided in source code, to speed up integration of the GPY device and simplify configuration. The license under which the GPY API can be used is provided in the file "LICENSE", which is delivered with the source code. The GPY API can be downloaded as a zip file from the Intel Customer portal.

The GPY API is intended to be used in user space, but can be used also in kernel space with minor adaptations and based on customer requirements.

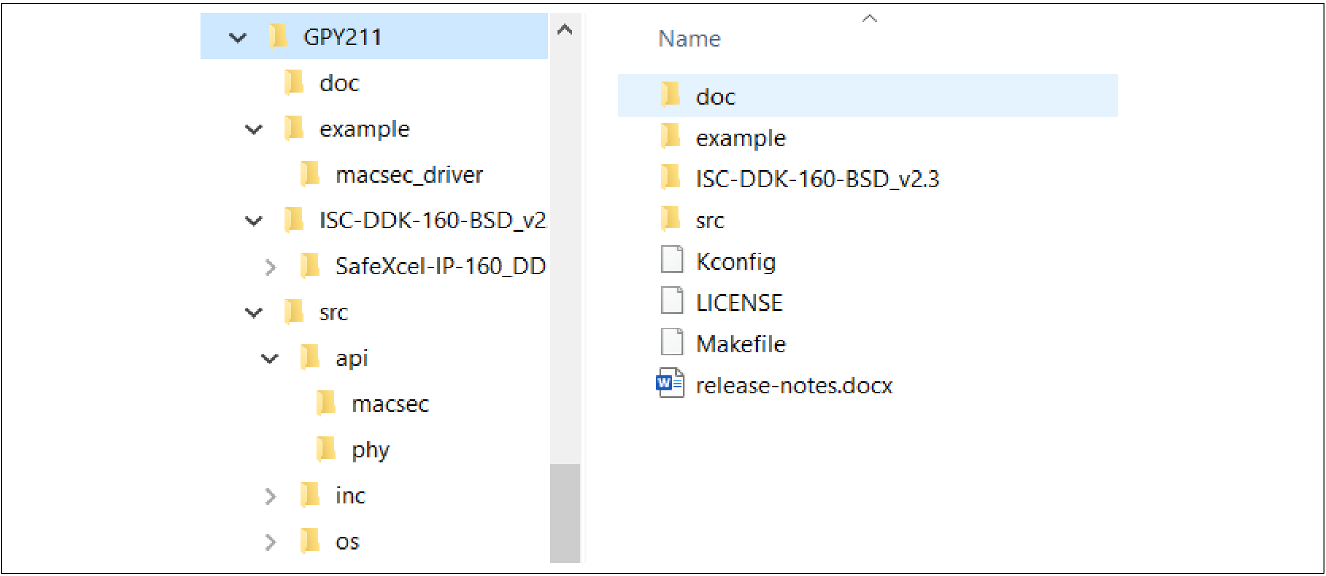
This Programmer's Guide applies to GPY API V2.5.1 and later versions.

1.1 Reference Documentation

To enable the user to generate the most recent reference of the GPY API related functions and data structures, the reference documentation is provided in the source code using Doxygen-formatted comments. Reference documentation in HTML format can be created by using the Doxygen tool. It can be downloaded for free from "**http://www.doxygen.nl**".

To obtain correct documentation, the Doxygen configuration file, which is provided with the source files of GPY API under <library_name>/doc/Doxyfile shall be used to create html files. Other output formats can also be created, if the configuration file is adapted. A Microsoft* Compiled HTML Help (chm) file will be distributed with the GPY API source code.

For GPYAPI description details, please refer to file as shown in **Figure 1**.


GPY211
doc
example
macsec_driver
ISC-DDK-160-BSD_v2
SafeXcel-IP-160_DD
src
api
macsec
phy
inc
os

Name
doc
example
ISC-DDK-160-BSD_v2.3
src
Kconfig
LICENSE
Makefile
release-notes.docx

Figure 1 File Path

1.2 Operating System Specific Functionality

The following functions are wrappers to operating system specific function calls:

- Memory allocation: malloc()
- Memory free: free()
- Micro-second delay: udelay()

2 Functional Description

2.1 General Information about GPY API Source Code Names

GPY API Source code is universal for all devices mentioned in **Chapter 1**, even if the used structure and function names are starting with "gpy2xx" or "gpy211".

2.2 Initialization

The most important data structure in GPY API is gpy211_device. It is used to represent a physical GPY device. This structure is divided into:

- MDIO access (provided by user)
- ID/versions
- TPI/Ethernet link status
- Other internal status

Before invocation of any API, the user needs to prepare gpy211_device, provide information for MDIO access, and invoke gpy2xx_init. The MDIO access members are listed below:

- void (*lock)(void *lock_data)
- void (*unlock)(void *lock_data)
- void *lock_data
- int (*mdiobus_read)(void *mdiobus_data, int addr, u32 regnum)
- int (*mdiobus_write)(void *mdiobus_data, int addr, u32 regnum, u16 val)
- void *mdiobus_data
- int phy_addr

For initialization of the GPY API, the user space application must provide the function pointers to the operating system specific MDIO bus access routines *mdiobus_read and *mdiobus_write and *mdiobus_data and the address of the PHY in phy_addr. The PHY address is the MDIO bus address of the PHY, which is usually configured by pin-strap.

The implementation of mdiobus_read and mdiobus_write and mdiobus_data must be able to handle both Clause 22 and Clause 45 access.

```
int (*mdiobus_read)(void *mdiobus_data, int addr, u32 regnum);
int (*mdiobus_write)(void *mdiobus_data, int addr, u32 regnum, u16 val);
void *mdiobus_data;
int phy_addr;
```

If the GPY API and other modules in the operating system concurrently access the MDIO bus, the functions lock() and unlock() and the pointer lock_data have to be provided to ensure that the MDIO access of the GPY API is thread safe. However, this is optional. If it is not provided, no locking is performed during MDIO access.

```
void (*lock)(void *lock_data);
void (*unlock)(void *lock_data);
void *lock_data;
```

An example of usage is provided in file: gpy211_driver.c

2.2.1 Interrupt Handling

The GPY can be configured via pin-strap to pull down or pull up its external interrupt pin for a variety of conditions. Since the GPY API is a collection of routines for a user space application, the GPY API does not directly handle interrupts. In order to be informed about interrupts, the user space application can use the operating system and SoC specific driver for external interrupts and wait with a separate thread with a suitable mechanism like the select() call for the occurrence of an interrupt from the GPY. The application can then use the GPY API to find out the cause of the interrupt generation.

The interrupt handling and configuration is part of the SoC SW and is OS specific.

An alternative way to monitor the PHY status is to periodically poll the PHY status register.

2.3 Field Firmware Upgrade (FFU)

The GPY executes firmware, which is stored in the chip internal memory. But it can also fetch its program from an external low-cost serial flash memory. The flash content can be upgraded in the field. This feature is called Field Firmware Upgrade (FFU).

The Field Firmware Upgrade allows feature and functional enhancements of the GPY in the field. This is especially useful to mitigate interoperability problems with new devices, which are not available in the field today.

Initially, the GPY is provided with a permanent on-chip firmware image in a one-time programmable memory (OTP). With a serial flash memory connected to the GPY's SPI interface, a new firmware image can be downloaded over the MDIO and the GPY to the flash. The GPY will fetch the upgraded firmware from this flash after a reboot.

For security reasons, the GPY will only accept firmware images that have been electronically signed by Intel.

If a flash image cannot be authenticated by the GPY, or a flash image download is aborted or fails, the GPY will default to run from the internal firmware image in OTP.

The FFU feature is only supported, if the GPY API is used as a user space application. A signed firmware binary for the GPY will be provided on the Intel Customer portal, when major feature or functional enhancements are available. The firmware version of the binary is contained in the filename as well as in the firmware binary code at offset address 0x11FE-0x11FF.

The GPY API provides 4 functions for easy integration of the FFU feature into the host software.

- gpy2xx_fw_frw_init – Starts the upgrade process
- gpy2xx_fw_fwr_page – Writes a 256 byte page. Must be called until all bytes have been written
- gpy2xx_fw_frw_uninit – Completes the upgrade process
- gpy2xx_fw_fwr_status – Polls the completion status of flash init and write page operations during firmware upgrade (optional)

2.3.1 Application Tasks

The application on the SoC is responsible for the following tasks:

- Image deployment to the SoC: A new GPY firmware binary is expected to be part of a larger image for update of the complete SoC firmware. How the new GPY firmware is made available on the SoC is outside the scope of this document.
- Verification of successful download: After a firmware download, the SoC should verify that the new image has been applied successfully on the GPY. This can be done by reading out the firmware version via MDIO after reboot of the GPY. If this firmware version number is identical to the version number of the downloaded image, the download was successful. If it is different, the GPY is operating from the internal OTP fallback image.
- Compliance to the programming sequence: The SoC must perform the flash programming according to the sequence described in **Figure 2** - FFU Flow Diagram using the GPY APIs provided by Intel.

