Real-time flexible and virtualized transponders for optical telecommunications

Alexandre Gouin

DOCTORAT BRETAGNE LOIRE / MATHSTIC

INSA RENNES

THÈSE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Télécommunications*

Par

Alexandre GOUIN

Transpondeurs temps-réel flexibles pour les télécommunications optiques et virtualisés

Real-time flexible and virtualized transponders for optical communications

Thèse présentée et soutenue à Rennes, le 31/05/2022
Unité de recherche : IETR
ISAR 12 / D22 - 12

Rapporteurs avant soutenance :

Olivier ROMAIN	PR, Université de Cergy Pontoise
Cédric WARE	PR, Télécom Paris

Composition du Jury :

Présidente :	Christelle AUPETIT-BERTHELEMOT	PR, XLIM
Examinateurs :	Cédric WARE	PR, Télécom Paris
	Olivier ROMAIN	PR, ETIS
	Matthieu GAUTIER	MCF-HDR, IRISA
	Angélique RISSONS	PR, ISAE - SUP AERO
	Jean-Christophe PREVOTET	PR, IETR
Dir. de thèse :	Fabienne NOUVEL	MCF-HDR, IETR
Co-dir. de thèse :	Patricia LAYEC	Ingénieure de recherche, Nokia Bell Labs

Invité :

Arnaud DUPAS	Ingénieur de Recherche, Nokia Bell Labs

Contents

Abstract

Optical transport networks, the high capacity network where optical links span from hundreds to thousand kilometers, have undergone major changes in recent years to accompany the ever growing demands in bitrate, service reliability and reduction in latency. The Elastic Optical Network paradigm is under a lot of development, taking advantage of the higher reconfigurability of optical hardware and increases of the control plane automation capabilities to make the networks more reliable and flexible.

In this manuscript, we focus on the optical transponder, the optical equipment that bridges the access networks where client signals come from and the optical transport network that routes them to their destination. This equipment has followed the trend of flexibility and is more reconfigurable than ever. From the modulation format, the optical launch power, the central frequency of the signal, the error correction code, etc... One can tune an optical transponder parameter to the current needs of the network. Also, the transponder is following the adoption of the virtualization techniques inside the optical network, which allows more advanced handling of its resources by the control plane and increases the automation capabilities of the optical network.

However fully flexible optical networks are not yet a reality, as some challenges have yet to be resolved. Firstly, the hardware is reconfigurable but has its limitations, especially the necessity to be put offline for the reconfiguration to be applied. As a change in the emitter can require adaptations in the receiver, efficient and synchronous changes of parameters are still a challenge. Also, the over centralization of monitoring and decision making in the control plane creates performance bottlenecks, which leads to higher time to detect a failure in the network and propose a change of configuration to resolve it. This makes reconfigurations rare and operational margins high for optical transponders, which will not run at their maximum potential.

We believe that to alleviate the aforementioned challenges for fully flexible optical networks, optical transponders should be given more decision power. By bringing more intelligence to the device, allowing it to process its own monitoring data, make decisions on the ideal set of parameters to resolve a degradation in quality of service and triggering itself a synchronous change of parameters with another transponder of the network will

allow for more active and frequent reconfigurations in the optical networks, increasing the networks overall flexibility and reliability.

Firstly, we provide in this thesis work an auto-negotiation protocol for optical transponders that allows for fast synchronous change of parameters. Integrated with a commercial transponder and an embedded monitoring and decision-making solution we have validated our solution against a centralized control plane in a network testbed in terms of fault detection and resolution times. We also validate our auto-negotiation solution with a real-time baud-rate variable setup and showcase a synchronized change of baudrate from 14GBaud to 7GBaud with the help of the protocol.

Secondly, we provide a neural network solution for embedded monitoring in the optical data plane. By allowing for a reconfiguration of the neurons weights of the network we believe that this solution can both provide advanced monitoring techniques for the optical equipment and allow smarter fault detection and reconfigurations in the data plane and provide computational resources for the optical control plane for remote monitoring which would accelerate decision making for the whole network.

List of publications

[i] **A. Gouin**, A. Dupas, and P. Layec, « In-Line Transmission Parameters Synchronization Protocol for Hitless Optical Coherent Communication », in 2020 International Conference on Optical Network Design and Modeling (ONDM), may 2020, p. 1-3. doi: 10.23919/ONDM48393.2020.9132995.

[ii] **A. Gouin**, A. Dupas, L. G. Renom, A. Benabdallah, F. Boitier, and P. Layec, « Dynamic auto-negotiation with real-time transponders in software defined optical networks », in 2021 Optical Fiber Communications Conference and Exhibition (OFC), june 2021, p. 1-3., doi: 10.1364/OFC.2021.W6A.6

[iii] **A. Gouin**, A. Dupas, L. Gifre Renom, A. Benabdallah, F. Boitier, and P. Layec, « Real-time optical transponder prototype with autonegotiation protocol for software defined networks », Journal of Optical Communications and Networking (JOCN), vol. 13, n. 9, p. 224, sept. 2021, doi: 10.1364/JOCN.427938.

[iv] **A. Gouin**, A. Dupas, , F. Boitier, F. Nouvel, J.C. Prévotet and P. Layec, "Real-time optical transponder prototype with auto-negotiation protocol for software defined networks", GDR SOC2, Juin 2021, p. 2.

[v] E. Dutisseuil, A. Dupas, **A. Gouin**, F. Boitier, and P. Layec, « Hitless Transmission Baud Rate Switching in a Real-Time Transponder Assisted by an Auto-Negotiation Protocol » (Top Scored), in 2022 Optical Fiber Communications Conference and Exhibition (OFC), march 2022, p. 4.

List of Figures

List of Tables

Acronyms

ADC Analog-to-Digital Converter. xxviii, 21, 22, 25, 27, 82, 84, 87

ANN Artificial Neural Network. 4, 20, 92–96, 98–101, 107, 113, 117, 118, 120–125, 127

BER Bit Error Rate. xxxiii, 19, 37, 50, 64, 65, 67, 72, 74

BVT Bandwidth Variable Transponder. 4, 42–44, 46

CMA Constant Modulus Algorithm. 26, 82, 84, 85

DAC Digital-to-Analog Converter. xxix, 21, 22, 24, 26, 102

DDR Double-Data-Rate. xxxv, 35, 57, 122

DSP Digital Signal Processing. xiii, xxix, xliii, 2, 11, 21, 22, 24–27, 29, 30, 35, 41, 44, 45, 47, 50, 53, 82, 85, 87, 117, 118

EDFA Erbium Doped Fiber Amplifier. 15, 102

FEC Forward Error Correction. xxix, 7, 23, 38, 45, 48, 64–67, 72, 74, 85, 88

FIFO First In First Out. li, 35, 121, 122

FPGA Field Programmable Gate Array. xxiv, xxxiv–xxxvi, xl, xliii, xlv–xlviii, li, lii, 4, 31–36, 40, 44, 45, 47, 53, 57–61, 63, 64, 70, 72, 76, 79, 80, 82, 85, 87, 88, 90, 91, 93, 101, 107, 109, 113–115, 117, 120, 122, 123, 125, 127

GMPLS Generalized Multi-Protocol Label Switching. 18, 37

LUT Look Up Table. 33, 34, 36, 44, 113, 117, 118

MAC Medium Access Controller. xxxiv, xxxv, 35, 57

MSE Mean Square Error. 95, 104, 114–116

OADM Optical Add-Drop Multiplexer. xxvii, 3, 10–13, 15, 41, 47

ODU Optical Data Unit. 7, 55, 56

OTN Optical Transport Network. xxiv, xxv, xxxiii, 3, 6, 7, 45

PRBS Pseudo Random Binary Sequence. 79, 85

PSD Power Spectral Density. 98, 99, 102

QPSK Quadrature Phase Shift Keying. xxix, 23, 26, 44, 49, 50, 52, 80, 84, 102

RAM Random Access Memory. xxxv, 35, 36, 76, 113, 117–119, 122

ROADM Reconfigurable Optical Add-Drop Multiplexer. xl, 13, 14, 17, 38, 41, 45, 71

ROM Read-Only Memory. 79, 107, 110, 111, 114–117, 119, 123

S-BVT Sliceable Bandwidth Variable Transponder. xxx, 4, 31, 42, 46, 47, 49

SDN Software Defined Network. xxvii, xxviii, xxxii, xl, 3, 18, 19, 37, 39, 41, 45, 50, 71, 73, 74, 76, 91, 92, 126

SNR Signal-to-Noise Ratio. xlv, 97–99, 103, 113, 125, 127

SSH Secure Shell. xxxvii, 64

tanh Hyperbolic Tangent. 103, 107, 110

UART Universal Asynchrounous Receiver-Transmitter. xxxvii, li, 64, 72, 87, 122, 123

VHDL Very High Speed Integrated Circuit Hardware Description Language. 36, 57

WDM Wave Division Multiplexing. xxiv, xxv, xxvii, xxviii, 3, 6, 7, 10, 12, 13, 16, 44

YANG Yet Another Next Generation. 38, 46

Résumé étendu en français

Les réseaux optiques sont devenus le moyen privilégié de faire transiter l'information à très grande vitesse et sur des grandes distances. Ils interconnectent les usagers, les villes, pays et régions et continents entre eux et leur utilisation ne cesse de croître, tandis que les besoins en débit et en réduction de latence ne cessent d'augmenter, notamment dans le contexte d'adoption de la 5G, des services dans le cloud et de l'internet des objets.

Pour subvenir à l'augmentation incessante des besoins, de nouvelles fonctionnalités et de nouvelles façons de concevoir les réseaux optiques ont émergées. Tout d'abord, l'adoption des communications cohérentes a permis d'augmenter sensiblement les débits possibles pour les communications optiques, en utilisant l'amplitude la phase et la polarisation du signal optique pour transmettre l'information. Aussi, les développements modernes des équipements ont permis de pouvoir reconfigurer le matériel, afin de pouvoir répondre à une baisse de qualité de service efficacement, en changeant un paramètre de transmission ou en reroutant le trafic dans le réseau. Aussi, l'adoption du plan de contrôle centralisé a permis le développement des capacités d'automatisation et de virtualisation des ressources dans les réseaux. Toutes ces évolutions sont à la base du développement des réseaux optiques flexibles d'aujourd'hui.

Si les réseaux ont gagné en flexibilité, il reste néanmoins des axes d'améliorations avant l'arrivée des réseaux réellement flexibles et automatisés. Tout d'abord, reconfigurer le matériel conduit encore souvent à des interruptions de service pendant que le matériel change ses paramètres, ce qui fait que les adaptations dans le réseau sont encore rares. De plus, la centralisation du plan de contrôle permet certes de gagner en efficacité et en automation, mais l'hyper concentration des tâches de surveillance et de prise de décision crée des délais et des pertes de performance pour la détection et la résolution des dégradations dans le réseau.

Dans cette thèse et au vu de ce contexte, nous nous sommes concentrés sur le transpondeur optique, équipement faisant le lien entre les réseaux d'accès d'où proviennent les données client et les réseaux de transport optique. Cet équipement est au centre des récents développements de flexibilité et de virtualisation.

Dans une première partie nous allons développer le contexte des réseaux de transport

optique, et des concepts de flexibilité et de virtualisation. Nous allons aussi présenter plus en détail le transpondeur optique et de ses évolutions modernes. Nous allons ensuite présenter une solution d'auto-négociation et de monitoring embarqué permettant le déclenchement de reconfigurations rapides et synchronisées pour compenser une dégradation en qualité de service. Nous allons mesurer ses performances et les comparer à un plan de contrôle centralisé pour résoudre une dégradation au sein d'un banc de test réseau. Nous allons aussi présenter une solution de monitoring avec réseau de neurones embarquée sur Field Programmable Gate Array (FPGA). Cette solution possède une fonctionnalité de reconfiguration des poids des neurones afin de changer la fonction effectuée par le réseau, permettant au plan de contrôle centralisé de réaliser du monitoring déporté, notamment sur des métriques sous utilisées et demandant un grand flux de données. Nous avons validé cette solution en termes de performances et de précision en réalisant un monitoring des contributions nonlinéaires à partir des spectres optiques.

Réseaux de transport optique

Couche physique

Les réseaux de transport optiques recouvrent les réseaux métropolitains (distance entre nœuds optiques de quelques dizaines à quelques centaines de kilomètres environ) et coeurs de réseaux (milliers de kilomètres entre nœuds), et sont au cœur des transformations récentes dans les réseaux optiques et des besoins croissants en débit et en fiabilité [2]. Ils sont basés autour de deux technologies majeures, le Multiplexage en Longueur d'Onde ou Wave Division Multiplexing (WDM), et le standard OTN (Optical Transport Network). Le WDM permet à une fibre optique de porter plusieurs longueurs d'onde multiplexées sur un signal. Cela permet d'augmenter significativement la capacité des réseaux optiques. Les signaux des clients du réseau optique peuvent être ajoutés ou retirés du signal WDM selon leur point d'arrivée ou de destination dans le réseau. Le WDM est représenté dans la Figure 1.

Le standard OTN est un groupe d'outils permettant de gérer les signaux clients et leurs longueurs d'onde du réseau de transport optique, et est basé sur un format de trame avec un grand nombre de champs permettant de monitorer, router et concaténer les signaux. Plusieurs générations de trames OTN existent et sont adaptées au débit maximum supporté (par exemple l'OTU4 supporte le transport de signaux client jusqu'à

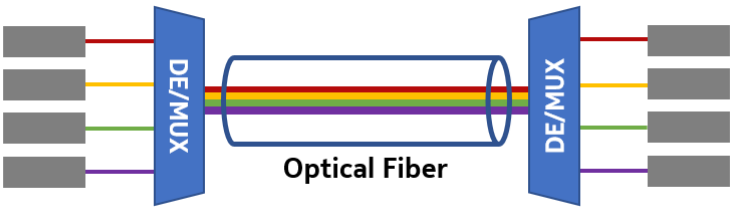


Figure 1 – Représentation de la technologie de Multiplexage en Longueur d'Onde (WDM)

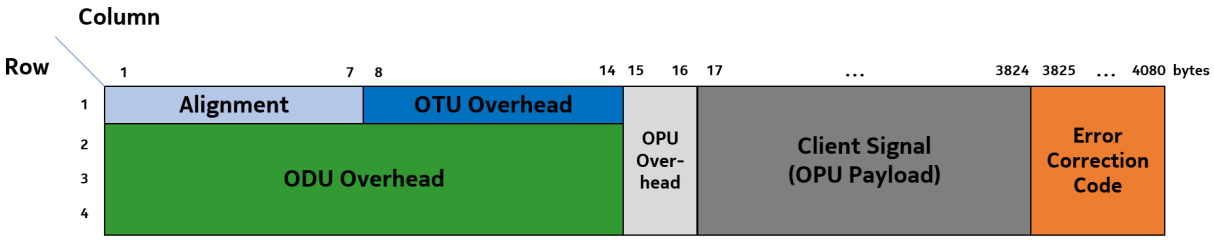


Figure 2 – Structure d'une trame OTN (Optical Transport Network). Plusieurs sous structures cohabitent dans la trame : OTU (Optical Transport Unit), ODU (Optical Data Unit) et OPU (Optical Payload Unit), chacun comprenant leurs propres champs (permettant le monitoring, le concaténation, etc... des signaux clients).

100Gbps), avec la possibilité de multiplexer plusieurs trames de générations inférieures, pour transformer plusieurs signaux à faible débit en un signal à haut débit. La structure d'une trame OTN est représentée dans la Figure 2.

Aujourd'hui, le paradigme principal dans les réseaux de transport optique est le Réseau Optique Elastique (Elastic Optical Network, EON), qui a été rendu possible grâce à l'introduction de plusieurs principes. Le premier changement ayant permis l'arrivée des réseaux élastiques est l'introduction des équipements reconfigurables au sein du réseau de transport optique, permettant d'adapter la configuration de chaque équipement aux besoins du réseau en débit ou en robustesse par exemple. L'autre changement a été l'utilisation de la grille flexible (flexgrid) de fréquence dans les réseaux WDM. Contrairement à la grille dite fixe traditionnellement utilisée, qui répartissait les signaux optiques dans un emplacement de 50GHz de large dans la bande de fréquence, la flexgrid permet une plus grande granularité en utilisant des slots plus fins (par exemple 12.5GHz de largeur de bande), qui peuvent être remplis de façon plus intelligente et plus adaptées aux besoins de chaque signal [3], comme montré dans la Figure 3. Cela permet aussi d'économiser de la bande spectrale et d'utiliser des signaux à bande passante plus large que 50GHz sans gâcher inutilement de l'espace.

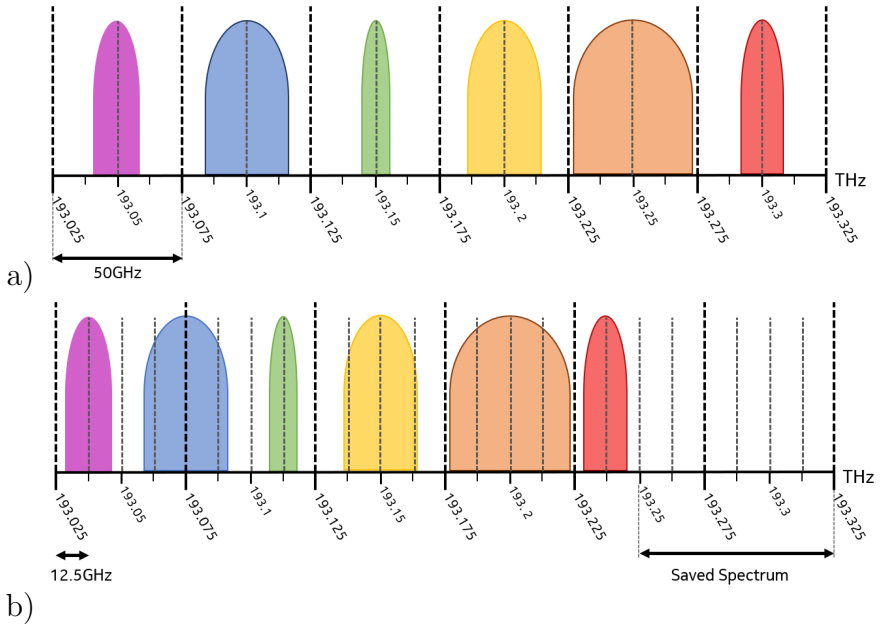


Figure 3 – Représentations de (a) la grille de fréquence fixe traditionnellemnt utilisée dans les réseaux de transport optique, et (b) la grille flexible utilisée dans les réseaux modernes élastiques.

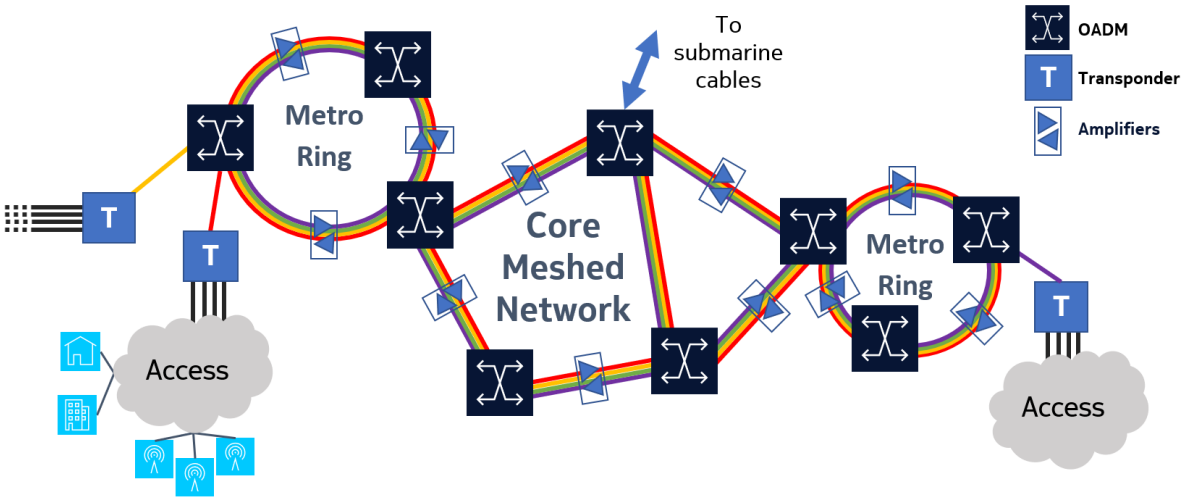


Figure 4 – Réprésentation de la couche physique d'un réseau de trasnport optique.

La Figure 4 montre une représentation de réseau de transport optique, avec ses différents équipements. Tout d'abord les transpondeurs optiques, qui font la liaison entre les réseaux d'accès, d'où proviennent les signaux des clients, et le réseau de transport optique. Ils multiplexent les signaux client sur un signal compatible WDM et envoient ce dernier à un Optical Add-Drop Multiplexer (OADM) qui routera ce signal vers sa destination dans le réseau optique, où le signal sera démultiplexé par le transpondeur en sortie. Dans les chemins optiques se trouvent aussi des amplificateurs qui permettent d'augmenter la distance maximale de transmission du signal optique, mais ajoutant des effets indésirables dans la transmission en contrepartie. Ces différents éléments ont pu évoluer au gré des avancées techniques, proposant différentes possibilités de configuration, et pouvant être reconfigurés à distance par un opérateur ou par un élément du plan de contrôle.

Plan de contrôle

Pour orchestrer efficacement et automatiquement les différents éléments du réseau, l'approche la plus commune aujourd'hui est de mettre en place un plan de contrôle optique centralisé. Ce plan de contrôle possède des fonctions de monitoring qui surveillent l'état des différents éléments de la couche physique afin de proposer des configurations appropriées pour les équipements optiques. Le paradigme actuel d'organisation du plan de contrôle est le Software Defined Network (SDN) [4]. Il se base sur une grande centralisation des fonctions de contrôle au sein d'un groupe d'entités séparé de la couche physique. La Figure 5 montre une représentation simplifiée d'un plan de contrôle SDN. Pour connaître précisément l'état du réseau, le plan de contrôle récolte régulièrement les données de monitoring des équipements optiques, comme par exemple la puissance de réception, le taux d'erreur (ou Bit Error Rate) avant et après code de correction d'erreur (Forward Error Correction, FEC). Plusieurs techniques de récupération des données de monitoring existent, les plus récentes se basant sur des flux des données, permettant une récolte rapide des données de monitoring et permettant au plan de contrôle de pouvoir s'abonner ou de se désabonner à ces flux et d'adapter ses stratégies de monitoring selon les besoins [5].

Influencés par l'informatique dans le *cloud*, les réseaux optiques ont commencé à intégrer de la virtualisation dans la gestion du réseau et des équipements. Le principe de la virtualisation est de découpler les fonctions logicielles du matériel, ce qui permet par exemple au plan de contrôle de créer des sous-réseaux avec des besoins particuliers (en latence, en sécurité, en débit, etc...). Les équipements peuvent être partitionnés en plusieurs équipement virtuels ou agrégés en un seul super équipement, des fonctions spécifiques

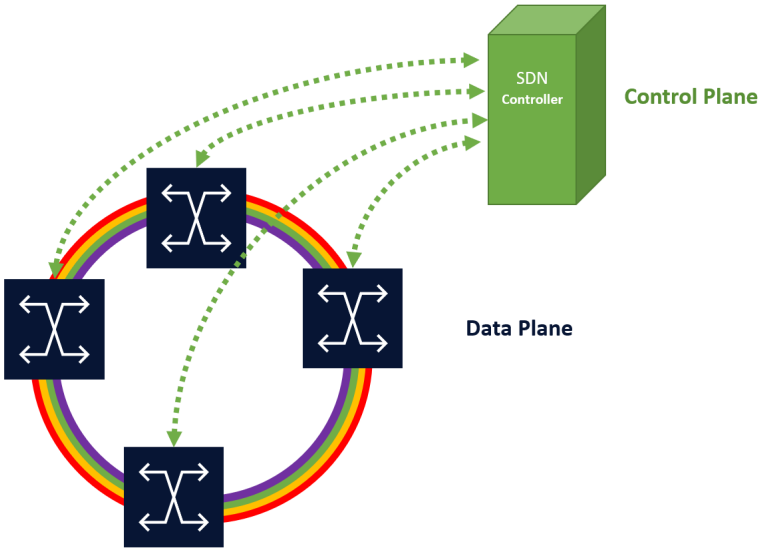


Figure 5 – Représentation simplifée d'un plan de contrôle centralisé SDN (Software Defined Network)

peuvent être déployées à distance (VNF, Virtual Network Functions), et même une instance de contrôleur réseau SDN peut être déployé pour répondre aux besoins spécifiques de ce sous-réseau [6], voir la Figure 6.

Dans ce contexte où la flexibilité des équipements optiques et du réseau tout entier a augmenté considérablement, et avec les besoins en débit croissants, les transpondeurs ont subi plusieurs changements pour s'intégrer aux nouvelles générations de réseaux optiques.

Généralités sur le transpondeur optique et le transpondeur optique reconfigurable

Le transpondeur est un équipement servant de point d'entrée et de sortie pour les signaux client dans les réseaux de transport optique. Dans les réseaux WDM, les signaux clients en entrée sont convertis en signaux électriques, traités, agrégés et transformés en signal optique dont la longueur d'onde est compatible avec la technologie WDM. Ce signal est envoyé sur le port ligne du transpondeur. A l'inverse, le signal reçu sur le port ligne est de son côté déconcaténé en plusieurs signaux client après traitement.

La Figure 7 représente l'architecture de base d'un transpondeur optique. On y trouve les convertisseurs analogiques-numériques (Analog-to-Digital Converters, ADCs) et nu-

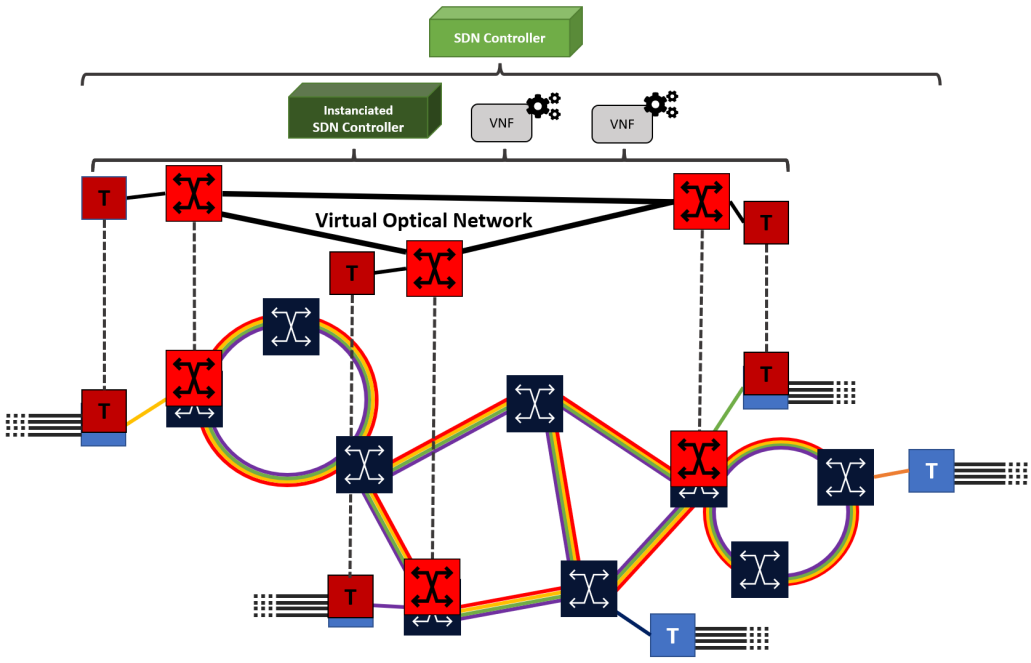


Figure 6 – Exemple de sous-réseau optique virtuel avec un contrôleur SDN instancié spécifiquement et des fonctions virtuelles VNF.

mériques-analogiques (Digital-to-Analog Converters, DACs), les éléments de traitement du signal (Digital Signal Processing, DSP), avec le code de correction d'erreur (Forward Error Correction, FEC), de pré-compensation de distorsion (Digital Pre-Compensation, DPC et Digital Pre-Emphasis, DPE), d'égalisation (EQ), de synchronisation, et la gestion de la modulation. On y trouve aussi les émetteurs et récepteurs optiques. Plusieurs types de modulations existent pour les transpondeurs optiques, qui se sont complexifiées avec les années. Les modulations de phase et d'amplitude sont très communes, comme la modulation à quadrature de phase QPSK ou la modulation d'amplitude en quadrature 16 états 16-QAM, les systèmes optiques profitent aussi des propriétés du signal lumineux avec notamment les modulations multiplexées sur la polarisation de la lumière, comme la PM-QPSK (voir Figure 8).

Si les transpondeurs étaient des éléments statiques du réseau optique à leurs débuts, ne supportant qu'une configuration pendant leur durée de vie, ils ont gagné de nombreuses possibilités de configuration. La longueur d'onde d'émission et de réception est devenue changeable, et le choix de fréquence a gagné en granularité pour être plus adapté à l'utilisation de la grille flexible dans les réseaux optiques. Les formats de modulation sont aussi un élément pouvant être reconfigurés pour s'adapter aux conditions changeantes du

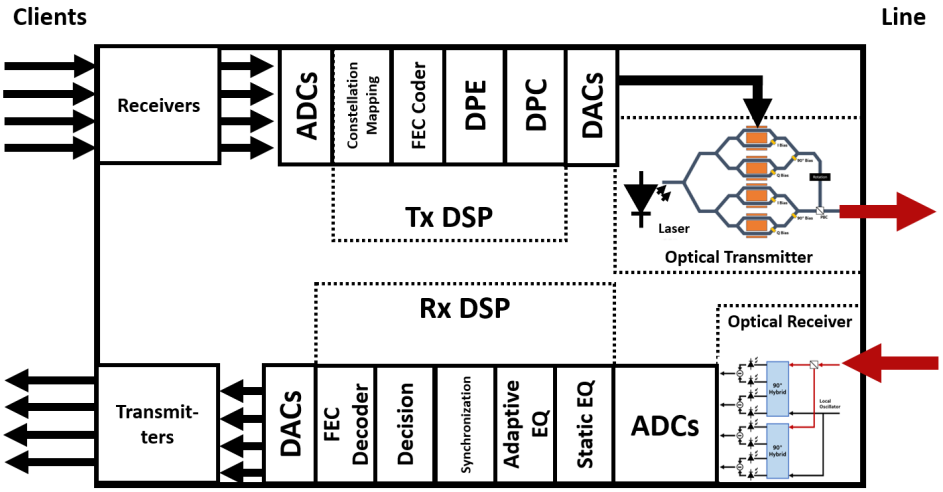


Figure 7 – Représentation graphique d'un transpondeur optique.

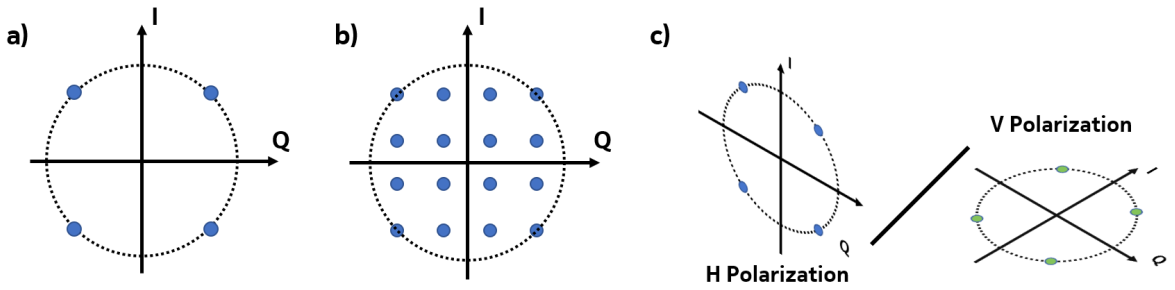


Figure 8 – Représentation des constellations pour les modulations (a) QPSK, (b) 16-QAM et (c) PM-QPSK.

réseau optique. Passer d'une transmission en 16QAM à une transmission en QPSK permet d'augmenter la résistance de la communication aux perturbations dans la fibre optique, au prix d'une baisse en débit. Ces changements de format de transmission doivent être répercutés sur les algorithmes de traitement de signal lors de la réception, pour éviter les incompatibilités.

L'intégration des concepts de virtualisation a aussi eu un impact majeur sur la conception des transpondeurs optiques. L'adoption des transpondeurs multi-flow, représenté en Figure 9, où le trafic client peut être librement distribué sur plusieurs ports lignes [7] selon la destination du signal, les besoins en distance de propagation ou en débit. Cela permet par ailleurs la création de transpondeurs virtuels dans le réseau pour s'adapter aux besoins de façon très flexible, voir Figure 10. Ces transpondeurs multi-flow reconfigurables et virtualisables sont appelés les Sliceable Bandwidth Variable Transponder (S-BVT).

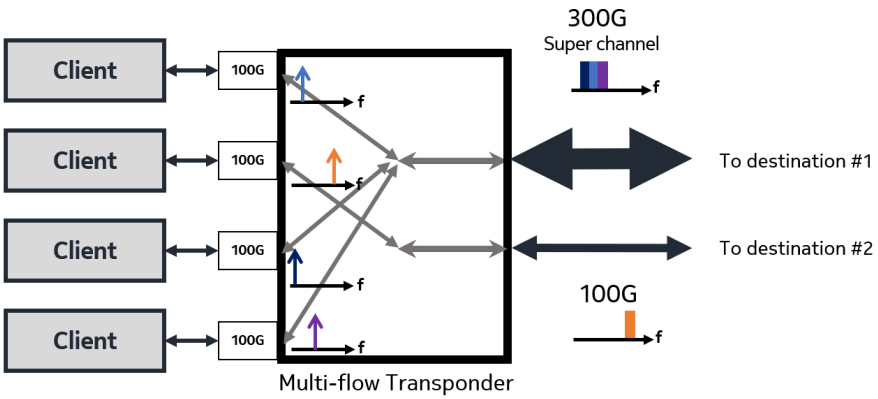


Figure 9 – Représentation d'un transpondeur multi-flow, où le trafic client provenant de quatre ports à 100Gb est reparti sur deux ports ligne.

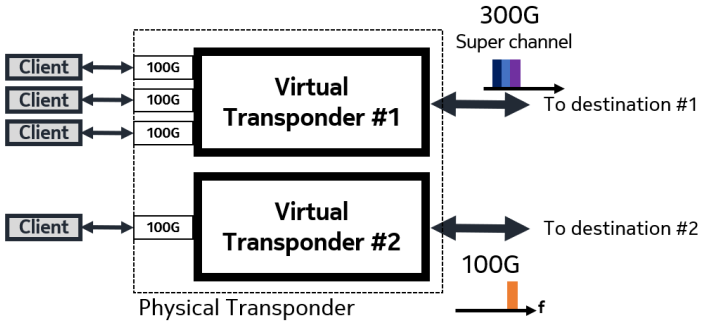


Figure 10 – Représentation de la virtualisation du transpondeur multi-flow de la Figure 9 en deux transpondeurs virtuels.

Un autre sujet de recherche actif autour des transpondeurs optiques est la création de boîtes blanches ("white boxes"), où des transpondeurs de un ou plusieurs vendeurs sont intégrés et complètement virtualisés. Le plan de contrôle ne s'adresse qu'à des agents spécifiques qui s'occuperont de transmettre les instructions aux différents transpondeurs. C'est un changement assez important dans la conception de ce qu'est un transpondeur et dans la manière dont ils sont opérés, réduisant possiblement les coûts mais répercutant certaines phases de développement aux opérateurs et non plus aux constructeurs [8]. Une représentation simplifiée d'une white box est présente en Figure 11.

La prochaine étape pour les transpondeurs optiques est l'adoption de techniques permettant de faire des changements de paramètres plus rapides, idéalement instantanés, ainsi que de pouvoir synchroniser ces changements dans le réseau. En effet, comme énoncé plus haut, changer un paramètre de transmission peut nécessiter des adaptions à la réception. Néanmoins aujourd'hui, avec la centralisation de la prise de décision au niveau

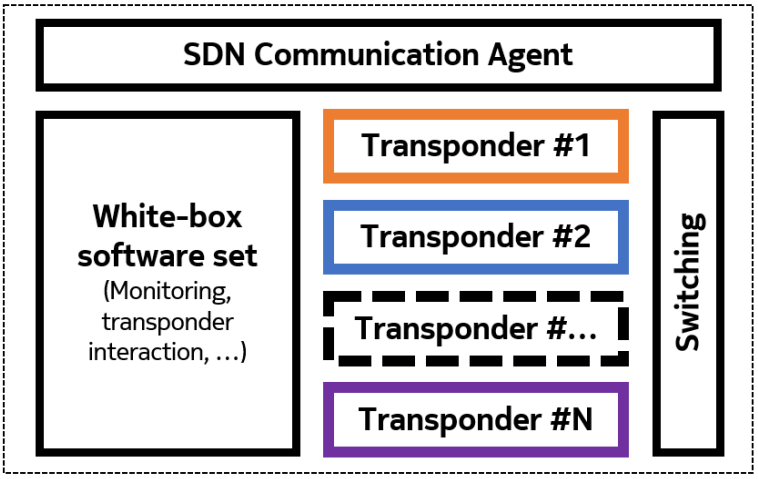


Figure 11 – Exemple d'implémentation de white box avec des transpondeurs.

du plan de contrôle qui doit superviser l'entièreté du réseau à chaque instant, respecter des délais raisonnables et synchroniser des opérations à travers le réseau s'avère être une tâche particulièrement ardue. Ces deux facteurs font que les reconfigurations du réseau se font majoritairement dans des périodes de maintenances ou lors de la rupture d'un lien optique à la suite d'une dégradation. Permettre des changement rapides et synchronisés est une piste prometteuse pour rendre le réseau optique plus flexible et plus robuste. On pourrait aussi envisager des reconfigurations pour optimiser l'utilisation des appareils optiques, et ainsi réduire leurs marges d'opérations. Cela serait possible en augmentant l'autonomie des transpondeurs optiques, afin de leur permettre de réagir rapidement et efficacement en cas de dégradation de qualité de service. Ce genre d'architectures partiellement décentralisée a été étudiée dans [9, 10], et ne remettent pas en cause le paradigme SDN.

Prototypage d'un transpondeur optique flexible pour réseaux SDN

Dans cette section nous présentons un prototype de transpondeur flexible se basant sur un protocole d'auto-négociation que nous avons développé, déclenché lors de la détection d'une dégradation de la qualité de service. La base du prototype est représentée en Figure 12. Elle est composée d'un transpondeur optique, dont les données de mon-

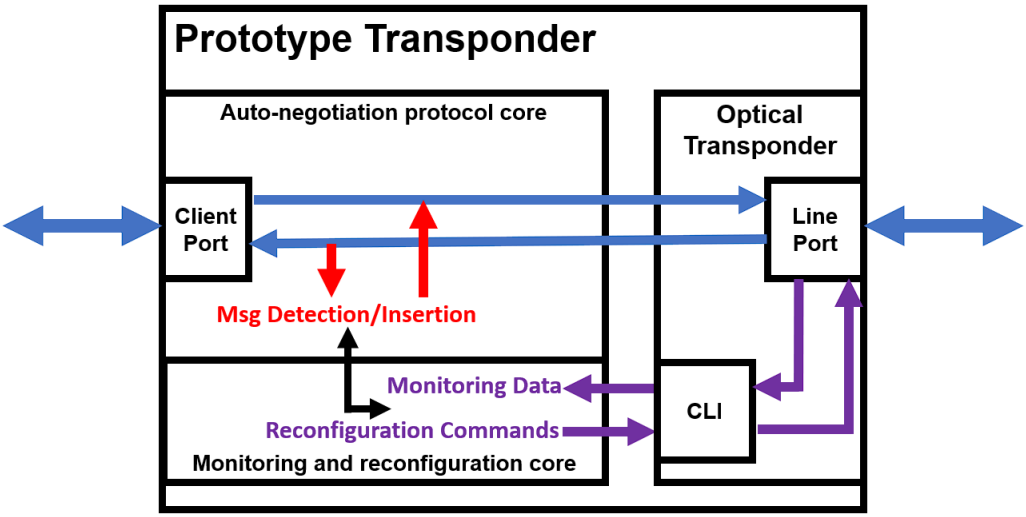


Figure 12 – Représentation de notre prototype de transpondeur avec monitoring et auto-négociation.

itoring (par exemple la puissance de réception et le taux d'erreur binaire ou BER, Bit Error Rate) sont récupérées et traitées par un bloc fonctionnel de monitoring, qui envoie aussi des commandes de reconfiguration au transpondeur et qui permet de déclencher un protocole d'auto-négociation. Le protocole d'auto-négociation sert à synchroniser rapidement un changement de paramètres entre deux transpondeurs communicants de façon bidirectionnelle. Nous allons présenter les différentes sections de ce prototype.

Protocole d'auto-négociation pour transpondeur optique

Dans la Figure 13 nous présentons les différents messages du protocole d'auto-négociation. Les messages du protocole sont échangés entre deux transpondeurs qui communiquent de manière bi-directionnelle. Les messages sont insérés dans le trafic Ethernet ou OTN, au début de la payload et dans des champs spécifiques respectivement. Le premier message est lorsque le transpondeur en réception détecte une dégradation de la qualité de service. Il envoie au transpondeur émetteur un message ALERT, lui indiquant qu'il doit changer un de ses paramètres pour rétablir la qualité de réception. Une fois que le transpondeur émetteur a décidé d'un nouveau paramètre à mettre en place, il envoie un message de requête RQST, indiquant quel paramètre va changer et à quelle valeur. Cette requête peut être acceptée au non par le récepteur, selon s'il est capable de s'adapter à ce changement ou non. La réponse est envoyée dans un message d'acknowledgement ACK, et

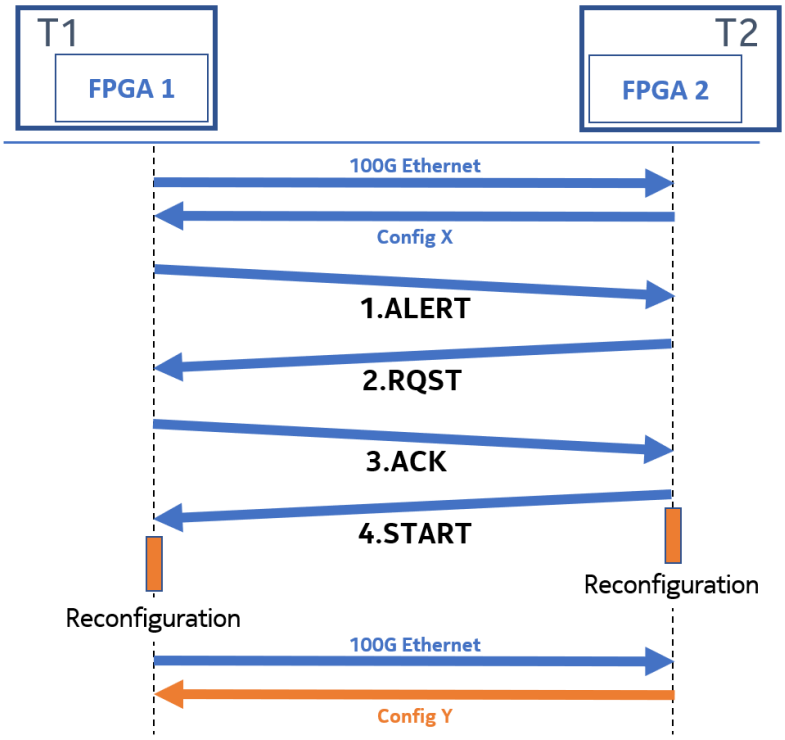


Figure 13 – Séquence des opération de notre protocole d'auto-négociation.

si le changement est accepté, le transpondeur émetteur envoie un message de déclenchement START juste avant de se reconfigurer, et le transpondeur émetteur se reconfigure une fois ce message reçu. Pendant la reconfiguration, le transmetteur peut être capable de stocker le trafic, qu'il redistribuera une fois la reconfiguration terminée. Si jamais le récepteur prend plus de temps à se reconfigurer, un délai pour la reprise du traffic peut être négocié entre les deux transpondeurs. Le but de ce protocole est de synchroniser les changements de paramètres et les différents messages nous permettent de mettre en place des mécanismes pour empêcher les incompatibilités entre matériels, tout en permettant de transformer des changements rapides de paramètres en changements sans perte de trafic, ce qui serait idéal pour améliorer la flexibilité des réseaux optiques.

Nous avons développé une plateforme de test pour valider notre solution d'auto-négociation sur Field Programmable Gate Array (FPGA), représentée en Figure 14. Un processeur logique Microblaze [11] sur le système d'exploitation temps réel FreeRTOS [12] est implémenté. Le trafic 100G est réceptionné dans et envoyé par le FPGA via des modules QSFP28 (Quad Small Form Factor Pluggable) et des cœurs logiques Medium Access Con-

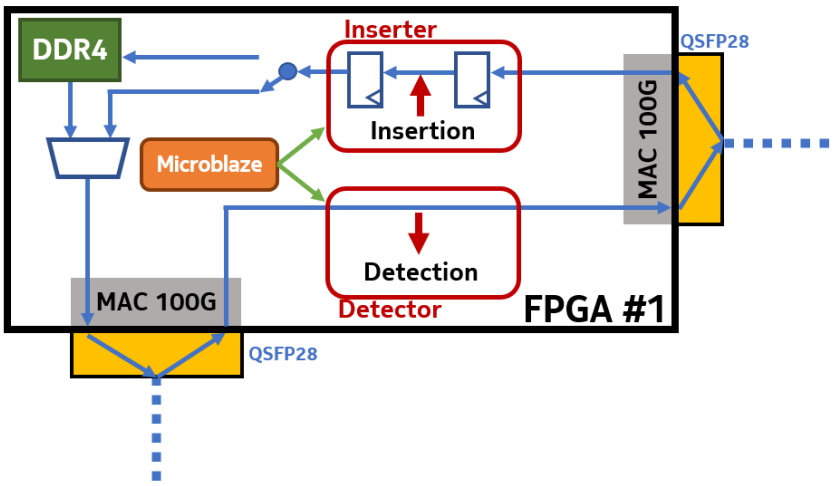


Figure 14 – Représentation schématique de l'implémentation temps-réel de notre protocole d'auto-négociation.

troller (MAC). De la mémoire Random Access Memory (RAM) DDR4 est utilisée pour stocker le trafic pendant les reconfigurations. Deux modules logiques d'Insertion et de Détection des messages du protocole ont été développés. Dans cette implémentation, les messages du protocole sont de 32 bits, et sont insérés et extraits du trafic 100G Ethernet. L'insertion des messages se fait entre deux registres clockés à 322MHz et sont détectés directement. Cela doit donc ajouter un total de 12ns d'ajout de latence dans le temps d'aller-retour si on utilise deux FPGA pour la communication bidirectionnelle :

$$2 * (2 * \frac{1}{322MHz}) = 12.4ns \tag{1}$$

Nous avons validé notre implémentation du protocole d'auto-négociation en mettant deux FPGAs avec notre design en communication bi-directionnelle, séparés par 2m ou 10km de fibre simple mode (Single Mode Fibre, SMF). Nous avons aussi un analyseur de réseau Spirent qui va générer et analyser le trafic Ethernet à 100G. La taille des trames Ethernet générées est variable, entre 128 et 1518 octets. Ce setup est représenté dans la Figure 15.

Pour la validation, le Spirent va générer pendant 10s du trafic Ethernet, à différentes charges de trafic (de 10 à 100%). Dans le même temps notre système insère 100 messages de protocole par secondes pour vérifier son fonctionnement. Cela est fait pour 2m ou 10km de fibre entre les FPGA. On analyse ensuite le trafic reçu par le Spirent en latence et en pertes de trames. Les résultats sont regroupés dans les Figures 16 (a) et (b) et dans les

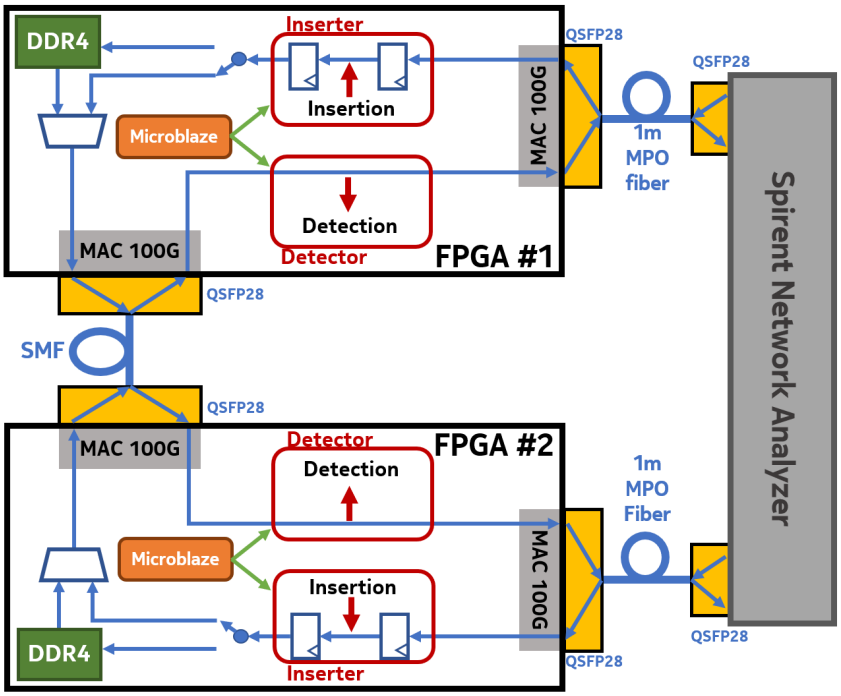


Figure 15 – Représentation de notre palteforme de test pour la validation de notre protocole d'auto-négociation, avec deux FPGAs pour l'insertion et la détection des messages d'auto-négociation, séparés par 2m ou 10km de fibre simple mode, et un Spirent pour générer et analyser du traffic Ethernet à 100Gbps.

Tableaux 1 (a) et (b) pour 2m et 10km de séparation respectivement. On va comparer ces résultats à un setup où la partie de gestion du protocole d'autonégociation est absente (les parties en rouge dans la Figure 15).

Comme attendu, la latence entre le design avec et le design sans la gestion du protocole d'auto-négociation est de 12ns en moyenne, quelles que soient la charge de trafic, la taille des trames ou la longueur de fibre entre les cartes FPGA. On remarque aussi une augmentation de la latence et du nombre de trames perdues lorsque le charge de trafic est de 100%, ce qui est plus dû aux limitation de notre FPGA que du design de notre protocole et de son implémentation.

Monitoring embarqué

Pour déclencher le protocole d'auto-négociation de manière pertinente, il faut pouvoir détecter quand la qualité de service se dégrade. Nous avons donc réalisé un prototype de monitoring embarqué pour un transpondeur commercial Nokia PSI-2T (Photonic Service

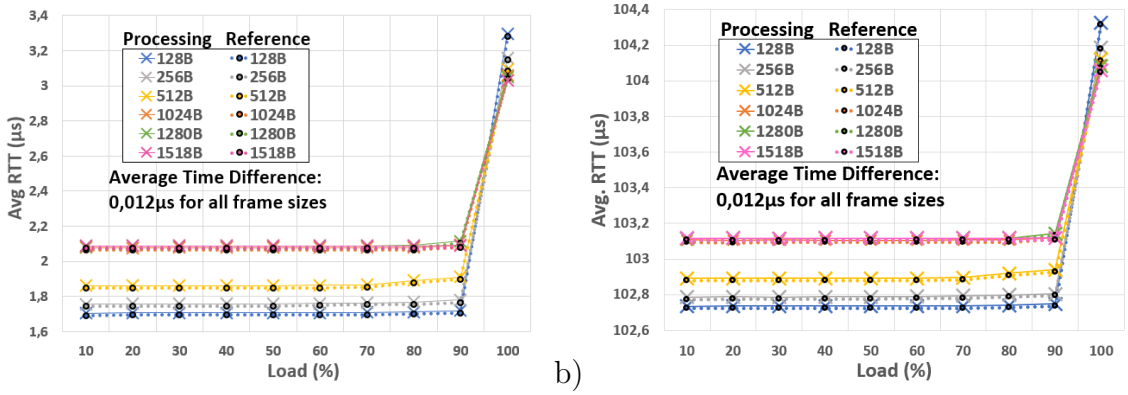


Figure 16 – Temps d'aller-retour (RTT) moyen en µs, selon les différentes tailles de trames en fonction de la charge en %, pour les setups avec et sans le protocole d'auto-négociation. Les cartes sont séparées par (a) 2m de fibre et (b) 10km de fibre.

	Frame size	128B	256B	512B	1024B	1280B	1518B
a)	Reference	0.012	0.009	0.01	0.009	0.013	0.048
	Processing	0.013	0.01	0.012	0.01	0.016	0.052
	Frame size	128B	256B	512B	1024B	1280B	1518B
b)	Reference	0.013	0.011	0.013	0.011	0.015	0.051
	Processing	0.013	0.01	0.012	0.01	0.016	0.052

Table 1 – Taux de trames perdues en ‰ selon les tailles de trames à 100% de charge, pour (a) 2m de séparation entre les cartes et (b) 10km de séparation. Les trames perdues sont nulles pour toutes les autres charges.

Interface-2T), représenté dans la Figure 17. Il est composé d'un FPGA avec un processeur logique Microblaze utilisant FreeRTOS, un transpondeur Nokia PSI-2T et un Raspberry Pi pour faire le lien entre le FPGA, relié par un lien Universal Asynchrounous Receiver-Transmitter (UART), et le transpondeur, avec qui il communique avec le protocole Secure Shell (SSH) via le port d'Interface en Ligne de Commande (Command Line Interface, CLI). Le temps d'aller-retour entre le FPGA et le PSI-2T est en moyenne de 108ms, et ce temps est majoritairement compris dans la communication SSH (environ 100ms en moyenne). Grâce au Raspberry Pi, il est possible de récupérer les données de monitoring du PSI-2T et de lui envoyer des commandes de reconfiguration depuis le Microblaze.

Pour notre prototype nous allons proposer de mesurer sur le PSI-2T la puissance de réception et le taux d'erreur binaire pré-FEC. Nous pensons qu'il s'agit de deux métriques simples et fiables pour détecter une baisse de qualité de service. Nous avons décidé de déclencher le processus d'auto-négociation quand la courbe de la puissance de réception

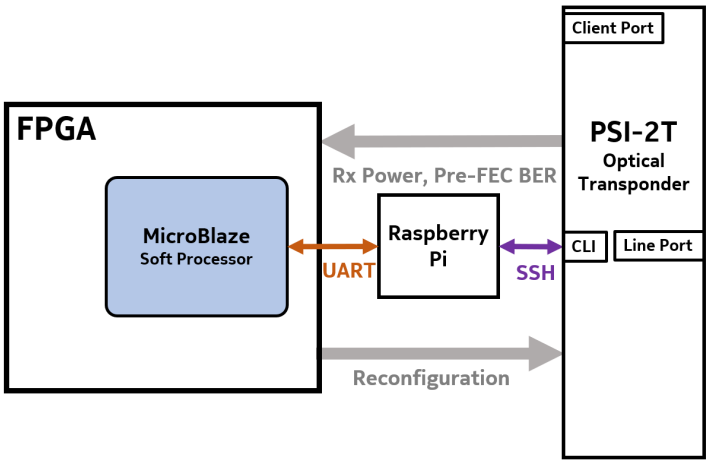


Figure 17 – Prototype de monitoring embarqué avec un transpondeur Nokia PSI-2T.

décroit au-delà d'un certain seuil et quand le taux d'erreur binaire dépasse un autre seuil.

La courbe de puissance de réception est calculée selon une fonction de régression linéaire, décrite dans [13]. La ligne de régression a pour équation :

$$\hat{y} = \alpha + \beta\hat{x} \tag{2}$$

Avec β la pente de la ligne de régression, qui a pour valeur :

$$\beta = \sum_i \beta_i y_i \tag{3}$$

Avec $i = 0, 1, 2...N-1$ où N est le nombre de samples utilisés pour le calcul. Suivant [13], et puisque nous avons avec notre système une fréquence de monitoring périodique (garantie par le système d'exploitation temps réel) on obtient comme valeur de β_i :

$$\beta_i = \frac{12 \cdot i - 6(N-1)}{N(N^2-1)} \tag{4}$$

Qui est une fonction facile à implémenter dans notre Microblaze, puisque la valeur de β_i ne dépend que du nombre de samples utiles pour le calcul, et il suffit de les multiplier avec les valeurs de puissance de réception capturées pour avoir la valeur de β. Pour pouvoir déclencher l'auto-négociation de façon pertinente il faut ensuite choisir un seuil pour la valeur β pour pouvoir détecter la dégradation et ne pas déclencher de fausses alertes. Nous avons réalisé des tests, en plaçant deux prototypes de la Figure 17 en communication

bidirectionnelle séparés par 10km de fibre, un Spirent pour générer du trafic Ethernet, et avec un Atténuateur Optique Variable (Optical Variable Attenuator, VOA) dans le chemin pour déclencher une dégradation de qualité de réception. Nous avons donc déclenché deux dégradations différentes : une rapide où l'atténuation passe de 10dB à 17dB en une fraction de seconde et une où l'atténuation est réalisée en 8 secondes. Pour les deux dégradations nous avons mesuré la puissance de réception et calculé les valeurs de β, en faisant varier le nombre de samples N entre 2 et 6. La Figure 18 montre les résultats pour (a) la dégradation rapide et (b) la dégradation lente.

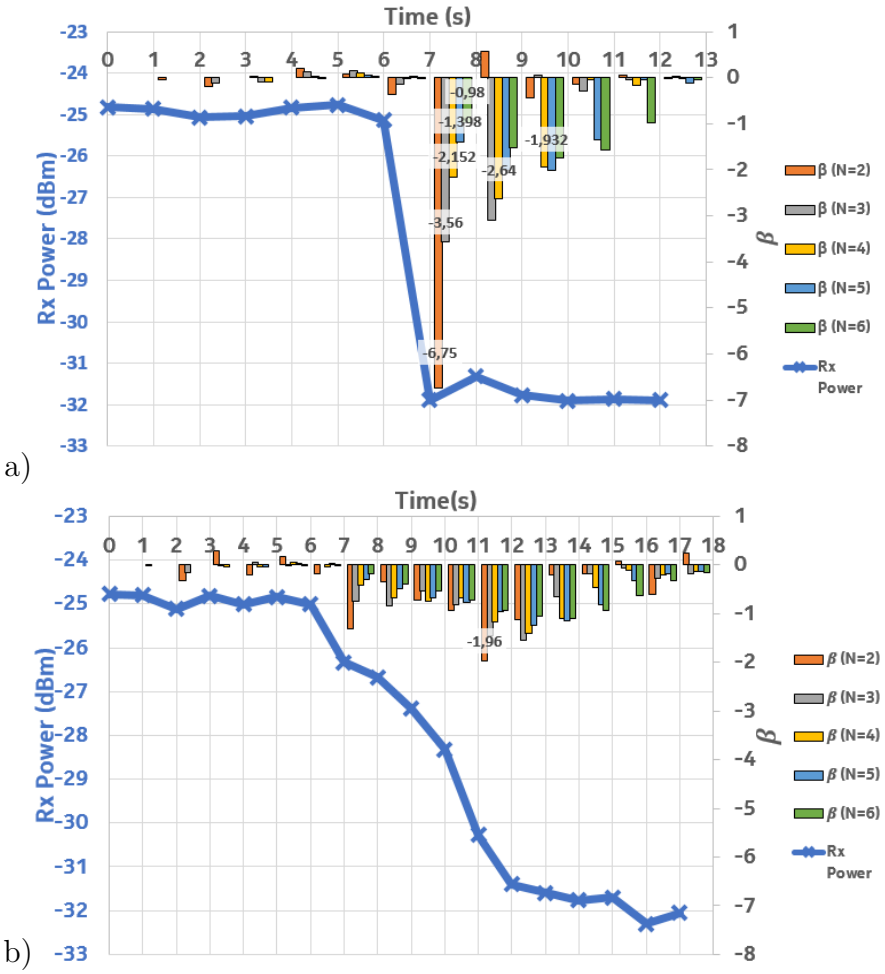


Figure 18 – Coefficient β de la courbe de puissance de réception en fonction du nombre de samples utilisés pour le calcul pour (a) une dégradation rapide et (b) un dégradation lente.

De nos mesures nous avons pu voir que le nombre de samples N influe fortement les valeurs minimales du coefficient β et le nombre de samples où la valeur de β reste basse

lors d'une dégradation. Moins il y a de samples, plus la valeur minimale de β est basse, mais en contrepartie la valeur de β reste basse moins longtemps. On voit aussi que si la dégradation est lente, la valeur minimale de la pente est moins élevée. Il faut donc choisir les seuils de façon réfléchie, selon le type de fautes à détecter.

Validation du prototype complet

Nous allons donc pouvoir présenter le prototype complet de transpondeur avec auto-négociation, qui est représenté en Figure 19. Nous avons les modules logique pour le protocole dans le FPGA, le PIS-2T et le Raspberry Pi pour faire le lien entre le transpondeur commercial et le Microblaze.

