# Un système de compression vidéo et de synchronisation multimodale des EEGs pour la télémédecine

Laurent Lambert

**THÉSE DE DOCTORAT DE**
**l'UNIVERSITÉ PIERRE ET MARIE CURIE**

Spécialité

**Informatique**

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

**Laurent Lambert**

Pour obtenir le grade de

**Docteur de l'université Pierre et Marie Curie**

Sujet de la thèse :

# Un système de compression vidéo et de synchronisation multimodale des EEGs pour la télémédecine

soutenue le 7 décembre 2017

devant le jury composé de :

| | |
|---|---|
| M. Alain Mérigot | Rapporteur |
| M. Michel Paindavoine | Rapporteur |
| M. Pierre Langlois | Examinateur |
| M. Mohamed Chetouani | Examinateur |
| M. Khalil Hachicha | Encadrant |
| M. Andrea Pinna | Encadrant |
| M. Patrick Garda | Directeur de thèse |

# Remerciement

# Table des matières

# Liste des figures

# Liste des tableaux

Chapitre I

# Contexte et problématique



De nos jours, les services hospitaliers utilisent des systèmes d'information de plus en plus élaborés afin d'améliorer le service apporté aux patients admis dans leurs services. Ces systèmes deviennent petit à petit le cœur de l'interaction entre les organismes de santé et les patients, que ce soit pour l'acquisition de données, leurs archivages ou leurs consultations. En effet, Les services de santé s'écartent du support *"papier"* des dossiers patient pour préférer les supports numériques archivés aussi bien à l'hôpital que chez un hébergeur de santé agrée. L'utilisation des équipements d'acquisition analogique est également délaissée au profit d'équipements numériques capables de réaliser des analyses plus poussées et précises. Ces deux exemples exposent une tendance dans la manière dont les soins sont dispensés en intégrant de plus en plus d'outils électroniques et informatiques dans le parcours de santé.

L'utilisation de ces outils apporte des solutions aux problèmes des méthodes traditionnelles (Réduction de la place de l'archivage, manque de suivi des dossiers patient, imprécision pendant la chirurgie, etc.). Cependant, ils soulèvent de nouvelles problématiques notamment de traitement des données numériques (confidentialité, de volume de données), de sûreté et de fiabilité.

## 1 Électroencéphalographie

L'électroencéphalographie reporte à la discipline de l'exploration fonctionnelle de l'activité cérébrale. Elle fait souvent référence à la réalisation d'un examen électroencéphalogramme (EEG) qui est une mesure de l'activité bioélectrique cérébrale. La première mesure d'une telle activité a été réalisée en 1929 par Hans Berger et est montrée dans la figure 1.

**Figure 1.** Premier encéphalogramme enregistré

L'électroencéphalographie est une discipline en constante évolution avec les nouvelles technologies ce qui a permis de passer des enregistrements sur bandes papier vers des formats numériques. Elle est une discipline irremplaçable par des méthodes d'imagerie morphologique pour l'exploration fonctionnelle du cerveau. L'examen sous-jacent est non-invasif, relativement facile à réaliser et non douloureux pour le patient. Il est répétable afin de pouvoir assurer le suivi des patients. Il est également peu onéreux pour l'assistance publique.

Il est réalisé en mesurant le potentiel électrique à l'aide d'électrodes à différentes positions à la surface du crâne . Le placement est standardisé par Jasper [Jasper, 1958]. Chaque électrode produit un signal électrique individuel. Cela correspond à un canal d'EEG. À partir de ces canaux, les neurophysiologistes réalisent des montages qui correspondent à des différences deux à deux de canaux. La variation du potentiel d'une électrode reflète l'activité locale du cerveau. L'ensemble des canaux ainsi que la mise en forme par des montages permettent de connaître l'activité globale du cerveau. L'enregistrement des tensions aux bornes de ces électrodes et l'analyse visuelle des signaux acquis permettent aux neurologues de diagnostiquer différentes pathologies telles que :

- le trouble épileptique (épilepsie)
- l'encéphalite (inflammation du cerveau)
- les chocs à la tête
- l'état de mort cérébrale
- les troubles de la conscience et de la vigilance (coma, confusion)
- les troubles du sommeil
- les myoclonies
- la maladie de Creutzfeldt Jakob
- etc.

Cependant, l'enregistrement de l'activité du cerveau seul ne permet pas, à un spécialiste, d'interpréter de manière fiable les évènements présents sur les traces enregistrées. Il est notamment difficile à un médecin de déterminer si une fluctuation visible sur les signaux EEG est due à une réaction à l'environnement ou à une pathologie. Ainsi, une variation brutale d'un des signaux EEG peut avoir deux significations. La première cause de cette fluctuation peut-être un trouble interne du cerveau, celle-ci entraînant une réaction musculaire. La seconde alternative est un changement dans l'environnement causant une réaction (lumière s'éteint, la porte se claque, etc.). Il est

important de tracer l'origine d'une anomalie révélée par les signaux EEG de manière à lever l'ambiguïté. C'est pour cette raison que les recommandations françaises en l'électroencéphalogramme [André-Obadia et al., 2014] préconisent l'enregistrement d'information complémentaire lors de la réalisation de cet examen. Les canaux de signaux EEG sont alors accompagnés d'annotations d'un technicien présent pour l'examen, d'un enregistrement de l'activité cardiaque (ECG), de l'activité musculaire (EMG), de l'activité oculaire (EOG) et d'une séquence vidéo/audio du patient. Nous utilisons le terme **modalité** pour parler d'un des types de données ci-dessus. Dans cet examen, nous distinguons quatre modalités : les signaux physiologiques (EEG, ECG, EOG et EMG), l'enregistrement audio, l'enregistrement vidéo et les annotations. Nous parlerons alors de multimodalité pour désigner l'ensemble des données de nature différente.

Pendant beaucoup d'années, l'EEG désignait un EEG de routine. Cependant, avec l'avènement des appareils numériques les examens se sont diversifiés. Le terme examen EEG est devenu assez ambigu car plusieurs examens EEG existent ayant chacun des particularités et des objectifs différents. Nous détaillons dans la suite les principaux examens EEG.

**EEG de routine**

L'examen EEG de routine est le plus court durant entre 20 et 30 minutes. Il est historiquement l'examen EEG standard. Il est pratiqué en centre hospitalier par un technicien. Des méthodes d'activation sont utilisées afin de stimuler le patient pour ensuite observer si des crises se produisent lors de la période d'examination. Ces activations, aussi appelées épreuves, sont :

- l'hyperpnée pendant laquelle nous demandons au patient d'augmenter l'amplitude de ces mouvements respiratoires en respirant plus fort.
- La stimulation lumineuse intermittente pendant laquelle le patient est soumis à des flashs de lumière. La fréquence des flashs est variable allant de 1 Hz à 30 Hz. Nous demandons au patient d'avoir les yeux ouverts puis fermés.

L'examen peut également être réalisé tard le soir afin d'examiner le patient dans un état de fatigue élevé.

**EEG de sommeil**

L'examen EEG de sommeil est utilisé quand le patient ne montre pas de signe anormal lors d'un examen EEG de routine. Dans ce cas, un examen EEG de sommeil qui révèle des états différents de l'activité du cerveau peut servir au diagnostic du patient. Dans ce cas, le patient peut soit dormir la nuit à l'hôpital en étant mis en observation soit dans la journée pendant une sieste.

**Figure 2.** Patient lors d'un examen EEG de routine.

**EEG prolongé**

L'examen EEG prolongé fait référence à un enregistrement sur une période de temps compris entre 2 et 24 heures. La durée est définie par les médecins en fonction des crises pouvant se produire dans la période d'observation.

**EEG ambulatoire**

L'EEG ambulatoire, également appelé EEG-HOLTER, est un enregistrement continu du patient avec l'avantage de ne pas confiner le patient dans l'hôpital. Lors de ce test, le patient est libre de sortir et de vaquer à ces occupations. Cela permet d'obtenir des enregistrements plus longs qu'en hôpital. Il peut durer de quelques heures à quelques jours. Il est généralement prescrit lorsque le patient subit des crises très aléatoires ne pouvant pas être déclenchées par des activations.

**La Vidéo-EEG**

La vidéo-EEG fait référence au fait de faire un examen EEG en enregistrant à l'aide d'une caméra le patient pendant la procédure. Il peut être appliqué à l'EEG de routine, de sommeil ou prolongé. Pendant longtemps, l'enregistrement vidéo n'était pas systématiquement réalisé pour les examens EEG. Il est devenu maintenant le standard pour le diagnostic de crise et d'état de mal épileptique. L'enregistrement simultané de la vidéo avec l'enregistrement EEG permet de comparer les évènements physiologiques avec les gestes du patient (sursauts, mouvements involontaires) et son environnement (porte qui claque, interaction avec une autre personne). D'après les recommandations françaises sur l'électroencéphalogramme [André-Obadia et al., 2014], il est soit très fortement recommandé soit obligatoire d'utiliser un enregistrement vidéo pour les

**Figure 3.** Patient en EEG ambulatoire.

différents types d'examens EEG. Il est donc nécessaire d'intégrer la vidéo dans les équipements permettant de réaliser des examens.