2.3.2 Upgrade Steps

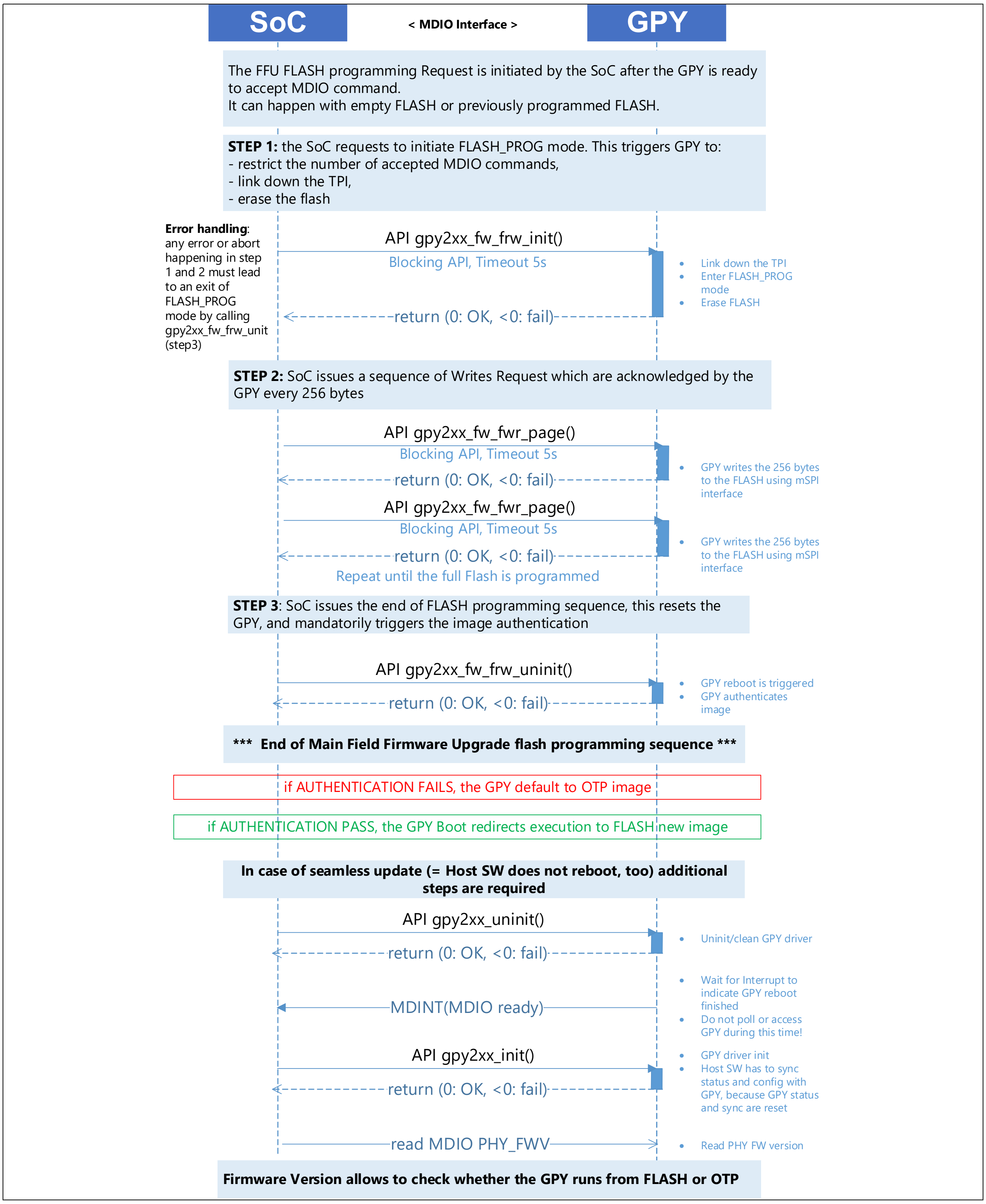

SoC
< MDIO Interface >
GPY
The FFU FLASH programming Request is initiated by the SoC after the GPY is ready to accept MDIO command.
It can happen with empty FLASH or previously programmed FLASH.
STEP 1: the SoC requests to initiate FLASH_PROG mode. This triggers GPY to:
- restrict the number of accepted MDIO commands,
- link down the TPI,
- erase the flash
Error handling: any error or abort happening in step 1 and 2 must lead to an exit of FLASH_PROG mode by calling gpy2xx_fw_frw_unit (step3)
API gpy2xx_fw_frw_init()
Blocking API, Timeout 5s
return (0: OK, <0: fail)
Link down the TPI
Enter FLASH_PROG mode
Erase FLASH
STEP 2: SoC issues a sequence of Writes Request which are acknowledged by the GPY every 256 bytes
API gpy2xx_fw_fwr_page()
Blocking API, Timeout 5s
return (0: OK, <0: fail)
GPY writes the 256 bytes to the FLASH using mSPI interface
API gpy2xx_fw_fwr_page()
Blocking API, Timeout 5s
return (0: OK, <0: fail)
GPY writes the 256 bytes to the FLASH using mSPI interface
Repeat until the full Flash is programmed
STEP 3: SoC issues the end of FLASH programming sequence, this resets the GPY, and mandatorily triggers the image authentication
API gpy2xx_fw_frw_uninit()
return (0: OK, <0: fail)
GPY reboot is triggered
GPY authenticates image
*** End of Main Field Firmware Upgrade flash programming sequence ***
if AUTHENTICATION FAILS, the GPY default to OTP image
if AUTHENTICATION PASS, the GPY Boot redirects execution to FLASH new image
In case of seamless update (= Host SW does not reboot, too) additional steps are required
API gpy2xx_uninit()
return (0: OK, <0: fail)
Uninit/clean GPY driver
MDINT(MDIO ready)
Wait for Interrupt to indicate GPY reboot finished
Do not poll or access GPY during this time!
API gpy2xx_init()
return (0: OK, <0: fail)
GPY driver init
Host SW has to sync status and config with GPY, because GPY status and sync are reset
read MDIO PHY_FWV
Read PHY FW version
Firmware Version allows to check whether the GPY runs from FLASH or OTP

Figure 2 FFU Flow Diagram

- **Step 1**:
  The SoC calls gpy2xx_fw_fwr_init(). This function requests the GPY to enter the flash programming (FLASH_PROG) mode. The GPY subsequently brings down the twisted pair interface (TPI) and restricts the accepted MDIO commands to the ones required for flash programming. The GPY erases the flash memory. The function polls every 500 µs to determine if the flash erasing process has completed. The function returns after the flash memory has been erased or after a 5 s timeout (timeout is configurable).
- **Step 2**:
  The SoC calls gpy2xx_fw_fwr_page() and writes the data to program the flash memory in blocks of 256 bytes. This function checks the completion of each 256 byte block write every 100 µs.
- **Step 3**:
  The SoC calls the gpy2xx_fw_fwr_uinit() function. This function signals the completion of the programming sequence to the GPY and triggers a reset with the MDIO command SDT_CTRL_RST. The GPY then executes the required reset and boot process including authentication of the firmware image.

The functions gpy2xx_fw_fwr_init() and gpy2xx_fw_fwr_page() are called with a timeout as a parameter. They behave differently, when the timeout is zero and when the timeout is non-zero.

When the timeout is zero, the functions will return immediately and not poll the PHY for completion of the flash erasure or program operation. In this case, the user application must regularly check the completion of the flash erasure or program operation by calling gpy2xx_fw_fwr_status.

When the timeout parameter is larger than zero, the function will poll the GPY regularly every 500 µs to check, if the flash erasure has been completed or every 100 µs to check, if the flash block programming has been completed. The function will return, when the flash operation is completed, or with an error code, when the timeout expired. A typical timeout is 5 s.

Note: The time required for the flash programming depends on the selected MDIO speed. For a 5 MHz MDIO clock rate, the flash update finishes typically in 20 s.

2.3.3 Security Precautions

The following features are implemented in the GPY chip for security purposes.

- The GPY uses the factory default firmware in its OTP, if the authentication of a flash image fails during boot up.
- If the FFU is started by the SoC, the GPY must be reset using gpy2xx_fw_frw_uninit() as the last step of the FFU process.
- The GPY is the only chip connected to the flash memory via the GPY's master SPI. The actual flash programming is performed by the GPY, triggered by the API commands of the GPY API issued via MDIO.
- The GPY prevents access to MDIO registers, which are not related to the programming of the flash, once the FFU programming sequence is started. FLASH PROGRAM Mode can be exited by issuing a STD_CTRL_RST (gpy2xx_fw_frw_uninit()) or HRSTN.
- The start of the flash programming triggers a TPI (Twisted Pair Interface) link down and ends normal operation until the GPY is reset.
- The flash upgrade only comes into effect after a GPY hardware reset (HRSTN pin) or by MDIO command STD_CTRL_RST (gpy2xx_fw_frw_uninit()).

2.3.4 Handling of Programming Failures or Exceptions

The SoC is responsible for handling failures in the flash programming sequence. The failure handling should be in accordance with the following principles:

- The GPY boot sequence always performs flash memory presence detection and authentication of the firmware image.
- The GPY never executes a flash firmware image if it has not been authenticated after boot.
- If the authentication fails, the GPY will execute the default build firmware in the OTP.
- If the flash programming is interrupted, the SoC must reset the GPY by driving the hardware signal HRSTN or with the help of the MDIO command STD_CTRL_RST.
- The SoC can abort the programming sequence by issuing a reset prior to completion of the flash programming. In this case the reboot will detect an authentication FAIL and the GPY will fetch the default build image from the OTP.
- It is the responsibility of the SoC to verify which firmware version is running and check that the firmware version is as expected (either OTP firmware version, or flash firmware version).
- If the flash content is corrupted, the GPY authentication at boot fails. The SoC can detect this by reading the firmware version. If this read indicates that the boot happened from OTP, the SoC can re-initiate a flash programming sequence.

2.4 Media Access Control Security (MACsec)

Media Access Control Security (MACsec) is an 802.1AE IEEE industry-standard security technology that provides secure communication for all traffic on Ethernet links. MACsec allows authorized systems that attach to and interconnect Local Area Networks (LANs) in a network to maintain confidentiality of transmitted data and to take measures against frames transmitted or modified by unauthorized devices.

MACsec facilitates:

- Maintenance of correct network connectivity and services.
- Isolation of denial of service attacks.
- Localization of any source of network communication to the LAN of origin.
- The construction of public networks, offering service to unrelated or possibly mutually suspicious customers, using shared LAN infrastructures.
- Secure communication between organizations, using a LAN for transmission.
- Incremental and non-disruptive deployment, protecting the most vulnerable network components.

2.4.1 Functional Description

MACsec provides industry-standard security through the use of secured point-to-point Ethernet links. The point-to-point links are secured after matching security keys which are exchanged and verified between the interfaces at each end of the point-to-point Ethernet link.

IEEE 802.1X-2010 defines a companion protocol, MACsec Key Agreement (MKA), which provides key exchange and allows mutual authentication of nodes that want to take part in a MACsec connectivity association. On Linux machines, this is implemented in “wpa_supplicant”. “wpa_supplicant” uses some authentication token (a pre-shared key, or a username/password pair) to establish a session with the authentication server, which can be a switch or a Linux host running “hostapd”. After authentication, keys are generated and exchanged (over an encrypted channel) and are used to configure MACsec secured link.

2.4.1.1 Architecture of A MACsec Network

Figure 3 shows two stations, A and B, connected to a point-to-point LAN that provides insecure bi-directional connectivity.