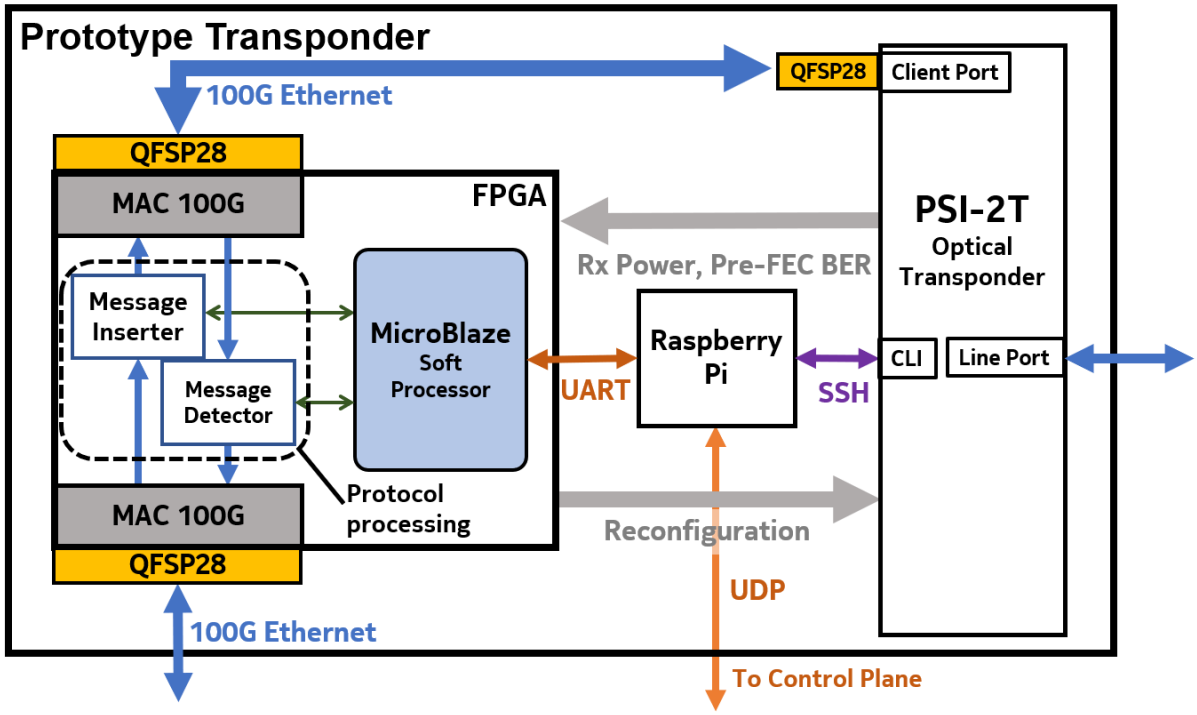


Figure 19 – Représentation de notre prototype complet avec FPGA, transpondeur commercial et Raspberry Pi.

La validation se passe au sein d'un banc de test réseau présenté en Figure 20, avec dans le plan de données deux transpondeurs prototypes en communication bidirectionnelle, 4 ROADMs (R1 à R4) et un VOA dans le chemin entre R1 et R2 pour déclencher à distance une dégradation de service et un Spirent pour générer et analyser le trafic 100G Ethernet. Dans le plan de contrôle il y a trois entités : le contrôleur SDN, le monitoring

system qui traite les données de monitoring et suggère des configurations du réseau au contrôleur SDN et l'Open Device Agent qui interface le plan de contrôle aux équipements du plan de données. Pour interfacer notre prototype au plan de contrôle nous avons développé un système de messages UDP (User Datagram Protocol) pour notifier quand une reconfiguration de notre prototype sera effectuée via auto-négociation et quand cette dernière a été effectivement effectuée, pour prévenir les conflits entre le plan de contrôle et notre solution.

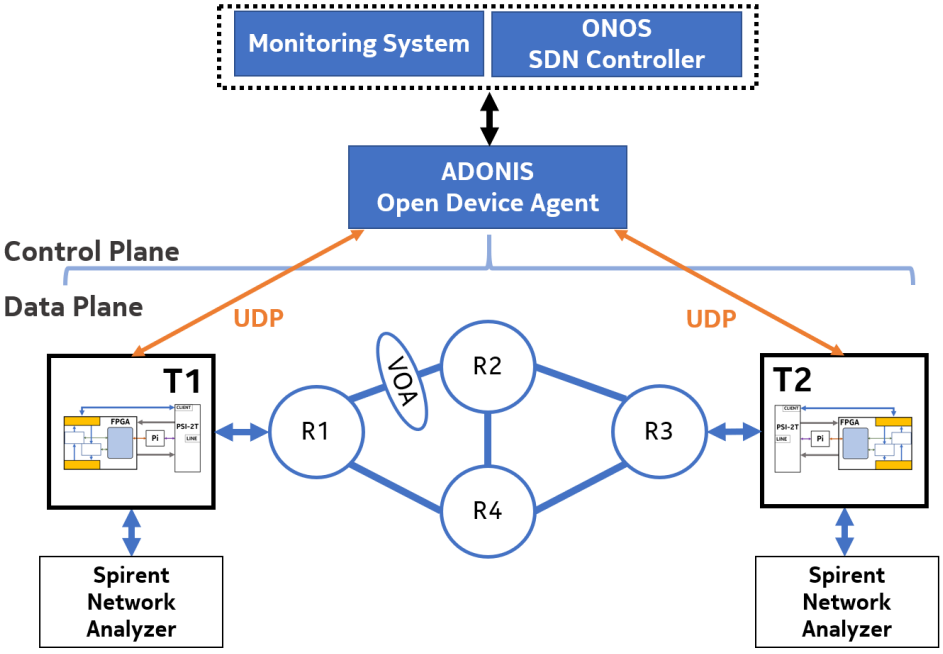
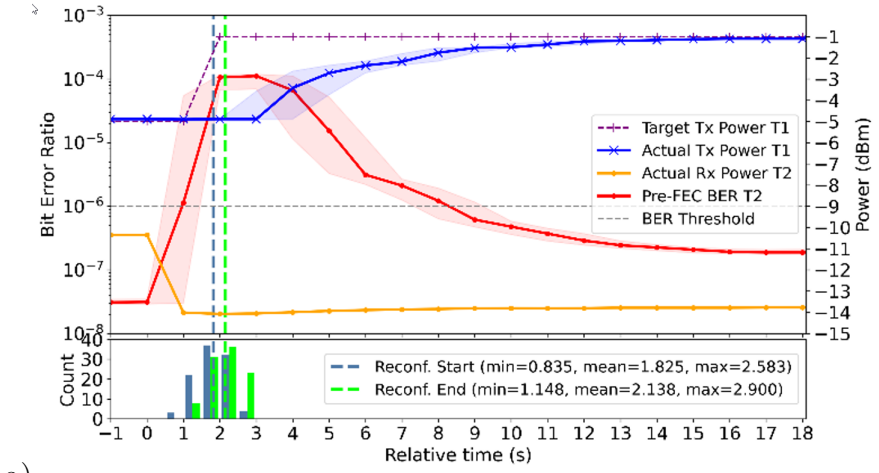


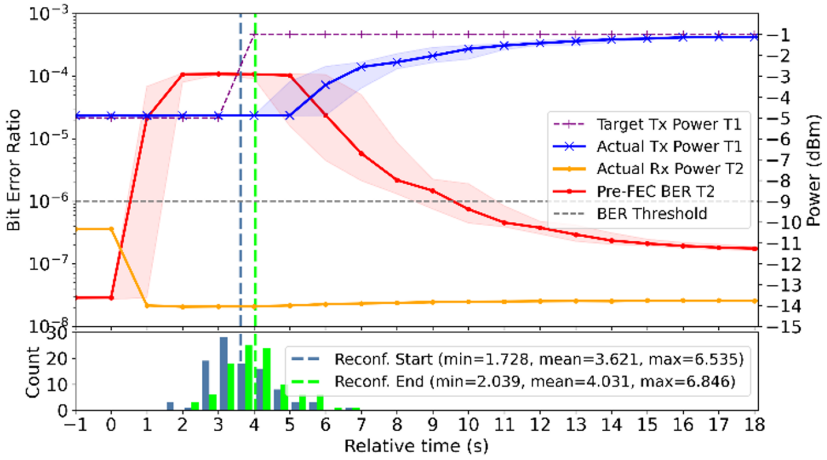
Figure 20 – Intégration de notre transpondeur prototype au sein d'un banc de test réseau.

Nous pouvons profiter de ces notifications et les comparer à des notifications similaires dans le plan de contrôle entre l'Open Device Agent et le contrôleur SDN pour faire des comparaisons de performance entre notre prototype et le plan de contrôle pour la résolution de dégradation de qualité de service.

Pour les deux setups, où la résolution de dégradation se fait par le prototype ou le plan de contrôle, nous avons choisi ces valeurs de seuils pour la détection de la dégradation : la courbe de puissance de réception (β) est calculée sur 4 samples et son seuil est de -1dB.s^{-1} et pour le BER pré-FEC le seuil sera de 1.10^{-6}. L'envoi de la commande d'augmentation de l'atténuation pour le VOA sera le t0 de notre setup. Les messages de notre prototype et du plan de contrôle indiquant que la reconfiguration va avoir lieu est labelisée Reconf.



a)



b)

Figure 21 – Résultats agrégés de 100 répétitions de notre protocole de validation pour (a) le setup où notre transpondeur prototype se charge de la détection et de la résolution de la dégradation induite par l'augmentation de l'atténuation dans l'atténuateur variable et (b) quand le plan de contrôle SDN s'occupe de la détection et la résolution. Les courbes représentent les valeurs médianes des métriques capturées, les parties transparentes représentent les 15.8 et 84.2 quantiles. Sous les courbes on a tracé les histogrammes de réception par le plan de contrôle des messages de notification Reconf. Start et Reconf. End.

Start et la notification indiquant que la reconfiguration s'est terminée est labélisée Reconf. End. La reconfiguration choisie pour nos tests est une simple reconfiguration de puissance d'émission pour compenser l'augmentation de l'atténuation, qui n'est pas une reconfiguration impressionnante techniquement mais qui permet de ne pas avoir d'interruption de service dans notre cas qui nous empêcherait d'avoir des résultats pendant des dizaines

de secondes. Les résultats médians de 100 répétitions de dégradation sont représentés en Figure 21.

On peut voir que notre prototype est bien capable de détecter et de résoudre la baisse en qualité de réception induite par l'augmentation de l'atténuation dans le VOA, nous pouvons aussi remarquer que notre prototype est de plus capable de résoudre la faute environ deux fois plus rapidement que le plan de contrôle centralisé, et avec moins de variabilité dans le temps. Cela est dû au fait que notre système, par sa localisation au plus près du matériel peut détecter plus vite la dégradation que le plan de contrôle qui est physiquement loin du matériel (et qui serait encore plus loin dans un réseau tel qu'implémenté par les opérateurs). De plus le plan de contrôle doit récolter toutes les données de monitoring de toutes les entités du réseau et les traiter ce qui crée des délais supplémentaires. Le protocole permet aussi de résoudre plus rapidement la dégradation que de passer par le plan de contrôle.

Intégration avec un setup temps-réel

Nous avons ensuite intégré notre solution d'auto-négociation au sein d'un ensemble émetteur-récepteur temps-réel capable de changer le baudrate de la communication sans interruption de trafic. Le setup est représenté en Figure 22. Le transmetteur est capable de passer d'une transmission de baudrate nominal à demi baudrate grâce à une technique d'entrelacement réalisée dans son FPGA, le Récepteur quant à lui peut décoder le trafic à baudrate nominal et demi baudrate sans adaptation grâce à une implémentation particulière de l'algorithme d'égalisation de module constant (Constant Modulus Algorithm, CMA) dans son FPGA réservé au DSP. Dans le Récepteur nous avons intégré une carte FPGA séparant les entêtes de la payload du trafic et permettant de gérer les messages de notre protocole d'auto-négociation. Une voie de contrôle est mise en place entre le Récepteur et le Transmetteur pour l'échange de message d'auto-négociation.

Nous avons donc déclenché grâce à notre protocole d'auto-négociation une reconfiguration du baud rate de 14GBd à 7Gbd. Les résultats capturés dans le FPGA sont dans la Figure 23. Le changement est effectué et synchronisé en 3 trames de 128 octets, notre système est capable de continuer à extraire des trames les entêtes et la payload aux deux baudrates.

Maintenant que nous avons pu montrer et quantifier les avantages de donner plus d'autonomie aux transpondeurs optiques pour permettre une plus grande réactivité et flexibilité pour les réseaux de transport optique, nous pouvons développer autour de cet

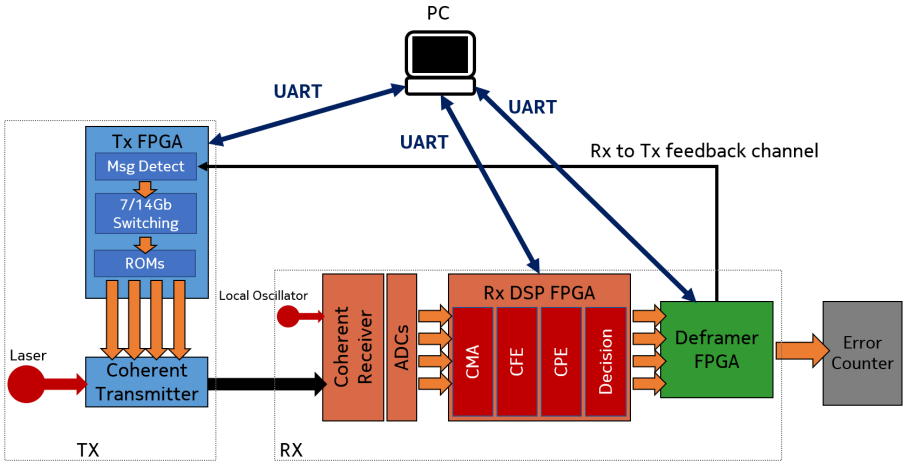


Figure 22 – Setup expérimental pour le changement de baudrate synchrone et sans perte de trafic en utilisant le protocole d'auto-négociation. Il est composé d'un Transmetteur (Tx) et d'un Récepteur (Rx) cohérents. Une voie de contrôle est mise en place entre la carte FPGA gérant le protocole d'auto-négociation et le Tx pour simuler une communication bidirectionnelle pour le protocole. Tous les FPGA du setup sont liés un ordinateur via un lien UART.

aspect, en développant des fonctions de monitoring plus élaborées, en donnant la possibilité au plan de contrôle d'exploiter les ressources computationnelles dans le réseau pour réaliser du monitoring complexe et de gagner en efficacité en déléguant certaines de ces tâches.

Service d'accélération et transpondeur

La recherche autour des techniques de monitoring dans les réseaux optiques montre un grand intérêt pour les techniques de Machine Learning [14], notamment les réseaux de neurones. Pour rappel, en Figures 24 et 25 nous avons représenté un réseau de neurones et la structure d'un neurone. Un réseau de neurones est structuré en plusieurs couches : une couche d'entrée, une couche intermédiaire dite cachée où à chaque neurone de cette couche chaque entrée est multipliée par un poids associé, sommées entre elles et avec un biais, puis passent dans une fonction d'activation. Les sorties des neurones de la couche intermédiaire vont ensuite dans la couche de sortie, où elles passent dans des neurones faisant les mêmes opérations. Les avantages des réseaux de neurones sont de pouvoir déceler des liens entre des données d'entrée et des phénomènes difficiles à modéliser mathématiquement, à condition d'être entraînés correctement, et d'être faciles à implémenter une fois entraînés.

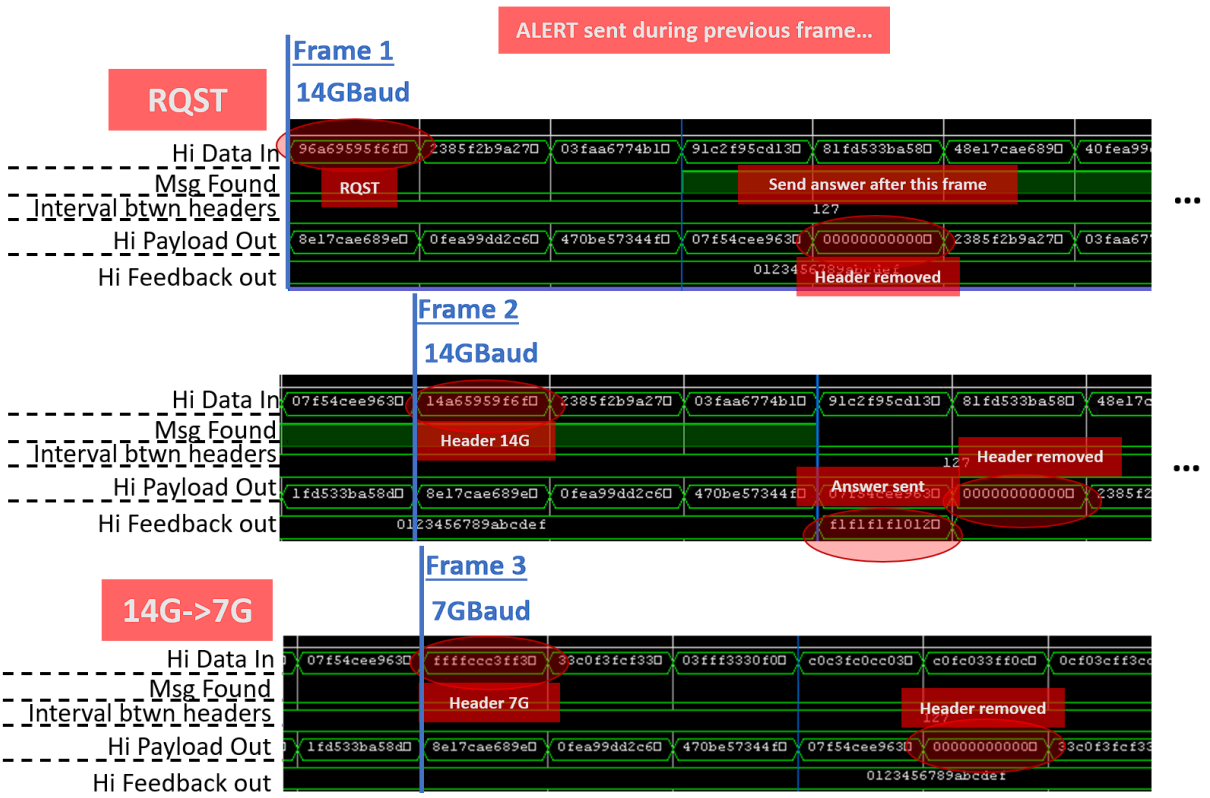


Figure 23 – Capture du FPGA gérant le protocole d'auto-négociation. On voit un changement de baudrate de 14GBaud à 7GBaud synchronisé grâce au protocole. Le message ALERT a été envoyé avant les trames capturées.

Dans les réseaux optiques par exemple, il est possible de déceler à partir des spectres optiques si le système est en régime fortement linéaire ou fortement non linéaire, voir Figure 26, mais il est très compliqué de quantifier précisément les contributions participant à chaque régime. Ce qui peut être intéressant car trouver le point d'équilibre entre les deux régimes permet de maximiser le ratio signal à bruit (Signal to Noise Ratio, SNR), voir Figure 27. [15] propose un réseau de neurones, prenant en entrée un spectre optique, avec une couche cachée de 10 neurones, et en sortie proposant une correction de puissance d'émission pour être au plus près possible du maximum de la courbe de SNR. Nous allons implémenter ce réseau de neurone sur FPGA et proposer un mécanisme de mise à jour des poids des neurones à distance pour permettre la mise à jour de la fonction proposée par le réseau de neurone, ce qui est une fonctionnalité intéressante dans le contexte des réseaux optiques virtualisés.

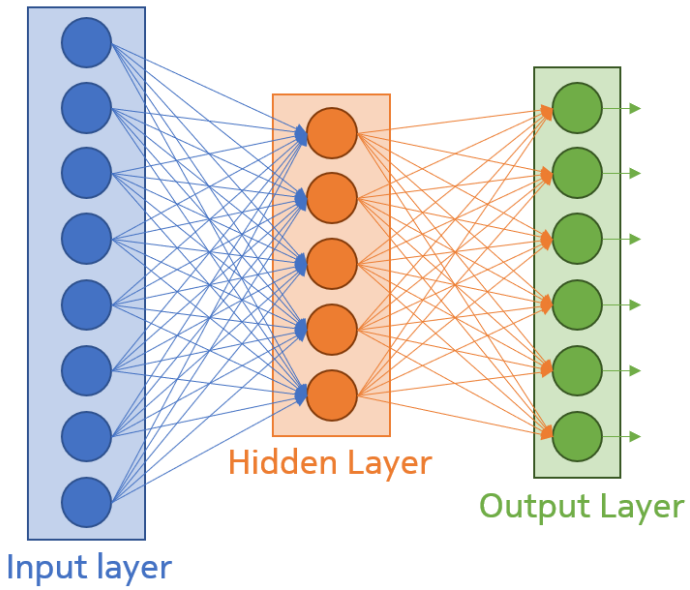


Figure 24 – Structure d'un réseau de neurone avec une couche cachée.

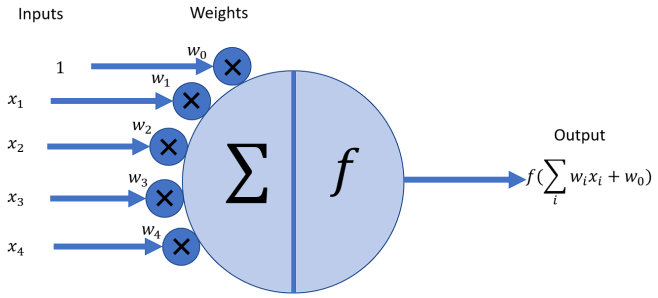


Figure 25 – Structure d'un neurone de réseau de neurone avec quatre entrées. Les entrées sont multipliées avec des poids correspondants, sommées avec un biais, et passent dans une fonction d'activation.

Réseau de neurone sur FPGA

Dans la Figure 28 nous présentons notre implémentation FPGA du réseau de neurone de [15], avec dix neurones dans la couche cachée et un neurone de sortie. Dans la Figure (a) nous montrons comment les flux de données circulent des entrées jusqu'à la sortie et dans la Figure (b) la logique de mise à jour des poids des neurones. Les données arrivent en série à l'entrée du réseau de neurone avec un signal de contrôle indiquant que les données sont valides. Dans notre implémentation les spectres optiques, nos entrées, sont samplées sur 251 symboles de 32 bits. Les entrées sont distribuées parallèlement à tous les neurones. Les sorties des neurones arrivent toutes en même temps grâce aux capacités

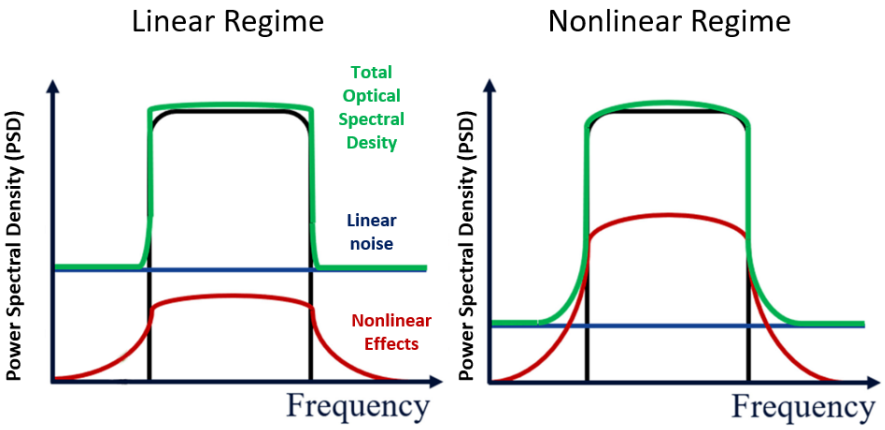


Figure 26 – Spectre optique dans le régime fortement linéaire et dans le régime fortement non-linéaire. Le canal modulé est représenté en noir, en bleu les contributions linéaires et en rouge les contributions non-linéaires. En vert est représenté le spectre total.

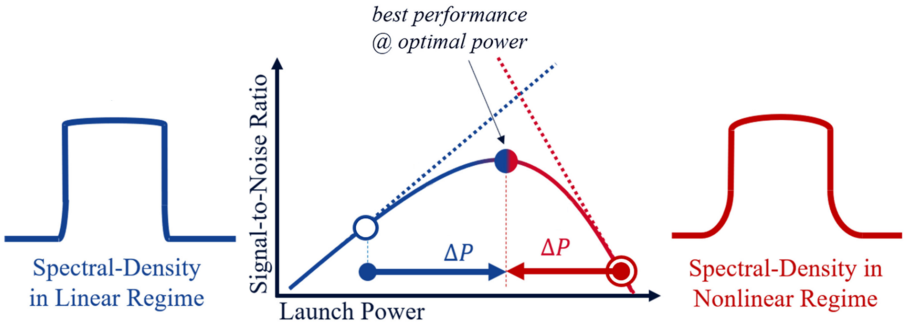


Figure 27 – Représentation du SNR d'un signal optique en fonction de la puissance d'émission. En bleu est représenté le système en régime fortement linéaire et en rouge en régime fortement non-linéaire. Il faut trouver la bonne puissance d'émission pour avoir la valeur de SNR au sommet de la courbe en cloche.

de parallélisme du FPGA et doivent donc passer par un sérialiseur avant d'être envoyées au neurone de sortie. Les poids arrivent aussi de manière sérialisée dans le réseau de neurone, avec leur signal de contrôle. Les poids du premier neurone de la couche cachée sont envoyés du premier au dernier poids puis le biais, puis le second neurone de la couche cachée etc.... Les poids du neurone de sortie arrivent en dernier. Les poids sont envoyés à tous les neurones en même temps, mais le signal de contrôle est réparti entre les neurones grâce à un démultiplexeur contrôlé par un compteur.

La structure des neurones de la couche cachée est présentée dans la Figure 29. Les neurones sont composés d'une mémoire contenant les poids. Pour multiplier les entrées avec leurs poids correspondants, un compteur permet de parcourir la mémoire séquen-

tiellement. Le compteur permet aussi de mettre à jour la mémoire lorsque des nouveaux poids arrivent dans le neurone. L'opération de lire la mémoire ou d'écrire dans la mémoire est décidée selon le signal de contrôle qui arrive dans le neurone. Les données hors du multiplicateur sont envoyées dans un accumulateur. Lorsque toutes les données d'entrées sont passées dans le neurone, le biais est ajouté à l'accumulateur. Puis la sortie de l'accumulateur est transformée et transférée à une mémoire contenant la fonction d'activation. La fonction d'activation des neurones de la couche cachée est la fonction tangente hyperbolique ($tanh$) qui est écrite comme :

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5}$$

Dans notre implémentation, les valeurs de fonction d'activation $tanh(x)$ stockées sont bornées suivant :

$$f(x) = \begin{cases} \frac{2}{1+e^{-2x}} - 1, & \text{if } x \in]-8, 8[, \\ -1, & \text{if } x \leq -8, \\ 1, & \text{if } x \geq 8. \end{cases} \tag{6}$$

Pour le neurone de la couche de sortie, la fonction d'activation est la fonction identité, donc dans notre implémentation la différence entre un neurone de la couche cachée et le neurone de la couche cachée est l'absence de cette mémoire.

Validation en simulation

Pour la validation en simulation de notre réseau de neurone sur FPGA, nous allons comparer les résultats en sortie de notre implémentation aux résultats d'un réseau de neurone généré sur Matlab. Nous allons faire passer 176 spectres dans les deux réseaux de neurones, et comparer la précision du réseau de neurone FPGA et le réseau de neurone Matlab en calculant l'erreur quadratique moyenne entre les deux. Nous allons faire aussi varier les tailles en bits des poids et de la fonction d'activation, ainsi que le nombre de valeurs stockées dans la mémoire contenant la fonction d'activation. Pour rappel, les valeurs en entrée sont codées sur 32 bits.

Tout d'abord, dans la Figure 30 nous présentons des captures de simulation fonctionnelle pour mesurer les performances en temps de notre réseau de neurone. On peut voir dans la Figure (a) qu'il y a 5 coups d'horloges entre la dernière valeur d'entrée et la sortie du neurone, ce qui est cohérent avec les opérations réalisées (à partir de la dernière

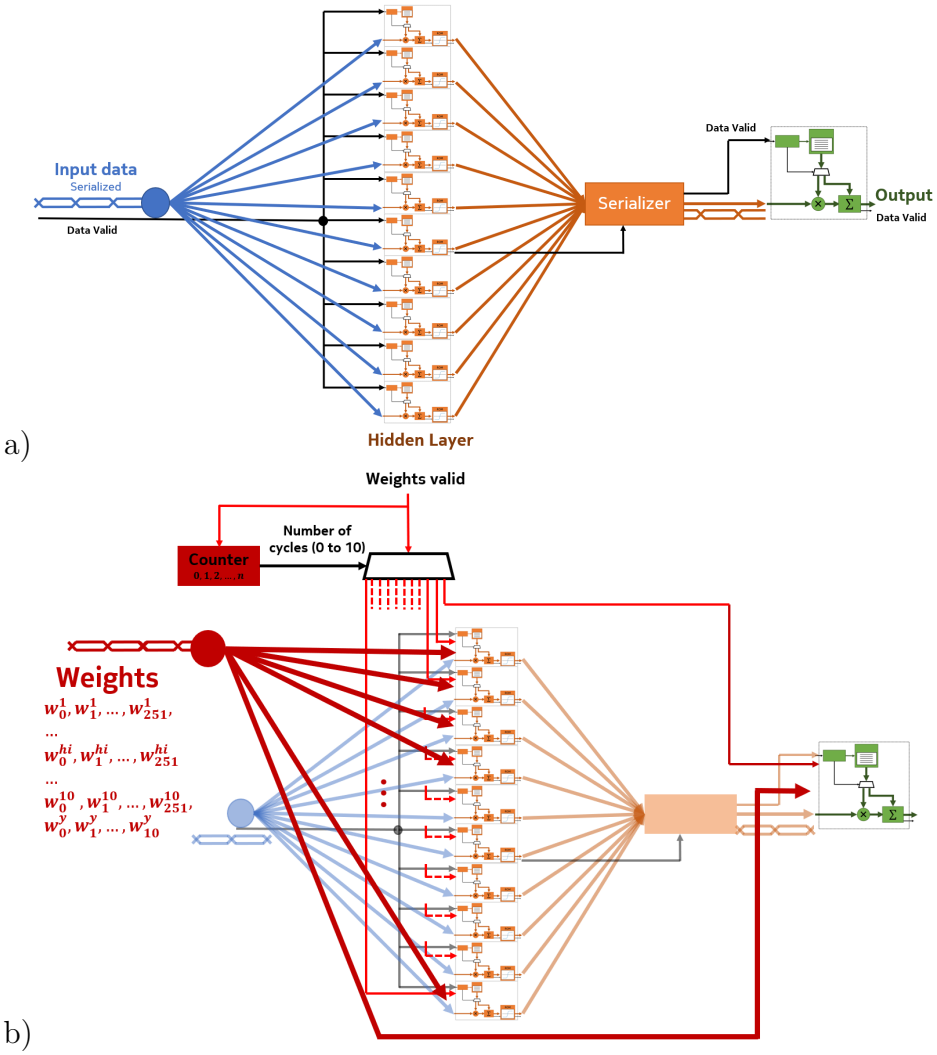


Figure 28 – Structure de notre réseau de neurones logique pour FPGA. La Figure (a) représente la gestion des flux de données circulent des entrées jusqu'à la sortie et dans la Figure (b) la logique de mise à jour des poids des neurones.

entrée : un coup d'horloge pour la multiplication avec le poids, un coup d'horloge pour l'accumulateur, un coup d'horloge pour l'ajout du biais, un coup d'horloge pour adapter la sortie de l'accumulateur et un coup d'horloge pour la fonction d'activation). On peut aussi voir dans la Figure (b) qu'il faut 2570ns entre la première donnée d'entrée et la sortie du réseau de neurone, et 200ns entre la dernière entrée et la sortie. A titre d'indication, Matlab donne un temps médian de 3.6ms pour réaliser les mêmes opérations, même si cette valeur est dépendant de nombreux facteurs le rapport est fortement à l'avantage de notre solution.

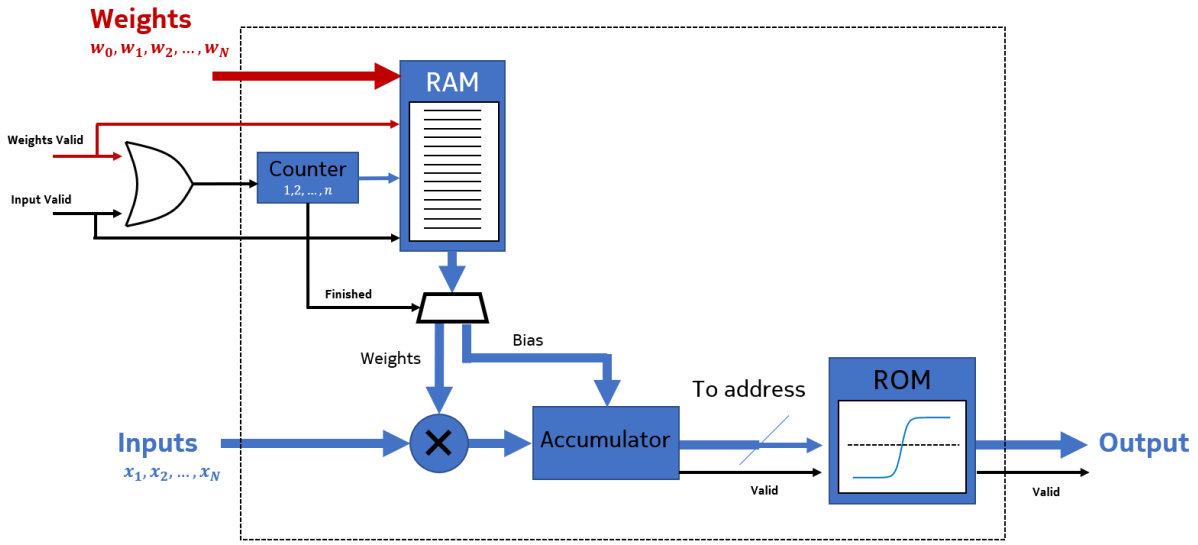


Figure 29 – Implémentation sur FPGA d'un neurone, avec en rouge la logique pour la mise à jour des poids et en noir les signaux de contrôle.

Dans le Tableau 2 nous montrons les résultats en précision comparé au réseau de neurone généré sur Matlab, selon différentes tailles en bits des poids et de la fonction d'activation et le nombre de valeurs de la fonction d'activation stockées. On voit que l'erreur quadratique moyenne au plus élevée, avec les paramètres les plus bas, est de $1.926 \cdot 10^{-3}$. La plus basse est de $5.572 \cdot 10^{-6}$, et n'est pas avec les paramètres les plus élevés. On voit bien la faible influence de l'augmentation de la taille en bit des valeurs de la fonction d'activation. La valeur récupérée dans la mémoire étant fortement liée aux calculs précédents, et le neurone de sortie ne possédant pas de fonction d'activation, la taille des poids a un impact beaucoup plus significatif sur la précision de la valeur finale en sortie du réseau de neurone. Le choix des différents paramètres doit donc être pris en considération lors du déploiement de la solution, car elle impacte non seulement la précision du réseau de neurones mais aussi l'utilisation de ressources de notre design.

Validation en implémentation

Pour la validation en implémentation, nous proposons un banc de test permettant de récupérer des spectres et des nouveaux poids à distance pour valider notre implémentation de réseau de neurones. Ce setup est représenté en Figure 31.

Dans ce setup nous avons un processeur logique Microblaze, tournant sur le système d'exploitation temps-réel FreeRTOS. Nous avons implémenté un cœur Ethernet à 1G

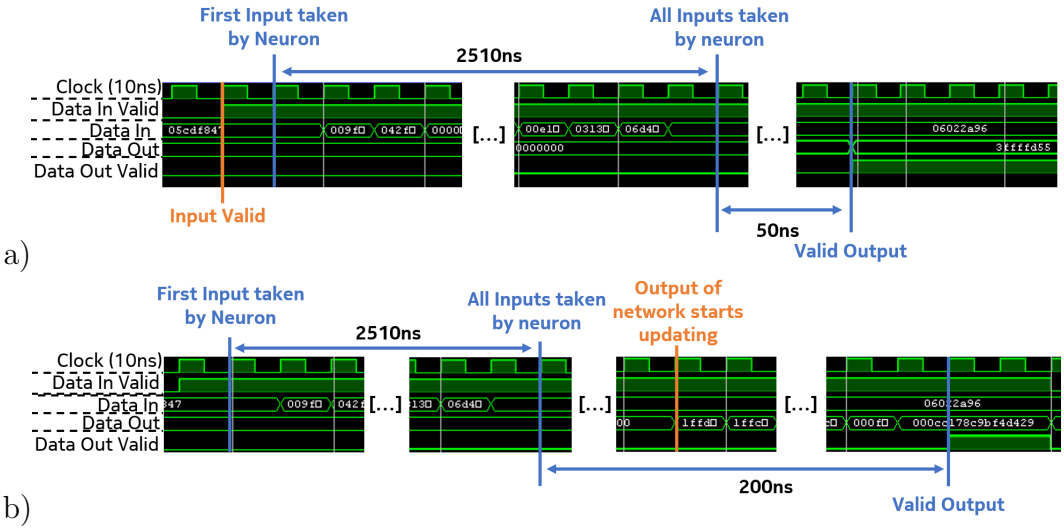


Figure 30 – Captures de simulation fonctionnelle (a) d'un neurone (b) du réseau de neurones.

(avec un cœur Direct Memory Access DMA et de la mémoire DDR4 pour gérer l'envoi et la réception de paquets) afin de pouvoir communiquer avec un serveur web qui contiendra les spectres pour la validation ainsi qu'un deuxième set de poids pour mettre en œuvre la réécriture des poids du réseau de neurones. Le premier set de poids sera appelé le set initial (Initial Set) et correspond au set de poids utilisé pour la validation en simulation, et le deuxième set sera appelé le set final (Final Set). Pour interfacer le Microblaze et le réseau de neurones, nous avons développé un wrapper compatible avec le protocole AXI-4 Stream et ajouté des First In First Outs (FIFOs). Le FPGA transmet les valeurs de sortie du réseau de neurones à un ordinateur via un lien UART. Comme précédemment les entrées sont encodées sur 32 bits, et pour les deux sets de poids il y a 176 spectres qui

Activation Function / Weights	Depth: 2^{10} Width: 16	Depth: 2^{11} Width: 16	Depth: 2^{12} Width: 16	Depth: 2^{10} Width: 32	Depth: 2^{11} Width: 32	Depth: 2^{12} Width: 32
Width: 16	$1.926.10^{-3}$	$1.768.10^{-3}$	$1.704.10^{-3}$	$1.927.10^{-3}$	$1.768.10^{-3}$	$1.705.10^{-3}$
Width: 32	$8.773.10^{-5}$	$2.234.10^{-5}$	$5.572.10^{-6}$	$8.773.10^{-5}$	$2.238.10^{-5}$	$5.604.10^{-6}$

Table 2 – Erreur quadratique moyenne (MSE) de notre implementation FPGA du réseau de neurones comparé à celui généré dans Matlab pour différentes tailles en bit des poids et la fonction d'activation (width) et différents nombre de valeurs de fonction d'activation (depth).

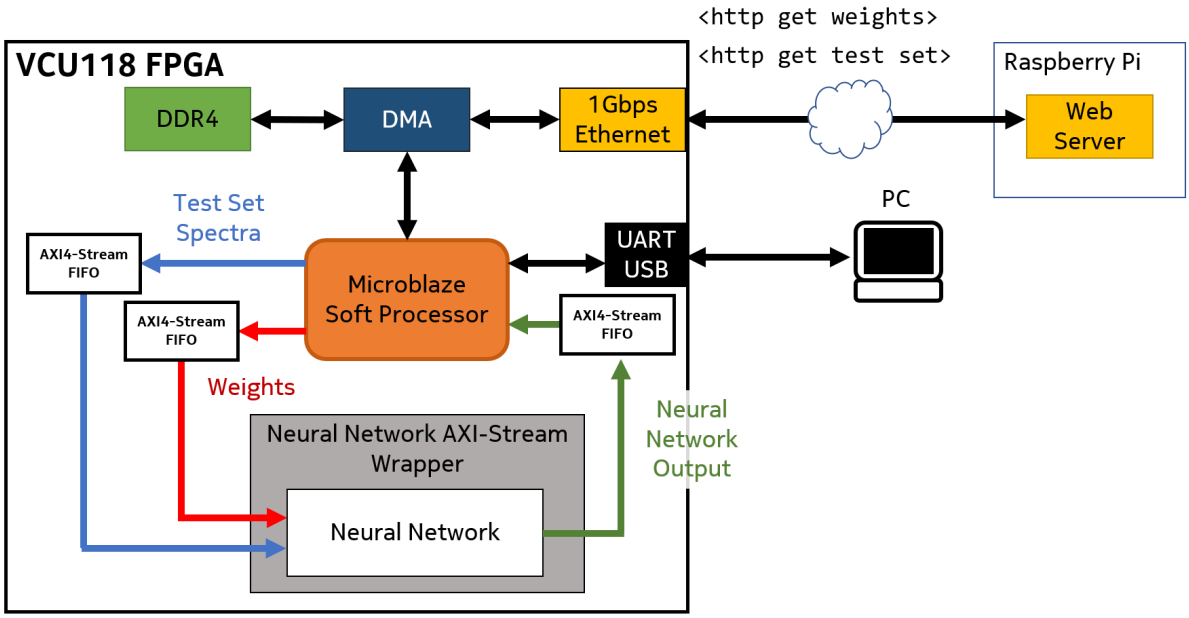


Figure 31 – Schéma de notre banc de test avec un processuer logique Microblaze, le réseau de neurone et un serveur web pour récupérer les spectres et les poids.

passeront dans le réseau de neurones. Les poids et la fonction d'activation sont écrits sur 8 bits, et 1024 valeurs de fonction d'activation sont stockées.

Dans le Tableau 3 nous avons regroupé les résultats pour cette validation. Toutes les valeurs de sortie du réseau de neurone sont bien récupérées, et ces valeurs correspondent bien aux valeurs obtenues en simulation. Le temps moyen entre la réception d'un spectre complet et la réception de la valeur en sortie du réseau de neurones est très proche, ce qui suppose un résultat quasi instantané, ce qui est confirmé par la capture de la logique d'entrée et de sortie du réseau de neurones, où il y a 230ns entre la dernière entrée et la sortie du réseau de neurone. Les 30ns ajoutées par rapport à la validation en implémentation proviennent exclusivement du wrapper.

Nous avons donc présenté et validé un réseau de neurone sur FPGA permettant la reconfiguration des poids des neurones, permettant donc de reconfigurer la fonction du réseau. Intégré à un transpondeur avec auto-négociation, cela permettrait d'avoir accès à des fonctions de reconfigurations avancées et donc procéder à des reconfigurations plus intelligentes, tout en ouvrant la possibilité au plan de contrôle de pouvoir utiliser ces ressources computationnelles pour réaliser du monitoring déporté. Cette solution pourrait être encore améliorée en permettant la reconfiguration dynamique du nombre d'entrées, du nombre de neurones de la couche cachée et du nombre de neurones de sortie, pour

	Initial Set	Final Set
Number of values acquired from NN	176 (100%)	176 (100%)
Correspondence with simulation values	100%	100%
Ticks to receive whole test set from webserver	474	466
Average ticks to receive one spectrum	2.693	2.648
Ticks to get all NN values	475	466
Average ticks between NN values	2.699	2.648

Table 3 – Résultats d'implémentation récupérés du Microblaze. Un tick processeur correspond à 10ms.

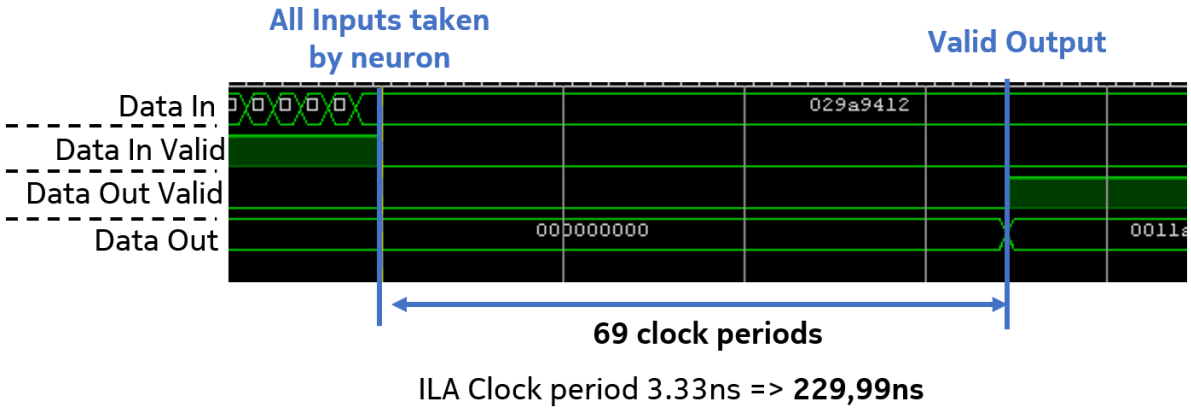


Figure 32 – Capture of the implemented neural network.

permettre d'augmenter les possibilités de fonctions réalisables.

Conclusion

Dans ce manuscrit de thèse, nous avons proposé des solutions pour apporter plus d'intelligence au transpondeur optique dans le contexte des réseaux de transport optiques flexibles. Nous pensons que ces possibilités supplémentaires permettront d'augmenter la flexibilité et la réactivité des réseaux optiques. Les possibilités offertes par l'ajout de ressources computationnelles dans le réseau et exploitables par le plan de contrôle pourront permettre au plan de contrôle de réaliser du monitoring délégué sur des ressources

sous-utilisées du réseau et nécessitant des gros flux de données. Ces solutions pourraient participer au développement et à l'adoption de réseaux optiques flexibles et hautement automatisés.

Chapter 1

Introduction

Optical fiber is currently the most commonly used medium to transport information at extremely high speed and at great distances. Since the boom of Internet usage at the end of the 90s, with the greater than ever number of devices capable of connecting themselves to the Internet and with applications requiring higher data-rates and greater reliability, optical networks have replaced the copper networks, and are deployed closer and closer to the homes of the end-users. Furthermore they are still considered the most suitable way to build networks for the forseeable future.

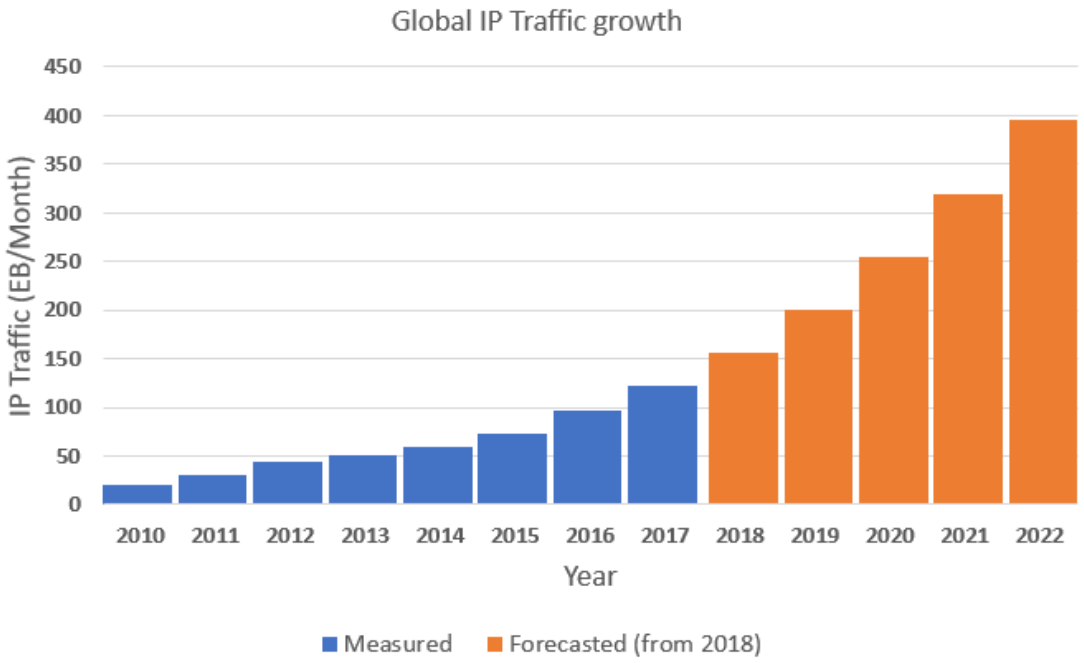


Figure 1.1 – Global IP traffic growth in Exabytes per Month measured and forecasted by Cisco. Measurements pulled from Virtual Networking Indexes of their respective years, and forecasts from the Virtual Networking Index 2017-2022 [1].

In [16], Cisco foresees that two thirds of the global population will have internet access and that the number of devices connected to IP networks will be three times the global

population by 2023, from 51% and 2.4 times the global population respectively in 2018. And with video having a big impact on data consumption and considering the adoption of Ultra HD video streaming (see Figure 1.2) and remote working favoring video calls [17], demands in capacity are expected to skyrocket in the upcoming years, see Figure 1.1. The adoption of 5G for the end-user and for the industry is also expected to increase the overall data consumption and create demands for low latency connection.

If backbone (or long-haul) links that connect countries and continents are concentrating a lot of data traffic, regional and metropolitan links are under more and more pressure. As [1], traffic within metropolitan networks will grow from 27% in 2017 to 33% in 2022, which means that over one-third of the Internet traffic will stay close to the end-user in the upcoming years. This is due to the increase of population in metropolitan areas, the adoption of Fiber To The Home which boosts data rates for users and changes their usage of the Internet, and the development of edge-cloud and edge-computing which puts application services closer to the end-user to reduce latency.

In response, optical network and optical equipment design have to adapt themselves to sustain these growths.

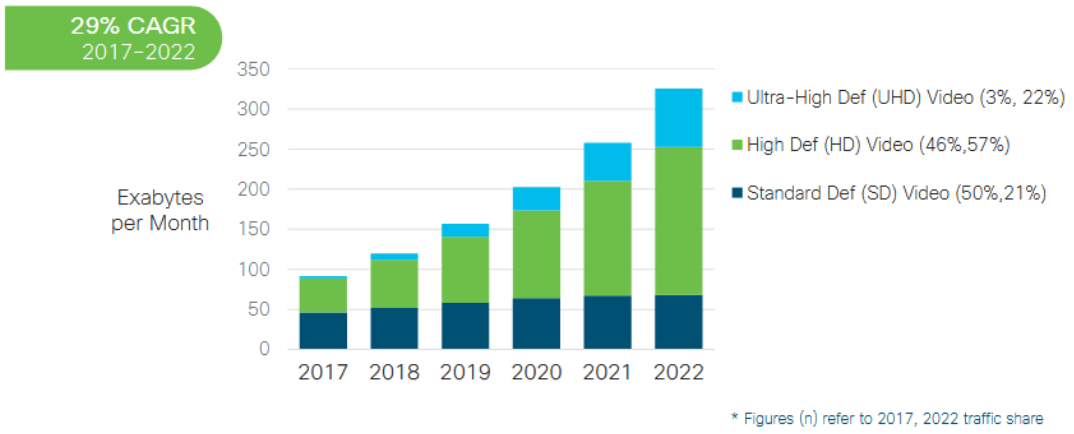


Figure 1.2 – Impact and growth of video application traffic and influence of the definition on this impact. Source [1]

Increase in computing resources available in optical equipment, deployment of coherent optical communication and research on advanced Digital Signal Processing (DSP) techniques has resulted in a boost of the maximum achievable data rates and the maximum achievable distances without added amplifiers. Other works have focused on increasing the possible configurations of the optical equipment. Being able to use a wide array of parameters depending on the circumstance can be extremely beneficial and allow for adaptations

during the whole equipment lifecycle. In the end these adaptations can lead to better reliability in the network overall and an increase in average capacity, if the equipment is frequently reconfigured to adopt an ideal set of parameters depending on the situation. These equipment that are able to reconfigure themselves to another set of parameters are qualified as flexible. However, increasing the possible configurations of the equipment of the network brings two main problems: it increases the network complexity and reconfiguring the equipment is beneficial only after the equipment is reconfigured, as during the operation it is most likely to be in a maintenance mode which renders it out of service. The former problem can be alleviated by introducing automated centralized intelligence in the network, that can oversee the equipment and their condition and propose automatically new set of parameters. This requires adaptation in the device so that it is able to communicate with a centralized intelligence and be remotely piloted. By doing this, the centralized control can interact in novel ways with the hardware by virtualizing it, i.e. separating the equipment hardware and software (or application). This allows the complete reorganization of the network to fulfill specific conditions (for example low latency, higher throughput, better security, etc...) at a global or a local scale. The latter problem can be solved by designing hardware that can be reconfigured quickly, and even better if it is able to be reconfigured quickly enough to avoid any loss of traffic, which can be qualified as hitless.

These solutions are currently being implemented, some even in commercial products, and have lead to a great number of structural changes in optical networks. The introduction of the flexible grid in optical Wave Division Multiplexing (WDM) transport networks and the adoption of the Optical Transport Network (OTN) toolbox has increased the possibilities for routing and managing the client traffic, the optical transponders (the entry and exit points of the client signals into the optical transport networks) and the Optical Add-Drop Multiplexers (OADMs, the switches of the transport networks) have evolved to gain more (re)configuration possibilities and the deployment of the Software Defined Network (SDN) control plane architecture has enabled more automation and novel hardware management techniques such as the virtualization. All these progresses have enhanced the capabilities of optical networks at the metropolitan and regional levels but more work has to be done to attain full programmability at a high speed and take advantage of the SDN architecture.

This manuscript will, in regard to this context, elaborate on the subject of real-time flexible and virtualized transponders for optical communications. This thesis work has

been realized under an industrial grant between the IETR lab of INSA Rennes and Nokia Bell Labs Paris-Saclay. This manuscript is organized in 4 chapters presenting the three years of research work.

Chapter 2 develops the overall context of the thesis work. This chapter opens by describing the optical transport network, with the relevant current technologies and equipment. Following this is a detailed description of the optical transponder, which is at the center of the research work of this manuscript.

Chapter 3 centers around the notion of flexibility, at the hardware and network levels, which are relevant for this thesis work. This is followed by a review of the evolutions of the optical transponders that have happened in the last decade, which has enabled the Bandwidth Variable Transponder (BVT) and the Sliceable Bandwidth Variable Transponder (S-BVT) and a state of the art on the current research on hitless and automatically reconfigurable transponders.

In Chapter 4 we present a real-time transponder prototype with monitoring and auto-negotiation capabilities for automated fast and synchronized reconfigurations. We present our prototype and its different elements, and validate it inside a network testbed and measure its performance against a centralized control plane in term of time to detect a degradation in quality of service and resolve the degradation by reconfiguring the transponder. We also integrate our auto-negotiation setup with a real-time transmitter and receiver able to instantaneously reconfigure the communication baudrate, and showcase a synchronized change of baudrate.

In Chapter 5 we present an embedded Artificial Neural Network (ANN) solution on Field Programmable Gate Array (FPGA), and validate it by implementing a function to monitor nonlinear contributions by analyzing optical spectra. We asses its timing and precision performance, and develop a function to update the network's neurons weights to reconfigure the performed function. This reconfiguration could be used so that the control plane can perform remote monitoring to speed up fault detection.

We finally conclude this manuscript by providing a overall summary of the work achieved and finishing on prospects on possible further developments.

Chapter 2

The optical transport network and the optical transponder

Optical networks are at the center of modern telecommunications, carrying traffic for a whole range of applications, interconnecting uncountable numbers of users and services at high speed. But with the increase in number of end-users, the development of new wireless communication technologies and the ever increasing data-hungry cloud services, optical networks had to evolve to sustain this non-stopping growth in network capacity demands.

Since the first optical fiber under a 20dB/km loss in 1970 [18], that kickstarted the development of optical networks, optical fibers has kept improving, using different materials for the core, with the currently most common using silica, allowing for long reach reliable transmission of the light, with losses around 0.15 dB/km [19]. These evolutions have been coupled with improvements in optical equipment design, allowing higher than ever capacities at greater distances. Optical signal detection used direct detection, recovering only light intensity, but improvements on signal processing has allowed around 2008 the on field deployment of coherent detection and communication [20]. In coherent detection the information is mapped onto the light intensity, phase and polarization, which greatly improves the spectral efficiency, the rate of transmission of information over a given bandwidth (in bits per second per hertz) and is now the standard in high speed optical communications.

With fast coherent optical communication possible, the next evolution of optical systems is the integration of flexibility concepts, allowing the tuning of the parameters of the equipment to better match the needs in capacity and/or reliability. These changes can range from the tuning of a laser to a different wavelength, the change of modulation format to achieve better bitrate or higher reach without added amplification or the rerouting of one or multiple signals across the whole optical network, and they can be triggered remotely or by an internal function inside the equipment. Alongside the increase

of possibilities in the data plane came in the control plane an increase in processing power and improvements in overall architecture. Where optical networks were globally extremely static, the 2010s have started the transition into fully automatic flexible optical networks, where the control plane collects the monitoring data of the equipment, detects drop in quality of service and acts accordingly, sending reconfiguration and routing commands. Coupled with the new telemetry services, boosting the rate of monitoring data collecting, and the implementation of more complex algorithms to process the data, future optical networks will become much more reliable and efficient.

This chapter presents in more details the context of the thesis work. The first section describes the optical transport network with its current technologies on the data plane and the control plane that are relevant for the work presented. Following this is a section covering the optical transponder equipment, the optical device at the center of this thesis work, with a comprehensive description of its structure and its components.

2.1 Optical transport networks

2.1.1 Optical layer architecture

Optical transport networks are the network high capacity highways between node pairs of the network. Organized in rings or meshed topologies, they can cover from tens to hundreds of kilometers for metropolitan (or metro) networks, or thousands of kilometers for core networks and are at the center of the exponential traffic growth [2].With the ever growing bandwidth and reliability needs with the implementation of 5G and the increasing number of device connected to the internet, optical transport networks have evolved to sustain this trend.

Today's main optical technologies for transport network are the Wave Division Multiplexing (WDM) equipment and the Optical Transport Network (OTN) standard, that are prevalent in modern operators networks. WDM is represented in Figure 2.1 and is a technology where a single optical fiber is carrying several multiplexed signal or signals over different wavelengths, and its usage helped boost significantly the capacity of optical networks, multiplexing multiple client signals that are then carried along the network and demultiplexed and dropped at their destination. The OTN standard works as a toolbox for optical networks, allowing to manage client signals and wavelengths in the network, providing a frame format with overhead carrying useful information for client signals

monitoring, routing and grooming over optical channels. The standard for OTN is published by the ITU (International Telecommunications Union) in [21]. As seen in Figure 2.2, the OTN frame is composed of multiple sub-headers and sections that encapsulates the client data. The OPU (Optical Payload Unit) encapsulates the client data and its header details the type of data that is transported. The Optical Data Unit (ODU) overhead has segments that helps monitoring the optical path and switching the data from a working optical path to a restoration path in case of a failure in the working optical path. The OTU (Optical Transport Unit) adds further monitoring of the optical link and error correction. A frame alignment sequence and a error correction code are added for more reliable communication. OPU, ODU and OTU are often accompanied with an index k representing the standard of the OTN signal, with a higher index indicating higher maximum bitrate. Also, k-indexed frames are able to transport multiple frames of lower indexes by time-multiplexing them, see Figure 2.3 and [22]. As an example, ODU3 frames are used to transport 40 Gigabit Ethernet frames in the optical transport network or up to four ODU2 signals, that are suitable to transport 10 Gigabit Ethernet Frames. 100 Gigabit Ethernet frames are currently transported in ODU4 frames, that can also transport up to two ODU3 frames. This structure allows the transportation of multiple low speed signals into a single high speed entity.

Furthermore, to accompany the ever-growing increase in traffic in transport network and the greater number of possible configurations of optical equipment, multiple techniques have enabled Elastic Optical Networks (EON), which is the new paradigm in optical network design [23]. The basis of EON is that the hardware is not considered as a static device but can be programmed during its lifetime in the network, possibly allowing the tuning of equipment parameters to match the current needs in capacity and/or reliability, for example the modulation format, the Forward Error Correction (FEC) code, the central frequency of the transmission etc... This freedom in the choice of configuration adds a much needed layer of flexibility to optical transport networks.