Quel que soit l'examen utilisé, il est généralement dirigé par un technicien qui guide le patient au cours de la procédure. Le technicien met en place les équipements pour faire l'enregistrement et impose les épreuves au patient. Au cours de l'examen, il annote les évènements se déroulant au cours de l'examen. À la fin de l'examen, l'enregistrement est confié à un médecin spécialiste en neurophysiologie pour réaliser une analyse des traces relevées lors de l'examen. Du fait que le médecin ne voit pas le patient pour faire le diagnostic, il est important que les données soient les plus complètes possible pour ne pas faire de diagnostics erronés.

Dans le cadre de cette thèse, nous prenons en compte principalement les caractéristiques de l'EEG de routine en utilisant la vidéo, tout en gardant en tête celles de l'EEG prolongé et du sommeil. L'examen ambulatoire lui n'est pas pris en compte dans les objectifs de cette thèse.

Finalement, une des spécificités de la réalisation des examens EEG en France tient aux compétences médicales et techniques qu'ils requièrent. La réalisation des actes d'électroencéphalographie demande des formations spécialisées pour les médecins interprétants les examens ainsi que pour les techniciens opérants les actes. De plus, l'évolution des connaissances, des techniques et des indications de l'acte d'électroencéphalogramme pousse le domaine à être fragmenté en discipline (Adulte, néonatale, infantile, en unité de soins intensifs, etc.). Ceci entraîne une évolution de la démographie des médecins compétents en discipline d'électroencéphalographie qui aujourd'hui apparaît être mal répartie sur le territoire français. Cela, combiné à des besoins non

homogènes dans les régions, entraîne une surcharge de certains centres hospitaliers ou un manque d'expertise dans certaines disciplines.

# 2 Télémédecine et TéléEEG

La télémédecine est, selon le code de santé publique (art. L.6316-1), « une forme de pratique médicale à distance utilisant les technologies de l'information et de la communication ». Elle ne vise pas à remplacer les méthodes médicales traditionnelles, mais apporte une réponse aux problématiques actuelles de demande de soins. C'est une des solutions pour l'élévation du parcours de soins, en particulier dans les zones les plus reculées. Elle constitue également une amélioration de l'efficacité économique d'actes médicaux. Elle doit cependant apporter les garanties de sécurité fondamentales de confidentialité, d'intégrité et disponibilité.

Cinq actes de télémédecine sont actuellement reconnus par le code de santé publique :

- la téléconsultation,
- la téléexpertise,
- la télésurveillance,
- la téléassistance médicale,
- la régulation médicale.

Dans le cadre de l'examen EEG et pour cette thèse, nous nous positionnons dans un acte de téléexpertise. La téléexpertise est une discipline envisagée par les institutions françaises pour remédier à un manque d'expertise d'établissements de santé dans certaines disciplines ou une surcharge d'établissements dans des zones denses ou sous budgétisés. Il y a téléexpertise quand un médecin demandeur fait appel à un second avis d'un médecin expert non présent sur site pour diagnostiquer un patient. Nous pourrions également parler de télédiagnostic. Celui-ci a pour principe de réaliser un examen EEG dans un centre (hospitalier, médecin généraliste, etc.) et d'être ensuite transmis pour expertise à un centre partenaire expertiseur. L'expertise vise à réaliser le diagnostic de l'examen au nom du centre demandeur. Tous ces transferts sont réalisés en utilisant des infrastructures réseau et informatiques. Une solution en téléexpertise apporte plusieurs avantages. Elle permet en cas d'urgence d'obtenir un diagnostic d'un médecin spécialisé sans déplacement du patient ou du médecin. Elle réduit également les coûts de transfert ou de déplacement des patients leur évitant de se déplacer vers des centres hospitaliers experts.

L'HAS[1] n'a pas définis la téléEEG comme faisant partie des axes prioritaires de télémédecine. Cependant, les objectifs attendus du développement de la télémédecine par la DGOS[2] dans leur guide méthodologique pour l'élaboration du programme

1. HAS : Haute Autorité de Santé
2. DGOS : Direction Générale de l'Offre de Soins

régional de télémédecine [GMEPRT] comprend la mise en place de téléEEG. Pour rappel, ces objectifs sont l'amélioration à l'accès des soins de santé de qualité pour tous et sur l'ensemble du territoire. La télétransmission idéale de TéléEEG doit permettre de transférer les dossiers patients aux médecins interpréteurs en préservant la confidentialité liée à toutes données médicales. Elle peut à terme optimiser les coûts en rapprochant les compétences neurophysiologiques aux patients habitant dans des zones reculées. Elle facilite l'accès à des soins de qualité tout en réduisant les coûts de déplacement pour le patient.

Des expériences en Espagne et au Royaume-Uni [Lasierra et al., 2009, Coates et al., 2012] exposent des retours très positifs sur l'utilisation de téléexpertise d'EEG. L'interprétation d'un examen EEG à distance s'est montrée à 92% fiable comparée à un examen local. De plus, 99% des patients qui ont été examinés en téléexpertise sont satisfaits de l'expérience et 75% préfèrent cette expérience à un examen local. Ceci principalement grâce à un accès à des soins de meilleure qualité et une réduction des temps de voyage. D'après la même étude, les patients réduisent leurs dépenses de 40 euros en moyenne et divisent par deux le temps perdu pour réaliser l'examen (en prenant compte les temps de trajet). Le temps passé pour réaliser l'examen en lui-même a augmenté de 2 minutes le faisant durer 47 minutes pour une téléexpertise. Cette augmentation est considérée comme raisonnable par l'établissement demandeur.

En France, la demande d'expertise de second avis est utilisée régulièrement. L'étude [Sauleau et al., 2016] montre les facettes de cette pratique. Dans un premier temps, 37% des centres hospitaliers qui ont répondu à l'étude affirment ne pas utiliser la téléexpertise car elle possède les compétences nécessaires. 60% des établissements qui l'utilise, jugent que la téléexpertise est réalisée régulièrement. D'après l'étude, les demandes de second avis n'utilisent que très peu souvent un réseau informatique comme peu le montrer la figure 4. Nous remarquons également que la pratique la plus courante est l'envoi d'examen par clé USB ou CD qui est généralement transporté par coursier. Beaucoup des centres interrogés ont émis la difficulté de faire téléexpertiser un examen EEG accompagné d'une vidéo synchronisée et se sont plaints de la non-compatibilité complète des logiciels de relecture d'examen entre fabricants. Dans 52% des cas, un centre doit gérer au moins deux formats d'enregistrement différents.

Jusqu'à présent, le développement de ces échanges est le fruit d'initiatives personnelles sans gouvernance ni encadrement par des institutions. Elles ont permis de répondre à des besoins cruciaux, tels que le réseau « BB-EEG» qui permet la lecture des EEG en néonatalogie pour la région des Pays de la Loire. Si le service médical rendu est indiscutable en permettant l'accès aux soins pour tous et la continuité des soins, des études médico-économiques n'ont pas encore été réalisées afin d'établir les lignes budgétaires qui pourraient faire l'objet d'une économie, tel que les transports

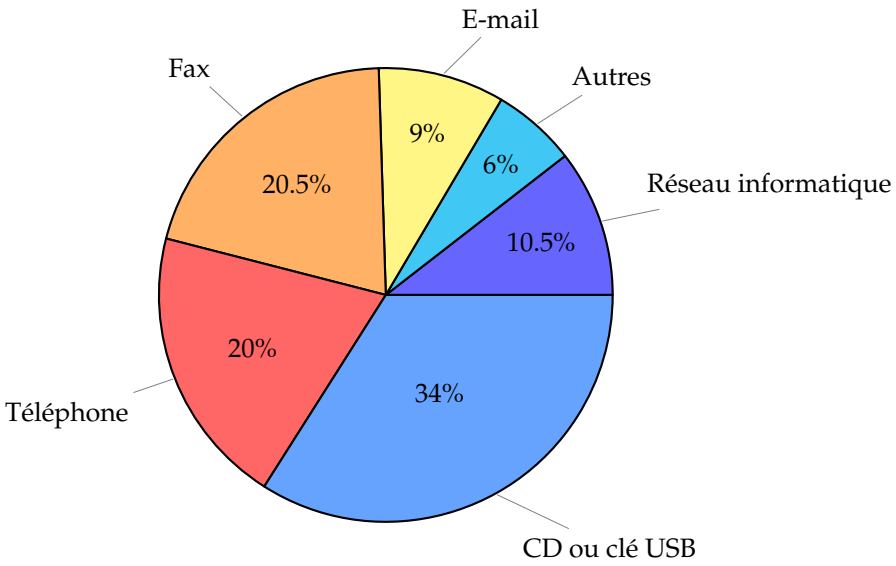



**Figure 4.** Répartition des demandes de second avis en fonction du support utilisé. Reproduit de l'article [Sauleau et al., 2016]

médicalisés, les journées de travail non réalisées pour se rendre parfois à un centre hospitalier distant de 150 km et les durées moyennes d'hospitalisation. En effet, bien qu'il soit prévu une prise en charge financière des actes de télémédecine (article R. 6316-5), quasiment aucun acte n'est à ce jour financé par l'assurance maladie.

