
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS


# TESTOVACÍ MODUL PRO VYBRANOU ČÁST STANDARDU IEEE 802.1Q

TESTER FOR CHOSEN SUB-STANDARD OF THE IEEE 802.1Q

DIPLOMOVÁ PRÁCE
MASTER'S THESIS


AUTOR PRÁCE — Bc. Nikola Avramović
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Lukáš Fujcik, Ph.D.
SUPERVISOR



BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

***Student:*** Bc. Nikola Avramović ***ID:*** 164708
***Ročník:*** 2 ***Akademický rok:*** 2018/19

**NÁZEV TÉMATU:**

## Testovací modul pro vybranou část standardu IEEE 802.1Q

**POKYNY PRO VYPRACOVÁNÍ:**

Cílem diplomové práce je analyzovat současné IEEE standardy ze skupiny 802.1Q a pro vybraný pod-standard navrhnout metodu automatického ověření správné implementace. Součástí práce bude realizace na bázi VHDL, kterou bude možné implementovat pro vybrané FPGA, a která pokryje zvolenou část standardu. Výstup musí být dostatečně detailní pro určení příčiny selhání, případně úspěšného provedení každého dílčího testu.

**DOPORUČENÁ LITERATURA:**

According to recommendations of supervisor

***Termín zadání:*** 4.2.2019 ***Termín odevzdání:*** 21.5.2019

***Vedoucí práce:*** doc. Ing. Lukáš Fujcik, Ph.D.
***Konzultant:*** Ing. Petr Grillinger, Ph.D.,TTTech Computertechnik AG, o.s

**doc. Ing. Lukáš Fujcik, Ph.D.**
*předseda oborové rady*

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

# Master's Thesis

Master's study field **Microelectronics**
Department of Microelectronics

***Student:*** Bc. Nikola Avramović ***ID:*** 164708
***Year of study:*** 2 ***Academic year:*** 2018/19

**TITLE OF THESIS:**

## Tester for chosen sub-standard of the IEEE 802.1Q

**INSTRUCTION:**

The main goal of master thesis is to analyze the group of IEEE 802.1Q standard and to design method for automatic testing of implementation by chosen sub-standard. Master project will include implementation of the tester, which could be realized in the VHDL, and which covered the chosen sub-standard. Tester output has to be sufficiently detailed to determined causes of failure, eventually to show successful execution of every single test.

**REFERENCE:**

According to recommendations of supervisor

***Assigment deadline:*** 4. 2. 2019 ***Submission deadline:*** 21. 5. 2019

***Head of thesis:*** doc. Ing. Lukáš Fujcik, Ph.D.
***Consultant:*** Ing. Petr Grillinger, Ph.D.,TTTech Computertechnik AG, o.s

**doc. Ing. Lukáš Fujcik, Ph.D.**
Subject Council chairman

# ABSTRACT

This master paper is dealing with the analysis of IEEE 802.1Q group of TSN standards and with the design of HW tester. Standard IEEE 802.1Qbu has appeared to be an optimal solution for this paper. Detail explanation of this sub-standard are included in this paper. As HW test the implementation, a protocol aware technique was chosen in order to accelerate testing. Paper further describes architecture of this tester, with detail explanation of the modules. Essential issue of protocol aware controlling objects by SW, have been resolved and described. Result proof that this technique has reached higher speed of testing, reusability, and fast implementation.



# KEYWORDS

Ethernet, TSN, IEEE, OSI, Generator, Monitor, DUT, ATE, PHY, MAC, FPGA, DFT, BIST, ATPG, testing, PA ATE, real-time, pattern, port, scheduling, traffic



# ABSTRAKT

Tato práce se zabývá analyzováním IEEE 802.1Q standardu TSN skupiny a návrhem testovacího modulu. Testovací modul je napsán v jazyku VHDL a je možné jej implementovat do Intel Stratix® V GX FPGA (5SGXEA7N2F45C2) vývojové desky.

Standard IEEE 802.1Q (TSN) definuje deterministickou komunikace přes Ethernet sít, v reálném čase, požíváním globálního času a správným rozvrhem vysíláním a příjmem zpráv. Hlavní funkce tohoto standardu jsou: časová synchronizace, plánování provozu a konfigurace sítě. Každá z těchto funkcí je definovaná pomocí více různých podskupin tohoto standardu. Podle definice IEEE 802.1Q standardu je možno tyto podskupiny vzájemně libovolně kombinovat. Některé podskupiny standardu nemohou fungovat nezávisle, musí využívat funkce jiných podskupin standardu. Realizace funkce podskupin standardu je možná softwarově, hardwarově, nebo jejich kombinací.

Na základě výše uvedených fakt, implementace podskupin standardu, které jsou softwarově související, byly vyloučené. Taky byly vyloučené podskupiny standardů, které jsou závislé na jiných podskupinách. IEEE 802.1Qbu byl vybrán jako vhodná část pro realizaci hardwarového testu.

Různé způsoby testování byly vysvětleny jako DFT, BIST, ATPG a další jiné techniky. Pro hardwarové testování byla vybrána „Protocol Aware (PA)“technika, protože tato technika zrychluje testování, dovoluje opakovanou použitelnost a taky zkracuje dobu uvedení na trh.

Testovací modul se skládá ze dvou objektů (generátor a monitor), které mají

implementovanou IEEE 802.1Qbu podskupinu standardu. Funkce generátoru je vygenerovat náhodné nebo nenáhodné impulzy a potom je poslat do testovaného zařízeni ve správném definovaném protokolu. Funkce monitoru je přijat ethernet rámce a ověřit jejich správnost.

Objekty jsou navrhnuty stejným způsobem na „TOP“úrovni a skládají se ze čtyř modulů: Avalon MM rozhraní, dvou šablon a jednoho portu. Avalon MM rozhraní bylo vytvořeno pro komunikaci softwaru s hardwarem. Tento modul přijme pakety ze softwaru a potom je dekóduje podle definovaného protokolu a „pod-protokolu “. „Pod-protokol“se skládá z příkazu a hodnoty daného příkazu. Podle dekódovaného příkazu a hodnot daných příkazem je kontrolovaný celý objekt. Šablona se používá na generování nebo ověřování náhodných nebo nenáhodných dat. Dvě šablony byly implementovány pro expresní ověřování nebo preempční transakce, definované IEEE 802.1Qbu. Porty byly vytvořené pro komunikaci mezi testovaným zařízením a šablonou podle daného standardu. Port „generátor“má za úkol vybrat a vyslat rámce podle priority a času vysílaní. Port „monitor“přijme rámce do „content-addressable memory”, která ověřuje priority rámce a podle toho je posílá do správné šablony.

Výsledky prokázaly, že tato testovací technika dosahuje vysoké rychlosti a rychlé implementace.

# KLÍČOVÁ SLOVA

Ethernet, IEEE, TSN, OSI, FMS, Generátor, Monitor, DUT, ATE, PHY, MAC, FPGA, DFT, BIST, ATPG, testovaní, PA ATE, reální-čas, šablona, port

AVRAMOVIĆ, N. TESTOVACÍ MODUL PRO VYBRANOU ČÁST STANDARDU IEEE 802.1Q. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2019. 76 s. Vedoucí semestrální práce doc. Ing. Lukáš Fujcik, Ph.D.

## ACKNOWLEDGEMENT

I would like to give special thanks to my supervisor doc. Ing. Lukáš Fujcik, Ph.D. for directing and guiding my work. Additionally, I am greatly thankful to the TTTech Computer Technik AG company for providing me with devices and supplying me with all necessary information which were more than helpful for the completion of this thesis.

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře vybudované v rámci projektu CZ.1.05/2.1.00/03.0072
**Centrum senzorických, informačních a komunikačních systémů (SIX)**
operačního programu Výzkum a vývoj pro inovace.

# CONTENTS

# INTRODUCTION

Test and measurement accuracies represent a key issue in any company interested in electronic design and manufacturing industry. To ensure both quality and reliability, every good company estimates more than 60% effort for testing [1]. This means that testing is more important than design, so hardware testers are continuing to be a challenge for high performance digital devices [2].

Automated testing equipment (ATE) has a key role to play in the various levels of product design, life cycle and development. ATE architecture has been designed for testing with multi-channel I/Os or even standard date busses. These test systems send dedicated stimuli to DUT and response vectors that rely on a fixed timing relationship between the tester and DUT. Conventional ATE architecture is dependent on pre-calculated signalling vectors and is not equipped for real-time processing and response generation to incoming signals. Protocol based interfaces (frames, date streams...) utilizing dynamic timing cannot be supported [2, 3].

By increasing the data rate in digital systems which goes up to several Gigabits per second (Gbps), the approach to generate a high-speed testing patterns equipment becomes more complex and harder to be implemented. The cost of this equipment rises exponentially and is still not flexible for reuse. FPGAs are very flexible devices in re-configuration, which is considered to be a very adequate platform for implementing a testing module [2, 3].

In the past Internet has been used for collecting information and communication, but progressively it evolved into commercial instrument which could be used for remote control across an existing network. Increasing data rate detected a lot of new issues. Such issues have been related mostly to the audio/video (AV) streaming. Developers for audio/video systems have started to use time-sensitive networking to achieve high quality requirements. IEEE 802.1Q Time Sensitive Networking (TSN) task group has developed a new standard for this purpose. The main goal of these systems is to achieve reliable and in time precise communication. TSN added a lot of different functions to standard Ethernet, mostly timing aware protocols.

By increasing functionality, the complexity of hardware design rises. These systems need faster and more precise technique equipment. This thesis will be oriented on how to improve testing of Time Sensitive Networking. Thesis will be divided in two parts. The first part is related to theoretical analysis of hardware techniques, their improvements, explanation of what TSN is, functionality of TSN and key sub-standard, and, finally, the description of the kit for hardware implementation. The second practical part is related to the analyses of TSN sub-standards, and architecture of hardware tester.

# 1 THEORETICAL ANALYSIS

TSN is a relatively new technology for real-time systems and therefore testing of these systems almost does not exist. In this chapter a technique used in the past for testing will be explained and how it has been improved. For the testing of TSN functionality of those systems will be analyzed. Also, a key sub-standard will be explained, considering the fact that is to be used in further work. In the last sub-chapter, the chosen hardware for testing will be shown.

## 1.1 Testing principles

The main goal of testing is to measure whether a device or system works correctly. The test should determine if the module is designed properly, or to prove if the design functions correspond to the requirements. As the design becomes more complex, operates on higher speeds, decreases to low consumption, the testing challenges get more complex. To meet these requirements, test techniques have continued to evolve. A better tester could be achieved by improving the testing elements, or to even allow the device to assist in its own evaluation. The basic test principles are presented below [3].

### 1.1.1 Automatic Test Equipment (ATE)

Conventional ATE is being used for testing different digital systems with a common interface. Digital systems have been checked by ATE in all kinds of testing purposes over the last three decades. "Nicholas DeWolf, the chief engineer at Semiconductor Manufacturer Transitron, teamed with Alex d'Arbeloff in 1961 to found Teradyne in Boston, they built the D133 diode tester, one of the first commercial test systems to use semiconductors in place of vacuum tubes. In 1966, DeWolff and Milt Collins designed the model J259 based on the Digital Equipment Corporation PDP-8 mini-computer became the first computer - controlled test system in the industry" [4].

AT the very beginning, ATE was used to test radio frequency (RF), mixed signals, optical and digital components. During the 80s, the speed of digital components rapidly increased, because of new technology. The manufacturers of ATE have added a MUX allowing to send parallel data to the device and that has achieved higher speed. At that time the engineers had introduced "shared resource" architecture, which was inexpensive, and it was simple for implementation, with just a few channels. For the device with many channels this technique has congested sharing resource. Tester engineers made "per-pin" architecture to resolve this issue. Each tester channel is comprised of a set of hardware features, specifically a test sequence memory, timing generator, and pin electronics, that is unique to that channel, shown in Figure 1-1 [2, 3].

ATE is controlled from the host computer, at the left side of the figure. The host is used to trigger the test and to analyse the results. “Per-pin” architecture has one ATE per pin, which is shown in the center of the figure. Patterns or stimuli for the testing are stored in sequence memory, which has communicated with the host commuter. The main purpose of the timing generator is to get the vector form input and to transform it into time domain representation of that vector on the output. Pin electronics have the ability to generate a vast array of voltage levels, varying signal drive strengths, and emulate termination of the signal. By increasing the pins in the device under test (shown on the right side), the cost of the ATE rises exponentially [3].

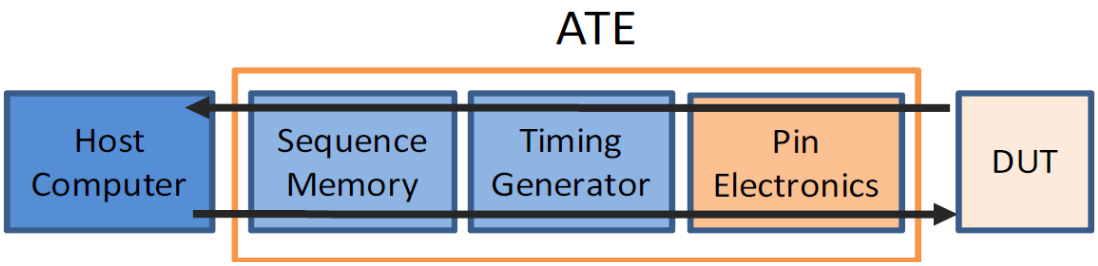



Figure 1-1: Basic ATE [3]

### 1.1.2 Design for testing (DFT)

Smaller system and devices could achieve coverage rates using black box fashion, by external pins. However, manufacturing technology decreased the size of transistors, which allowed creation of a more complex design. Complex designs need larger number of pins, due to which ATEs lose the ability to effectively test the design. To improve test coverage and reduce test costs, new architecture needs to be created. Design for testing to resolve this issue, because it uses existing logic in a design to enhance the testability of the underlying design. One of architecture segments is a scan chains representing a series of registers configured as a large sequential shift register. Data can be sent into a device or be read from the output with a minimal number of device pins, allowing us to check the data deep inside the design and that could not be reached with a proper test. Owing to the location of the cells, they are called Boundary Scan Cells [3].

Boundary scan, also known as JTAG (Joint Test Action Group) is defined in standard IEEE 1149.1 [5, 6]. A JTAG interface is a serial signalling system comprised of five signals. The main advantage of the JTAG is a small number of pins, which provides a very high amount of deep logic access. This protocol could be implemented in a single chip or it can be implemented in multiple designs (shown in Figure 1-3), requiring only a single test point connection for a board level test solution. The internal components for a JTAG enabled device are shown in Figure 1-2 [3].

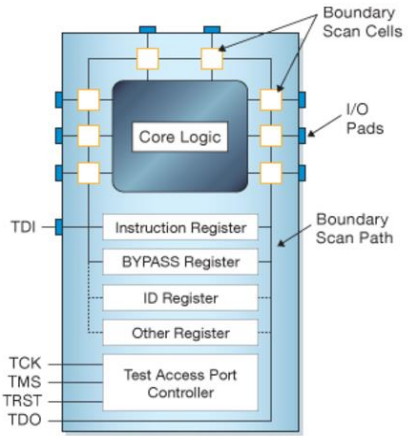



Figure 1-2: Schematic Diagram of a JTAG enabled device [5]

"The JTAG interface, collectively known as a Test Access Port, or TAP, uses the following signals to support the operation of a boundary scan.

- **TCK** (Test Clock) – this signal synchronizes the internal state machine operations.
- **TMS** (Test Mode Select) – this signal is sampled at the rising edge of TCK to determine the next state.
- **TDI** (Test Data In) – this signal represents the data shifted into the device's test or programming logic. It is sampled at the rising edge of TCK when the internal state machine is in the correct state.
- **TDO** (Test Data Out) – this signal represents the data shifted out of the device's test or programming logic and is valid on the falling edge of TCK when the internal state machine is in the correct state.
- **TRST** (Test Reset) – this is an optional pin which, when available, can reset the TAP controller's state machine" [5].

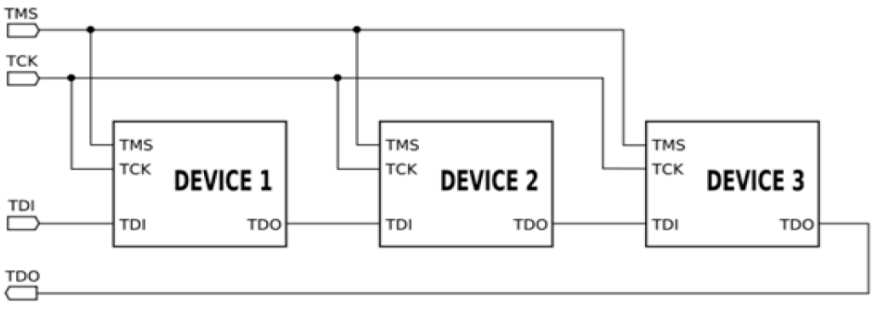



Figure 1-3: JTAG signal connections [3]

### 1.1.3 Built-in Self-test (BIST)

Another popular DFT method is a technique called built-in self-test (BIST), and it was first applied into test memory devices in late 1980s [7]. BIST is a technique where the applied tester is located inside the hardware elements. Basic BIST solution implementation is comprised of a pattern generator, a response analyzer, and a test controller [3].

"BIST method is generally specified into two major types: online and offline. Online BIST is designed to function when a device is in normal operation mode, in other words, it is able to detect errors in real time. Offline BIST is designed to perform testing functions when the device is not in normal operation mode, such as on power up procedure. Both types of BIST require very similar architecture compare to the under-testing device. The main components of the BIST are test pattern generators (TPGs) and output response analyzers (ORAs). A BIST controller component controls the operation of TPG and access outputs from ORA to determine the test results. These components are usually built by using registers and finite state machines (FSM) [2]." Pattern in TPG can statically be stored in memory or can be generated dynamically through the use of a random generator (e. g. LFSR). ORA can similarly use stored responses or process a signature analyzer [3].

One of BIST designs is shown in Figure 1-4, this design mixed both types (online and offline). This system was built inside a chip and requires only an access mechanism like the test access port (TOP) to start. Before starting any functional test (powered on mode), BIST can check if logic is working properly. After initialization TPG (in this case pseudorandom pattern generator (PRPG)) sends out test patterns, data pass through scan chains and then collects the responses in ORA (in this case a multiple-input signature register (MISR)). The final content of the ORA is a signature, which determines a pass/fail result. The signature is sent out via TAP and then compared with a pre-calculated or expected value [8].

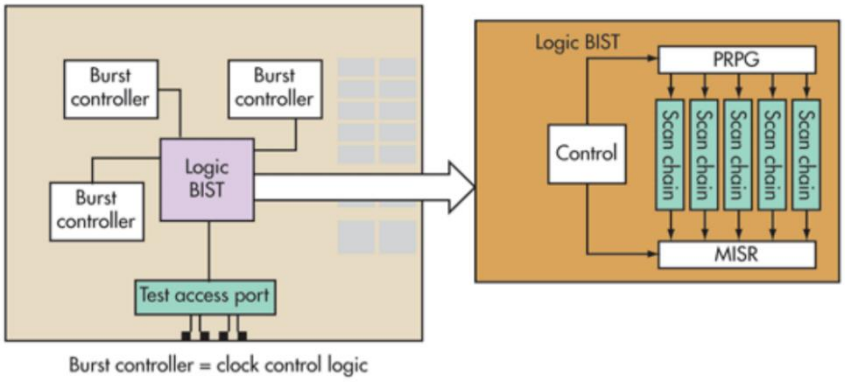



Figure 1-4: Logic Built-In Self-Test [8]

In some applications BIST includes scrubbing. Scrubbing is a protection mechanism for fault-tolerant circuitry, which means that PRPG generates specific set of patterns for checking of fault-tolerant functions. [9]

BIST can be used as a counter check. If the design went into a not allowed state, then it can set an internal error signal which increases error counter. According to an error priority BIST can stop design.