The other key technology enabling elastic optical networks is the introduction of flexible grid in WDM-based optical networks. While WDM networks traditionally use a fixed grid where client signals are allocated on a specific 50GHz slot, flexible grid allows for a finer granularity in frequency slots. The flexible grid motivation was to allow mixed bit rate or mixed modulation format systems to allocate frequency slots [3], enabling better spectrum occupation in regards to the network needs in capacity and possibly savings in spectrum occupancy as shown in Figure 2.4. However as the spectrum slots allocated

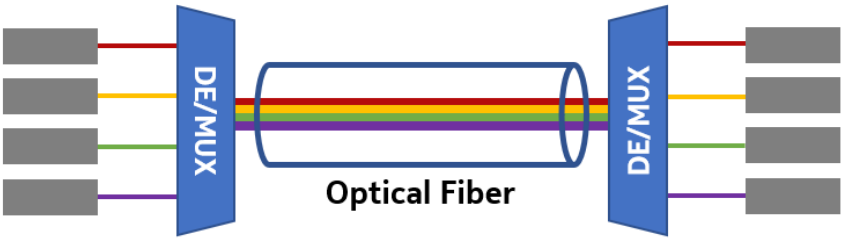


Figure 2.1 – Simple representation of the WDM technology

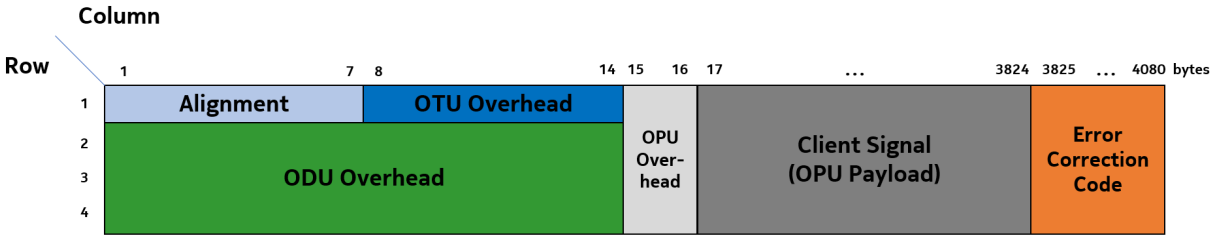


Figure 2.2 – Structure of an OTN frame

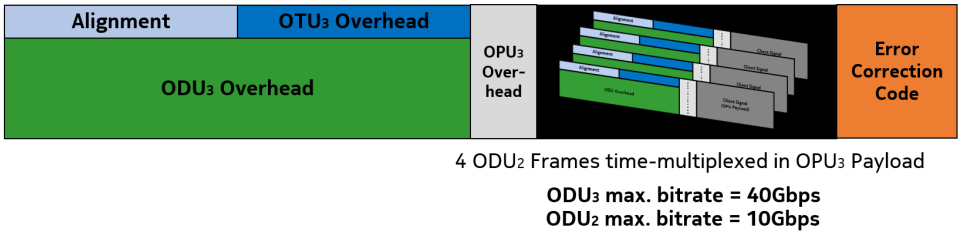


Figure 2.3 – Representation of an OTU3 frame transporting 4 ODU2 frames using time multiplexing.

across the whole network mustn't overlap each other if they take the same lightpath, because it degrades the quality of reception at the optical node, spectrum allocation becomes much more difficult in a flex-grid scenario, because of the increased number of signal that can be allocated on the spectrum [24, 25]. An example of signal overlapping on the flexgrid due to poor slot assignment can be seen in Figure 2.5. Coupled with the introduction of the super channel technique, see Figure 2.6 that allows multiple signals to be modulated, multiplexed and transmitted as a single entity, creating a signal with possibly greater bandwidth than 50GHz [26], spectrum allocation requires very mindful orchestration to ensure a good quality of service in network.

Figure 2.7 is a schematic representation of an optical transport network data plane. We can see multiple network segments. Firstly the access segment is the part of the network closer to the end-user, providing connectivity for wireless mobile network base stations,

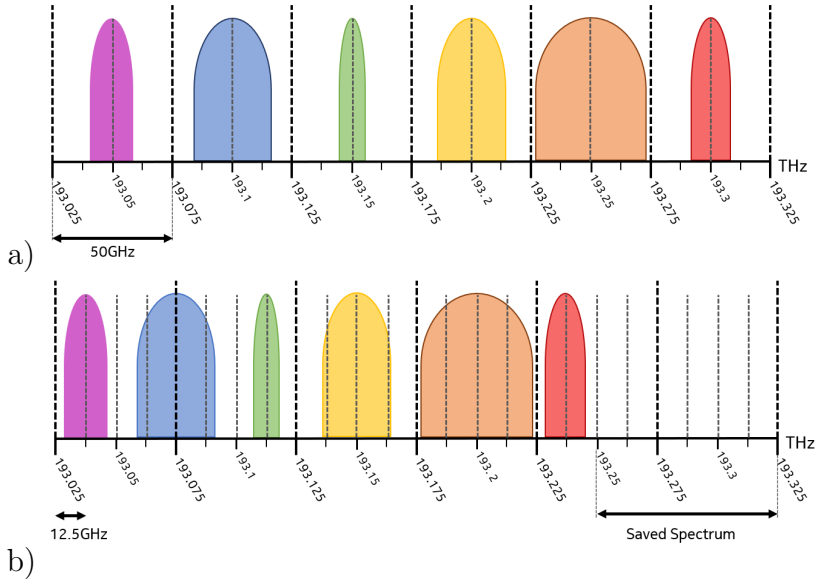


Figure 2.4 – Example of spectral slot allocation in a)fixed and b)flex grid scenario, with a 12.5GHz granularity for b)

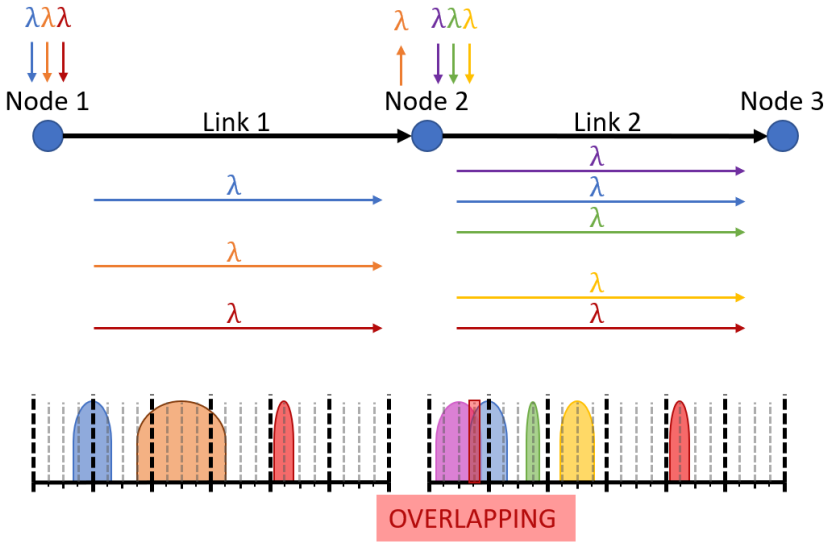


Figure 2.5 – Example of a spectral slot assignment causing overlapping in a flex-grid scenario

enterprises and residential endpoints for example. The traffic coming to and from the access segments is named the client signal in transport networks and is received by an optical transponder that serves as an entry and exit point for the signal in the metro network segment. The client signal is multiplexed onto a single carrier and sent to the

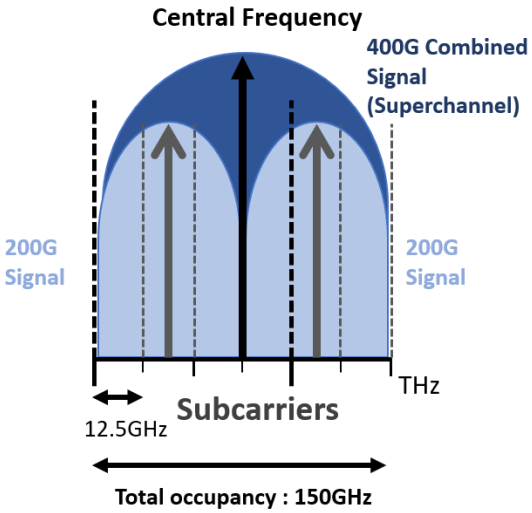


Figure 2.6 – Example of a super channel in the flexgrid, where two 200G signals are combined into a 400G super channel

metro segment, where it is multiplexed on WDM signal, transported and dropped at its destination out of the metro segment by Optical Add-Drop Multiplexers (OADMs), where it is demultiplexed by an optical transponder and go to its final destination in the access segment. If the final destination is in another metro ring, the signal can go through the core segment, interconnecting multiple metro networks together, and go if necessary to another core segment by going through submarine cables for example. Traditionally, the metro segment's topology is organized in rings and the backbone network's is meshed, the meshed topology allowing for more restoration possibilities in case of a degradation [27], though it is more and more common to see the metro segment organized in the meshed topology.

2.1.2 Optical line equipment

In this subsection we will present the optical equipment composing the optical line in transport networks (see Figure 2.8)

2.1.2.1 Optical Transponder

The optical transponder, represented in Figure 2.9 is an optical-electrical-optical equipment, that gathers the low speed signals, called "colorless signals", on client ports to multiplex them on a WDM-compatible carrier, or "colored signal", that is going to be sent to

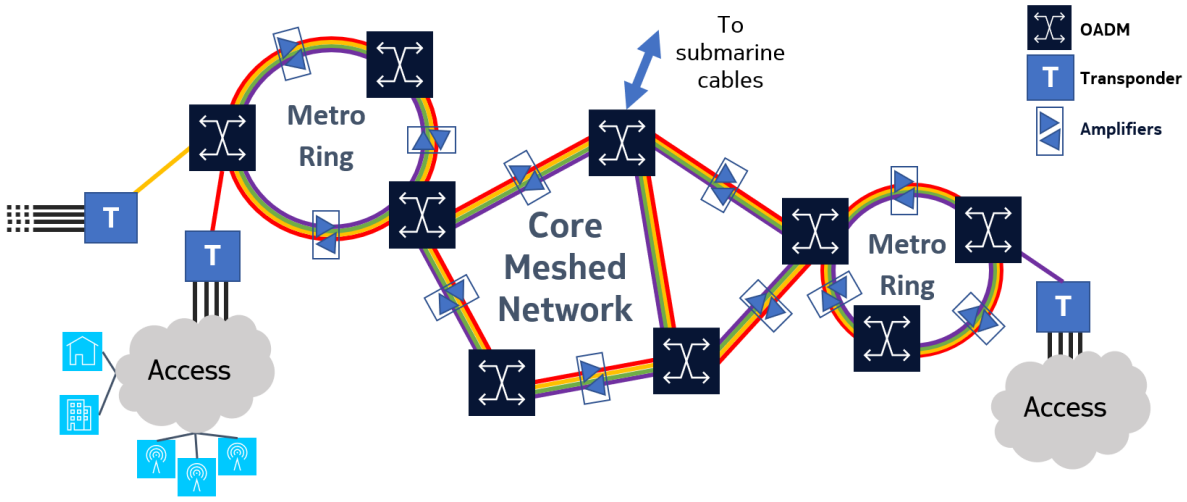


Figure 2.7 – Representation of an optical transport networkd data plane.

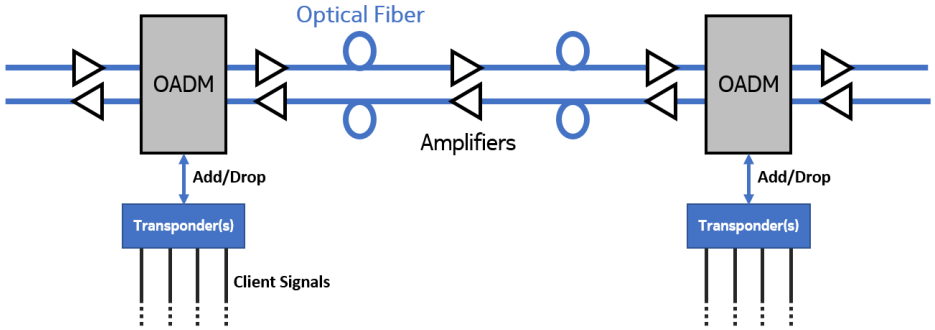


Figure 2.8 – Optical line in transport networks, with Transponders, OADMs and amplifiers

an OADM where it will be routed to its destination. The reverse operation is done when receiving a signal on a line side. At transmission and reception, Digital Signal Processing (DSP) is applied to the signal, to compensate for the degradation on the optical link, map the information on modulation formats and help the detection of the received signal. The optical transponder is going to be more thoroughly described in Section 2.2 as it is the optical equipment at the center of this thesis work.

Advancements in system design have permitted the implementation of reconfigurable transponders capable of multiple bit rates, which is ideal to adapt the communication in regards to the current condition of the network and the demand in capacity, by adapting the modulation format, changing the error correction code, etc... This system will be more thoroughly described in section 3.3.

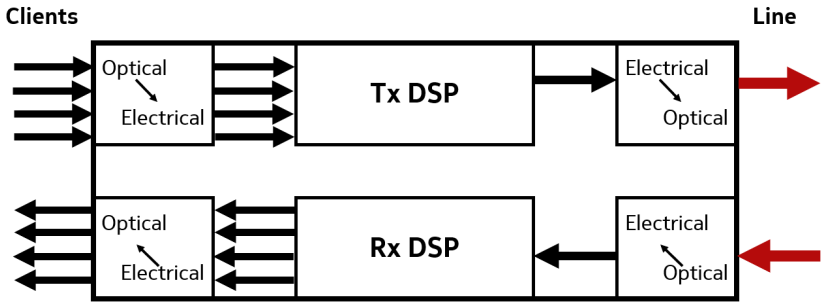


Figure 2.9 – Basic representation of an optical transponder

2.1.2.2 OADM

OADMs act as optical switches in the network. When one WDM signal is going through an OADM, the wavelengths that have to be dropped are sent to an output port to an optical transponder, wavelengths from optical transponders at the add ports are added to the traffic and transported further into the network. An implementation example of a parallel OADM is represented in Figure 2.10.

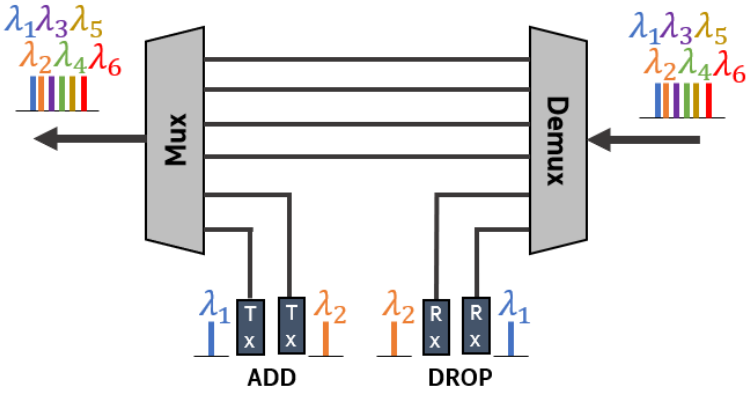


Figure 2.10 – Implementation example of a parallel OADM

In an OADM, the signal is demultiplexed and the channels are added dropped at the same time, and re-multiplexed and transmitted with the added channels. The parallel implementation is suitable when a great number of channel is to be added and dropped, but is very costly when it is not the case as you need to pass the full signal into the de/multiplexers, which filter the signal [28] and impacts the quality of communication as will be discussed in 2.1.3. Another possibility is to make band selective implementation of OADMs, as seen in Figure 2.11 to alleviate some drawbacks of the demultiplexing of the

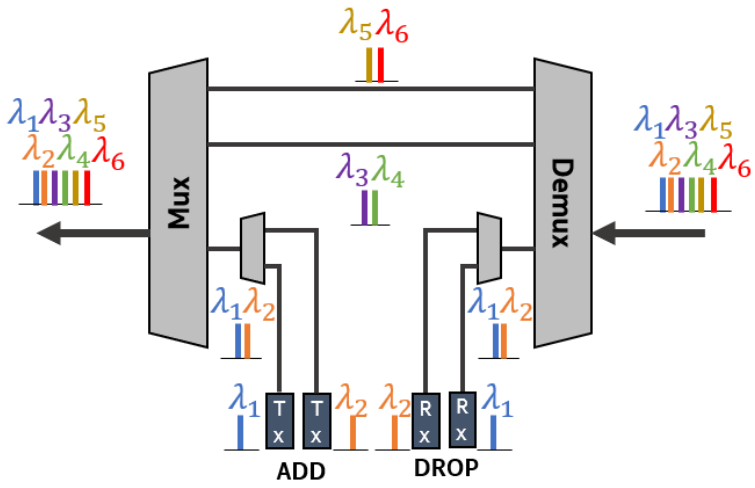


Figure 2.11 – Implementation examples of a parallel modular OADM

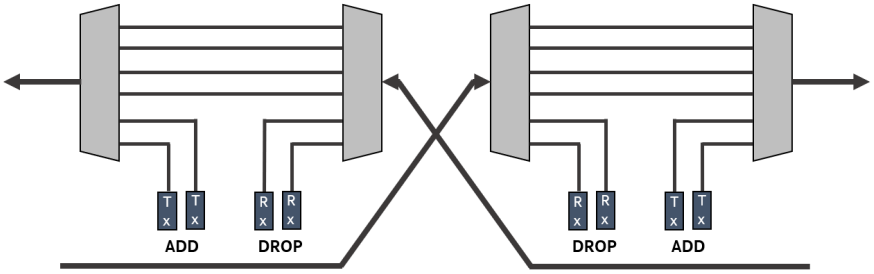


Figure 2.12 – Implementation example of a parallel two-degree OADM

full signal by demultiplexing larger parts of the bandwidth, containing multiple channels.

Another key point of OADMs is the possibility to have multiple entry and exit point for the WDM signal, called degrees. A two degree OADM having two inputs and outputs, and a three degree three etc... Two degrees OADM are central in ring network topologies, and three and more degrees OADMs are used in meshed network topologies. Figure 2.12 shows a two-degree OADM implementation.

The new development in reconfigurable optical devices also enabled the implementation of Reconfigurable Optical Add-Drop Multiplexers (ROADMs), contrary to classical (called fixed) OADM, ROADMs as their name imply are reconfigurable by the operator to add and drop signals and support the flexible grid. They are implemented using Wave Selection Switches (WSS) and Multicast Switches (MCS). A WSS has one common port and multiple multi-wavelengths ports. When the WSS is at an output of the ROADM,

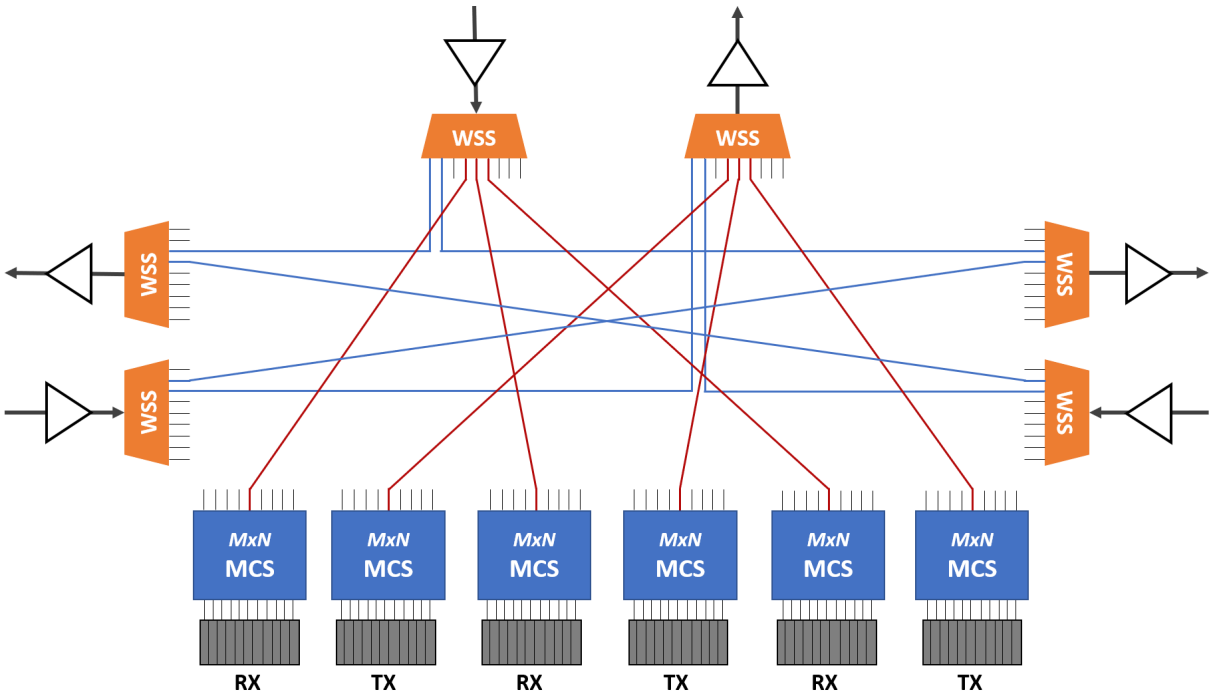


Figure 2.13 – Example architecture of a 3-degree CDC ROADM with example signals. Blue signals are signals going through the ROADM to be dropped at a later point and red signals are signals added and dropped in the device

all wavelengths at the multi-wavelength ports are sent to the common port, when it is at an input port the wavelengths at the common port are routed to a multi-wavelength port. A MCS has a multiple input and output ports on each side, and every wavelength can be routed from and to any port of the device, so they are used for the add-drop ports of ROADMs. Recent Colorless Directionless Contentionless (CDC) ROADMs allow for dynamic and efficient switching and routing of the lightpaths of the network, with add and drop ports that are not wavelength selective (colorless), with any add and drop port being able to be switched to and from any direction of the ROADM (directionless) with no wavelength blocking when two ports use the same wavelength (contentionless) [29,30]. Figure 2.13 shows an example architecture of a CDC ROADM, with its add-drop portion based on Multicast Switches (MCS), and its three degrees with WSS at the transmission and detection. This opens the possibility to program and reprogram routes and even plan backup routes in the optical transport network, in the case of for example a degradation in the link between two ROADMs [31].

2.1.2.3 Amplifiers

To increase maximum reach of optical signal and mitigate the attenuation that happens inside OADMs, amplifiers are placed in the network, increasing the optical power. Optical amplifiers used in transport networks are mainly Erbium Doped Fiber Amplifiers (EDFAs), providing good gain and relatively large bandwidth. However, using amplifiers increases undesirable effects in the transmission. The most important is the addition of Amplified Spontaneous Emission (ASE) noise in the transmission, which is an additive noise dependant on the incoming signal power in the amplifier. They are usually placed at entry and exit point of optical equipment (notably OADMs), and in the optical link between OADMs to increase the optical reach, as seen in Figure 2.14.

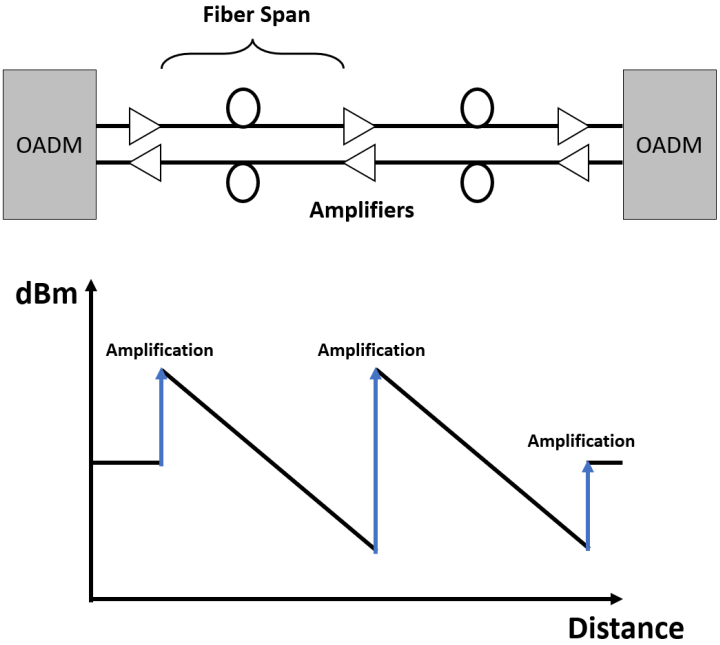


Figure 2.14 – Optical link between two OADMs, with amplifiers placed at entry and exit points of the equipment and in the middle of the link. The fiber portion between two amplifiers is called a fiber span.

2.1.3 Optical link impairment

Like any communication channel, the optical link can be subject to many phenomena that degrade the quality of the transmitted signal, which by extension makes the proper reception of the information more difficult. We will briefly describe some of the most

common effects occurring during the optical signal propagation through the fiber.

Chromatic dispersion As the WDM transmission is using multiple wavelength to transmit the information on the optical link, the WDM optical signal is susceptible to Chromatic Dispersion (CD). This comes from the fact that the light's group velocity and phase velocity depend on the wavelength. This implies that the further the signal goes, the more the chromatic dispersion will accumulate, creating a broadening of the WDM signal and possible bit detection errors, as seen in Figure 2.15. The chromatic dispersion can be measured at reception of the signal and is expressed in ps/nm·km.

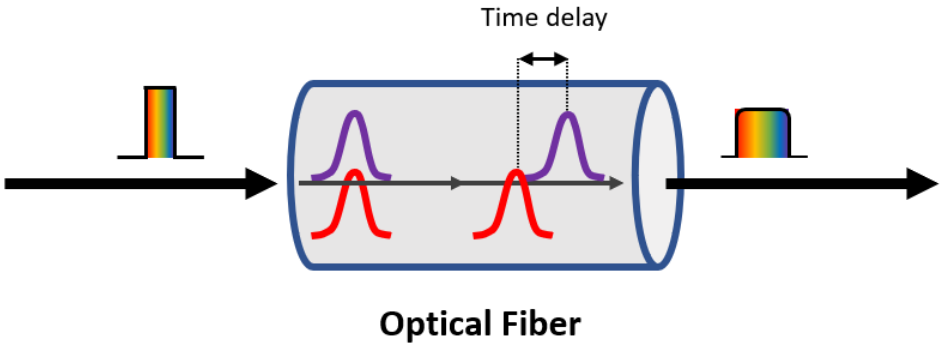


Figure 2.15 – Representation of the effects of Chromatic Dispersion.

Nonlinear effects Another important effect happening in the optical fiber during the light signal propagation is the nonlinear Kerr effect. It corresponds to the phenomenon where the refractive index of the fiber (but more generally crystals glass or gas) changes when light propagates in the material. The light creates an electric field in the fiber, and the change in refractive index is proportional to the square of the field strength, making this perturbation nonlinear and resulting in inter symbol interferences [32]. Another nonlinear effect of interest is the Raman scattering, where when two signals with different wavelengths are propagating in the fiber, the signal with the longer wavelength get amplified, and in the contrary the shorter wavelength signal loses intensity. This is due to the response of the optical medium not being instantaneous when light is propagating through it.

Filtering When propagating in the network, the light signal goes through a variety of filters, notably when traversing a WSS. As the number of optical node in the network

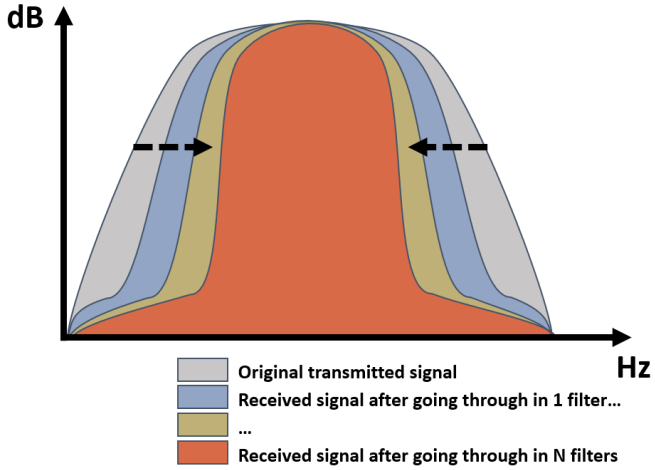


Figure 2.16 – Effects of cascaded filtering on the spectrum of a received signal.

increases, the number of filtering increases, degrading the signal. The more filters the signals pass through, the more the spectrum is distorted, especially at the lateral parts of the spectrum [33], as seen in Figure 2.16.

Environmental effects Environmental effects have a noticeable influence on the optical transmission, as they directly impact the fiber and degrade its capacities. The most common is if the fiber gets in movement, by the effect of vibration in the close environment for example, which impacts the intensity and the polarization of the signal going through the fiber. Other environmental effects such as temperature and pressure can come up and degrade the quality of transmission, but are more uncommon.

2.1.4 Optical transport network control plane

The management of optical networks has seen enormous changes in the past decades, enabling not only a more efficient management of optical equipment in the network in regards to their increase in capabilities and complexity, but also making huge steps towards autonomous operations.

In optical networks the control plane is a set of control processes that is responsible for the establishment and the maintenance of the service, supervising the equipment in the network. By establishing a control loop where the equipment (all or part of the ROADMs, the transponders and the amplifiers for example) are continuously monitored and the parameters of the network are frequently re-evaluated, the control plane is able

to automatically detect and resolve issues that may arise in the networks lifetime, such as a degradation of performance, a new lightpath that needs to be created to route the traffic more efficiently etc...

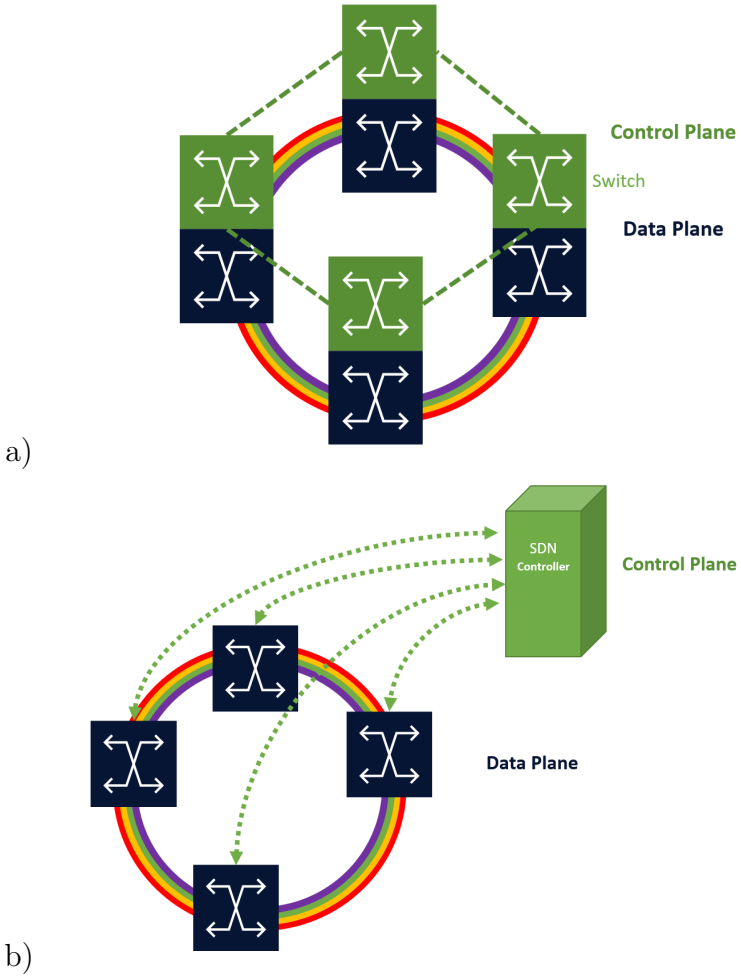


Figure 2.17 – Simple representation of the a) traditional control plane architecture and b) the SDN architecture.

The most adopted paradigm in optical network control research is currently the Software Defined Network (SDN). Its main principle is that a central software program dictates the overall network behavior [4]. This is an evolution from previous network management solutions, namely the Generalized Multi-Protocol Label Switching (GMPLS) (General Multi-Protocol Packet Switching), where the decision making and routing of the network was very distributed and located near each node of the data plane, making the scalability of the management and control of the network complex [34] (Figure 2.17 (a)). This

reduction in complexity has done a great deal in enabling flexibility in the network, making easier the introduction of new device, establishing new lightpaths and changing the configuration of an optical device [35]. Figure 2.17 (b) shows a simple representation of the SDN Architecture, with the clear separation of the data and control plane, the latter being populated only with the SDN Controller. While this representation shows the centralization of the control and management functions in a single entity, it is not the only architecture discussed in recent research. As the number of equipment to supervise and their complexity increase, the need for a dedicated Monitoring and Data Analytics system to collect and process the monitoring data in the network has become apparent [36]. This helps the SDN controller by offloading some functions and processing power, and helps implementing modern and complex data processing techniques such as Machine Learning. Coupled with open standards to interface the equipment to the network controller and the addition of an agent to abstract and expose devices to the controller [37], optical network can be implemented with equipment from multiple vendors, as from the SDN controller point of view it interacts only with virtual devices. This allows to dynamically create and modify subdivision, or slices, of the network to respond to specific demands [6] (eg. a low latency communication between a node pair of the network). Other works have questioned the over centralization of the control tasks, proposing more distributed and local approaches, while not contradicting the SDN paradigm. The implementation of agents to offload some tasks of the control plane closer to the data plane, creating a distributed or hierarchical structure shows some benefits in reducing the latency in the decision making process at the cost of higher complexity in implementation [9, 10].

Monitoring of optical layer

With optical equipment embracing a more flexible way of functioning and with the optical control plane gaining more and more functionalities, optical monitoring has been more and more prevalent in enabling autonomous operations in optical networks. With all the effects occurring in the optical line as described in 2.1.3, it is crucial to keep surveilling the optical equipment and detect when the impairments are impacting too much the communication. This can be measured by periodically collecting several monitored values in the optical equipment and acting accordingly. For example, retrieving the reception power (or Rx Power) or the Bit Error Rate (BER), the number of bits altered by noise, is a good way to determine when the communication is degrading when the former is decreasing and the latter increasing. A lot of parameters are available in modern optical

equipment [38] and it is at the operator's discretion to decide on a set of parameters that the control plane should closely monitor and at which rate, as collecting all the monitored values at the maximum achievable rate possible takes a lot of processing power and memory.

Traditionally, the process of collecting the monitoring data from the equipment to the control plane used very basic protocols, suitable for the very low pace at which monitoring was performed in optical devices (usually every 15 minutes) and the processing power in the optical control plane. But with the increase in computational performance in both the control plane and the optical equipment, new possibilities have emerged for optical monitoring. First of all a new generation of protocols have enabled the use of telemetry services [5]. Network telemetry is the service of real-time streaming the optical monitoring values from the optical equipment to the control plane. This service can be activated on demand, over a certain amount of time, and can be at a very fast rates (even below the second if the optical equipment is able to update its values at this rate). This service can be used in addition to more traditional (and slower) monitoring services: telemetry focusing the attention of the control plane on a subset of equipment, helping detecting failures more rapidly and reliably, while the slower service collects data for long term analysis of the network or raising alarms for network devices that are under less reliability constraints. The other benefit of the increase of processing power is that the control plane is able to use more complex algorithms to process the monitoring data, enabling even machine learning algorithms [39, 40] to perform performance evaluation or to re-route the traffic in more efficient ways or after a degradation. Current researches on machine learning on the control plane are aiming at using more complex models to better represent the optical network and its complexity to take better decisions [41] or using the more complex and more powerful Artificial Neural Network [42] for example. Other works are aiming at implementing machine learning algorithms directly into the optical equipment [43] to gain efficiency.

2.2 Optical Transponders

2.2.1 Architecture

The optical transponder is an optical/electrical/optical device that serves as an entry and exit point for the client traffic in the optical transport network. When the optical client

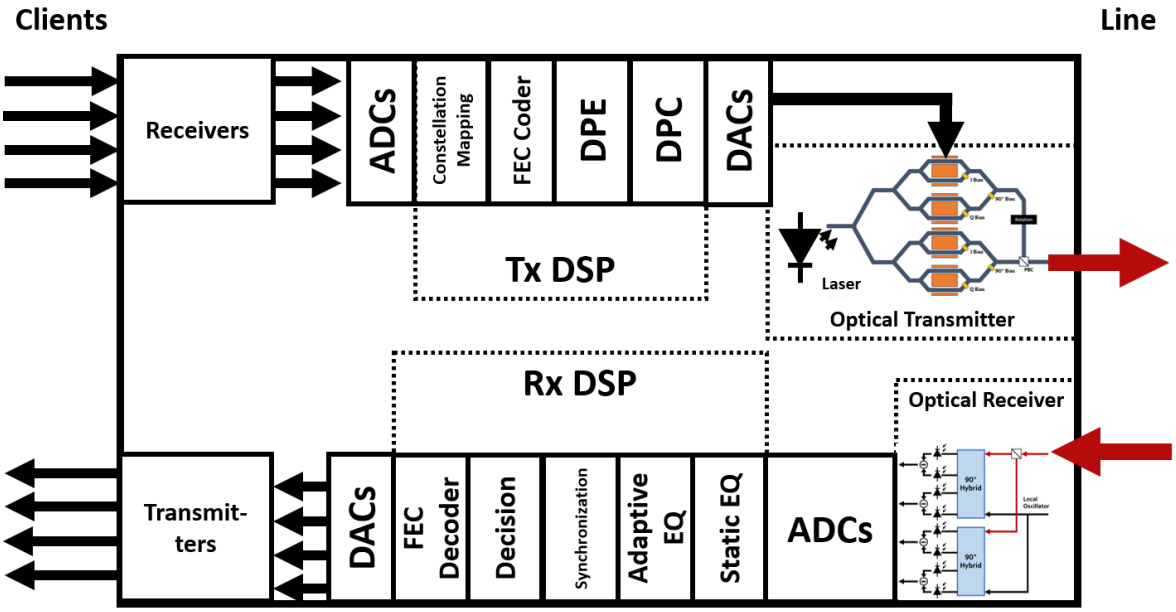


Figure 2.18 – Representation of an optical transponder

signal is received on a client port, it is converted into an electrical signal, and processed and amplified, then optically converted and aggregated with other client signals into a single WDM-compatible wavelength to a line port that is connected to the WDM network. The reverse operation is done when the signal is received on the line port, the client signals are de-multiplexed, processed to compensate for the distortion in the optical channel and to retrieve the data symbols, and then routed to the proper client ports. Figure 2.18 shows a basic presentation of an optical transponder and in this following section we will detail its components and subsections.

2.2.2 Digital and Analog conversion

The DAC or Digital-to-Analog Converter is the electrical component that transforms the digital data into electrical voltages. In optical transponders it converts the signals of the DSP that will be sent to the modulators of the transmitter, as it will be explained in 2.2.4. On the other side, the Analog-to-Digital Converter (ADC) transforms a voltage into a digital signal, and in the optical transponder transforms the received signal into data that will be processed in the receiver DSP on the line side.
Both these components are mainly characterized in terms of resolution and sampling speed. The resolution corresponds to the number of bits available to perform the desired

operation and describe the incoming or outgoing waveform, and the sampling speed is the periodicity at which the sampling is performed. Figure 2.19 shows that augmenting the sampling rate and the resolution helps better describe the signal in bits (for the ADC) and in output waveform (for the DAC). However, it is not possible to maximize on both these characteristics in optical systems and a trade-off has to be made in the design as in optical systems a converter with a high sampling rate has a low resolution and vice versa. Another characteristic is the Effective Number of Bits (ENOB) which is the actual resolution of the equipment when taking in consideration the quantization and electrical noise of the equipment, and usually decreases with the frequency [44]. Another problem is the bandwidth of the components, that requires special processing to mitigate, as will be discussed in 2.2.3.

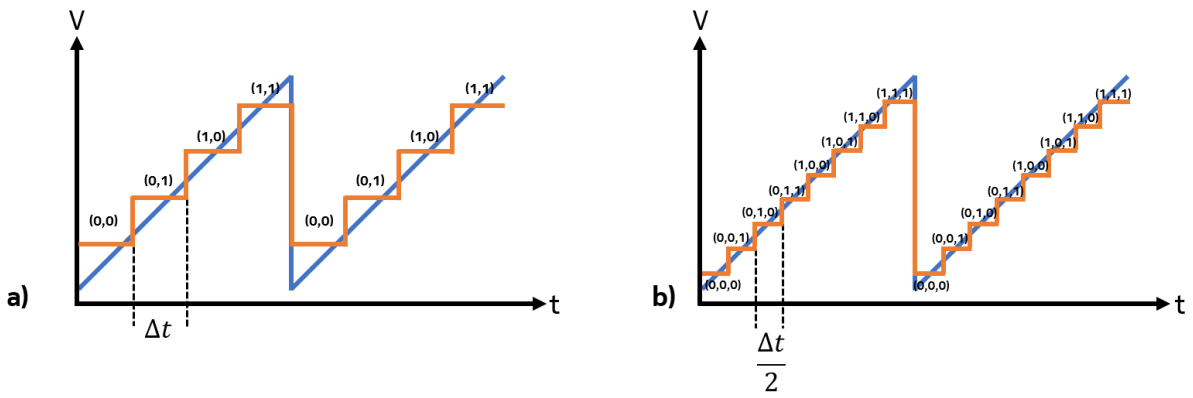


Figure 2.19 – Basic representation of the operation of a DAC and an ADC converting a ramp signal (the blue line), with (a) 2-bit resolution and Δt sampling rate, (b) 3-bit resolution and double sampling rate. The orange line represents the output waveform from the DAC and the ADC conversion is represented by the bit vectors.

2.2.3 Digital signal processing

The development of high-speed DAC and ADC coupled with the improvement in DSP techniques has been key in enabling coherent optical communication. This has enabled the reception of complex modulation formats using light polarization (that will be discussed in 2.2.4) and the compensation of optical line impairments. DSP algorithms and applications are traditionally deployed in optical transponders using ASICs (Application-Specific Integrated Circuit) specifically designed for this purpose.

2.2.3.1 Transmitter side DSP

Modulation formats If the line side of the optical transponders was first using very simple modulation formats such as OOK (On-Off Keying), the ones primarily used nowadays by the device have become more complex, helping achieve greater bitrates. Phase modulation have become more common in transponders (such as the the Quadrature Phase Shift Keying, or QPSK), but with the advent of the coherent detection in optical systems, modulation using the polarization of the light have started to be used with success with for example Polarization Multiplexed-QPSK (PM-QPSK) [45] or PM-16QAM (Quadrature Amplitude Modulation) [46], using the fact that the light can be polarized vertically or horizontally, see Figure 2.20. However, increasing the bitrate thanks to the use of a more complex modulation format makes the signal more susceptible to interferences, which complicates the detection process, and reduces the maximum achievable reach of the optical signal.

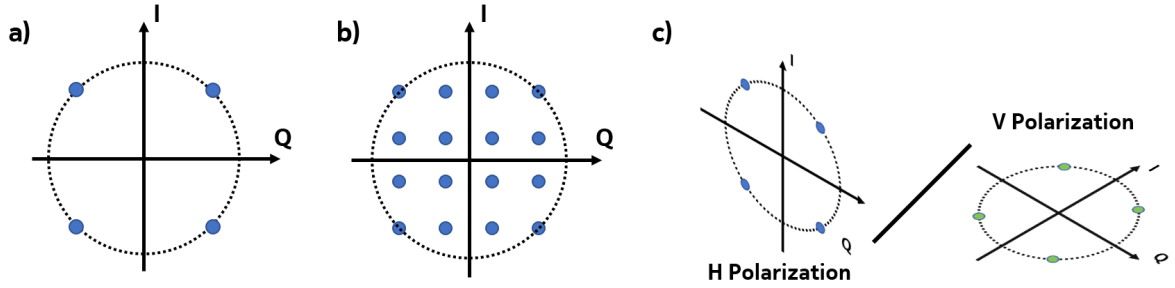


Figure 2.20 – Representation of (a) the QPSK, (b) 16-QAM and (c) PM-QPSK modulation formats

Forward Error Correction Forward Error Correction (FEC) is a technique where this transmitted data is pre-coded at transmission, and this coding helps at detection to see if an error is present on the detected signal, and potentially correct this error. This functions by adding redundancy to the transmitted data. This can greatly improve the quality of communication between a transmitter-receiver pair, at the cost of reduced spectral efficiency due to the redundancy added in the data stream. Multiple error coding schemes exist for optical communications, as reviewed in [47]. This requires a FEC Coder and a Decoder in the transmitter and the receiver respectively. FEC Codes commonly used in optical networks are Staircase, Reed-Solomon or LDPC (Low Density Parity Check) codes [47–50]

Pre-Emphasis In DSP, Digital Pre-Emphasis (DPE) is used to reduce the distortions caused by the low pass response and the limited bandwidth of the DAC in the transponder, as described in 2.2.2. For DPE, the signal is filtered, resulting in lower peak-to-average power ratio, as seen in Figure 2.21. If this technique increases the reception quality especially at higher baud rates [51], it also creates a trade-off in output power and DAC bandwidth [52].

Pre-Distorsion Compensation On the transmitter side, Digital Pre-Compensation (DPC) techniques can be used, as it allows to reduce the degradation introduced by optical components such as the DAC and the transmitter, and increases the spectral efficiency of the transmission [53]. They are especially useful when using high order modulation formats and higher baud-rates, that cause additional skew in the transmitter [54] for example.

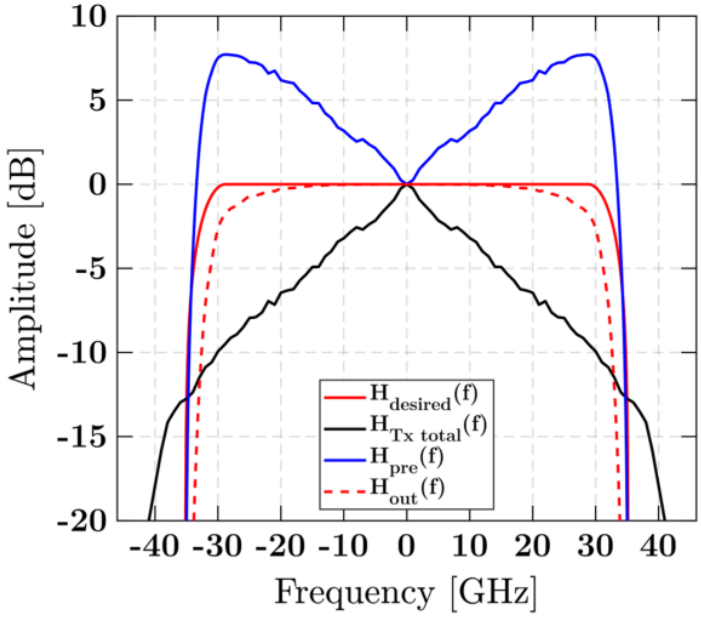


Figure 2.21 – Representation of the digital pre-emphasis processing. $H_{desired}$ represents the ideal signal spectrum, $H_{TxTotal}$ the transfer function of the DAC, H_{pre} the transfer function of the DPE filter and H_{out} the resulting spectrum after the DPE

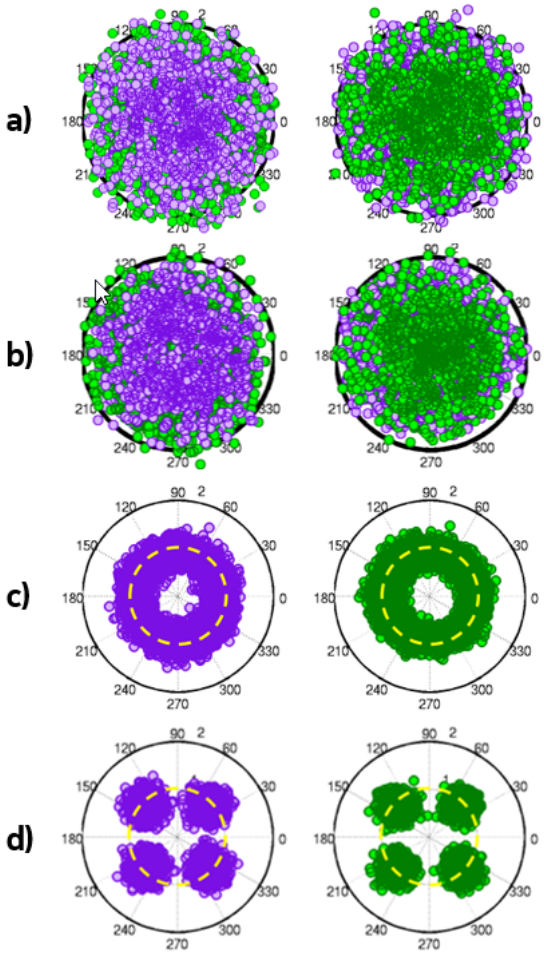


Figure 2.22 – Representation of DSP steps on the reception of a PM-QPSK signal, with on the left the Horizontal polarization and on the right the Vertical one. (a) is the signal coming from the ADC, (b) is the signal after CD compensation, (c) is the signal after CMA, and (d) is after carrier and phase recovery

2.2.3.2 Receiver side DSP

On the receiver side, DSP algorithms are used to help the system properly decode the detected signal coming from the ADC as explained previously, and are separated into two main categories: equalization and synchronization. The main goal of the equalization algorithms is to compensate for the phenomenon that take place in the optical channel between the transmitter and the receiver that degrades the quality of the received signal. The synchronization aims to correct the timing and frequency differences between the transmitter and the receiver so that the signal can be finally decoded.

Equalization The main equalization techniques in coherent receivers are the static and the adaptive equalization. The static equalization aims at compensating for example the losses due to the Chromatic Dispersion (CD) as described in 2.1.4. The adaptive equalization goal is to track and compensate time-varying losses due to polarization effects. For modulations using a single amplitude (PM-QPSK for example, as described in 2.2.4), a Constant Modulus Algorithm (CMA) is used to ease the decision on the symbols and perform polarization demultiplexing [55]. Other algorithms exist for more complex modulation formats, such as Radius Directed Equalization [56]. The static equalization requires a large static filter, whereas the adaptive equalization is done via a set of adaptive FIR filters. Both could be done in a single filter but separating the two eases the design of the system [57].

Synchronization Just equalizing the signal is not enough to make decision on the received symbols. As the transmitter and the receivers are not synchronous, it is important to recover the frequency and the phase of the received signal, using Carrier Phase Estimation (CPE) and Carrier Frequency Estimation (CFE) algorithms. These algorithms can function in a "blind" manner [56], or with an a-priori knowledge of the incoming signal ("pilot-aided") [58]. A representation of the results of DSP on the reception of a PM-QPSK signal is represented in Figure 2.22.

2.2.4 Transmitter

To modulate the optical signal, equipment use mainly Mach-Zender Modulators (MZM), which basic structure is represented in Figure 2.23. The MZM splits the light from a laser into two paths, and shifts the signal phase with two electrodes, one in each arm. The electrode is driven by the outputs of the DACs of the transponder. An additional DC Bias Voltage is used in one arm to introduce an additional phase-shift, allowing I/Q modulation. To realize I/Q modulations MZM can be stacked (this structure is called Dual-Parrallel-MZM, or DP-MZM) and for Polarization Multiplexed modulations, another stacked DP-MZM (for the modulation of the second polarization) and both a device rotating the polarization of the signal and a Polarization Beam Combiner (PBC) combining the two polarized signals are necessary [59], see Figure 2.24.

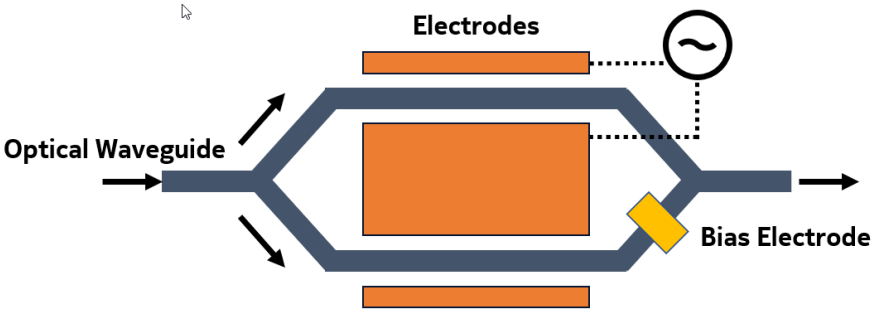


Figure 2.23 – Basic structure of a Mach-Zender Modulator

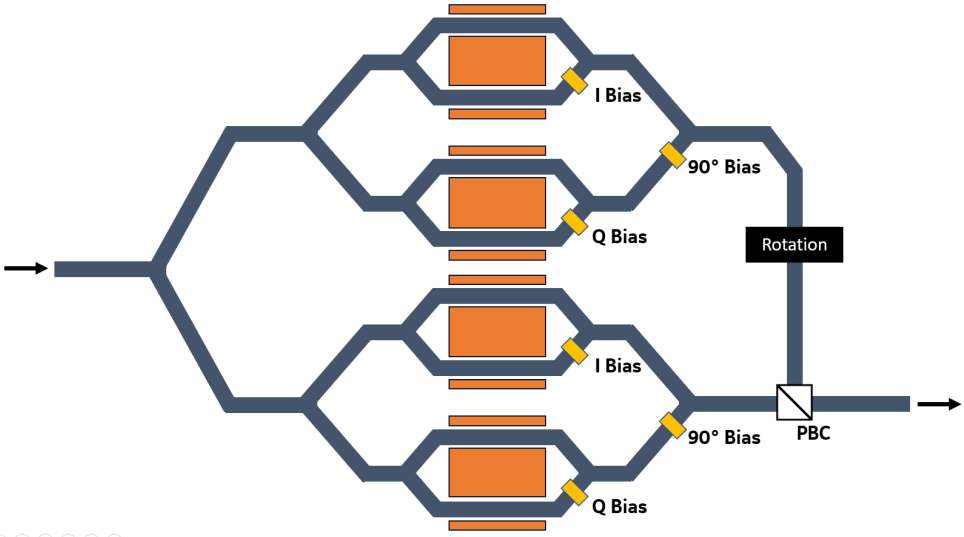


Figure 2.24 – Coherent I/Q modulator using multiple MZMs

2.2.5 Receiver

To recover the signal on the line side, coherent optical transponders use 90° hybrids. They are couplers with two inputs and four outputs with the two inputs being the received signal and an Optical Local Oscillator (OLO) signal. The outputs of the 90° hybrid are the received signal mixed with the four quadrature states of the OLO. These outputs are sent to photodetectors that transform the optical signal into an electrical one, and are subtracted in pairs. This results in two outputs: the real and imaginary parts of the signal, that will go into the ADCs and will be processed by the DSP. The mathematics behind the 90° hybrid are more thoroughly described in [60]. Figure 2.25 represents a coherent optical receiver where two 90° hybrids are used, one for each polarization, with

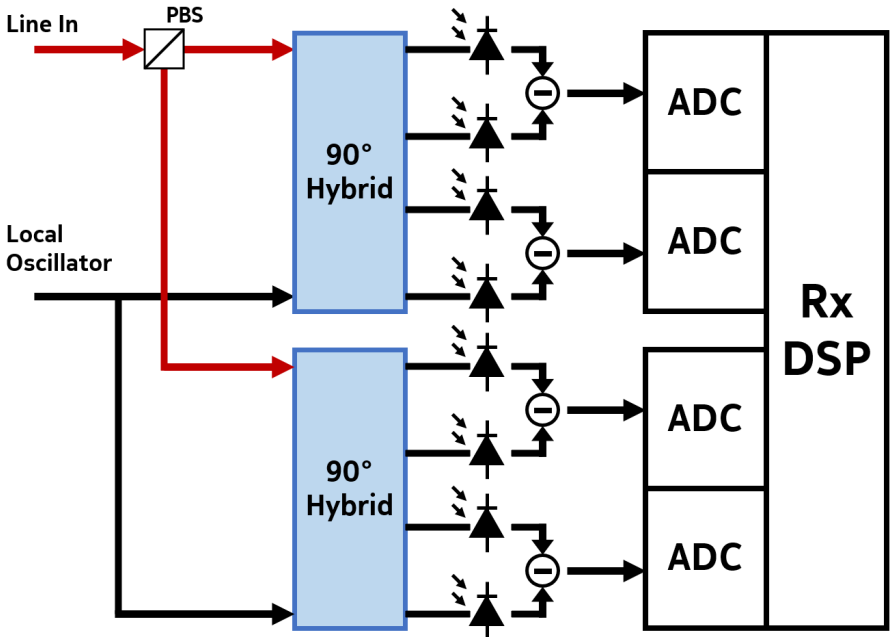


Figure 2.25 – Representation of a coherent receiver

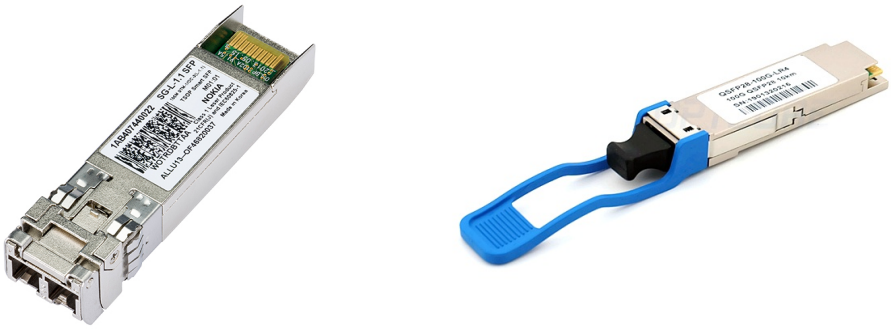
the incoming signal being split in polarization by a Polarization Beam Splitter (PBS).

2.2.6 Physical optical interface

To physically connect the optical fiber to and from the optical transponder ports, line or client side, optical pluggable interfaces are often used in commercial equipment. These interfaces are called transceivers and take care of the emission, reception and optical to electrical conversion and vice versa [1]. The current most common pluggable transceiver for client signals is the Small Form-factor Pluggable (SFP) and its derivatives that have been developed to enable greater data-rates. The first SFP module was introduced in 2001 and was fit to transmit up to 1Gbps [61]. Revisions have come, increasing the bitrate on a single channel (SFP+ up to 10Gbps, 2009, SFP28 up to 25Gbps, 2014) and quadrupling the number of channels and the maximum bitrate attainable (QSFP up to 4Gbps, 2006, QSFP+ up to 40Gbps, 2012, QFSP28 up to 100Gbps, 2014), and since 2016 even offering 8 channels and bitrates up to 400Gbps with the QSFP28-DD (Double Density). SFP modules come with their set wavelengths and their maximum reach, which separates some modules in two categories: Short Reach (SR, maximum reach usually around hundreds of

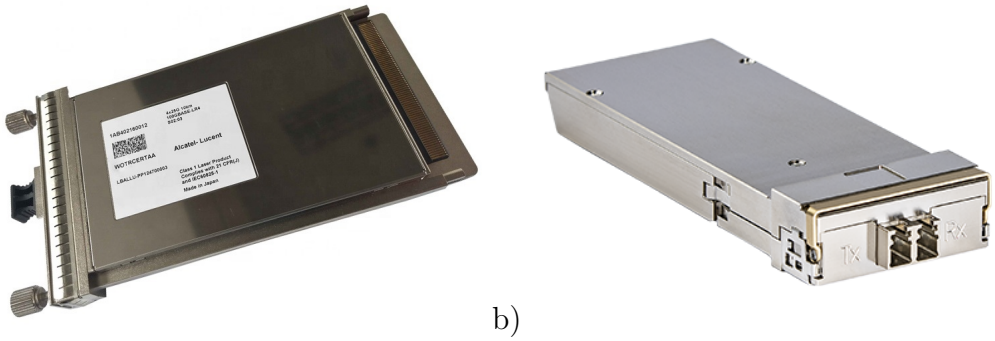
1. Though pluggable modules are called transceivers, the term is used more generally to designate any system that does the same things

meters) and Long Reach (LR, maximum reach usually around tens of kilometers).



a) b)

Figure 2.26 – Photograph of (a) a SFP Module and (b) a QSFP module



a) b)

Figure 2.27 – Photograph of (a) a CFP module, (b) a CFP2 module

For the line ports, the current more common solution of a pluggable transceiver module is the C form-factor (CFP) [62]. These modules were originally designed circa 2009 to handle 100Gbps communication (hence C for *Centum*, hundred), with revisions bringing even more capacity and/or improving power performance and global footprint (CFP2 up to 200Gbps and smaller form-factor, 2013, CFP4 up to 100Gbps but with an even smaller form-factor and power consumption, 2014, CFP8 up to 400Gbps, 2017). One of the main differences with the SFP modules is that the CFP are built from the ground up to use multiple channel (CFP could make 100Gbps by using 10 channels at 10G or 4 channels at 25G for example), which explains their utility for line port interfacing. As DSP circuits continue to get smaller and smaller, CFP modules can come in two different types, DCO (Digital Coherent Optics) which includes the DSP for coherent optical coherent communication and ACO (Analog Cohernt Optics) which does not, and a transponder

needs to be built accordingly to handle only one of these types or both (on separate ports), which also limits interoperability between CFP-DCO modules of different manufactures with their own DSP systems.

2.3 Conclusion

In this chapter we provided an overview of the context of this thesis work, that will be helpful to understand the following chapter around the questions of reconfigurability, notably in the context of the optical transponder. We discussed the optical transport network, we started by presenting its data plane and its control plane, with the current evolution towards elastic optical networks. We then presented and described the optical transponder, the device that is the primary focus of this thesis work, with its main component and architecture.

Chapter 3

Flexibility and reconfigurable transponders

The concept of flexibility is not exclusive to communication networks and covers a very general assumption in engineering: flexibility is the capacity of oneself to meet a varied set of requirements, ideally depending on the current environment. These requirements can be power, speed, cost, reliability, security, etc... This concept can be applied to a variety of domains, such as power system design [63], antenna design [64] and more. In all cases, it is the search for close-to ideal performance even in unpredictable situations, achieved by designing the subject in a particular manner, usually by allowing structural changes during the subject lifetime.
In optical communication networks, the concept of flexibility can be studied on two different levels, with their own different requirements and implications: system-wise and network-wise. We will start by studying what these two approaches are, and for the system approach we will present the Field Programmable Gate Array (FPGA) circuit that will be central for the works of this thesis and its relevance for flexible system design, and for the network one the evolutions and tools now available in flexible optical networks. We will then discuss the evolution from fixed transponders to modern Sliceable Bandwidth Variable Transponder (S-BVT) and the challenges that face optical transponder design in order to be fully compatible with and facilitate the implementation of flexible optical networks.