# 3 Projet Smart-EEG

Le projet Smart-EEG est un projet de télémédecine regroupant un consortium de quatre industrielles (2CSI, ACACIA, CIRA et PARTELEC) et de deux hôpitaux (hôpitaux Lariboisière et Européen Georges-Pompidou) et de deux académiques (ETIS et LIP6). L'objectif premier du projet est de répondre au besoin de télétransmission d'examen EEG en France. Il a été financé grâce au 15 ème appel d'offres FUI. Le projet vise à développer une solution complète, de l'acquisition à l'archivage, d'examen EEG. Elle doit s'intégrer aux systèmes d'information de santé aussi bien au niveau du système d'information hospitalier (SIH) que du dossier médical personnel (DMP) ou équivalent au niveau international. Elle doit être couplée à un espace de stockage dédié aux examens EEG, géré par un hébergeur agréé de données de santé afin de couvrir tous les besoins potentiels. Enfin, cette plateforme doit permettre non seulement de répondre à la problématique aiguë actuelle du TéléEEG de diagnostic et du TéléEEG d'urgence dans le cadre de l'accès aux soins pour tous, mais aussi de développer la formation des professionnels avec des dossiers types et faciliter l'expertise pour un second avis. Le projet est structuré en tâches dédiées à chaque partenaire :

- l'analyse des spécifications techniques et de l'utilisation d'une plateforme de

TéléEEG idéale et la validation des solutions proposées (Lariboisière, AP—HP).

- L'étude de l'intégration des mesures physiques EEG dans un fichier image et des moyens techniques pour y parvenir (métadonnées) (CIRA).
- La conception d'une architecture de compression des signaux EEG et de synchronisation à haut niveau et la réalisation d'un démonstrateur électronique (LIP6).
- La conception d'un système d'acquisition et de synchronisation à bas niveau des signaux physiologiques et le développement d'un prototype de laboratoire (ETIS).
- L'étude et la réalisation d'un logiciel d'interface homme-machine (IHM) permettant au praticien de piloter le dispositif d'enregistrement EEG (ACACIA).
- L'étude et le développement d'une plateforme logicielle d'intermédiation dédiée au télédiagnostic EEG permettant de connecter les EEG numériques existants sous un format commun (PARTELEC).
- L'étude, le développement et l'hébergement d'une solution interopérable en entrée-sortie permettant de disposer de l'ensemble des informations EEG (2CSI).
- Le protocole de validation de l'image (HEGP et Lariboisière, AP—HP).

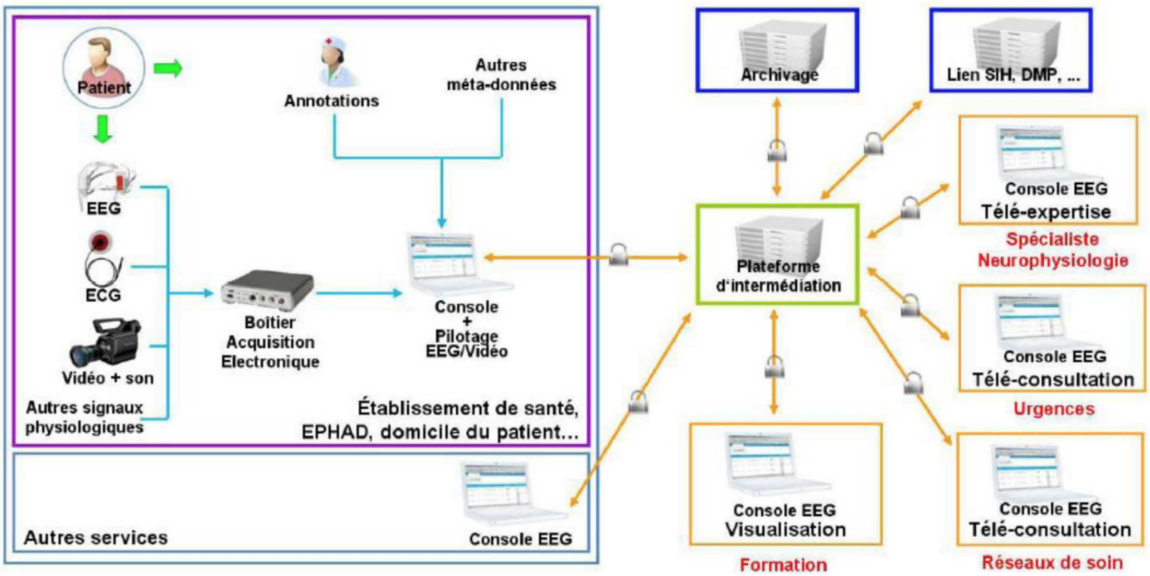



**Figure 5.** Diagramme du parcours de soins défini par le projet Smart-EEG. Repris de l'annexe technique du projet Smart-EEG.

Le projet Smart-EEG définit un parcours de santé pour les examens EEG en téléexpertise. La figure 5 retrace ce parcours de soin. Dans un premier lieu, le patient est examiné par un technicien à l'hôpital. Lors de cet examen, un boîtier d'acquisition enregistre les informations pertinentes pour le bon diagnostic du patient. Celles-ci sont évidemment des signaux EEG, mais également des signaux physiologiques annexes tels que les signaux ECG, EMG et EOG. L'intégralité de l'examen est également filmée afin

d'avoir un maximum d'information sur le déroulement et connaître les perturbations qui pourraient intervenir lors de l'examination. Le déroulement de l'examen est supervisé par un technicien par le biais d'une console locale. La console permet au technicien d'avoir un retour direct des données acquises par le boîtier ainsi que de piloter le boîtier dans les différentes phases de l'examen. Il y a trois phases lors de l'examen, la phase d'installation et de calibration, la phase d'acquisition et la phase de finalisation. Lors de la première phase, le patient est installé et des sondes lui sont appliquées afin de capturer les signaux physiologiques. Le technicien s'assure de la bonne mise en place des sondes en visionnant les signaux sur sa console. Il entre également toutes les données du patient et de l'établissement qui constitue le dossier de l'acte. Une fois la mise en place bien faite, le technicien peut alors décider de commencer à enregistrer des signaux qui constitueront les données de l'examen. Au cours de l'acquisition, le patient subira plusieurs épreuves comme l'hyperpnée et la SLI [1]. Pendant ce temps, le technicien annote au mieux les évènements se produisant lors de l'examen afin d'enrichir les informations environnementales. Une fois la période de 20 à 30 minutes d'examen passée, le technicien finalise l'examen. À ce moment, la fin de l'examen est reçue par la console locale et il est préparé pour être envoyé à une plateforme d'intermédiation. Cette plateforme se charge de stocker l'examen pour la relecture par d'autres professionnels de santé. L'examen reste sur ce serveur le temps qu'il soit expertisé. Différents centres d'expertise peuvent alors consulter l'examen à distance de manière à émettre un diagnostic pour le patient. Ce diagnostic fait partie du dossier examen et est alors stocké avec celui-ci sur la plateforme d'intermédiation. Quand le dossier est classé, il est alors archivé vers un archiveur de santé agrée par la plateforme d'intermédiation. Dans le cas où le dossier devait être relu, la plateforme peut récupérer l'examen depuis l'archiveur et le restitué pour une téléexpertise. La base de données d'examen peut également servir à faire de la formation aux hôpitaux universitaires. La plateforme d'intermédiation a aussi un rôle d'interopérabilité. Elle doit être capable de lire des examens enregistrés d'en d'autres formats que celui produit par l'équipement du projet Smart-EEG et de les intégrer dans sa base de données. Tous les examens doivent pouvoir être relus de manière transparente par le logiciel de relecture.

Les travaux de cette thèse s'inscrivent dans les contributions du LIP6 au projet Smart-EEG visant à constituer une architecture numérique permettant la compression et la synchronisation des signaux EEG avec une vidéo de la scène. Les travaux ont également abouti à des contributions dans les travaux des autres partenaires, notamment dans la définition des besoins, la conception du système d'acquisition bas niveau et la réalisation de la console de prévisualisation.