### 1.1.4 Automatic test pattern generation (ATPG)

A larger device has a complex design, and in some cases, it is not important to check whether every component is working properly as it was shown in chapter 1.1.3. To test a complex system, sometimes it is necessary to apply patterns from the external tester and observe the results. These designs need to use embedded compression to reduce the

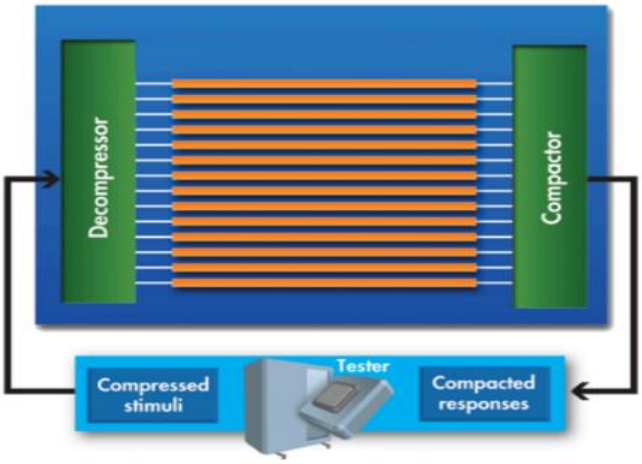



Figure 1-5: Automatic Test Pattern Generation (ATPG) [8]

pattern volume and test time. Compression by dividing the design's scan chains into smaller balanced chains that are connected between a decompressor and compactor (Figure 1-5). This is one more DFT technique called Automatic Test Pattern Generation (ATPG), and it needs a small number of patterns, and only a few primary I/Os need to be connected to the external tester [10] [8].

During BIST test, any unexpected value will corrupt the test responses and result in an incorrect signature, even if that pattern should just pass through that logic. These could not happen in ATPG, because it used deterministic patterns to target the faults. The advantage of an ATPG is that it needs less effort to implement the tester, and if BIST has been used to pseudorandom patterns, it is in that case both difficult to observe and to control. The main disadvantage of ATPG is the external test unit, which increases the cost of testing environment. Both ATPG and BIST can achieve the same level of diagnostic resolution. Mostly, the diagnostics for BIST designs are done offline after collecting the information on failing signatures. For ATPG, the diagnostics can be done concurrently with the production testing [8].

## 1.2 Testing improvements

As shown before, every technique has its own advantages and disadvantages. Regarding that fact, in early 2000's manufacturers have introduced a new "open-architecture" test platform. In this architecture, a tester could be fully customized from motherboard, CPU, memory and other auxiliaries to build an application-specific system. New test platform has forced the manufacturer to implement a new technology (SoC, FPGA…) in ATE systems. Furthermore, it was necessary to mix different testing techniques (BIST, JTAG, ATPG…), because of the demand for higher performance, reduced costs, better accuracy,

increased memory requirement and other advanced testing features.

### 1.2.1 ATE Extension

Many different applications need to meet very high bandwidth and source-synchronous signalling requirements, in which neither the native capabilities of an ATE system nor similar test solutions can provide the needed performance. To meet the higher performance, extension modules can be added to the ATE system or other testing modules to extend capabilities of a basic system [11]. Extension module consists of a driver and

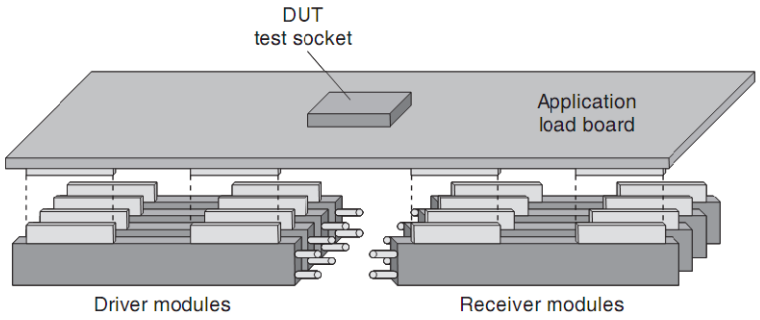



Figure 1-6: Intermediate driver and receiver modules extending the capability of the testing module [3]

receiver modules integrated between the testing module and the DUT (shown in Figure 1-6). These modules can buffer, switch, or multiplex a collection of stimuli signals from the ATE to an alternate configuration or higher data rate and subsequently applied to the DUT [3].

The main challenge in the basic test module could the usage of protocol signals (frames, data streams…), which could be resolved by adding an extension module. Deterministic data will be sent by a basic test module, and random data will be adding an extension, which can be calculated by some technique (e.g. CRC). Also, a receiver will be checked for random data and other determinism will be sent to basic module. Older testing modules could be reused with this approach to reduce the cost of special equipment.

### 1.2.2 FPGA-BASED testing platform

“Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. FPGAs can be reprogrammed to desired application or functionality requirements after manufacturing [12].” FPGAs can be used to dedicate hardware blocks for more complex logic elements such as clock distribution networks, block memories, processors, I/O blocks and even high-speed transceivers. Today FPGAs are available that utilize less then 28nm CMOS technology and support I/O rates of more than 100Gbps (using dedicated serial I/O pins). Due to the big potential of FPGAs in digital design leading companies found the hybrid circuit with embedded microprocessors, peripheral components and programmable cells to form a complete "system-on-a-programmable-chip” (SoC).

Designs can be created at the gate level, register-transfer level or behavioural level described in a low-level hardware description language (HDL) such as VHDL, ABEL or

Verilog. Moreover, design could be described in high-level HDL, such as OpenCL, Chisel, CλaSH, SpinalHDL …

Digital designers use a lot of discrete chips in testing modules that result in complicity in devices. FPGAs reduced number of chips in order to make a dedicated connection between digital cells and to improve other features of a testing module. Those circuits are especially useful for implementing large blocks of combinational logic. Nowadays, FPGAs are using dozens or hundreds of I/O pins, hundreds of thousands of equivalent gates operations, memory blocks, DSP blocks, with almost unlimited set of configurations. Hardware design could be made through the use of a schematic capture, hardware description languages, or the implementation of predefined intellectual property (IP) cores [3].

The use of FPGA in a testing module is not an innovative concept. One solution is provided by Intel FPGA, which has a variety of cores that can be utilized in ATE products. A typical instrument card is shown in Figure 1-8 as an ATE system. According to the application of ATE, a designer could implement different intellectual property (IP) cores (also known as digital logic core DLC) into programmable logic [13].

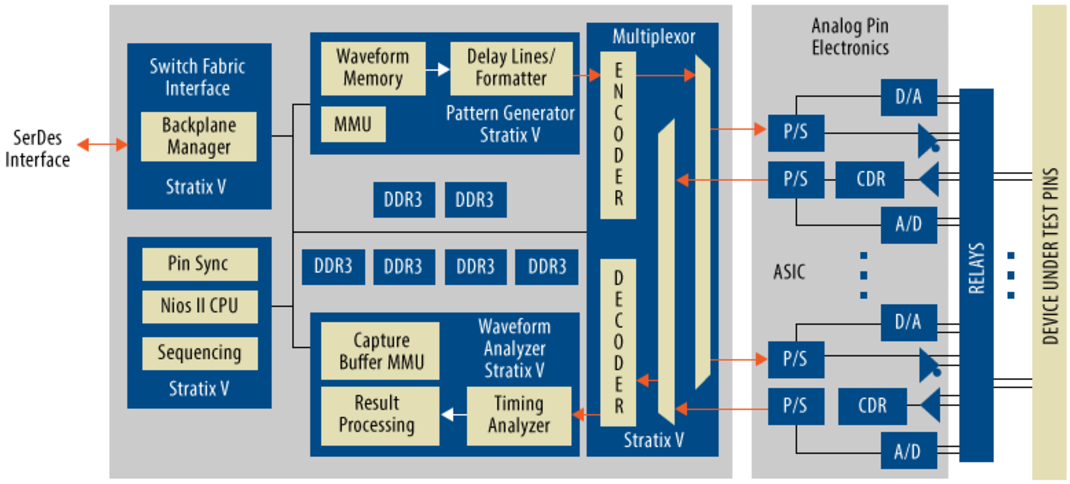



Figure 1-8: Typical ATE test station in FPGA [12]

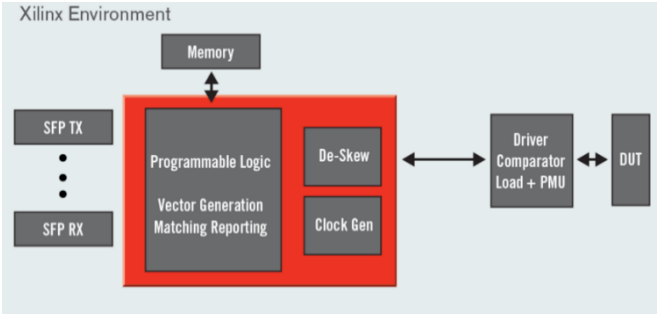



Figure 1-7: Testing of high speed electronic pins [13]

XILINX proposes another solution with SoC testers, by testing a broader range of IO protocols. That company offers big flexibility for realizing the volume of SoC and

Memory testers. For example, a vector generator and an analyzer for testing of large numbers are shown in Figure 1-7 for high performance IOs which are connected to and application specific standard product (ASSP). Electronic pins have given the 1Gbps+ link speed, driven by PLL and DLL based clock generation, with skew management capabilities [14]. These modules can be implemented as explicit circuits on the same interface board as the DLC or as reusable and removable plug-in boards. Test vectors can be stored within the FPGA memory resources, in secondary memory elements, or synthesized in real-time by state machines encoded in the FPGA [3].

### 1.2.3 Protocol Aware ATE

As the above designs show, the use of FPGAs have been the improved/extended ATE. But it was not a sophisticated tester, because ATE cannot natively and in real time handshake with simple protocols such as JTAG, PCI, SPI, I2C or I2S, not even with more complex ones such as DDR or PCIE. This issue resolved a "hybrid" concept which is called the Protocol Aware ATE (PA-ATE). According to the author Andrew C. Evans from Broadcom Corporation, A-ATE is simply defined as "the ability for the ATE pin or pin group to natively emulate real time chip I/O at the protocol level [15]." This hybrid test methodology combines modern DFT and standard test, with coverage along at-speed functional testing to "top up" coverage, in order not only to obtain highest transition delay coverage, but also to obtain "interoperability" or "system level" test coverage. [15]

PA-ATE should not implement just the system requirements, but also need to operate at a higher level of abstraction for the chip I/O. This protocol could resolve a main problem with multiple clock domain crossings; process, voltage, and temperature variations; as well as asynchronous interfaces between IP blocks. [15]

**What approach is better for testing, Test-Bench or ATE?**

Both are, depending in which design faze you are. Test-Bench has been used to test the device during development and it is extremely flexible. On the other hand, ATE is fast, automated, economical test equipment, but inflexible. In Figure 1-9 a) a large-scale iteration ATE is shown. In this case the testing starts with a Verilog simulation written with high level requirements sending stimuli to the model through model's interface. During the simulation stimulus and responses are written to a VCD file. This stimulus/response is translated from VCD file and compiled into an ATE format. Time

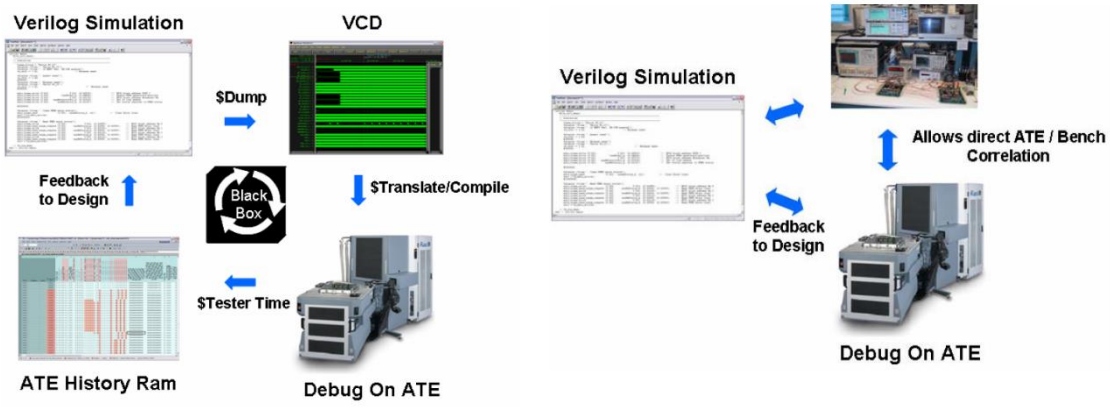



Figure 1-9: a) Large Scale Iteration b) PA ATE Homogeneous flow [14]

for testing of chip depends on the complexity and length of simulation. If bugs occur in device, it has been hard to find the bug. For test engineers this device becomes a “black box”, because of thousands or even millions of vectors. [15]

PA-ATE has been operating at a high level of abstraction as mentioned, the test is based on language (test patterns) instead of fixed captured simulations. Test patterns are specific pad rings for tested device, not the IP block. When the pad ring of testing IP is written at a higher level of abstraction, the test IP can be used on any device which includes those pad rings. Figure 1-9 b) shows how the iterating can be removed by running the same bench level of abstraction. Removing could help create a much-needed homogeneous environment between the bench, design verification and ATE. [15]

The implementation of PA-ATE could happen in three ways: Virtual Emulation, Bench-Functional ATE and Hybrid approach. First, Virtual Emulation creates a true ATE master for serial and parallel interface and all protocol transactions from high level function calls. This will be used for ATE sequencer with microcode which includes looping, subroutine branching, handshaking with host PC, and requires real time methods of updating and reading. Functional (Bench) approach will add more instrumentation in the ATE (e.g. BIST). Regarding the Bench level of abstraction, it will be a more physically specific function interface. The third Hybrid approach will be integrating the protocol aware functionality between digital pins and host PC, which could be a natively emulated protocol. This way could be resolved with programmable logic device (FPGA, SoC, CPLD), which will be implemented vector memory, pattern/timing generators and formatters. FPGA will be bypassed pins, that will be an unknown transition from host side. Also, FPGA might change logic inside by programming through standard host protocol [15].

Relationship between SW and HW is shown in Figure 1-10 a), PA-ATE consist of: **PA Port** (An ATE pin or group of pins that emulate a given protocol), **PA Object** (An object-oriented programming model for the PA Port which contains properties and methods for a given protocol) and **PatGen (**Standard pattern generator or sequencer). Communication between host and pattern generator in hardware already exists as shown in Figure 1-9 a). The critical item will be a two-way communication between PA Port-PatGen and PA Port-Pa Object, improving that communication infrastructure which could be provided for: running real time transactions from host; downloading and uploading transactions from/to PA Port, downloading and uploading memory images from/to PA Port, **START**ing or **TRIG**gering the PA Port from the PA Object or PatGen; as well as or handling polling and interrupt requests [15].

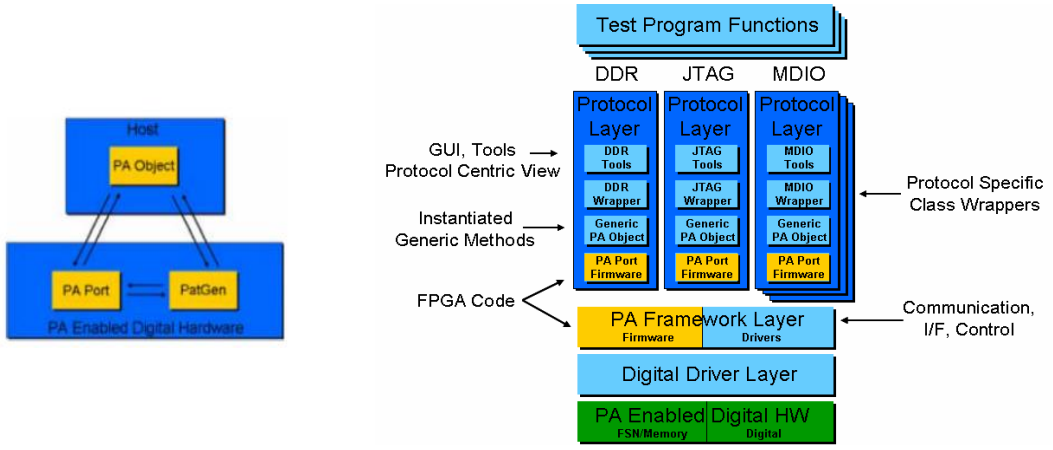



Figure 1-10: PA ATE a) control/data flow b) SW Architecture

To effectively use the PA, it is extremely important that the software language and tools should be more "protocol" centric, not "ATE" centric. Language also needs protocol specific firmware for the FPGA. Figure 1-10 b) shows software architecture, that consist of horizontal layers for functional protocols, or future protocols, protocol layer is virtual and scalable (made up out of four components), PA Framework layer should program FPGA, and other layers in FPGA [15].

## 1.3 Brief history of Internet and Time sensitive networks (TSN)

The first communication system with Internet Protocol was founded in the US Department of Defense in the early 1960's, called ARPANET. Access to the ARPANET was expanded in 1981 when the National Science Foundation (NSF) founded the Computer Science Network (CSNET). In the middle of expansion of ARPANET Robert Metcalfe wrote the doctoral dissertation about Ethernet. Back then it was called ALOHANET, which was later developed by the company Xerox PARC. Company Xerox has protected the Ethernet name, so another company started to use the name Internet. The IEEE group made IEEE 802 the standard for Internet, while the first three layers of ISO/OSI model called Ethernet. Since the mid-1990's, the Internet has had a revolutionary impact on culture and commerce, also Internet has influence in almost every type of communication (el. mail, social networking, Voice over Internet Protocol (VoIP), two-way interactive video calls, World Wide Web (www)…). At the very beginning Internet took over just 1% of the global communication landscape, until nowadays that Internet takes up more than 97% of the telecommunication network. [16]

The main purpose of Internet is to collect information and communication, but progressively it was included into a commercial instrument through which people could book services, pay bills and so on. At present, different types of cameras and sensors began to be connected to the Internet of Things which allows objects to be sensed and controlled remotely across existing network [16]. As the list of objects increased with the connection to the Internet, the new term Internet of Everything was created. Regarding these requirements, the network needs to use real-time systems.

### 1.3.1 Real-time systems

Ethernet technology has proven to be an incredible method of communication in the IT world. "It is a very well standardized and open technology that is easily accessible to everyone, provides a wide range of bandwidth and physical layer options, and has significant support in a diverse range of application areas. Up until now there has been no real-time support in IEEE standardized Ethernet, leading to a number of proprietary modifications of Ethernet being used in industrial and transportation systems where real - time communication is a critical requirement. These solutions have typically been developed for specific tasks or domains, e.g. Profinet, EtherCAT and Ethernet/IP which compete for recognition in industrial automation. While these protocols perform their specialized tasks capably, they have limits when it comes to combining with standard (classical) Ethernet networks and devices. The scalability of adapted Ethernet solutions for different industries is also limited as each is tailored for a specific application area. For this reason, the IEEE Time-Sensitive Networking task group has been working since 2012 on standardizing real-time functionality in Ethernet. TSN (Time-Sensitive Networking) is the set of IEEE 802 Ethernet sub-standards that are defined by the IEEE TSN task group. The new standards describe several mechanisms for improved or even guaranteed real-time delivery of Ethernet traffic. Most prominently, TSN defines the first IEEE standard for time-triggered message forwarding in a switched Ethernet network, and therefore fully deterministic real-time communication within the 802 suite of

standards [17].”