3.1 System flexibility

In [65], the authors characterized a flexible system as "a system designed to maintain a high level of performance through real-time changes in configuration when operating conditions or requirements change in a predictable or unpredictable way". In more details they talk about two main properties that a flexible system possesses: adaptability and

robustness. Adaptability is the ability to change one or multiple parameters, with a flexible system being able to modify at least one. There is also a distinction between systems able to change their parameters while functioning in real-time (active) or not (passive). Robustness on the other hand is the set of parameters that are set and not allowed to be changed. This is due to the high cost of flexible hardware, that does not always allow for fully programmable devices, and these set parameters lowers the number of uncertainties in the system.

In summary to develop a flexible architecture for a communication system, emphasis should be put on enabling reconfigurations of the device. Ideally, a fully flexible device should be able to change its parameters on the fly, without interruption, and has a great array of possible configurations (or configuration sets), so that it is able to adapt itself to any situation. This can be achieved, fully or in part, by smartly designing the equipment so that it is able to perform a multitude of different operations and/or using reconfigurable hardware, the latter being investigated thoroughly by researchers.

Reconfigurable hardware is a specific type of hardware that can be programmed and reprogrammed after production, and is often put against Application Specific Integrated Circuits (ASICs). Whereas ASICs (or other application specific hardwares such as Domain Specific Accelerators) are very specialized and performant, but with lower lifecycle and higher non-recurring engineering costs, reconfigurable hardware offer more flexibility and in result more efficiency, at a greater cost per unit [66, 67]. Reconfigurable hardware also have their advantages against General Purpose Processors or Graphic Processing Units with their higher energy efficiency, at the cost of a higher specialization. Reconfigurable hardware are in conclusion at a middle point, offering flexibility, performance and efficiency [68], and are a prime candidate for expanding the flexibility of systems [69].

Multiple type of reconfigurable hardware exists, for multiple applications. One such example of reconfigurable hardware that will be used in this thesis work is the Field Programmable Gate Array, or FPGA, and will be more thoroughly discussed.

3.1.1 The FPGA hardware

The FPGA is an integrated circuit that is programmable and reprogrammable after manufacturing, and is one of the most common type of reconfigurable hardware used in the industry. An FPGA is composed of multiple interconnected and configurable logic blocks, which allow for a great variety of possible designs and implementations and efficient performance on precise and/or repetitive tasks, which makes this hardware suitable for

flexible hardware design prototyping and even for production.

At their basis, FPGAs are an evolution of the Programmable Array Logic (PAL) of the 70s and 80s. A PAL is a logic device that is composed of inputs, an AND logic gate array that is programmable, and can combine any input and their inverse with any other input, and OR logic gates at the output. By programming the logic gate, a variety of operations can be performed with this simple structure, especially when the output is fed back to the input. On the other hand, the FPGA is not using an AND array for combinational operations but is using Configurable Logic Blocks (CLBs). The CLBs are mainly composed of Look Up Tables (LUTs), registers used as latches or as Flip-Flops (FFs), a carry chain with logic and multiplexers. An example of a Logic Block is represented in Figure 3.1, using four inputs and two 3-bit LUT (i.e. a 3-bit input LUT).

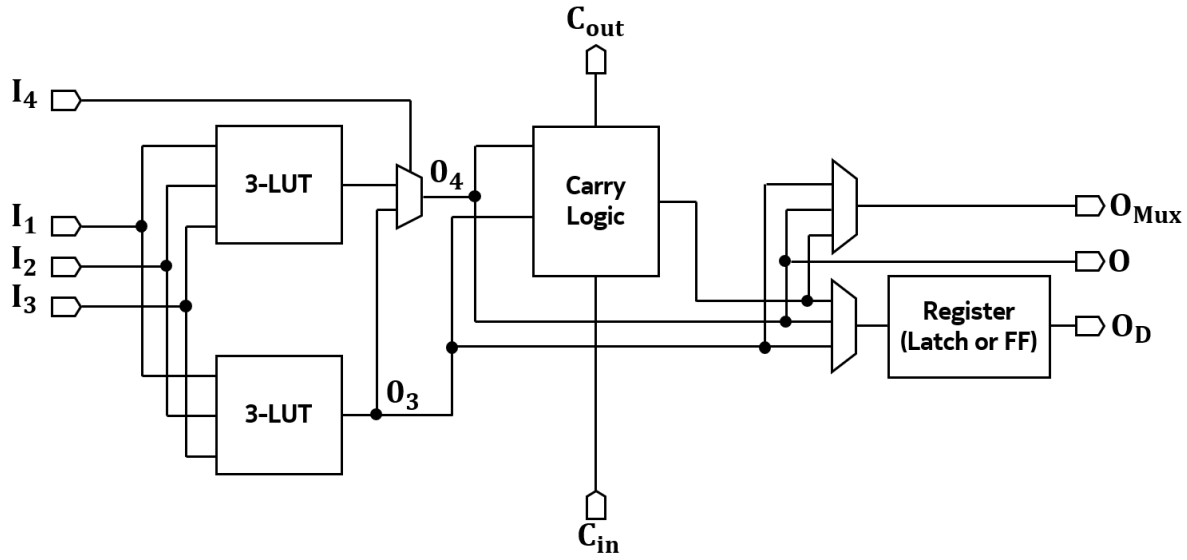


Figure 3.1 – Architecture example of a CLB with four inputs.

At the leftmost side of Figure 3.1, you can see the two LUTs. A LUT is a logical structure that acts like an array or a truth table: depending on the value on the inputs of the LUT, an output will be fetched (in CLBs this output will be 1-bit). This structure basically performs a logical operation between inputs by acting as a memory, and can be programmed to fit the designer's needs. Having two LUTs with the same number of inputs side by side is a common practice in modern CLB design. If two 3-bit LUTs are used it allows to perform either a 4-bit operation (the O_4 signal in the figure), thanks to a fourth input driving a multiplexer at the output of the LUTs, or perform two independent 3-bit

operations (using both O_4 and O_3), giving flexibility to the FPGA architecture. A carry is going through the CLBs, and is added or subtracted if needed to the outputs of the LUTs. The carry goes to and comes from other CLBs in the device, and helps realizing more complex operations without having to mobilize an additional CLB. The outputs are sent either directly outside of the CLB, or to multiplexers and registers, that can be used as latches or FFs. The presence of registers in the CLBs allows the pipelining of the data, which is an essential part of FPGA application design. The structure presented is a simpler version of the slice structure present in Xilinx 7 Series FPGAs, where each slice contains 8 5-LUTs (grouped into 4 6-LUTs using the same principle as in the Figure 3.1), that can be multiplexed with each other to make 2 7-LUTs or one 8-LUT, and with one CLB containing two slices (so 16 5-LUTs in total) [70].

CLBs in the FPGA are placed in an interconnection matrix, where every one of them can route their input or outputs to other CLBs (with the help of switches) or into Inputs/Outputs connected to outside the FPGA, as per Figure 3.2.

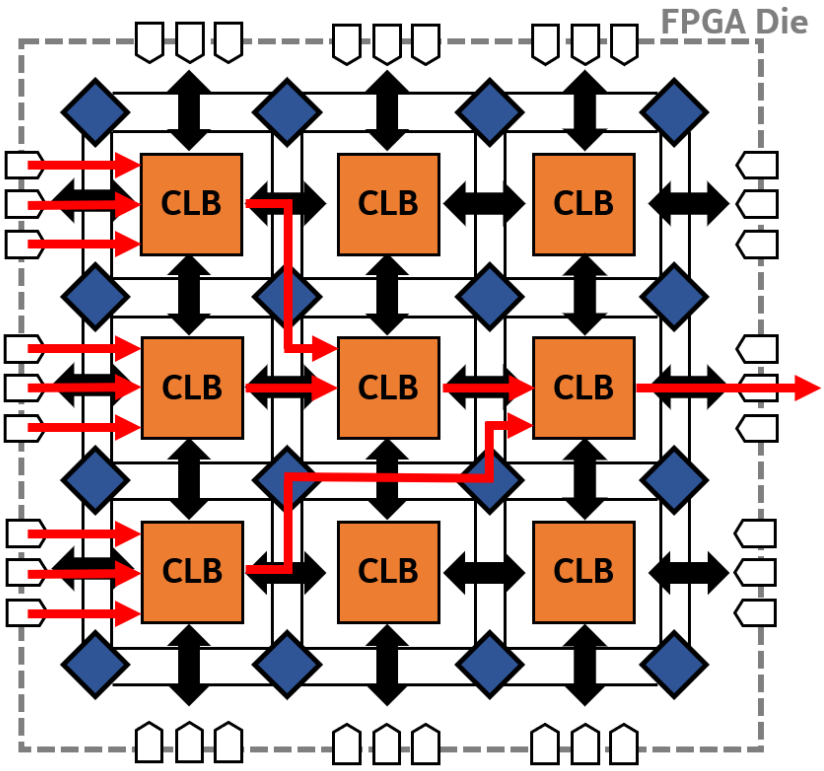


Figure 3.2 – Representation of the routing inside an FPGA. The blue diamonds represent the switches, the red arrows show a dataflow example, coming in the FPGA, processed by and routed to the CLBs and going out the die.

Modern FPGAs are more complex than a collection of logic blocks interconnected with each other. To ease the design of applications, to gain in efficiency, to save area and/or allow more complex functions, many other components and hard-coded logic exist in the FPGA die. The most common ones are the Digital Signal Processing (DSP), the Block Random Access Memory (RAM)/First In First Out (FIFO), high speed input/outputs transceivers and the clock routing and generation elements but many more exist, some allowing very specific functions. For example: an Ethernet Medium Access Controller (MAC) logic core to encode and decode Ethernet data streams, a Double-Data-Rate Random Access Memory (DDR RAM) controller to interface RAM modules to the FPGA, ... The DSP blocks are logic blocks that perform advanced operations on the signal. Their main purpose is to perform multiplication of two bit arrays, but they can also be used for other operations such as floating point operation. They are pipelined and extremely efficient, making them suitable for FIR filters implementations for example. The FIFO and Block RAM are memories that can be used in FPGA designs and have the advantage to be more area efficient than CLBs programmed to perform the same function and some implementations allow error correction. The high speed inputs/outputs transceivers provide high speed connectivity to and from the FPGA, going up to 58 Gbps in the latest commercial releases. Clock generation and routing elements are central in FPGA design. As the logic can be registered across the whole FPGA, proper clock generation and management is necessary for the logic to perform the desired operations at the desired timing. Phase Locked Loops (PLLs) are present in the FPGA fabric, so that from an external clock a or multiple internal clocks are generated at any desired frequency. Generating clocks internally has advantages, as external clocks can be jittery and generated clocks can be generated with specific phases, which can be helpful for FPGA designers. Clock routing areas and resources also allow for very efficient clocking distribution across the whole die. In Figure 3.3, we represented a modern FPGA structure where resources are placed in rows which allows for efficient routing for FPGA designs.

All these resources are available for the design of FPGA applications, which has some specifications. Where processors applications are developed using mainly procedural languages (single-threaded, or with limited multi-threading capabilities), FPGA applications are commonly developed using Hardware Description Languages (HDL). These languages as their name implies describe the behavior of digital logic circuits, which implies that it focuses on bit-wise operations, takes into account the propagation of the signals (the main variables) in the circuit and more crucially has an emphasis on timed operations.

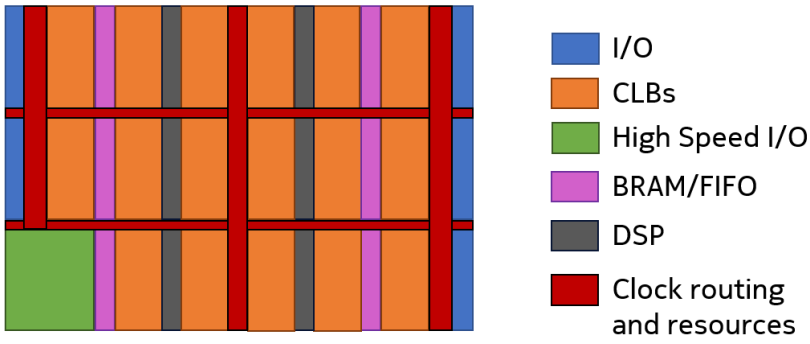


Figure 3.3 – Example layout of an FPGA.

The two most common HDLs are Verilog and VHDL (VHSIC [1] HDL). The FPGA application designer does not need to program its application down to the LUT level, a broad description of the system behavior and the needed operation is enough as compilers will translate (synthesize) the HDL program into a netlist, targeting a specific FPGA. [2] The synthesis operation allows to check if the written application can be implemented onto the targeted device, preview the resource occupation of the design in the FPGA (how many LUTs, BRAM, I/Os etc...) and see if the design can be performed correctly timing-wise or if adaptations are necessary. When the design is fit for the target FPGA, an full implementation is computed, placing and routing all the logic elements, and a bitstream is created which will be used to program the hardware. A bitstream stores the entirety of the design, from the values to be implemented in LUTs to the data that needs to be stored in RAM at initialization and the routing necessary for the application. It is also possible to create partial bitstreams, that only reconfigure a part of the FPGA logic, which allows active reconfiguration of the device, with reconfiguration times between the ms and the s. With this, FPGAs are suitable for flexible design prototyping, by creating applications capable of adapting themselves with clever designs to be implemented on ASICs, or even used in systems and reprogrammed during the device lifetime. They offer great performance with pipelining and are the principal hardware used in flexible system design.

1. Very High-Speed Integrated Circuit
2. Though HDLs are most traditional way to program applications for FPGAs, modern tools allow for higher level programming, using languages such as C/C++ or Matlab, which will be translated into HDL before synthesis by compilers.

3.2 Network flexibility

A flexible network is a network able to dynamically adapt its resources in order to match the requirements of each connection [71]. It is different than the system flexibility as it is more a question of planning and orchestration, using the flexible equipment in the pool of available resources, taking into consideration the addition of new hardware or the degradation of existing equipment.

The addition of flexibility in routing and flexible hardware in the resource pool changes the network planning process. Where network planning was aiming to maximize the performance of the network while reducing overall costs, the addition of flexible hardware changed the dynamic. Now the network planning has to take into consideration the maximization of the degrees of liberty of the reconfigurable hardware, so that it is capable of updating itself in case of an unforeseen event. In [72], authors state that planning the optical network targeting minimum BER rather than maximum overall performance allows for better flexibility, ensures that the systems operate at optimum performance and proves more resilient to transmission effects, which in short means that the long terms benefits of flexibility is more desireable than searching for immediate performance.

As discussed in 2.1.4, the control plane architecture adopted for the new flexibility paradigm is the Software Defined Network, where a centralized entity is managing the resources of the network. This structure is more adapted for flexibility against more distributed approaches because the complexity of path computation and the establishment of new optimal parameters for flexible hardware is helped by the centrality of the controller which oversees the whole network, or at least part of the network. In the example of optical networks and citing [73], Generalized Multi-Protocol Label Switching (GMPLS) control plane architecture, which was the distributed control plane before SDN, started implementing a Path Computation Element that computed network routes with constraints. This Path Computation Element was centralized, and progressively gained capacities in deciding ideal connection parameters and establishing the paths itself, which in term made it closer and closer to a SDN controller, underlying the need for centralized control for this new paradigm. Furthermore, with centralized control comes new possibilities for interacting with the flexible hardware, one of the most important in recent research is the virtualization.

3.2.1 SDN and Hardware Virtualization

Influenced by cloud computing, where computer system resources are available on-demand, virtualization is the ability to generate virtual networks and virtualized networks functionalities which are decoupled from the hardware. It allows to create sub-divisions in the network, or virtual networks, that can run with different requirements as the base network (low latency, high security, higher performance, etc...) by interacting with the hardware in specific ways and running specialized applications on the equipment of the network.

To implement virtualization in a network, it is important to ease the communication with the central controller and the hardware in the network. Several tools exist for controller-hardware communication and interaction, where the principle is that the control plane should be able to retrieve and modify easily the parameters of any equipment in the network. This is achieved by abstracting the devices under the controller supervision using data modeling, which is the process of representing a physical or non-physical object into a list of comprehensible parameters with how they relate to each other. Data modeling has the advantage of exposing clearly the hardware to the controller and to allow easy manipulation of the parameters. Ideally in an elastic optical network, all equipment of the same type (transponder, ROADM, amplifier) from any vendor follow the same data-model so that the control plane can operate them as generically as possible. The most common data modeling language is Yet Another Next Generation (YANG) [74]. In YANG, the type of parameters and their possible values are listed such as the modulation format, the type of FEC supported by the equipment, the number of ports of any type etc... Detailed examples are given in [75]. YANG data models are also used in optical networks to present the monitoring data from the equipment to the network controller [5]. YANG data models can be written in any language that can be supported by the tools and communication protocols used by the network controller (popular examples are XML and JSON, among others...). To propose common sets of YANG data-models and interfaces for proper optical network management, operators gathered in collaborative working groups, which gave the OpenConfig [76] and the OpenROADM [77] initiatives, with OpenConfig data-model sets being more generic than OpenROADM ones [3] but less efficient for optical path computation and deployment [78]. To directly communicate with the optical equipment, protocols such as NETCONF [79] are used, though traditional protocols are relatively slow for the monitoring data gathering, so as explained in 2.1.4

3. Though OpenROADM is also compatible with optical transponders

protocols enabling telemetry services have been developed to palliate this problem. The two most common for this usage are gRPC [80] and gNMI [81] (an evolution of gRPC). Figure 3.4 summarizes the communication process between the controller and the optical hardware. This figure shows the necessity to implement interfaces in the optical equipment for communication with the network controller, and modern optical equipment have a controller that manages remote communications and protocols.

Now able to communicate efficiently with the hardware and with the help of data models, the network controller perceives the hardware as "black boxes", ie. an abstract system producing outputs from inputs, with no concern on how the outputs are produced. This fully abstracted view of the hardware allows the network controller to virtually manipulate the equipment and organize the network as it needs. One example of these manipulations are the partition and the aggregation. As shown in Figure 3.5, partition is the action of slicing a device into multiple "smaller" virtual devices, and aggregation is the reverse operation, combining multiple devices into a "bigger" one. By separating an equipment into multiple virtual ones, it is possible to change the parameters of each individual virtual device independently, and by aggregating multiple device together it is possible to create a "super device" who can have for example better throughput or more inputs and outputs to manage the traffic.

Creating and handling virtual devices allows the operator firstly to handle equipment from any combination of vendors, creating "disaggregated" networks, this also allows the network controller to create Virtual Optical Networks (VONs), or network slices, that are subdivisions of the physical network acting as different networks with different requirements such as security, latency, capacity, reliability,... These VONs are allocated a number of virtual devices and an amount of frequency slots and can be dynamically created and deleted depending on the current needs in the network. The allocation of frequency resources has been facilitated by the introduction of the flexible grid and the greater reconfigurability of current optical hardware enables fine tuning of the virtual device parameters to respect the constraints posed on the VON. When a great number of VONs are expected in the network, the network controller can also allocate resources in the control plane to instantiate a SDN controller to supervise one or multiple VONs [6]. Another layer of virtualization in the network is the Network Function Virtualization (NFV) [82], where even the functions of the optical equipment (such as switching, firewalls, ...) are also decoupled from the hardware. These virtualized functions or Virtual Network Functions (VNFs) can be deployed remotely to a server in the control plane or

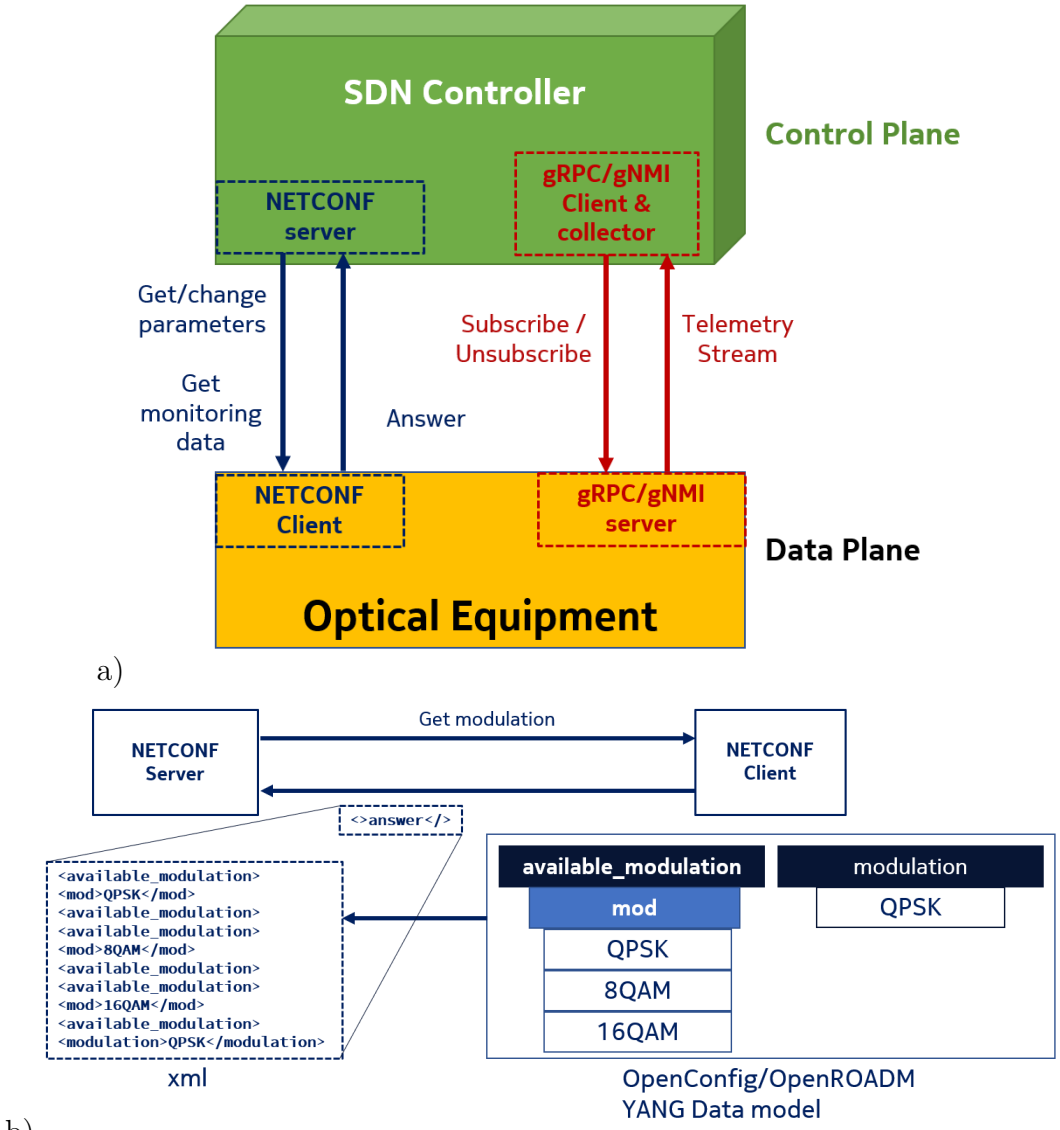


Figure 3.4 – a)Representation of controller-equipment communication b)Example of a server asking a client to give him his list of modulation formats and the current format in use. The answer sent to the server is coded into an xml file for this example, but the answer can be coded in any language the protocol for server/client communication supports

a free computational resource available in the network depending on the current needs, which is especially useful for VONs that have requirements in security, latency etc... VNFs usually run on temporary virtual machines but work is done towards full FPGA-based VNF functions for packet processing [83] or help accelerate the VNFs [84]. Figure 3.6

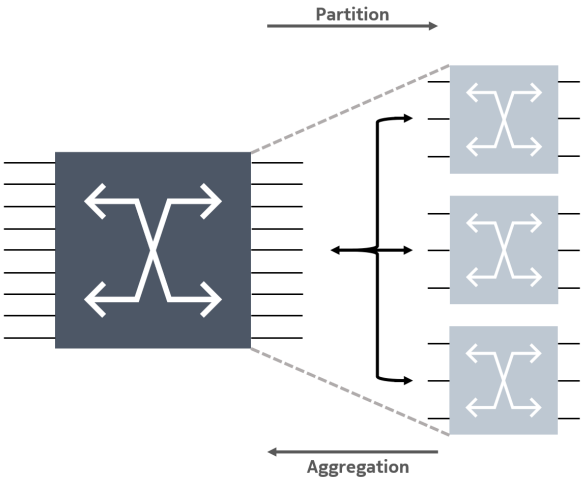


Figure 3.5 – Base principle of hardware virtualization using partition and aggregation

shows a representation of a virtual optical network.

3.3 Reconfigurable optical transponders

As explained, flexibility is a prevalent concept in modern network design, and this has enabled and has been enabled by the design of new flexible optical equipment. We touched on the OADM evolution to ROADM in 2.1.2.2 which became more and more programmable with the ability to route any wavelength from and to any of its ports, which became a very important instrument of flexible optical networks, as the control plane can freely route the wavelength through the whole network by smartly programming the ROADMs under its supervision. But another equipment that is central in the establishment of flexible optical networks is the programmable (or flexible) transponder. Optical transponders used to be static devices with a set list of parameters, or a very low number of configuration possibilities. The progresses in optical technology and DSP techniques has enabled the optical transponders to have a wide array of possible parameters for the optical transmission and are able to adapt themselves to the ever changing conditions of the network. This also allowed the integration of virtualization techniques to better integrate the transponder into the SDN optical networks and increase its capabilities. This section will cover the advances in hardware and software that enabled this new generation of optical transponders.

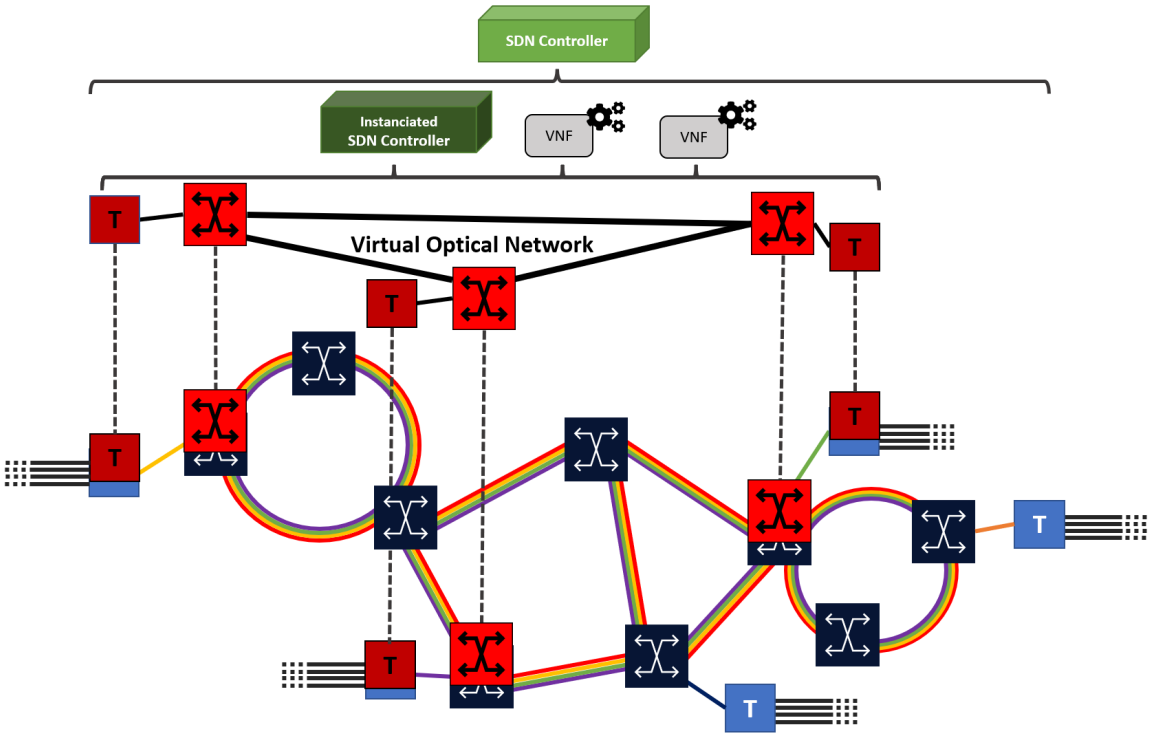


Figure 3.6 – Example representation of a VON in an optical network with VNFs

3.3.1 The Sliceable Bandwidth Variable Transponder

The Sliceable Bandwidth Variable Transponder, or S-BVT, is the most adopted architecture for optical transponders in optical network design and research. As its name implies, it is capable of be sliced according to the virtualization paradigm into multiple transponder, with each sliced transponder having each a different bandwidth depending on their needs. This optical device is the evolution of Bandwidth Variable Transponders (BVTs) and classical ("fixed") transponders that preceded them. We will firstly discuss the advances that allowed for reprogrammability in optical transponders before talking about the advent of sliceability in optical transponders, that led to the optical devices of today.

3.3.1.1 Bandwidth Variable Transponder

One of the first steps that led to the S-BVTs that are commonly used today is the introduction of reprogrammability of the optical transponder. Evolutions in system design allowed for transmission parameters change, such as laser wavelength, modulation

format, data-rate etc... These possibilities when exploited smartly can allow the operator to increase the quality of reception of the signal by reducing the modulation format order for example, as pictured in Figure 3.7, and on the other way increase the capacity of the optical link by increasing it. Having this layer of flexibility in transponder configuration helps better pre-plan the network, increase the capacity of the whole network in the long-term and reduce the number of equipment needed in the optical network, as a single transponder can be used for multiple applications [85]. A reconfigurable transponder is called a flexible transponder or most commonly in the literature a Bandwidth Variable Transponder or BVT.

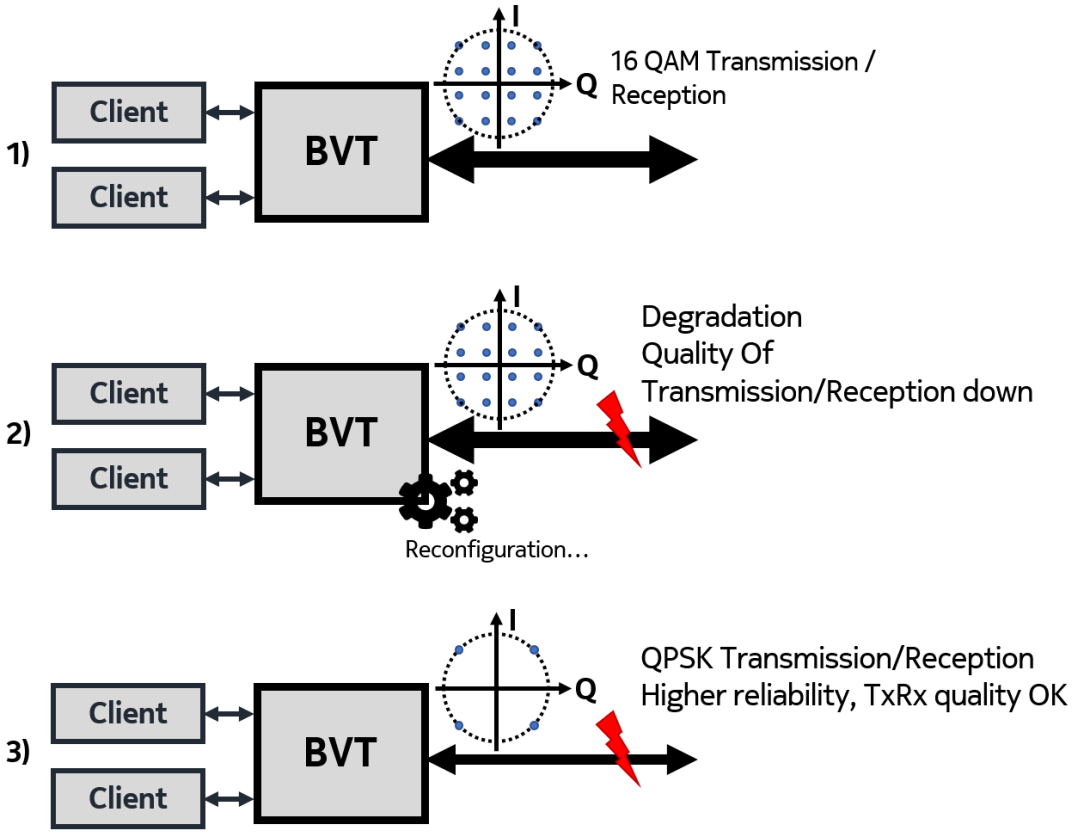


Figure 3.7 – Classical use case for BVTs, decreasing the modulation format order to increase the transmission/reception resiliency to degradations in the optical path

One technology that helped the implementation of the flexible transponders is the tunable laser. As the name implies it is a laser that is able to change its wavelength. They were already in used in fixed grid networks, needing only to tune to channels centers that were periodically spaced due to the 50GHz grid. But the flexible grid scenario and the

introduction of the superchannel technique have made necessary for the tunable laser to be able to virtually generate any wavelengths used in WDM networks [86]. One possibility is to use an array of reconfigurable lasers, which increases significantly the number of available wavelengths, which creates higher engineering and energy costs [87].

One way to dynamically increase the capacity or the reliability of the optical communication between two transponders is to change the modulation format. For example, changing from 16QAM to QPSK, which is more resistant to perturbations in the optical channel, when the communication quality degrades, or vice versa if there is a need to boost the capacity of the link. This requires an optical transmitter capable of transmitting in multiple formats. Several solutions have been proposed. By using a Dual-Drive MZM (DD-MZM, a MZM whose both arms are driven by two independent electrodes) chained with a DP-MZM, the author in [88] allows for a switch between QPSK and 16-QAM by changing the bias voltage and the amplitudes of the DD-MZM or a DD-MZM and a Phase Modulator to switch between multiple PSK and QAM formats in [89]. Similarly in [90] by using tunable interleaver filters in an optical transmitter, the author is capable of switching between QPSK, QPSK-OFDM and 16-QAM. [91] presents a transmitter structure using a FPGA for traffic processing and a fast programmable PM-16QAM/QPSK transmitter circuit that can perform a change of modulation format in under a second. In [92], the author demonstrates a FPGA based transmitter with multi-format generation by re-writing the contents of a LUT-based structure composing the modulation format encoder which allows for fast modulation format change (around 5ns) between BPSK and 64QAM at 28GBd. In [93], authors show in a field trial that a switch of modulation format using a production-grade 64GBaud transponder from PM-QPSK to PM-16QAM takes approximately 35s, which shows that even if modulation format reconfiguration is available is commercial products, fast switching is still far from being a reality.

Having a transmitter capable of switching between multiple formats is good but not enough to propose a full transponder able to transmit and receive multiple modulation formats. As the modulation format changes, adaptations are necessary in the DSP of the transmitter and/or the receiver. From the mapping of the symbols to the phase recovery, most modulation formats require some changes and building the DSP accordingly is critical. As explained in 2.2.3.2, modulus recovery is different if the modulation format employs a unique modulus for symbols or not, and carrier and phase recovery can be performed using algorithms that can adapt themselves depending on the order of the modulation [94]. The other thing to take into consideration in the DSP of BVTs is the

fact that increasing the order of the modulation also decreases the maximum optical reach of the signal, which in turn increases the amount of impairments that should be (pre-)compensated to ensure ideal quality of reception of the signal [95]. Algorithms and their implementations should be flexible and be able to adapt themselves to the change of modulation format, as employing the "ideal" algorithm for impairment compensation with the current modulation also reduces the power consumption of the DSP [96]. Another way to increase the reliability or on the contrary increase the spectral efficiency of the communication between two transponders is to adapt the FEC codes, increasing or reducing the redundancy depending on the situation. In [97] the author proposes a reconfigurable FPGA implementation of both TX and RX DSP where the FEC encoding and decoding can be changed depending on the need of the optical channel.

3.3.1.2 Sliceability and virtualization

With the changes in optical transponder hardware and the push towards more virtualization enabled by the research on SDN controlled optical networks, the concept of multi-flow, or sliceable transponder has been more and more central. The first important characteristic of a sliceable transponder is that it is multi-flow, traffic should come from a N number of client interfaces and should be routed to a M number of line interfaces [7] and secondly client signals that go to the same destination in the network should be concatenated using the super channel technique and the possibilities offered by the OTN framing [98], see Figure 3.8. The multi-flow ability is also extremely helpful in the case of a link failure to redirect the optical flows and mitigate its impact [99], as shown in Figure 3.9, and in addition their ability to efficiently route the traffic into multiple different paths helps reducing the number of necessary add/drop ports in ROADMs [100].

With this multi-flow ability, it is now possible to, with the proper setup, create virtual transponders, slicing the equipment into multiple transponders, each having one or more line and client interfaces depending on the needs, putting the desired client signals into the transport network where they will go to their desired destination [101], see Figure 3.10. This flexibility in structure helps using the hardware more efficiently and brings down the overall costs in the network, notably in the number of used equipment, as a single sliceable multi-flow transponder can act as multiple classical transponders that can be configured and reconfigured freely depending on the current needs of the optical networks [98, 102]. A virtualizable multi-flow reconfigurable transponder is commonly

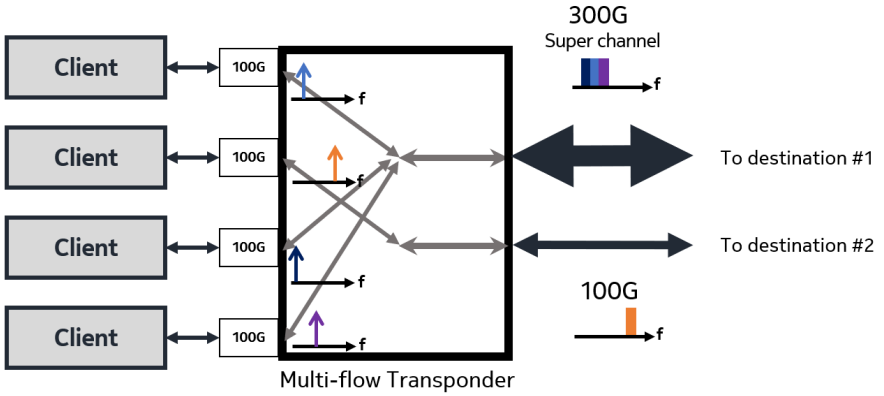


Figure 3.8 – Representation of the principle of a multi-flow transponder

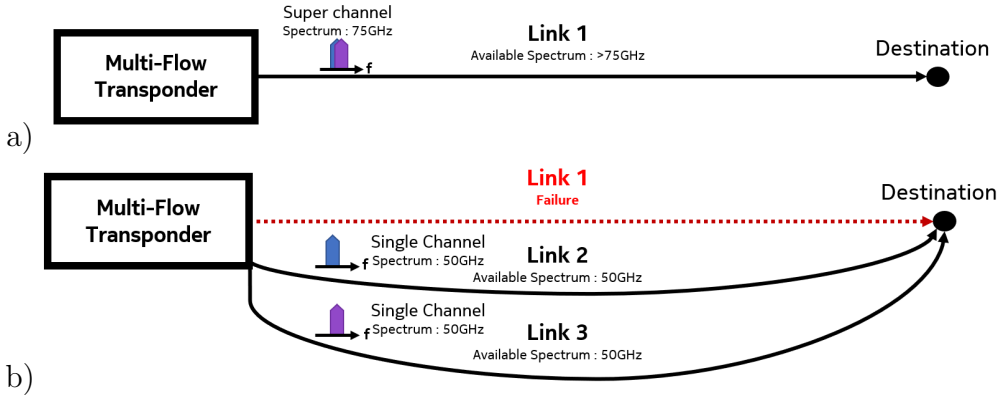


Figure 3.9 – Restoration after a link failure using a multi-flow transponder

called a Sliceable Bandwidth Variable Transponder (S-BVT) [4]. To efficiently perform such operations, the control plane should be able to allocate efficiently the resources of the network, taking into account the higher degrees of flexibility and complexity allowed by the S-BVTs [103], and a proper YANG model should be implemented to efficiently configure and virtualize the hardware [104].

One example implementation of a S-BVT can be found in [105]. It is based on a programmable multi wavelength laser source that can generate three sub-carriers, i.e. three channels, with said channels being routed to three different receivers in the network, with each receiver having a tunable laser used as a local oscillator to receive properly each sub-carriers. The authors in [106] present a multi-flow transmitter circuit able to operate with a single or two flows by respectively combining or using independently two IQ modulators. In [107], the authors propose a S-BVT architecture using a programmable

4. The term V-BVT for Virtualizable BVT can also be found in the literature

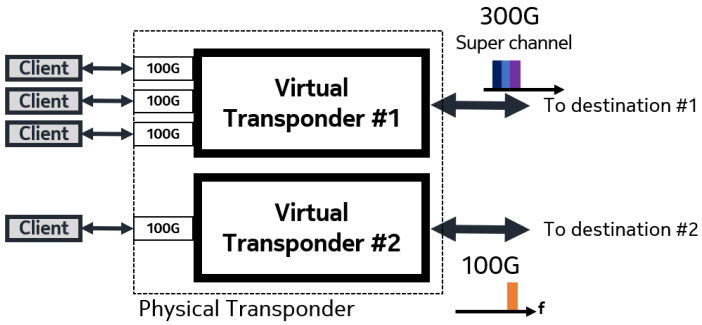


Figure 3.10 – Representation of how the multi-flow transponder in Figure 3.8 can be virtualized into two virtual optical transponders by the control plane

distance module, which for each sub-channel of a super channel determines an ideal modulation order depending on it's final destination and/or on its dropping point in the link, saving spectrum resources. Authors in [108] present a S-BVT architecture employing a FPGA-based DSP allowing generation of a number of channels that can be superior to the number of sub-carriers generated by the multi-wavelength sources in the system. This gives the benefit of being able to generate sub-channels with very low capacity, boosting the flexibility of the device.

Another current topic in transponder virtualization is the introduction of the white boxes in the optical transport network. White boxes are physical devices that can be composed of a multitude of optical devices from any number of vendors, with additional software enabling virtualization of the full device. The white boxes are controlled by a centralized controller, communicating with software agents in the white boxes. Allowing the use of equipment of any vendor inside the optical network (where traditionally a single vendor can take care from the equipment such as the OADMs and the transponders to the controller) is expected to reduce the overall costs in equipment, at the cost of greater engineering, deployment and supervision costs for the operator, a radical change in how the optical network is classically operated [8]. One software tool for generic implementation and deployment of optical hardware and white boxes using transponders is the Transponder Abstraction Interface (TAI) [109] API, which abstracts parts of the hardware, such as the transceiver modules, to ease the interfacing of multi-vendor components. In Figure 3.11 an example representation of a white box is depicted.

An example of a white box implementation can be found in [110], where the authors use two transponders (with each having different characteristics), monitoring modules and filters for add-drop multiplexing and switching and an agent for the network controller

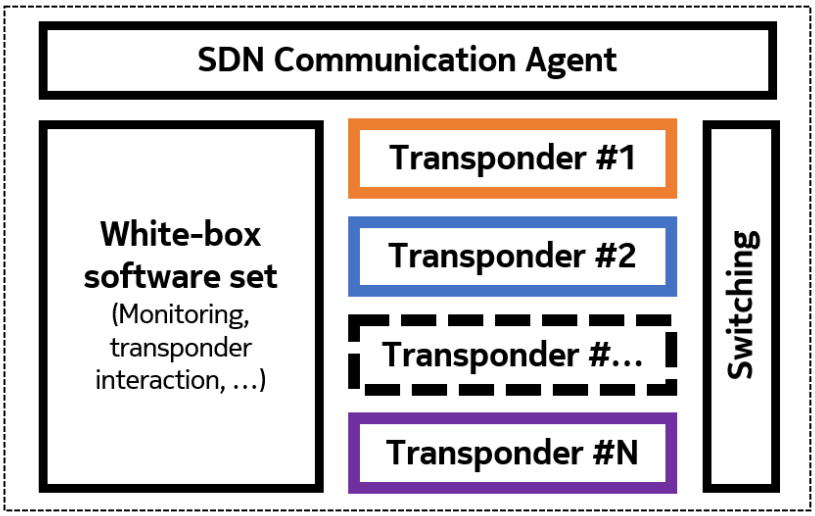


Figure 3.11 – Representation example of a white box using transponders, each one of them possibly from different vendors. An software agent handles communication with the control plane, and software helps handling and monitoring the array of transponders.

communication using NETCONF. In [111] shows a full 100G and 200G communication using a white-box transponder and multi-vendors CFP2 transceiver interfaces, with both DCO and ACO modules (as explained in 2.2.6), using TAI to abstract the hardware for the network controller. All these solutions for transponder virtualization helped the network controller better interact the resources available in the network and lead to a better utilization of the network elements.

3.3.2 Automatic hitless reconfigurable transponders for flexible optical networks

As we described in the previous subsections, transponders are now more flexible than ever, and more and more parameters are reconfigurable. However the reconfiguration process can still be a problem to handle in optical networks: the change of modulation format, laser frequency, FEC code, etc... leads to the transponder (or at minimum the port that is reconfigured) being out of service for a period of time that can range from a few milliseconds to multiple minutes. This is detrimental for the overall flexibility of the network, as reconfigurations become exceptional events, with transmission and reception parameters and optical links set in stone until a new equipment is added or a failure occurs. Even more so, with more and more devices under supervision by the optical control plane,

handling efficiently every single deterioration in service quality that can happen in the network and detecting them before they increase the bit error rate above tolerable levels is also very challenging logistically. Furthermore, synchronizing a change of parameters between a transmitter and a receiver to prevent any incompatibilities that may happen because of a reconfiguration is crucial to make sure the communication can still happen (eg. the transmitter now uses QPSK modulation after a reconfiguration while the receiver is still configured to receive 16-QAM or is still reconfiguring itself to handle the change of modulation). All these problems are creating a bottleneck in the adoption and full exploitation of flexible networks. One solution to these problems is the development of automatic and hitless reconfigurable transponders.

Hitless reconfigurable transponders are able to reconfigure themselves without an interruption of traffic, making them ideal for elastic optical networks [2], as reconfiguring the hardware to a more suitable set of parameters has potentially no drawbacks. A few transmission parameters have been shown to be reconfigurable hitlessly. In [112] is proposed a multi-wavelength laser source able to tune the spacing of the sub-carriers hitlessly, which is ideal for an implementation of a hitless reconfigurable S-BVT, allowing hitless spectrum occupancy adaptation. In [113,114], authors show architectures of flexible transmitter able to hitlessly change baud-rate. Switching from an initial baud-rate to a two times inferior baud-rate (as in [113] from 14Gbd to 7Gbd) is achieved by repeating a symbol twice (or four times to achieve a switch from 28 to 7 GBd in [114], with a simulated transmitter). Changing baud-rate by repeating symbols is efficient and a relatively easy solution to implement compared to a change of modulation order, and keeps the advantage of increasing maximum reach at the cost of lower data rate.

Even though hitless reconfiguration of the transmission parameters of a transponder would be ideal for flexible networks, over-reliance on the centralized control plane hampers the reliability of the communication and reactivity of the systems. Alarms raised by the equipment and sent to the control plane are added to a queue and delays to process the degradations in the optical path accumulate even further when the number of optical links to manage in the control plane increases [115]. Research on alternative and less centralized architectures of control plane have shown that allowing some decentralization control with local monitoring and decision agents placed strategically in the network reduces the quantity of data that have to be transmitted to the control plane and decreases overall reaction time in case of a soft failure [10]. Even though it is unrealistic to make transponders fully independent of the network controller, most notably because the control plane has

awareness of the condition of the whole network which is crucial for optical link creation and management, adding more intelligence to the device is a good solution to get fast reconfigurations and ensure good transmission and reception of the optical signal. Enabling intelligence in the transponder can be made by exploiting the monitoring data already collected by the device and having means of taking action based on the available information. In [116], the author proposes a concept of autonomous intelligent transponder able to change its modulation format to increase the capacity of the bidirectional link between two of the proposed equipment. The transmitters start at a low modulation format order (in their implementation QPSK) and gradually increases this parameter depending on the BER of the received signal. While this approach reduces the risk of using a modulation format that crosses a set threshold of BER, it does not talk about the synchronization between the two transponders, and supposes the bi-directional link is always the same in both directions quality-wise. Also progressively increasing the modulation format is good to increase performance but continuous reconfigurations implies multiple interruption of traffic and prevents further reconfigurations if the ideal modulation format has been found and a degradation in the optical path occurs.

In [115], authors propose a solution of pre-programmed transponders to handle degradations in the optical path. The SDN controller pre-computes restoration plans as finite state machines for transponders (with a set of parameters to apply for a type of optical link impairment) that the transponders will apply automatically and independently, and notify the control plane when the reconfiguration is finished. While the pre-programmed solution offers high scalability and reduces the risks of incompatibility after the reconfiguration process as the SDN controller has decided on the parameters sets, it does not tackle again the link between the two transponders, and how the Tx transponder is aware on the degradation at the reception side.

One way to handle the receiver adaptation problem is to have a receiver capable of detecting a change of transmission parameter and adapt on-the-fly to the change. In [117] adds a rate change controller in both Tx and Rx to signal a change of modulation format by adding symbols in the data stream and let the receiver adapt the DSP algorithms and hitlessly detect the change of modulation format order. In [118] proposes a 3-DSP system, with each DSP subsystem able to properly receive a different baudrate (one for nominal, one for half and one for quarter symbol rate). The output from the DSP for nominal symbol rate is used as a training sequence for the others, and when a change of baudrate happens in the transmitter the DSP algorithms can quickly converge and hitlessly detect

the incoming signal, though this work does not discuss the actual implementation of such a system as it is demonstrated offline. While these solutions are convincing, with the future of commercial optical transponder being more and more virtualized, with equipment that can come from multiple vendors with heterogenous capabilities in transmission and reception, it is very important to ensure that a change of parameters can be performed in the transmitter and processed in the receiver. This proves to be a challenge if we consider that the autonomous transponder is a way to go for future flexible optical networks, it is both important to have as many possibilities of reconfigurations as possible, with as little impact as possible on the traffic. If these conditions are met, the number of reconfigurations happening in the optical network would drastically increase, both in number and frequency, leading to optical equipment being used more optimally and the optical network being more reliable at the same time.

3.4 Conclusion

This chapter covered the evolutions over the last years of the optical transponder, the equipment at the center of this thesis work. We reviewed how the drive for flexibility in system design and in network management has influenced the newer functionalities of optical transponders. However we also discussed the challenges surrounding optical transponder design to make fully flexible optical networks a reality, namely the integration of hitless and automatic reconfiguration capabilities.

Chapter 4

Real-time optical flexible transponder prototype for software defined networks

4.1 Introduction

There is a trend to bring more intelligence to the network elements in the last years to meet the increasing need for bandwidth and reliability in optical networks [2]. Commercial transponders have a great number of adjustable parameters, such as baud-rate, modulation format, wavelength and more advanced features such as probabilistic shaping, making them able to adapt to match traffic demands [25]. However, changing a parameter on-the-fly is not common due to the necessity to perform an interruption of service during the change of parameter. In [93], the authors demonstrated in their field trial with commercial hardware that switching from 100Gb/s with Quadrature Phase Shift Keying (QPSK) to 200Gb/s with 16QAM takes about 35s. Work has been done towards fast reconfiguration, e.g. [91] reported about 1s reconfiguration, [113] showed 10µs reconfiguration which allows for zero loss of traffic (hitless behavior). Transponder are still widely used in a static manner on field and require optical network margins to cover evolutions along the years such as traffic conditions, physical layer impairments and equipment aging.

Close-to-zero margin network operation, which is now a very active research area [9, 119, 120], allows to fully unleash this flexibility by reconfiguring a connection after the detection of a degradation (soft-failure). To do so, monitoring of the physical layer is therefore required. Data are then generally collected at the management and control plane to build a significant database for the decision-making process. With streaming telemetry [5], it is possible to collect data much faster, in the order of seconds, improving the accuracy of data correlation between different connections, but also (if available) environmental conditions. This remains limited by the bandwidth of the control plane network.

The centralized architectures [121, 122] are very useful (i) for long-term behavior whose objective is to anticipate any issue with what-if scenarios; (ii) for troubleshooting whose objective is to understand why a (soft) failure occurs; (iii) for real-time reconfiguration whose objective is to react fast to prevent outages.

Alternative architectures have emerged to address the latter. Stream processing has been proposed in [123] to generate alarms from a video and telemetry streams. Pre-computed instructions can be loaded in network element agents to perform delegated restoration [124]. In case of transponder reconfigurations, distributed decision is proposed in [116] based on an offline data-aided DSP knowledge. However, [116] does not tackle the communication protocol between the two transponders. We believe real-time reconfigurations can benefit from (soft-)failure detection and local decision-making to react faster [10]. It also helps reducing the data that needs to be transmitted to the control plane which may become cumbersome with the massive monitoring and data-driven paradigms envisioned for next generation autonomous optical networks.

In this chapter, we evaluate a real-time transponder with local decision and reconfiguration capabilities based on the use of commercial equipment. We start by presenting this prototype transponder in Section 4.2.We introduce, in 4.3, the proposed auto-negotiation protocol for optical transponders, which enables fast and synchronous change of parameter. The auto-negotiation protocol is based on the inline insertion of small messages. We implement it to operate in real-time using an FPGA platform and validate its latency and packet loss with traffic up to 100 Gbps. In 4.4, we integrate our auto-negotiation FPGA implementation and a Raspberry Pi with a commercial Nokia PSI-2T transponder, and we discuss how to locally detect an impairment and trigger the auto-negotiation protocol. In Section 4.5 we discuss the interfacing of our solution with a centralized control plane, that will be necessary for Section 4.6 where we integrate our prototype transponder inside a network testbed. We experimentally measure the time to detect a fault and to restore the quality of transmission, and compare it to the centralized control plane. In 4.7 we introduce our auto-negotiation setup in a real-time setup with variable baudrate real-time transmitter and receiver and showcase a synchronized change of rate of transmission.

4.2 Prototype transponder presentation

In Figure 4.1 we introduce our prototype transponder with auto-negotiation capabilities and monitoring data processing. When the prototype transponder detects a degra-

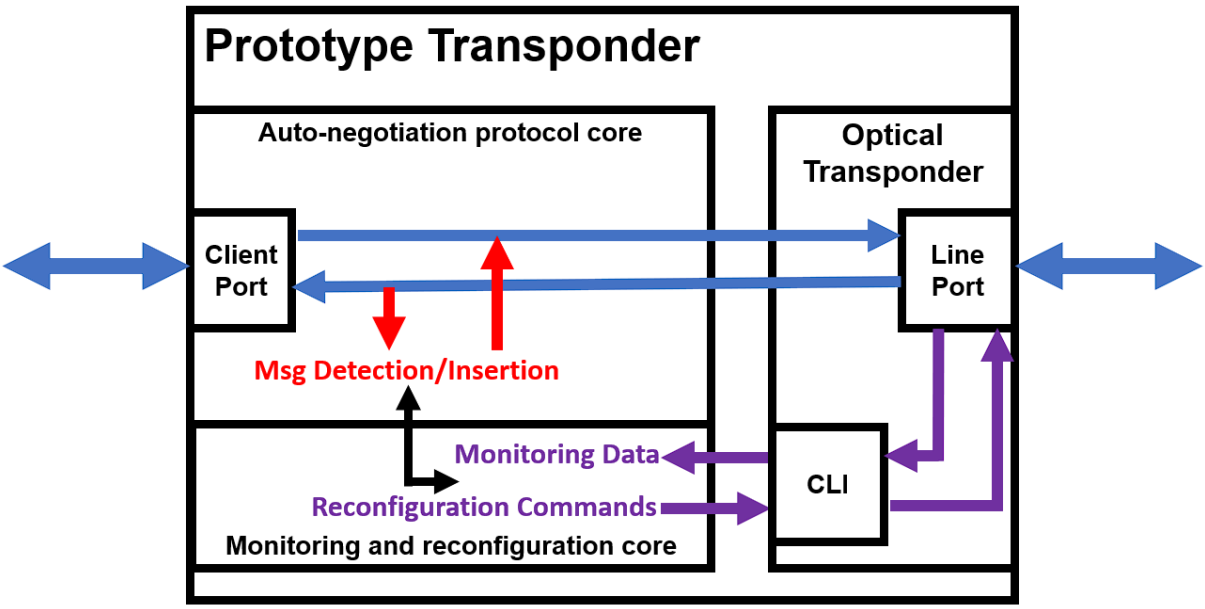


Figure 4.1 – Representation of our prototype transponder with auto-negotiation capabilities.

dation in traffic quality, it starts the auto-negotiation process with the transponder it is in communication with to adjust one transmission parameter and restore the quality of service to an acceptable level.

It is composed of three main functional blocks. The first one is an optical transponder in itself, as presented in Section 2.2, with reconfiguration capabilities. A monitoring and reconfiguration core will retrieve its different monitoring data, process them to detect the drop in quality of service and send reconfiguration commands using the Command Line Interface of the transponder. Finally an auto-negotiation core will add and extract auto-negotiation messages to synchronize the change of parameters with another prototype transponder it is communicating with.

In the following sections, we will provide further descriptions of the functionalities and implementation of the auto-negotiation and monitoring and reconfiguration cores of this prototype transponder, and validate its performance against a centralized control plane in a network testbed.

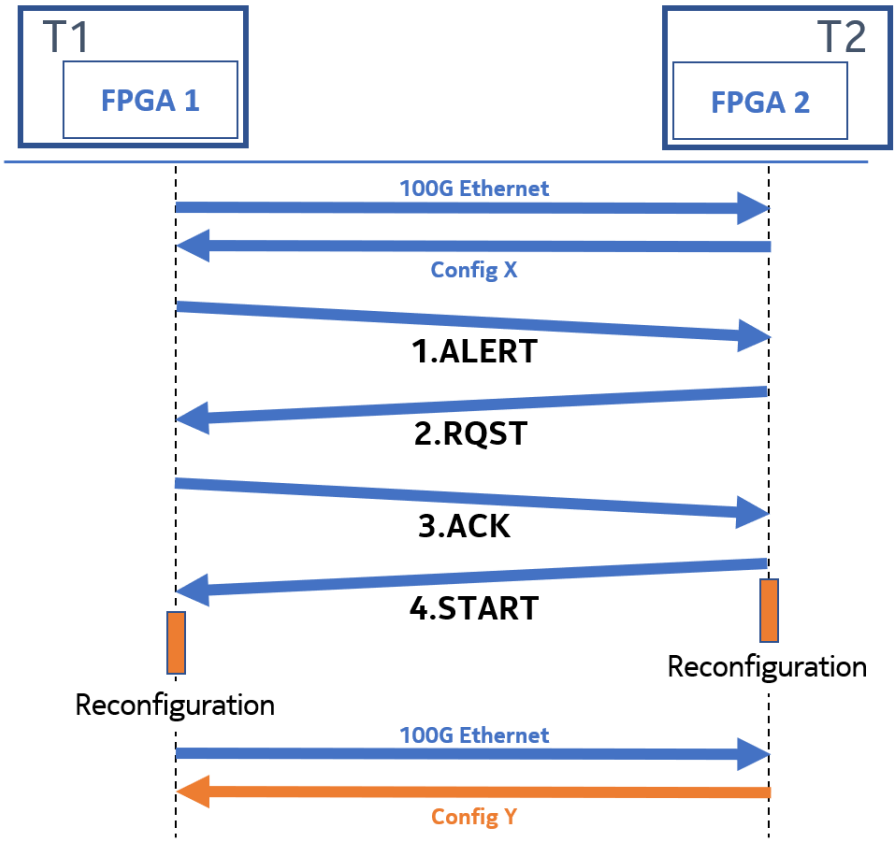


Figure 4.2 – Sequence of operations of our auto-negotiation protocol

4.3 Auto-Negotiation Protocol

4.3.1 Design

In this section, we design a communication protocol between two transponders to adjust transmission parameters and to synchronize when this reconfiguration starts. It can be set off centrally by the effect of a network controller or locally after internal monitoring and anomaly detection.

Whenever a transmitting transponder has to readjust one of its transmission parameters, it sends a message to the receiving transponder by adding a few bytes into the data stream. It can be located, for instance, just after the header of the Ethernet frame or in the Optical Data Unit (ODU) header in the EXP (experimental) or GCC (general communication channel, a channel for designer/manufacturer proprietary communication)

fields [21]. In all cases, the messages are required to be kept short: only two to four bytes are available per ODU header, and it is not desirable to insert too much data into the Ethernet traffic.
In Figure 4.2 we represent the workflow of our auto-negotiation protocol. If the receiving transponder T1 detects a drop in quality of received signal thanks to an internal monitoring function, it sends an ALERT (step 1) message to warn the transmitter T2 about the degradation. This step is optional when the procedure is triggered by the centralized control plane. Then, the transmitter T2 selects a new appropriate transmission parameter and sends a RQST (request) message (step 2). This message contains the parameter name and value that will change. It also contains the number of training frames that will be sent by the transmitter after the change of parameters, if needed. Training frames are empty data frames that are meant to help the receiver algorithms to converge to the new set of parameters or delay enough the resumption of communication while the receiver is reconfiguring itself. Parameter and value identifiers are written in the message and shared between transponders. When the message is received and extracted, the request is acknowledged (ACK) (step 3), if the receiver T1 can accept the configuration change and, when needed, if the number of training frames seems sufficient. When the ACK message is received by the transmitter T2, the change has been negotiated and the transmitter T2 sends a START message when it starts its reconfiguration (step 4). If the change of transmission parameters requires a short-enough interruption of traffic, the traffic is stored in memories during the process. When the receiver T1 extracts the START message, it starts its own reconfiguration, if needed. When the transmitter T2 is ready it sends the negotiated training frames, then sends the stored frames and, finally, normal operations resume. This protocol ensures the auto-negotiation property of the communication between two transponders. It can also exhibit a hitless property in specific reconfigurations (e.g. fast enough change). We chose to implement the auto-negotiation protocol in-line, i.e. in the header of the Ethernet frame or in the header fields of the ODU frame, to favor transponder reconfigurations. Usually the network controller needs to address many tasks at once related to the network devices, making challenging to ensure a tight synchronization between the two transponders during a reconfiguration.