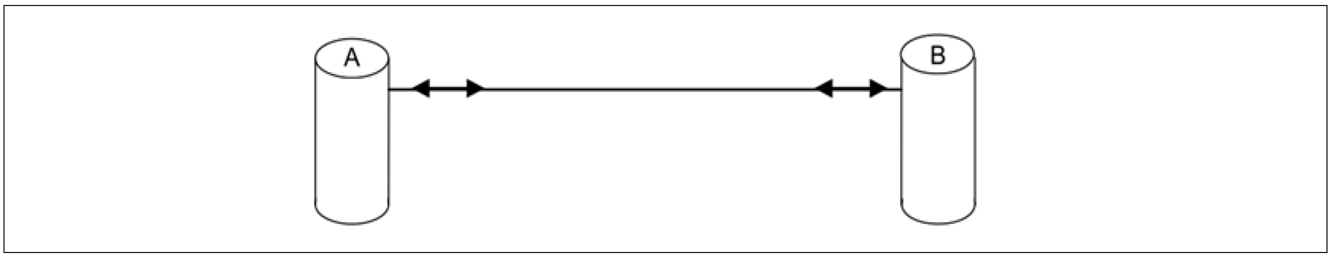


Figure 3 Point - to - Point LAN

Figure 4 depicts the secure Connectivity Association (CA) created by MACsec Key Agreement (MKA) following mutual authentication and authorization of A and B.

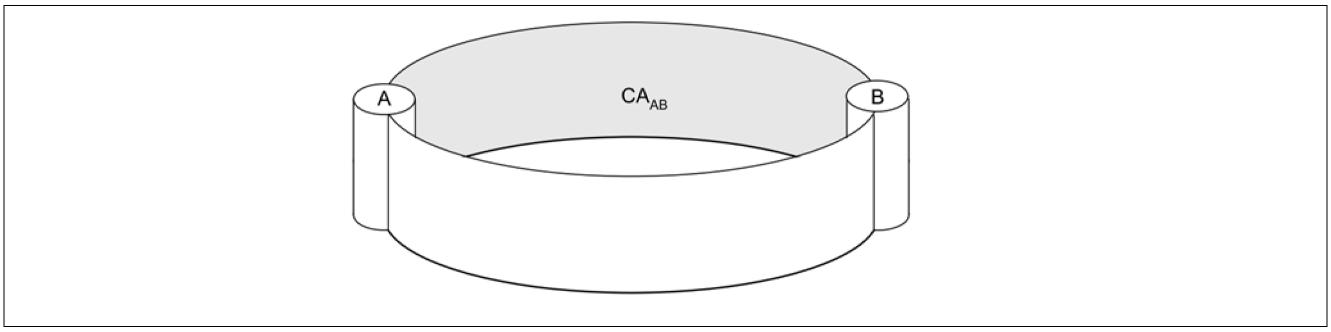


Figure 4 Connectivity Association by MKA

Figure 5 shows the two unidirectional Secure Channels (SCs) that support the CA.

This is derived from the 802.1AE-2006 Secure Provision chapter [6] to illustrate the relations between CA, SecY, SC, SA. Here it represents SecY instead of an exact computer station (there may be multiple SecYs in one computer).

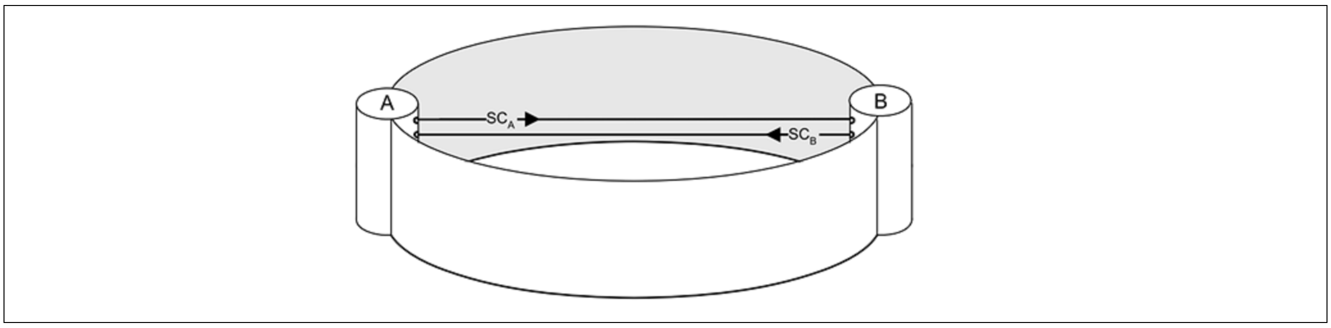


Figure 5 Connectivity Association by MKA

A and B are individual MAC Security Entities (SecYs).

A transmit frame to B over SC_A supported by an overlapped sequence of Security Associations (SAs). Each SA uses a fresh Secure Association Key (SAK) to provide the MACsec service guarantees and security services for a sequence of transmitted frames.

2.4.2 MACsec Support in GPY

MACsec is security enhancement feature normally instantiated at the MAC layer. GPY provides this additional feature for hosts which does not have MACsec in its layer-2 engine (MAC) or wants to achieve wire speed MACsec handling with hardware offload.

GPY has one instance of MACsec engine. It complies with IEEE 802.1AE and supports both 128-bit and 256-bit AES-GCM operations, 16 Secure Channels (SCs), as well as 32 Secure Associations (SAs) in each direction. **Figure 6** shows the overview of MACsec engine integrated inside GPY.

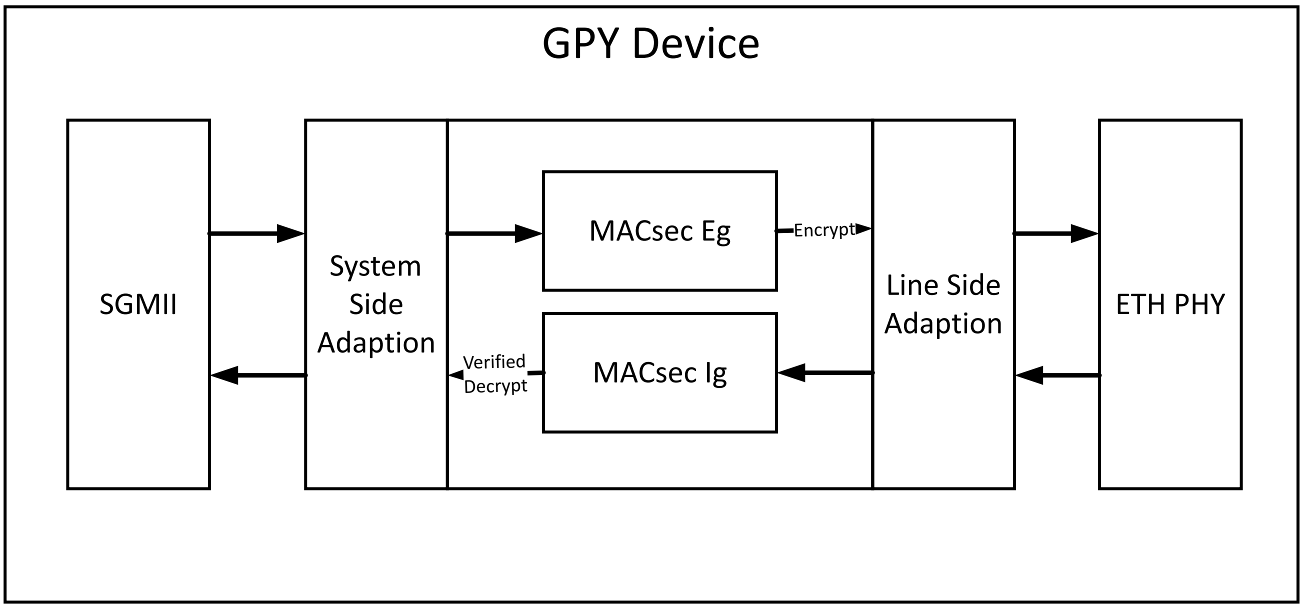


Figure 6 MACsec Engine in GPY

2.4.2.1 Input Classification Engine

The input classification engine inspects the received frame data and performs the following functions:

- Control Frame Classification
- VLAN Tag Detection – Programmable function to detect VLAN tags and extract information required for further classification as shown in **Figure 7**
- MACsec Tag Detection as shown in **Figure 8**
- Non-matching Flow Control
- SA Matching and Flow Control
- SA Matching/Non-matching Multiplexer