“TSN achieves deterministic real-time communication over Ethernet by using global time and a schedule which is created for message paths across multiple network components. By defining queues which transmit their messages based on a time schedule, TSN ensures a bounded maximum latency for scheduled traffic through switched networks. In control applications with strict deterministic requirements, such as those found in automotive and industrial domains, TSN offers a way to send time-critical traffic over a standard Ethernet infrastructure. This enables the convergence of all traffic classes and multiple applications in one network [17].”

Table 1-1 shows all TSN sub-standards with description and current status [18]. Key sub-standards are 802.1AS/Cor2, 802.1Qbu, 802.1Qbv, 802.1Qcc.

Table 1-1 : TSN sub-standards

| TSN Sub-standard | Description | Status |
|---|---|---|
| 802.1Qbu | Frame pre-emption | Published |
| 802.1Qbv | Scheduled traffic | Published |
| 802.1Qbz | 802.11 Bridging | Approved |
| 802.1Qca | IS-IS Path Control and Reservation | Published |
| 802.1CB | Frame Replication and Elimination | Sponsor Ballot |
| 802.1Qcc | SRP enhancements & performance improvements | WG ballot |
| 802.1Qch | Cyclic queueing and forwarding | Published |
| 802.1AS/Cor2 | Tech and ed corrections | Approved |
| 802.1AS-REV | Time synch enhancements | WG ballot |
| 802.1Qci | Per Stream Filtering & Policing | Approved |
| 802.1Qcj | Auto Attach to PBB | Editor's draft |
| 802.1CM | Profile for Fronthaul | TG Ballot |
| 802.1Qcp | 802.1Q YANG data model | WG ballot |
| 802d | URN Namespace | Published |
| 802.1Qcr | Asynchronous Traffic Shaping | Editor's draft |
| 802.1CS | LRP (new registration protocol) | Editor's draft |
| 802.1Q-REV | Bridges and Bridged Networks | WG ballot recircle |
| 802.1CBcv | FRER YANG & MIB | develop PAR |
| 802.1Qcw | TSN (Qbu, Qbv, Qci) YANG | develop PAR |

## 1.4 Functionality of Time Sensitive Networks and key sub-standards

This section describes the foundational TSN mechanisms used to build distributed, real-time systems. These foundational mechanisms include the following [19]:

- Time synchronization
- Traffic scheduling
- Network configuration

And key knowledge of functionality of the TSN:

- TSN is layer 2 only, application not considered.
- Sub-standards are independent. They can be used (theoretically) in any combination. They refer to each other.
- Standard define configurable parameters. Managed objects, define way to configure and operate the functionality

As foundational mechanisms and key knowledge are introduced, these systems could involve certain mechanisms including redundancy, frame pre-emption, ingress policing, and security.

### 1.4.1 Time Synchronization

The key to providing determinism is a shared concept of time. “Examples of a distributed clock synchronization include synchronizing to a Global Positioning System (GPS) satellite synchronizing a controller’s internal clock to a Network Time Protocol (NTP) time server, or a group of controllers synchronizing to a common time source using the IEEE 1588 Precision Time Protocol (PTP) [19].”

**IEEE 1588**

IEEE 1588 is standard whose mechanism provide synchronizing of clocks connected via multicast-capable networks, including Ethernet. One function of the standard is to provides the fault tolerant synchronization between heterogeneous networked clocks. Protocol for this synchronization was defined as Precision Time Protocol (PTP). “Using PTP, all participating clocks are synchronized to the highest quality clock in the network [19].”

The most important feature in IEEE 1588 standard is a Best Master Clock Algorithm (BMCA), which defines a distributed algorithm to synchronize all clocks participating in the system. “Using the BMCA, participants advertise their clock’s capabilities so that clocks can be ranked against each other. If a participant observes that higher-ranking clock is present on the network, the participant will cease advertising its own clock, ultimately enabling participants to converge on the single clock each will synchronize to. This highest-ranking clock is called the grandmaster clock, used to synchronize all other slave clocks. If the grandmaster clock is removed from the network, or if its characteristics change so that it is no longer the highest-ranking clock, the BMCA enables

the participating clocks to automatically converge on a new highest-ranking clock, which becomes the new grandmaster. Slave clocks synchronize to the grandmaster using bidirectional multicast communication [19].”

**IEEE 802.1AS**

IEEE 802.1AS was developed by Audio/Video Bridging (AVB) task group to optimize distributed time synchronization in professional audio/video systems. This sub-standard are interpretation of IEEE1588, it was defined as interoperable mechanism for synchronizing time in the context of standard Ethernet. “As a profile of IEEE 1588, 802.1AS-2011 configures and selects from available IEEE 1588 options, including the following:

- Use of modified Best Master Clock Algorithm (BMCA), enabling fast convergence on a grandmaster clock source
- Use of peer delay mechanism for measuring network path delay
- Transport of time synchronization messages over IEEE 802.3 (Ethernet), 802.11 (Wi-Fi), and Coordinated Shared Networks (CSN)
- Mandatory participation by bridges, including participation in best master clock selection
- Inclusion of measurement of nearest-neighbour frequency offset in end stations and bridges for better synchronization accuracy and faster grandmaster convergence
- Specification of a performance requirement such that any two time-aware systems separated by seven or fewer hops will be synchronized to within 1 μs of each other [19]”

With adding a new feature in IEEE1588 Precision Time Protocol (PTP), task group of IEEE802.1AS also added with different name, generalized Precision Time Protocol (gPTP) [19].

**IEEE 802.1ASrev**

Sub-standard IEEE802.1AS has not had features which could include a redundant grandmaster clock. Also, it could not support for multiple concurrent timescales, using multiple gPTP domains, to enable the synchronization or correlation to multiple simultaneous time sources. IEEE TSN task group has made a new sub-standard, as revision of IEEE802.1AS to add new advanced functions. As is shown in Table 1-1 this sub-standard is still in progress [19].

### 1.4.2 Traffic Scheduling

To ensure the quality of service and proper operation of time-sensitive systems, a bridged network connection must ensure the high-priority traffic which can predictably meet bandwidth and latency requirements, especially in the presence of same-priority or best - effort traffic. In late 2000’s IEEE 802.1 AVB/TSN task group developed mechanisms, a credit-based traffic shaper used in audio/video systems. To improve this sub-standard TSN task group developed time-aware, scheduled traffic. Other task of TSN group was how to recognize time-sensitive streams from other flows. [19] In this chapter

it will be explained how these tasks were resolved by TSN sub-standards.

**IEEE 802.1Q**

These standards allow data to be tagged adding a 16-bit header to an Ethernet frame, shown in Figure 1-11. The tag consists of 12-bits Virtual LAN (VLAN) ID, 3-bits for Priority Code Point (PCP) and 1-bit for Drop Eligible Indicator (DEI) [20]. IEEE802.1Q uses PCP tag to determine the traffic for the given Ethernet frame. There is a maximum of eight priorities as the tag is defined by three bits ($2^3 = 8$). Specifically, bridges are configured to map PCP values to exact traffic, and as frame arrives, the bridge directs them to the appropriate queue based on the value of the frame's PCP. [19]

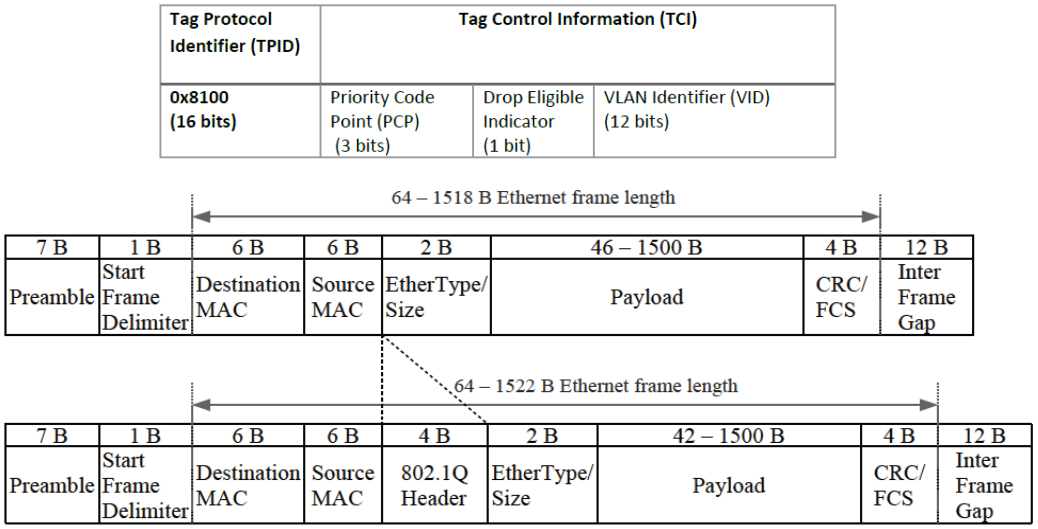



Figure 1-11: Extension of standard Ethernet frame

PCP involved just in layer two, as in some cases the end system could not recognize the time-sensitive stream. This issue has been resolved by adding per-stream filtering and policing, (specified in IEEE802.1Qci) to ensure non-time-sensitive streams which cannot use the TSN resources.

**IEEE 802.1Qav**

IEEE 802.1Qav was developed to meet the bandwidth of audio/video streams to preserve the bandwidth of best effort traffic, also known as Forwarding and Queuing for Time - Sensitive Streams (FQTSS). This sub-standard defines a credit-based traffic shaper and it used to provide continuous stream in audio/video systems. Credit-based shaper allows data to be transmitted if there are enough credits left. Credits are earned when it there is no transmission and consumed when data is being transmitted.

**IEEE 802.1Qbv**

The main function of TSN is deterministic schedule traffic in queues through switched networks, defined in IEEE802.1Qbv. This concept is known as the time-aware shaper (TAS). With the time-aware shaper concept it is possible to control the flow of queued traffic from a TSN enabled switch. Ethernet frames are identified and assigned to queues based on the priority field of the VLAN tag, shown in Figure 1-12. Each queue is defined

within a schedule, and the transmission of messages in these queues is then executed at the egress ports during the scheduled time windows. Other queues will typically be blocked from transmission during these time windows, therefore removing the chance of scheduled traffic being impeded by non-scheduled traffic. This means that the delay through each switch is deterministic and that message latency through a network of TSN-enabled components can be guaranteed. [17]

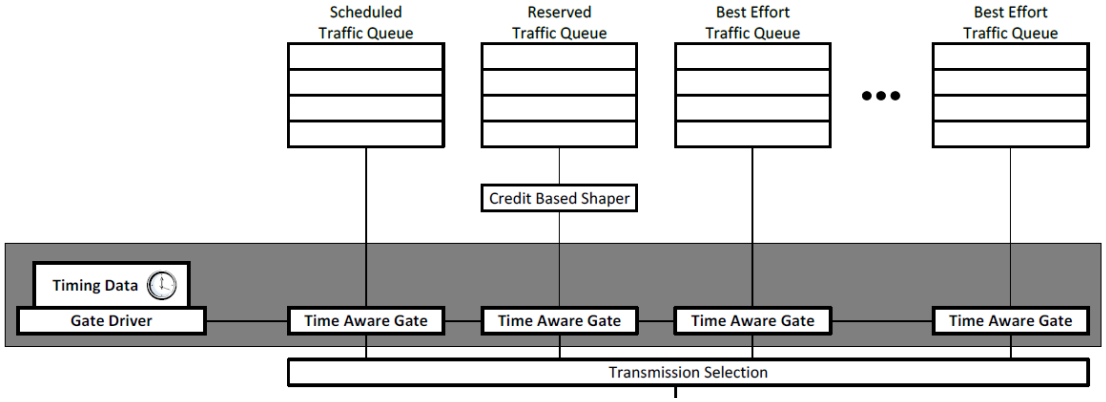



Figure 1-12: Time-Aware traffic scheduler [18]

The TAS introduces the concept of transmission gates. A gate has two states, ‘open’ and ‘closed’. The transmission selection process, which selects the next message for transmission at the egress, will only select messages from those queues whose gates are in the ‘open’ state. The state of the gates is also defined by the network schedule. Closing gates to non-scheduled traffic queues is another method of providing immunity to time-critical messages in order to guarantee bounded maximum latency through the network. While the TAS guarantees that critical messages are protected against interference from other network traffic, it does not necessarily result in optimal bandwidth usage or minimal communication latency. Where these factors are important, a pre-emption mechanism can be used. [17]

**IEEE 802.1Qbu and IEEE 802.3br**

IEEE 802.1Qbu works together with IEEE 802.3br (Interspersing Express Traffic Task Force) on a standardized pre-emption mechanism. This standard addresses the fact that the TAS described in IEEE 802.1Qbv avoids transmission jitter by blocking lower priority queues (for the duration of one maximum interfering frame) in advance of the transmission point of the critical frame. In cases where minimal latency for scheduled messages is desired, the TAS mechanism may not be the optimal solution. Therefore, on links where pre-emption as defined by IEEE 802.1Qbu is supported, the transmission of the standard Ethernet or jumbo frames can be interrupted in order to allow the transmission of high-priority frames, and then resumed afterwards without discarding the previously transmitted piece of the interrupted message. There are a few use cases in which a communication option to pre-empt an ongoing transmission is beneficial, e.g. to allow immediate transmission of a scheduled message and ensure minimum communication latency, or to facilitate maximum bandwidth usage on network links with a large amount of scheduled traffic. [17]

IEEE 802.1Qbu referred to the IEEE 802.3br, as described in sub-standard: “An M_CONTROL.request primitive is mapped to an IEEE 802.3 MA_CONTROL.request

primitive having the same parameters. If the MAC supports IEEE 802.3br Interspersing Express Traffic, then PFC M_CONTROL.requests are mapped onto the MAC control interface associated with the express MAC (eMAC). An IEEE 802.3 MA_CONTROL.indication primitive is mapped to an M_CONTROL.indication primitive having the same parameters." [21]

### 1.4.3 Network Configuration

Essential concept of TSN includes time synchronization and traffic shaping, requiring dynamic configuration based on application requirements. In this chapter a brief concept of configuration mechanisms will be describe. [19]

**IEEE 802.1Qat**

IEEE 802.1Qat was developed as a bandwidth reservation mechanism for credit-based traffic shaper described in 1.4.2. Stream Reservation Protocol (SRP) has been specified in this standard, which defines a plug-and-play configuration mechanism to set up and tear down stream reservations. [19]

"Using SRP, stream sources (Talkers) declare bandwidth requirements prior to transmitting stream data. These bandwidth requirements are communicated to the network using talker advertise messages that describe stream quality of service requirements including traffic class, data rate, and accumulated worst-case latency. A talker advertise message is propagated through the bridged network toward the potential Listeners as long as the bandwidth requirements can be met in bridges along the path. As the talker advertise message propagates, the accumulated latency is updated at each hop so that Listeners know the worst-case latency. If a bridge cannot support the stream's bandwidth requirement, it sends a talker failed message, and the stream reservation fails.

Listeners receiving talker advertise messages indicate their intent to receive the stream data by sending listener ready messages back to the Talker. As listener ready messages propagate through the bridged network, bridges lock down the resources needed to deliver the stream data according to its quality of service requirements. When the Talker receives a listener ready message, it can begin transmitting the stream. [19]"

**IEEE 802.1Qcc**

This sub-standard provides improvements of existing protocol such as SRP. These include support for more streams, configurable SR (Stream Reservation) classes and streams, better description of stream characteristics, support for Layer 3 streaming, deterministic stream reservation convergence, and UNI (User Network Interface) for routing and reservations. TSN configuration can be achieved statically by a network designer, or dynamically by a network service. "For example, the Path Computation Element (PCE) as developed by the IETF (RFC 4655) and the corresponding Path Computation Communication Protocol (RFC 5440) could be extended not only to find routes through a network, but also to configure the communication schedules for time - aware shaping [17]."

Figure 1-13 shows fully centralized configuration model, Talkers send stream data to Listeners and vice versa, over a bridged network.

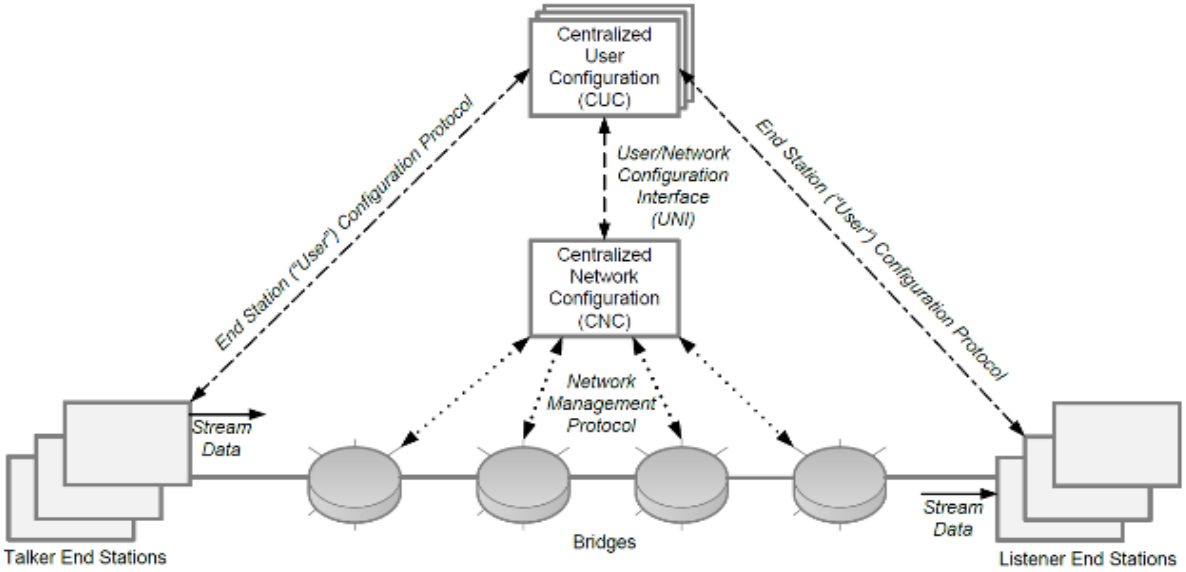



Figure 1-13: 802.1Qcc Fully Centralized Configuration Model [18]

## 1.5 Intel Stratix® V GX FPGA (5SGXEA7N2F45C2)

Intel StratixV GX is an important FPGA circuit for this thesis. Therefore, it will be used to implement the HW tester.

BittWare's S5-PCIe-DS (S5PE-DS) is a PCIe x16 card featuring two high-bandwidth, power-efficient Intel Stratix V GX or GS FPGAs. Designed for high-end applications, the Stratix V provides a high level of system integration and flexibility for I/O, routing, and processing. The S5PE-DS provides up to 64 GBytes of DDR3 SDRAM as well as options for RLDRAM3 and QDRII+. Providing additional flexibility are four front-panel QSFP cages, allowing 4 40GigE interfaces, 16 10GigE, or 4 QDR/FDR InfiniBand interfaces direct to the FPGAs built in PHYs for the lowest possible latency. With almost 2 million logic elements available (952,000 per FPGA), the board is ideal for high-performance computing, and with the reduced latency provided by the network interfaces, ideal for high frequency trading, military/government agency secure communications, and network processing applications. For more information [22].