4.3.2 Implementation

We designed a system to validate a real-time implementation of our protocol to be implemented with an optical transponder, illustrated in Figure 4.3. We implemented it

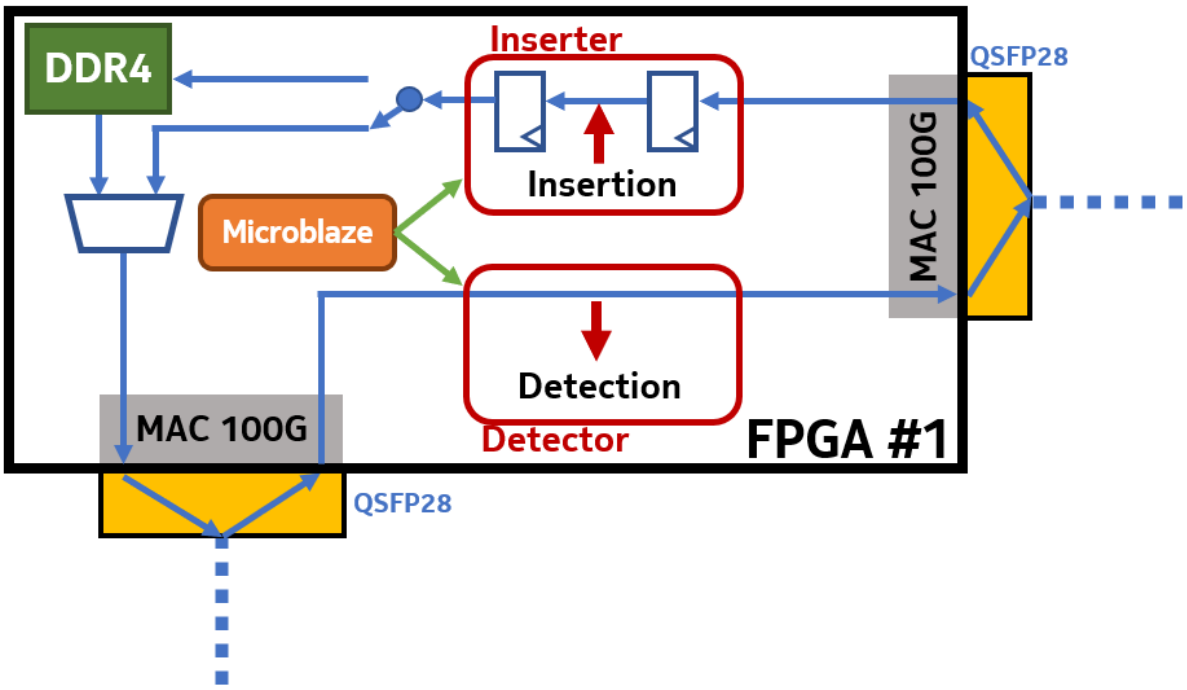


Figure 4.3 – Schematic representation of the real-time implementation of our auto-negotiation protocol, that will be integrated with an optical transponder in the full prototype (Figure 4.1)

using a Xilinx Virtex Ultrascale+ FPGA (see 3.1.1) evaluation board. In the FPGA a Microblaze [11] soft processor is implemented. It is running the FreeRTOS [12] real-time operating system (RTOS), an operating system which behave predictably in time-sensitive operations. All 100G ports are equipped with QSFP28 (Quad Small Form-factor Pluggable) modules (see 2.2.6). Standard 100G Medium Access Controller (MAC) cores are used in the FPGAs to properly decode 100G Ethernet traffic. To store the traffic during the reconfiguration if it requires a short interruption of traffic, we use DDR4 memory (in this implementation, 2GB). We developed in VHDL two logical modules to handle the protocol operations inside the FPGA. These two logical modules are an Inserter, that inserts inside the incoming Ethernet traffic a message, according to a set of parameters transmitted by the processor, and a Detector, that detects, extracts, and stores the messages that are found on the data path. With the help of AXI-4 Interface, the MicroBlaze is able to manage and control the modules. As the message insertion takes place between two 322MHz clocked shift registers, and we will use two FPGA boards with the two aforementioned logical modules for full communication, we can expect an

maximum addition of around 12ns in round trip time:

$$2 * (2 * \frac{1}{322MHz}) = 12.4ns \tag{4.1}$$

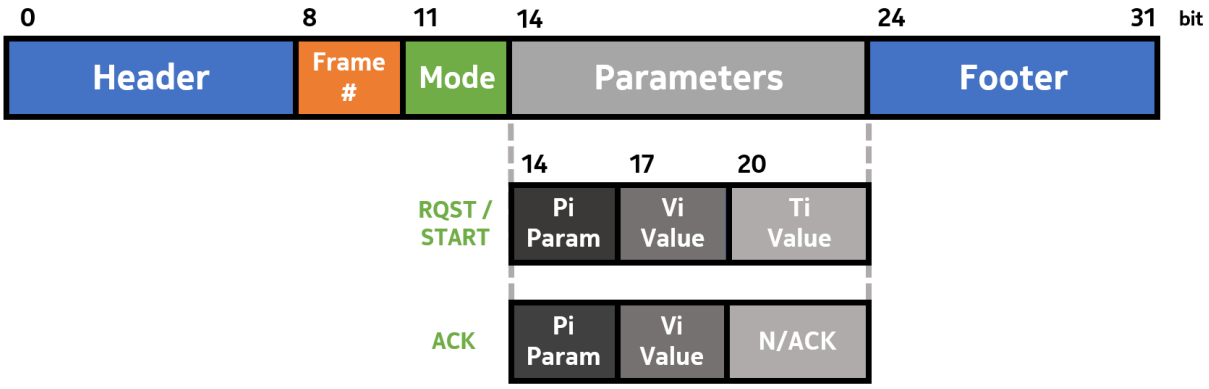


Figure 4.4 – Composition of the frames of the protocol in our implementation

Figure 4.4 describes the protocol frames. We decided to opt for a fast and simple implementation of 32-bit frames inserted in Ethernet traffic, at the start of the payload. They are composed of 8-bit header and footer, a 3-bit frame number for control purposes, a 3-bit Mode field to identify the type of message (ALERT, RQST, ACK, START), and a 10-bit Parameters field dependent on the mode. The ALERT message Parameters field is constant. The RQST and START messages have a 3-bit requested parameter identifier (Pi) field, a 3-bit value identifier (Vi) field detailing to which value the parameter will change to, and a 4-bit training frames number identifier (Ti) field. The ACK message has the same Pi and Vi parameter and value identifiers fields and replaces the Ti field to a 4-bit N/ACK acknowledgement field, where the receiver indicates if the request is acknowledged or not, or if it needs more training frames to adapt itself. This implementation of the protocol frame is an example that can be extended depending on the use case. In this implementation, we chose the parameters and the values that we are able to reconfigure. This choice should be bounded to avoid creating interferences with adjacent channels. For example, we use a 50GHz grid with 32GBd signals. Shifting the central frequency should be compatible with the 50GHz grid or expanding the bandwidth from 32GBaud to 44GBaudd is something common in commercial solutions. The bounding of the parameters can be done by the control plane, the operator or at implementation.

Figure 4.5 presents log extracts of the FPGA MicroBlaze processor with frames examples. We can see that we have the same sequence of operation as in Figure fig:chap4:protocolOperations,

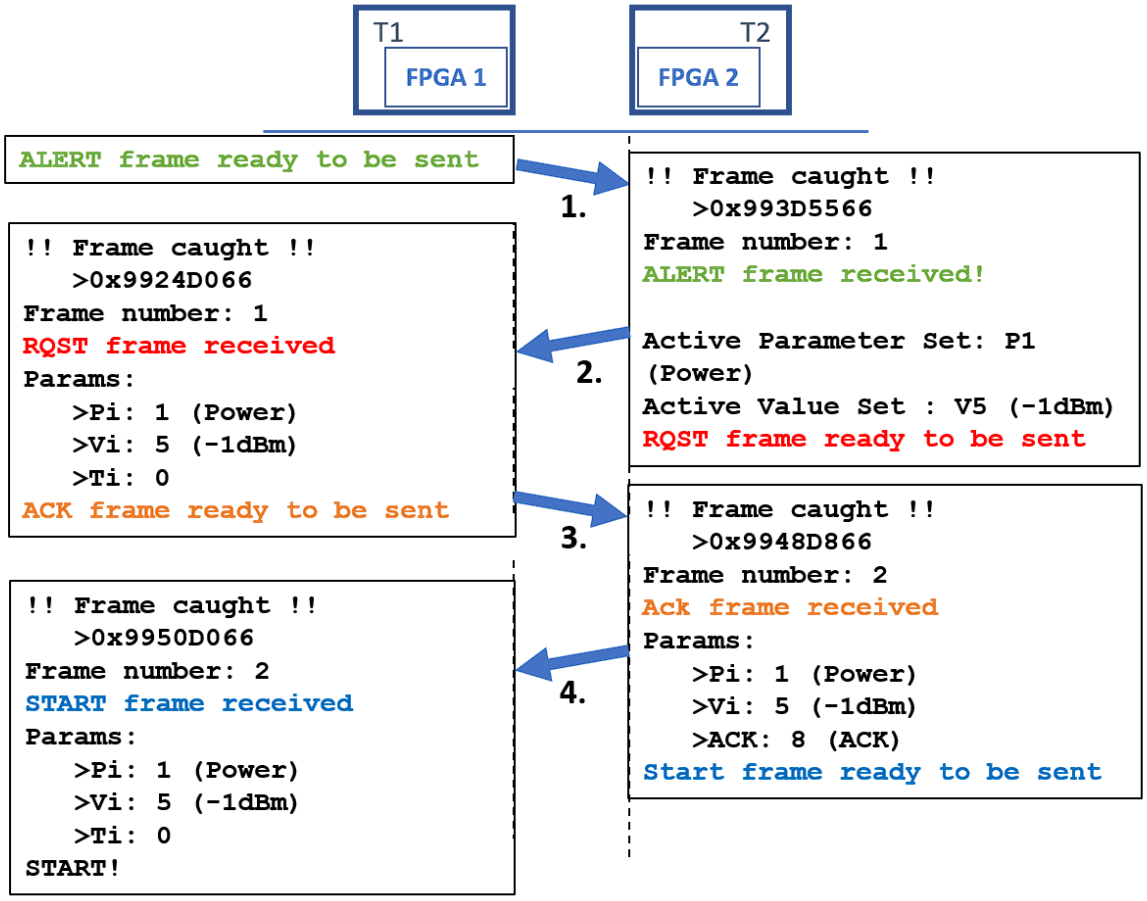


Figure 4.5 – Log extracts of the MicroBlaze processor on the FPGA cards during protocol operations. Indexes refer to Figure 4.2

where the frame number field increments for each message sent by a transponder and that the Pi and Vi identifiers are the same in both devices so they can understand each other.

4.3.3 Validation

To characterize our design with logical modules (labelled as "processing" in our results), we measured the latency of our system and compared it to a reference design (labelled as "reference") which is the same as our real-time implementation but without the Inserter and Detector logical modules (the parts in red in Figure 4.3). The setup for validation is represented in Figure 4.6 and pictured in Figure 4.7. It comprises of two of our FPGA boards with our protocol processing (as in Figure 4.3) separated by either 2m or 10km Single Mode Fiber (SMF). We use a Spirent Network Analyzer to generate and

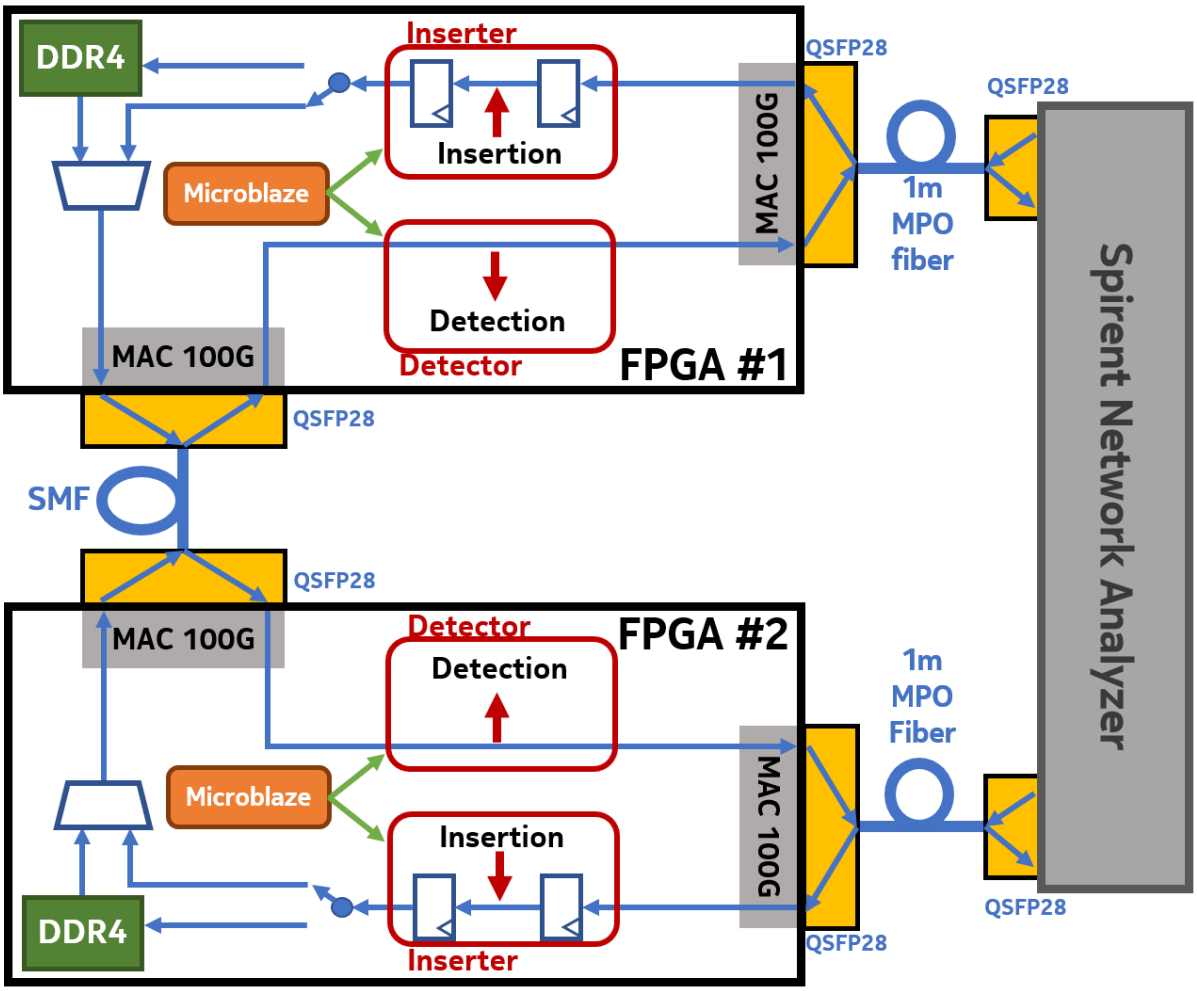


Figure 4.6 – Representation of our setup to validate our protocol implementation from 4.6, using two FPGA boards and one Spirent Network Analyzer to generate traffic and measure packet loss and latency

analyze 100Gb Ethernet traffic. The Spirent is able to generate traffic at multiple port loads and at multiple Ethernet frame lengths, and measure many metrics, with the two that we will exploit in our setup being the latency (how much time it takes for a packet to go from its source to its destination) and the packet loss (the number of packets that are received with errors or not received at all).

In the first experiment, the FPGAs are connected to each other via 2m SMF (Single Mode Fiber) and are connected to the Spirent network analyzer via 1m MPO (Multi-Fibre Push On) fiber. With the Spirent, we generate 10s of 100G Ethernet traffic at varying port loads and frame sizes, respectively from 10 to 100% and from 128 to 1518 bytes.

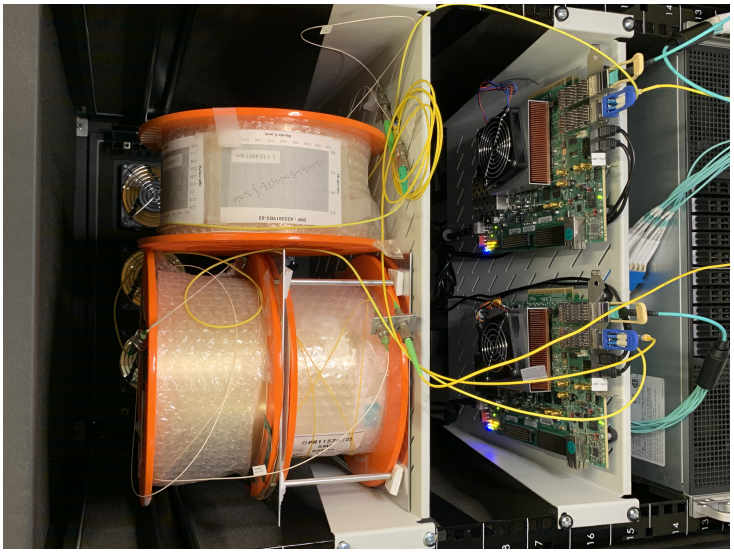


Figure 4.7 – Photograph of the setup with the two FPGA boards connected to each other and separated by 10km SMF. The Spirent is not pictured but is connected to the FPGA with the MPO (Multi-Fibre Push On, blue on the photo) fiber.

Our system also inserts the 32-bit auto-negotiation messages into the traffic at the rate of 100 messages per second. The quantity of messages per second has been chosen to insert a significant number of messages during the generation of traffic, but we do not expect real-life applications to insert as many messages in the traffic, as reconfigurations of transmission parameters are still an exceptional process. After that, we analyze the outgoing traffic from our systems and the results in latency, measured as a round trip time (RTT), and in frame loss, measured as parts per thousand frames received with errors by the analyzer. The results are in 4.8 (a) and Table 4.1 (a), respectively.
A second experiment is realized with 10km of SMF fiber between both FPGA boards, each equipped with one QSFP28-LR module (see photo in Figure 4.7). The Spirent, our detector and our inserter behave just as in the previous experiment. The 10km fiber length has been chosen as it is the maximum distance for our QSFP28-LR modules. The results for latency and frame loss are compiled in Figure 4.8 (b) and Table 4.1 (b).

In the graphs in Figure 4.8 (a), if we take for example the 512 bytes frame size and we compare the difference in round trip time for loads at 50% and 100% we obtain for the reference design a RTT of 1.85µs and 3.082µs respectively, and for the processing design 1.862µs and 3.094µs respectively. If we select the same parameters when the two FPGAs are separated by 10km SMF (Figure 4.8 (b)) we have a round trip time of 102.88µs and 104.112µs for the reference design, and 102.892µs and 104.124µs for the processing design.

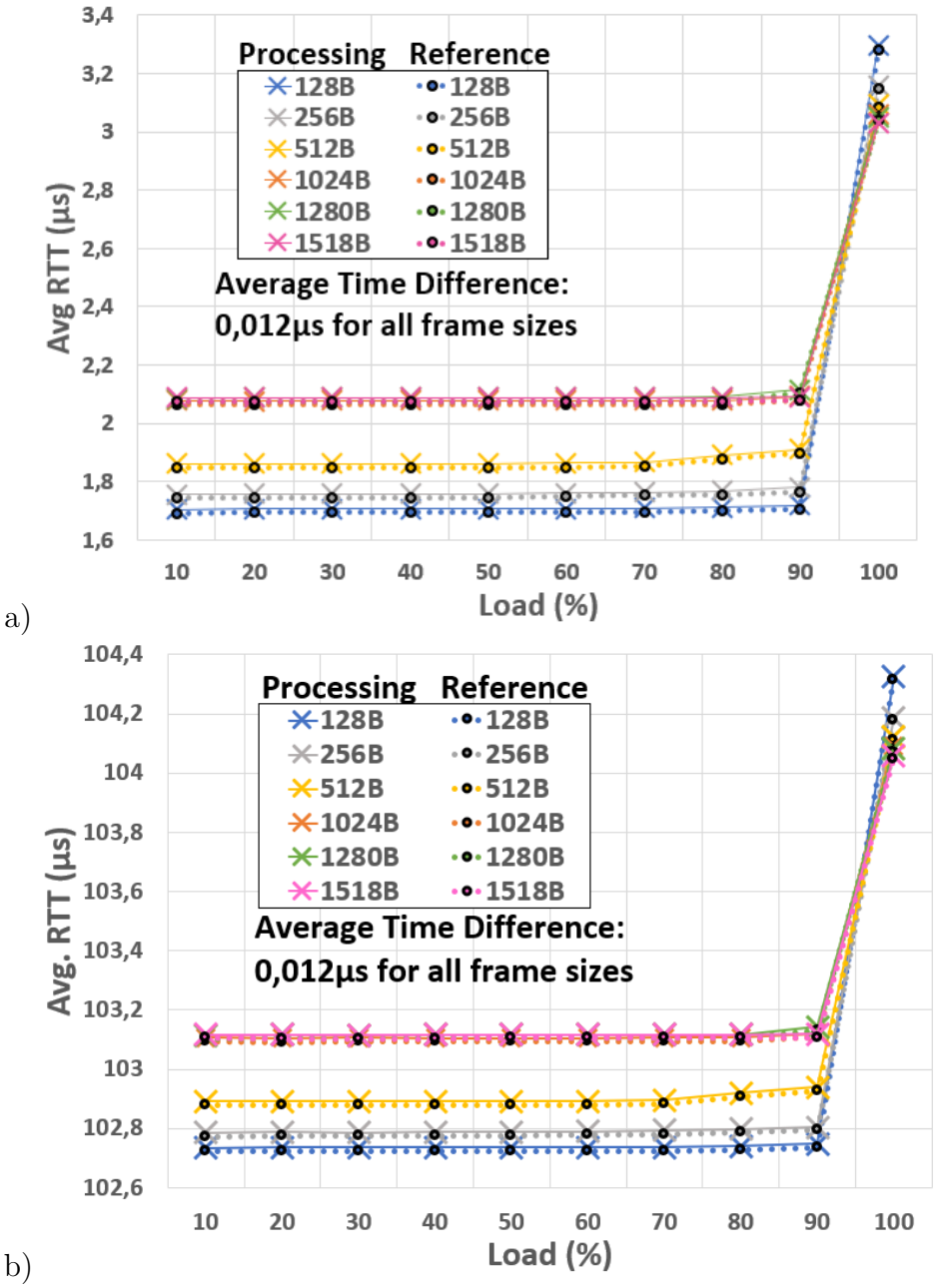


Figure 4.8 – Average Round Trip Time (RTT) in µs for frame sizes from 128 bytes to 1518 bytes, as a function of port load in %. The two FPGAs are separated by a) 2m, b) 10km SMF

	Frame size	128B	256B	512B	1024B	1280B	1518B
a)	Reference	0.012	0.009	0.01	0.009	0.013	0.048
	Processing	0.013	0.01	0.012	0.01	0.016	0.052
	Frame size	128B	256B	512B	1024B	1280B	1518B
b)	Reference	0.013	0.011	0.013	0.011	0.015	0.051
	Processing	0.013	0.01	0.012	0.01	0.016	0.052

Table 4.1 – Frame loss in ‰ for multiple frame sizes from 128 bytes to 1518 bytes, at 100% port load with a) 2m, b) 10km SMF separating the two FPGAs. Frame loss is null for all other loads

In these cases, the RTT difference is 12ns and is unaffected by the distance and the port load. On average, the reference design and the design with processing have 12ns of RTT difference for all frame sizes, with a standard deviation of 1ns. This measurement is in line with how we implemented our system (see Figure 4.3) and the result of equation 4.1. Frame loss is comparable between the two designs in both 2m and 10km fiber separation as the difference is very small, for example for 2m separation 0.01‰ and 0.012‰ for the reference design and the processing design respectively for a frame size of 512 bytes at 100% load, and is likely to come from our measurement precision. We may conclude on the frame loss by saying that it is coming from the hardware shortcomings at 100% port load and not from the implementation of our protocol.

We have validated our implementation of the auto-negotiation protocol, we believe that this protocol will allow for more reliable and more frequent transponder reconfigurations across the whole network, from metropolitan networks to core networks with longer transmission distances. But we need to implement a mechanism to trigger it automatically in case of a degradation in quality of service so that we can show its full potential for future automatic elastic optical networks.

4.4 Embedded monitoring processing and decision-making

We implemented a mechanism to automatically detect a failure in the link between the two transponders. The failure will automatically trigger the auto-negotiation process presented in Section 4.3.

We made the prototype presented in Figure 4.9 using a FPGA with a Microblaze

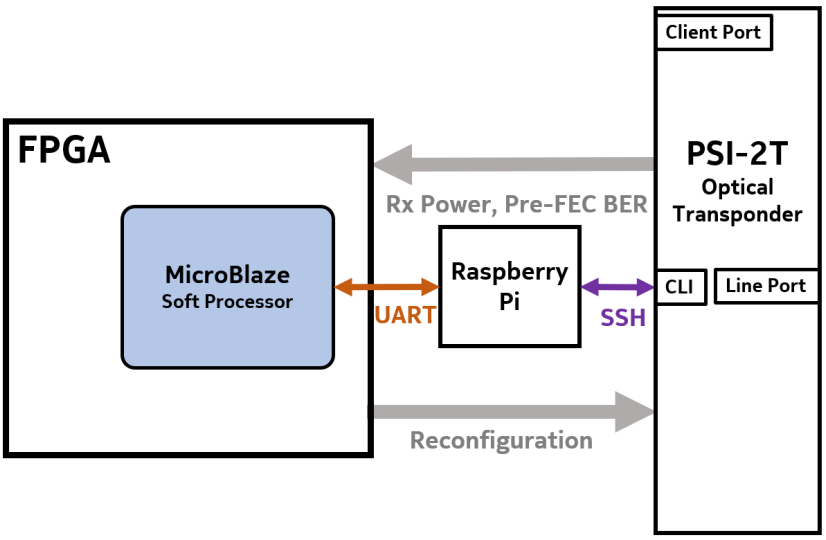


Figure 4.9 – Embedded monitoring and decision making prototype for commercial transponder Nokia PSI-2T

soft processor, running FreeRTOS. The Raspberry Pi helps us interface our FPGA easily with the Command Line Interface (CLI) of the commercial transponder Nokia PSI-2T (Photonic Service Interface-2T), sending reconfiguration commands from the FPGA to the transponder and sending the monitoring data from the PSI-2T to the FPGA board. The FPGA communicates with the Pi via an Universal Asynchrounous Receiver-Transmitter (UART) link and the Pi communicates with the transponder via Secure Shell (SSH). This setup with FPGA, Pi and commercial transponder is easy and fast to deploy for experimental purposes, rather than developing all the features we need directly inside the product. The round-trip time between the FPGA and the PSI-2T is 108ms on average, with the RTT between the Pi and the PSI-2T being 100ms on average, mostly due to the SSH communication. These times would be drastically reduced at the cost of increased development cost if we opted for developing our solution inside an existing transponder, but they are sufficiently low for our experimental purposes.

Our experimental setup reconfigures the Tx Power of the commercial transponder. This reconfiguration needs no interruption of service and can be used to compensate a loss in quality of reception. This loss can be quantified in terms of Rx Power and Pre-FEC BER by periodically collecting the monitoring data of the optical receiver transponder. The RTOS running in the Microblaze soft processor guarantees us that the monitoring data gathering is triggered periodically and consistently.

We can detect a degradation of quality of transmission thanks to the computation of the slope of the Rx Power with a linear regression function, and we can decide to act when the slope coefficient goes below a configurable threshold. It can be combined with a performance metric such as pre-FEC BER; if the Rx Power slope is below a threshold and if at the same time the Pre-FEC BER is above its threshold, we trigger an alarm and start the auto-negotiation process described in the previous section. This allows to restore the quality of transmission to an appropriate level. Ideally these thresholds should ensure that the received signal BER stays below the FEC limit, the limit at which the FEC decoder cannot correct all errors in the received signals. This limit depends on the FEC code used in the equipment, for the PSI-2T it is around 1.10^{-3}, so the BER threshold should be lower than this to prevent any errors before we trigger any reconfigurations. We will study the linear regression function that will be used in our failure detection system and how its coefficients change during a degradation to fix a correct threshold.

The linear regression function that has been implemented is described in [13] with the regression line:

$$\hat{y} = \alpha + \beta\hat{x} \tag{4.2}$$

where $\hat{y}$ are gathered samples $(y_0, y_1, ..., y_{N-1})$ at times $\hat{x} = (x_0, x_1, ..., x_{N-1})$, α the y-intercept of the regression line and β the slope of the best-fit line. β is given by:

$$\beta = \sum_i \beta_i y_i \tag{4.3}$$

In this equation, β_i, is given by:

$$\beta_i = \frac{N \cdot x_i - \sum_k x_k}{N(\sum_k x_k^2) - (\sum_k x_k)^2} \tag{4.4}$$

with i and $k = 0, 1, 2 \ldots N-1$, where N is the number of samples used for the computation of the linear regression function. As the collection of the monitoring data of the optical transponder is triggered periodically by the RTOS, still following [13], we can write $x_k = x_0 + k$ and finally:

$$\beta_i = \frac{12 \cdot i - 6(N-1)}{N(N^2-1)} \tag{4.5}$$

With equally spaced samples we compared results from Equations 4.5, 4.4 and the Excel linear function [125]. As they give the same result we only included 4.5 in this thesis,

the easiest function to implement in our prototype. We investigate how the number of monitored samples to process influences the detection performance of the degradation. We put two PSI-2T in point-to-point communication, separated by 10km SMF fiber. In the link between the two transponders, a VOA (Variable Optical Attenuator) is placed to trigger a link fault, decreasing the received power and increasing the Pre-FEC BER in one of the transponders. We also use a Spirent Network Analyzer to generate Ethernet traffic at 100Gbps. See Figure 4.10 for a representation of the test setup and Figure 4.11 for a photograph.

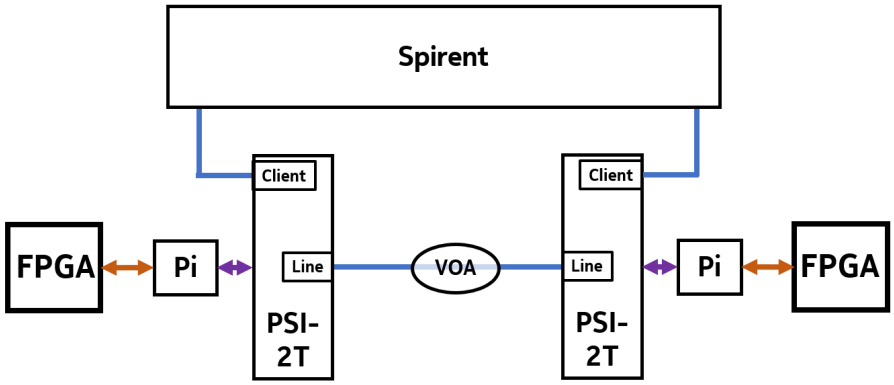


Figure 4.10 – Setup to analyze the influence of the degradation of signal quality on the value of the slope coefficient β.

We tested a fast and a slow degradation, as shown in Figure 4.12. In blue we plotted the Rx Power in the PSI-2T that will be impacted by the attenuation increase of the VOA, and we represented in bars the value of the slope coefficient β for N samples. In Figure 4.12 the VOA attenuation starts at $10dB$ and goes to $17dB$ (a) in a fraction of a second (fast degradation) and in (b) in 8 seconds (slow degradation). In Figure 4.12 (a) we can see the Rx Power dropping from $-25.14dBm$ at 6s to $-31.89dBm$ at 7s due to the attenuation increase. As the two PSI-2T are communicating via a direct link, the loss of around $7dB$ in Rx Power is in line with the attenuation going from $10dB$ to $17dB$ in the VOA. The β coefficients vary a lot depending on the number of samples used for the computation. We can note that, first, the coefficients attain their maximum at different times, at 7s for both $N = 1$ and 2, 8s for $N = 4$, 9s for $N = 5$ and 6. Adding samples for the computation delays the maximum value in the case of a fast degradation. Secondly, the higher the number of samples, the longer the β coefficient stays significantly low. For example, if we take -1 for reference as a low β coefficient, for $N = 4$, β is lower than -1

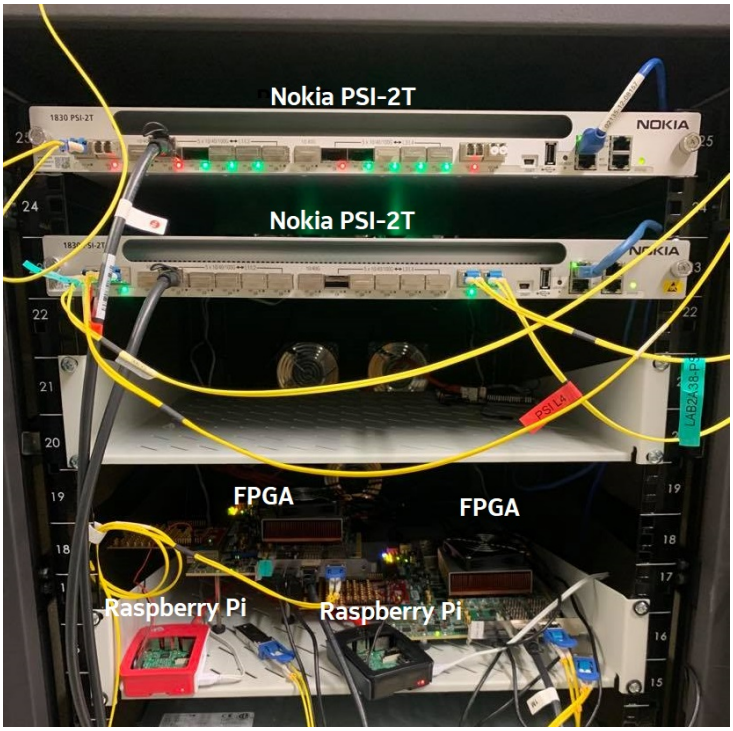


Figure 4.11 – Annotated photograph of the setup, with two FPGA boards, two PSI-2Ts and two Raspberry Pi, connected to the Spirent (not pictured).

between 7s and 9s, and for $N = 5$ between 7s and 10s. Thirdly, having a higher number of samples for the computation makes the value of β more resistant to noise, which reduces the risk to trigger false alarms. In Figure 4.12 (b) we can see the Rx Power curve starts and finishes around the same values than in Figure 4.12(a) and, as intended, the drop in power takes several seconds. A slow degradation favors smaller β values compared to (a), with only one value being close to -2 ($N = 2$ at 11s, $\beta = 1.96$). These results show the importance of the choice of samples and coefficient threshold for our monitoring workflow. As we want to control the value of the Pre-FEC BER as the Rx Power decreases, we need to choose the β coefficient and the number of samples accordingly to the type of fault that we want to detect. If we want to detect a link fault that happens brutally, such as Q-drops as described in [126], we need to have a number of samples relatively high to have a greater window of detection for the BER. For a fault that impacts more slowly the Rx Power, the number of samples is less important, but the slope coefficient should be chosen with care to ensure the detection of the impairment. Also, the value of the coefficient should be adapted to the fact that in a real network the values of Rx Power and its variations are going to be different due to the other equipment in the links.

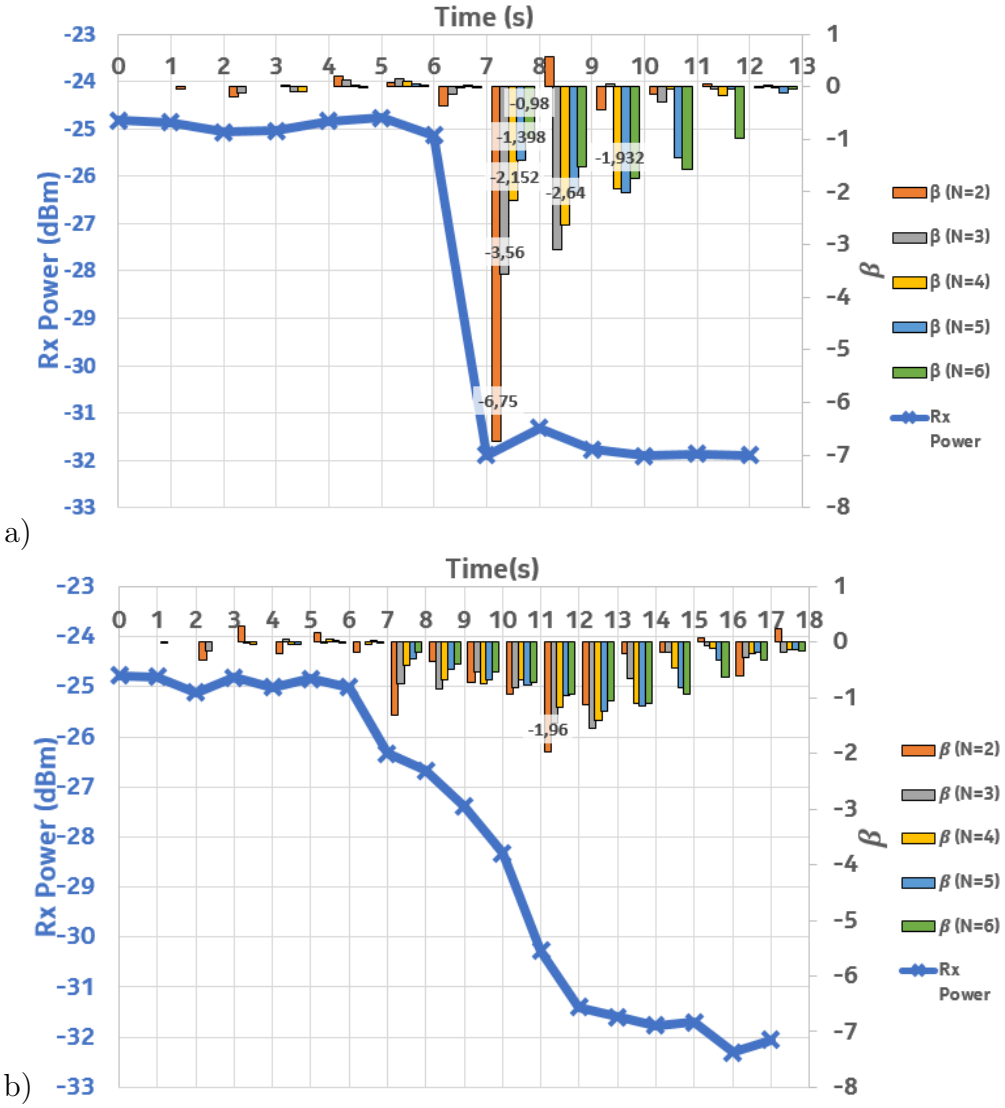


Figure 4.12 – Slope coefficient β value depending on the number of samples for the computation for a (a) fast, (b) slow Rx Power degradation.

4.5 Network Controller Interfacing

To validate our prototype of Figure 4.1, we will integrate it in a network testbed, and compare its performance against the control plane in detecting and resolving a degradation in quality of service. However we don't envision our solution to be totally independent and replace the control plane, so we need to work on the integration.

Using the Raspberry Pi that is in our platform as presented in Section 4.4, we can develop a notification solution to prevent conflicts with the control plane.

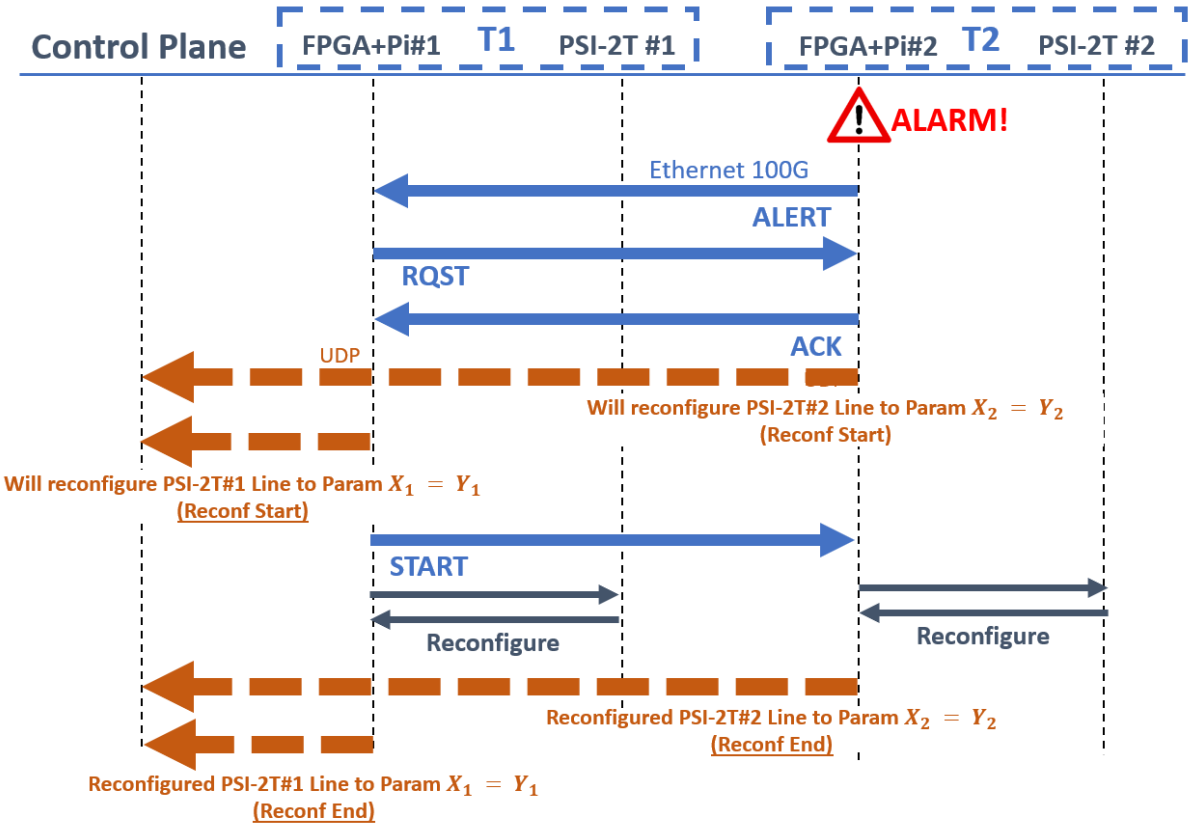


Figure 4.13 – Example of an auto-negotiation procedure between two of our prototype optical transponder (see Figure 4.15) with the integration of notifications to the control plane

The first message we implemented is a *"Reconf. Start"* message. Whenever our prototype will reconfigure itself, it sends a small UDP message to the control plane, indicating which parameter will change and to which value. When the reconfiguration is finished, a *"Reconf. End"* message is sent, confirming that the change has been done successfully. Figure 4.13 represents an example of an auto-negotiation procedure between two of our prototype transponders, with UDP notifications to the control plane implemented. These notifications in this case are sent after the acknowledgment of the new set of parameters and after the reconfiguration respectively. These messages prevent conflicts with the control plane, receiving a *"Reconf. Start"* message will prevent it from intervening and the *"Reconf. End"* allows it to take actions on the PSI-2T again and allows it to bring its information on the equipment up to date.

The other message we implemented is a mechanism that allows the control plane to

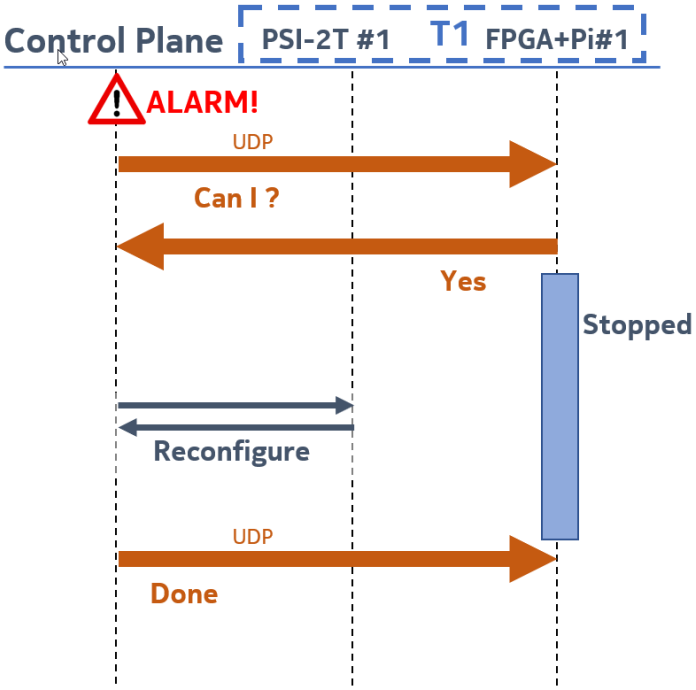


Figure 4.14 – Example of a centralized reconfiguration using the UDP message to stop our prototype from creating conflicts

stop the alarm triggering and the auto-negotiation process of our prototype transponder. This is message is useful in the case of a complex reconfiguration that can impact multiple equipment of the network and where the global view of the network from the control plane is necessary, such as changing the central frequency of the PSI-2T Line port for example. Figure 4.14 shows an example of this message exchange.

These messages will help the integration of our prototype transponder in a network with a centralized control plane, and will help us validate its performance.

4.6 Prototype transponder validation

Now that we have presented and validated the individual blocks of the prototype transponder as presented in Figure 4.1, we can present our final prototype implementation. As presented in Sections 4.3 and 4.4, we have a Virtex Ultrascale+ FPGA handling the auto-negotiation protocol and monitoring operations. The auto-negotiation logic is clocked at 322MHz, and the Microblaze at 100MHz. The Raspeberry Pi handles the communication with the commercial transponder, and as presented in Section 4.5, sends

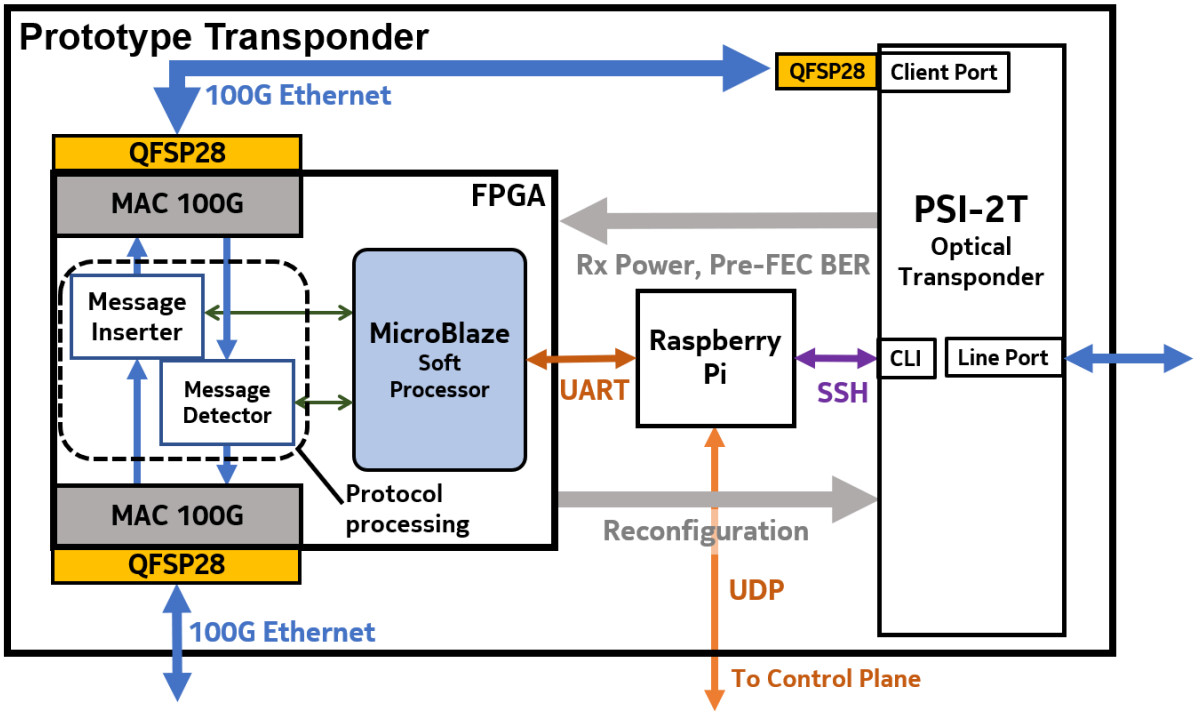


Figure 4.15 – Representation of the prototype (presented in Figure 4.1) full implementation, with FPGA, Raspberry Pi and Nokia PSI-2T commercial optical transponder.

and receives messages from the control plane. The prototype is pictured in Figure 4.15.

We integrated our real-time transponder with auto-negotiation architecture as shown in Figure 4.15 inside a network testbed for validation.

The testbed and its architecture are shown in Figure 4.16. On the data plane side, we have a meshed network with four ROADMs (R1 to R4), our two real-time prototype transponders T1 and T2 as described in Figure 4.15 and a Spirent network analyzer that generates and analyzes Ethernet traffic up to 100Gbps. A VOA allows to remotely trigger a fault in the link between R1 and R2. On the control plane, we have three main entities: the monitoring system, the ADONIS (Aggregator/Disaggregator for Optical Network equipment) open device agent and the ONOS SDN controller. ADONIS abstracts the network equipment into logical devices and exposes them to the SDN controller [37]. The Open Network Operating System (ONOS) SDN controller manages the logical devices in the network. The monitoring system receives and processes the data collected by ADONIS from the different equipment of the network, takes decisions based on collected data, and submits configuration instructions to the SDN controller. To bring stability on optical power in each optical port and have a more realistic scenario overall, we stabilize

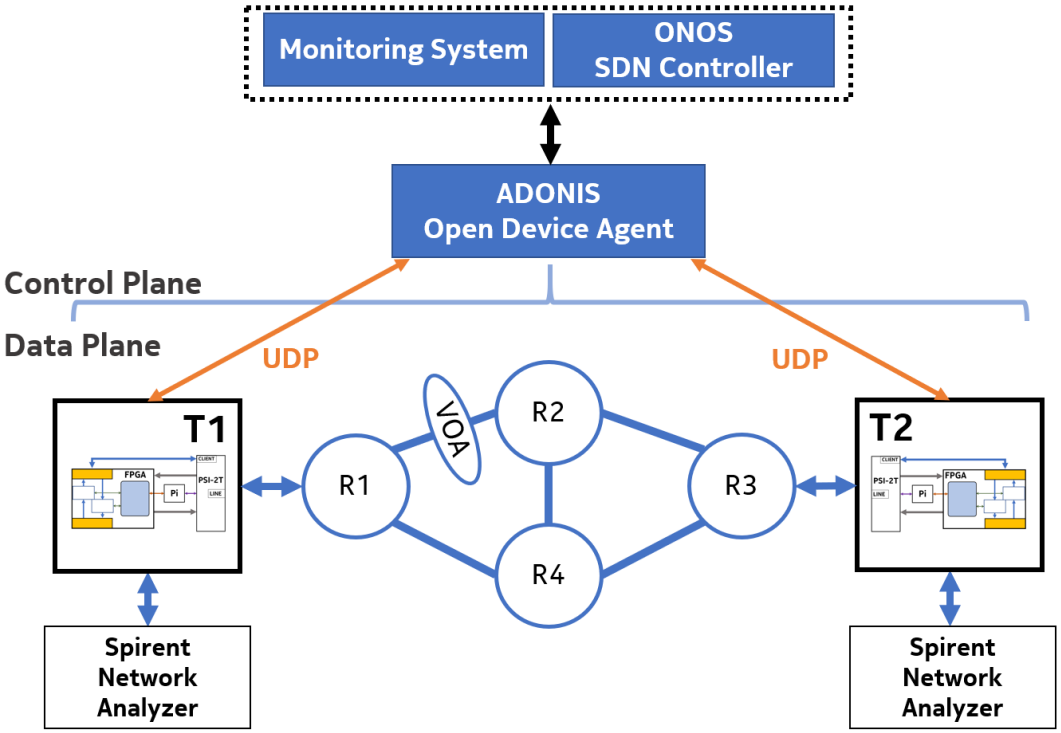


Figure 4.16 – Integration of our real-time prototype transponder from Figure 4.15 with auto-negotiation inside a network testbed

the testbed with 10 load channels generated through an ASE source and control them by means of ADONIS emulation capabilities [37].

In Figure 4.17 (a) and (b) we describe the two workflows we developed. Workflow in Figure 4.17 (a) is used when the FPGAs perform the dynamic auto-negotiation of the transponders. Every second, the Microblaze inside the FPGA sends an UART message asking the Raspberry Pi to retrieve monitoring data from the optical PSI-2T. The periodicity of the monitoring is maintained by the RTOS. When data are retrieved by the FPGA it computes the slope of the last samples of Rx Power using the linear regression algorithm described in the previous section. If the slope is below a defined value and the pre-FEC BER is above a threshold at the same time, an alarm is triggered, and the reconfiguration is prepared to restore the transmission quality to an acceptable level, using the auto-negotiation protocol described in Section 4.3. As mentioned in Section 4.4, we chose to reconfigure the Tx Power of T1, as this needs no interruption of service. Changing other parameters like the modulation format or baud rate with the used commercial transponder would stop the traffic for tens of seconds, preventing us to plot anything significant

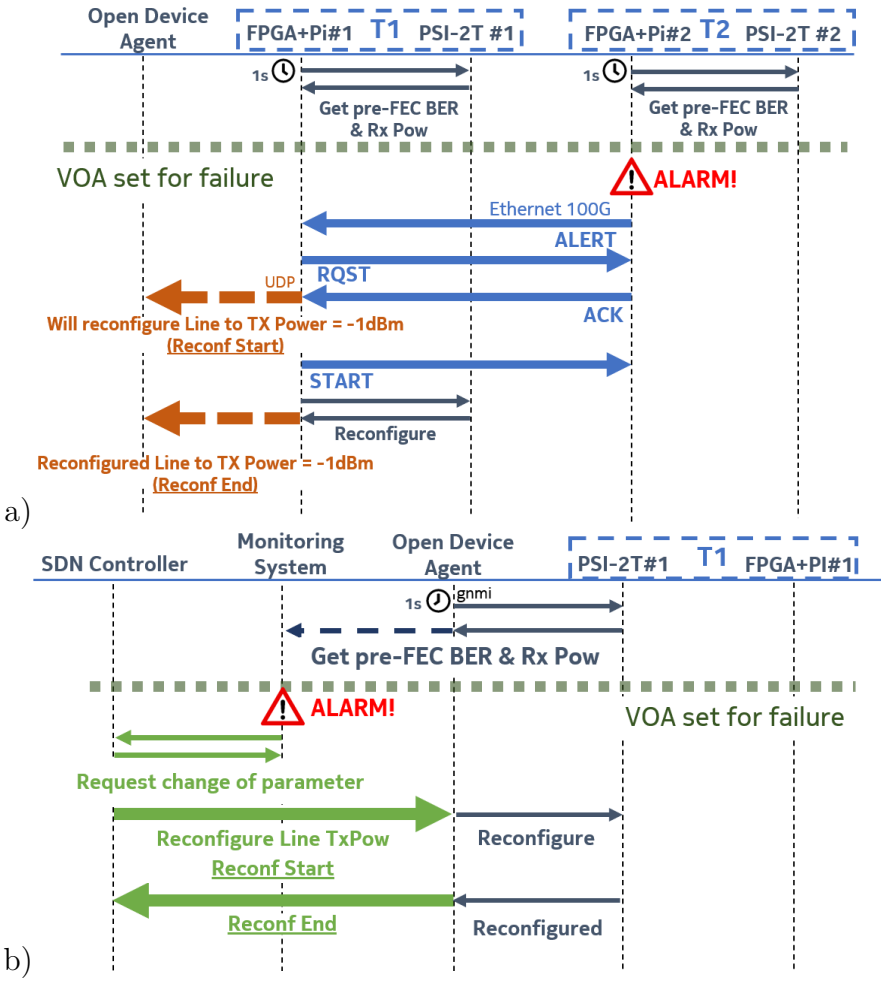


Figure 4.17 – Workflow for validation in a network testbed, with (a) FPGA-controlled setup and (b) SDN controller-controlled setup

during the process. Our solution is nonetheless compatible other parameters if these parameters are able to reconfigure rapidly, for instance in future commercial transponders. By changing the Tx power in this experiment, this also means only transponder T1 must reconfigure itself to mitigate the link fault and no adaptation has to be made in transponder T2. At the reception of the ACK message the transponder sends the "Reconf. Start" (as described in Section 4.5) UDP message to the Open Device Agent. When the reconfiguration is completed a “Reconf. End” message is sent to let the control back to the control plane and notify on the changes done to the hardware.

The workflow for the SDN-controlled reconfiguration is showed in Figure 4.17 (b). To enable comparison, the same reconfiguration algorithm as the transponder auto-negotiation

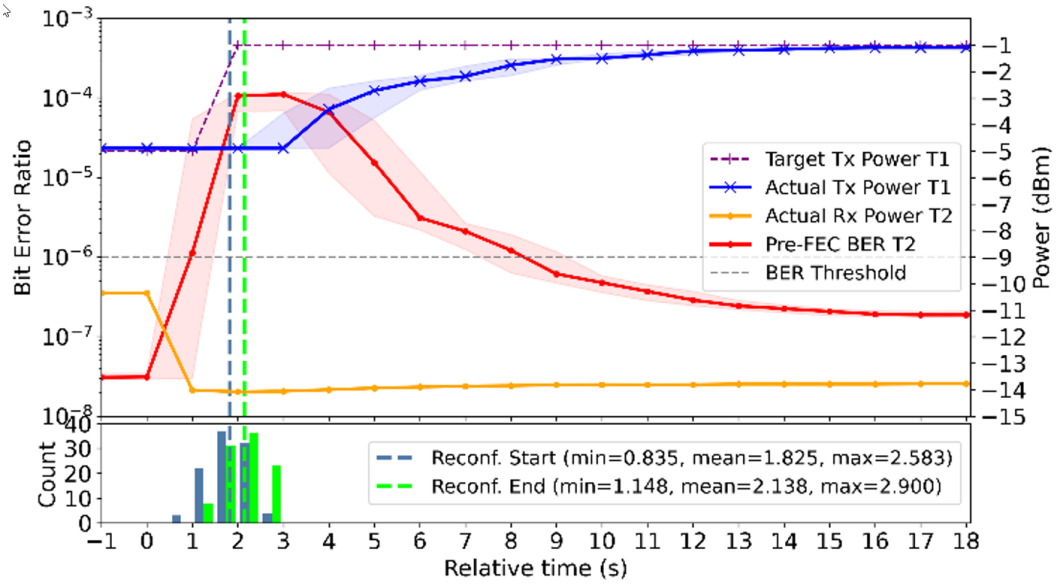
Reconf. Start	Auto-neg setup	SDN setup	Difference
min	0.835s	1.728s	0.893s
mean	1.825s	3.621s	1.796s
max	2.583s	6.535s	3.952s
Reconf. End	Auto-neg setup	SDN setup	Difference
min	1.148s	2.039s	0.891s
mean	2.138s	4.031s	1.8932s
max	2.900s	6.848s	3.948s

Table 4.2 – Minimum, average and maximum time for notification messages Reconf. Start and Reconf. End for both setups.

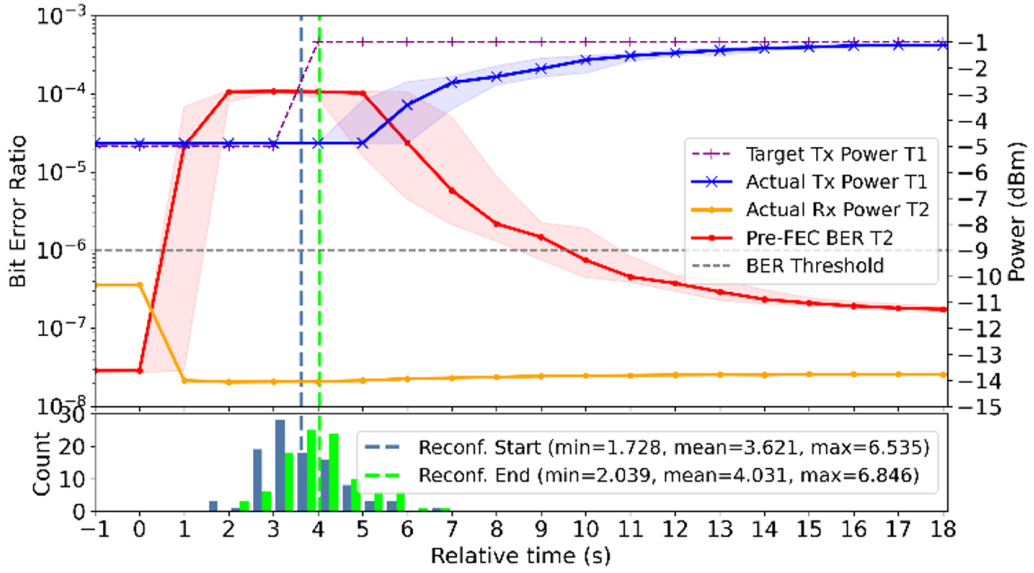
setup is used in the SDN controlled setup. Every second the open device agent gathers monitoring data from the transponders and sends them to the monitoring system to be processed. If the Rx Power slope and the Pre-FEC BER are crossing set thresholds, the monitoring system raises an alarm and asks for the SDN controller to adjust the Tx Power of the transmitter transponder. The SDN controller send the reconfiguration request to ADONIS through a standard OpenConfig [76] NETCONF edit-config message, and ADONIS replies when the PSI-2T reconfiguration is completed. We label these messages as "Reconf Start" and "Reconf End", respectively. With these two workflows, we can compare dynamic auto-negotiation with respect to centralized SDN-controlled reconfigurations in terms of latency to detect the link fault and to recover the quality of transmission by adjusting the TX Power of the transmitter transponders.