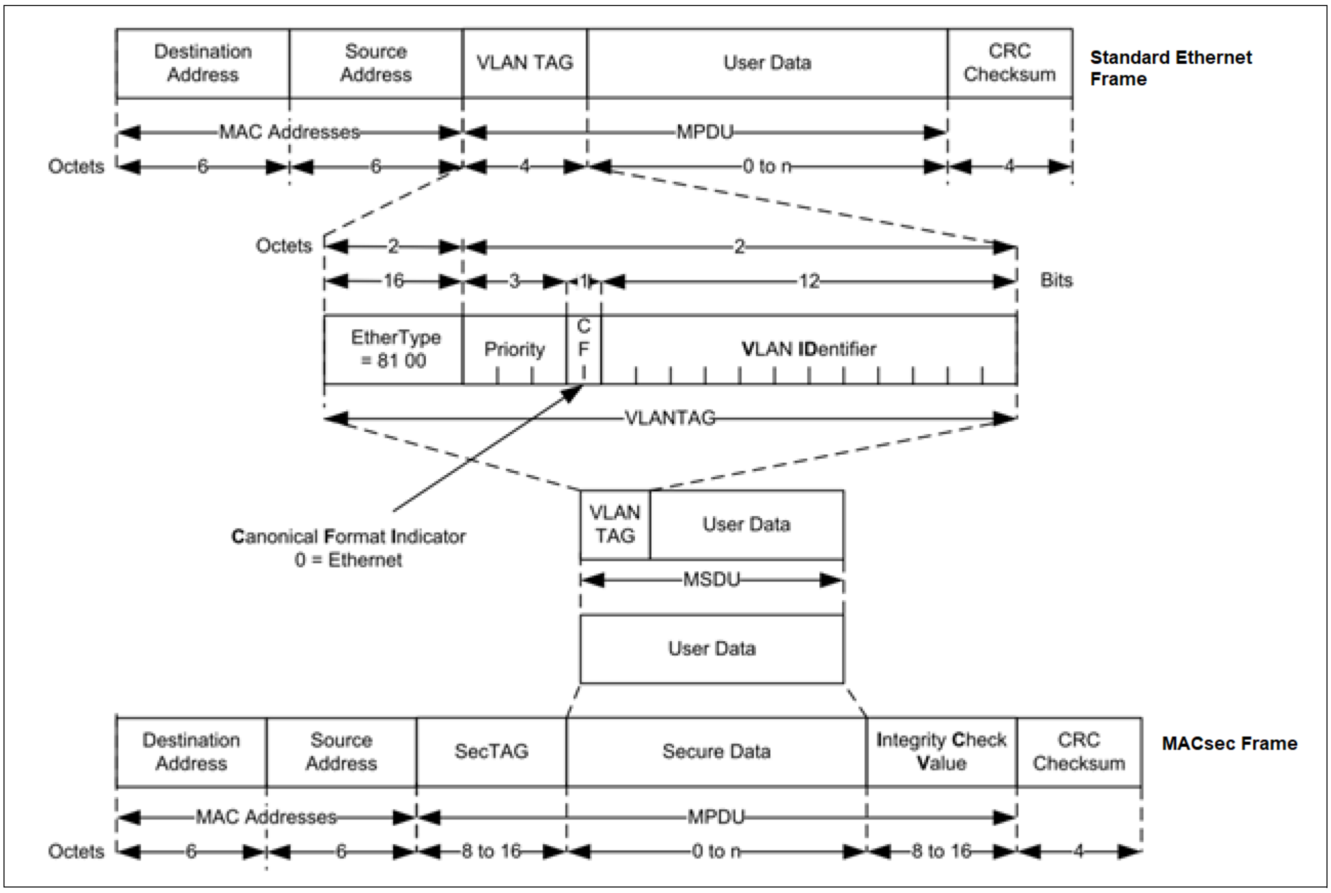


Figure 7 VLAN Tag Detection

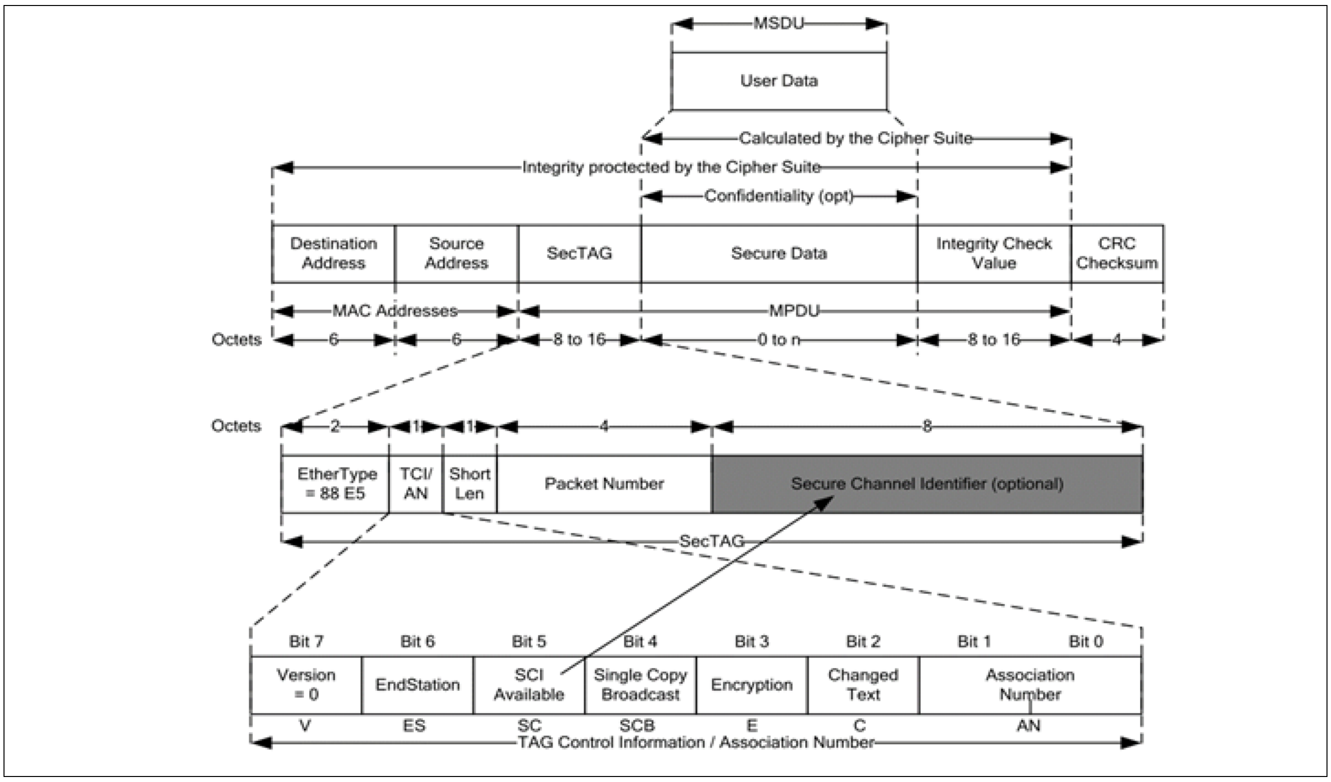


Figure 8 MACsec Tag Detection

2.4.2.2 Packet Processing Engine

This is the engine to do encapsulation/decapsulation processing. It is capable of autonomously encapsulating or decapsulating a MACsec frame, including header insertion and removal. It does not perform MACsec header parsing but relies on Input Classification Engine and SA flow configurations to provide the processing “token” that tells it how to process the received frame. The Packet Processing Engine checks the lifetime of the transmit SA and if it expires, it provides the next SA to the corresponding flow control register of the Input Classification Engine.

2.4.2.3 Consistency Checking Engine

The Consistency Checking Engine checks the contents of a frame at the output stage against a set of programmable rules for consistency. A programmable per-rule priority level resolves any overlap between these rules

2.4.2.4 Output Post-processing Engine

The Output Post-processing Engine checks the classification and processing results against a fixed set of MACsec compliance rules, resulting in a drop decision if the rules are violated. Additionally, it performs programmable MTU checking on the output frame with individual global and per-VLAN-user-priority MTU settings. It combines these internal decisions made by the Classification and Consistency Checking engines into a final pass/drop decision.

2.4.2.5 Statistics Update Engine

This engine takes care of the actual statistics counter updating as instructed by the Output Post-process engine.

2.4.3 MACsec Support in GPY API

The MACsec solution needs implementation of MACsec Key Agreement (MKA), MACsec Secure Entity (SecY), Secure Channel (SC), etc. **Figure 9** is an example of Linux implementation with 2 stations: Linux Host and a Switch.

1. wpa_supplicant on the host communicates with the MKA agent on the switch’s MACsec port.
2. After the host and the switch have mutually authenticated each other in step 1, they both configure a pair of Secure Channels (SCs) with matching identifiers (SCIs) in step 2.
3. The switch will then generate a key for each direction (step 3). These keys will be used to encrypt and decrypt the actual traffic.
4. In step 4, Secure Associations (SAs) using these keys are configured on both host and switch.
5. wpa_supplicant translates the information derived through MKA and configures the Linux Kernel’s MACsec implementation.
6. From that point on, the kernel sends packets protected by MACsec on the “macsec0” interface, a separate network device dedicated to encrypted traffic (Controlled Port).
7. Step 3 and 4 are later repeated (as many times as SC persist), while wpa_supplicant keeps running, to transition to a new key when the current key expires.

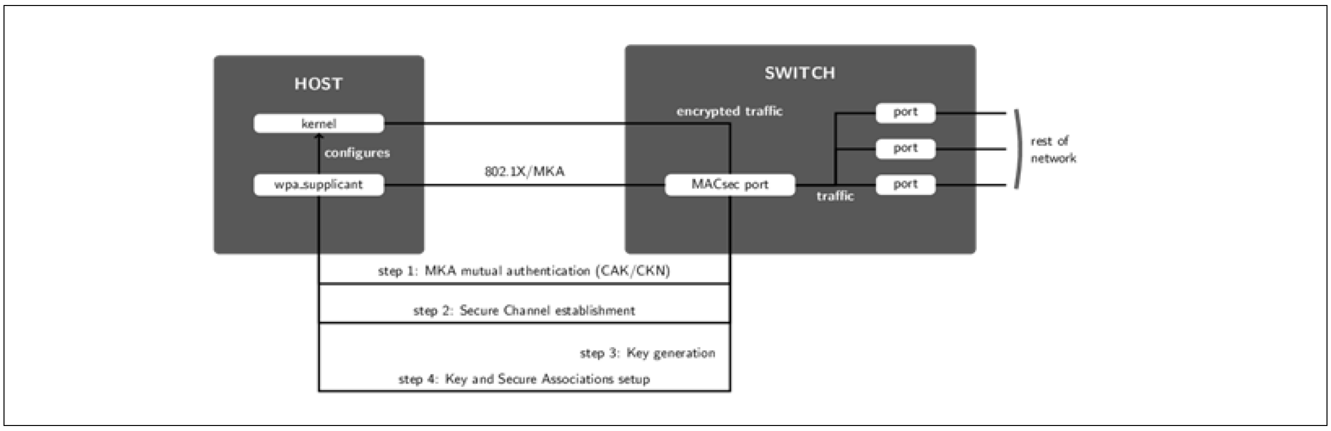


Figure 9 Linux Host - Switch

Linux Kernel abstracts the MACsec implementation to following operations:

- Add SecY
- Delete SecY
- Update SecY
- Get SecY (and SCs)
- Add Receive SC
- Delete Receive SC
- Update Receive SC
- Add Transmit SA
- Delete Transmit SA
- Update Transmit SA
- Add Receive SA
- Delete Receive SA
- Update Receive SA

The GPY API provides functions to support implementation of these abstracted operations.

2.4.4 MACsec API Categories

Table 1 lists the categories of MACsec APIs with brief description and possible mapping to Linux MACsec operation.