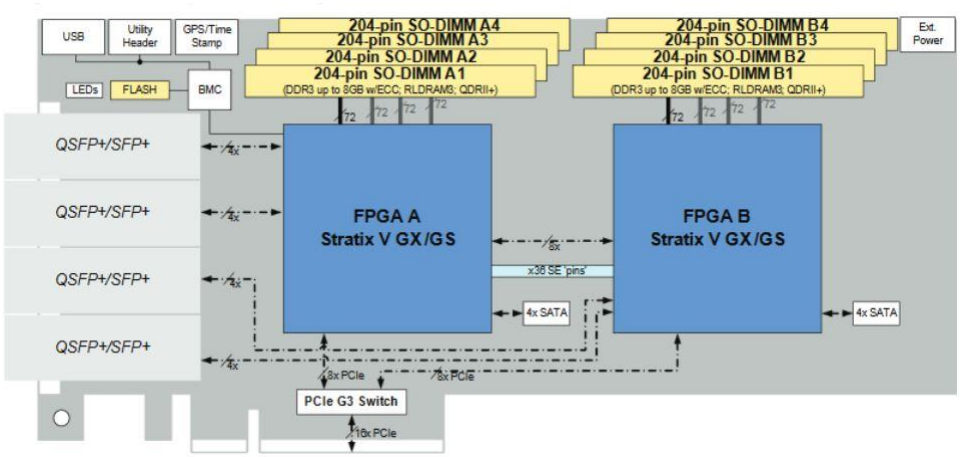



Figure 1-14: Block diagram Stratix V [22]

# 2 PRACTICAL PART

In accordance with the subject of the master thesis, this chapter will be divided in two parts. The first one will be consisted of analyses of IEEE TSN sub-standards and selection good enough to be tested in hardware which sub-standard will include. The second one will deal with regards to design of automatic tester for chosen sub-standard.

Architecture of tester will be created in order to be realized in HDL language. Moreover, architecture of tester will be in non-interactive (automatic) mode, which can provide proper output for analysis.

## 2.1 Analyzing or the IEEE TSN sub-standards

As shown in chapter 1.4 IEEE TSN sub-standards define functionality of whole real-time systems. Some sub-standards are realized in software, hardware and some of them have been combined. Table 2-1 shows assumptions of realization of key sub-standards into software vs hardware. Considering the fact that this semester project is related to hardware tester, the sub-standards belonging to software implementation will not be used in further analyses.

Table 2-1: Relationship between SW related and HW related TSN sub-standards

| TSN sub-standard SW related | TSN sub-standard HW related |
|---|---|
| IEEE 802.1AS | IEEE 802.1Qbv |
| IEEE 802.1Qca | IEEE 802.1Qbu |
| IEEE 802.1Qcc | IEEE 802.1Qch |
| | IEEE 802.1Qci |
| | IEEE 802.1CB |

The key functions of IEEE TSN sub-standards are independent, meaning that they can be used theoretically in any combination. Some sub-standards cannot be used without others. For example, sub-standard IEEE 802.1Qbv (Scheduled Traffic) cannot be used without IEEE 802.1AS or IEEE 802.1ASrev (Time Synchronization), because timing - aware protocol needs information about the precise time. Another, example is IEEE 802.1Qci (Filtering and Policing) which needs a lot of other sub-standards, shown in Figure 2-1.

IEEE 802.1Qbu and IEEE 802.3br are one of the sub-standards that could be proper to be implemented independently. Moreover, these sub-standards have been published by IEEE TSN task group, so special permission from IEEE to use them in this thesis is not necessary.

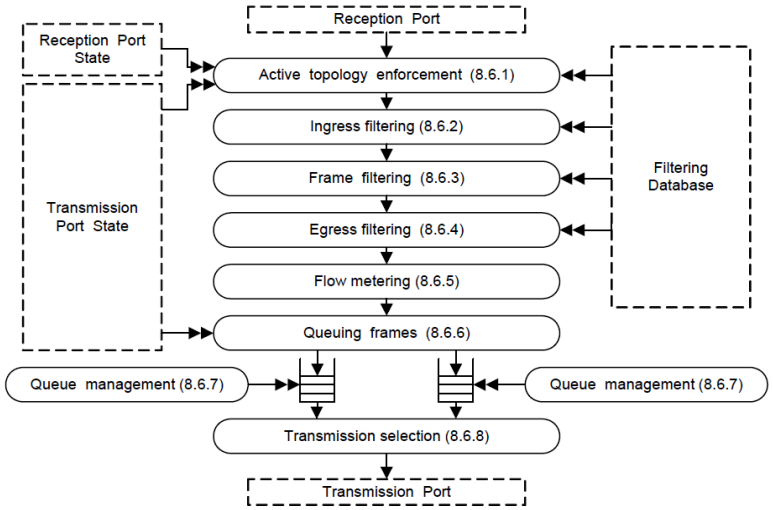



Figure 2-1: Filtering and policing [22]

### 2.1.1 IEEE 802.1Qbu and IEEE 802.3br

As explained in chapter 1.4.2 IEEE 802.1Qbu (in further text just Qbu) main function is to pre-empt packet transaction, this sub-standard is referred to IEEE 802.3br (in further text just BR) sub-standard. These sub-standards are defined as a relationship between express transaction and pre-emption transaction. More precisely, if the TSN network uses more clients consisting of high priority client(s) (time sensitive clients) and client(s) with low priority (e.g. best effort clients), it is necessary to add prioritisation into that network (time sensitive have higher priority than best effort). Low priority clients need to wait for the proper time slot, which decreases dramatically the speed of the transaction during the transmission of big packets. If the client has pre-emption, which means that packet will be split into smaller pieces (fragments), it means that it has effective use of network.

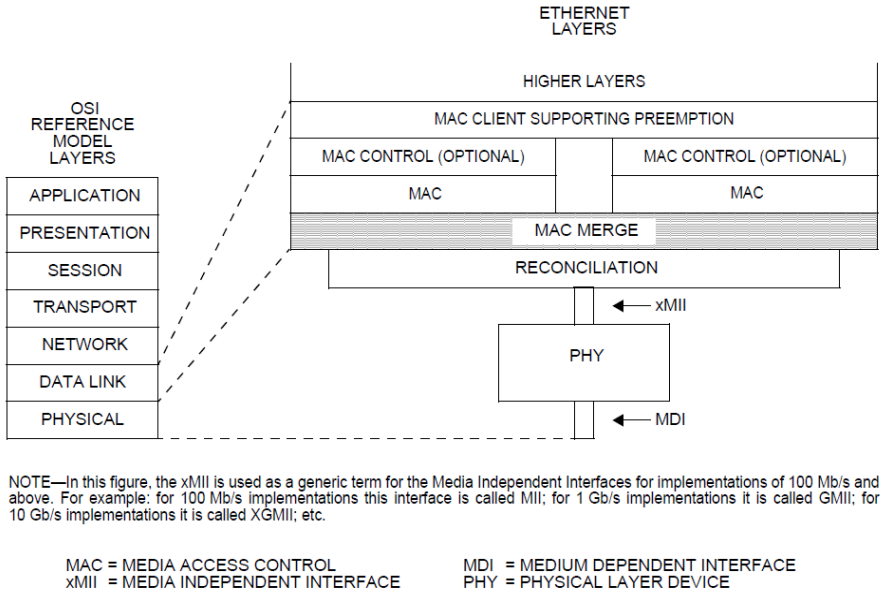



Figure 2-2: Relationship of MAC Merge sublayer to the ISO/IEC Open Systems [23]

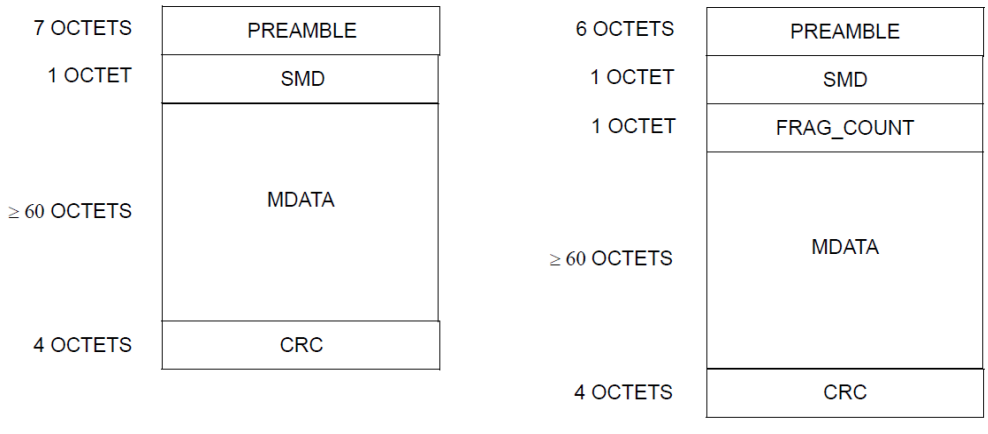
According to the BR sub-standard pre-emption must be implemented in the data link ISO/OSI referent model, between MAC client and reconciliation sublayer, shown in Figure 2-2. [23] This sub-standard consists of three parts:

- **express Media Access Control (eMAC):** The instance of a Media Access Control sublayer that is the client of a MAC Merge sublayer service interface that handles express traffic. [23]
- **preemptable Media Access Control (pMAC):** The instance of a Media Access Control sublayer that is the client of a MAC Merge sublayer service interface that handles preemptable traffic. [23]
- **MAC Merge sublayer:** An optional sublayer that supports interspersing express traffic with preemptable traffic by attaching an express Media Access Control (eMAC) and a preemptable Media Access Control (pMAC) to a single Physical Signaling Sublayer (PLS) service. [23]

**MAC Packet**

Figure 2-3 shows difference between preemptable and express MAC packet. That difference is in the preamble length, SFD/SMD and FCS. Express packet has standard Ethernet structure: 7 bytes of preamble, 1 byte SMD (same as SFD), data (min 60 bytes, max 1500 if it is not jumbo packet) and 4 bytes of FCS (CRC). Preemptable MAC has three types of packet structure [23]:

1) a complete preemptable packet. It looks the same as express, excepting SMD, which will be explained in the further text. [23]
2) an initial fragment of a preemptable packet. It looks the same as previous complete preemptable packet type. [23]
3) a continuation fragment of a preemptable packet, which has 6 bytes of preamble length, fragment count, continues SMD, FCS for whole packet. [23]




a) b)

Figure 2-3: a) express packet, a complete preemptable packet or an initial fragment of a packet
b) containing a continuation fragment of a packet

**Start mPacket Delimiter (SMD)**

"The value of the SMD indicates whether the packet contains an express packet, the start of a preemptable packet (initial fragment or complete packet), or any of continuation fragments of a preemptable packet." All valid SMD values are defined in Table 2-2. [23]

Table 2-2: SMD values [23]

| Packet type Notation Frame | Notation | Frame count | count Value |
|---|---|---|---|
| verify packet | SMD-V | - | 0x07 |
| respond packet | SMD-R | - | 0x19 |
| express packet | SMD-E | - | 0xD5 |
| preemptable packet start | SMD-S0 | 0 | 0xE6 |
| | SMD-S1 | 1 | 0x4C |
| | SMD-S2 | 2 | 0x7F |
| | SMD-S3 | 3 | 0xB3 |
| continuation fragment | SMD-C0 | 0 | 0x61 |
| | SMD-C1 | 1 | 0x52 |
| | SMD-C2 | 2 | 0x9E |
| | SMD-C3 | 3 | 0x2A |
| fragment count | FRAG_COUNT0 | 0 | 0xE6 |
| | FRAG_COUNT1 | 1 | 0x4C |
| | FRAG_COUNT2 | 2 | 0x7F |
| | FRAG_COUNT3 | 3 | 0xB3 |

SMD for express packet has the same format as SFD for standard ethernet packet, see Table 2-2.

An mPacket carrying a complete (non-fragmented) packet has SMD-S packet start octet according to the Table 2-2.

The first fragment of "fragmented" packet has the same value as over non-fragmented packet (SMD-S). SMD for continuation fragments are defined as SMD-C value and include an additional fragment counter octet (frag_count) following the SMD-C. The frag_count is a modulo-4 counter that increments for each continuation fragment of the preemptable packet. The frag_count protects against mPacket reassembly errors by enabling detection of the loss of up to 3 packet fragments. [23]

"The SMD in an mPacket carrying a complete (non-fragmented) preemptable packet or any of the fragments of a preemptable packet also indicates the frame count. Information about the frame count prevents reassembling an invalid packet if the final mPacket of one preemptable packet and the initial fragment of the next preemptable packet are lost. The frame count is a modulo-4 count. The term "SMD-S" refers to any of

the four SMD values (SMD-S0, SMD-S1, SMD-S2, and SMD-S3) in an mPacket carrying the start of a preemptable packet. The term "SMD-C" refers to any of the four SMD values (SMD-C0, SMD-C1, SMD-C2, and SMD-C3) in an mPacket carrying a continuation fragment of a preemptable packet." [23]

"Two additional SMD values, SMD-V and SMD-R, identify mPackets used to verify that a link can support the preemption capability." [23]

**MAC Merge sublayer transmit behaviour when preemption is disabled**

"When preemption is disabled, the packets presented by the pMAC and eMAC pass through the MAC Merge sublayer without alteration, i.e., the MAC Merge sublayer transmits packets rather than mPackets. If both the eMAC and pMAC have packets ready to transmit and no packet is being transmitted, the eMAC packet is transmitted. If a pMAC packet is being transmitted and the eMAC has a packet to transmit, the packet from the eMAC is transmitted after transmission of the pMAC packet completes." [23]

**Benefit of preemption**

"The preemption capability is most useful at lower operating speeds. For example, the duration of a 2000 octet packet on a 100 Mb/s link is 160 μs and on a 1 Gb/s link is 16 μs. The time to transmit a maximum length packet is an upper bound on the additional delay before a MAC Client can send an Express frame when the preemption capability is not used. At higher operating speeds this additional delay gets smaller in proportion to the speed, reducing the advantage of the preemption mechanism." [23]

## 2.2 Overall Architecture

Different techniques for testing have been shown previously in the thesis. BIST has an advantage for detecting bugs online (in real-time) and offline, but in case that the application needs to turnoff this part of the module, testing engineer has to remove/disable it from the design. BIST are related just for one module and it is hard to reuse this part of module. On the other hand, ATPG (more in 0) are good for reusability, but not to debug every component in real-time. Testing by conversation of simulated vectors takes a lot of computing time, but it is good for achieving requirements. To achieve higher speed of testing, reusability, and faster implementation PA technique in combination with other techniques (BIST, ATPG…) could be used.

In chapter 2.1 the analysis of the TSN sub-standards and reason why Qbu will be tested is shown. Tester will be created to fulfill these sub-standards, which are deeply explained in that chapter. Figure 2-4 shows how whole SoC design for Qbu sub-standard looks like with hardware tester which supports this sub-standard. On the left side is NIOS II (with memories) processor supported by Intel FPGA, connected to interconnect Avalon Memory Map IP, also supported by Intel FPGA. In the middle of the block diagram are Monitor and Generator Objects, which are PA-HWT. On the right side is the MAC design consisted of eMAC (rx/tx) and pMAC (rx/tx). All of these components (IPs) are created, first two are commercial IPs. MAC which supports Qbu sub-standard has been done by Filip Kliment master project, more in [24]. This master project is related to the PA-HWT.

### 2.2.1 Overview Intel intellectual properties (Intel's IP)

NIOS II processor has been chosen as an optimal solution because the development kit is made by Intel FPGA. "This processor is general purpose RISC processor core with the following features:

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- Optional shadow register sets
- 32 interrupt sources
- External interrupt controller interface for more interrupt sources
- Single-instruction 32 × 32 multiply and divide producing a 32-bit result
- Dedicated instructions for computing 64-bit and 128-bit products of multiplication
- Optional floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter" [25].

Memories for supporting this processor could be used: Flash memory, SRAM memory or SDRAM memory.

Interconnect IP is basically set of smaller modules, that are connected into proper way supported by Intel Quartus® tool. Fundamental modules are network on chip, bridges, memory mapped modules… [26]

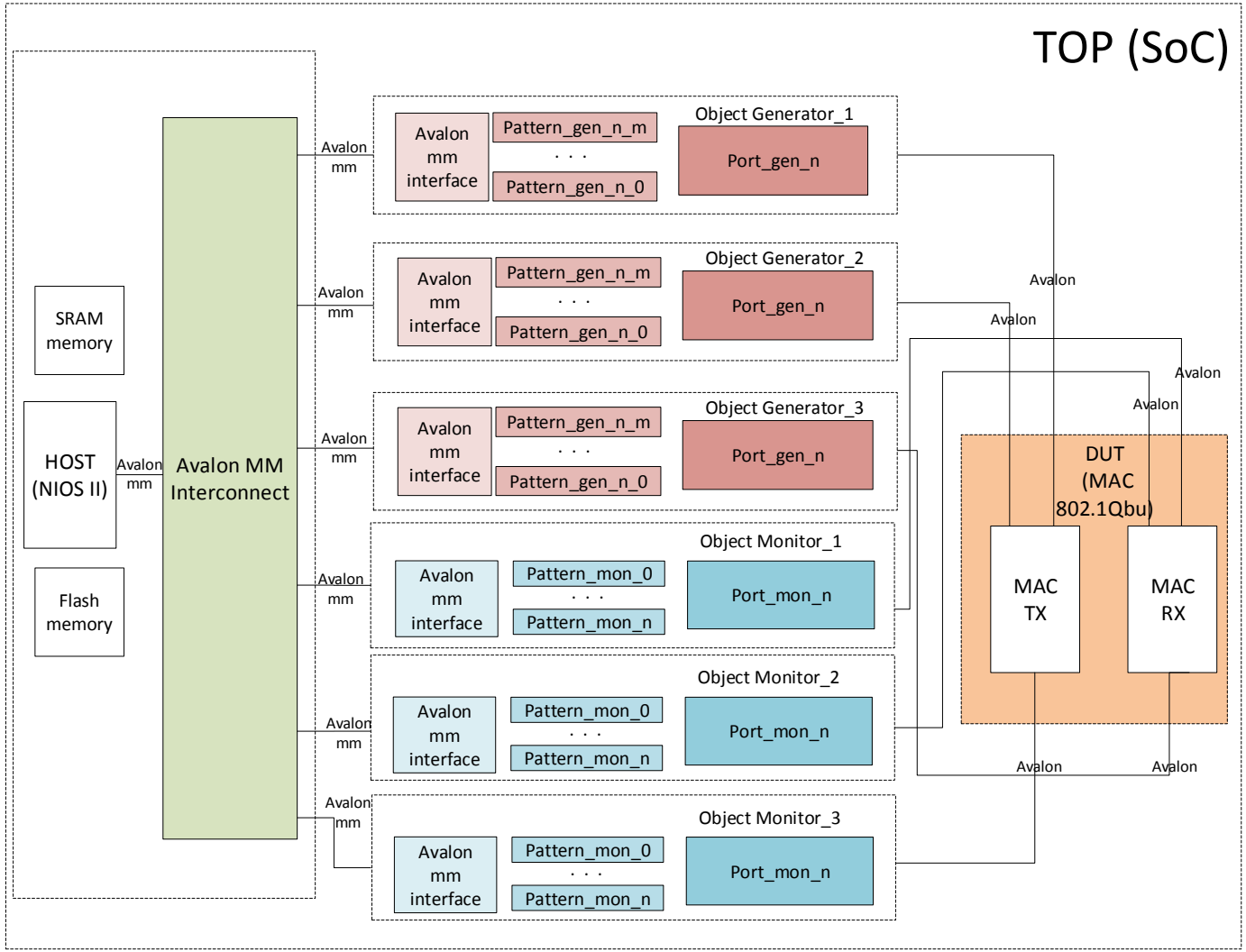



Figure 2-4: Hardware implementation of complex SoC design

### 2.2.2 Protocol Aware hardware tester

Figure 2-5 shows the structure of the protocol aware hardware tester (in further text PA-HWT). This tester has two objects (generator and monitor) which needs to fulfil Qbu sub-standard. Generator's function is to create proper stimuli (random or non-random), and to transfer it into DUT according to standard. Monitor's function is to receive egress frames and to check frame correctness according to the DUT requirements. IO control interface has a function of translating commands from software (SW) and to use those commands inside the objects in automatic mode. The precise description of these components will be described in further chapters.