We carried out 100 repetitions for each experiment and we aggregated the results. To aggregate the values, we time-aligned the beginning of each repetition to the trigger of the link fault induced by the VOA and truncated the fractional part of the samples timestamp [1]. We configured the detection algorithm with the following parameters: the Rx Power linear regression slope β coefficient threshold is set to $-1dB.s^{-1}$ and the BER threshold is set to 10^{-6}. Fulfilling both conditions is interpreted by the algorithm as a significant power variation affecting BER that needs to be addressed. In Figure 4.18, we plotted the median values of T1 output power (blue curve), T1 target output power (violet), T2 pre-FEC BER (red) and T2 input power (orange). We also plotted the histogram of the notification messages times, with detailed results also gathered in Table 4.2. In Figure 4.18 (a), after the T2 pre-FEC BER crosses the BER threshold we can see the

1. This is made to better align the measurements of Rx Power, Pre-FEC BER, etc... for plotting. However, we kept and analyzed the full timestamp for the notification messages "Reconf. Start" and "Reconf. End".



a)



b)

Figure 4.18 – Aggregated experimental results for auto-negotiation setup (a) and SDN setup (b). At the top of each figure we plotted the median values of the monitored metrics. The confidence interval of the pre-FEC BER and output power corresponds to the 15.8 and 84.2 quantiles. At the bottom we plotted the histogram for the notification messages "Reconf. Start" and "Reconf. End". Timing results are also gathered in Table 4.2.

start of the auto-negotiation reconfiguration in dashed blue vertical line. It occurs 1.825s (mean value) after the trigger of the VOA. Right after, we can see that the mean value of T1 target output power changes, showing the effectiveness of the reconfiguration. The end of the reconfiguration, represented in dashed green line, happens at 2.138s (mean value), resulting in a mean reconfiguration duration of 313ms. Figure 4.18 (b) shows the results for the centralized reconfiguration with the SDN controller. The start of the centralized reconfiguration in dashed blue line occurs at 3.621s (mean value) and the reconfiguration ends at 4.031s, just after the change of the T1 target output power. The mean reconfiguration duration for this setup is 410ms. If we compare results obtained in Figure 4.18 (a) and (b), we observed that the auto-negotiation reconfigurations start 2 times faster, and finish 1.3 times faster, with respect to the centralized SDN reconfigurations. Also, the histograms show the variability of the experiments. We observed that the auto-negotiation reconfiguration exhibits much less variability of both start and end reconfiguration messages. For instance, the difference between min and max values equals 1.75s in Figure 4.18 (a) versus 4.81s in (b) for the end reconfiguration messages. As expected, auto-negotiation reconfigurations accelerate the decision-making process and act faster in the physical layer. These results are in the context of a network testbed in a lab, but gains should be noticeable in real-life situation. Even if the transponders would be at a greater distance from each other increasing the round trip time between the auto-negotiation processing logic, the control plane would also be at a greater distance from the network hardware and have a larger number of equipment to monitor and functions to process. Besides, having a shorter distance is more challenging in our solution as the protocol processing needs to react faster as the message exchange is quicker. We believe that future fast reconfigurable devices would take benefit of our solution by reducing the time needed by the hardware to recover to acceptable quality of service values and bring forward its hitless behavior.

In the end, the resources taken by the whole design of the system on the FPGA is detailed in Table 4.3 and in Figure 4.19. The resource utilization is above 10% for the block RAM (BRAM, 19.81%), the inputs outputs (IO, 18.03%) and transceivers (GT, 15.38%) and usage is at 5.17% overall. We can conclude that our design takes only a fraction of the available resources in our die. So, in the future, we can consider adding more functions in the logic for more elaborate monitoring functions and integrating the functions of the Raspberry Pi in the FPGA design.

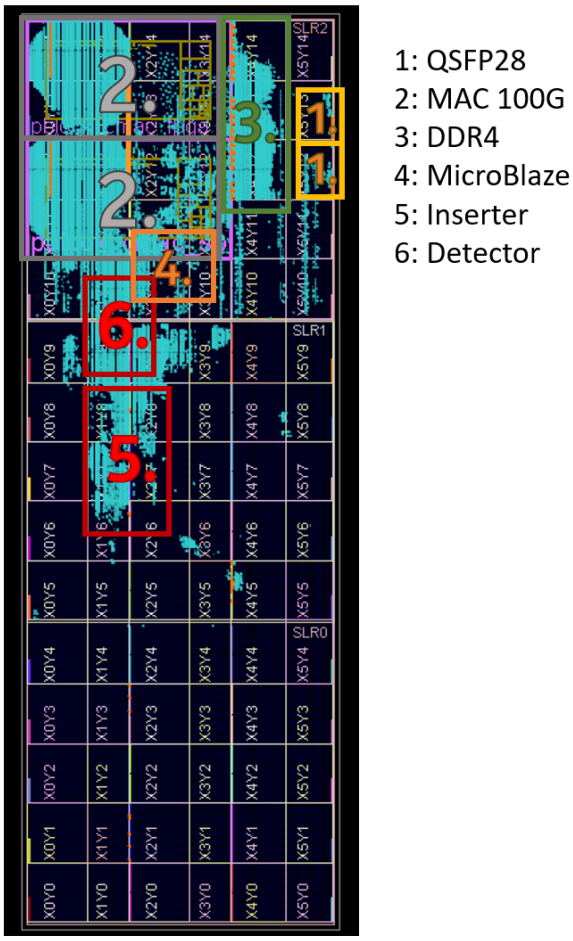


Figure 4.19 – Representation of the resource utilization of our design in the FPGA die

Resource	Utilization	Available	Utilization %
LUT	68371	1182240	5.78
LUTRAM	7424	591840	1.25
FF	138201	2364480	5.84
BRAM	428	2160	19.81
URAM	90	960	9.38
DSP	24	6840	0.35
IO	150	832	18.03
GT	8	52	15.38
BUFG	16	1800	0.89
MMCM	2	30	6.67
PLL	5	60	8.33

Table 4.3 – Resource Utilization Report for our FPGA design

4.7 Integration with a baud-rate variable setup

We have validated our prototype transponder with auto-negotiation for synchronous change of optical parameters with commercial equipment, but we can also show its relevance for future hitless transponders and demonstrate a more elaborate synchronous change of parameters rather than a change of transmission power.

Changing the baud-rate in an optical network can be beneficial for multiple reasons. One of them is the possibility to reduce the spectral occupancy of the signal, which is interesting in case of a degradation that requires to send the traffic down another optical path, that may have less spectral slots available for example, or more filtering for example. Optical defragmentation, i.e. the operation of reorganizing and optimizing optical spectra and routes to reduce the overall spectral occupancy of the network and reduce the blocking probability of future requests, may also need to change the baudrate of equipment. For this we are using real-time transmitter and receiver, that allow for hitless change of baud-rate and no reception error after the change. The setup used in this section is presented in Figure 4.20.

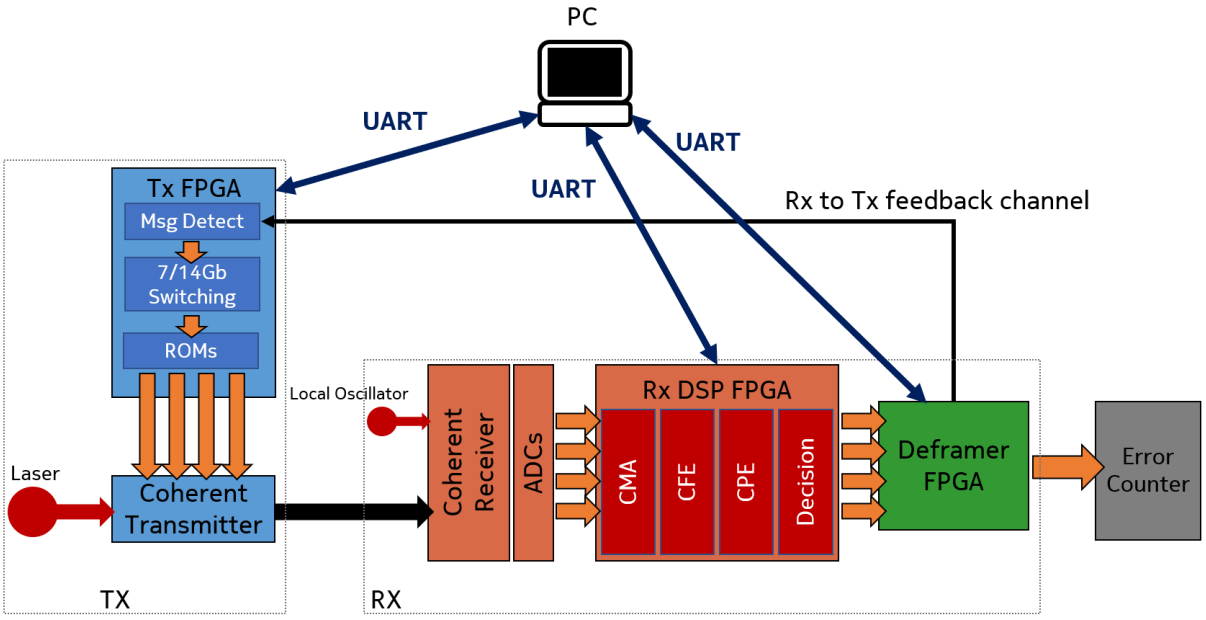


Figure 4.20 – Experimental setup for the synchronous hitless change of baud-rate using the developed protocol. It is composed of a coherent transmitter (Tx) and a coherent receiver (Rx). A feedback channel is setup between the Rx and the Tx for bidirectional auto-negotiation protocol operations. All the FPGA boards in the setup are managed using an UART link that is connected in our experiment to a computer.

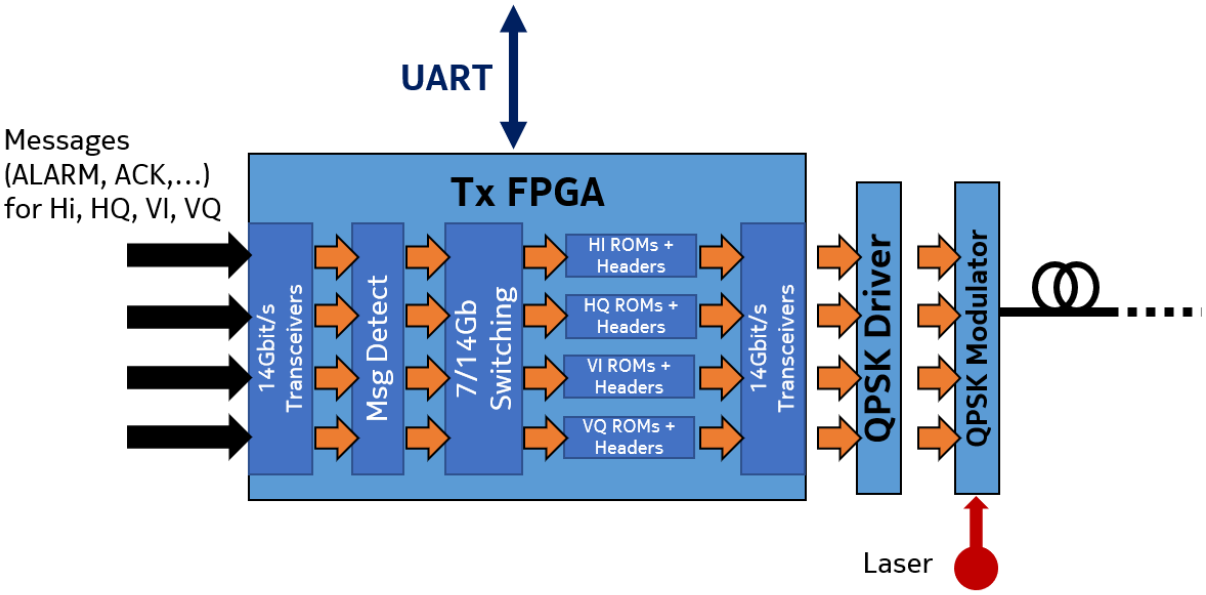


Figure 4.21 – Representation of the variable baud-rate transmitter

This setup is comprised of a real-time coherent transmitter, a real-time coherent receiver and a deframer FPGA board which is an extension of the receiver that will process the protocol messages in the receiver. We will now describe more thoroughly each part of this setup, starting with the transmitter.

4.7.1 Transmitter

The transmitter used for this experiment is represented in Figure 4.21 and photographed in Figure 4.22. It is based on the prototype transceiver presented in [113], and used in a defragmentation scenario in [127]. It is implemented on an Xilinx Virtex-7 FPGA, and is able to transmit a signal at either 14GBaud or 7GBaud using a single clock rate in the FPGA fabric, and to perform this change of data rate hitlessly. For both rates, the data sent by the Tx is 8192-bit frames composed of a 64-bit header followed by a PRBS-7 [2] (Pseudo Random Binary Sequence) with a word length of 64 bits. The PRBS is stored in Read-Only Memorys (ROMs) and each polarization and component of the modulation has its own. The Horizontal polarization will be noted as H, Vertical as V, the In-phase component as I and Quadrature as Q. This gives us eight PRBS to store in ROMs: one for for HI, HQ, VI and VQ each, and one for each used baud-rate

2. Which has a period of $2^7 - 1$ words i.e. 127 words

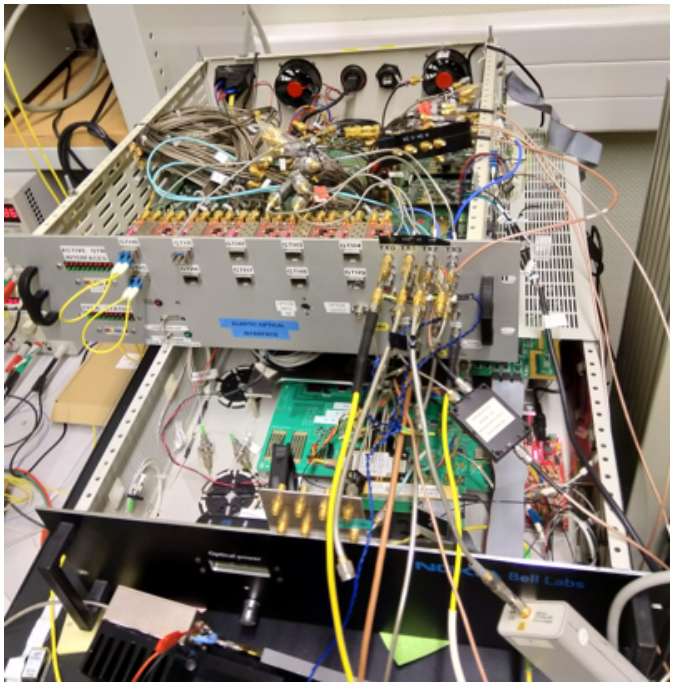


Figure 4.22 – Photograph of the variable baud-rate transmitter

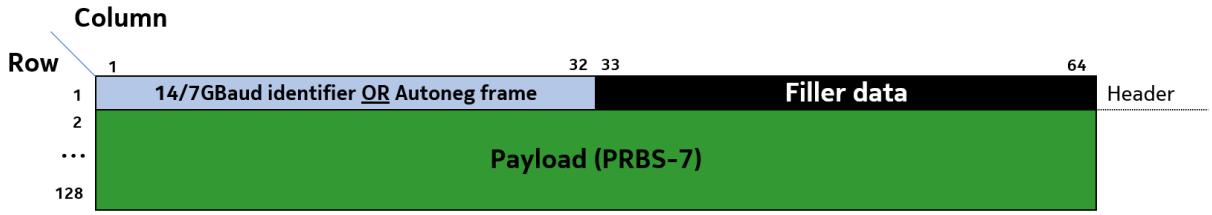


Figure 4.23 – Frame structure sent by the transmitter for this experiment.

(14GBaud and 7GBaud). The header is indicating that the transmitter is working at 7 or 14GBaud, or contains the auto-negotiation protocol message from the transmitter. The frame structure sent by the Tx is represented in Figure 4.23.

A specific channel is set-up between the receiver and the transmitter so that the receiver can send the ALARM and ACK messages to the Tx and perform the auto-negotiation. A logical module is handling the reception of the messages for all components of the QPSK signal (HI, HQ, VI, VQ) and triggers the change of baud-rate in the FPGA logic. In Figure 4.24 you can see the eye diagrams of a clock signal transmitted by the variable baud-rate Tx at 14 and 7GBaud and in Figure 4.25 you can see the constellation of the transmitted signal.

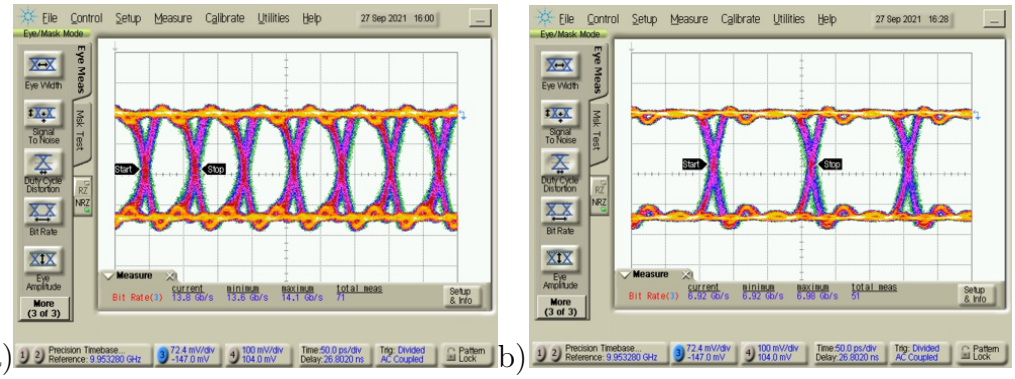


Figure 4.24 – Eye diagram of the optical signal sent by the transmitter at (a) 14GBaud and (b) 7GBaud.

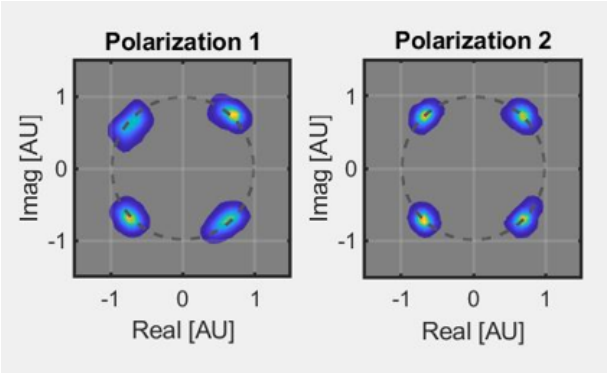


Figure 4.25 – Constellation of the PM-QPSK signal, for the Horizontal and Vertical polarizations, at 14GBaud.

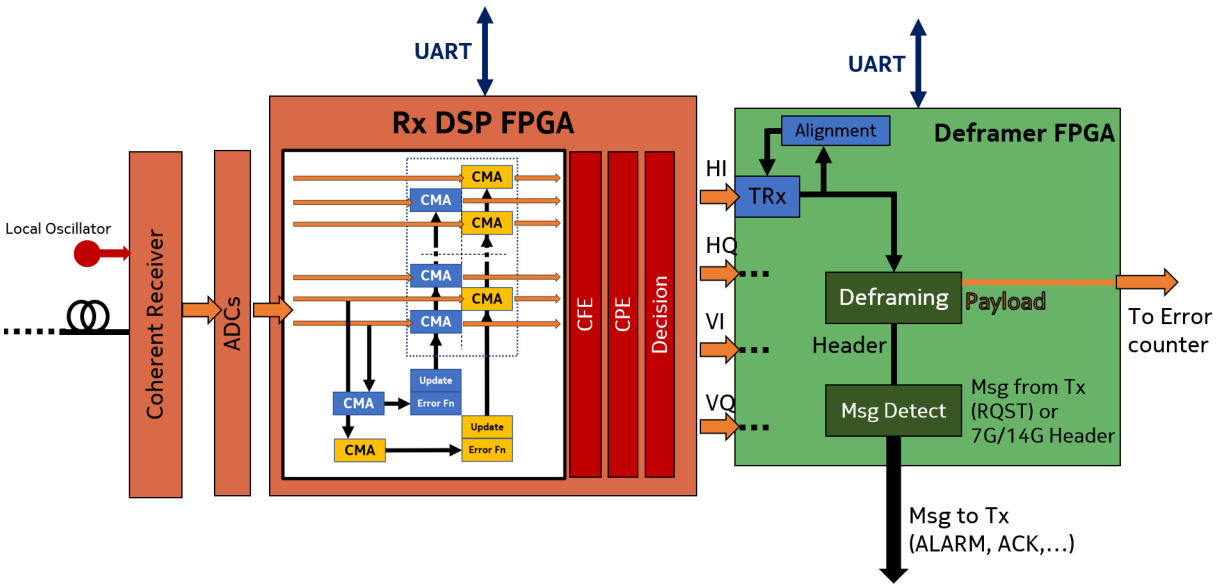


Figure 4.26 – Full representation of the Receiver with the coherent receiver and the ADC, the DSP FPGA and the deframer FPGA board.

4.7.2 Receiver and deframer board

The receiving element of the setup is separated in two main parts: the receiver that recovers the symbols at both 7 and 14GBaud, and a deframer board that separates the header from the payload of the frames and handles the auto-negotiation protocol operations. It is represented in Figure 4.26.

Receiver DSP The signal recovered by the coherent receiver and sampled from the Analog-to-Digital Converters (ADCs) goes to an FPGA with a coherent DSP implemented, see Figure 4.26. As explained in 2.2.3, to properly decode the symbols the signal needs to be processed by algorithms, that recover its amplitude its frequency and its phase in our case. In this work, the particular implementation of the Constant Modulus Algorithm (CMA) algorithm is allowing the system to function at both 7 and 14GBaud, and will be described in the following paragraphs. We will not go over the description of the CPE and CFE algorithms, that have been introduced in 2.2.3 and which you can find more information in [128].

In Figure 4.27 (a) we represented a traditional implementation of the CMA algorithm as a Finite Impulse Response (FIR) filter structure, processing samples of the received

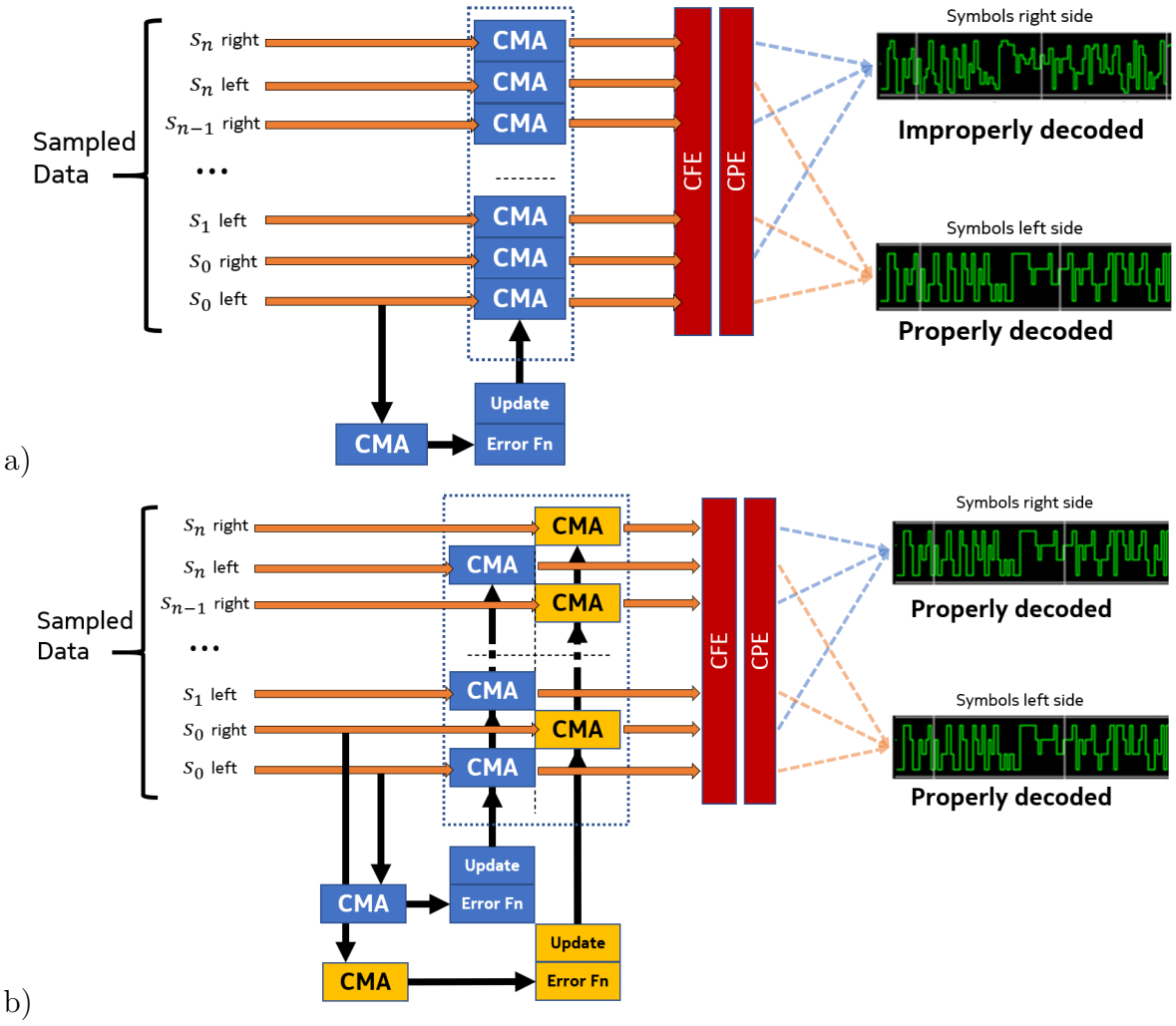


Figure 4.27 – Structure (a) of a traditional FIR-CMA architecture and (b) the used architecture for the real-time receiver of this experiment. On the right side of both figures, s_0 is given as an example of a recovered symbol at half baud-rate.

data from the ADC. As can be seen in the figure, the FIR-CMA is decomposed into N-parallelized FIR-CMA filters, with N being the number of symbols processed by the filter at the same time. In the Figure, s_k left refers to the first two samples of the k^{th} symbol, and s_k right the third and fourth samples, the incoming signal is at 2 Samples Per Symbol (SPS). The goal of the CMA algorithm is, for modulation formats employing a single amplitude for their constellation (as seen in Figure 4.25, it is applicable for PM-QPSK) to recover said amplitude before phase and carrier recovery (see Figure 2.22).
As a reminder, the output of a FIR-filter can be described as

$$y[k] = h^T s = \sum_{i=0}^{N-1} h[i]s[k-i] \tag{4.6}$$

Where $y[n]$ is the output signal out of the filter, $s[k]$ the received symbols, $s = (s[k], s(k-1), ..., x(k-(N-1))^T$, $h[i]$ the tap weights of the filter and $h^T = (h[0], h[1], ..., h[N-1])$. For the CMA algorithm, the cost function to minimize is:

$$\epsilon = 1 - |h^T s|^2 \tag{4.7}$$

Which corresponds to searching for filter taps that make the output signal amplitude equal to one. For the CMA algorithm, the gradient of the cost function is estimated stochastically and tap weights are updated in accordance, following the steepest descent method (see [129] for more details). This is represented in the bottom of Figure 4.27 (a), where a sample is passed through a FIR-CMA, and an error function estimates the update to apply to the taps of the filters of the other FIR-CMA filters. This gives the update algorithm for the filter taps:

$$h := h + 2\alpha\epsilon s^*(h^T s) \tag{4.8}$$

With α being the step size, s^* the conjugate of s, and $:=$ being the assignment operation.

In Figure 4.27 (b), we pictured the CMA architecture of the receiver used for these experiments. This structure is different as the one in (a) and allows the processing of the data at two baud-rates: nominal and half (for the experiments 14 and 7GBaud respectively). Rather than using a single update structure, two are used, one for the left part of the symbols and one for the right part. When the (b) implementation will recover the symbols correctly, the (a) implementation will struggle recovering both parts of a symbol at divided baud-rate, due to the fact that at half baud-rate the incoming signal will by at 4 SPS. This principle can be extended for 4, 8, etc... times divided baud-rates. The

proposed CMA implementation is detailed more thoroughly in [130].

Deframer board After being decoded by the DSP FPGA boards, the symbols go to an FPGA deframer board, that will handle the auto-negotiation protocol (for HI, HQ, VI and VQ). The deframer board is represented in the right side of Figure 4.26, and is the part of the experiment that has been under my supervision.

The first role of the deframer board is to separate the header and the payload from the incoming traffic. At first this has been implemented by directly detecting the headers. However, the auto-negotiation protocol should be able to work even at low performance, because otherwise it is unable to trigger a change of parameters to recover from a degradation. Furthermore for this experiment no FEC code is used, so an altered bit cannot be recovered by such mean. We decided to rather use correlation to detect the header in the incoming traffic. The traffic is compared to expected bits of the header, and if the number of common bits between the incoming data and the expected header is about a programmable confidence level the header is extracted and processed accordingly. This is a more taxing operation than direct detection, which can be realized in a single clock cycle. By doing the operation $NOT(data_{in}\ XOR\ header_{tocompare})$ the resulting bit vector will be the bits that are in common between the two vectors, which can be counted and compared against the programmable confidence level. The counting operation is the one that takes the most time for the logic, the result of the correlation operation (i.e. the detection or not of the header) can be obtained in 2 clock cycles. We use an additional clock cycle to extract the header as can be seen in Figure 4.28 to ease the synthesis of the design complexity-wise. In Figure 4.29, the detection and removal of a RQST message for the auto-negotiation protocol is captured. We also showed the answer sent on a separate channel after the reception of the RQST message. This answer is sent to the Tx via the feedback channel. As can be seen in the figure, the answer is delayed to be sent when the next frame is received, mostly to simulate propagation time.

After the header extraction, the payload is sent to an error counter, that will count the number of incorrect bits in the PRBS. If a auto-negotiation protocol message is detected in the header, it is processed and if necessary an answer is sent in the feedback channel between the Rx and the Tx.
Additionally, it is extremely important that the frame is arriving in the FPGA properly aligned. A logical module is handling this operation, by sending a command the the transceiver (TRx in Figure 4.26) used for the communication between the Rx DSP FPGA

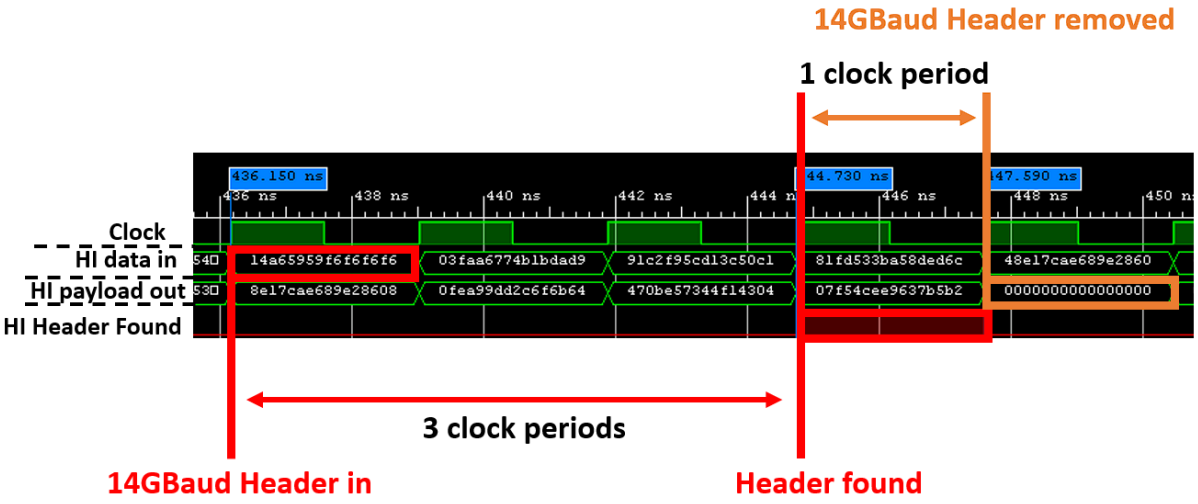


Figure 4.28 – Capture of header detection and removal by the deframing logic in simulation. The header in this example is the header indicating 14GBaud transmission. The clock period is 2.86ns (350MHz).

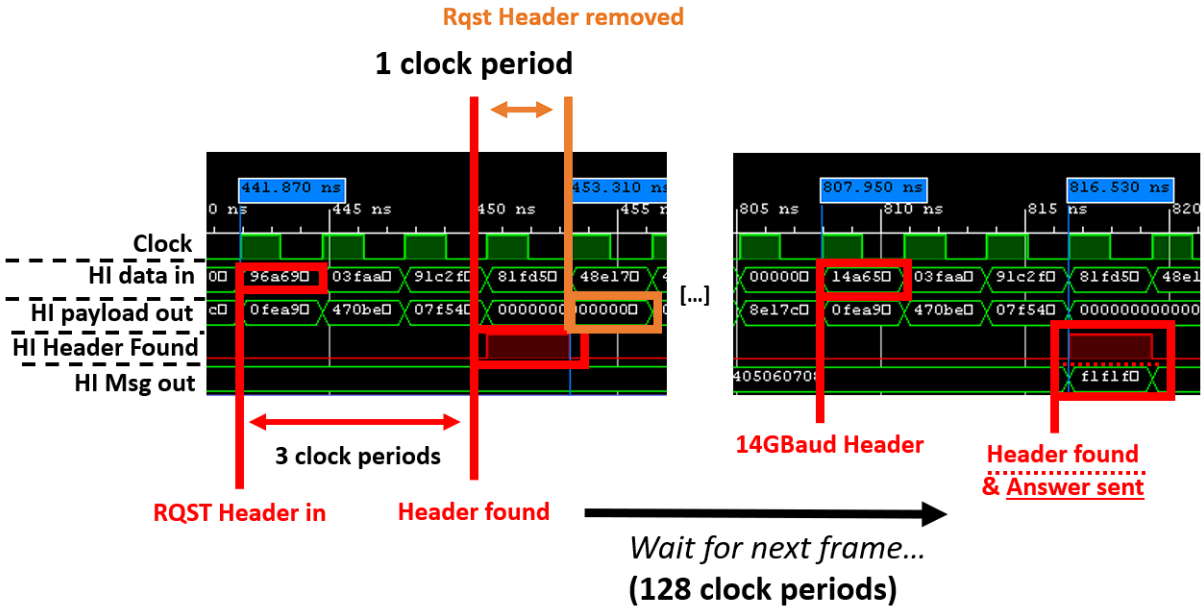


Figure 4.29 – Capture of RQST message header detection and removal by the deframing logic in simulation. After the message is detected, we delay the answer until the next frame is received.

and the deframer. This command delays (or "slides") the received data vector out of the transceiver by one bit, and this can be used until the data vector is properly aligned. This is either handled automatically by the logical core (sending the slide command until it is able to recognize the header in the traffic) or manually by the user using the UART link to trigger the command.
In Figure 4.30 you can see a photograph of the Rx part of the setup with the coherent receiver, the ADC, the FPGA DSP and the deframer FPGA board.

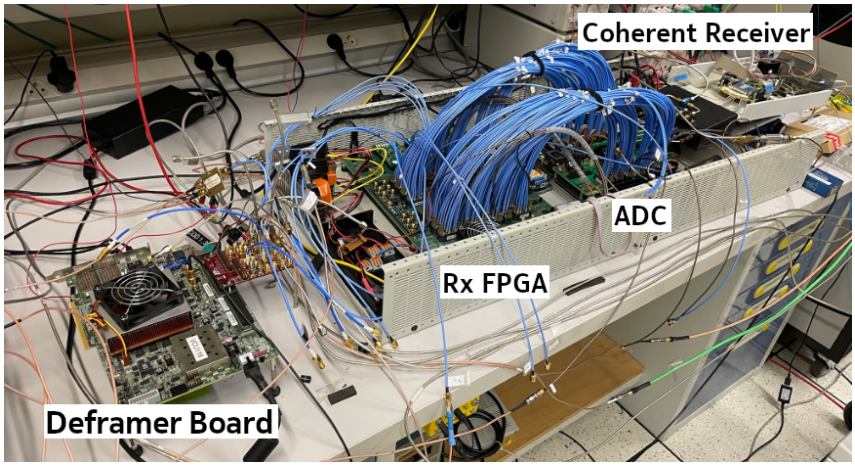


Figure 4.30 – Annotated photograph of the variable baud-rate receiver

4.7.3 Validation

To validate our setup we will trigger a change of baud rate, synchronized thanks to the auto-negotiation protocol, and show that it results in no interruption of traffic nor any additional error in the recovered payload.

The validation setup is as presented in Figure 4.20. The Receiver DSP and the Deframer FPGA boards are clocked by a 350MHz clock generated by the ADCs, and the transmitter is clocked at 437.5MHz.

To pilot the setup, human-machine interfaces were developed on Labview to access easily the UART-accessible parameters on the different FPGA boards of the setup. These interfaces are also the way we will trigger the auto-negotiation process. In Figure 4.31 we pictured the interface for the deframer board. Different parameters of the board transceivers can be edited to improve communication between the boards, the frame alignment procedure can be triggered, correlation thresholds can be changed for the detection of the header, and the ALERT message can be sent to start the auto-negotiation

procedure. For this experiment, as boards are separated by only a few meters of fiber (between the Transmitter and Receiver) or coaxial cables (between the Receiver and the Deframer and between the Deframer and the Transmitter), we can expect that we will not have any errors in reception even without FEC, and set the thresholds for the detection of the headers in the Deframer board at maximum (every 32 bits of the headers need to be detected).

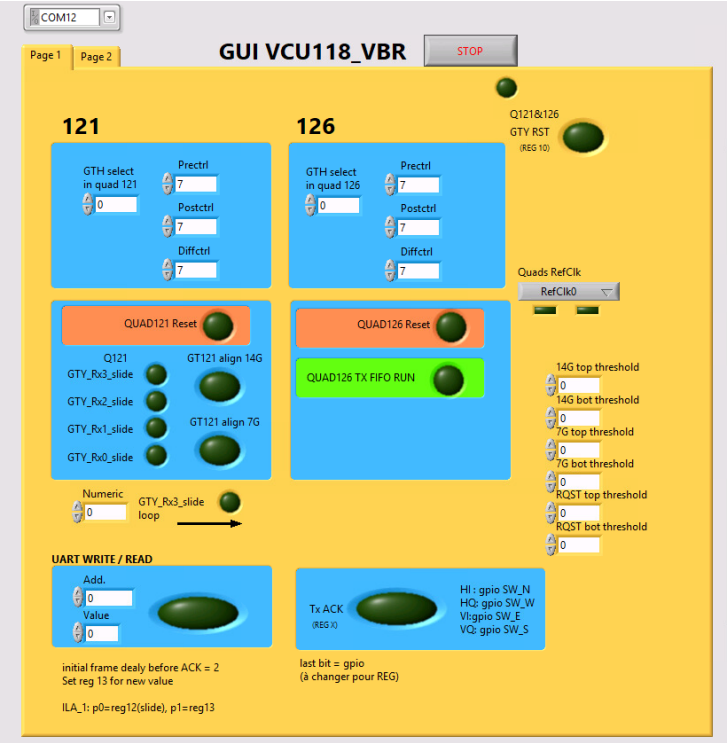


Figure 4.31 – Labview developed interface to pilot the deframer board, connected via UART.

In Figures 4.32, 4.33 and 4.34 we show captures from the Deframer board, Transmitter and Receiver respectively showcasing a synchronized change of baudrate. The captures are made after an initial frame alignment procedure. The captures from the Deframer and Transmitter showcase a change from 14 to 7GBaud, and the capture from Receiver showing a change from 7 to 14GBaud, but the change of baudrate can be done in either way and does not invalidate our other results. In Figures 4.32 and 4.33 we can see the exchange of messages between the Deframer board and the Transmitter. The auto-negotiation process starts when the ALERT message is sent by the Deframer FPGA board and received by the Transmitter. The Transmitter then sends the RQST message indicating that he is going

to change its baudrate from 14GBaud to 7GBaud. When it is received by the Deframer, it sends its ACK acknowledgement message to confirm the change of parameter. The next frame, the 7GBaud traffic starts. As can be seen in Figure 4.34, the change of baudrate (in the case pictured, from half-rate to full-rate, but is valid for the other way) is received with no visible error, and the the 7G header is detected and removed by the Deframer board with no problem. The error counter validates the errorless change of baudrate, by detecting no error before, during and after the change in the payload, for changes from full to half baudrate and vice versa. The change is negotiated and performed in three frames from the transmitter (counting from the reception of the ALERT message to the first 7G frame), which is the minimum delay we would be able to achieve. This is partly due to the fact that all parts of our setup are close to each other but this is encouraging for future experimentation, for example in a full network testbed.

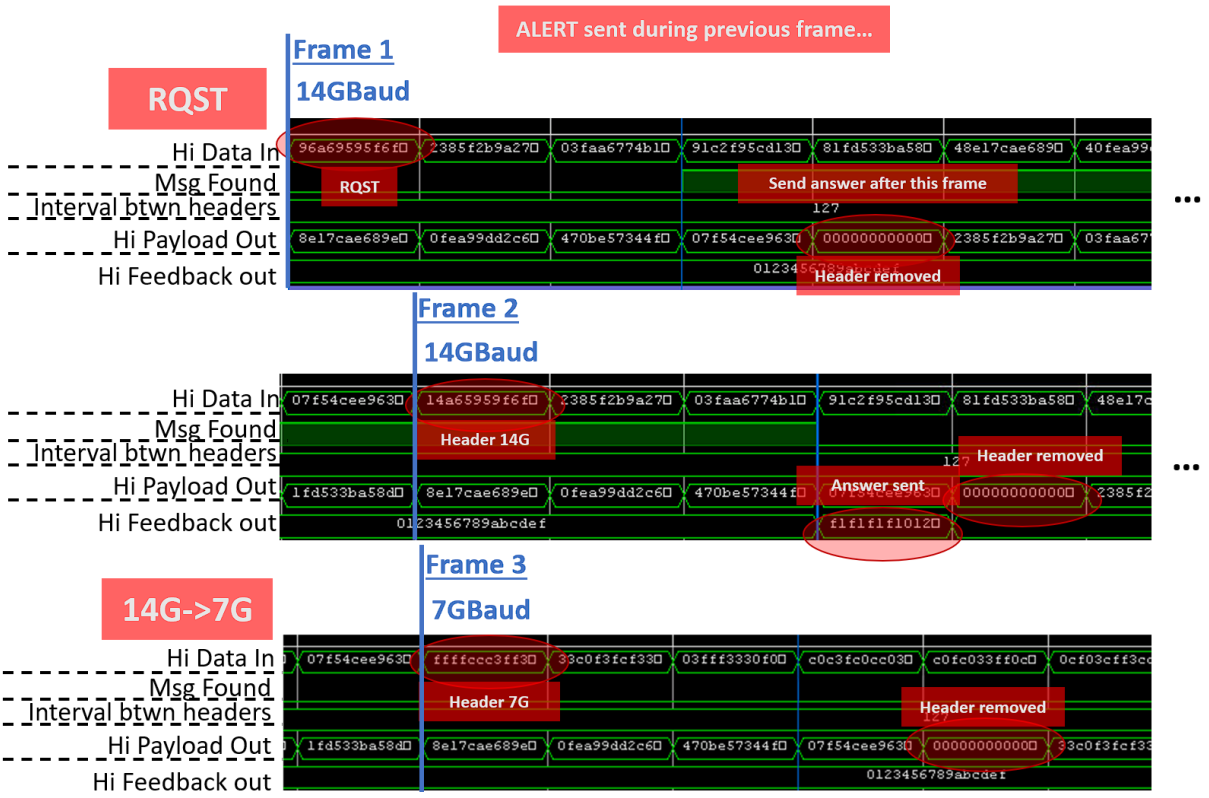


Figure 4.32 – Capture from the Deframer FPGA board showcasing a change of baudrate from 14GBaud to 7GBaud with the auto-negotiation protocol. The ALERT message was sent before the first pictured frame.

We have validated the hitless change of baudrate of this setup, which shows promises for the future of reconfigurable optical transponder. This setup could be expanded upon,

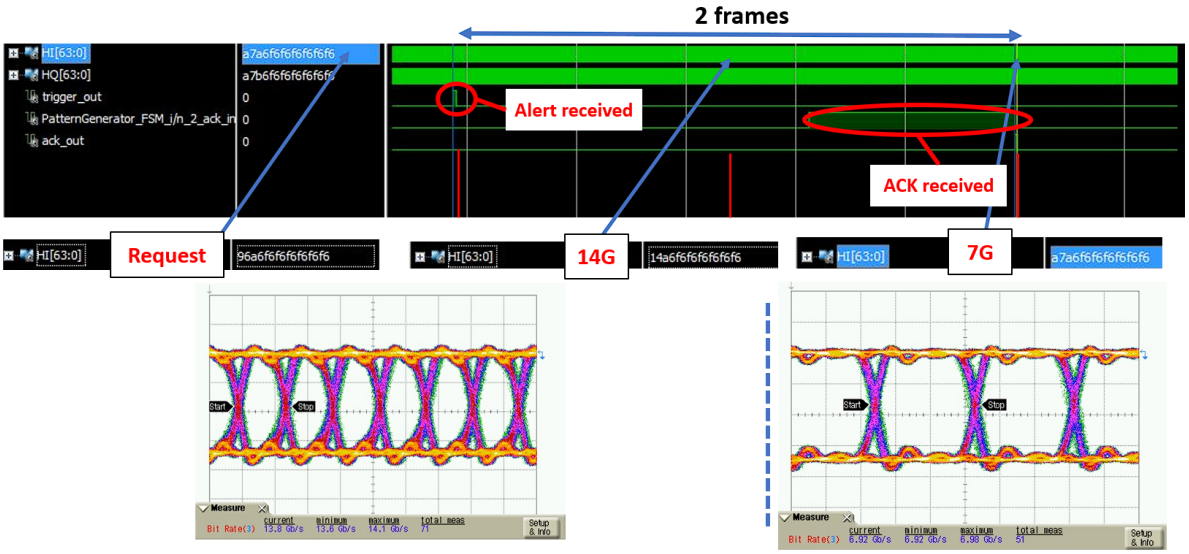


Figure 4.33 – Capture from the Transmitter showcasing a change of baudrate from 14GBaud to 7GBaud synchronized with the auto-negotiation protocol. Below the captures, we inserted oscilloscope captures of the eye diagram before and after the change of baudrate.

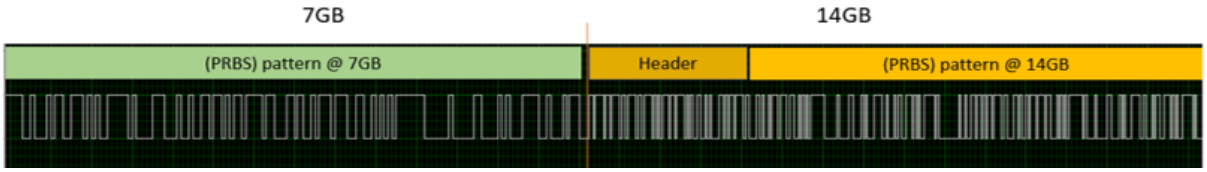


Figure 4.34 – Capture from the Receiver showcasing a change of baudrate from 7GBaud to 14GBaud.

introducing monitoring capabilities, presenting additional scenarios for reconfiguration of optical parameters such as the laser wavelength for example, ideally in a network testbed, and showcase the potential of the auto-negotiation protocol for reconfigurable optical transponder.

4.8 Conclusion

We presented and validated a prototype of a real-time optical transponder with dynamic auto-negotiations triggered by events extracted from optical performance monitoring to quickly restore drops in quality of transmission. We validated our FPGA implementation of the auto-negotiation protocol at 100Gbps, showing that our protocol processing

only adds 12ns in latency, and successfully demonstrated that our solution detects and resolves link faults two times faster, reconfigures the hardware in 1.3x less time and with less time variability than a centralized SDN-based control plane with similar monitoring and reconfiguration capabilities, showing the advantages of adding our dynamic auto-negotiation solution in optical transponders. We also showcased our auto-negotiation protocol with a real-time fast baudrate reconfigurable setup, which is a promising solution for the future of optical transponders. We can build on our solution by providing more elaborate monitoring workflow, which would provide more intelligence to the optical device, triggering smarter automatic reconfigurations, and work towards full automatic and reconfigurable optical networks. Our auto-negotiation solution implementation left room in the FPGA fabric so we can develop on the solution itself and implement logic-based monitoring functions to have a better knowledge of the network and reconfigure the hardware more intelligently.

Chapter 5

Transponder and acceleration services

The adoption of the Software Defined Network (SDN) paradigm in optical network has changed how networks are operated. Thanks to intensive monitoring of the optical layer, the state of the network can be known at each instant or be analyzed over long period of times to detect unseen phenomena and perform post-failure analysis [38]. Both of these analyses help perform optimization of the network overall or optimize the quality of service of a single optical link if desired or necessary. This intense monitoring also helps resolving or prevent altogether interruptions of traffic by reconfiguring part of the optical data plane, increasing the network reliability. This has led the path toward fully automated and elastic optical networks.

Novel monitoring techniques have been developed for modern optical networks to take advantage of the greater availability of monitoring data in the data plane, the increase in computational power in the control plane and its orchestration ability. For example, machine learning techniques (such as Artificial Neural Networks) have gained traction in recent years, as they are a good set of tools to answer the ever growing complexity of optical networks [39], notably in the context of Elastic Optical Networks. They are able to exploit the great diversity of available optical monitoring parameters and allow for modelization of complex phenomena, such as the nonlinear effects [15], fiber bending [131], etc... It is expected that machine learning will heavily contribute to the full automation of optical transport networks [14].

However, while it is true that optical network automation is becoming a reality, several factors are undermining these advancements. First of all, while there is a great number of different metrics to exploit to measure a great number of different phenomena happening in the network, the more data we collect from the data plane and send to the control plane, the more the centralized control plane will be throttled down. Telemetry has been developed to both decrease data collection time using streaming techniques and allows

the possibility to subscribe and unsubscribe from monitoring streams depending on the current needs [5]. It is a currently popular solution, however it does not resolve all problems stated earlier. These are partly due to the centralization of the monitoring and decision-making processes, and novel architectures propose to alleviate this by providing closer to the device intelligence [10], which can reduce both the quantity of data that has to be transferred to the control plane and reduces the latency of the network by reducing the workload of the control plane.

In this section we will present a solution for Artificial Neural Network (ANN) deployment inside a Field Programmable Gate Array (FPGA), both to enable elaborate monitoring techniques for the optical data plane and for acceleration of machine neural-based functions of the optical control plane. We will firstly introduce the concept of ANNs and the neural network-based nonlinearity monitoring function that we will implement to validate the logical neural structure we will develop. This logical structure will be validated in simulation in terms of precision and timing before we will expand on a weight update logic to allow for reconfiguration of the implemented function. The setup will be finally validated at implementation stage.

5.1 Neural network-based power monitoring

5.1.1 Generalities on neural network algorithms

A neural network, or more precisely in our context an Artificial Neural Network (ANN), is a set of mathematical algorithms which takes inspiration on how data is processed by the neurons of a biological brain. It comprises neurons, which are structures that take one or multiple inputs, process them by applying weights, pass the sum of weighted inputs plus a bias into an activation function, and output a single value that can be sent to other neurons further in the network. Neurons are organized in layers, the input layer, the hidden layer (that can contain multiple layers) and the output layer. This basic structure is represented in Figure 5.1, and the layout of a neuron is represented in Figure 5.2.

An ANN is a practical way to approximate a complex algorithm [132] or perform efficient pattern recognition [133], provided the network is trained appropriately. There are several possible ways to train a network, depending on the desired usage of the ANN. For example, one can train a network to do regression, pattern recognition etc... by providing inputs with their desired outputs and train the network to generalize for future unseen

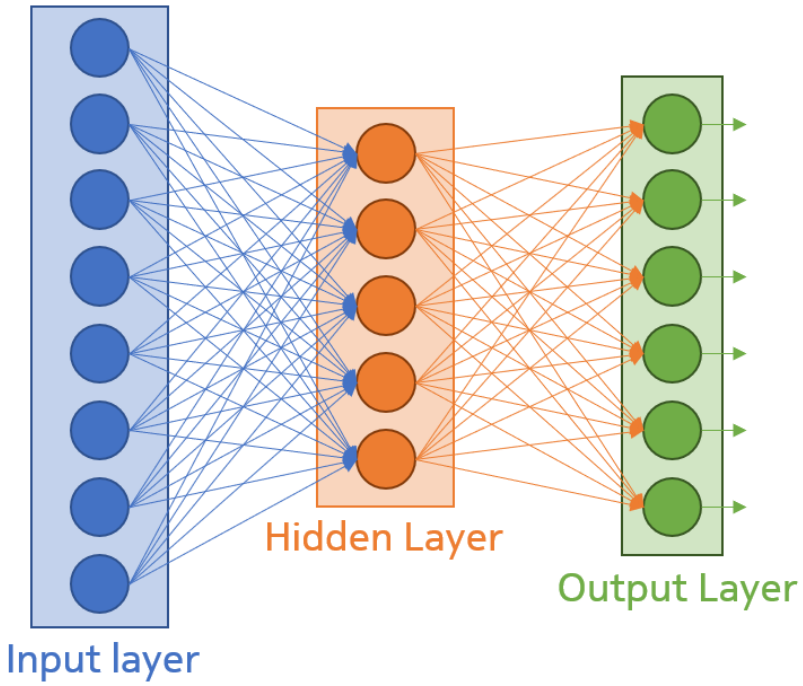


Figure 5.1 – Basic structure of an Artificial Neural Network with a single layer of five neurons in the hidden layer, eight inputs and six outputs.

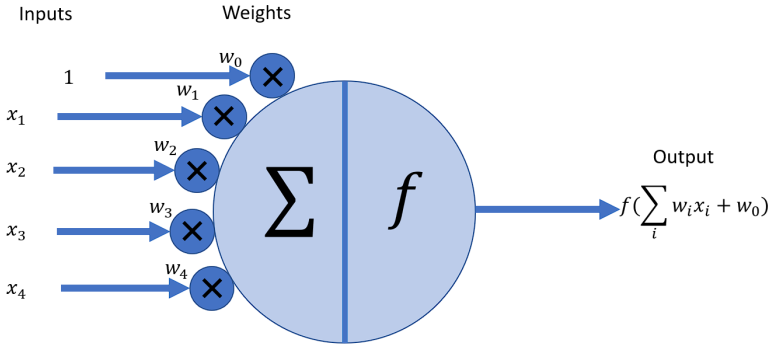


Figure 5.2 – Structure of a neuron of an artificial neural network with four inputs. Inputs are multiplied by their weights, summed altogether with an additional bias and processed through an activation function.

cases. By calculating the gradient of the network cost function, the weights in the neurons can be updated, usually following dynamic programming optimization techniques. This results in the ANN becomes progressively more performant at its task until a stopping condition is met. This process is called supervised learning.

Identifying the correct set of weights so that the cost function is minimized requires a great number of data and repetitions. The data can be processed ten, hundreds, thousands

or more times until a satisfactory result is found. A full cycle of processing the training data set is called an epoch. Two common phenomena are important to avoid while training an ANN: they are called underfitting and overfitting. Underfitting happens when the network is unable to find a relationship between the inputs and the expected outputs, which results in a very low performance and high bias network. This can be avoided by increasing the training time and/or providing a bigger and more varied data set. Overfitting on the contrary is when the network is overspecialized in finding the relationship between the inputs and the expected outputs of the data set used for the network training. This results in an ANN which is unable to generalize to unseen cases.

To prevent overfitting, one popular solution is to separate the data set used for the training into three subsets. The first and usually the bigger proportion-wise is the *Training Set*. This is the data that is going to be passed into the network, have its output compared to the expected output using a metric such as the Mean Square Error (MSE), and used to update the weights of the neurons of the network. The second subset is the one used to limit overfitting and is called the *Validation Set*. This part of the data is also going into the network, but its output value will be used to detect when to stop the training of the ANN. On principle, as the error metric decreases on the *Training Set* output as more epochs are performed, the error metric on the outputs of the *Validation Set* is supposed to decrease as well. However, when the error on this set is starting to increase, this is the signal indicating when the training should be stopped to prevent overfitting. The theory behind this is that if the data set is large in both number and possible cases, a big enough *Validation Set* will cover a large enough number of cases to detect when the ANN gets over specialized. The size of the *Validation Set* depends on the size of the *Test Set*, usually between a tenth and a third of the total data, and is chosen randomly from the entirety of the data set. Finally, the last subset is the *Test Set*. It is used to measure the final performance of the neural network. [1]

Detailed mathematical results on neural network training can be found in [134], and a schematized representation of neural network training is represented in Figure 5.3.

In the end, a well trained neural network is a good way to approximate a function or perform complex pattern recognition by using a relatively simple mathematical system, which makes it very suitable for embedded deployment. The main obstacle is the creation or acquisition of the data set and the training process as a whole. Furthermore, a neural

1. It is noteworthy to understand that a network trained with the same data set from scratch multiple times will rarely result in the same weights in the neuron and in the same resulting accuracy.

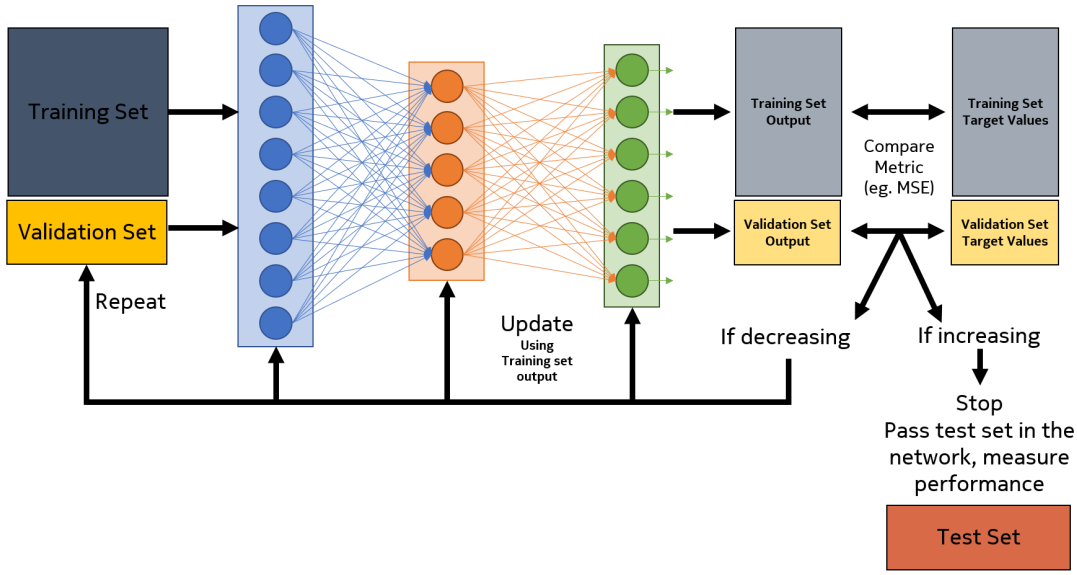


Figure 5.3 – Schematization of the training of a neural network.

network with more neurons and layers will tend to be more performant on more complex tasks [2], but will be significantly more complicated to train. Yet, neural networks see a lot of traction in optical networks. One advantage that ANN-based monitoring has in optical networks is that some optical monitoring data retrieved from the equipment can be complex to exploit due to the interactions between several system parameters [39]. A well-trained neural network can make relevant assumptions on the state of the network or an equipment from the raw data at a lower computational cost. On this idea, ANNs can exploit data that is usually left unused for real-time monitoring. One such example is the exploitation of the optical spectrum for nonlinearity monitoring.

5.1.2 Neural network-based optical nonlinearity monitoring and launch power optimization

In this subsection we will present the neural network that will be implemented on FPGA for optical transponder monitoring. This is based on the paper [15].
Even if in Chapter 4 we provided a solution for reconfiguring the optical launch power to improve the quality of communication between two optical transponders, this operation

2. While this tends to be true, a more complex neural network may be less efficient than a smaller one. For example two or more of its neurons in the same layer may have the same weights, which is redundant. Searching for the ideal size of neural network can be useful.