Table 1 MACsec API Categories

Category	Briefing	Possible Map to Linux MACsec Operation
Initialization APIs	Group of functional APIs for MACsec initialization and cleanup.	General Initialization
Miscellaneous Control config APIs	Group of functional APIs for configuring misc controls.	
Secure Association Matching (SAM) No-Match Control config APIs	Group of functional APIs for configuring SAM Non-Match pkt action.	
Control Packet config APIs	Group of functional APIs for control packet classification.	
Output Post Processor config	Group of structures for Output Post Processor config storage.	
Crypto Core config	Group of structures for Crypto Core config storage.	
Interrupt Controller config	Group of structures for Interrupt Controller config storage.	
Counters Control config APIs	Group of functional APIs for controlling counters.	Add SecY Update SecY Delete SecY
Ingress Consistency Check (ICC) config APIs	Group of functional APIs for configuring ingress consistency check.	
Statistics get APIs	Group of functional APIs for getting statistics.	Get SecY
Secure Association (SA) Expiry Summary config APIs	Group of functional APIs for configuring SA expiry summary.	
Transform Record config APIs	Group of functional APIs for configuring transform record (SA).	Add Transmit SA Delete Transmit SA Update Transmit SA Add Receive SA Delete Receive SA Update Receive SA
Secure Association Matching (SAM) rule params config APIs	Group of functional APIs for configuring SA matching rule params.	
Secure Association Matching (SAM) Flow Control Action (FCA) config APIs	Group of functional APIs for configuring SAM flow control action.	
Secure Association Matching (SAM) Non-Match Flow Control Action (FCA) config APIs	Group of functional APIs for configuring SAM Non-Match flow control action.	
Secure Association Matching (SAM) Entry Enable config APIs	Group of functional APIs for configuring SAM entry enable flags.	
Secure Association Matching (SAM) Entry Enable Control config APIs	Group of functional APIs for configuring SAM entry enable control.	

2.4.5 Example Linux Driver

In delivery package, there is one Linux driver example on how to implement MACsec offload with GPY API. It is located in directory “\example\macsec_driver” as shown in Figure 1.

2.5 Precision Timing Protocol

The GPY provides support for Precision Time Protocol (PTP) which is used to precisely synchronize clocks at the system level. PTP is used to synchronize the time of day (ToD) of slave clocks to a master clock over Ethernet networks. It is standardized in IEEE 1588-2002 and as in second revision (1588v2) in IEEE 1588-2008. PTP implementation is not a pure PHY topic but rather a system topic, which also requires the implementation of the PTP protocol in software. The GPY has build in hardware support to provide high precision and easy to use PTP clock synchronization.

The GPY supports PTP with the following features:

- Implementation of a 64 bit counter, which counts with a granularity of 1.2 ns.
- Two 16 entry deep FIFOs with over flow and underflow support, one to store the timestamps and CRCs of outgoing packets and one to store the timestamps and CRCs of incoming packets.
- Timestamps are captured with high precision in transmitted and received PTP packets when the start of frame delimiter of a packet is detected.
- Timestamps can be generated triggered by the edge of an input signal to allow synchronous latching of the timestamp of the GPY's counter and the counter of an external master or slave clock to calculate counter differences. GPC1 or GPC2 can be selected as the input for the trigger signal.
- A recurring pulse signal can be generated either at the GPC1 or GPC2. The signal can be a Pulse Per Second (PPS) signal or a signal with another configurable frequency.

The GPY API provides below listed functions for easy integration of the PTP feature into the host software.

- gpy2xx_ptp_enable - Enables and configures the PTP (1588) function.
- gpy2xx_ptp_disable - Disable PTP (1588) function.
- gpy2xx_ptp_adjfreq - Adjusts the frequency of the hardware clock.
- gpy2xx_ptp_adjtime - Sets the shift time of the hardware clock.
- gpy2xx_ptp_settime - Sets the system time of the PHY.
- gpy2xx_ptp_gettime - Reads the current system time of the PHY.
- gpy2xx_ptp_getcfg - Gets the PTP (1588) TS configuration.
- gpy2xx_ptp_fifostat - Gets timestamp FIFO status.
- gpy2xx_ptp_resetfifo - Resets the FIFO status.
- gpy2xx_ptp_set_tsctrl - Configures IEEE 1588 Timestamp (TS) control settings.
- gpy2xx_ptp_set_ppsctrl - Configures IEEE 1588 Pulse Per Second (PPS) function settings.
- gpy2xx_ptp_set_ptoctrl - Configures IEEE 1588 PTP Timestamp Offload function settings.
- gpy2xx_ptp_getrxts - Gets Rx packet timestamp snapshot from FIFO.
- gpy2xx_ptp_gettxts - Gets Tx packet timestamp snapshot from FIFO.

2.5.1 Functional Description

The time stamp is inserted in a PTP event message. The PTP protocol is executed by the STA at the OSI layer above UDP/IP or MAC layer. The PTP protocol can choose 1-step or 2-step time stamping, both are supported by the GPY.

2.5.1.1 1-Step Timestamping

1-Step Timestamping is used to reduce the number of PTP messages. In this scheme, the GPY MAC inserts the time stamp in the sync message on the fly when it passes through the GPY MAC layer. Packet flow in the GPY to MAC is as described in **Figure 10**.

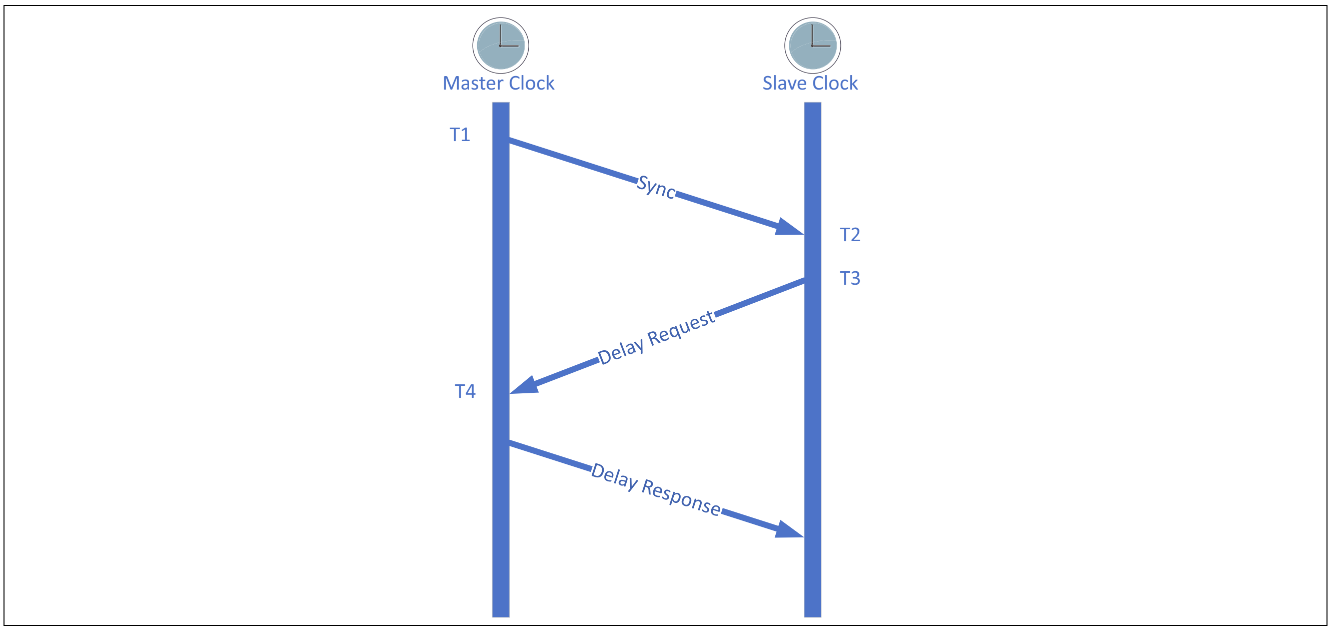


Figure 10 1-Step Timestamping

2.5.1.2 2-Step Timestamping

2-Step Timestamping scheme uses a Follow_Up message to carry the time stamp of the corresponding sync message. The time stamp is not inserted in the sync message on the fly when the packet is being transmitted, but later in the next PTP message. This scheme allows the GPY to perform the hardware assisted precise time stamping capture, using the PHY layer to precisely indicate when the packet Start-ofFrame Delimiter (SFD) symbol is sent out or received on the physical layer. The time stamp, together with the corresponding packet CRC is stored in a memory area on the GPY. The STA reads this time stamp using the MDIO interface. Packet flow in the GPY to MAC is as described in **Figure 11**.

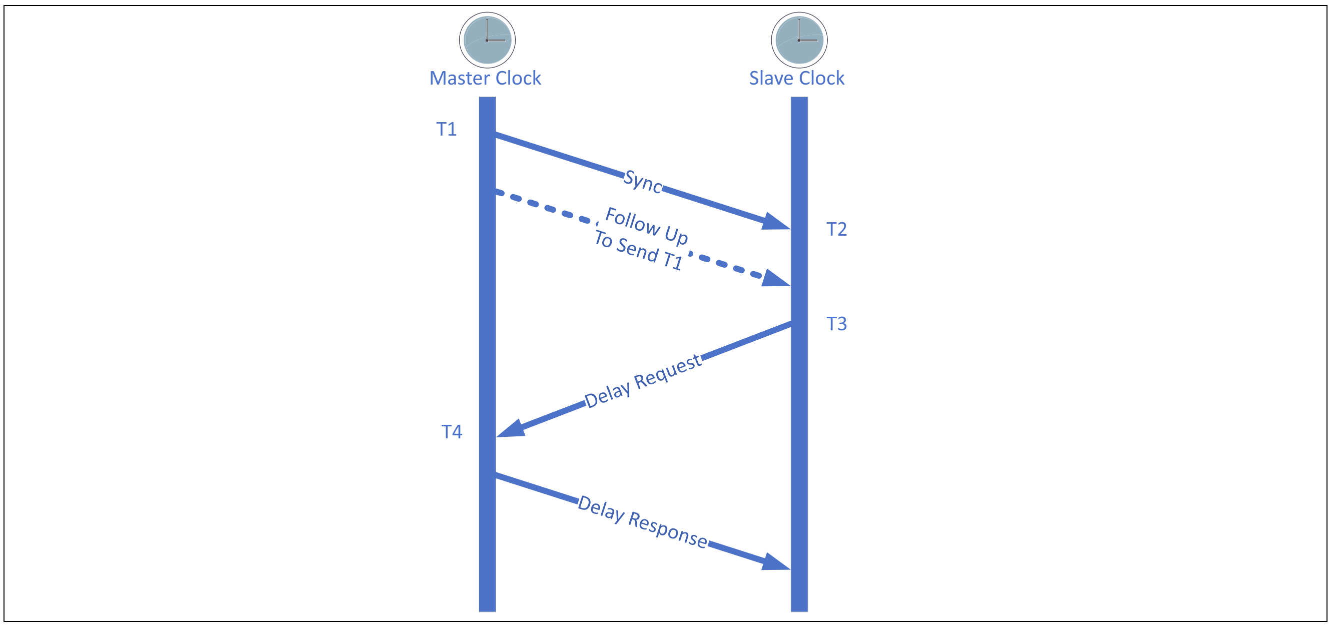


Figure 11 2-Step Timestamping

2.5.2 Typical System Application

The system level implementation of typical application scenario for Precision Timing Protocol is as shown in Figure 12. The objective is to synchronize the Slave Clock running on its own crystal to the Master Clock running on its own crystal over an Ethernet link or network.