1. SLI : stimulation lumineuse intermittente

# 4 Problématique

La première tâche réalisée par les membres du consortium du projet Smart-EEG a été de définir les verrous technologiques présents dans les équipements d'acquisition prévenant leur utilisation pour des actes d'électroencéphalographie en téléexpertise. D'après l'analyse de la procédure d'acquisition et des infrastructures réseau disponibles dans les centres hospitaliers pour la transmission d'examen EEG, le dispositif doit répondre à deux verrous technologiques : les différentes modalités acquises doivent être synchronisées et la fréquence d'échantillonnage des vidéos doit permettre l'identification de certains mouvements pour les rapporter à des évènements corticaux identifiés sur les tracés EEG.

**Synchronisation**

Il est important de pouvoir garantir une synchronisation exacte entre les modalités pour que lors de l'établissement d'un diagnostic la solution technique mise en œuvre ne soit pas mise en cause pour une erreur médicale. Elle doit donc garantir des propriétés déterministes sur la réalisation de la synchronisation des outils employés pour faire l'acquisition.

Le deuxième aspect de la synchronisation est celui du stockage de l'examen. Nous entendons par synchronisation les informations temporelles que nous sauvegardons et structurons dans un fichier pour pouvoir reconstituer le déroulement de l'examen lors d'une relecture. Cet aspect aborde en effet les problèmes de synchronisation de données mais également de flux compressés et de norme d'interopérabilité. Nous étudierons les formats de fichier apte à être utilisé pour la réalisation d'un flux multimodales transférable par internet et stockable sur des serveurs d'archivage.

**Vidéo**

Le deuxième verrou que nous avons pu identifier dans les équipements présents dans les hôpitaux est la qualité de la vidéo acquise en examen. L'enregistrement vidéo sert à identifier des évènements sporadiques qui pourraient avoir un effet sur le patient et donc avoir un impact sur les signaux physiologiques. Cela permet d'apporter plus d'informations sur le déroulement de l'examen. Il est aussi possible que l'information portée par l'image soit utile à identifier des spasmes que le patient pourrait subir. En particulier, il est important d'avoir une bonne définition sur les mains et sur la tête du patient. Les médecins ont rapporté le besoin d'avoir, en plus d'une grande définition spatiale sur la vidéo, une plus grande définition temporelle afin de pouvoir détecter des contractions musculaires très rapides tels que des myoclonies. Les études [Brown et al., 1991] rapportent qu'une myoclonie peut durer aussi peu que 20 millisecondes. Cette contrainte est prise en compte pour définir la définition temporelle

pour l'acquisition vidéo.

La grande définition demandée par les médecins engendre un volume de données vidéo très conséquent pouvant aller jusqu'à plus de 600 Mo/s. En gardant à l'esprit que cette séquence vidéo doit être envoyée à un centre pour faire le diagnostic dans le cadre de téléexpertise et que l'examen sera ensuite archivé pendant vingt ans pour des mesures légales, il est alors évident que la séquence vidéo doit être compressée de manière très efficace afin de diminuer la place à allouer aux examens.

**Système**

Le dernier point, qui engendre des contraintes supplémentaires, se ramène aux contributions du LIP6 qui doivent aboutir à un démonstrateur de laboratoire permettant de faire la synchronisation et la compression des modalités d'un examen EEG en temps réel.

Le dispositif développé devra être assez performant pour ne pas bloquer d'examen. Par cela, nous parlons du temps de traitement que pourra nécessiter l'examen pour sa transmission au centre hospitalier diagnostiqueur. Le dispositif mis à disposition des équipes de soins devra alors répondre rapidement et ne pas gêner aussi bien physiquement que par des temps d'attente dus au service. Ceci nécessitera de faire une analyse des tâches coûteuses en temps et de les optimiser ainsi que de les embarquer dans un dispositif compact pour ne pas encombrer les salles d'examen.

Finalement, toujours dans le cadre du projet, ce démonstrateur devra pouvoir servir de base d'évaluation de la possibilité d'appliquer la télémédecine aux examens EEG. De ce fait, il est important de réaliser un dispositif respectant toutes les contraintes que l'on peut trouver dans le milieu médical. Notamment, le dispositif devra être évalué et certifié, par une institution accréditée, pour les risques électriques et électromagnétiques aussi bien pour le patient que pour l'opérateur. Le dispositif devra également s'intégrer dans les salles d'examens de manière la plus proche possible de ce qu'il se fait actuellement. Il faudra proposer un système capable de tenir sur un chariot d'équipement EEG. De plus, du point de vue du technicien, la configuration et la mise en place devront être similaires aux autres équipements pour ne pas déranger leur travail. Pour finir, celui-ci doit être transparent à l'utilisation et ne doit pas rajouter d'étape dans la procédure de l'examen. Le démonstrateur devra aussi répondre aux objectifs économique et commercial du projet. Chaque exemplaire de la solution doit se positionner en dessous de 10000 €.

Ces trois axes nous permettent alors de guider le déroulement de ce manuscrit ainsi que de dresser une problématique pour ces travaux de thèse. Nous la résumons ainsi :

Comment **définir** et **concevoir** un dispositif d'examen **EEG** capable de **synchroniser** l'acquisition de signaux physiologiques avec une séquence vidéo à haute fréquence pour ensuite les **compresser** et les **transmettre** en temps réel dans le but d'être **téléexpertisé** ?

## 5 Conclusion

Dans ce chapitre, nous avons détaillé le contexte de ces travaux de recherche afin d'apporter quelques réponses à des décisions faites dans le cadre de cette thèse. Nous avons pu développer les besoins uniques que peut imposer l'utilisation de la télémédecine pour l'examen EEG. De ces besoins, nous en avons déduit trois axes de recherche que nous poursuivons tout le long de cette thèse. Nous pouvons préciser le déroulement des cinq prochains chapitres de ce manuscrit.

Dans le chapitre **II,** nous analysons les solutions apportées par l'état de l'art pour la problématique posée. Nous établissons un état de l'art des systèmes de téléEEG marquants. Nous passons ensuite à un état de l'art de chacun des verrous technologiques que le projet Smart-EEG a désigné comme bloquant. Nous commençons alors par définir les techniques de synchronisation d'acquisition simultanée. Nous étudions ensuite les méthodes d'encapsulation de flux multimédia permettant de conserver la synchronisation lors de l'envoi de l'examen à centre expert. Nous terminons ensuite par l'étude des solutions existantes d'algorithmes de compression utilisable pour le flux vidéo d'un examen EEG.

Le chapitre **III,** appelé synchronisation, apporte dans un premier lieu une solution à la problématique de synchronisation lors de l'acquisition en utilisant une méthode que nous appelons la synchronisation de bas niveau. Ensuite, nous proposons une méthode afin de transmettre des données compressées sur un réseau informatique que nous appelons la synchronisation haut niveau.

Dans le chapitre **IV,** nous décrivons un nouvel algorithme de compression de séquence d'image ROI-Waaves. Nous commençons par préciser les caractéristiques de la prise de vue de l'enregistrement vidéo. Nous décrivons ensuite étage par étage les traitements réalisés par le codeur ROI-Waaves. Ce chapitre n'aborde que la partie algorithmique du compresseur et non son implémentation qui est détaillée dans le chapitre suivant.

Le chapitre **V,** aborde la conception et l'implémentation du dispositif de compression et de synchronisation. Il est développé dans ce chapitre la nécessité de développer deux versions du dispositif, une première implémentation, dite logicielle, dont la majeure partie des traitements vidéo (notamment la compression vidéo) est réalisée

sur processeur et une seconde implémentation, dite matérielle, dont la compression vidéo est implémentée par des IP FPGA et qui est embarquée sur une seule carte de développement. Nous décrivons dans cette partie l'implémentation matérielle de l'algorithme décrit dans le chapitre IV.

Le chapitre **VI** contient les évaluations des solutions apportées dans les chapitres III, IV, V. Nous commençons par analyser l'apport qualitatif puis quantitatif de la synchronisation. Nous évaluons ensuite le codeur ROI-Waaves sur les aspects de qualité, de rapport de compression et de complexité du codeur. Pour finir, nous quantifions les ressources utiles des deux implémentations.

Pour finir, nous présentons nos conclusions sur ces travaux et proposons des perspectives dans le dernier chapitre.

Chapitre II

# État de l'art

# 1 Systèmes de télémédecine pour la vidéo-EEG

## 1.1 Historique

En 1968, un démonstrateur est présenté [Bird and Murphy Jr, 1974] faisant le lien entre l'aéroport Logan de Boston et l'hôpital général du Massachusetts (trois miles ou environ cinq kilomètres). Cette installation permettait de faire une consultation entre un physicien et un patient en utilisant un système de conférence audio/vidéo bidirectionnel (le patient et le docteur pouvaient se voir et se parler). Le docteur réalisant l'examen est assisté par un technicien présent avec le patient pour effectuer l'examen (placement du stéthoscope, cadrage de la vidéo, etc.). Les données que le docteur reçoit sont l'électrocardiogramme, le rythme cardiaque, le rythme respiratoire et la pression artérielle.