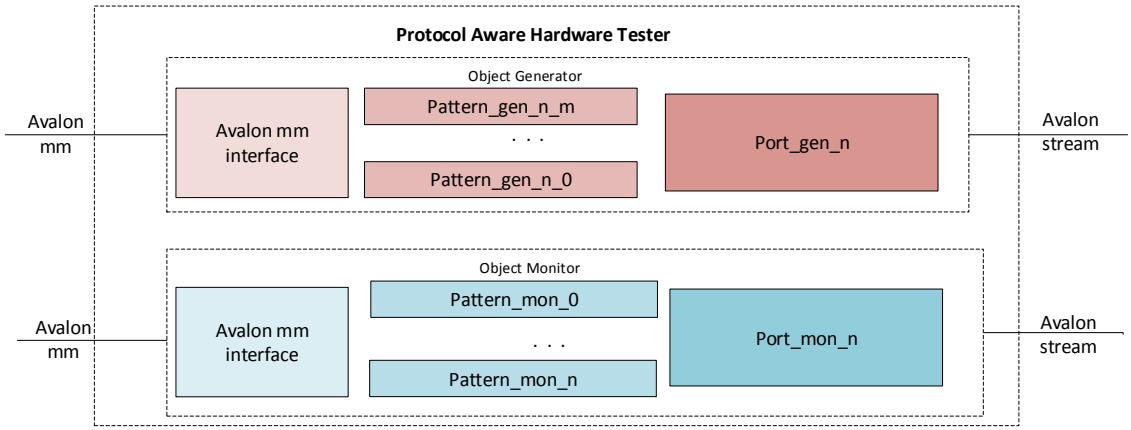



Figure 2-5: Protocol Aware Hardware Tester

### 2.2.3 Interfaces

Interfaces used in this project is mostly chosen because of available kit, described in chapter 1.5. Other reasons are standards free to be used without any licence, reusable commercial IPs and relative simplicity of usage.

Interfaces used in this project are: Avalon Memory Map (Avalon MM), Avalon Streaming (Avalon ST) and Ethernet.

**Why using Avalon instead, the Ethernet?**

Avalon ST could be a choice because of the vendor, but Qbu sub-standard is Ethernet standard. To reconcile these standards, it was created a combination of the two. Input/output signals are used as Avalon ST (sop, eop, data, valid, ready), streams sent or received through these signals are standard Ethernet, improved by Qbu TSN sub-standard.

**Avalon stream**

Avalon ST is defined in source and sink fashion. In this thesis will be used following terms and concepts:

- "Source and Sink Interfaces and Connections—When two components connect, the data flows from the source interface to the sink interface. The Avalon Interface Specifications calls the combination of a source interface connecting to a sink interface a connection.

- Backpressure—Backpressure allows a sink to signal a source to stop sending data. Support for backpressure is optional. The sink uses backpressure to stop the flow of data for the following reasons:
    - When the sink FIFOs are full
    - When there is congestion on its output interface
- Transfers and Ready Cycles—A transfer results in data and control propagation from a source interface to a sink interface. For data interfaces, a ready cycle is a cycle during which the sink can accept a transfer." [27]

Source signals used in this thesis are data, valid, sop, eop. Sink signals used in this thesis is ready. Ready signal has been permanent set on true for the input and output of the objects, the inside part of the design is used as control signal. Table 2-3 shows how Avalon ST interface could be replaced with ethernet mii interface.

Table 2-3: Ethernet and Avalon ST signals connection

| Avalon ST source | Ethernet |
|---|---|
| - | TX_CLK/RX_CLK |
| valid | TX_EN/RX_EN |
| data | TXD/RXD |
| sop | TX_EN/RX_EN (rising edge) |
| eop | TX_EN/RX_EN (falling edge) |

**Avalon Memory Map**

Avalon MM interface implement read and write function between master(s) and slave(s). This interface is used for communication between processor (master) and objects (slaves). According to the vendor documentation [25], NIOS II has 32 bit Avalon MM output. Interconnect module could bridge this 32 bit Avalon MM interface to 8 bit Avalon MM. According to this fact, it was proved as good choice for this design.

Avalon MM has 6 different transfers mode:

1. Typical Read and Write Transfers
2. Transfers Using the waitrequestAllowance Property
3. Read and Write Transfers with Fixed Wait-States
4. Pipelined Transfers
5. Burst Transfers
6. Read and Write Responses

In the thesis will be used burst transfer, because a burst executes multiple transfers as a unit, rather than treating every word. Figure 2-6 and Figure 2-7 show burst transaction.

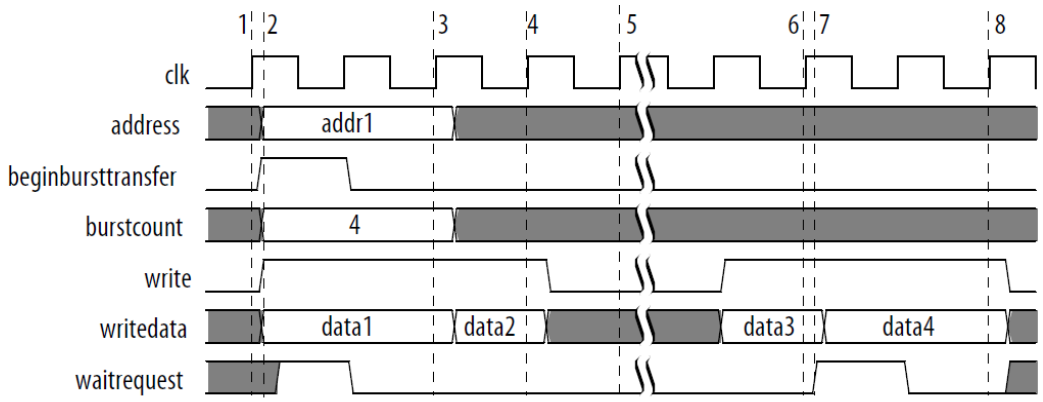



Figure 2-6: Avalon memory map write burst transaction

"The numbers in this timing diagram mark the following transitions:

1. The master asserts `address`, `burstcount`, `write`, and drives the first unit of `writedata`.
2. The slave immediately asserts `waitrequest`, indicating that the slave is not ready to proceed with the transfer.
3. `waitrequest` is low. The slave captures `addr1`, `burstcount`, and the first unit of `writedata`. On subsequent cycles of the transfer, `address` and `burstcount` are ignored.
4. The slave captures the second unit of data at the rising edge of `clk`.
5. The burst is paused while `write` is deasserted.
6. The slave captures the third unit of data at the rising edge of `clk`.
7. The slave asserts `waitrequest`. In response, all outputs are held constant through another clock cycle.
8. The slave captures the last unit of data on this rising edge of `clk`. The slave writes burst ends." [27]

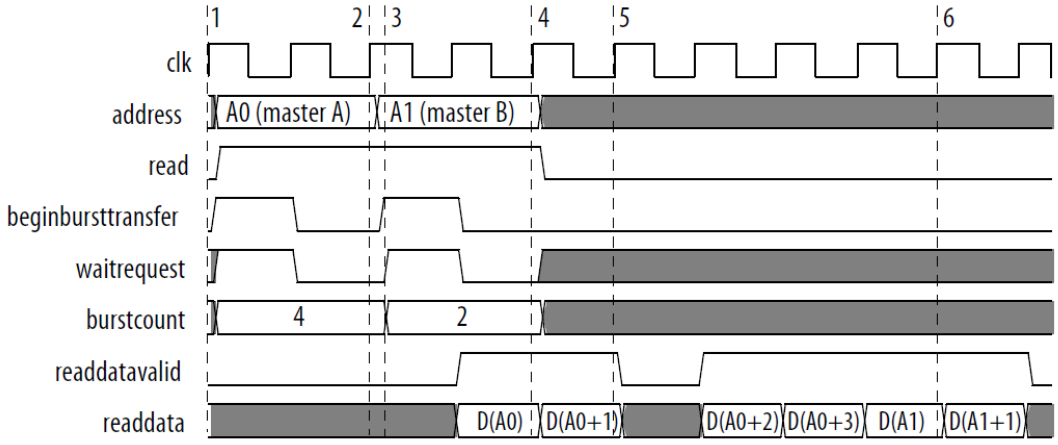



Figure 2-7: Avalon memory map read burst transaction

"The numbers in this timing diagram, mark the following transitions:

1. Master A asserts `address (A0)`, `burstcount`, and `read` after the rising edge of `clk`. The slave asserts `waitrequest`, causing all inputs except `beginbursttransfer` to be held constant through another clock cycle.
2. The slave captures `A0` and `burstcount` at this rising edge of `clk`. A new transfer could start on the next cycle.
3. Master B drives `address` (`A1`), `burstcount`, and `read`. The slave asserts `waitrequest`, causing all inputs except `beginbursttransfer` to be held constant. The slave could have returned read data from the first read request at this time, at the earliest.
4. The slave presents valid `readdata` and asserts `readdatavalid`, transferring the first word of data for master A.
5. The second word for master A is transferred. The slave deasserts `readdatavalid` pausing the read burst. The slave port can keep `readdatavalid` deasserted for an arbitrary number of clock cycles.
6. The first word for master B is returned." [27]

## 2.3 IO control interface

Chapter 1.2.3 has already explained main concept of PA or "hybrid" technique for testing. First issue to be resolved is a relationship between SW and HW. In this case we can observe this issue from the point that we have two layers of abstraction. Higher layer is SW, lower one is HW. From the SW point of view objects are virtual modules processing an action according to the commands. That action could be simple, e.g. to send one frame, or complex, e.g. send frames in dedicated time slot. From the HW point of view objects are physical modules waiting for commands to start the action. Also, action could be simple, e.g. set in frames, and send them, or complex, e.g. set n frames, half of them will be sent as priority (express) frames, and half of them will be pre-empted (fragmented) frames and will send those frames according to the dedicated time slot.

Host can approach to the objects according to the interface map, that is defined in the appendix B.

### 2.3.1 Command packet structure

The issue explained above is resolved with defining a protocol and sub-protocol. Figure 2-8 shows this solution. Protocol structure has header and payload. Header consists of START and END command. Payload contains a set of sub-protocols. The sub-protocol has one-word (16 bits, or 2 bytes) header and one-word payload. Headers of sub-protocol are defined commands (command constant) which are not allowed to be used for a payload. Commands constants are shown the following appendix A.

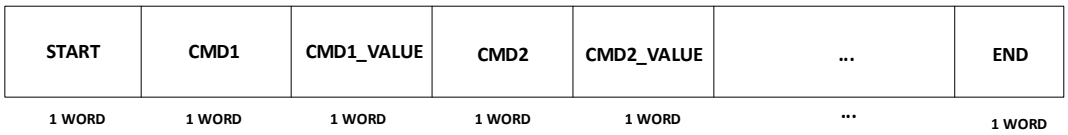



Figure 2-8: Protocol and sub-protocol structure

SW "see" object with dedicated commands, for control of that object SW should send commands defined by protocol. E.g. START, CMD1, CMD1_VALUE, END.

HW are waiting for a set of commands (e.g. same as above). INIT command sets input registers by command registers and then waits for END command that triggers a transaction (start with transaction). Deeper explanation will be in the sub-chapter 2.3.3.

Figure 2-9 shows block diagram of IO control interface. This module consists of command FIFO, command decoder and status object. Command FIFO is used to store input commands which are sent by Avalon MM master (e.g. processor). Command decoder is reading store commands and decoding according to that command constant. Status object has been created to store status information (fatal errors, number of sent frames, error headers…).

**Constrains**

One-word of payload in the sub-protocol is limited in the case that SW needs to send complex command value.

**Possible improvements**

One-word header can be updated to send two-words of header, which will consist of command header and the length of command values, then SW can send complex set of command values.

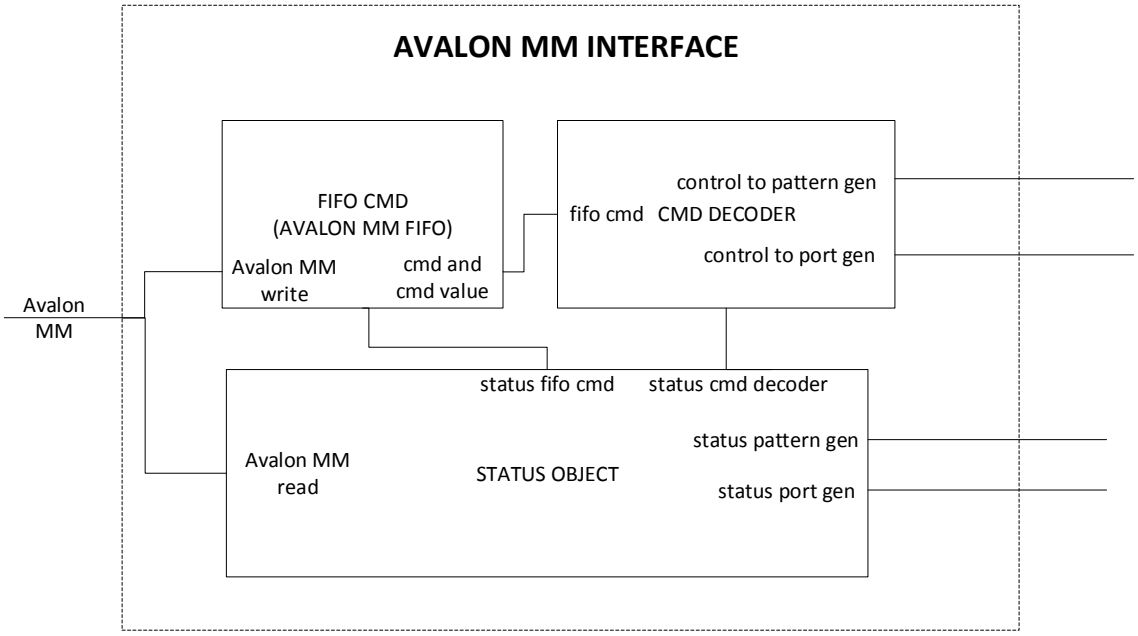



Figure 2-9: IO control interface block diagram

## 2.3.2 Command FIFO

Command FIFO has been created to receive data from Avalon MM master burst transfers, or it can be called Avalon MM slave. The FIFO got burst count at the beginning of the transaction and then reads input data according to that count. If FIFO is full, then Avalon MM transaction is stopped.

**Constrains**

FIFO is designed to store just 32 bytes, because the max burst count is 32. Max burst count has set because of the address width, which is 5 bits.

**Possible improvements**

It will be useful to increase FIFO depth, because it does not represent a huge problem with resources on chip.

## 2.3.3 Command Decoder

Command decoder (CD) reads data from command FIFO and then gathers those bytes and creates a data word (2 bytes). According to data word value, function is the one to choose whether that word is command or command value. Command value must not be the same as any other command constant. This module is setting the output register according to the command and command value. e.g. If CD receives FRAME_NO command and X"0004" as command value, it will set output register "frame_no" to 4.

Figure 2-10 shows CD finite state diagram. At the beginning CD remains in the IDLE state, which means that this module is waiting for commands. After that, when it receives START, then sets default values and waits for another sub-protocol. Sub-protocols could be in any combination, as it shown in Figure 2-10. When CD receives END command, it stops with reading from FIFO and it sets start signal ("start_sending" or "start_receiving") to start transaction in the modules. If CD receives just START and END, it will provide default transaction (e.g. sending of 1 frame, with incremental payload generator and with frame length 64 bytes). All command constants are shown in the appendix A.

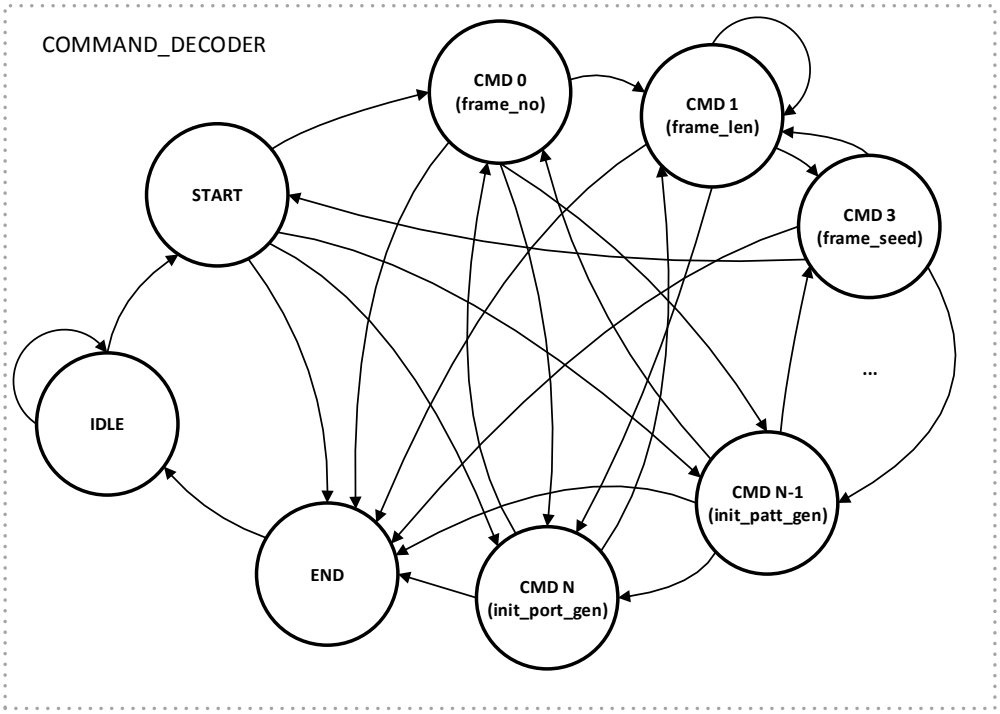



Figure 2-10: Command decoder finite state machine (output registries have been added just as example)

**Constrains**

Commands value must not be the same as any other command constant.

**Possible improvements**

It would be convenient to have start of sub protocol header and end of sub protocol header defined; in that case, the same constant for command and command value could be used. However, two values will not be allowed.

### 2.3.4 Status object

Status object is the module which is mapping status registers, counters and memories whit Avalon MM interface. Every object has defined addresses for status information. Programming interface map is defined in appendix B.

Table 3-1 shows monitor status registers. These registers are divided into two types. The first one is fatal errors and the second one is regarded to status counters (correct frame

counter, error frame counter…). Fatal error represents bug in the tester. The recommendation is to stop with testing when these errors are occurred.

Table 3-2 shows fatal errors inside of the generator object, for now these errors are just set; add into modules, but still it is not used in the logic. For implementation of these status register inside of logic, design needs precise requirements and verification plan.

**Constrains**

- Generator status errors are not added into logic.
- Counters used for status monitor have data length just 1 byte.
- Clearing counters after host read, does not supports in all counter.

**Possible improvements**

- Define precise (low level) requirements, according to these requirements test design and add error register inside of the design
- Increase counters length to 4 bytes, because of the long testing (sending and receiving millions of frames)
- Add logic inside of the design that will be clear counters after reading

## 2.4 Generator object

Generator's basic function is to create and send stimuli, according to the test case. With new protocol aware view, generator should be more flexible or "hybrid". "Hybrid generator" means that generator should have two layers of abstraction, SW and HW. SW generator is object which has some behaviour. These behaviours can be used according to the commands. In the case that SW needs to change behaviour of an object, HW needs to be updated. HW generators are created according to the specific functions and requirements and it needs to be fully controlled by SW.