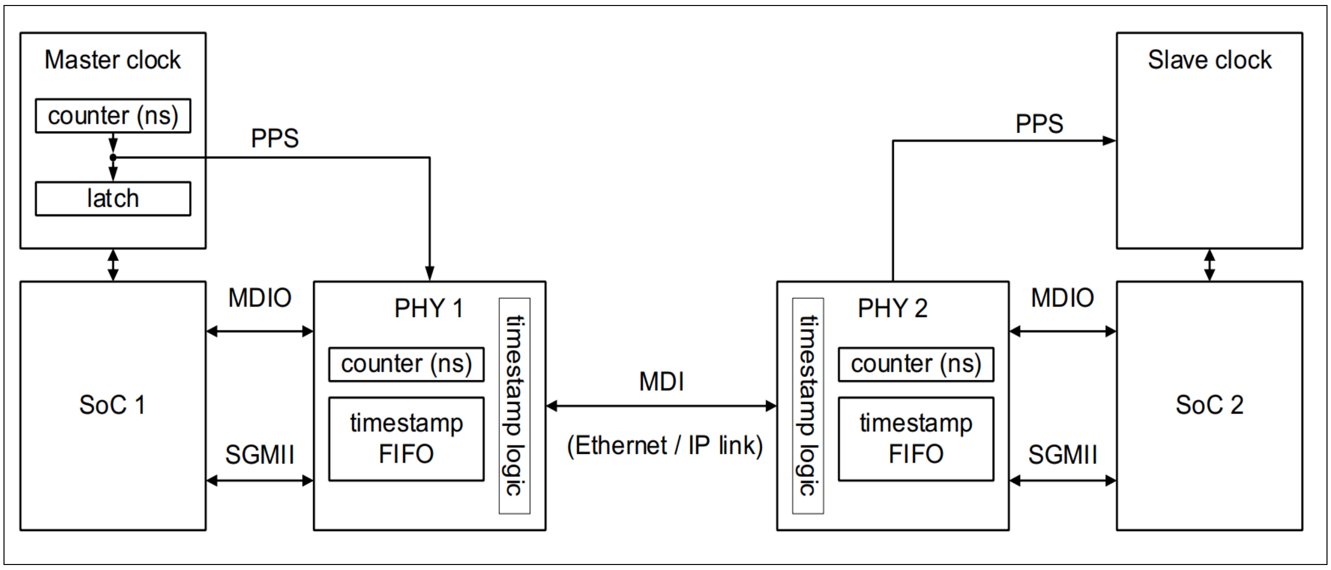


Figure 12 PTP System Application

Synchronization is done in the following sequence: The application on the SoC is responsible for the following tasks:

- Synchronization of PHY 1's Counter to the Master Clock Counter
- Synchronization of PHY's Counter to PHY 1's Counter
- Synchronization of the Slave Clock

2.5.2.1 Synchronization of PHY 1's Counter to the Master Clock Counter

In a first step, the counter of PHY 1 has to be set correctly. The SoC configures the master clock to generate a pulse per second (PPS) signal and stores the counter value in a latch with the rising edge of the PPS pulse. The SoC configures the PHY to store its counter value in a timestamp FIFO with the rising edge of the PPS pulse. After the PPS pulse has fired, SoC 1 reads out both timestamps and calculates the exact time difference between the master clock and PHY 1's clock. The SoC then programs the counter difference into PHY 1 and PHY 1 adds this difference to or subtracts the difference from the counter value.

PHY 1's counter is now precisely adjusted to the master clock's counter. During operation the SoC regularly repeats the procedure to adjust the counters. SoC 1 also adjusts PHY 1's counter clock frequency in order to minimize the frequency difference. Without adjusting the frequency, the difference between the master clock's counter and the PHY's counter would deviate significantly during clock adjustments. For a 20 ppm difference of PHY 1's crystal compared to the master clock, the deviation at the end of a 1 s interval would already be 20 µs.

2.5.2.2 Synchronization of PHY 2's Counter to PHY 1's Counter

SoC 1 and SoC 2 now exchange 4 PTP packets for one-step time-stamping or 5 PTP packets for 2 step PTP time-stamping. The PTP packets are recognized by the PHYs in transmit and receive direction and the timestamp of each PTP packet is stored in a FIFO together with the CRC of the packet. Very high accuracy is achieved, because the PHYs record the timestamps exactly when the start-of-frame delimiter of the Ethernet packet is detected.

After executing the PTP protocol, SoC 2 has a precise estimate of the time difference between PHY 1's counter and PHY 2's counter and can adjust PHY 2's counter exactly by the time difference. After this adjustment, the counters in the two PHYs are precisely synchronized. During operation, the procedure is repeated regularly and SoC 2 will use the frequency adjustment capability of PHY 2's counter to also minimize the counter difference before a new adjustment.

2.5.2.3 Synchronization of the Slave Clock

A similar procedure like the one in step 1 will be used, to synchronize the slave clock counter to PHY 2's counter.

2.5.3 IEEE 1588 Software

Figure 13 illustrates a typical 1588 software stack to run with GPY PHY device.

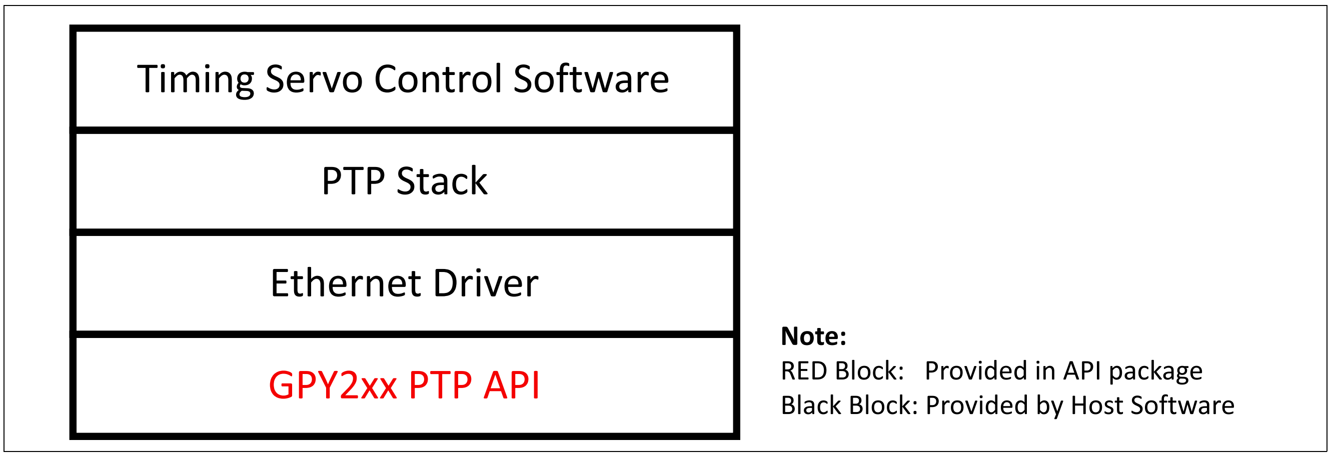

Timing Servo Control Software
PTP Stack
Ethernet Driver
GPY2xx PTP API
Note:
RED Block: Provided in API package
Black Block: Provided by Host Software

Figure 13 1588 Software Stack

Host needs to implement the timing servo control software, PTP stack and Ethernet driver. GPY API provides support for Ethernet driver to configure PTP capability in GPY, get timestamp, and most important adjust time and frequency. Linux, for example, provides framework to support PTP stack and LinuxPTP as timing servo control software.

Figure 14 is an example scenario where the slave clock is syncing to the master clock using the PTP protocol with help of GPY.

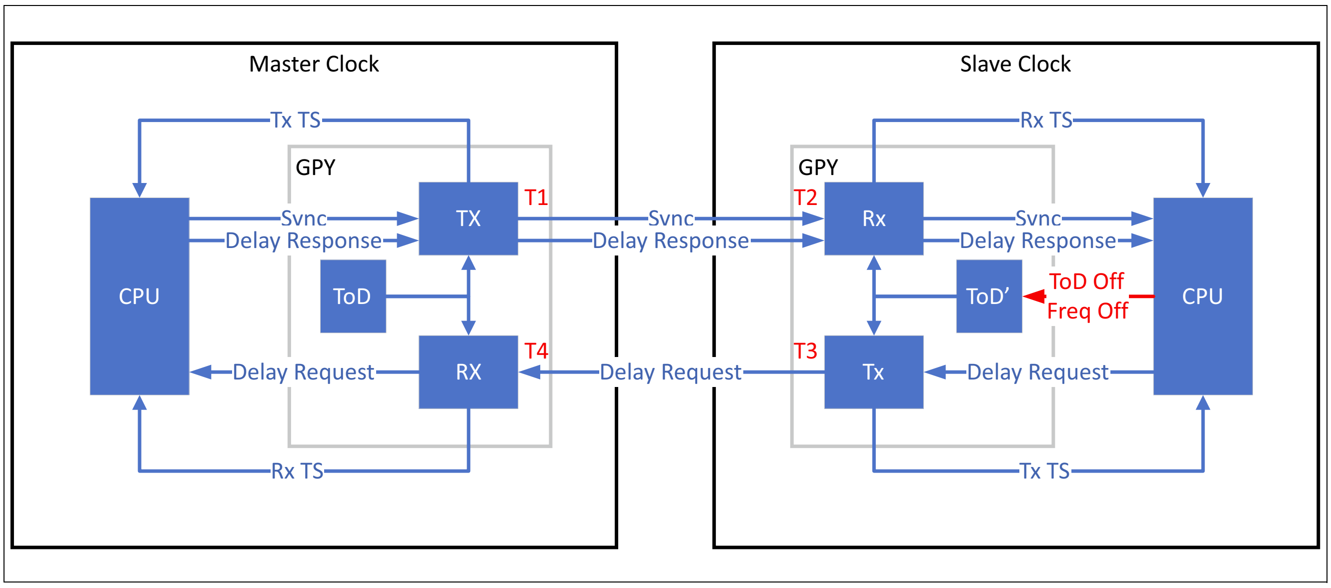


Figure 14 Master Slave 1588 PTP System

Following are the steps involved in the 1588 PTP system flow:

1. Master clock sends a Sync message which carry T1 (1-step) to all slaves.
2. Slave clock gets T1 from the Sync message and records T2 when receiving the Sync message.
3. Slave clock sends Delay Request to Master clock and records T3 when Delay Request is transmitted.
4. Master clock records T4 when receiving Delay Request and sends T4 to Slave clock with Delay Response.
5. Repeat steps 1 and 2, and record timestamps as T5, T6.
6. Software on Slave clock calculates (example):
   a) Mean path delay: MPD = [(T2 - T1) + (T4 - T3)] / 2
   b) Time of Day Offset (ToD Off): T6 - T5 - MPD
   c) Frequency Offset (Freq Off): (Fo - Fr) / Fr where Fr = 1 / (T5 - T1), Fo = 1 / (T6 - T2)
7. Software set ToD Off and Freq Off to ToD

To initialize PTP function, use following APIs:

- gpy2xx_ptp_enable
- gpy2xx_ptp_set_tsctrl
- gpy2xx_ptp_settime

If PPS is required in the system:

- gpy2xx_ptp_set_ppsctrl

To get timestamp capture on Tx or Rx:

- gpy2xx_ptp_gettxts
- gpy2xx_ptp_getrxts

To get timestamp FIFO status or reset FIFO:

- gpy2xx_ptp_fifostat
- gpy2xx_ptp_resetfifo

To adjust ToD:

- gpy2xx_ptp_adjtime

To adjust frequency:

- gpy2xx_ptp_adjfreq

For more information, please refer to the API reference for detailed usage.

2.5.4 Handling of Programming Failures or Exceptions

Following are the possible failure cases:

- The master clock can stop operation. Master SoC host has to detect this error case. Master SoC can either switch to another master clock, drop the Ethernet link or inform the slave SoC through a PTP message, that the master clock is down. In case of link loss and a PTP message, the slave SoC will have to take the desired action like switching to a precise clock for holdover or stopping the slave clock.
- The Ethernet link can be dropped. Slave SoC will be informed by its GPY with an interrupt, when the Ethernet link is down. Slave SoC then has to take the desired action like switching to another clock or stopping the slave clock.

In both error conditions the PHY has no active role in the error handling.

2.6 Synchronous Ethernet

The GPY allows a Synchronous Ethernet (Sync-E) interface to support transportation of a source-referable clock from a clock master to clock clients. If the GPY is in slave mode then the GPY receives the synchronization clock from the Ethernet cable and provides it to the system either on pin GPC1 or GPC2. If the GPY is in master mode then the GPY receives the clock from the system either on pin GPC1 or GPC2 and sends it over the Ethernet cable as a clock master.

The Sync-E feature is not supported when the internal DC-DC is used. The GPY supports the following Sync-E input or output reference clock speeds:

- EEC-1 class: 2048 kHz
- EEC-2 class: 1544 kHz
- PSTN class: 8 kHz

The GPY API provides below listed functions for easy integration of the Sync-E feature into the host software.

- gpy2xx_synce_cfg - Configure GPY to enable/disable Sync-E feature, clock frequency selection, GPC selection, Master/Slave mode, data rate on TPI.
- gpy2xx_synce_get - Get the GPY Sync-E related configuration such as enable/disable Sync-E feature, clock frequency selection, GPC selection, Master/Slave mode, data rate on TPI.

2.6.1 Functional Description

Synchronous Ethernet (Sync-E) is used to synchronize frequencies over an Ethernet link or and Ethernet network. In the system perspective, Sync-E assists the synchronization of PTP counters to the SyncE frequencies. PTP wall clock increment rate is driven by physical layer. Sync-E operates in two modes as listed below:

- **Sync-E Lock Mode** - In Sync-E Lock Mode mode of operation, the Ethernet link between the master and slave is connected and the slave frequency is locked to that of the master frequency. A ppm change in the master GPY frequency is reflected at the slave GPY frequency. The deviation of the slave GPY frequency should be within 16 ppb from the center frequency of the master GPY.
- **Synce-E Holdover Mode** - When the GPYs are in Sync-E lock mode and operational, for any reason if the Ethernet link is broken between the master GPY and slave GPY then the slave GPY enters the Sync-E Hold Mode. This transition is indicated with MDINT interrupt to host and the event is recorded in PHY_ISTAT.LOR register if PHY_IMASK.LOR is enabled. The deviation of the slave GPY frequency should be within 4.6 ppm from the center frequency of the Sync-E Lock Mode.

Different Sync-E configuration parameters are detailed in **Table 2**.

Table 2 Sync-E Configuration Parameter

S.No	Parameter	Configuration
1	enSyncE	Enable or Disable Sync-E feature 0 - Disable 1 - Enable
2	syncEClk	Select Sync-E reference clock frequency 0 - CLK_PSTN 1 - CLK_EEC1 2 - CLK_EEC2
3	MasterMode	GPY mode of operation 0 - Slave Mode; Reference Clock output at GPC based on GPCSel 1 - Master Mode; Reference Clock input at GPC based on GPCSel

Table 2 Sync-E Configuration Parameter (cont'd)

S.No	Parameter	Configuration
4	DataRate	GPY Data Rate 0 - 1 Gbps 1 - 2.5 Gbps
5	GPCSel	Selection of GPC pin to input/output reference clock. 1 - GPC1 2 - GPC2

2.6.2 Typical System Application

The system level implementation of typical application scenario for Sync-E is as shown in **Figure 15**. The objective is to synchronize the Slave Clock reference frequency to the Master Clock reference frequency over an Ethernet link or network.

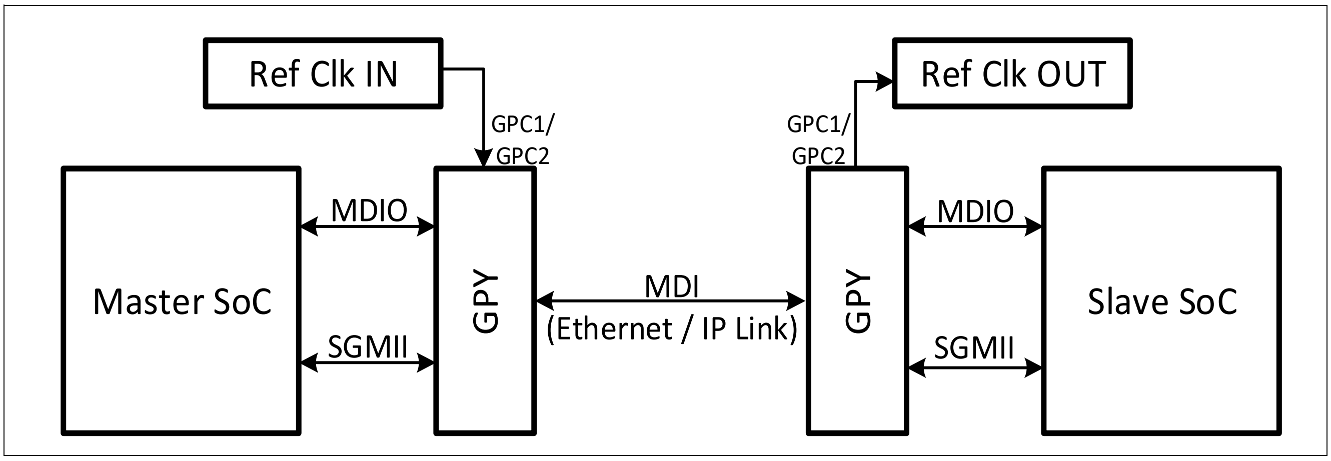


Figure 15 Sync-E System Application

A clock signal of 1.544 or 2.048 MHz or 8 kHz is the input to Master GPY on a dedicated pin (GPC1/GPC2). Master GPY synchronizes its internal clock to this Sync-E input clock. Slave GPY recovers the reference clock (1.544 or 2.048 MHz or 8 KHz) from the signal received over the Ethernet link and provides it on a dedicated pin (GPC1/GPC2). The frequencies of the Sync-E signals at master GPY and slave GPY are exactly the same, but the phase difference between the two clocks is undefined.

Energy Efficient Ethernet (EEE) cannot be used together with Sync-E as there will be quiet periods in EEE operation as per standards.

2.6.3 Handling of Programming Failures or Exceptions

Requirements for Sync-E systems are usually that the reference clock at slave GPY has to be switched in a very short time frame of a few milliseconds to a relatively precise holdover clock when the Ethernet link is broken.

Following are the possible fail cases:

- When the input reference clock to the Sync-E master GPY stops for any reason, the master SoC has to detect this situation and can either command its GPY to drop the Ethernet link or send a specific message over the Ethernet link to inform slave SoC of the loss of the signal.
- When the Ethernet link drops, slave SoC will be informed by its GPY in case of a drop of the Ethernet link. Slave SoC then has to take appropriate action based on the interrupt trigger from its GPY.

2.7 SGMII / SGMII+ Configuration

The SGMII Specification [7] defines how 10 Mbps, 100 Mbps and 1 Gbps data can be transmitted over the same interface, which operates at 1.25 Gbaud. For 100 Mbps frames are simply repeated 10 times and for 10 Mbps, frames are repeated 100 times.

When the twisted pair interface links up in 2.5GBASE-T mode, the SGMII interface must be switched to operate in SGMII+ (also called HSGMII) mode with 3.125 Gbaud to be able to transport the 2.5 Gbps data from the twisted pair interface.

That means, when changing the twisted pair rate from 10M/100M/1G to 2.5G or from 2.5G to 10M/100M/1G, the SGMII interface must be reconfigured. Unfortunately, there is no standard available, which allows to operate the SGMII/SGMII+ interface at one rate and cover all 4 rates 10M, 100M, 1G and 2.5G on the twisted pair interface.