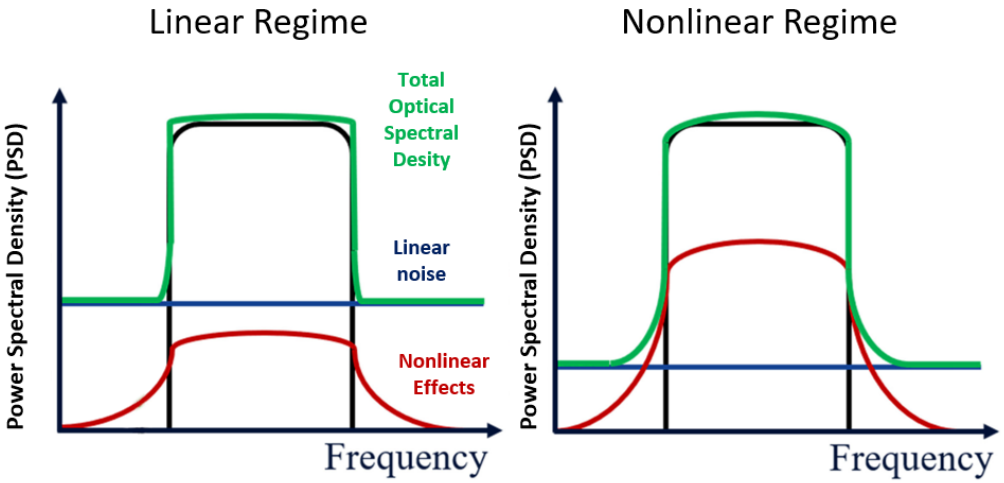


Figure 5.4 – Optical spectrum on a single channel in highly linear and highly nonlinear regime. The modulated channel is showed with the black line. In blue is represented the white linear noise, in red the nonlinear effects, and in green is the received spectrum with contributions of both linear and nonlinear effects.

is not always beneficial for the system. This is due to the fact that increasing the optical launch power can result in the system going into the highly nonlinear regime, increasing the contribution of the Kerr nonlinear interference on the channel, resulting in a Signal-to-Noise Ratio (SNR) degradation. The optimal launch power creates a trade-off between linear and nonlinear contributions. Examples of linear contributions are noise coming from amplifiers (Amplified Spontaneous Emission noise, or ASE noise as presented in Section 2.1.2.3) or hardware impairments, and nonlinear contributions are for example Kerr effect or Raman scattering (see Section 2.1.3). In Figure 5.4, we represented an optical spectrum in highly linear and highly nonlinear regime. As we can see, the shape of the spectrum is affected depending on the regime it is in, due to the higher contribution of the linear or nonlinear impairments. If we represent the SNR as a function of the launch power we can usually observe a bell curve, with on the left side the linear regime and on the right side the nonlinear regime. Finding the optimal launch power correction is in short to search for an ideal power change ΔP so that the resulting SNR is the highest point of the bell curve, represented in Figure 5.5.

Traditionally, nonlinear and linear contributions had to be measured individually and independently, with the problem of nonlinear impairments being complex to detect. But as we can see from Figures 5.4 and 5.5, we are able to determine from the spectrum shape the regime the system is in, and potentially also estimate a power correction to apply to

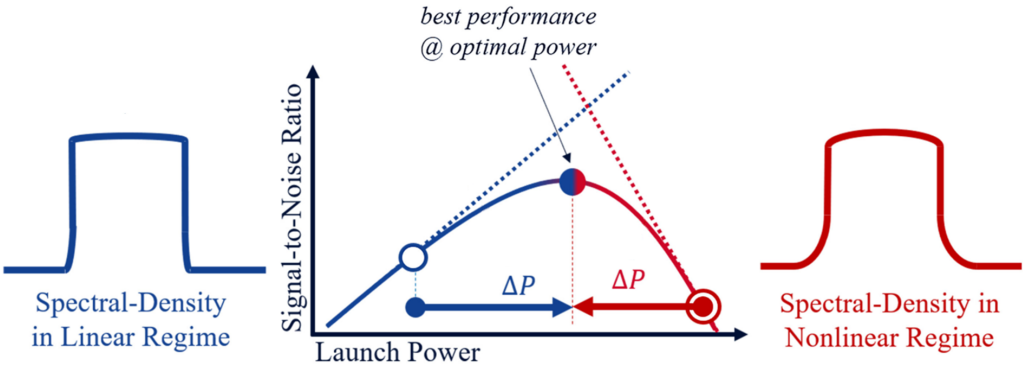


Figure 5.5 – Representation of the SNR of a received spectrum as a function of the launch power. In blue is shown the highly linear regime and in red the highly nonlinear regime. Atop the bell curve is the optimal launch power to maximize the SNR.

attain maximum SNR. This makes this task very suitable for ANNs, as all information necessary seems to be carried by the spectrum. In extension, training the network should be perfectly possible by providing it with varied spectrum with a good range of launch powers and other transmission parameters, so that it has access to multiple relations between launch power and SNR.

In [15], the authors propose a ANN architecture to detect from the Power Spectral Density (PSD) of a received signal the regime of the system, linear or nonlinear, and provide a power correction to maximize the SNR. The proposed ANN architecture in the article is represented in Figure 5.6. It takes in input a 315-long PSD vector normalized in the range [0;1] and outputs an estimation on the optimal power correction $\widehat{\Delta P}$. The hidden layer possesses ten neurons that have the sigmoid activation function ($f(x)$ in Figure 5.6). The single neuron in the output layer has the identity activation function. The test prediction plot of the neural network is provided from the article [15] in Figure 5.7, and the error histogram is presented in Figure 5.8.

This application has the advantage of that even if the ANN is not perfectly precise, as seen in Figures 5.7 and 5.8, applying a power correction that is slightly off the ideal one doesn't result in a massive detriment in SNR compared to the ideal correction, and even in worst case scenario when applying the predicted correction $\widehat{\Delta P}$ results a degradation in SNR, the degradation is likely to be minimal. This is represented in Figure 5.9, sourced from [15]. In this figure the SNR is calculated following this formula:

$$\Delta SNR = SNR(P_{launch} + \widehat{\Delta P}) - SNR(P_{launch})[dB] \tag{5.1}$$

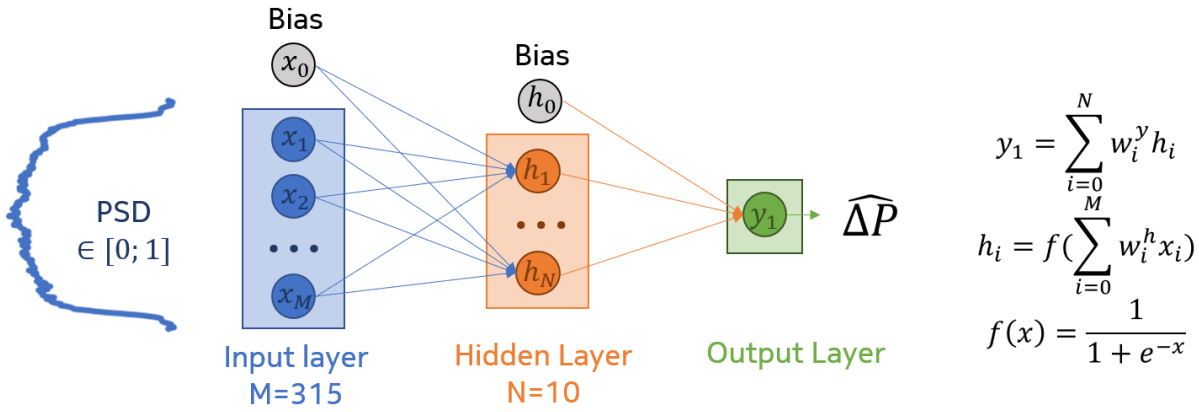


Figure 5.6 – Proposed Artificial Neural Network for power monitoring. The input is a normalized Power Spectral Density (PSD) of a single channel on 315 samples. The hidden layer is composed of a single layer of ten neurons. The single output produces an estimation of the power correction to apply to maximize the SNR $\widehat{\Delta P}$. Biases are always equal to one.

and the black line represents the maximum achievable SNR gain following the Gaussian Noise Model Theory [135] and is approximately the reverse bell curve. The predicted power correction results in a great majority of cases in a gain in SNR, and losses are concentrated around where the ideal power correction was small (less than a dB) and results in a maximum of -0.12dB loss.

This ANN-based optical power monitoring and its realization proposition is very interesting for our use case. The optical spectrum of a signal can be accessed relatively easily if in the network an Optical Spectrum Analyzer (OSA) or an oscilloscope has been deployed to monitor the state of the spectrum in a section of an optical network. And as [15] points out, the spectral density has a lot of untapped potential to exploit for the future of optical monitoring. We can imagine further exploitation of the PSD in the future, and maybe more utilization of ANN-based monitoring.
However, we run into problems we already touched upon previously. With optical monitoring being more prevalent with more and more available parameters and with more equipment to manage, overall reactivity and efficiency of the network's control plane resources can be hampered. This is particularly true for the future exploitation of PSD, as individual channel spectrum can be reduced to a few hundred samples, the PSD can be captured for multiple channels at the same time over a very long bandwidth, which requires elaborate pre-processing to retrieve and extract single channels even before applying the monitoring algorithms. As we pointed out in Chapter 4, providing closer to the

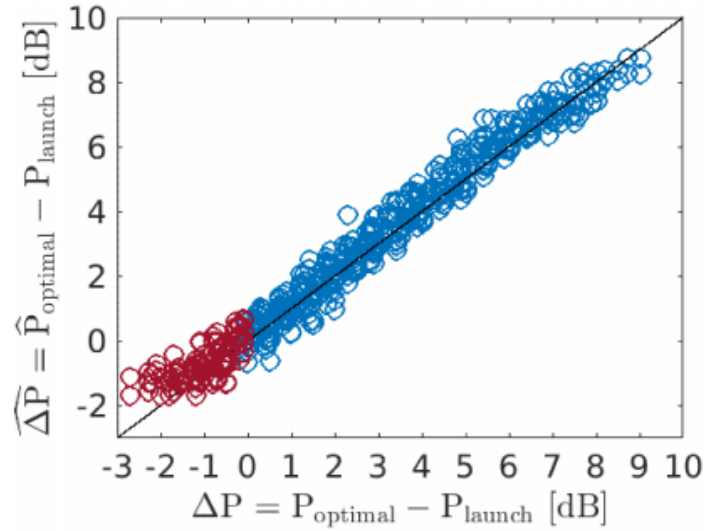


Figure 5.7 – Neural network self-test prediction scattered plot. The black line represents the perfect theoretical power correction ΔP and circles represent the prediction $\widehat{\Delta P}$ out of the ANN. Blue circles represents when the system was operating in the linear regime, and in red when the system was operating in nonlinear regime.

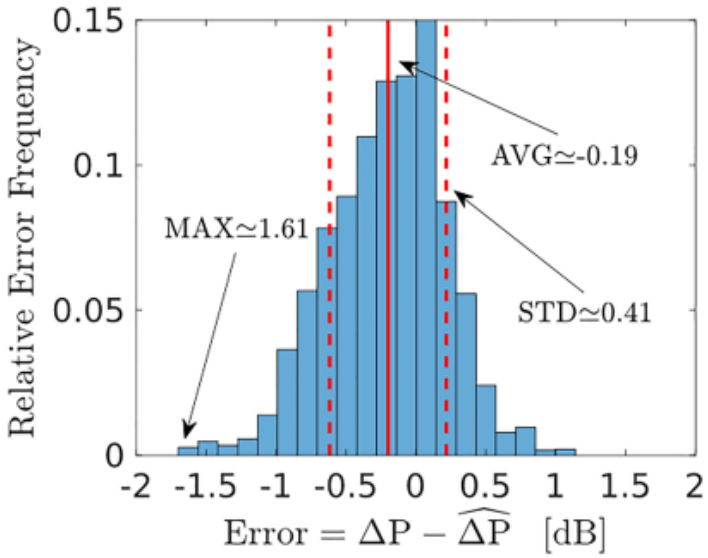


Figure 5.8 – Error histogram of the ANN, AVG labeling the average error, STD the standard deviation and MAX the maximum absolute error.

hardware monitoring and decision making can make the network more reactive overall. We can also envision that providing local monitoring can reduce the number of times or the quantity of information to transfer to the control plane.

Another thing to consider is the genericness of Artificial Neural Networks. By changing

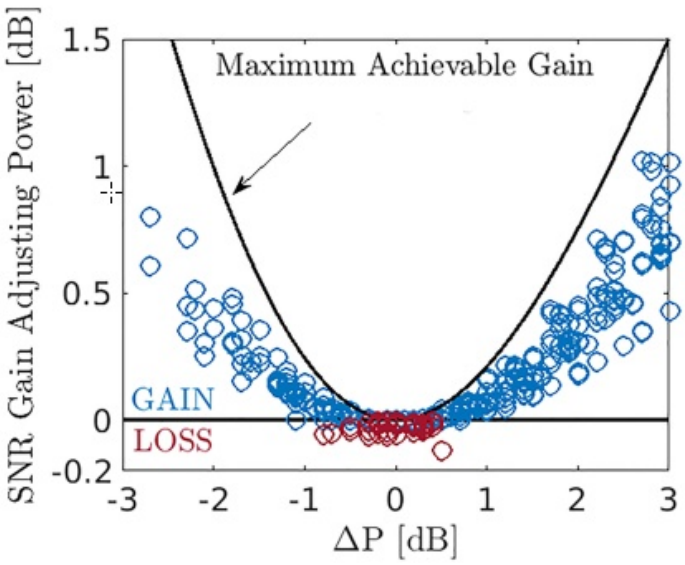


Figure 5.9 – Resulting gain in SNR when applying the predicted power correction $\widehat{\Delta P}$, compared to the maximum achievable gain when applying the ideal power correction ΔP. The blue circles show when the correction results in a SNR gain, and in red when it results in a loss. Maximum achievable SNR gain with the test set is around 1dB, and loss around -0.1dB.

the weights in the neurons, one ANN can produce another function on the same inputs. Proposing a generic architecture of ANN close to the hardware, employing fast reconfigurable hardware such as FPGA could be used both for accelerating monitoring tasks in the control plane and propose real-time ANN-based monitoring. This solution will be explored in the following sections, by providing a real-time implementation of the previously described monitoring algorithm. We will also provide for a method of fast neuron weights update, which can be exploited by the optical network controller to remotely reconfigure the function of the implemented ANN.

5.2 FPGA implementation

In this section we will detail the implementation of the ANN presented in Section 5.1.2, from the data set we will use, to the adaptations we had to make on the neural network and finally to the network implementation in the FPGA.

5.2.1 Data set and neural network generation

Data set To start things up, we had to do some changes on the neural network proposed in [15] due to some limitations. First of all, we are basing ourselves on another data set that the one presented in the article. Whereas the data capture was made using an oscilloscope, we are employing a data set realized on an Optical Spectrum Analyzer (OSA), which has a reduced resolution. This means that the individual channel spectrum will not be sampled on 315 points, creating a 315 input neural network as in Figure 5.6, but on 251 points, resulting on a 251 input neural network.

The PSD used for the data set are from a 32.5GBd PM-QPSK (Polarization Multiplexed-Quadrature Phase Shift Keying) signal, generated with a 1524.32nm external-cavity laser, with IQ modulators being driven by Digital-to-Analog Converters (DACs) with 0.01 roll-off factor root-raised-cosine pulse shaping. The signal is sent in a loop of three 100km single mode fiber spans (i.e. a 300km loop). At the entry of each span the signal is amplified by an Erbium Doped Fiber Amplifier (EDFA) with a launch power whose value range from 10dBm to 22dBm. For the data set, 20 acquisitions have been made for each combination of 22 equally spaced different launch powers and 1 to 4 cycles of the loop (i.e. for distances from 300km to 1200km). This results in a total of 1760 acquired spectrum. In Figure 5.10 are shown the PSDs for all distances at (a) 10 and (b) 20dBm launch power. At 10dBm the system should operate in linear regime, at 20 in nonlinear regime. We can see the impact of the linear and nonlinear contributions on the edges of the spectra as represented in Figure 5.4.

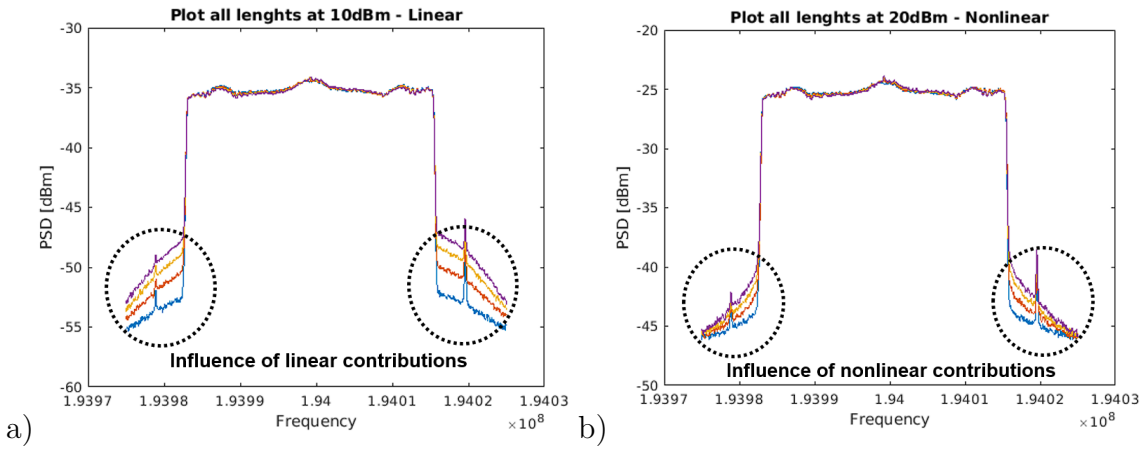


Figure 5.10 – Plot of the PSD for all lengths at (a) 10dBm launch power, (b) at 20dBm launch power.

We can then calculate the SNR of each spectrum and plot the bell curves for each loop cycle. In Figure 5.11 we plotted the SNR as a function of the launch power for our data set. For each distance, we interpolated the SNR curve using a polynomial of degree 3, resulting in bell curves, just as in 5.5. We then for each bell curve determine their maximum, which will be our target value in terms of SNR for each spectrum. This finally allows us to determine the ideal power correction value for each launch power of each number of loop cycles. These values are the expected output values of our neural network that will be used for its training.

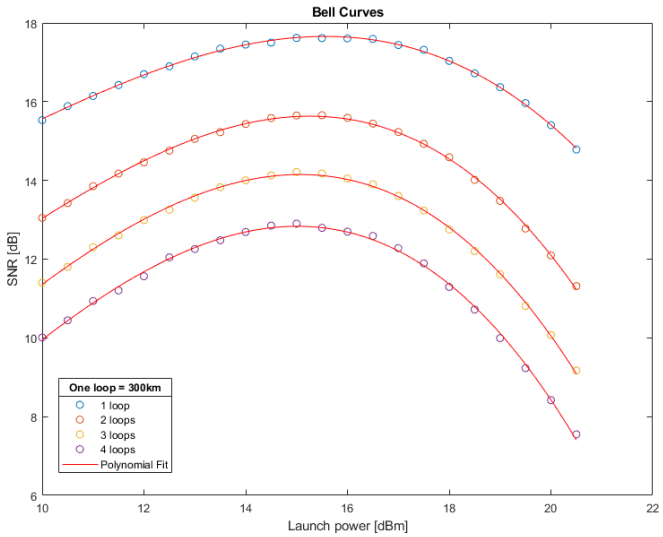


Figure 5.11 – Plot of the SNR as a function of then launch power with its associated bell curves for distances from 300 to 1200km. The solid lines represent the fit line from a polynome of degree 3.

Neural network generation for FPGA implementation Now that we have a useable data set we can generate and train our neural network. While in [15] the neural network generation and training was handled using TensorFlow [136], a popular software library, we decided to use the `fitnet` [137] function in Matlab which was more practical and fast to use for our needs.

As presented in the previous section, we are generating a neural network with a single hidden layer of ten neurons, with a single output neuron, but with 251 inputs rather than 315 as touched upon previously. We also changed the activation function of the neurons in the hidden layer from the logistic sigmoid to the Hyperbolic Tangent (tanh) function,

which is the default activation function for fitnet. As a reminder, the sigmoid function is written as:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5.2}$$

The tanh function can be considered a rescaling of the sigmoid function and written as:

$$tanh(x) = 2S(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{5.3}$$

Both functions are plotted in Figure 5.12. Both give similar results for neural network training [134].

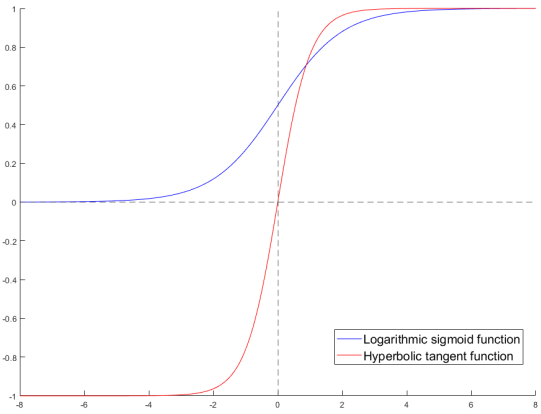


Figure 5.12 – Plot of the logarithmic sigmoid and the hyperbolic tangent functions.

The neuron in the output layer still has the identity activation function.
For the neural network training, the data set is separated between training, validation and test subsets in 80-10-10 proportions. The neural is trained using the Levenberg-Marquardt backpropagation algorithm [138]. In Figures 5.13 to 5.16, we present the results of a neural network training.

In Figure 5.13 we can observe the performance of the neural network after each epoch of training. We can see that the MSE between the output of the network and the expected result keeps diminishing, which means that our neural network increases in performance. We can also observe that the MSE is different between the train, validation and test subsets, and that when the MSE increases for the validation set the value of the weights is set. Figures 5.14 to 5.16 show different measurements of performance and error for our network. If we compare Figures 5.7 and 5.8 with 5.14 and 5.15 respectively, we can see that our network is less performant than the one in [15]. This is partly due to the fact

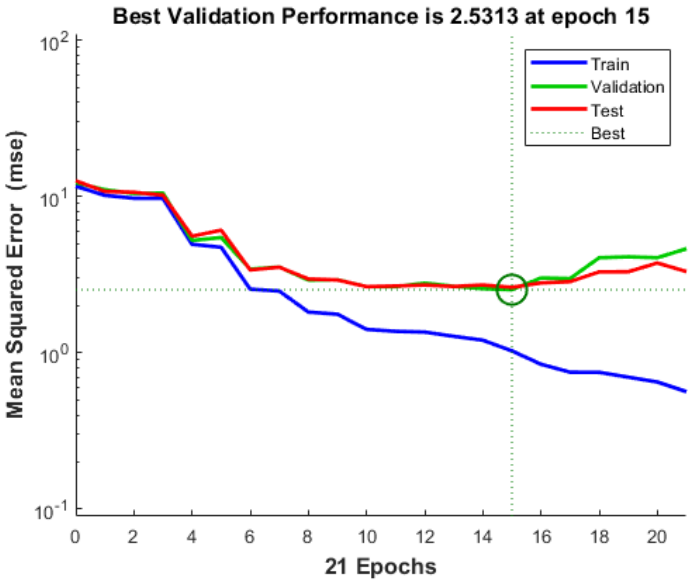


Figure 5.13 – Plot of the evolution of the neural network performance on the data set as more epochs are being realized. The green circle represents the point at which the early stopping mechanism is triggered.

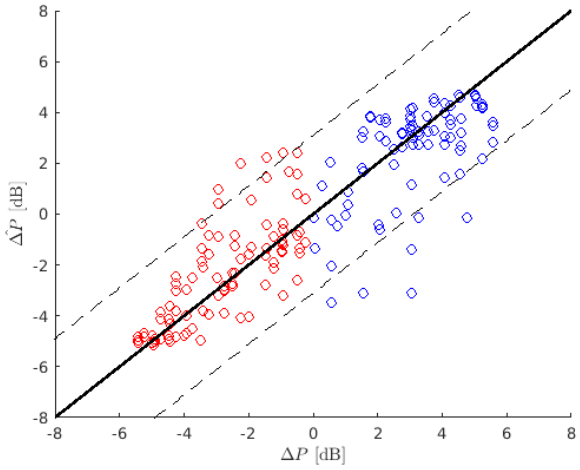


Figure 5.14 – Plot of the predicted result $\widehat{\Delta P}$ from the neural network as a function of the the expected result ΔP. The blue dots represent when the system was in highly linear regime and the red dots when the system was in highly nonlinear regime. The black line represents the ideal fit line where $\Delta P = \widehat{\Delta P}$, and dashed lines the standard deviation of the neural network output.

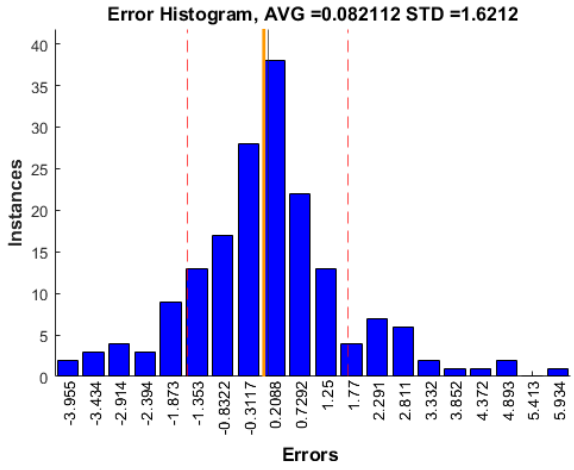


Figure 5.15 – Error histogram after a training of the neural network. Orange vertical line shows 0 error.

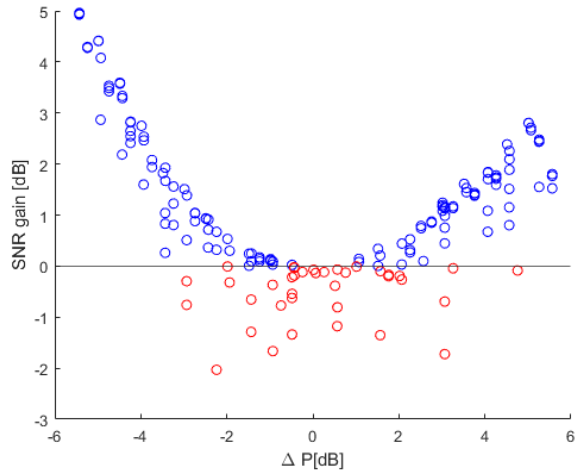


Figure 5.16 – Resulting gain in terms of SNR after a training of the neural network.

that the equipment used for the acquisitions of our data set is an OSA with less resolution than the oscilloscope used in [15]. However as we can see in Figure 5.16, our function still performs correctly, allowing for good gains in terms of SNR, with only a few losses [3]. In Figure 5.17 we plotted the resulting weights for all the neurons in the hidden layer for all 251 inputs.

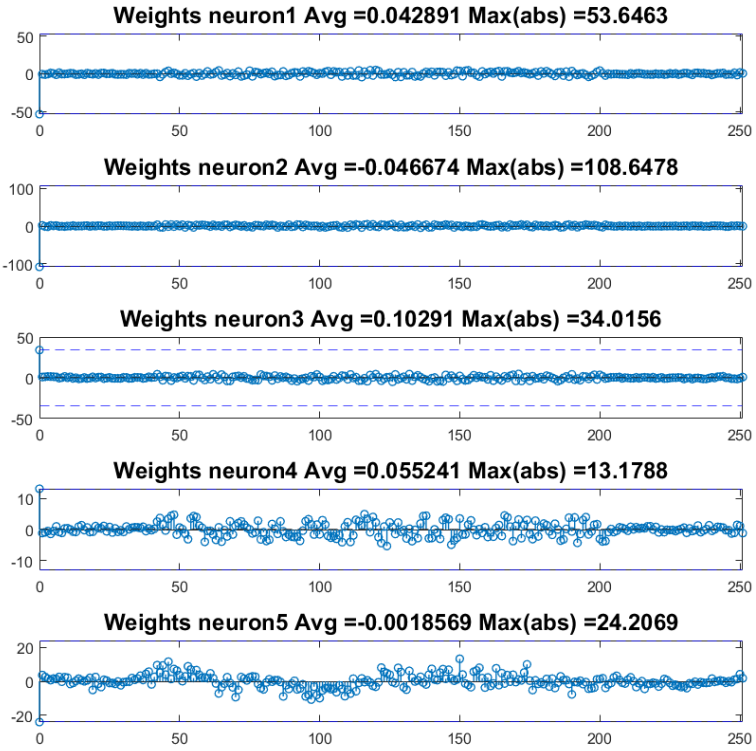
5.2.2 Logical Implementation

5.2.2.1 Neuron and network hardware structure

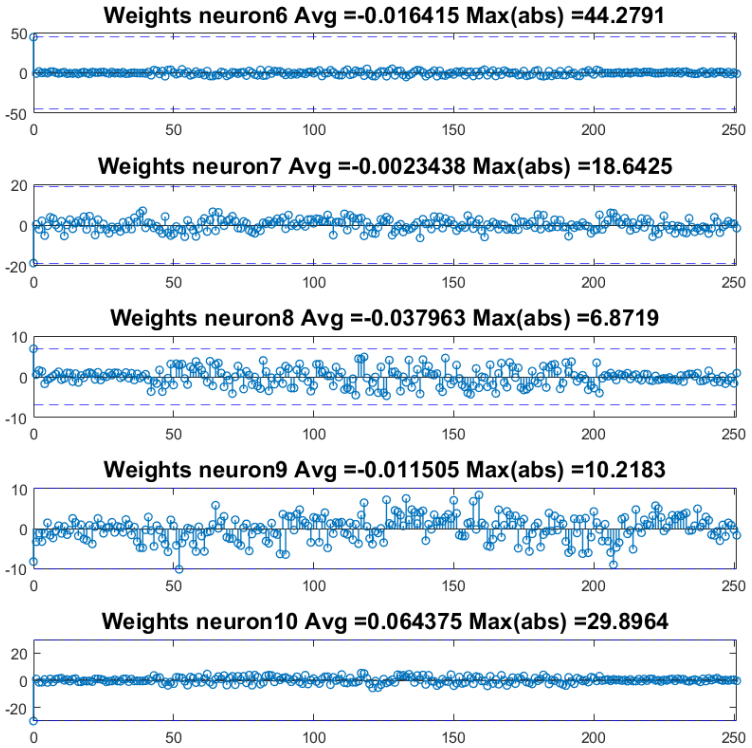
Now that we have a trained network we can design its logical implementation.
One of the first steps in designing an ANN for FPGA is to start by designing the neuron. As explained previously and shown in Figure 5.2, a neuron is constituted of a multiplication, addition and an activation function. This is straightforward to implement in logic, except for some choices in design. The neuron structure is represented in Figure 5.18.

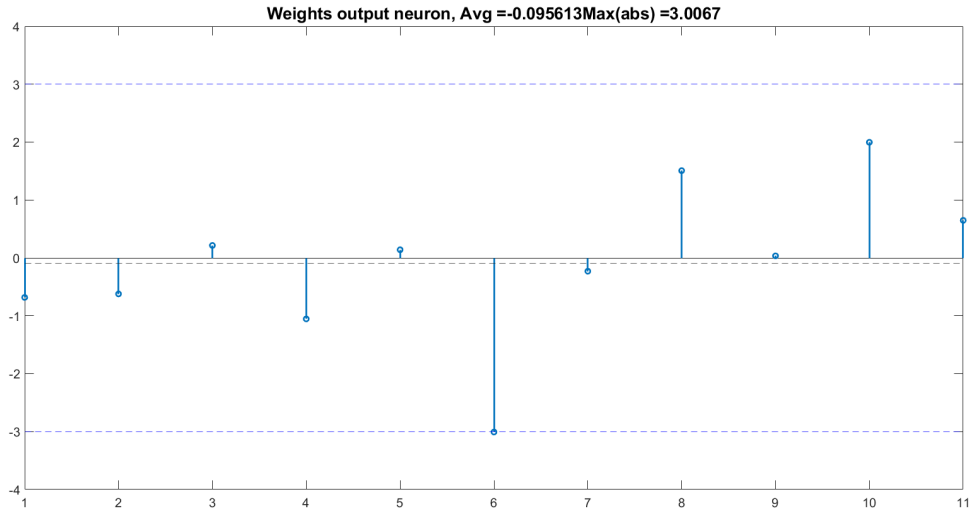
First, the data comes into the neuron serialized and not in parallel, which is much easier to handle in logic. This does not undermine the main advantage of the FPGA, which is the parallelization of the operations, as all neurons of a layer can function all at the same time. This allows for a boost in processing time, as we will demonstrate later. The first operation is the multiplication of the input with its corresponding weight. The weights are stored in a Read-Only Memory (ROM), which is read sequentially with the help of a counter. The weights out of the ROM are multiplied with their respective input. The output of the multiplier goes into an accumulator, where the bias weight is added directly. When the summation is done, its result goes in the address port of a ROM which stores the values of the tanh activation function, and the output of the ROM serves as the output of the neuron. All these operations are synchronized using control signals, indicating when the input is valid, when the counter reached the number of inputs (to redirect the output of the ROM with the weights and more importantly the bias towards the accumulator) and when the accumulator has reached its final value to and when the output of the activation function is valid.
This implementation is correct for the neurons of the hidden layer. The single neuron of the output layer follows the same principle, without the ROM with the activation function.

3. Performance could be improved by having a better data set to train the network, or fine tuning the ANN, but this is out of the scope of this thesis work and still perfectly suitable for use.


Weights neuron1 Avg =0.042891 Max(abs) =53.6463
Weights neuron2 Avg =-0.046674 Max(abs) =108.6478
Weights neuron3 Avg =0.10291 Max(abs) =34.0156
Weights neuron4 Avg =0.055241 Max(abs) =13.1788
Weights neuron5 Avg =-0.0018569 Max(abs) =24.2069

a)


Weights neuron6 Avg =-0.016415 Max(abs) =44.2791
Weights neuron7 Avg =-0.0023438 Max(abs) =18.6425
Weights neuron8 Avg =-0.037963 Max(abs) =6.8719
Weights neuron9 Avg =-0.011505 Max(abs) =10.2183
Weights neuron10 Avg =0.064375 Max(abs) =29.8964



b)

Figure 5.17 – Plot of the weights (a) for the ten neurons of the hidden layer, (b) for the output layer neuron, with their average value and their maximum absolute value (index 0 is the bias weight w_0^i as noted in Figure 5.6, 1 is the weight for the first input w_1^i etc...).

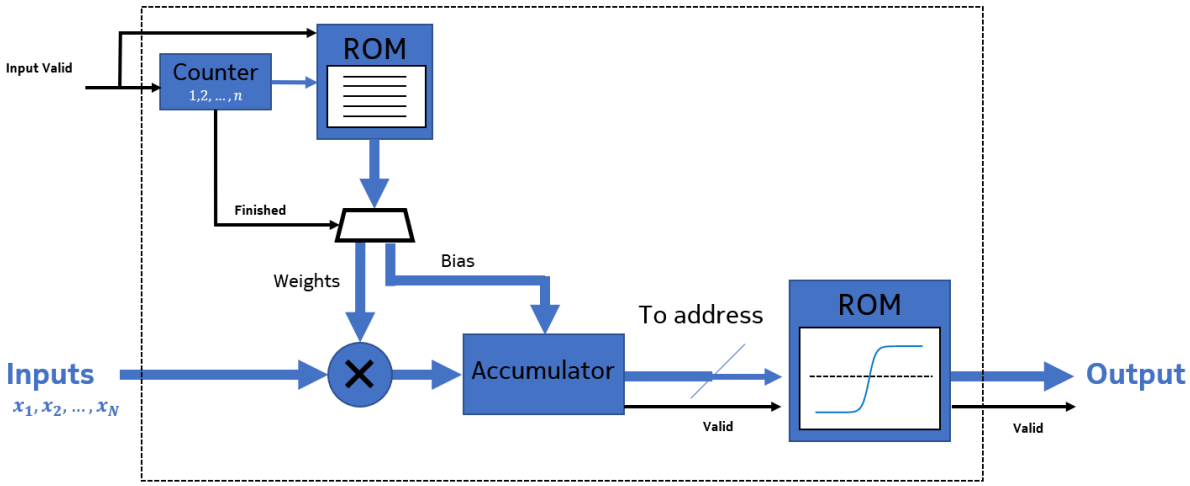


Figure 5.18 – Schematization of the logical implementation of a neuron. Control signals are represented in black.

One important thing to consider is the number of bits used for each operation. We will work with inputs whose values are in [0;1] and with weights with real decimal values as can be seen in Figure 5.17. As it is complex to use floating point signals and operations in FPGA, we will be using fixed point arithmetic. Signed bit vectors will be written with

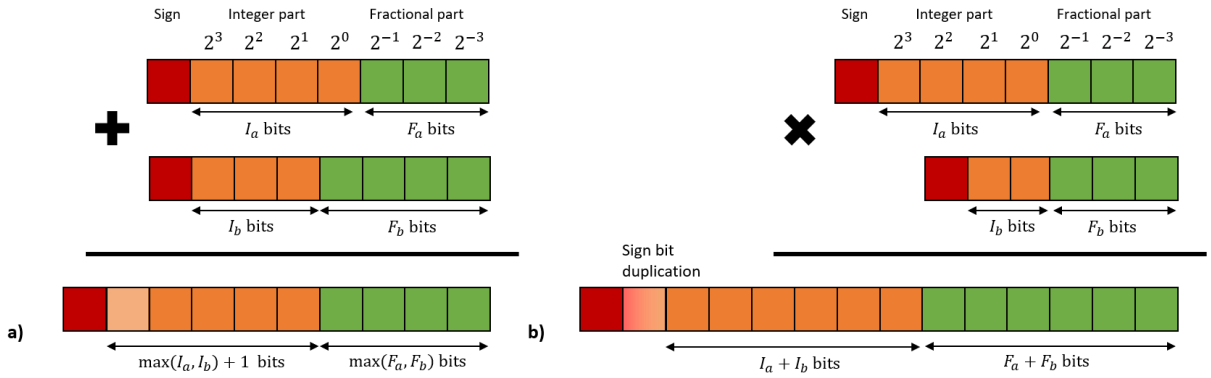


Figure 5.19 – Rules of the signed fixed point arithmetic for (a) addition and (b) multiplication.

the two's complement method. As a reminder, using a signed fixed point bit vector means that the most significant bit will be used as the sign bit, and a certain number of bits will be reserved for the fractional part of the value carried by the bit vector [4]. In Figure 5.19 we show how is composed the resulting bit vector of a signed fixed point addition and multiplication. One area of concern is the accumulation that could overflow the bit vectors, as theoretically we should add 251 bits to the output compared to the input bit vector to prevent it completely. In practice, and knowing the range of the weights produced by the neural network training, if we have enough margin in number of bits in the integer part at the first operation, the chances that we overflow the bit vector are low. So finally, rather than adding 251 bits to the bit vector, we decided to treat the accumulation function like an addition operation and follow the rules as showed in Figure 5.19 (a), and add a single bit to the bit vector out of the accumulator.

To save space in the ROM storing the activation function we firstly set bounds to the function values. As can be seen in Figure 5.12, the tanh function gets close to its limits (-1 and 1) very fast, and it is not very interesting to store the values of the function above a certain point. So we decided to bound the function such as

$$f(x) = \begin{cases} \frac{2}{1+e^{-2x}} - 1, & \text{if } x \in]-8, 8[, \\ -1, & \text{if } x \leq -8, \\ 1, & \text{if } x \geq 8. \end{cases} \tag{5.4}$$

4. This implies that the most significant bit of the fractional part will correspond to 2^{-1}, the next one 2^{-2}, etc... for a positive value

This means that we only need to check part of the integer portion of the incoming bit vector and we don't need to store all the possible values of the activation function past our bounds. This allows us to use the space we wish to allocate to the activation function more sparingly.
To reduce further the space taken by the ROMs carrying the activation functions, we have also chosen to limit the number of values stored, i.e. limiting the depth of the ROMs. We choose to fix a depth and fill the memories accordingly. As an example, if we set the size of the ROM address port to X, we will store 2^X values, which gives a step size for the stored values equal to

$$\frac{HighBound - LowBound}{2^X - 1} = \frac{8 - (-8)}{2^X - 1} \tag{5.5}$$

This implies that the address port on the ROMs may not have the same size as the bit vector out of the accumulator, and a reduction of the bit vector is required. Logically, bits of the integer part representing values above the bound and extra bits in the fractional part need to be cut off.
Also, the result of the accumulation function is written as two's complement but, as design choice we wrote the activation function in the ROM naturally. This means that the value of our low bound (-8) is stored at the lower address 0 and the value for our high bound (8) is stored at the highest address. To convert the vector out of the accumulation function to the correct address, an inversion of the sign bit is required [5].
A representation of these operations is represented in Figure 5.20.

Logically, the bit vector out of the ROM will have a high fractional part as it will only need a single sign bit and a single integer bit. This means that we have the possibility to have a very nice precision for the activation function value. For example if the output of the activation function ROM is a vector with a 14-bit fractional part (from a bit 16-bit vector with one sign bit and one integer bit for example), we can have values precise to the 2^{-14}, i.e. $6.1035 \cdot 10^{-05}$.

Now that we have the structure of a neuron, we can deploy them to make our neural network. In Figure 5.21 we represented our neural network structure.

We can see that we have our ten neurons from Figure 5.18 for the hidden layer that receive the serial input data, and a single neuron with no activation function ROM at the

5. As an example, for a signed 4-bit vector written as two's complement, the lowest value -8 is written `1000`. Its corresponding address is `0000`. For 7, written `0111`, the address is `1111`. This is true both for integer and fixed-point bit vectors.

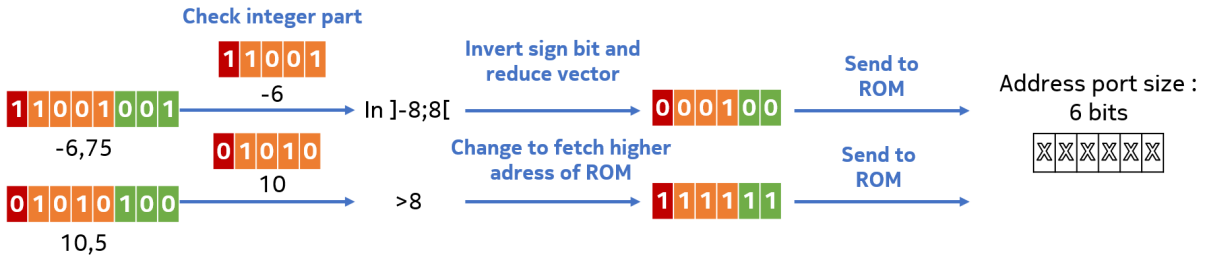


Figure 5.20 – Representation of the adaptations made on the bit vector from the accumulator to fetch the value from the value from the activation function ROM. As an example we took initial signed vectors with four bits integer part and three bits fractional part and a ROM depth of 6.

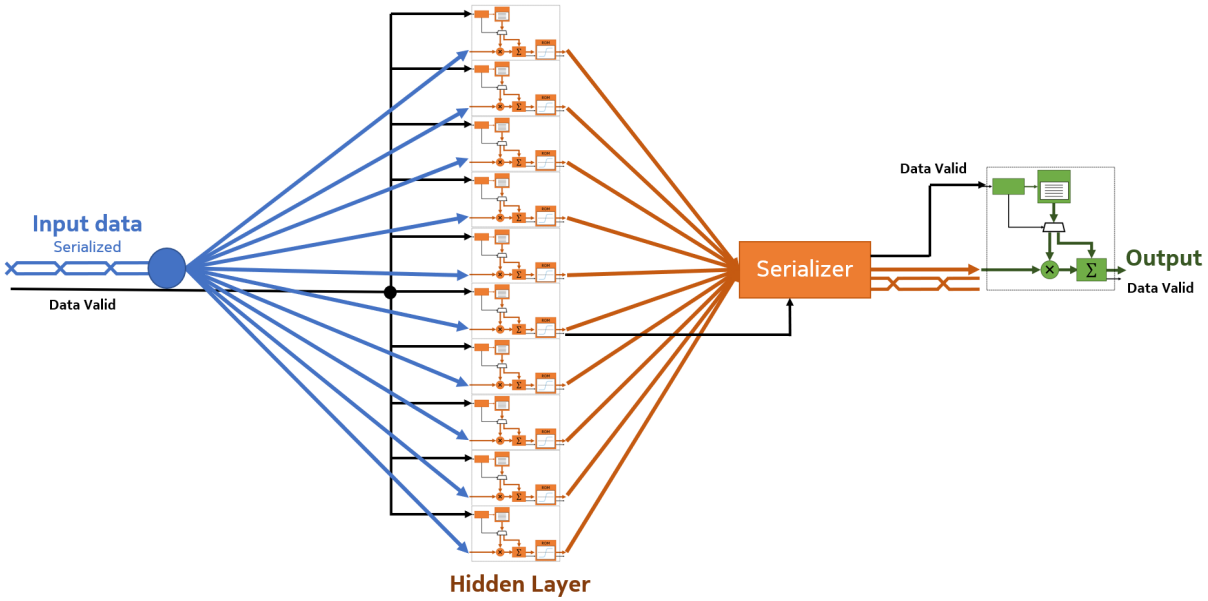


Figure 5.21 – Structure of the neural network logical implementation for FPGA. Neurons in the hidden layer refer to the neuron structure presented in Figure 5.18, output layer neuron is the same, but with no activation function ROM.

output. Between the hidden layer and the output, we put a serializer, as all neurons from the hidden layer will output their value at the same time, as they all work in parallel.

5.2.2.2 Simulation validation and resource utilization

We will now validate our ANN implementation for FPGA. We will begin by discussing the different sizes in bits of the different values we are going to employ (inputs, weights, activation function). It is an important consideration because the bigger the values of these elements in bits, the more resources will be mobilized in the FPGA to implement the neural network. As a reminder from Section 3.1.1, operations in an FPGA are carried by Look Up Tables (LUTs) and data is stored using RAM, which have a limited size and are in limited number in the fabric. Also the bigger the bit vectors, the harder it will be for the implementation tools to route the different signals in the FPGA. Even though in our particular application a relatively bad precision will not have a significant impact on the SNR gain as shown in Figure 5.9, we still need to demonstrate that our neural network solution is capable in other scenarios of outputting a precise enough value. Finally, to potentially avoid the possibility of overflow for the accumulation, the weight bit vectors should have a big enough integer part as discussed in Section 5.2.2.1.

Firstly, we settled for the inputs for unsigned vectors of 32 bits, with one bit for the integer part. We based ourselves for the format of this vector from the files outputted from a Finisar OSA. This led to some adaptation. Before doing any operation, we add a sign bit to the bit vector to change it from unsigned to signed. This added bit doesn't change a lot the base of our system. This means that the bit vector coming in the multiplier of the hidden layer neurons is 33 bits wide.

For the weights and the activation function, we can experiment and see some impact from the values we decide to use. As we can see from the structure of the neural network implementation in Figure 5.21, the size of the output bit vector will depend only on the sizes of the weights and the activation function vectors. More precisely, following the rules in Figure 5.19, the size of the output bit vector will be exactly:

$$Nbits_{activation} + Nbits_{weights} + 1 \tag{5.6}$$

This means that the precision of the result out of the FPGA will highly depend on the precision of the weights (i.e. their size) and the precision of the activation function. The precision on the activation function value will depend both on the size of the vector stored

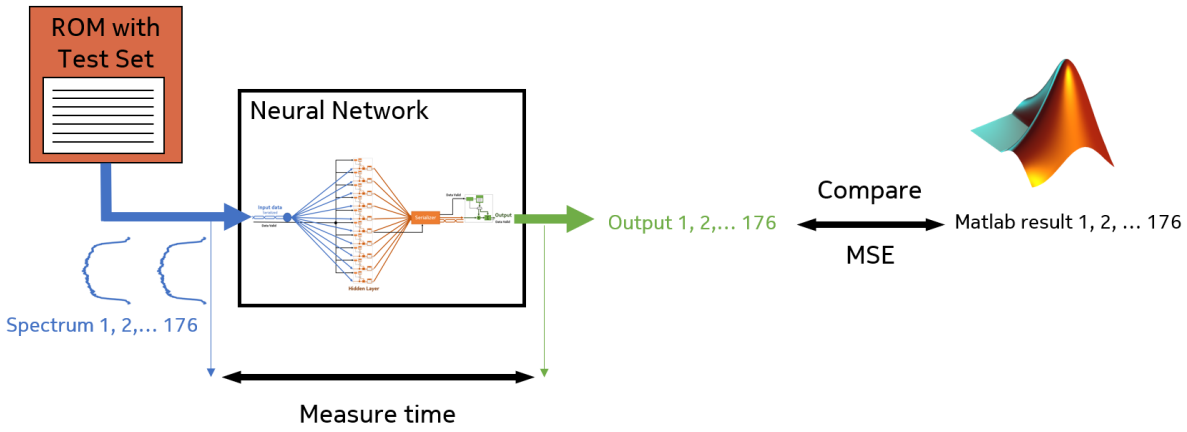


Figure 5.22 – Simulation setup to validate our neural network and measure its performance in terms of timing and precision.

in the ROMs (the width of the ROM), the number of values stored in the memories (the depth of the ROM, which determines the step size of the stored function), but also directly depends on the precision on the weights and the inputs, as they directly impact which value of the activation function ROM is going to be fetched.

For the simulation validation, we generated and trained a neural network in Matlab. As previously stated, the data set is comprised of 1760 251-samples spectra and is separated in 80-10-10 proportions between Training, Validation and Test sets. This trained neural network has been represented in Figures 5.13 through 5.17. We then put the generated weights and the activation function into our logical implementation of the network. We will store the full Test Set (i.e. 176 251-samples spectra) in a ROM that will provide the input data for our simulation. Samples are already normalized and written as unsigned 32-bit bit vectors with a single bit integer part. We will firstly measure the time it takes for our implementation to provide an output. Also, for each spectrum, we will compare using MSE the resulting output $\Delta\widehat{P_{FPGA}}$ from the FPGA with its corresponding $\Delta\widehat{P_{Matlab}}$ from the generated network in Matlab. This will provide us the precision of our implementation. We will also test different sizes of bit vectors for the activation function and the weights and numbers of activation function values stored in the ROMs. In all cases, weights are written as signed bit vectors with 7-bit integer part, the activation function as signed bit vectors with a one-bit integer part. The setup is represented in Figure 5.22.

In Figures 5.23 & 5.24 we show the simulation results and highlighted the timing performance for the neurons and for the whole network respectively. As we can see, with

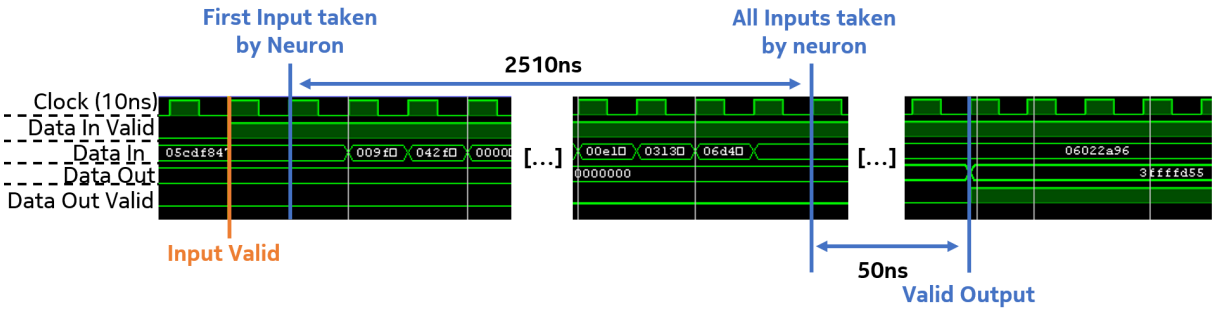


Figure 5.23 – Extract of the simulation results focused on a neuron of the network.

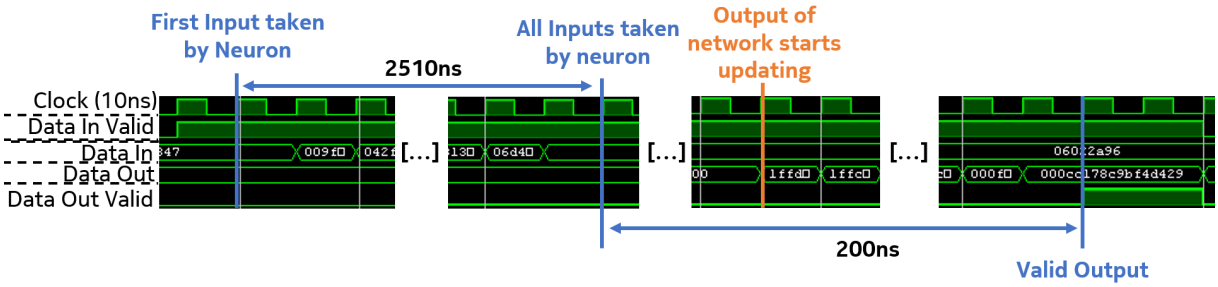


Figure 5.24 – Extract of the simulation result of the FPGA neural network.

a 100MHz (10ns period) clock, the neurons take 50ns after the last input has been acquired by the neuron to have a valid output, or 2560ns if we count starting at when the first input has been received by the network. Counting from the last input taken by the neuron, this result is coherent with the data flow: one clock cycle for multiplication between weight and input, one clock cycle to make the accumulation, one clock cycle to add the bias, one clock cycle to transform the bit vector from the accumulator and a last one to fetch the value from the activation function ROM. On the whole network level, it takes 200ns from the last input fed into the network to the output of the $\Delta\widehat{P_{FPGA}}$ result. This gives a total of 2710ns from the first input to the output result. As an indicator we measured in Matlab the time it takes to make the same operations. The median time to have an output is 3.6ms[6]. This is roughly a 753 times gain when compared to Matlab.

In Table 5.1 we provided the precision measurements in MSE of our FPGA neural network against the one generated in Matlab. We have 12 different setups. The activation function ROM depth values are 10, 11 or 12, which give 1024, 2048 and 4096 stored values respectively, and step sizes of $15.6{\cdot}10^{-3}$, $7.82{\cdot}10^{-3}$ and $3.91.10^{-3}$ respectively following

6. Measured using the `timeit` function. This timing value is heavily dependent on multiple factors, and as stated is given as indicator.

Activation Function / Weights	Depth: 2^{10} Width: 16	Depth: 2^{11} Width: 16	Depth: 2^{12} Width: 16	Depth: 2^{10} Width: 32	Depth: 2^{11} Width: 32	Depth: 2^{12} Width: 32
Width: 16	$1.926.10^{-3}$	$1.768.10^{-3}$	$1.704.10^{-3}$	$1.927.10^{-3}$	$1.768.10^{-3}$	$1.705.10^{-3}$
Width: 32	$8.773.10^{-5}$	$2.234.10^{-5}$	$5.572.10^{-6}$	$8.773.10^{-5}$	$2.238.10^{-5}$	$5.604.10^{-6}$

Table 5.1 – Mean Square Error of our FPGA Artificial Neural Network versus the same one generated in Matlab (lower is better) with various weights ROM width and activation function ROM depth and width.

Equation 5.5. The width of the activation function and weights ROMs are 16 or 32 bits. We can see from the results that the MSE is at worst $1.926{\cdot}10^{-3}$, and at best with our list of setups $5.604{\cdot}10^{-6}$. The precision even at its lowest is satisfactory for our application as described earlier in 5.1.2, and extremely good at its highest for any other application. In particular, the width of the weights bit vectors has a huge impact on the precision of the network, which is expected as they impact both the result out of the network and the value of the activation function out of each neuron. As an example, for an activation function ROM with a depth of 10 and width of 16 bits, the MSE has a 21.9 times decrease from $1.926{\cdot}10^{-3}$ to $8.773{\cdot}10^{-5}$ when switching the weights width from 16 to 32 bits.

Increasing the depth or width of the activation function ROM has limited effects on the precision of the network comparatively. Increasing the depth of the activation function ROM from 10 to 12 gives a 1.13 times decrease of MSE of $2.22{\cdot}10^{-4}$ from $1.926{\cdot}10^{-3}$ to $1.704{\cdot}10^{-3}$ with weights at 16-bit width, and when 32 bits the decrease in MSE is 15.6fold of $8.216{\cdot}10^{-5}$, from $8.773{\cdot}10^{-5}$ to $5.572{\cdot}10^{-6}$, which is not a really substantial gain with such an already good enough precision.

Lastly, changing the width of the activation function does not have a noticeable impact on the precision at fixed depth and weights width. As stated before, the activation function is written as a signed 16 or 32 bit vector with a single integer part bit, which means that the value of the activation function is precise to the 2^{-14} and to the 2^{-30} respectively, $6.104.10^{-5}$ and $9.313{\cdot}10^{-10}$ respectively. It is very likely that this increase in precision is not important, as the starting value can be considered good enough for the operations of the neural network. Also the weights have more impact on the different operations of the network and benefit more from an increase in precision. Comparatively, increasing the weights width from 16 to 32 makes the weights value precise from 2^{-8} to 2^{-24}, $3.906{\cdot}10^{-3}$ to $5.960{\cdot}10^{-8}$, as they are written as signed bit vectors with a 7-bit integer part.

In Table 5.2 we represented the resource utilization results of the implementation of our neural network. We implemented the ANN with the same set of possible setups as in Table 5.1. We measured the usage of LUTs, Block RAM or Ultra Ram (in our specific case, only URAM is used, BRAM is given as an indication if no URAM was available in the FPGA) and DSP slices of a VCU118 Virtex Ultrascale+ FPGA evaluation board. In all cases, LUT usage is contained, from 1316 at the lowest (10,16)/16 setup (respectively the activation function ROM depth 2^{10} bits, width 16 bits and the weights width 16 bits) to 2407 at the highest (12,32)/32 setup. Also, we can see that increasing the weights width roughly doubles the number of DSP used in the FPGA fabric from 22 to 42, and greatly increases the base BRAM usage. However the URAM usage does not increase, and only increases when we change the width and/or depth of the activation function ROM. This is likely due to how the memories of our design are handled by the implementation tool. In any case, the BRAM usage is high, starting at 62.4% usage (1348 cells) with lowest settings and going up to 99.4% (2147 cells) at higher settings. This highlights that depending on the targeted environment, memory usage of our design with the proposed setups can be a problem, and could even prevent the design meeting timing constraints if no memory with similar or higher density as URAM is available. We can make the same remark with the DSP slices, whereas usage in our case is low relatively (0.32% or 0.64%), on lower end FPGAs this could be problematic.

In conclusion, the choice of bit vectors width and ROM depth is a trade-off between precision and overall occupancy of the resources of the FPGA. Depending on the application requirements in precision and the targeted environment and its resource availability, one must chose these parameters accordingly. But we can definitely say that the choice of the bit vector weights is the most important of them all, as the gains in increasing the size of the activation ROM width or depth does not provide a significant increase in precision.

5.2.3 Network weights update

One of the goals we want to achieve with our logical neural network structure is to be able to remotely perform monitoring tasks for the control plane. The nonlinear contributions monitoring function we have presented takes as inputs optical spectra, which is a data heavy metric that may delay the decision-making process, as data has to be retrieved and processed before a decision can be made. As shown in Chapter 4, providing closer to the device intelligence and decision power can increase the reactivity and reliability of the optical network, as the centralized control plane has to monitor the whole network in

Setup | Resource Usage	LUTs	BRAM / URAM	DSP
(10,16)/16	1316 (0.11%)	1348 / 7.5 (62.4% / 0.78%)	22 (0.32%)
(11,16)/16	1321 (0.11%)	1357 / 10 (62.8% / 1.04%)	22 (0.32%)
(12,16)/16	1335 (0.11%)	1365 / 15 (63.8% / 1.56%)	22 (0.32%)
(10,32)/16	1613 (0.14%)	1716 / 10 (79.4% / 1.04%)	22 (0.32%)
(11,32)/16	1623 (0.14%)	1726 / 15 (79.9% / 1.56%)	22 (0.32%)
(12,32)/16	1637 (0.14%)	1736 / 25 (80.4% / 2.60%)	22 (0.32%)
(10,16)/32	2087 (0.18%)	1790 / 7.5 (82.9% / 0.78%)	42 (0.61%)
(11,16)/32	2093 (0.18%)	1800 / 10 (83.3% / 1.04%)	42 (0.61%)
(12,16)/32	2105 (0.18%)	1810 / 15 (83.7% / 1.56%)	42 (0.61%)
(10,32)/32	2387 (0.20%)	2127 / 10 (98.4% / 1.56%)	44 (0.64%)
(11,32)/32	2395 (0.20%)	2137 / 15 (98.9% / 1.56%)	44 (0.64%)
(12,32)/32	2407 (0.20%)	2147 / 25 (99.4% / 2.60%)	44 (0.64%)

Table 5.2 – Summary of FPGA resource utilization of the implemented neural network design. Setup (X,Y)/Z corresponds to the depth of the activation function ROM (2^X values stored), the width of the activation function bit vectors (Y bits) and the width of the weights bit vectors (Z bits) respectively. BRAM (Block RAM, 18kb or 36kb) usage is given as an indicator, as the implementation tool will fall back into using only the URAM (Ultra RAM, 288kb) indicated for our case. In short, the BRAM usage corresponds to the usage if the FPGA didn't have access to URAM.

real-time, which delays individual monitoring tasks. We believe that this neural network structure that can be reconfigured by the control plane depending on the current needs in terms of reliability and reactivity can help improve the optical network performance overall. To achieve this, we will now implement the elements to update the weights in the ANN logic to make our architecture more generic and able to reconfigure its current function.