En 1974, Walter Holzer présente un nouveau dispositif [Holzer, 1974] techniquement similaire à celui de Bird et Murphy . Celui-ci est déployé à Puerto Rico et permet de faire des télédiagnostics entre hôpitaux. Ce système, opérant dans un périmètre de trente miles (environ quarante-huit kilomètres), fonctionne sur les lignes de transmissions standards des années soixante-dix aux États-Unis (U.S. 525-line system). Le patient est enregistré par le biais de trois caméras. Le signal vidéo est aussi envoyé à un centre de contrôle pour les ajustements des caméras, mais ce signal peut être coupé par le docteur pour des raisons de confidentialité. En parallèle, un circuit dédié est mis en place pour la transmission des signaux physiologiques. Ce circuit permet au docteur de visualiser l'EEG (quatre voix), l'ECG (trois voix), l'EMG (deux voix), la température, la pression sanguine, le rythme cardiaque, la fréquence respiratoire et le bruit du cœur (via un stéthoscope). Toutefois, seulement quatre signaux peuvent être transmis et visualisés à un instant donné. Toute cette transmission de données est réalisée sur trois lignes de transmission dédiées (deux pour la vidéo et une pour les signaux). Tout l'examen est opéré de façon synchrone entre le patient et le docteur.

Sur cette tendance, beaucoup de projets de télémédecine ont vu le jour entre les années 70 et 2000 aux États-Unis ([Preston, 1995]), au Canada ([Lindsay et al., 1987], [House, 1991]) en Europe ([De Luca, 1998], [Vaz et al., 1991]) et en Afrique([House et al., 1987]). Tous ces projets restent baser sur les technologies analogiques ce qui a pour incidence d'être hautement sujet au bruit du transport des données. Il est également à noter que beaucoup de ces projets utilisent des voies séparées pour la vidéo/audio et les signaux physiologiques ce qui peut avoir pour effet de désynchroniser les deux modalités. Au passage au numérique plusieurs rapports ont montré des expériences de vidéo-EEG à distance ([Chacko and McCullagh, 2014], [Campos et al., 2012]). Cependant, les deux expériences précédentes ont soulevé un problème sur la synchronisation des signaux avec l'enregistrement vidéo. Pour

comprendre ce problème, nous avons fait un état de l'art des systèmes d'EEG actuel pour comprendre leur fonctionnement.

## 1.2 Solutions actuelles

Pour notre cas d'étude, nous voulons étudier les solutions commerciales disponibles sur le marché afin de comprendre les limitations qu'ils présentent. Nous analysons également les solutions proposées par les plus gros vendeurs d'EEG. Nous analysons dans cette étude, les capacités d'acquisition physiologique (nombre de canaux d'acquisition, la fréquence et la résolution), d'acquisition vidéo (images par seconde, résolution, nombre de flux), d'exportation d'examens vers d'autres formats, les connectiques utilisées pour brancher les appareils ensemble.

- **Micromed**
  La solution de *Micromed* pour l'examen vidéo-EEG est *BrainQuick*, disponible en plusieurs configurations allant de 29 à 48 canaux échantillonnant à 24 bits et 16 kHz. L'enregistrement vidéo proposé est acquis à 25 images par seconde et en full-HD. Le logiciel utilise un format d'acquisition propriétaire mais peut également exporter vers EDF. Aucune indication n'est fournie sur la connectique utilisée pour les équipements mais sur le modèle présent à l'hôpital, nous avons constaté un boîtier d'amplification connecté en USB et une caméra Sony du commerce également connecté en USB.

**Figure 6.** Photo du dispositif *BrainQuick*

- **Neurosoft**
  La solution de *Neurosoft*, *Neuron-Spectrum-Video*, est compatible avec tous les amplificateurs de la société. Les produits sont capables d'acquérir de 9 à 40 canaux

de signaux physiologiques à 5 kHz. Il est possible d'acquérir jusqu'à trois flux vidéo mais ses caractéristiques ne sont pas spécifiées. Cependant, l'acquisition s'effectue par le biais d'une carte d'acquisition vidéo composite connectée en PCI. Les différents amplificateurs sont eux connectés en USB

**Figure 7.** Photo du dispositif *Neuron-Spectrum-Video*

- **Natus**
  La solution de *Natus* (anciennement *DeltaMed*), *brainbox* combiné au logiciel *coherence*, enregistre 29 à 40 canaux en fonction de l'amplificateur utilisé (aucune autre spécification n'est donnée). La vidéo est connectée par Ethernet et l'amplificateur par USB ou Ethernet. La vidéo est acquise en HD (1280x720 pixels) sans fréquence d'images précisée.
- **Mitsar**
  La société *Mistar* propose une solution de vidéo-EEG *Mitsar-Vidéo-EEG* compatible avec un seul amplificateur de la même marque. L'amplificateur est capable d'acquérir 25 canaux à 2 kHz avec une résolution de 24 bits. La solution est accompagnée de deux caméras vidéo, une caméra pour le fond de la pièce et une caméra pour effectuer un zoom sur le patient. Ces deux caméras sont connectées au PC via une carte d'acquisition composite. L'amplificateur lui est connecté en USB. Point intéressant, c'est la seule solution dont nous avons pu avoir le manuel d'utilisation de l'appareil dans lequel on peut avoir les connectiques exactes ([MistarDoc]).
- **Neurowerk**
  Neurowerk est une société allemande proposant la solution Neurowerk EEG. La solution peut être composée d'amplificateurs allant de 34 à 90 canaux. Ils acquièrent à une fréquence de 512 Hz en 16 bits. Un add-on est disponible pour

rajouter la capacité d'enregistrer un flux vidéo. Cette caméra est connectée par Ethernet et enregistre en Full-HD sans fréquence précisée. C'est la seule solution qui précise que le flux est encodé en utilisant les codecs des standards MPEG-1, MPEG-2 ou MPEG-4.

**Tableau 1.** Tableau comparatif des solutions actuelles

| Solution | Physiologique | Vidéo | connectique ampli/caméra |
|---|---|---|---|
| *BrainQuick* | 29 à 48 canaux, 16 kHz, 24 bits | full-HD, 25 fps | USB/USB |
| *Neuron-Spectrum-Video* | 9 à 40 canaux, 5 kHz | 3 flux | USB/PCI |
| *Brainbox* | 29 à 40 canaux | HD | USB/Ethernet |
| *Mistar-EEG* | 25 canaux, 2 kHz, 24 bits | 2 flux, HD | USB/PCI |
| *Neurowerk EEG* | 34 à 90 canaux, 512 Hz, 16 bits | Full-HD | USB/Ethernet |

Le tableau 1 résume les solutions actuelles. Ces solutions ont des similitudes dans leur fonctionnement. Toutes ces solutions utilisent chacune un format d'archivage propriétaire dont les spécifications ne sont pas publiques. Elles sont toutes basées sur un même principe pour réaliser l'acquisition, un boîtier d'acquisition de signaux physiologique est connecté par USB à un PC. Ce PC exécute un logiciel d'acquisition prévu pour faire fonctionner le boîtier sous Windows. Le logiciel pilote également l'acquisition du signal vidéo par des interfaces variées qui vont de l'USB à l'Ethernet. Lors de nos visites à l'hôpital Lariboisière, nous avons pu nous rendre compte que les caméras étaient des fois rajoutées par les techniciens plutôt que d'utiliser les caméras des dispositifs.

## 1.3 Les limitations

Actuellement, les logiciels d'enregistrement et de relecture des EEG ne sont pas compatibles entre eux. À ce jour, les deux formats utilisés permettant l'interopérabilité des examens EEG sont l'EDF et l'EDF+. Ils permettent de relire un examen sur un appareil différent de celui sur lequel l'examen a été acquis. Les autres formats propriétaires obligent les centres souhaitant travailler ensemble à posséder le même équipement d'acquisition et à utiliser un lecteur « compatible ». Cependant lors de la conception du format EDF et EDF+, le visionnage de la vidéo n'était pas encore primordial. De ce fait, les formats EDF et EDF+ par construction ne peuvent garantir une synchronisation fine entre un fichier de signaux physiologiques et un enregistrement vidéo. Cette contrainte technique nous a donc conduit à nous intéresser à la manière dont nous pouvons stocker des données acquises pour que l'information de synchronisation soit conservée lors de la relecture de l'examen.