Further information can be found in a release note [5] and datasheet covering this subject.

2.8 Thermal Management

Thermal management in GPY component level is handled at 2 stages as follows:

- Thermal Mitigation as explained in **Chapter 2.8.1**
- Thermal Shutdown as explained in **Chapter 2.8.2**

2.8.1 Thermal Mitigation

The temperature sensor on the GPY continuously measures the temperature and updates the temperature code value in the VSPEC1_TEMP_STA.TEMP_DATA register. The host SoC can read the VSPEC1_TEMP_STA register to monitor the temperature of GPY.

Figure 16 describes the flow of events during the thermal mitigation. When the temperature is above 110°C, GPY interrupts the SoC with the MDINT signal. It is the responsibility of the SoC to take the appropriate action. The GPY relies on the host SoC for cooling measures such as to increase the cooling fan speed adequately enough to bring down the temperature to less than 110°C or scale down the link-speed or link-down the GPY or any other measures.

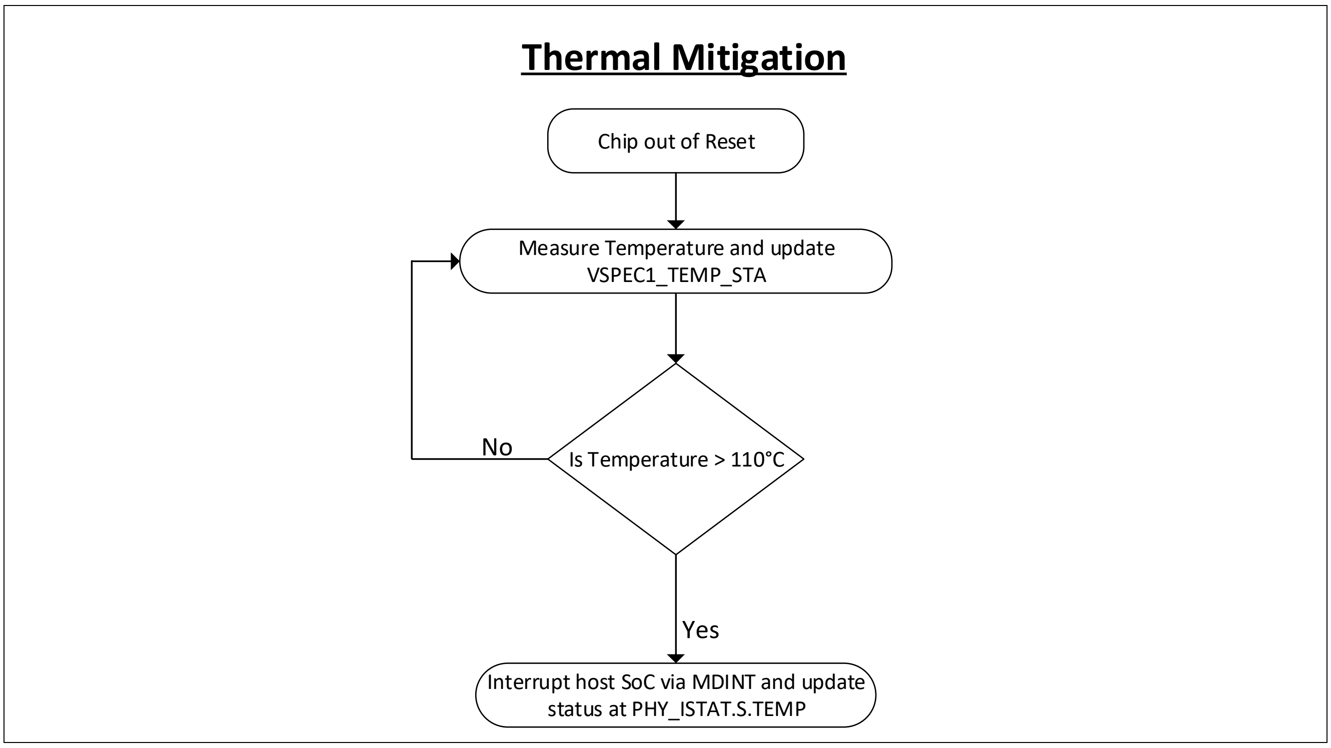

Thermal Mitigation
Chip out of Reset
Measure Temperature and update VSPEC1_TEMP_STA
Is Temperature > 110°C
No
Yes
Interrupt host SoC via MDINT and update status at PHY_ISTAT.S.TEMP

Figure 16 Thermal Mitigation

Conversion formula to calculate T_j in Celsius = (-2.7882E-11)*N^4 + (1.0108E-7)*N^3 + (-1.9390E-04)*N^2 + (3.1220E-1)*N + (-5.0322E+1), with N = decimal return value from VSPEC1_TEMP_STA.TEMP_Data.

2.8.2 Thermal Shutdown

Thermal shutdown is a safety feature for the GPY to avoid the permanent damage. As explained in **Chapter 2.8.1**, GPY interrupts the SoC with MDINT signal to take thermal mitigation action when the temperature is greater than 110°C. If the cooling actions taken by the host SoC to bring down the temperature is not adequate enough to bring down the temperature of GPY then at 125°C, GPY autonomously links-down and restarts auto-negotiation with the maximum rate of 1G (2.5G is disabled). When GPY is operating in 1G then the GPY cools down to temperatures lesser than 110°C. However, the recommendation is to take the thermal cooling actions to avoid operating the chip at dangerously high temperatures. **Figure 17** describes the flow of events during the thermal shutdown.

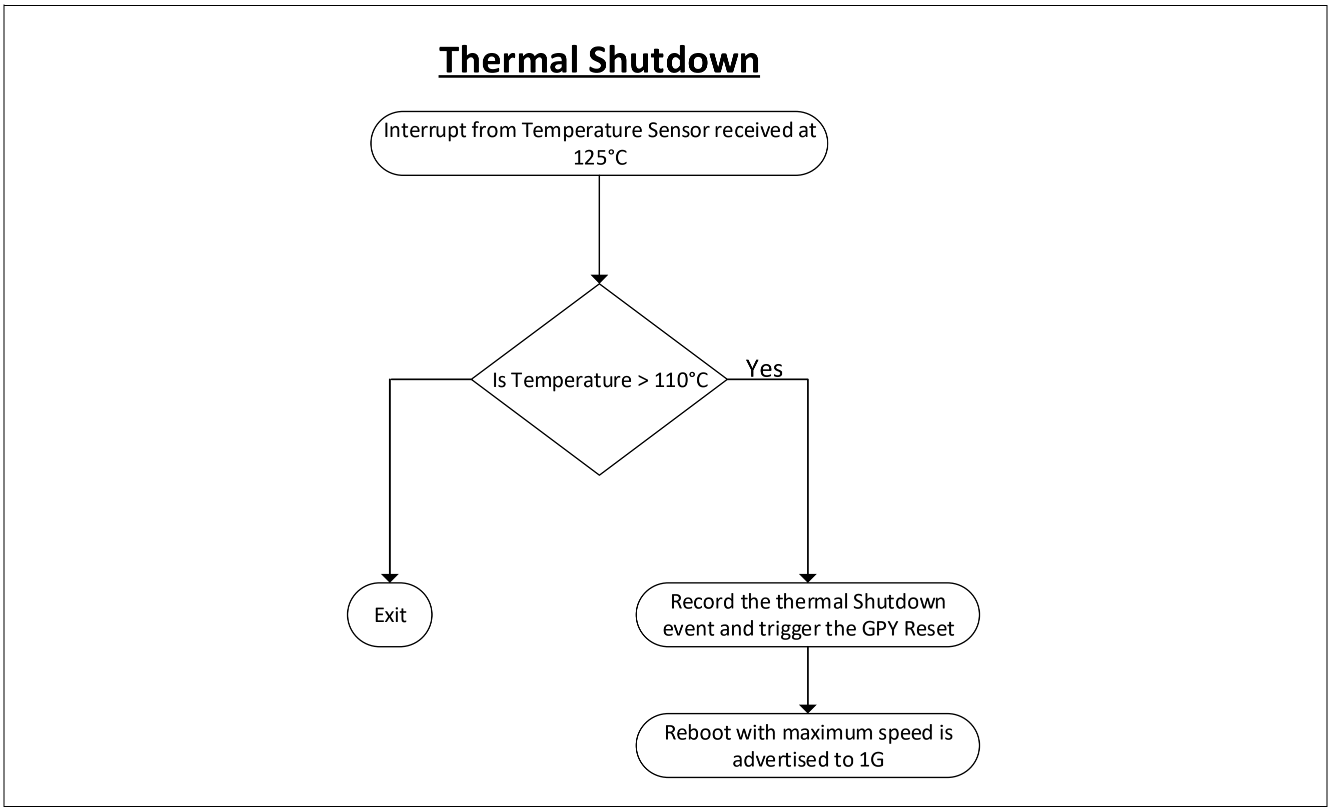

Thermal Shutdown
Interrupt from Temperature Sensor received at 125°C
Is Temperature > 110°C
Yes
Exit
Record the thermal Shutdown event and trigger the GPY Reset
Reboot with maximum speed is advertised to 1G

Figure 17 Thermal Shutdown

After the thermal shutdown event is recorded, the GPY operates with 1G as the maximum speed. After the host SoC taking necessary actions to avoid the temperature build up, the host should set the VSPEC1_NBT_DS_CTRL.FORCE_RST bit to enable GPY to advertise 2.5G as maximum speed.

References

[1] Intel® Ethernet Network Connection GPY211 (GPY211B1VC) Data Sheet Rev. 1.1

[2] Intel® Ethernet Network Connection GPY212 (GPY212B1VC) Data Sheet Rev. 1.1

[3] Intel® Ethernet Network Connection GPY215 (GPY215B1VI) Preliminary Data Sheet Rev. 1.1

[4] Intel® Ethernet Network Connection GPY115 (GPY115B1VI) Preliminary Data Sheet Rev. 1.1

[5] Intel® Ethernet Network Connection GPY211B1VC/GPY212B1VC Release Notes Rev. 1.1, in preparation

[6] IEEE 802.3-2018: “Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications”, IEEE Computer Society

[7] Serial-GMII Specification: Revision 1.8, Cisco Systems, November 2 2005

Attention: Please refer to the latest revisions of the documents.