Figure 2-11shows generator object block diagram and how generator object was implemented in the HW. Generator object is consisted of four parts: IO control interface, generator pattern (1 and 2) and generator port. Input module IO control interface is used to communicate with SW (more in chapter 2.3). This module decoded commands and commands values, according to these inputs finite state machine (FSM) sets output registers. These registers are sent to proper module (generator pattern or generator port). After receiving an END command, output "start" signal is set, which means that transaction will start in all modules in the same time. During transaction every module has BISTs or testing registers connected to object status module (address mapping module). Generator pattern has basic function to generate stimuli random or non-random, according to commands. Generator port is using to manage input data streams in order to have proper output (in this thesis Ethernet with BR).

Sub-standard BR has defined two types of transaction, express (priority) and pre-empted (fragmented). Regarding this sub-standard object, generator has two generators pattern. One of them should emulate priority transaction, other one should emulate fragmented transaction. SW will be the one to select which pattern is priority and which one is fragmented. Generator port needs to choose which data stream has priority according to commands. Commands can define waiting time, e.g. priority frames will send each 100 clock cycles, fragmented frames need to be sent each 50 clock cycles; priority frames are waiting until time runs out. If the time runs out in the same clock cycle "priority" frame has bigger priority and "fragmented" frame is waiting until priority processed.

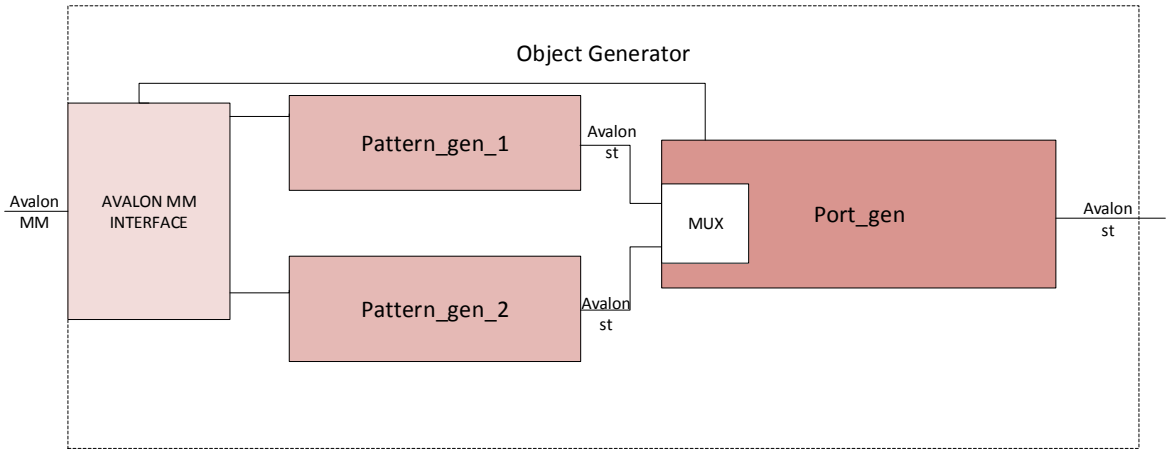



Figure 2-11: Generator object

### Commands

Commands are the same for all generators, pattern or port. If the object has multiple patterns (as it has in this design) and if an issue occurs to the pattern, commands should be sent. In order to resolve this issue, there were created two commands INIT_PATTERN_GEN and INIT_PORT_GEN. These commands are the ones to decide which generator pattern or port will be sent to previous commands. Despite the fact that there is only one generator port, command (INIT_PORT_GEN) needs to be sent, for initializing of port.

**Example how to send set of commands into two patterns:**

```
-------------------------------------------------
START;
FRAME_LENGTH;
X"0401";
FRAME_NO;
X"0003";
...
INIT_PATTERN_GEN;
X"0001"; -- set pattern generator 1
-------------------------------------------------
FRAME_LENGTH;
X"0301";
FRAME_NO;
X"0006";
...
INIT_PATTERN_GEN;
X"0002"; -- set pattern generator 2
END;
-------------------------------------------------
```

### Constrains

The communication between patterns and port is realised by Avalon ST, with ready and valid control signals. In the case that transaction should be controlled, it needs to reckon on delay of 4 cycles. This delay occurred because of the fully synchronous design (every output is registered). E.g. when ready signal is set, it needs one cycle to occur on output and then, another module receives ready signal (one more cycle). Then sets/resets output transaction (one more cycle), and at the end transaction is occurred in the first module (fourth cycle).

### Possible improvements

To change interface, AXI streaming may be a good choice. See AXI streaming documentation [28].

### 2.4.1 Generator pattern

Generator's pattern primary function is to generate stimuli and to prepare data (to calculate CRC, to store in the FIFO/memory…). Command registers have been sent from IO control interface (command decoder) and these registers are inputs for this module.

Generator pattern includes five modules and one important function, shown in Figure 2-12. These modules are: control pattern, linear feedback shift register (LFSR), linear generator, pattern mux, FIFO Avalon ST and one function cyclic redundancy check (CRC).

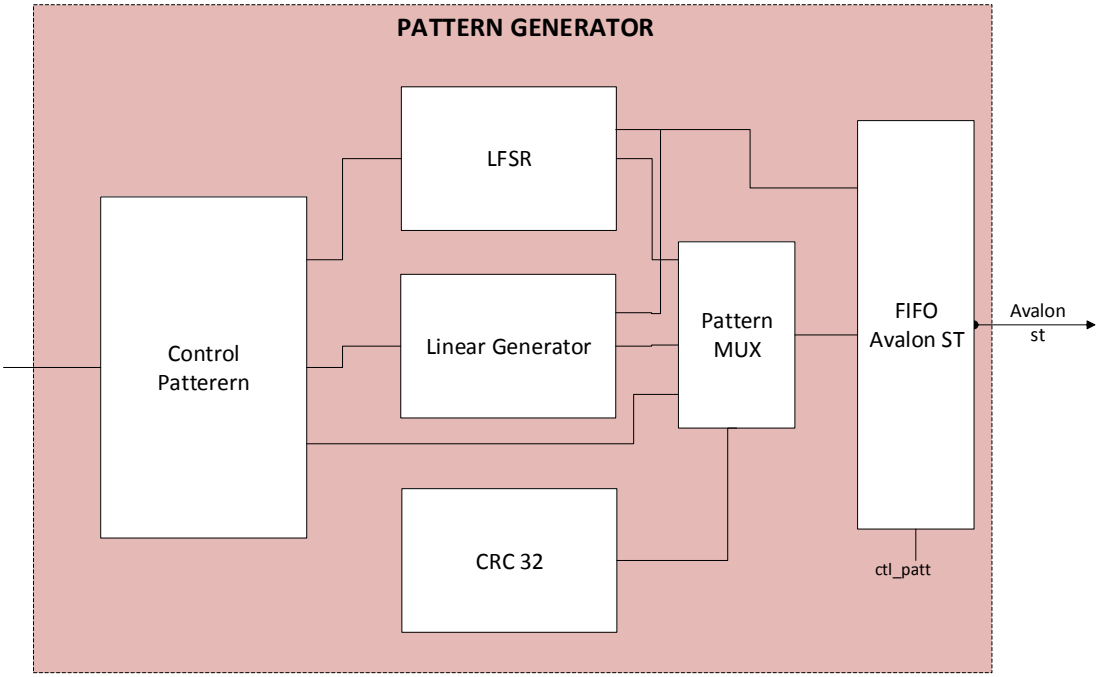



Figure 2-12: Generator pattern

Control pattern receives commands from command registers and then provides functions according to the FSM shown in Figure 2-13. After reset, this module is waiting to be initialised (as it describes in chapter 2.3; waiting for INIT command). When INIT have been received, FSM "jump" to INIT state and control registers are fulfilled with input registers. If some of input registers are not set, these control registers will use default value. Control modules are waiting to be "start_sending" set (command END) and that is how the transaction starts. If there is at the least one "frame_no", the next state will be SEND_ONE_FRAME. In this state outputs are set, and stimuli start to generate. Stimuli can be generated inside of LFSR (random generator) or in linear generator (linear incrementation of data, e.g. 3,6,9…). If function CRC32 is enabled, then all data being sent into FIFO are calculated. Data length are set by command (min ethernet packet could be 64, max 1500 bytes). When generator send last byte of data, FSM is going to state SEND_CRC32 (if this function is enabled, other vice it is going to CHECK_FRAME_NO). All data plus CRC (4 bytes) is being stored into FIFO Avalon ST. Then FSM is going to state CHECK_FRAME_NO, while control module checks whether pattern sends all the frames. If pattern was not sent, then the FSM would be changed into SEND_ONE_FRAME and all transaction would be continuous again. Otherwise, it is going to INIT state in order to wait for a new set of commands.

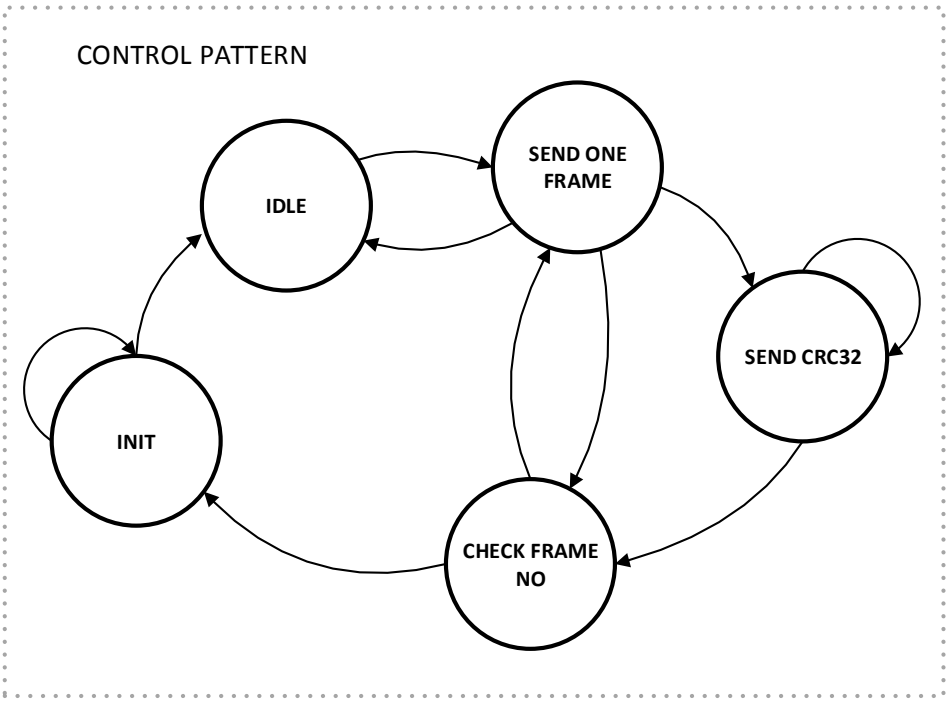



Figure 2-13: Control pattern FSM

LFSR is a very simple random generator working as a shift register and XORing the outputs according to dedicated pattern. The first indicated value calls seed and it can be set by seed command. This random generator is deterministic, which is good for checking operation. The generator and the monitor will have the same value if it initializes by the same seed. LFSR has an advantage of not demanding on logic, because it uses just few logic cells.

Figure 2-14 shows basic random operation of LFSR generator.

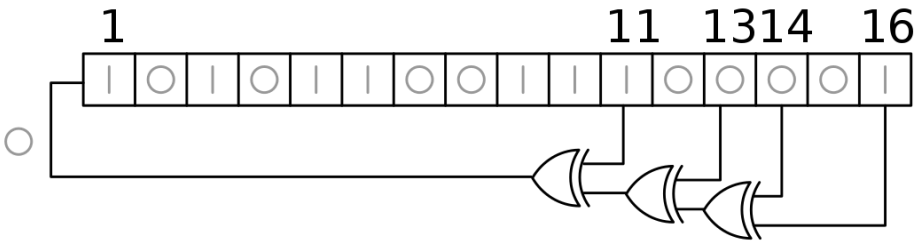



Figure 2-14: Linear feedback shift register basic function [29]

Linear (non-random) generator basically is a counter. After it gets "starting_sequence" (starting value), this value is incremented by eight. Pattern multiplexer selects inputs by control signal. FIFO Avalon ST stores generated frames with CRC and sends them into generator port through Avalon ST. FIFO can store max 4 Kbytes of data (data+crc).

**Commands**

Commands for pattern generator are:

- FRAME_NO – number of sending frames
- FRAME_LENGTH – frame length
- FRAME_SEED – seed for random generator
- SEED_INCREMENT – starting value for non-random generator
- RANDOM_DATA – enable random generator
- LINEAR_DATA – enable linear generator
- CRC_EN – enable CRC function

**Constrains**

- LFSR sometimes repeats values and it is also not allowed to start with zero value.
- Linear generator has just incrementing by eight.
- FIFO Avalon ST can store just one frame, because it is storing sop/eop in one register. If frame is not read, new frame will rewrite sop/eop.

**Possible improvements**

- It can use few LFSR in cascade or "Jumping register" generator, which is more demanding on logic, see [30].
- Add more options to increment by increment, until it reaches some value and then repeats it…
- Store starting indexes into array (small vector), then according to it reads frame read sop/eop.

### 2.4.2 Generator port

Generator port should receive data stream from generator(s) pattern and it should process it into output, in the proper way. Moreover, this module should implement standard used for DUT. Standard BR specifies express (priority) transaction and pre-empted (fragmented) transaction. This module needs to create proper test scenario to fully test DUT.

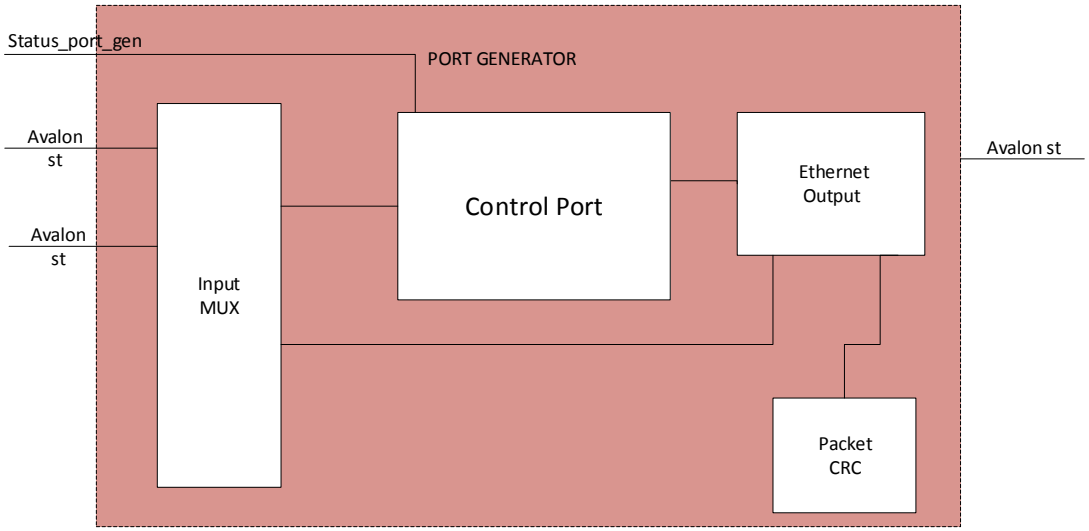



Figure 2-15: Generator port

In previous chapters, it has been explained that in this design will be used two generators pattern; one for priority transaction and another one for fragmented. These patterns can change the roles by new set of commands. Generator port includes three modules and one important function. These are following: input multiplexer, control port, ethernet output and packet CRC function. Input multiplexor selects from which generator pattern it will be read. Control port manages whole generator port. Ethernet output transfers input data stream to ethernet stream. When frame is ready, ethernet output sends preamble, SMD, FRAG_COUNT (if needed), data, CRC (if CRC is enabled) and terminates it.

Essential function of control module is timing management. Figure 2-16 shows how this module should read data stream from patterns. FRAME_WAIT_PRIORITY and FRAME_WAIT_FRAG commands set waiting time for priority and fragmented frames. Starting time is the same for both patterns. The first frame appeared in FIFO Avalon ST starts to lose time (counter decrements), when arrives the second, it also starts to lose time. In the case that both frames arrive in the same time (which is normal situation, because of the same pattern logic) it starts to lose time in the same clock cycles. Figure 2-16 shows one example when both frames arrive in the same time. If waiting time for fragmented frame is less than for priority frame, in that case the fragment will be sent first. The “losing” time is same for bought frames, according to this rule, after the fragment is sent, time of priority will decrease for left time (T2). Fragment two also loses its time during the T2, and then waits for time T3 and so on.

With this function port could emulate different idle time for frames, which can occur

in real networks. This different time can be random for DUT, but for monitor it is dedicated and that allows better testing.

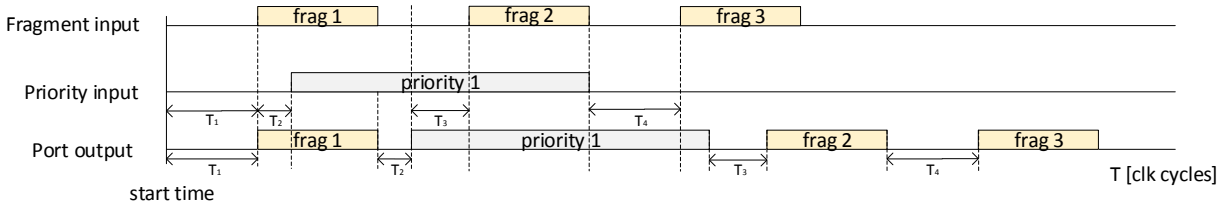



Figure 2-16: Timing diagram for fragments and priority frames

FSM for control port have been shown in Figure 2-17. The first state is INIT and it is reading from command registers for this module. When "start_sending" is set, FRAME_WAIT state has been selected and starts with decreasing counters for priority and fragmented frames (if there are prepared in the FIFO). After the transmission of all frames, FSM "jump" to INIT state to wait for the new set of commands.

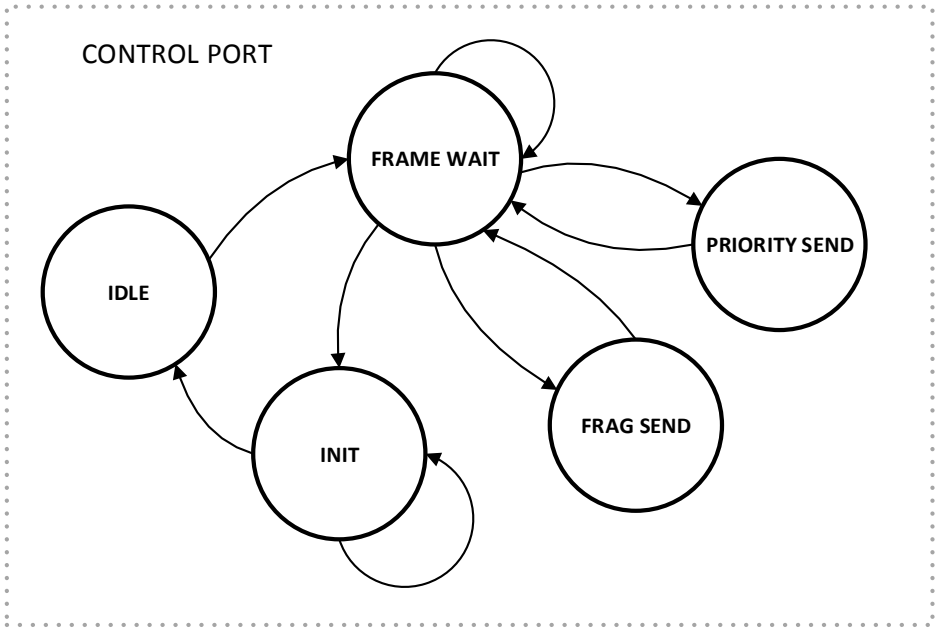



Figure 2-17:FSM for control port

When fragments are sending, control module sets CRC counting for whole packet. Packet is big according to the command NO_OF_FRAGMENT. Preamble length, SMD – S or SMD – C, FRAG_COUNT has added based on fragment number. E.g. if fragment pattern sends 10 frames, and NO_OF_FRAGMENT is 3; every third frame will be new packet.