Neural network optimizations The first step in developing the weights update functions is to allow to write into the memories. This implies some design choices. We based ourselves on the fact that when updating the weights from one set to another, it is highly probable that all the weights have to be updated at once. We are not therefore allowing to cherry pick a weight to update, which would add unneeded complexity. So, to update the weights of our neural network, one must provide all the new weights, sequentially, and our logic should handle the addressing and proper writing of the new set.

In Figure 5.25 we pictured the new logical structure of the neuron. The first and obvious change is to add a writing port to the memory, effectively changing it from a

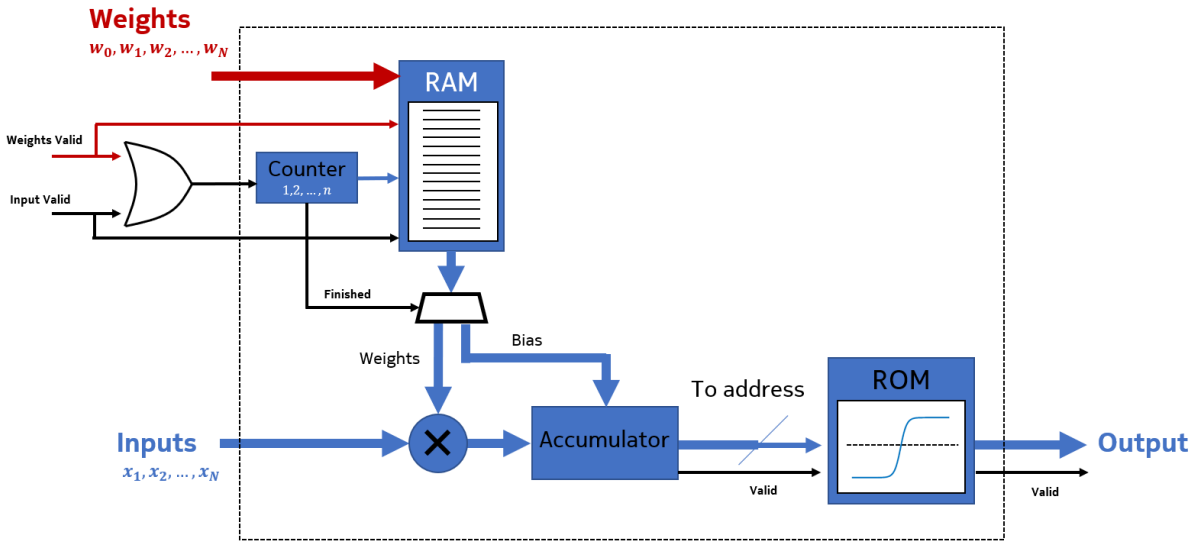


Figure 5.25 – New logical implementation of the neuron allowing for weights re-writing.

ROM to a RAM. By providing the weights sequentially and a control signal indicating when the weights are valid, we can control the writing of the neuron in the RAM by re-using the counter used for reading the memory to handle the address, by placing an OR gate that takes the *input valid* and *weights valid* signals before the counter.

On the network level, some other adaptations have to be made, as represented in Figure 5.26. Contrary to how the serialized inputs are handled, all the weights do not have to be handled by all neurons. The signal indicating that the weights are valid should be sent at a neuron when its weights are inputted and be set low in any other case. We handle this by putting a demultiplexer that sends the *weights valid* signal to the correct neuron, thanks to counter that counts from 0 to 251 and which sends the number of full counting cycles to the demultiplexer. This means that when the weights start arriving to the logic, the counter is at 0, the number of cycles is null, so the weights are sent to the first neuron of the hidden layer. Then after all 252 weights (one for each input and one for the bias) of the first neuron has been sent, the number of cycles increments itself and the second neuron receives the weights. This repeats for all the neurons of the network, with the eleventh and last cycle being for the output neuron, as it has less weights to change.

Microblaze and Ethernet interface integration Now that we are able to write a new set of weights in the logic, we can develop a setup to validate our neural network at implementation stage. One aspect we want to show from our setup is that the function

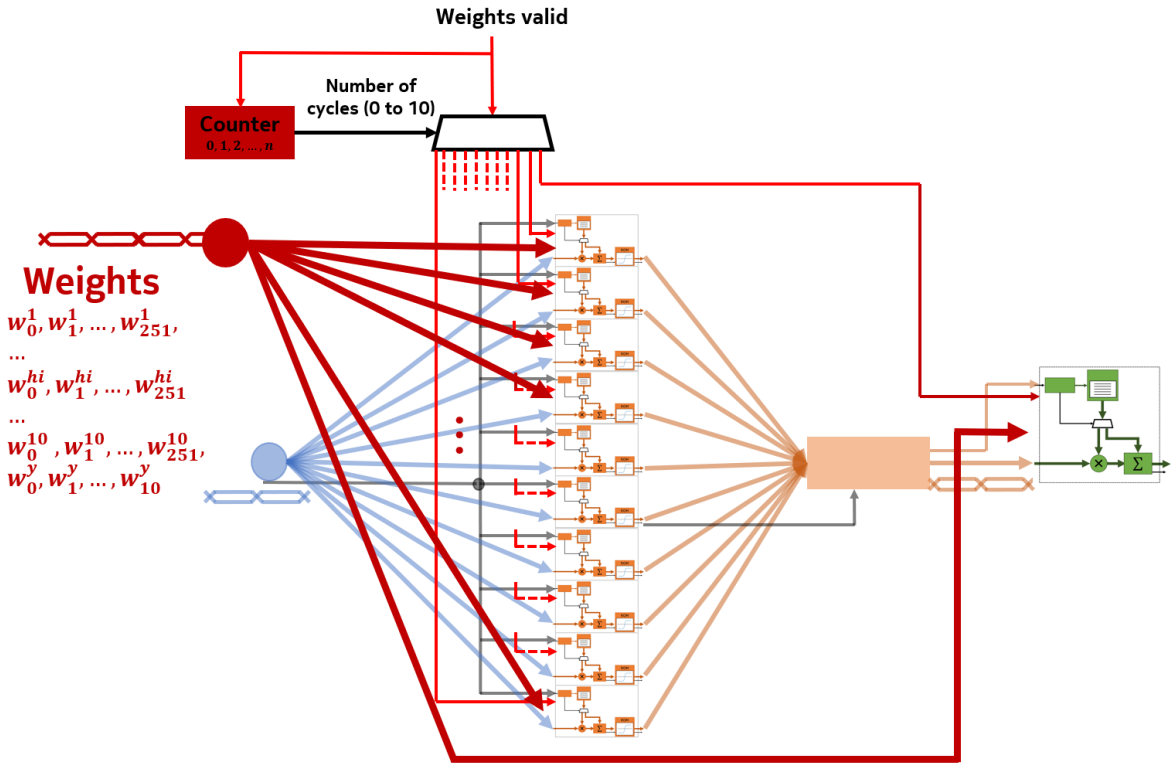


Figure 5.26 – Highlight on the weight update logic for the FPGA ANN.

of the ANN can be reconfigured by re-writing the weights remotely, for example by a centralized control plane. For this we can integrate a Microblaze soft processor in our FPGA to fetch new weights and fetch the inputs of our network from another equipment (for example an oscilloscope, an OSA etc. . . deployed in a node for example).

Introducing the processor in the design requires some adaptation. As it will provide for us the inputs and the weights, the Microblaze and the ANN need to communicate with each other. We have developed a wrapper for our neural network with a specific finite state machine to handle the AXI4-Stream (Advanced eXtensible Interface 4-Stream) protocol operations, which is compatible with the Microblaze. Specifications for the protocol are available in [139]. The base signals for this protocol are the *data* signal, the transmitted data that can be multiple bits wide, the *data valid* signal, that indicates if the data signal contains valid values that should be captured by the receiver, and the *ready* signal indicating that the receiver is able to receive data. Other optional signals can be used, such as the *data last* signal, used when sending a packet, a contiguous series of data, and indicating the end of a packet. This signal is implemented by default on the AXI4-Stream ports of the Microblaze.

First of all, in the wrapper we will implement three AXI4-Stream ports, one for the inputs of the network, one for the weights writing, and one for the output of the network. As we used *input valid*, *weights valid* and *output valid* control signals in the logic, we can directly use them for the protocol. As the number of expected values is known a priori, we do not need to take into account the *data last* signal for the input ports. For the output, as we output a single value, we can map the *data valid* signal to the *data last* signal. We only need to program additional *ready* signals for the input ports to signify when our system is able to receive data. We can let a AXI4-Stream First In First Out (FIFO) handle the *ready* signal of the Microblaze for the output of the neural network. We handle the *ready* signal for the input ports of the network simply by having them raised while nothing is happening in the neural network, and when we receive data either weights or ANN input, we put down the other. We do not want the weight update to start while input data are sent to the neural network and vice versa. We also added a few clock cycles to reset the logic of the ANN after an output has been produced, while the wrapper keeps the final value up for the Microblaze. This wrapper finite state machine adds 30ns delay for the logical ANN to output a value, as you can see in the simulation results of Figure 5.27. In the Figure you can also see the *Weights Ready* signal from the wrapper being put to 0 when the *Data In Valid* is raised, and raised again after an output has been produced and the logic has been resetted.

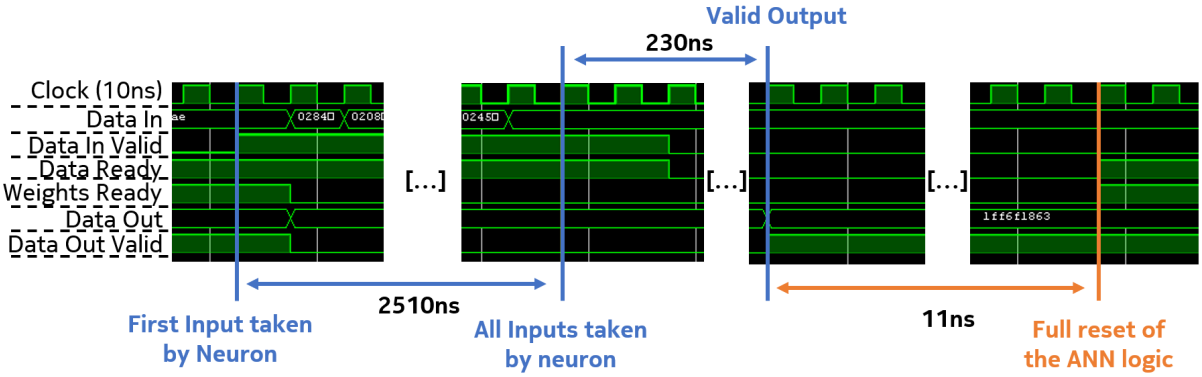


Figure 5.27 – Simulation extracts of the validation of the logical neural network wrapper.

Now that we have a way to communicate with the processor, we can integrate the Microblaze in our design. We represented the full design for our setup in Figure 5.28.

As we can see, between the Microblaze and the logical neural network we will implement AXI4-Stream FIFOs. They will help us for two reasons. For the output port of the network they will handle the *ready* signal of the Microblaze. For the input ports of the

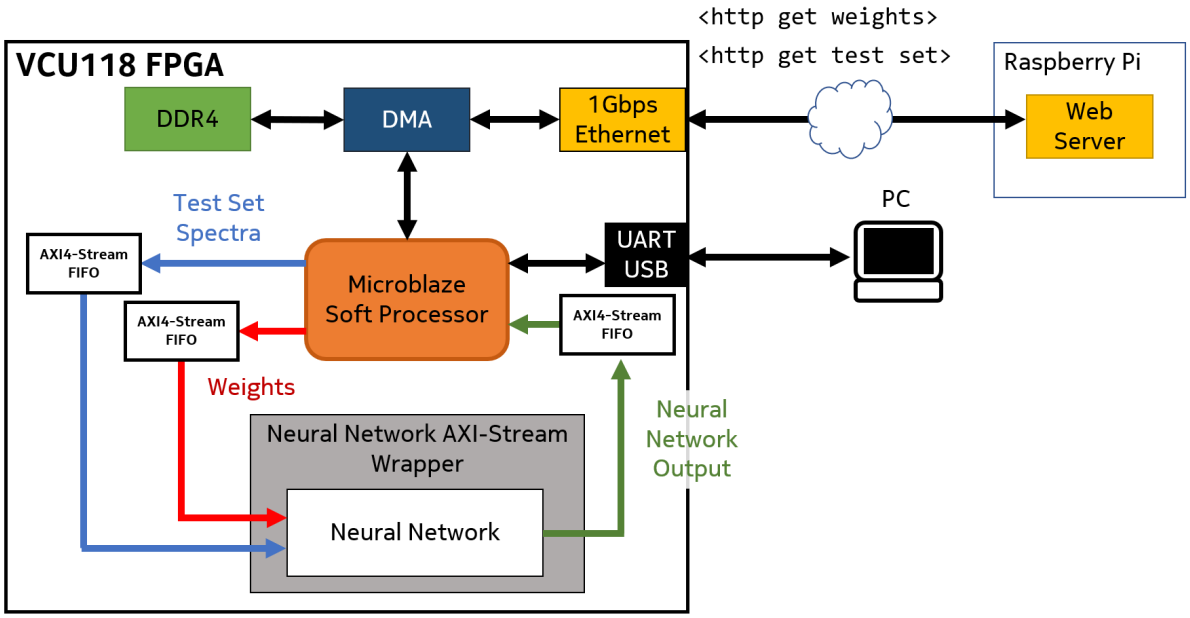


Figure 5.28 – Schematization of our setup with a Microblaze soft processor, the logical neural network and a web server to collect test sets and weights.

network, the FIFOs help concatenate the discontinuous flow of data from the Microblaze into continuous packets to send to the neural network. Also, we will deploy a Raspberry Pi with a web server to store the test sets and the weights the Microblaze will send to the logical ANN. . This means we need to implement an Ethernet core (at 1Gbps in our case) in the FPGA, coupled with Double-Data-Rate (DDR) RAM and a DMA (Direct Memory Access) to handle the transfer of data to and from the processor. The Microblaze is running the FreeRTOS real-time operating system, with the `lwip` library [140] to handle the TCP/IP stack and send HTTP request to the Raspberry Pi web server. An UART-USB link is deployed between the Microblaze and a computer to retrieve data from the logical processor.

5.2.4 Implementation and validation

We will now validate our setup, in Figure 5.29 we represented the main FreeRTOS tasks we developed for our scenario. The first task is to handle the acquisition of data and weights from the web server, the transmission to the neural network and the second to handle the retrieval of the data from the neural network. We will firstly acquire from the web server the test set associated with the weights that are set in the neural network at

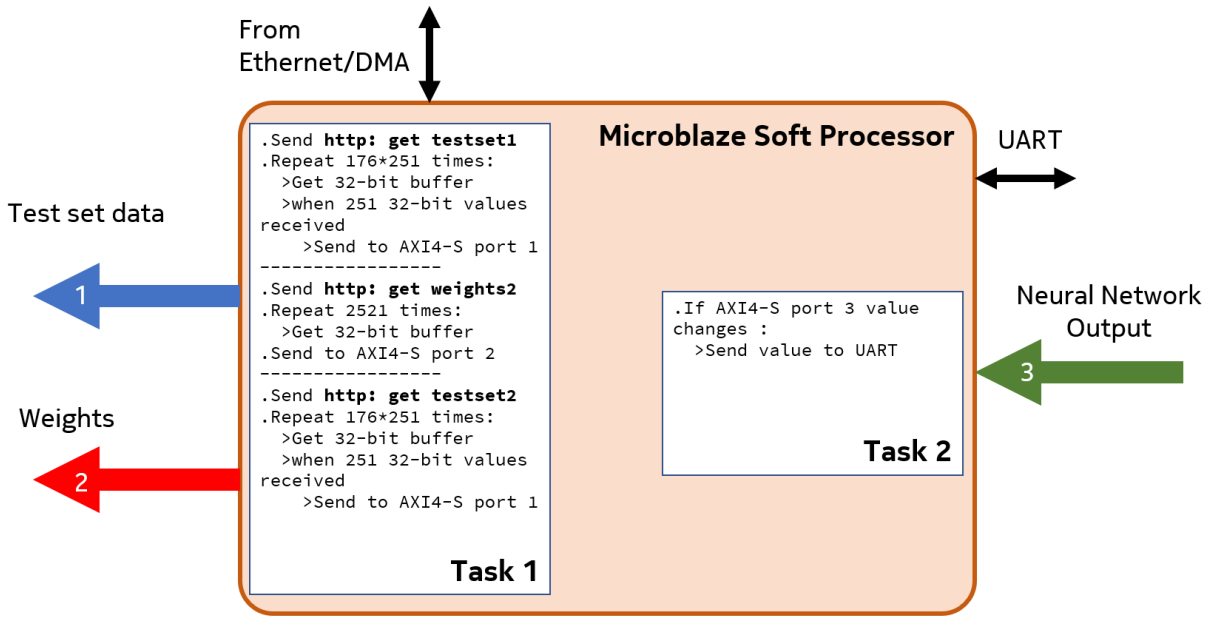


Figure 5.29 – Tasks developed in the Microblaze to handle our validation scenario.

implementation stage and send them to the neural network. We will call this first set the Initial Set. The weights associated with this Initial Set is from the neural network used in simulation and represented in Figures 5.13 through 5.17. After all test set spectra for this set has been sent to the network, we will fetch a new set of weights and send them to the ANN. We will then fetch the test set associated with this set of weights and do the same steps as previously. We will call this second test set the Final Set. As we send the test sets we will collect the values out of the neural network and send them to a PC via the UART-USB link. The test sets are written as 4 bytes hexadecimal unsigned fixed point values, with a single bit integer part. The weight sets are written in the Raspberry Pi test server and the FPGA as 2 bytes hexadecimal signed fixed point values, with 7 bits integer part. The activation function ROMs is 10 bits deep and 16 bits wide. The main logic is clocked with a 100MHz clock (10ns period). This gives processor ticks of 10ms for the Microblaze with FreeRTOS.

In Table 5.3 we gathered the validation and timing results from the Microblaze. We counted the number of values received from the neural network, their correspondence with the simulation results, and the time to acquire all spectra from the webserver and all the results from the neural network. First of all, we can see that we receive the correct number of values from the neural network, with 100% correspondence with the simulation results.

	Initial Set	Final Set
Number of values acquired from NN	176 (100%)	176 (100%)
Correspondence with simulation values	100%	100%
Ticks to receive whole test set from webserver	474	466
Average ticks to receive one spectrum	2.693	2.648
Ticks to get all NN values	475	466
Average ticks between NN values	2.699	2.648

Table 5.3 – Validation and timing results from the Microblaze processor. A tick corresponds to 10ms. Ticks to receive whole test sets ommits the time to receive the HTTP header from the web server (1 tick for Initial Set, 2 for Final Set).

The timing results show that the time to receive all 176 spectra from a test set from the web server is near identical to the time to receive all output values from the neural network. For the Initial Set, 474 ticks (4.74s) are necessary to receive the test set, and 475 ticks are necessary to receive all output data from the neural network. For the Final Set both actions require 466 processor ticks.

In Figure 5.30 we show an extract from a capture of the transfer of data to and from the neural network and the Microblaze. The capture was realized by an Integrated Logic Analyzer clocked at 300MHz (3.33ns period, this clock was solely used for debug purposes). As we can see, there are 69 clock period between the last input to the neural network and the output of the ANN, which gives roughly 230ns for the neural network to give an output, which is coherent with what we designed and validated in simulation in Section 5.2.3.

We have validated our logical neural network implementation, which can be reconfigured by changing the neuron weights remotely. We believe that this solution can be improved upon by allowing the reconfiguration of the number of neurons in the hidden layer and the number of inputs and outputs of the network and make this structure truly flexible. This logical neural network could also be validated by being deployed in a network testbed with a centralized control plane, with the control plane "activating" the neural network by sending the weights, the number of neurons and inputs if such features have been implemented, and other necessary info. The result of the neural network result

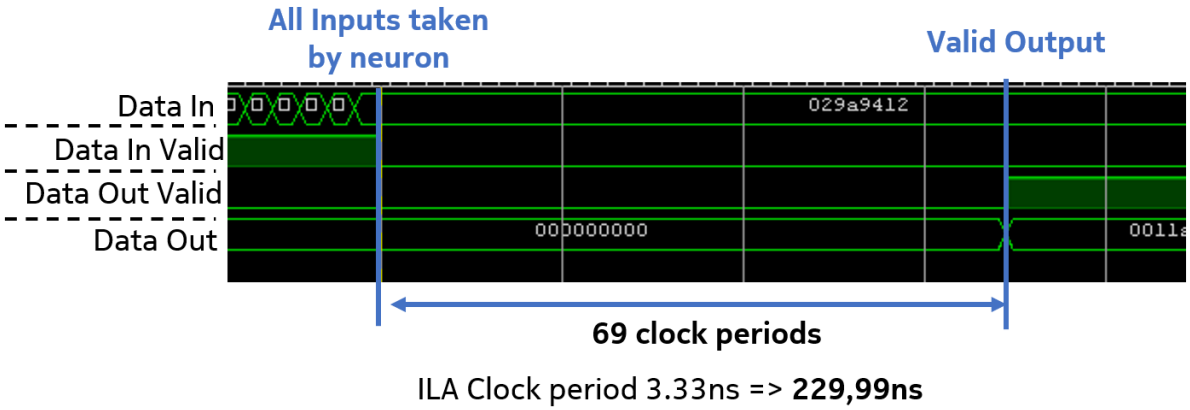


Figure 5.30 – Capture of the implemented neural network.

could be sent to the controller and could even trigger a reconfiguration of the optical network hardware, possibly using the auto-negotiation protocol developed in Chapter 4. This would show if and how this solution could accelerate the monitoring of the neural network by processing underutilized and data-heavy metrics using FPGA hardware deployed in the optical data plane. The solution shows promise to make neural networks more reliable and more reactive, and can be a stepping stone for fully automatic reconfigurable optical networks.

5.3 Conclusion

In this chapter we have developed and validated a logical ANN structure for FPGA. We based ourselves on an ANN-based monitoring function that from an optical spectrum outputs a launch power correction to have an ideal tradeoff between linear and nonlinear optical contributions and maximize the SNR for the network structure and we allowed for weights update to remotely change the function performed by the neural network. We validated the network implementation in terms of precision and timing. We believe this solution proves promising, as it will enable to provide advanced monitoring functions, exploit underutilized optical metrics, relieve the control plane of some monitoring and computing tasks, and would couple well with the auto-negotiation protocol presented in Chapter 4 to improve the reliability and flexibility of optical networks and optical hardware.

Chapter 6

Conclusion and perspectives

In this thesis work we have provided solutions to bring more intelligence to optical transponder in the context of elastic optical networks. We believe that adding decision making capabilities and the ability to realize synchronized change of parameters would increase the whole optical transport network flexibility. We also believe that the control plane could take advantage of the computational power we provide in the optical equipment and perform remote monitoring tasks to gain performance, notably on underused and data heavy metrics. The proposed solutions could be useful tools for the adoption of fully automated and highly flexible optical networks of the future.

In more details, our contributions were:

1. We designed and validated an auto-negotiation protocol for optical transponders, where changes of parameters can be negotiated, acknowledged and performed by inserting messages in the traffic. We deigned this protocol so that fast synchronized changes of parameters can be performed, with incompatibilities prevention and make possible changes of parameters with no loss of data.
2. We developed an embedded monitoring solution for commercial optical transponders, and studied the evolution of the received power and the evolution of the regression function line of this metric in case of a quick or slow degradation, so that we can trigger our auto-negotiation protocol at the appropriate time to restore the quality of service.
3. We integrated our auto-negotiation protocol and embedded monitoring setup with a commercial optical transponder and validated it inside a network testbed with a centralized control plane. We added a notification service in our setup to make our solution compatible with SDN optical control plane, and measure our setup in terms of performance in detecting a degradation and restoring the quality of service. Our solution proved roughly two times faster than the control plane and less time variable to perform these tasks, showcasing the possible time gains in providing close to the hardware intelligence.

4. We integrated our auto-negotiation solution with a real-time baudrate variable setup where we trigger a synchronized change of parameters.
5. We developed an FPGA-based neural network. We based ourselves on an ANN structure for nonlinear contributions monitoring, by taking optical spectra as inputs, and outputting a power correction to maximize the SNR. We validated it in terms of precision against a Matlab implementation.
6. We implemented a neuron weight update mechanism for our FPGA ANN, which enables the reconfiguration of the function performed by the neural network and remote monitoring by the control plane. We validated it using a web server to fetch both the inputs of our network and the corresponding set of weights.

This work can be extended in multiple ways, notably:

1. The auto-negotiation protocol can be utilized more thoroughly with the baudrate variable setup. It could be used to reconfigure more parameters than the baudrate, and we could integrate this setup inside a network testbed with a control plane. We could trigger different degradations and showcase that we are able to perform the right adaptation for the degradation to restore the quality of service.
2. We developed a notification solution for control plane interaction, but full data models development and integration with currently used protocols for data and control plane interactions would be a plus.
3. The FPGA-based neural network could be developed to be more reconfigurable. The reconfigurability of the weights is a start for full flexibility, but being able to reconfigure at least the number of inputs and the number of outputs would help greatly in allowing more use-cases for it. Ideally, the reconfigurability of the whole hidden layer, from the number of neurons, the activation functions and even the number of layers, would enable even more possibilities.
4. The neural network solution could be integrated with the embedded monitoring and auto-negotiation setup, and be tested in a network testbed environment to make real-time measurements, and trigger the auto-negotiation solution when a degradation is detected or an optimization of a transponder parameter can be performed. This would also be an opportunity to showcase the remote monitoring possibility for the control plane.
5. A solution for the weights distribution for the developed neural network could be developed, that could allow for example the deployment and update of the neural

network solution at multiple points of the network.

BIBLIOGRAPHY

[1] Cisco, "Cisco visual networking indec (2017-2022)." [Online; available 2021] https://oarklibrary.com/file/2/be52d44d-f201-475b-876d-6649efa14d85/e5235174-57e4-419c-a1bd-832c9fea6d07.pdf.

[2] P. Layec, A. Dupas, D. Verchère, K. Sparks, and S. Bigo, "Will metro networks be the playground for (true) elastic optical networks?," *Journal of Lightwave Technology*, vol. 35, p. 1260–1266, Mar 2017.

[3] I. T. union (ITU), "G.694.1 : spectral grids for wdm applications: Dwdm frequency grid." [Online; available 2021] https://www.itu.int/rec/T-REC-G.694.1/en.

[4] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, p. 114–119, Feb 2013.

[5] F. Paolucci, A. Sgambelluri, F. Cugini, and P. Castoldi, "Network telemetry streaming services in sdn-based disaggregated optical networks," *Journal of Lightwave Technology*, vol. 36, p. 3142–3149, Aug 2018.

[6] R. Muñoz, R. Vilalta, R. Casellas, R. Martínez, T. Szyrkowiec, A. Autenrieth, V. Lopez, and D. Lopez, "Sdn/nfv orchestration for dynamic deployment of virtual sdn controllers as vnf for multi-tenant optical networks," in *Optical Fiber Communication Conference (2015), paper W4J.5*, p. W4J.5, Optical Society of America, Mar 2015.

[7] M. Jinno, H. Takara, Y. Sone, K. Yonenaga, and A. Hirano, "Multiflow optical transponder for efficient multilayer optical networking," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 56–65, 2012.

[8] E. Riccardi, P. Gunning, O. G. de Dios, M. Quagliotti, V. Lopez, and A. Lord, "An operator view on the introduction of white boxes into optical networks," *Journal of Lightwave Technology*, vol. 36, no. 15, pp. 3062–3072, 2018.

[9] K. Christodoulopoulos, C. Delezoide, N. Sambo, A. Kretsis, I. Sartzetakis, A. Sgambelluri, N. Argyris, G. Kanakis, P. Giardina, G. Bernini, and et al., "Toward efficient, reliable, and autonomous optical networks: the orchestra solution [in-

vited]," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, p. C10–C24, Sep 2019.

[10] L. Velasco and M. Ruiz, "Optical network automation [invited tutorial]," in *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, p. 1–4, Jul 2020.

[11] Xilinx, "Microblaze soft processor core." [Online; avalaible 2021] https://www.xilinx.com/products/design-tools/microblaze.html.

[12] FreeRTOS, "Real-time operating system for microcontrollers." [Online; available 2021] https://www.freertos.org/.

[13] C. S. Turner, "Slope filtering: An fir approach to linear regression," *IEEE Signal Processing Magazine*, vol. 25, p. 159–163, Nov 2008.

[14] D. Rafique and L. Velasco, "Machine learning for network automation: Overview, architecture, and applications [invited tutorial]," *Journal of Optical Communications and Networking*, vol. 10, p. D126–D143, Oct 2018.

[15] M. Lonardi, J. Pesic, P. Jennevé, P. Ramantanis, N. Rossi, A. Ghazisaeidi, and S. Bigo, "Optical nonlinearity monitoring and launch power optimization by artificial neural networks," *Journal of Lightwave Technology*, vol. 38, p. 2637–2645, May 2020.

[16] Cisco, "Cisco annual internet report (2018-2023)." [Online; available 2021] https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[17] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "Implications of the covid-19 pandemic on the internet traffic," in *Broadband Coverage in Germany; 15th ITG-Symposium*, p. 1–5, Mar 2021.

[18] F. P. Kapron, D. B. Keck, and R. D. Maurer, "Radiation losses in glass optical waveguides," *Applied Physics Letters*, vol. 17, p. 423–425, Nov 1970.

[19] M.-J. Li, "Optical fiber evolution over the past 5 decades," in *Frontiers in Optics / Laser Science (2020), paper FM4D.1*, p. FM4D.1, Optical Society of America, Sep 2020.

[20] H. Sun, K.-T. Wu, and K. Roberts, "Real-time measurements of a 40 gb/s coherent system," *Optics Express*, vol. 16, p. 873–879, Jan 2008.

[21] International Telecom Union (ITU), "G.709 : Interfaces for the optical transport network." [Online; available 2021] https://www.itu.int/rec/T-REC-G.709/en.

[22] International Telecom Union (ITU), "Series G: Transmission systems and media, digital systems and networks, supplement 43: Transport of IEEE 10GBASE-R in optical transport networks (OTN)." [Online; available 2021] $https : //www.itu.int/rec/dologin_pub.asp?lang = e\&id = T-REC-G.Sup43-201102-I!!PDF-E\&type = items$.

[23] M. Jinno, "Elastic optical networking: Roles and benefits in beyond 100-gb/s era," *Journal of Lightwave Technology*, vol. 35, p. 1116–1124, Mar 2017.

[24] B. C. Chatterjee, N. Sarma, and E. Oki, "Routing and spectrum allocation in elastic optical networks: A tutorial," *IEEE Communications Surveys Tutorials*, vol. 17, no. 3, p. 1776–1800, 2015.

[25] Y. R. Zhou and K. Smith, "Practical innovations enabling scalable optical transmission networks: Real-world trials and experiences of advanced technologies in field deployed optical networks," *Journal of Lightwave Technology*, vol. 38, p. 3106–3113, Jun 2020.

[26] G. Bosco, V. Curri, A. Carena, P. Poggiolini, and F. Forghieri, "On the performance of nyquist-wdm terabit superchannels based on pm-bpsk, pm-qpsk, pm-8qam or pm-16qam subcarriers," *Journal of Lightwave Technology*, vol. 29, p. 53–61, Jan 2011.

[27] J. M. Simmons, *Optical Network Design and Planning*. Springer, May 2014. Google-Books-ID: SlclBAAAQBAJ.

[28] P. G. Arbués, C. M. Machuca, and A. Tzanakaki, "Comparative study of existing oadm and oxc architectures and technologies from the failure behavior perspective," *Journal of Optical Networking*, vol. 6, p. 123–133, Feb 2007.

[29] L. Zong, G. N. Liu, H. Zhao, T. Ma, and A. Lord, "Ultra-compact contentionless roadm architecture with high resilience based on flexible wavelength router," in *Optical Fiber Conference (OFC) 2014*, p. 1–3, Mar 2014.

[30] L. Lu, Y. Li, L. Zong, B. Mukherjee, and G. Shen, "Asymmetric cdc roadms for efficient support of bi-directionally asymmetric traffic demands," *Journal of Lightwave Technology*, vol. 39, p. 4572–4583, Jul 2021.

[31] A. Castro, L. Velasco, M. Ruiz, and J. Comellas, "Single-path provisioning with multi-path recovery in flexgrid optical networks," in *2012 IV International Congress on Ultra Modern Telecommunications and Control Systems*, p. 745–751, Oct 2012.

[32] A. Mecozzi and R.-J. Essiambre, "Nonlinear shannon limit in pseudolinear coherent systems," *Journal of Lightwave Technology*, vol. 30, p. 2011–2024, Jun 2012.

[33] H. Yang, P. Wright, B. Robertson, P. Wilkinson, P. R. Dolan, A. Lord, and D. Chu, "Impact of wss filtering penalty on the capacity of elastic wdm ring optical networks," in *Optical Fiber Communication Conference*, p. Th2A.43, OSA, 2018.

[34] R. Alvizu and G. Maier, "Can open flow make transport networks smarter and dynamic? an overview on transport sdn," in *2014 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, p. 1–6, Jun 2014.

[35] S. Dizdarević, H. Dizdarević, M. Škrbić, and N. Hadžiahmetović, "A survey on transition from gmpls control plane for optical multilayer networks to sdn control plane," in *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, p. 537–544, May 2016.

[36] L. Gifre, J.-L. Izquierdo-Zaragoza, M. Ruiz, and L. Velasco, "Autonomic disaggregated multilayer networking," *Journal of Optical Communications and Networking*, vol. 10, p. 482, May 2018.

[37] L. Gifre, F. Boitier, C. Delezoide, M. Ruiz, M. Buffa, A. Morea, R. Casellas, L. Velasco, and P. Layec, "Demonstration of monitoring and data analytics-triggered reconfiguration in partially disaggregated optical networks," *Optical Fiber Communication Conference (OFC) 2020 (2020), paper M3Z.19*, p. M3Z.19, Mar 2020.

[38] Z. Dong, F. N. Khan, Q. Sui, K. Zhong, C. Lu, and A. P. T. Lau, "Optical performance monitoring: A review of current and future technologies," *Journal of Lightwave Technology*, vol. 34, p. 525–543, Jan 2016.

[39] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, "An overview on application of machine learning techniques in optical networks," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, p. 1383–1408, 2019.

[40] F. N. Khan, C. Lu, and A. P. T. Lau, "Optical performance monitoring in fiber-optic networks enabled by machine learning techniques," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, p. 1–3, Mar 2018.

[41] T. Panayiotou, G. Savva, B. Shariati, I. Tomkos, and G. Ellinas, "Machine learning for qot estimation of unseen optical network states," in *Optical Fiber Communication Conference (OFC) 2019*, p. Tu2E.2, OSA, 2019.

[42] T. Tanaka, S. Kuwabara, H. Nishizawa, T. Inui, S. Kobayashi, and A. Hirano, "Field demonstration of real-time optical network diagnosis using deep neural network

and telemetry," in *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, p. 1–3, Mar 2019.

[43] E. Giacoumidis, Y. Lin, M. Blott, and L. P. Barry, "Real-time machine learning based fiber-induced nonlinearity compensation in energy-efficient coherent optical networks," *APL Photonics*, vol. 5, p. 041301, Apr 2020.

[44] S. Varughese, J. Langston, V. A. Thomas, S. Tibuleac, and S. E. Ralph, "Implementing dacs in high speed optical link simulations," in *Advanced Photonics 2017 (IPR, NOMA, Sensors, Networks, SPPCom, PS)*, p. SpTh1F.2, OSA, 2017.

[45] M. Birk, P. Gerard, R. Curto, L. E. Nelson, X. Zhou, P. Magill, T. J. Schmidt, C. Malouin, B. Zhang, E. Ibragimov, and et al., "Real-time single-carrier coherent 100 gb/s pm-qpsk field trial," *Journal of Lightwave Technology*, vol. 29, p. 417–425, Feb 2011.

[46] C. Li, Z. Zhang, J. Chen, T. Ding, Z. Xiao, F. Shah, J. Mitra, H. Xiang, and X. Cui, "Advanced dsp for single-carrier 400-gb/s pdm-16qam," in *Optical Fiber Communication Conference (2016)*, p. W4A.4, OSA, 2016.

[47] G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris, and I. Tomkos, "A survey on fec codes for 100 g and beyond optical networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, p. 209–221, 2016.

[48] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: Fec for 100 gb/s otn," *Journal of Lightwave Technology*, vol. 30, p. 110–117, Jan 2012. arXiv: 1201.4106.

[49] I. Djordjevic, O. Milenkovic, and B. Vasic, "Generalized low-density parity-check codes for optical communication systems," *Journal of Lightwave Technology*, vol. 23, no. 5, pp. 1939–1946, 2005.

[50] L. Schmalen, D. Suikat, D. Rösener, V. Aref, A. Leven, and S. ten Brink, "Spatially coupled codes and optical fiber communications: An ideal match?," in *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 460–464, 2015.

[51] D. Rafique, T. Rahman, A. Napoli, and B. Spinnler, "Digital pre-emphasis in optical communication systems: On the nonlinear performance," *Journal of Lightwave Technology*, vol. 33, p. 140–150, Jan 2015.

[52] D. Rafique, N. Eiselt, H. Griesser, B. Wohlfeil, M. Eiselt, and J.-P. Elbers, "Digital pre-emphasis based system design trade-offs for 64 gbaud coherent data center inter-

connects," in *2017 19th International Conference on Transparent Optical Networks (ICTON)*, p. 1–4, Jul 2017.

[53] A. Napoli, P. W. Berenguer, T. Rahman, G. Khanna, M. M. Mezghanni, L. Gardian, E. Riccardi, A. C. Piat, S. Calabrò, S. Dris, and et al., "Digital pre-compensation techniques enabling high-capacity bandwidth variable transponders," *Optics Communications*, vol. 409, p. 52–65, Feb 2018.

[54] G. Khanna, S. Calabrò, B. Spinnler, E. De Man, and N. Hanik, "Joint adaptive pre-compensation of transmitter i/q skew and frequency response for high order modulation formats and high baud rates," in *Optical Fiber Communication Conference*, p. M2G.4, OSA, 2015.

[55] K. Kikuchi, "Polarization-demultiplexing algorithm in the digital coherent receiver," in *2008 Digest of the IEEE/LEOS Summer Topical Meetings*, p. 101–102, Jul 2008.

[56] I. Fatadin, D. Ives, and S. Savory, "Blind equalization and carrier phase recovery in a 16-qam optical coherent system," *Journal of Lightwave Technology*, vol. 27, p. 3042–3049, Aug 2009.

[57] M. S. Faruk and S. J. Savory, "Digital signal processing for coherent transceivers employing multilevel formats," *Journal of Lightwave Technology*, vol. 35, p. 1125–1141, Mar 2017.

[58] M. Pajovic, D. S. Millar, T. Koike-Akino, R. Maher, D. Lavery, A. Alvarado, M. Paskov, K. Kojima, K. Parsons, B. C. Thomsen, and et al., "Experimental demonstration of multi-pilot aided carrier phase estimation for dp-64qam and dp-256qam," in *2015 European Conference on Optical Communication (ECOC)*, p. 1–3, Sep 2015.

[59] R. A. Griffin, S. K. Jones, N. Whitbread, S. C. Heck, and L. N. Langley, "Inp mach–zehnder modulator platform for 10/40/100/200-gb/s operation," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 19, p. 158–166, Nov 2013.

[60] X. Chen and S. Chandrasekhar, *Springer Handbook of Optical Networks*, ch. Optical Transponder Components, p. 137–154. Springer International Publishing, 2020.

[61] Small Form Factor Committee, "Sfp (small formfactor pluggable) transceiver," 2001. [Online; avalaible 2021] https://members.snia.org/document/dl/26184.

[62] CFP Multi-Source Agreement, "Cfp specifications." [Online; available 2022] https://www.cfp-msa.org.

[63] A. Akrami, M. Doostizadeh, and F. Aminifar, "Power system flexibility: an overview of emergence to evolution," *Journal of Modern Power Systems and Clean Energy*, vol. 7, no. 5, pp. 987–1007, 2019.

[64] M. K. A. Rahim, M. R. Hamid, N. A. Samsuri, N. A. Murad, M. F. M. Yusoff, and H. A. Majid, "Frequency reconfigurable antenna for future wireless communication system," in *2016 46th European Microwave Conference (EuMC)*, p. 965–970, Oct 2016.

[65] A. Olewnik and K. Lewis, "A decision support framework for flexible system design," *Journal of Engineering Design*, vol. 17, p. 75–97, Jan 2006.

[66] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Computing Surveys*, vol. 52, p. 1–39, Jan 2020.

[67] T. Nowatzki, V. Gangadhan, K. Sankaralingam, and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 27–39, 2016.

[68] F. Berthelot, F. Nouvel, and D. Houzet, "A flexible system level design methodology targeting run-time reconfigurable fpgas," *EURASIP Journal on Embedded Systems*, vol. 2008, p. 1–18, 2008.

[69] R. Tessier, K. Pocek, and A. DeHon, "Reconfigurable computing architectures," *Proceedings of the IEEE*, vol. 103, p. 332–354, Mar 2015.

[70] Xilinx, "7 series fpgas configurable logic block user guide (ug474)," 2016. [Online; available 2021] https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf.

[71] I. Tomkos, S. Azodolmolky, J. Solé-Pareta, D. Careglio, and E. Palkopoulou, "A tutorial on the flexible optical networking paradigm: State of the art, trends, and research challenges," *Proceedings of the IEEE*, vol. 102, p. 1317–1337, Sep 2014.

[72] R. Pastorelli, G. Bosco, S. Piciaccia, and F. Forghieri, "Network planning strategies for next-generation flexible optical networks [invited]," *Journal of Optical Communications and Networking*, vol. 7, p. A511, Mar 2015.

[73] R. Casellas, R. Martínez, R. Vilalta, and R. Muñoz, "Control, management, and orchestration of optical networks: Evolution, trends, and challenges," *Journal of Lightwave Technology*, vol. 36, p. 1390–1402, Apr 2018.

[74] Internet Engineering Task Force, "Yang - a data modeling language for the network configuration protocol (netconf)." [Online; available 2022] https://datatracker.ietf.org/doc/rfc6020.

[75] J. Akhtar, "Yang modeling of network elements for the management and monitoring of elastic optical networks," in *2015 IEEE International Conference on Telecommunications and Photonics (ICTP)*, p. 1–5, Dec 2015.

[76] OpenConfig, "Vendor neutral, model-driven network management." [Online; available 2021] https://www.openconfig.net.

[77] OpenROADM, "The open roadm multi-source agreement (msa) defines interoperability specifications for reconfigurable optical add/drop multiplexers (roadm)." [Online; available 2021] http://openroadm.org/.

[78] T. Szyrkowiec, A. Autenrieth, and W. Kellerer, "Optical network models and their application to software-defined network management," *International Journal of Optics*, vol. 2017, p. 1–9, 2017.

[79] Internet Engineering Task Force, "Network configuration protocol (netconf)." [Online; available 2021] https://datatracker.ietf.org/doc/html/rfc6241.

[80] gRPC, "A high performance, open source universal rpc framework." [Online; available 2021] https://grpc.io/.

[81] gNMI, "Unified management protocol for streaming telemetry and configuration management that leverages the open source grpc framework.." [Online; available 2021] https://github.com/openconfig/reference/tree/master/rpc/gnmi.

[82] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[83] T. Lan, Q. Han, H. Fan, and J. Lan, "Fpga-based packets processing acceleration platform for vnf," in *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 314–317, 2017.

[84] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet, "Vnf-aapc: Accelerator-aware vnf placement and chaining," *Computer Networks*, vol. 177, p. 107329, 2020.

[85] N. Rossi, T. Zami, and J. Pesic, "Does ageing of margins become more attractive in wdm networks with 64 gbaud transponders and regenerators?," in *45th European Conference on Optical Communication (ECOC 2019)*, p. 1–4, Sep 2019.

[86] M. Imran, P. M. Anandarajah, A. Kaszubowska-Anandarajah, N. Sambo, and L. Potí, "A survey of optical carrier generation techniques for terabit capacity elastic optical networks," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, p. 211–263, 2018.

[87] A. Napoli, M. Bohn, D. Rafique, A. Stavdas, N. Sambo, L. Poti, M. Nölle, J. K. Fischer, E. Riccardi, A. Pagano, A. Di Giglio, M. S. Moreolo, J. M. Fabrega, E. Hugues-Salas, G. Zervas, D. Simeonidou, P. Layec, A. D'Errico, T. Rahman, and J. P. F.-P. Giménez, "Next generation elastic optical networks: The vision of the european research project idealist," *IEEE Communications Magazine*, vol. 53, p. 152–162, Feb 2015.

[88] H. Y. Choi, T. Tsuritani, and I. Morita, "Ber-adaptive flexible-format transmitter for elastic optical networks," *Optics Express*, vol. 20, p. 18652–18658, Aug 2012.

[89] H. Y. Choi, T. Tsuritani, and I. Morita, "Multi-format and multi-rate transmitter for flexible and elastic optical networks," in *2012 17th Opto-Electronics and Communications Conference*, p. 773–774, Jul 2012.

[90] H. Takara, T. Goh, K. Shibahara, K. Yonenaga, S. Kawai, and M. Jinno, "Experimental demonstration of 400 gb/s multi-flow, multi-rate, multi-reach optical transmitter for efficient elastic spectral routing," in *37th European Conference and Exposition on Optical Communications (2011), paper Tu.5.A.4*, p. Tu.5.A.4, Optical Society of America, Sep 2011.

[91] S. Yan, A. F. Beldachi, F. Qian, K. Kondepu, Y. Yan, C. Jackson, R. Nejabati, and D. Simeonidou, "Demonstration of real-time modulation-adaptable transmitter," in *2017 European Conference on Optical Communication (ECOC)*, p. 1–3, Sep 2017.

[92] R. Schmogrow, D. Hillerkuss, M. Dreschmann, M. Huebner, M. Winter, J. Meyer, B. Nebendahl, C. Koos, J. Becker, W. Freude, and et al., "Real-time software-defined multiformat transmitter generating 64qam at 28 gbd," *IEEE Photonics Technology Letters*, vol. 22, p. 1601–1603, Nov 2010.

[93] Y. R. Zhou, K. v. Smith, S. West, M. Johnston, J. Weatherhead, P. Weir, J. Hammond, A. Lord, J. Chen, W. Pan, and et al., "Field trial demonstration of real-time optical superchannel transport up to 5.6 tb/s over 359 km and 2 tb/s over a live 727 km flexible grid optical link using 64 gbaud software configurable transponders," *Journal of Lightwave Technology*, vol. 35, p. 499–505, Feb 2017.

[94] B. Baeuerle, A. Josten, F. Abrecht, M. Eppenberger, E. Dornbierer, D. Hillerkuss, and J. Leuthold, "Multi-format carrier recovery for coherent real-time reception with processing in polar coordinates," *Optics Express*, vol. 24, p. 25629–25640, Oct 2016.

[95] A. Napoli, M. Nölle, D. Rafique, J. K. Fischer, B. Spinnler, T. Rahman, M. M. Mezghanni, and M. Bohn, "On the next generation bandwidth variable transponders for future flexible optical systems," in *2014 European Conference on Networks and Communications (EuCNC)*, pp. 1–5, 2014.

[96] C. Dorize, P. Layec, and G. Charlet, "Dsp power balancing for multi-format wdm receiver," in *2014 The European Conference on Optical Communication (ECOC)*, pp. 1–3, 2014.

[97] C. Kachris, G. Tzimpragos, D. Soudris, and I. Tomkos, "Reconfigurable fec codes for software-defined optical transceivers," in *2014 13th International Conference on Optical Communications and Networks (ICOCN)*, p. 1–4, Nov 2014.

[98] M. Jinno, H. Takara, B. Kozicki, Y. Tsukishima, Y. Sone, and S. Matsuoka, "Spectrum-efficient and scalable elastic optical path network: architecture, benefits, and enabling technologies," *IEEE Communications Magazine*, vol. 47, no. 11, pp. 66–73, 2009.

[99] N. Sambo, P. Castoldi, A. D'Errico, E. Riccardi, A. Pagano, M. S. Moreolo, J. M. Fàbrega, D. Rafique, A. Napoli, S. Frigerio, and et al., "Next generation sliceable bandwidth variable transponders," *IEEE Communications Magazine*, vol. 53, p. 163–171, Feb 2015.

[100] T. Tanaka, A. Hirano, and M. Jinno, "Impact of transponder architecture on the scalability of optical nodes in elastic optical networks," *IEEE Communications Letters*, vol. 17, no. 9, pp. 1846–1848, 2013.

[101] V. Lopez, A. González De Dios, O. Gerstel, N. Amaya, G. Zervas, D. Simeonidou, and J. P. Fernandez-Palacios, "Target cost for sliceable bandwidth variable transponders in a real core network," in *2013 Future Network Mobile Summit*, pp. 1–9, 2013.

[102] J. P. Fernandez-Palacios, V. Lopez, B. Cruz, and O. G. de Dios, "Elastic optical networking: An operators perspective," in *2014 The European Conference on Optical Communication (ECOC)*, pp. 1–3, 2014.

[103] Y. Ou, A. Hammad, S. Peng, R. Nejabati, and D. Simeonidou, "Online and offline virtualization of optical transceiver," *Journal of Optical Communications and Networking*, vol. 7, p. 748–760, Aug 2015.

[104] R. Martínez, R. Casellas, M. S. Moreolo, J. M. Fabrega, R. Vilalta, R. M. noz, L. Nadal, and J. P. Fernández-Palacios, "Proof-of-concept validation of sdn-controlled vcsel-based s-bvts in flexi-grid optical metro networks," in *Optical Fiber Communication Conference (OFC) 2019*, p. W1G.5, Optical Society of America, 2019.

[105] N. Sambo, G. Meloni, F. Paolucci, M. Imran, F. Fresi, F. Cugini, P. Castoldi, and L. Poti, "First demonstration of sdn-controlled sbvt based on multi-wavelength source with programmable and asymmetric channel spacing," in *2014 The European Conference on Optical Communication (ECOC)*, pp. 1–3, 2014.

[106] V. Katopodis, D. Felipe, C. Tsokos, P. Groumas, M. Spyropoulou, A. Beretta, A. Dede, M. Quagliotti, A. Pagano, A. Vannucci, N. Keil, H. Avramopoulos, and C. Kouloumentas, "Multi-flow transmitter based on polarization and optical carrier management on optical polymers," *IEEE Photonics Technology Letters*, vol. 28, no. 11, pp. 1169–1172, 2016.

[107] Ujjwal and T. Jaisingh, "Design and development of a new architecture of sliceable bandwidth variable transponder," *Opto-Electronics Review*, vol. 25, p. 46–53, May 2017.

[108] L. Velasco, A. Castro, A. Asensio, M. Ruiz, G. Liu, C. Qin, R. Proietti, and S. J. B. Yoo, "Meeting the requirements to deploy cloud ran over optical networks," *J. Opt. Commun. Netw.*, vol. 9, pp. B22–B32, Mar 2017.

[109] Telecom Infra Project (TIP) Open Optical and Packet Transport (OOPT) working group, "Tai (transponder abstraction interface) repository and documentation." [Online; available 2022] https://github.com/Telecominfraproject/oopt-tai.

[110] N. Sambo, K. Christodoulopoulos, N. Argyris, P. Giardina, C. Delezoide, A. Sgambelluri, A. Kretsis, G. Kanakis, F. Fresi, G. Bernini, and et al., "Experimental demonstration of a fully disaggregated and automated white box comprised of different types of transponders and monitors," *Journal of Lightwave Technology*, vol. 37, p. 824–830, Feb 2019.

[111] G. Francia, R. Nagase, W. Ishida, Y. Sone, L. Kumar, S. Krishnamohan, and V. Lopez, "Disaggregated packet transponder field demonstration exercising multi-

format transmission with multi-vendor, open packet optical network elements," in *Optical Fiber Communication Conference (OFC) 2020*, p. Th3A.1, OSA, 2020.

[112] M. Scaffardi, V. Vercesi, A. Sgambelluri, and A. Bogoni, "Hitless reconfiguration of a ppln-based multiwavelength source for elastic optical networks," *Journal of Optical Communications and Networking*, vol. 8, p. 85, Feb 2016.

[113] A. Dupas, P. Layec, D. Verchere, V. Quan Pham, and S. Bigo, "Ultra-fast hitless 100gbit/s real-time bandwidth variable transmitter with sdn optical control," in *Optical Fiber Conference (OFC) 2018*, Nov 2018.

[114] V. N. Rozental and D. A. A. Mello, "Hitless rate switching for dynamically reconfigurable optical systems," *IEEE Photonics Journal*, vol. 7, no. 2, pp. 1–9, 2015.

[115] N. Sambo, A. Giorgetti, F. Cugini, and P. Castoldi, "Sliceable transponders: Preprogrammed oam, control, and management," *Journal of Lightwave Technology*, vol. 36, p. 1403–1410, Apr 2018.

[116] B. Spinnler, A. X. Lindgren, U. Andersen, S. Melin, J. Slovak, W. Schairer, K. Pulverer, J. Mårtensson, E. De Man, G. Khanna, and et al., "Autonomous intelligent transponder enabling adaptive network optimization in a live network field trial," *Journal of Optical Communications and Networking*, vol. 11, p. C1, Sep 2019.

[117] Z. Zhang and C. Li, "Hitless multi-rate coherent transceiver," in *Advanced Photonics 2015 (2015), paper SpS3D.2*, p. SpS3D.2, Optical Society of America, Jun 2015.

[118] V. N. Rozental, G. Bruno, M. Camera, and D. A. A. Mello, "Novel equalizer architecture for hitless rate switching in energy-efficient optical systems," in *OFC 2014*, p. 1–3, Mar 2014.

[119] D. W. Boertjes, M. Reimer, and D. Côté, "Practical considerations for near-zero margin network design and deployment [invited]," *Journal of Optical Communications and Networking*, vol. 11, p. C25–C34, Sep 2019.

[120] P. Ramantanis, C. Delezoide, P. Layec, and S. Bigo, "Revisiting the calculation of performance margins in monitoring-enabled optical networks," *Journal of Optical Communications and Networking*, vol. 11, p. C67–C75, Oct 2019.

[121] S. Yan, A. Aguado, Y. Ou, R. Wang, R. Nejabati, and D. Simeonidou, "Multilayer network analytics with sdn-based monitoring framework," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 9, p. A271–A279, Feb 2017.

[122] L. Velasco, L. Gifre, J.-L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, and F. Cugini, "An architecture to support autonomic slice networking," *Journal of Lightwave Technology*, vol. 36, p. 135–141, Jan 2018.

[123] J. E. Simsarian, M. N. Hall, G. Hosangadi, J. Gripp, W. v. Raemdonck, J. Yu, and T. Sizer, "Stream processing for optical network monitoring with streaming telemetry and video analytics," in *2020 European Conference on Optical Communications (ECOC)*, p. 1–4, Dec 2020.

[124] N. Sambo, A. Sgambelluri, F. Cugini, A. D'Errico, and P. Castoldi, "Enabling locally automated reconfigurations in disaggregated networks," in *2019 24th OptoElectronics and Communications Conference (OECC) and 2019 International Conference on Photonics in Switching and Computing (PSC)*, p. 1–3, Jul 2019.

[125] Excel Support Team, "Slope function." [Online; available 2021] https://support.microsoft.com/en-us/office/slope-function-11fb8f97-3117-4813-98aa-61d7e01276b9.

[126] C. Delezoide, P. Ramantanis, and P. Layec, "Leveraging field data for the joint optimization of capacity and availability in low-margin optical networks," *Journal of Lightwave Technology*, vol. 38, p. 6709–6718, Dec 2020.

[127] A. Dupas, D. Verchere, Q. Pham Van, P. Layec, L. Brameriet, A. Carer, B. Haentjens, and E. Le Rouzic, "Easy optical defragmentation with sdn controlled tunable transmitter," in *2018 European Conference on Optical Communication (ECOC)*, p. 1–3, Sep 2018.

[128] S. J. Savory and D. S. Millar, *Springer Handbook Of Optical Networks*, ch. DSP for Optical Transponders, p. 156–174. Springer Handbooks, Springer International Publishing, 2020.

[129] C. Edwin K. P. and Z. Stanislaw H., *An Introduction to Optimization*, ch. Gradient Methods, p. 125–153. Wiley, 2008.

[130] E. Dutisseuil, A. Dupas, A. Gouin, F. Boitier, and P. Layec, "Hitless transmission baud rate switching in a real-time transponder assisted by an auto-negotiation protocol," in *Optical Fiber Communication Conference (2022)*, p. 4, OSA, 2022. (Accepted, not published [Jan. 2022]).

[131] T. Tanaka, T. Inui, S. Kawai, S. Kuwabara, and H. Nishizawa, "Monitoring and diagnostic technologies using deep neural networks for predictive optical network

maintenance [invited]," *Journal of Optical Communications and Networking*, vol. 13, p. E13–E22, Oct 2021.

[132] A. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, p. 930–945, May 1993.

[133] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, A. M. Umar, O. U. Linus, H. Arshad, A. A. Kazaure, U. Gana, and M. U. Kiru, "Comprehensive review of artificial neural network applications to pattern recognition," *IEEE Access*, vol. 7, p. 158820–158846, 2019.

[134] C. C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, ch. An Introduction to Neural Networks, pp. 1–52. Springer International Publishing, 2018.

[135] A. Bononi, N. Rossi, and P. Serena, "On the nonlinear threshold versus distance in long-haul highly-dispersive coherent systems," *Opt. Express*, vol. 20, pp. B204–B216, Dec 2012.

[136] TensorFlow, "An end-to-end open source machine learning platform." [Online; available 2022] https://www.tensorflow.org/.

[137] Mathworks, "Fitnet: Function fitting neural network." [Online; available 2022] https://www.mathworks.com/help/deeplearning/ref/fitnet.html.

[138] M. Hagan and M. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, p. 989–993, Nov 1994.

[139] Arm Limited, "Amba axi-stream protocol specification." [Online; available 2022] https://developer.arm.com/documentation/ihi0051/b/.

[140] Free Software Foundation, Inc., "lwip - a lightweight tcp/ip stack." [Online; available 2022] https://savannah.nongnu.org/projects/lwip/.

DOCTORAT BRETAGNE LOIRE MATHSTIC

INSA RENNES

Titre : Transpondeurs temps-réels flexibles pour les télécommunications optiques et virtualisés.

Mot clés : transpondeur optique, temps-réel, flexibilité, monitoring, virtualisation

Résumé :

Les réseaux de transport optique subissent énormément de pression pour satisfaire les besoins croissants en capacité et en réduction de latence. Ces dernières années, l'adoption des concepts de flexibilité et de virtualisation a permis d'augmenter le débit et la réactivité du réseau.

Dans ce manuscript, nous nous concentrons sur le transpondeur optique. Les développements récents ont rendu cet équipement flexible et virtualisable, mais les reconfigurations forcent la mise en arrêt temporaire du transpondeur et demeurent encore rares. L'hyper concentration des prises de décision dans le plan de contrôle centralisé réduit aussi la réactivité du réseau en cas de dégradation du lien optique.

Tout d'abord nous présentons un protocole d'auto-négociation pour transpondeur optique qui permet des changements de paramètres rapides et synchronisés. Intégré avec un transpondeur commercial et une solution de monitoring et de prise de décision embarquée, nous avons validé notre solution face à un plan de contrôle centralisé dans un banc de test réseau.

Ensuite nous présentons ne solution de monitoring embarqué employant un réseau de neurones pour le plan de données optiques. Nous permettons avec notre solution de reconfigurer le poids des neurones et nous pensons que cette solution peut permettre des détection et résolutions de dégradation plus intelligentes et mettre a disposition du plan de contrôle des ressources computationnelles pour accélérer la prise de décision dans l'entièreté du réseau.

Title: Real-time flexible and virtualized transponders for optical telecommunications.

Keywords: optical transponder, real-time, flexibility, monitoring, virtualization

Abstract: Optical transport networks are under a lot of pressure to satisfy the demands in capacity and in reduction of latency. The adoption of flexibility and virtualization concepts in recent years has helped increase the overall throughput and the reactivity of the network.

In this manuscript, we focus on the optical transponder. Efforts have been made to make the device more flexible and virtualized, but reconfigurations of the hardware provoke interruptions of service and are still rare overall. Overcentralization of the decision making hinders the reactivity in case of a degradation in the optical link.

Firstly, we provide in this thesis work an auto-negotiation protocol for optical transponders that allows for fast synchronous change of parameters. Integrated with a commercial transponder and an embedded monitoring and decision-making solution we have validated our solution against a centralized control plane in a network testbed.

Secondly, we provide a neural network solution for embedded monitoring in the optical data plane. By allowing for a reconfiguration of the neurons weights of the network we believe that this solution can allow smarter fault detection and reconfigurations in the data plane and provide computational resources for the optical control plane for remote monitoring which would accelerate decision making for the whole network.