Les différents équipements que nous avons pu étudier dans les hôpitaux proposent tous une acquisition vidéo en add-on à un boîtier de signaux physiologiques. Cette acquisition vidéo est orchestrée par un ordinateur qui pilote également le boîtier d'acquisition de signaux physiologiques. L'ordinateur qui réalise cette tâche n'offre

aucune garantie temps réel, aussi bien au niveau de l'architecture du PC que des logiciels qui s'exécute au-dessus (OS et logiciel). Les actions permettant de contrôler un appareil seront reçues après un laps de temps non connu par le logiciel du PC. Le pilotage des équipements engendre donc de l'indéterminisme quant au temps de réponse des équipements. Il faut bien noter que l'acquisition vidéo et l'acquisition physiologique ne sont reliées que par l'intermédiaire du PC. Pour ces raisons, il est difficile au système de garantir des contrôles déterministes entre les deux appareils ce qui implique qu'il est difficile d'avoir une précision déterministe sur l'acquisition des données. Il est à noter que le mécanisme que nous voulons mettre en place doit s'adapter à des composants pris sur l'étagère standard dont nous ne pouvons changer le comportement interne. Nous établirons alors un état de l'art des mécanismes d'acquisition disponibles pour contourner cette limitation de temps réel.

Finalement, en comparant les caractéristiques techniques des acquisitions vidéo, nous pouvons tirer deux conclusions : les flux vidéo acquis ne remplissent pas les contraintes que les médecins imposent afin de diagnostiquer des pathologies comme les myoclonies qui nécessitent une fréquence d'acquisition à 100 images par seconde. De plus, les algorithmes de compression utilisés pour compresser ce flux de la famille MPEG (H264, MPEG-1/2/4, ASF) ne sont compris dans aucun cadre légal pour leur utilisation dans le domaine médical.

Il est à noter qu'aucune solution commerciale ne permet de réaliser une acquisition à 100 images par seconde synchronisée avec les signaux physiologiques de manière sûre et compressant les données pour un usage de téléexpertise. Une grande partie de cette thèse aura donc comme objectif de mettre au point un dispositif répondant à ces contraintes.

# 2 Synchronisation

Dans cette section, nous établissons les solutions sur le sujet de la synchronisation. Dans un premier temps, nous regardons les méthodes possibles pour synchroniser une acquisition de données pour passer dans un deuxième temps sur les formats de fichier permettant de conserver la synchronisation lors d'une transmission.

## 2.1 Synchronisation d'acquisition

Le problème de la synchronisation ne concerne pas uniquement la conservation de la synchronisation lors des différentes étapes de la vie d'un examen qui est établi par le format de fichier mais également de son exactitude. En effet, pour faire l'acquisition simultanée de deux modalités utilisant deux systèmes d'acquisition différents, il doit

être mis en place un mécanisme de synchronisation entre les deux systèmes. Sans ce mécanisme, il n'est pas possible de faire correspondre les données des deux systèmes. Pour faire cette synchronisation, deux grandes méthodes sont possibles :

- **Acquisition par signaux**, de manière centralisée, en acquérant donnée par donnée par un dispositif réalisant la synchronisation et la correspondance temporelle. Les systèmes d'acquisition sont commandés par des déclencheurs afin d'acquérir les données. Ces signaux de déclencheur sont alors très ressemblants à des horloges.
- **Acquisition par temps**, de manière distribuée, en synchronisant une horloge sur deux systèmes qui cale alors leur acquisition sur cette horloge. En synchronisant le temps, les deux systèmes estampillent les données avec le temps auquel elles ont étées acquises.

Quelle que soit la méthode utilisée, il est indispensable de faire cette coopération en utilisant des médias répondant à des contraintes temps réel. Si ces mécanismes ne remplissent pas cette contrainte alors aucune garantie ne peut être assurée sur la précision de la synchronisation réalisée.

### 2.1.1 Acquisition par signaux

**PXI**

Les plateformes PXI développées par National Instrument [NI] sont des plateformes d'acquisition de données modulaires basées sur l'architecture PXI-Express. Elle permet la composition d'un système d'acquisition par des cartes filles compatibles et développé spécifiquement pour cette architecture. La figure 8 représente cette architecture.

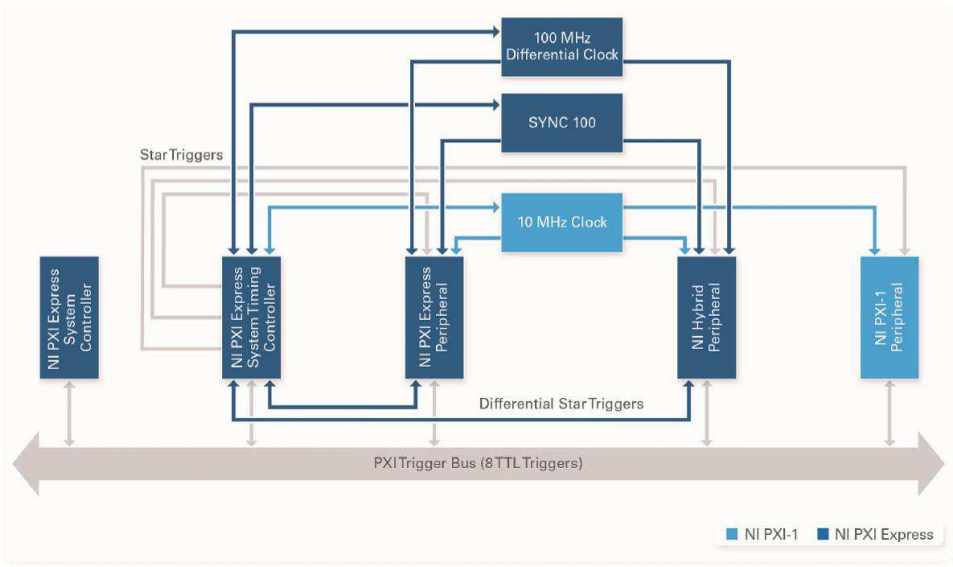



**Figure 8.** Architecture PXI-Express. Source de l'image.

L'architecture PXI-Express est une mise à jour pour des besoins de débit et de précision de l'architecture PXI. L'architecture PXI-Express est rétrocompatible avec le

PXI. La partie PXI comprend un module générant une horloge de référence à 10 MHz distribuée à tous les composants ainsi qu'un réseau de déclencheurs organisés en étoile. Les spécifications du PXI définissent la longueur des câbles transportant ces signaux afin de maintenir des timings similaires sur les cartes connectées au PXI. Au passage au PXI-Express, l'architecture a été dotée d'un module délivrant une horloge de référence différentielle à 100 MHz ainsi qu'une horloge de synchronisation 10 fois plus lente (10 MHz) et un réseau différentiel de déclencheur. Dans cette architecture, chaque carte génère une horloge d'échantillonnage contrôlant l'acquisition par un DAC sur la carte. Sur cette architecture, il est possible de définir plusieurs méthodes de synchronisation en fonction des capacités des cartes disponibles.

- La méthode la plus simple est d'avoir une carte (maître) qui délivre directement par le bus PXI son horloge d'échantillonnage aux autres cartes (esclaves). La méthode a le mérite d'être facilement mise en place mais a deux inconvénients, le premier est d'avoir des délais de propagation inconnus entre les cartes et le deuxième est de contraindre les esclaves à acquérir au rythme du maître.
- La deuxième méthode consiste à ne partager que le signal de déclenchement de début d'acquisition entre plusieurs cartes. La synchronisation subit un décalage dû à la propagation du signal ainsi que d'une dérive car les horloges des cartes ne sont pas synchronisées. Cette méthode est probablement la moins bonne en termes de précision.
- La troisième méthode a pour principe que chaque carte dérive son horloge d'échantillonnage à partir de l'horloge système. Le début de l'acquisition est défini par un maître qui émet un déclencheur après un front de l'horloge de synchronisation. Le début de l'acquisition sera effectif à la prochaine période de l'horloge de synchronisation comme nous pouvons voir sur la figure 9. Cette synchronisation est la plus précise mais nécessite que les cartes soient capables de se synchroniser de cette manière (en interne)

Le principal désavantage de l'utilisation d'un PXI est de ne pas pouvoir s'adapter à tous les composants. Principalement, si le composant n'est pas prévu pour être piloté par les mécanismes de l'architecture, il est alors impossible de les faire cohabiter. Le deuxième point bloquant des PXI est son coût onéreux qui ne permet pas de constituer un système rentrant dans le budget d'un produit commercial.