**Commands**

- SET_FRAG – enable fragment pattern
- PRIORITY_INPUT – set which pattern will be priority
- NO_OF_FARGMENT – fragments per packet
- PACKET_CRC_EN – packet CRC enable
- FRAME_WAIT_PRIORITY – waiting time for priority frames
- FRAME_WAIT_FRAG – waiting time for fragmented frames
- PREAMBLE_LENGTH – set preamble length (default value is 7, for continued fragments is default value -1)
- SFD – start frame delimiter constant
- SMD_I – start mPacket delimiter
- SMD_C – continuous mPacket delimiter
- TERMINATED – terminated constant

**Constrains**

- Fragments must have same length in the packet.
- Waiting time is same between priority/fragmented frames.

**Possible improvements**

- For different fragment length needs to be added special function in the pattern generator.
- Different waiting time between frames could be reach by randomisation of wait command.

## 2.5 Monitor object

Core function of the monitor is to receive data stream and to check whether data is correct. Monitor object has same "hybrid" point of view as generator object (see chapter 2.3). "Hybrid monitor" means that monitor should have two layers of abstraction, SW and HW. SW monitor is object which has some behaviour. These behaviours can be used according to the commands. In the case that SW needs to change behaviour of object, HW needs to be updated. HW monitor are created according to the specific functions and requirements, and it needs to be fully controlled by SW.

This is nice example of reusability of the protocol aware concept. Basically, generator object and monitor object have same IO control interface and pattern modules, with different port modules which are related to ethernet protocol.

Figure 2-18 shows monitor object block diagram, how monitor object was implemented in the HW. Monitor object is consisting of four parts: IO control interface, monitor pattern (1 and 2) and monitor port. Input module IO control interface is used to communicate with SW, more in chapter 2.3. This module decoded commands and commands values, according to these inputs finite state machine (FSM) sets output registers. These registers are sent to proper module (monitor pattern or monitor port). After receiving an END command, output "start" signal is set, which means that transaction will start in all modules in the same time. During transaction every module has BISTs or testing registers, which is connect with object status module (address mapping module).

Monitor port receives ethernet stream into content-addressable memory that select relabel output pattern. This module also checks ethernet header according to BR sub-standard. Monitor pattern receives one type frames, priority or fragmented one and compare data with generated data by themselves.

Status output has two types, first one is fatal errors regarding to monitor's modules. Second one is regard to check of transaction (number of received correct frames, number of received error frames, number of received packets, error headers…). These counters are store in their own modules, but that modules are connected to status object which addressed Avalon MM to these counters. When SW reads status data, counters clears.

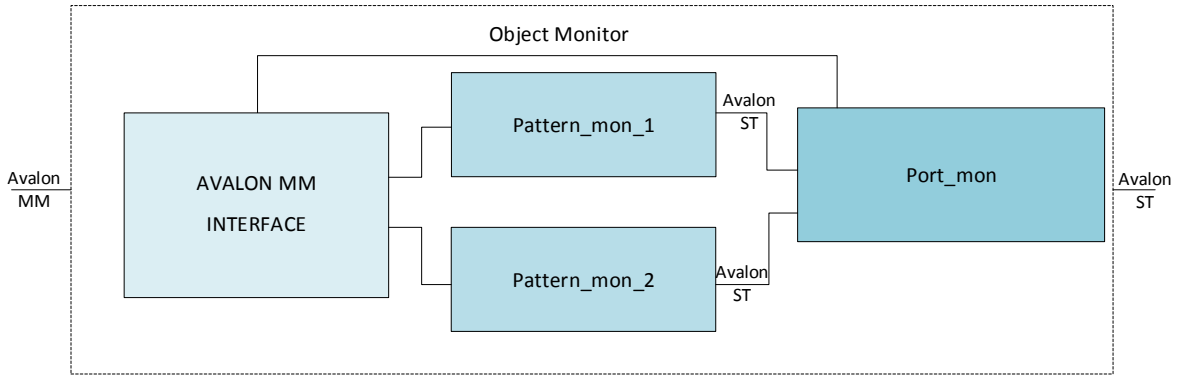



Figure 2-18: Monitor object

**Commands**

Commands is same as generator's see chapter 2.4, with suffix MON.

**Example how to send set of commands into two monitors pattern:**

```
-------------------------------------------------
START;
FRAME_LENGTH;
X"0401";
FRAME_NO;
X"0003";
...
INIT_PATTERN_MON;
X"0001"; -- set pattern monitor 1
-------------------------------------------------
FRAME_LENGTH;
X"0301";
FRAME_NO;
X"0006";
...
INIT_PATTERN_MON;
X"0002"; -- set pattern monitor 2
END;
-------------------------------------------------
```

**Constrains**

- Constrain which regards to Avalon ST interface is same as in generator object see chapter 2.4.
- Counters width, which is 1 byte length, it's small value for testing of huge number of frames, just 256 frames can be checked.

**Possible improvements**

- First one has explained in chapter 2.4.
- Increase counter to be two-words length. It would be nice to have memory for error frames.

### 2.5.1 Monitor port

Monitor port has to receive ethernet BR frames and to send data into related pattern. Idle, preamble, SMD and terminate ethernet constants need to be send into header check. Header checker compares these constants according to the input command. If the constants are not set by commands this module use default values.

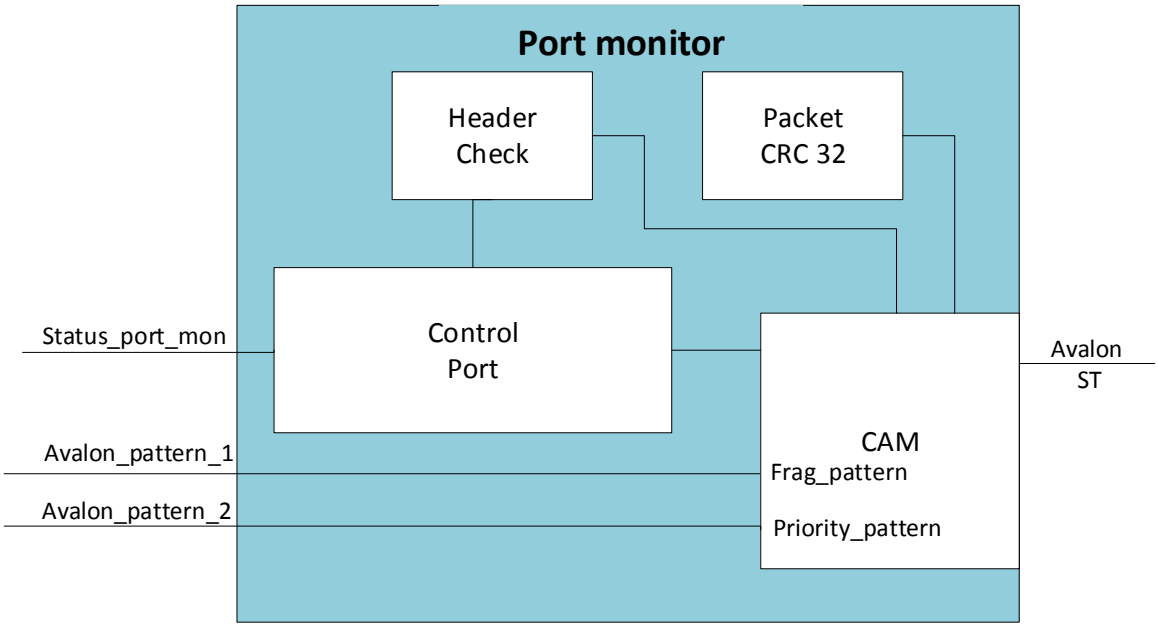



Figure 2-19: Monitor port

Figure 2-19 shows monitor port block diagram (HW implementation), it is including content-addressable memory (CAM), control port, header check and packet CRC function. Control port is using to initialize input register with command register when the PORT_INIT sets. Header check has comparing function, it compares constants, check the packet number and prepare SFD-S/C and FRAME_COUNT for next expected frame. SFD-S/C and FRAME_COUNT have changed according the BR standard. All comparing outputs are stored in the counter, which is connected to status object. When SW reads that counters, status object sends information and counters are cleared.

CAM are hardware search engines, mostly used for routers. CAMs use input data as an address for lookup function, the address lookup function examines the packet's destination address and chooses an output port associated with that address. “An example of a simplified routing table is displayed in Table 2-4. All four entries in the table are 5-bit words, with the don't care bit, X, matching both a 0 and a 1 in that position. Because of the $X$ bits, the first three entries in Table 1 represent a range of input addresses, i.e. the entry on Line 1 indicates that all addresses in the range of $10100_2$—$10111_2$ are forwarded to port $A$ [31]”.

Table 2-4: Simplified routing table

| Line No. | Address (Binary) | Output port |
|---|---|---|
| 1 | 101XX | A |
| 2 | 0110X | B |
| 3 | 011XX | C |
| 4 | 10011 | D |

In this design CAM will used BR sub-standard, therefore preamble length and SFD, SMD-S/C are used for routing input frames to the selected output pattern. Priority pattern needs to be chosen by SW (command), the pattern which is not priority will be used as fragment pattern. If the fragmented pattern does not have SET_FRAG command set, that pattern is turned off.

**Commands**

- PRIORITY_OUTPUT – set priority pattern
- SET_FRAG – enable fragment pattern
- NO_OF_FARGMENT – set packet length (how much fragments have packet)
- PACKET_CRC_EN – enable packet CRC function
- PREAMBLE_LENGTH – set preamble length (default value is 7, for continued fragments is default value -1)
- SFD – add comparing constant
- SMD_I – add comparing constant
- SMD_C – add comparing constant
- TERMINATED – add comparing constant

**Constrains**

This module has issue with counters, it was explained in chapter 2.5

### 2.5.2 Monitor pattern

Monitor pattern primarily receives frames and then compares it with generated frame by itself. Ethernet data with CRC is received into monitor pattern and during this receiving monitor pattern generates expected data with CRC. When both (received and prepared) streams are ready, scoreboard compares them. If the whole frame is correct, "correct_frame_counter" increases, otherwise "error_frame_counter" increases. After reading these counters by SW, counters are cleared.

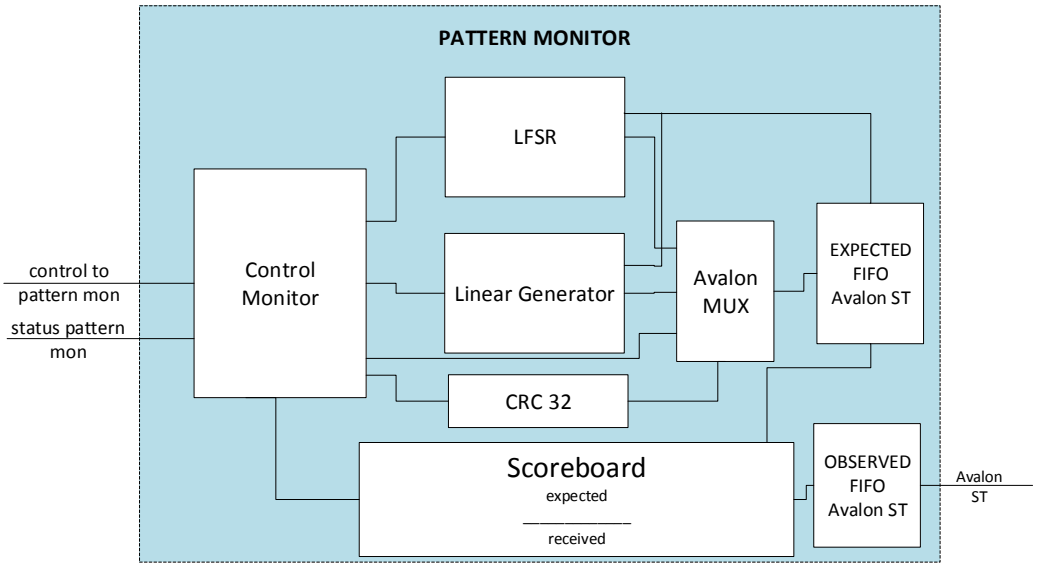



Figure 2-20: Monitor pattern

Figure 2-20 shows monitor pattern block diagram. Basically, monitor pattern is very good example of reusability, since more than 80% are reused modules from generator pattern. This module contains: control monitor, linear feedback shift register, linear generator, Avalon mux, expected FIFO Avalon ST, observed FIFO Avalon ST and scoreboard. Reused components are: control monitor, linear feedback shift register, linear generator, Avalon mux and expected FIFO Avalon ST. For more information about these modules, see chapter 2.4.1.

Input data stream from generator port saves in the observed FIFO Avalon ST and those data have been read by scoreboard. Scoreboard waits for new data in observed FIFO and when data arrive, it triggers reading from both FIFOs (expected and observed). This design has been created in order to expect that generated data by monitor pattern is much faster than receiving data stream. According to this expectation, comparing starts when data come into observed FIFO. Scoreboard has just two options to check; whether data is correct or not correct. Status counters are connected to status object module mapping with IO control interface.

**Commands**

- FRAME_NO – number of sending frames
- FRAME_LENGTH – frame length
- FRAME_SEED – seed for random generator
- SEED_INCREMENT – starting value for non-random generator
- RANDOM_DATA – enable random generator
- LINEAR_DATA – enable linear generator
- CRC_EN – enable CRC function
- SCOREBOARD_EN – enable scoreboard

**Constrains**

- Constrain regarding to Avalon ST interface is the same as in the generator object see chapter 2.4.
- Comparing has been created to compare data and CRC (whole input data stream).

**Possible improvements**

- Possible improvements regarding to Avalon ST interface are the same as in the generator object see chapter 2.4.
- Buffer N cycles input data, when eop sets compare separately CRC.

## 2.6 Verification of protocol aware hardware tester

Verification of digital design is necessary and very complex considering the fact that domain itself contains many technologies, languages, and methodologies. Figure 2-21 shows verification flow. In the very beginning of the fazes, a verification team needs to define verification approach, establish verification environment, write requirements …

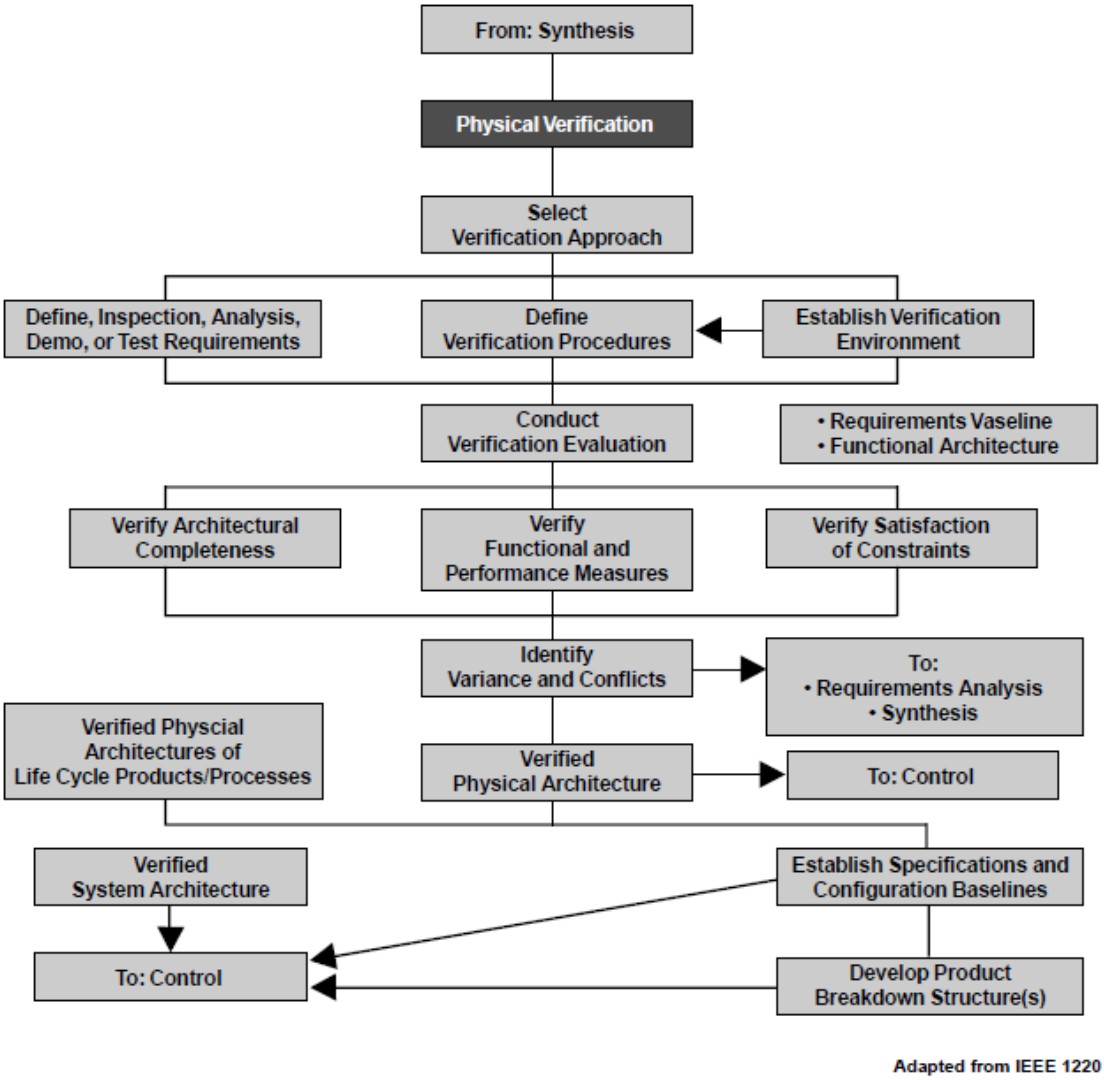



Figure 2-21: Verification flow chart [32]

This project has been created as demo to show a new PA-HWT technique. Regarding this fact, complex verification is not needed. In this thesis only basic system requirements have been set demonstrating how to send and to receive multiple ethernet (with BR sub-standard) frames. VHDL environment has been chosen, because whole design is created in this language.

Figure 2-22 shows block diagram of VHDL test bench for PA-HWT simulation. It contains Avalon MM bus functional model (BFM) and DUT (pa_hw_tester). Avalon MM BFM is used to create Avalon MM burst transaction (see chapter 2.2.3). Avalon MM Master Mon and Avalon MM Master Gen are being used to emulate communication

between DUT and processor (e.g. NIOS II).

DUT is connected from one side with these BFMs, and from the other loopback has been created. With this environment basic “smoke” test can be created, which needs to send and to receive frames.