### 2.1.2 Acquisition par temps

**GPS**

Beaucoup de systèmes d'acquisition utilisent les informations GPS provenant de GPS pour se constituer une base de temps commune. Cette utilisation à principalement l'intérêt d'être disponible sur toute la surface de la Terre et de fournir une synchronisation

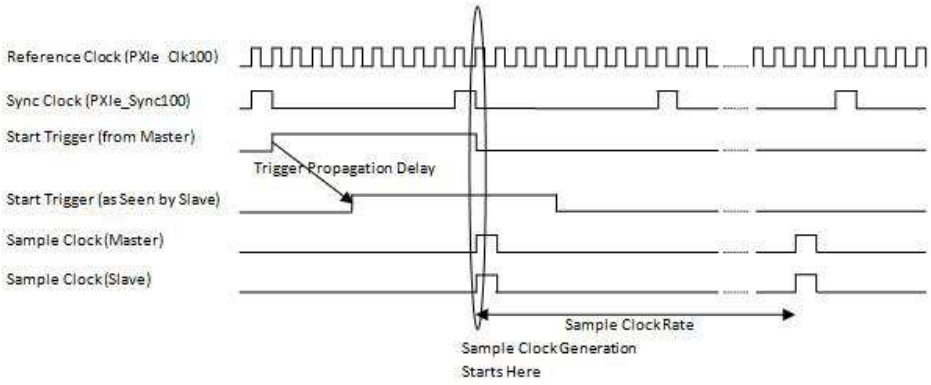



**Figure 9.** Timing de la synchronisation de dérivé d'horloge d'échantillonnage. Source de l'image.

très précise (10 à 20 ns sur des distances intercontinentales [Lewandowski et al., 1993]).

**PTP**

PTP [Eidson and Lee, 2002] est un protocole de synchronisation d'horloge basé sur le protocole réseau Ethernet. Il offre une synchronisation inférieure à 100 ns sur un réseau de données. Il nécessite du matériel dédié ainsi qu'un réseau bien contrôlé pour limiter les variations d'horloge. Le mécanisme de synchronisation est de type maître/escale dans lequel l'esclave se synchronise au maître. Pour ce faire, le protocole est constitué de quatre messages.

- **Sync** : le premier message envoyé par le maître vers l'esclave. Le contenu du message est inutilisé et permet simplement d'entamer une synchronisation. À la réception du message, l'esclave enregistre dans son temps l'heure de réception du message ($t_1$).
- **Follow_Up** : ce deuxième message est envoyé afin de transmettre à l'esclave l'heure d'envoi du message Sync. Le contenu de ce message est donc l'heure d'envoi du premier message dans la base de temps du maître ($t_2$).
- **Delay_Req** : le troisième message est, au contraire des deux premiers messages, envoyé par l'esclave vers le maître. Le message n'a pas de contenu. L'esclave note dans sa base de temps l'heure d'envoi du message ($t_3$).
- **Delay_Resp** : le dernier message est envoyé par le maître en réponse au message Delay_Req avec comme contenu l'heure de réception du message Delay_Req par le maître dans sa base de temps ($t_4$).

La figure 10 récapitule ces transactions. À partir de ces quatre temps ($t_1$, $t_2$, $t_3$ et $t_4$), l'esclave calcule son décalage deux différences le $MS_{difference}$ et le $SM_{difference}$

$$MS_{difference} = t_2 - t_1 = decalage + MS_{delai} \tag{II.1}$$

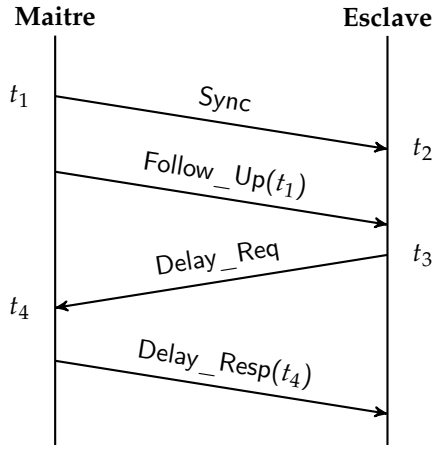



**Figure 10.** Diagramme UML de séquence du protocole PTP

$$SM_{difference} = t_4 - t_3 = -decalage + SM_{delai} \tag{II.2}$$

Dans ces équations, *decalage* est le décalage de l'horloge de l'esclave vers celle du maître, $MS_{delai}$ est le délai de transmission du maître vers l'esclave et $SM_{delai}$ est le délai de transmission de l'esclave vers le maître. Deux hypothèses sont faites sur les délais de transmission, ils sont constants et symétriques. Nous avons donc $MS_{delai} = SM_{delai} = delai$. En résolvant le système d'équations précédent nous trouvons les équations de *decalage* et de *delai* suivant :

$$decalage = \frac{MS_{difference} - SM_{difference}}{2} \tag{II.3}$$

$$delai = \frac{MS_{difference} + SM_{difference}}{2} \tag{II.4}$$

Cette procédure est répétée à intervalle régulier (2 secondes) pour synchroniser l'horloge. Entre deux intervalles, les deux horloges sont en course libre et donc dérive naturellement.

Dans la même tendance, nous pouvons également citer [Mills, 1991] qui n'atteint pas les mêmes performances de précision mais est capable de synchroniser une hiérarchie d'ordinateur sur internet.

**Tableau 2.** Récapitulatif des méthodes de synchronisation

| Solution | Type | Coût | Implémentation | Adaptable | Déterministe | Contrainte |
|---|---|---|---|---|---|---|
| PXI | Signaux | +++ | + | + | oui | |
| GPS | Temps | + | +++ | +++ | non | Extérieur |
| PTP | Temps | ++ | ++ | +++ | non | |

Le tableau 2 récapitule les méthodes de synchronisation présentes dans l'état de l'art. Dans ces méthodes, l'utilisation du protocole PXI demande de refabriquer tous les composants pour s'adapter au système ce qui demanderait un coût trop élevé aussi

bien en temps de développement qu'en coût production. L'utilisation de GPS que nous pourrions utiliser pour utiliser une horloge distribuée sur plusieurs cartes est impossible à cause de l'utilisation en intérieur. Finalement, les protocoles comme PTP atteignent des bonnes performances de synchronisation de données mais exploitent des propriétés statistiques sur lequel nous ne pouvons pas baser notre système car elles sont non déterministes.

## 2.2 Les formats de données

Dans cette sous-section, nous décrivons deux modèles de conteneur de données pour ensuite faire un état de l'art des conteneurs multimédias. Il est bon de noter la différence entre un conteneur et un codec. Un codec est un standard de codage visant à compresser une modalité. Un conteneur est un standard de stockage visant à multiplexer plusieurs modalités. Dans cette section, nous discuterons du premier aspect qui traite du multiplexage.

### 2.2.1 Modèles de conteneur

Un flux de données multimodales peut être représenté de manière hiérarchique [Blakowski and Steinmetz, 1996]. Au plus bas niveau, nous retrouvons des échantillons. Un échantillon est une information indivisible en temps. Elle représente la valeur d'une quantité physique à un moment précis. Pour une séquence vidéo, une image est un échantillon tandis que pour un signal unidimensionnel comme de l'audio, un échantillon est un réel. Dans le dernier cas, les échantillons d'une fenêtre de temps sont encapsulés ensemble afin de créer un LDU[1]. Un LDU est une donnée atomique que traite un conteneur de données. Les échantillons à l'intérieur d'un LDU peuvent être compressés dans un bitstream. Pour cette raison, les LDU sont généralement traités comme des données agnostiques c'est-à-dire qu'aucune hypothèse n'est faite sur le contenu des données d'un LDU. Les LDU sont classés par type de données pour former le dernier niveau hiérarchique : les pistes (vidéo, audio, donnée physiologique). Ils peuvent alors être entrelacés par le conteneur qui se charge de conserver la synchronisation des données. La plupart du temps, les LDU sont classés par ordre chronologique. Pour être complet, un niveau supérieur existe qui multiplexe des flux. Il est utilisé par exemple dans la diffusion télé afin de séparer des chaînes. La figure 11 montre cette hiérarchie d'encapsulation.

**Modèle à échantillonnage constant**

Ce modèle se base sur l'hypothèse que tous les LDU d'une piste sont acquis à une

1. LDU : Logical Data Unit

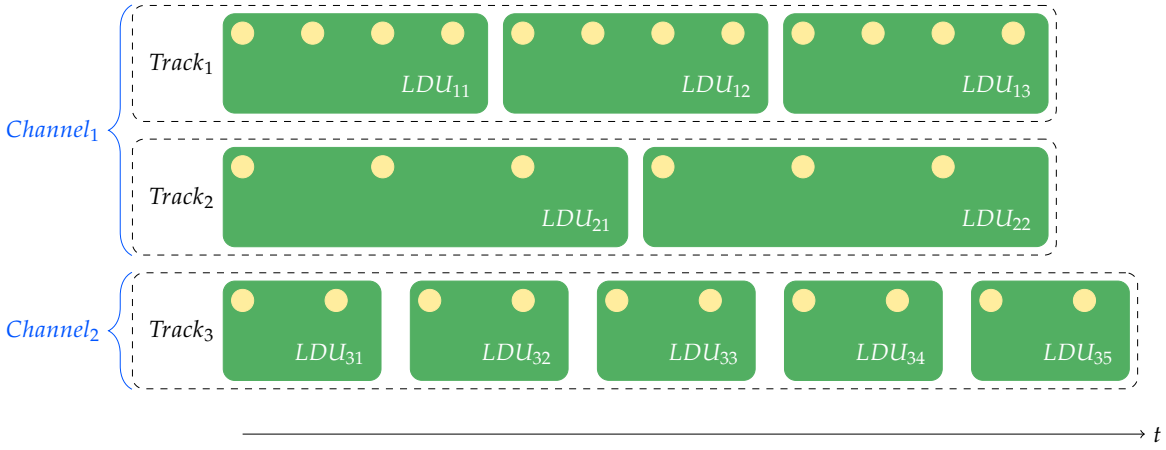



**Figure 11.** Hiérarchie d'encapsulation dans les conteneurs multimédias.

fréquence d'échantillonnage constante. Donc un LDU est placé temporellement dans sa période de temps. La pire erreur temporelle d'un LDU est quand il est exactement entre deux périodes d'horloge donnée par l'équation :