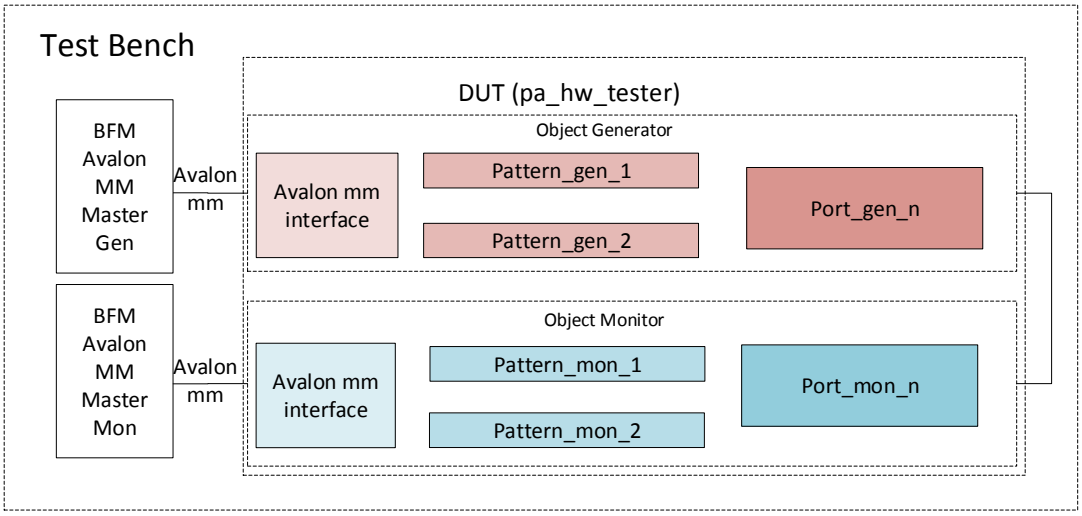



Figure 2-22: Test bench for PA-HWT

### 2.6.1 Avalon Memory Map Master Bus Functional Model

Avalon MM BFM has been created to emulate master with basic functions: burst write, burst read, and compare burst data. Burst write or read is constrained to max 32 bytes, because of the address width (5 bits).

When test procedure (TP) calls (for burst write add writing address and data; for burst read add reading address) function in BFM, BFM sets proper command. According to that command BFM processes proper function/procedure. Functions or procedures set output signal and transaction is start.

### 2.6.2 Test procedures

Test procedures have been created to initialize BFM, to send commands into generator and monitor and to read status registers. Basic function of PA-HWT have been tested with two test procedures, “tp_smoke_test_000” and “tp_error_status_000”.

First one “tp_smoke_test_000” send same number of frames as it expected. “Correct” counters show that was received expected number of frames.

Second test procedure (tp_error_state_000) sends frames with different frame length then it was expected. This test shows that PA-HWT “error” counters work properly according to the expectation.

## 2.7 Results

Requirements for PA-HWT have been to send and receive multiple ethernet (with BR sub-standard) frames. This function has been approved with the test procedures (explained in the chapter 2.6.2). Code coverage for PA-HWT are relatively big according to the fact that this design was tested with two test procedures. Expression Terms has been covered the most 95%. The lowest cover was on branches around the 66%. Coverage results are stored in the project/verification/results.

Figure 2-23 shows Quartus flow summary after the implementation PA-HWT into the Stratix V GX FPGA (5SGXEA7N2F45C2) development kit. Logic utilization (in ALMS) are 126,351 (54 %), that occupied more than half chip area. Huge number of ALMS can be expanding that FIFO memories have been created on the register bases. Implementing the memories by vendor can help this number to be reduced.

**Flow Summary**

<<Filter>>

| | |
|---|---|
| Top-level Entity Name | ea_top_pa_hw_tester |
| Family | Stratix V |
| Device | 5SGXEA7H3F35C3 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 126,351 / 234,720 ( 54 % ) |
| Total registers | 233117 |
| Total pins | 90 / 664 ( 14 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 512 / 52,428,800 ( < 1 % ) |

Figure 2-23: Quartus flow summary

# 3 CONCLUSION

Different test techniques have been explained in this thesis, some of which are still at the beginning. By increasing the Ethernet speed and developing a new real-time system testing needs to be transferred to HW in order to achieve high speed requirements. Separate writing of test cases is still limited in simulation and physical testing, so in the future this will be another milestone for the automation of verification and testing.

Analyses of IEEE 802.1Q TSN group of standards has shown that time synchronization is the main function for the proper operating of that system. Traffic Scheduling demonstrates how real-time systems receive/transmit time aware protocols. Centralisation of network configuration shows how the whole system should be controlled by using one protocol type. Some of TSN standards are more related to SW, some to HW. Theoretically they can be used in any combination, but some of them refer to each other. According to these facts IEEE 802.1Qbu was chosen, as it can be implemented without any other standard. This sub-standard have created to avoid transmission jitter by blocking lower priority queues in advance of the transmission point of the critical frame. Sub-standard has defined as a relationship between express transaction and pre-emption transaction.

Different testing techniques have various approach on how to test a digital system. Protocol Aware (PA) is one of techniques that can test systems and in real-time can change their components. Another advantage is mixing with other techniques (e.g. BIST), with the implementation in HW which can achieve high speed. Reusability and fast implementation are very important to decrease time to market, which is supported by PA technique.

Tester architecture was designed in PA technique, it has two approaches; SW and HW. Objects in SW abstraction have dedicated behaviours. These behaviours can be used according to the commands. In the case that SW needs to change behaviour of an object, HW needs to be updated. From the other hand objects in HW need to be created according to the specific functions and requirements. Thus, HW object must be fully controlled by SW.

PA tester have been created to support IEEE 802.1Qbu sub-standard, so that supports express transaction and pre-emption transaction. The top-level of generator and monitor objects are created in the same way, it is including four modules: IO control interface, two patterns and one port. IO control interface module has been built to resolve an issue which regards to communisation between SW and HW. It receives SW packet and decode that packet according to defined protocol and sub-protocol. Sub-protocol contains command and command value. These commands and commands values are used to control whole object modules. Pattern is used to generate/compare random or non-random data streams. Two patterns were implemented to support two types transaction; express and pre-emption. Port are specified to communicate with DUT according to IEEE 802.1Qbu sub-standard. Generator port used to select and send frames according to priority and time slot. Monitor port receives frames into content-addressable memory that select relabel output pattern. IO control interface is mapped outputs with tester status counter or status vector.

Functional verification has been done in VHDL test environment. Test procedures have been processed by ModelSim tool. Measured cod coverage shows that two test procedures can covered FEC Expression Terms, which means that this tester can be used as demo version of PA test technique. For commercial usage it needs complete verification plan, requirements plan…

Tester was implemented using QUARTUS II in FPGA Stratix V circuit. This has shown that the tester occupies 126,351 ALM‘s, which is less than 54 %. The reason is that FIFO memories have been written as registers, replacing them by generated IP place on chip will be reduced.

The goal of this master thesis was fully covered. IEEE 802.1Q standard have been analyzed, as reliable sub-standard have been chosen IEEE 802.1Qbr. Tester outputs are sufficient detailed or determined causes of failure for chosen sub-standard.

# BIBLIOGRAPHY


[1] C. Manjula and D. Jayadevappa, “FPGA BASED WIRELESS UNIVERSAL DIGITAL AUTOMATED TEST EQUIPMENT,” no. WIRELESS UNIVERSAL DIGITAL AUTOMATED TEST EQUIPMENT, 2016.

[2] T.-H. Chen, “HIGH-SPEED, LOW COST TEST PLATFORM USING FPGA TECHNOLOGY,” Georgia Institute of Technology, Atlanta, 2016.

[3] C. E. Gray, “An FPGA Based Architecture for Native Protocol Testing of Multi-Gbps Source-Synchronous Devices,” Georgia Institute of Technology, Atlanta, 2012.

[4] Anon., “Testing The Future For Over Fifty Years,” Teradyne, [Online]. Available: http://www.teradyne.com/about-teradyne/teradyne-history. [Accessed 11 11 2017].

[5] Anon., “Technical Guide to JTAG,” XJTAG, 12 11 2017. [Online]. Available: https://www.xjtag.com/about-jtag/jtag-a-technical-overview/.

[6] Anon., “JTAG Interface, TAP Test Access Port,” IEEE, 12 11 2017. [Online]. Available: https://www.electronics-notes.com/articles/test-methods/boundary-scan-jtag-ieee1149/jtag-interface-tap-test-access-port-connector.php.

[7] R. Dekker, F. Beenker and L. Thijssen, “REALISTIC BUILT-IN SELF-TEST FOR STATIC RAMs,” Delft, 1989.

[8] V. Neerkundar, “What’s The Difference Between ATPG And Logic BIST?,” 10 03 2014.

[9] R. Obermaisser, “Time-triggered communication,” in *Time-Triggered Communication*, New York, CRC press, 2011, p. 575.

[10] T. Kirkland and R. M. Mercer, “ALGORITHMS FOR AUTOMATIC TEST PATTERN GENERATION,” Dallas, 1988.

[11] D. C. Keezer, D. Minier, M. Paradis and B. F., “Modular Extension of ATE to 5 Gbps,” Georgia, 2004.

[12] Anon., “Field Programmable Gate Array (FPGA),” Xilinx, 2017. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html. [Accessed 18 11 2017].

[13] Anon., “SEMICONDUCTOR ATE,” Intel, [Online]. Available: https://www.altera.com/solutions/industry/test-and-measurements/applications/semiconductor/tes-ate.html. [Accessed 23 11 2017].

[14] Anon., “Semiconductor Automated Test Equipment,” XILINX, [Online]. Available: https://www.xilinx.com/applications/test-measurement/automated-test-equipment.html. [Accessed 23 11 2017].

[15] A. C. Evans, “The New ATE: Protocol Aware,” in *INTERNATIONAL TEST CONFERENCE*, Irvine, 2007.

[16] Anon., “International Technology Roadmap for Semiconductors 2.0,” ITRS, /, 2015.

[17] Anon., “IEEE TSN (Time-Sensitive Networking): A Deterministic Ethernet Standard,”

TTTech, 2015. [Online]. Available: https://www.tttech.com/download/technologies/deterministic-ethernet/time-sensitive-networking/file/1c49796102083798d7f3968fa4c3c2ae3b59a471/. [Accessed 30 11 2017].

[18] Anon., "Project list," Group of company, 1 12 2017. [Online]. Available: http://www.802tsn.org/home/project-list.

[19] E. Gardiner, "Theory of Operation for TSN-enabled Systems," Avnu Alliance, 2017.

[20] IEEE group of autors, "IEEE Standard for Local and metropolitan area networks-Bridges and Bridged Networks," IEEE, New York, 2014.

[21] I. g. o. autors, "Bridges and Bridged Networks Amendment 26: Frame Preemption," in *IEEE Computer Society*, New York, 2016.

[22] Datasheet, "Stratix V GX FPGA Development Board," Intel, 09 2015. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/manual/rm_svgx_fpga_dev_board.pdf. [Accessed 30 10 2017].

[23] I. g. o. autors, "IEEE 802.3br Amendment 5: Specification and Management Parameters for Interspersing Express Traffic," in *IEEE Computer Society*, New York, 2016.

[24] KLIMENT, F. NÁVRH VYBRANÉ ČÁSTI STANDARDU IEEE 802.1Q . Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 55 s. Vedoucí diplomové práce doc. Ing. Lukáš Fujcik, Ph.D..

[25] Anon, "Nios II Classic Processor Reference Guide," 17 6 2016. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf. [Accessed 5 5 2019].

[26] I. F. g. o. authors, "Applying the Benefits of Network on a Chip Architecture to FPGA System Design," [Online].

[27] I. F. g. o. autors, "Avalon® Interface Specifications," 26 09 2018. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf. [Accessed 16 05 2019].

[28] Anon, "AXI Reference Guide," 11 3 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Accessed 15 5 2019].

[29] Anon, "Linear-feedback shift register," 2 8 2004. [Online]. Available: http://iml.univ-mrs.fr/~kohel/tch/MATH3024/Tutorials/Solutions/tutorial08.pdf. [Accessed 10 5 2019].

[30] "Cascade Jump Controlled Sequence Generator and Pomaranch Stream Cipher," [Online]. Available: http://www.ecrypt.eu.org/stream/p2ciphers/pomaranch/pomaranch_p2.pdf.

[31] K. Pagiamtzis, "Content-Addressable Memory Introduction," 01 03 2006. [Online]. Available: https://www.pagiamtzis.com/cam/camintro/. [Accessed 13 12 2017].

[32] B. Manning, "Verification Process," [Online]. Available: http://acqnotes.com/acqnote/careerfields/verification-process.

[33] Anon., "Bridges and Bridged Networks—Amendment: Per-Stream Filtering and Policing," IEEE P802.1Qci, New York, 2016.

# DEFINITIONS AND ACRONYMS

**Architecture** VHDL module body (implementation)

**ASIC** Application Specific Integrated Circuit

**ATE** Automatic-testing Equipment

**AVB** Audio and Video Bridging

**BIST** Built-In Self-Test

**CPLA** Complex Programmable Logic Array

**DDR** Double Data Rate

**DFT** Design for Testability

**DUT** Design Under Test

**e.g.** exempli gratia (In English: for example)

**entity** Specification of a VHDL module interface (ports and generics)

**etc.** et cetera

**FIFO** First In First Out

**FPGA** Field-programmable gate array

**Gbps** Giga-bit per second

**HDL** Hardware Description Language

**IP** Intellectual Property (commonly used for a complex VHDL design)

**I/O** Input/output

**IC** Integrated Circuit

**IEEE** Institute of Electrical and Electronics Engineers

**JTAG** Joint Test Action Group

**LFSR** linear Feedback Shift Register

**LSB** Least Significant Bit

**LUT** Look-up Table

**MAC** Media Access Control

**MSB** Most Significant Bit

**OSI** Open Systems Interconnection

**PHY** Physical Layer

**PLA** Programmable Logic Array

**PA** protocol aware

**PA-HWT** protocol aware hardware tester

**PRBS** Pseudo-Random Binary Sequence

**RTL** Register Transfer Logic

**signal** Basic VHDL interconnect element (used to connect ports between modules

**TSN** Time-Sensitive Networks

**USB** Universal Serial Bus and device pins as well as to define registers

**VHDL** VHSIC Hardware Description Language

**VHSIC** Very High Speed Integrated Circuit

# FIGURES

# TABLES

# APPENDIX

# A USER GUIDE

ModelSim and QUARTUS II tool were used for testing and implementation.

In order to run tester in the program ModelSim was recommended to:

- Run the Command Prompt (CMD), adjust the correct folder (e.g. *C:/Users/.../Desktop/ PA_HW_tester_for_Qbu*), then run the *setup_env*. When the *setup_env* is compiled run the *run_tp_error_state_000.bat* or *run_tp_smoke_test_000.bat*. After that the run of the ModelSim program is automatic.

For implementation of the PA-HWT was recommended to:

- Turn on the QUARTUS II program, open the project *PA_HW_tester_for_Qbu*=> *impl* => *top_quartus* => *result* => open *PA_HW_TESTER.qpf*

# B PROGRAMMING INTERFACE

Objects have input address range defined as 32 byte address space, with no defined order. Wherever host write (in this range), it will be written into first free FIFO space.

Table 3-1 shows output address map for monitor object. “fatal_errors” has been defined as one vector indicated internal status monitor, other counters defined transaction status.

Table 3-2 shows output address map for generator object. For now, there is three vectors of fatal errors.

Table 3-1: Monitor status object registers

| **Status name** | **Offset** | **Bit** | **Status registers** |
|---|---|---|---|
| fatal_errors | 0 | 0 | pattern_moni(1).one_frame_err |
| | | 1 | pattern_moni(2).one_frame_err |
| | | 2 | port_moni.cam_error |
| | | 3 | port_moni.patt_not_ready |
| | | 4 | port_moni.sfd_error |
| | | 5 | port_moni.preamble_error |
| | | 6 | port_moni.idle_error |
| | | 7 | reserved |
| monitor port counters | 1 | smd_i_err_count | |
| | 2 | smd_c_err_count | |
| | 3 | crc_packet_err_count | |
| | 4 | frag_no_err_count | |
| | 5 | preamble_length_err_count | |
| | 6 | correct_crc_pck_counter | |
| monitor pattern counters | 7 | error_frame_counter | |
| | 8 | correct_frame_counter | |

Table 3-2: Generator status object registers

| **Status name** | Offset | **Bit** | **Status registers** |
|---|---|---|---|
| fatal_err_patt_1 | 0 | 0 | lfsr_error |
| | | 1 | linear_error |
| | | 2 | crc_error |
| | | 3 | control_error |
| | | 4 | mux_error |
| | | 5 | fifo_avalon_error |
| | | 6 | Reserved |
| | | 7 | Reserved |
| fatal_err_patt_2 | 1 | 0 | lfsr_error |
| | | 1 | linear_error |
| | | 2 | crc_error |
| | | 3 | control_error |
| | | 4 | mux_error |
| | | 5 | fifo_avalon_error |
| | | 6 | Reserved |
| | | 7 | Reserved |
| fatal_err_port | 2 | 0 | bridge_error |
| | | 1 | output_error |
| | | 2-7 | Reserved |

# C COMMAND CONSTANTS

Table 3-3: Generator commands

| Command name | Command constant |
|---|---|
| INIT_PATTERN_GEN | X"FF90" |
| INIT_PORT_GEN | X"FF91" |

Table 3-4: Generator pattern commands

| Command name | Command constant |
|---|---|
| C_GEN_CMD_IDLE | X"FFA0" |
| C_GEN_CMD_START | X"FFA1" |
| C_GEN_CMD_END | X"FFA2" |
| C_GEN_CMD_FRAME_NO | X"FFA3" |
| C_GEN_CMD_FRAME_LENGTH | X"FFA4" |
| C_GEN_CMD_FRAME_SEED | X"FFA5" |
| C_GEN_CMD_SEED_INCREMENT | X"FFA6" |
| C_GEN_CMD_RANDOM_DATA | X"FFA7" |
| C_GEN_CMD_LINEAR_DATA | X"FFA8" |
| C_GEN_CMD_CRC_EN | X"FFA9" |

Table 3-5: Generator port commands

| Command name | Command constant |
|---|---|
| SET_FRAG | X"FFB1" |
| PRIORITY_INPUT | X"FFB2" |
| NO_OF_FARGMENT | X"FFB3" |
| PACKET_CRC_EN | X"FFB4" |
| FRAME_WAIT_PRIORITY | X"FFB5" |
| FRAME_WAIT_FRAG | X"FFB6" |
| PREAMBLE_LENGTH | X"FFB7" |
| SFD | X"FFB8" |
| SMD_I | X"FFB9" |
| SMD_C | X"FFBA" |
| TERMINATED | X"FFBB" |

Table 3-6: Monitor commands

| Command name | Command constant |
|---|---|
| C_MON_CMD_INIT_PATTERN_MON | X"FF90" |
| C_MON_CMD_INIT_PORT_MON | X"FF91" |

Table 3-7: Monitor pattern commands

| Command name | Command constant |
|---|---|
| C_MON_CMD_IDLE | X"FFA0" |
| C_MON_CMD_START | X"FFA1" |
| C_MON_CMD_END | X"FFA2" |
| C_MON_CMD_FRAME_NO | X"FFA3" |
| C_MON_CMD_FRAME_LENGTH | X"FFA4" |
| C_MON_CMD_FRAME_SEED | X"FFA5" |
| C_MON_CMD_SEED_INCREMENT | X"FFA6" |
| C_MON_CMD_RANDOM_DATA | X"FFA7" |
| C_MON_CMD_LINEAR_DATA | X"FFA8" |
| C_MON_CMD_CRC_EN | X"FFA9" |
| C_MON_CMD_SCOREBOARD_EN | X"FFAA" |

Table 3-8: Monitor port commands

| Command name | Command constant |
|---|---|
| C_MON_CMD_PRIORITY_OUTPUT | X"FFB2" |
| C_MON_CMD_SET_FRAG | X"FFB1" |
| C_MON_CMD_NO_OF_FARGMENT | X"FFB3" |
| C_MON_CMD_PACKET_CRC_EN | X"FFB4" |
| C_MON_CMD_PREAMBLE_LENGTH | X"FFB7" |
| C_MON_CMD_SFD | X"FFB8" |
| C_MON_CMD_SMD_I | X"FFB9" |
| C_MON_CMD_SMD_C | X"FFBA" |
| C_MON_CMD_TERMINATED | X"FFBB" |