$$\mathcal{E} = \frac{1}{2 \times f_{LDU}} \tag{II.5}$$

La précision temporelle des LDU peut être améliorée en insérant des échantillons factices contenant des valeurs inatteignables ou des échantillons dupliqués. En faisant cela, la fréquence des LDU augmente ce qui permet de réduire l'erreur minimale du modèle. Cependant, en insérant des échantillons supplémentaires, la taille du fichier augmente. Cela devient un compromis entre la précision temporelle et la taille du fichier. La figure 12 illustre deux pistes synchronisées par un conteneur à échantillonnage constant.

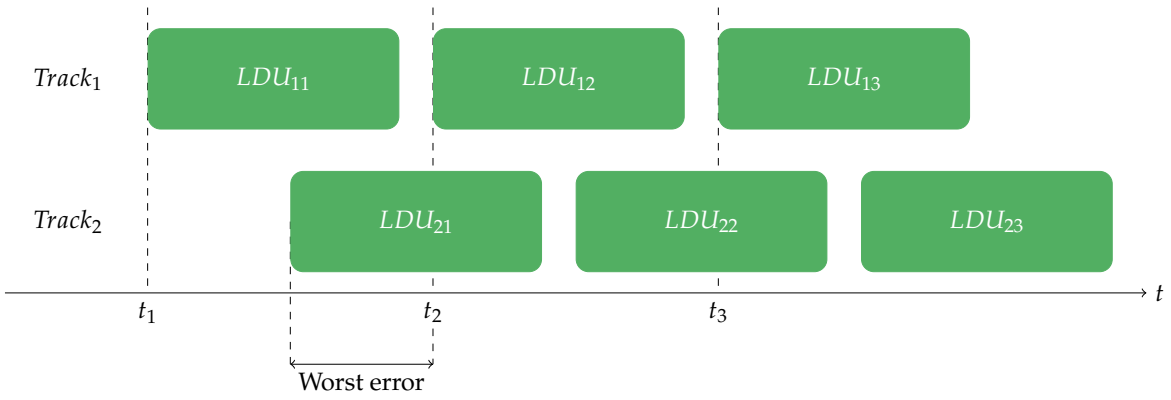



**Figure 12.** Deux pistes synchronisées par un conteneur à échantillonnage constant.

**Modèle à timestamps**

Le deuxième modèle fait l'hypothèse que les LDU ne sont pas à rythme constant. De ce fait, ils sont placés sur une chronologie virtuelle appelée horloge de référence. Cette horloge a une fréquence constante. Chaque période de cette horloge est numérotée de manière monotone et constante. Une valeur de ce compteur représente donc le nombre de périodes passées depuis le début du comptage. La fréquence à laquelle l'horloge de référence est cadencée définit le grain le plus fin auquel nous pouvons placer un LDU. L'erreur minimale de ce modèle ne dépend plus de la cadence d'acquisition des données mais de la fréquence de l'horloge de référence.

$$\mathcal{E} = \frac{1}{2 \times f_{refclk}} \tag{II.6}$$

La largeur du compteur (nombre de bits pour stocker les valeurs) définit le temps à partir duquel le compteur déborde. Généralement, la taille du compteur est choisie afin d'éviter ce cas de figure (des entiers de 64 bits évitent le problème). La figure 13 illustre deux pistes synchronisées par un conteneur à timestamps. L'horloge de référence est visible sur l'échelle de temps.

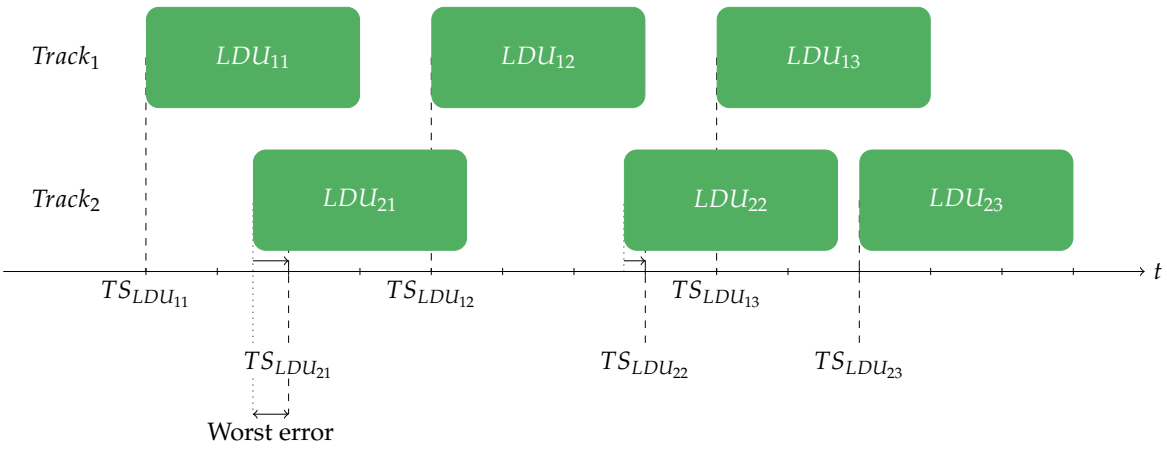



**Figure 13.** Deux pistes synchronisées par un conteneur à timestamps.

### 2.2.2 Conteneurs multimédias

**EDF/EDF+**

Le standard EDF [Kemp et al., 1992] est très largement utilisé dans les hôpitaux afin d'enregistrer des données physiologiques, au point que tous les équipements médicaux utilisent le format EDF ou y sont compatibles. Cette utilisation répandue permet une interopérabilité minimale entre la plupart des équipements des centres hospitaliers même s'ils n'utilisent pas les mêmes outils. De ce fait, conserver la rétrocompatibilité avec le format EDF est aspect important de ces travaux. Cependant, par construction le

format EDF n'a pas été étudié pour incorporer des données multimédias comme une séquence vidéo. De plus, les données enregistrées dans un fichier EDF sont des signaux physiologiques bruts non compressées. Dans le cadre d'un projet de télémédecine, il est nécessaire de pouvoir compresser les données qui devront être téléexpertisées.

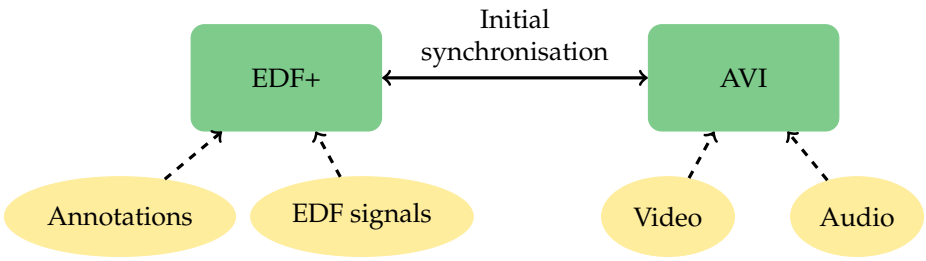



**Figure 14.** Synchronisation des données d'un examen enregistrée en EDF/EDF+

Un fichier EDF+ est composé de deux parties, un entête général contenant la description de l'examen et des signaux contenus. Il est ensuite suivi des enregistrements de données contenant les échantillons acquis. Un enregistrement contient les échantillons de tous les canaux pour une fenêtre de temps. La taille de cette fenêtre et le nombre de canaux sont fixes et définis dans l'entête. La taille d'un échantillon est limitée à 16 bits. Les échantillons sont placés à une fréquence constante qui est définie dans l'entête.

La différence entre EDF+ et EDF réside dans le support des annotations et des données non contiguës en ajoutant un canal de métadonnées dans un fichier EDF classique. La place réservée à ce canal est constante dans tous les enregistrements du fichier qu'elle soit utilisée ou non. Quand la place n'est pas utilisée, l'espace est perdu car il ne contient aucune information.

Afin de stocker les données d'un examen EEG, le standard EDF+ prévoit le stockage des données physiologiques dans le fichier EDF lui-même et le stockage de la vidéo et de l'audio de l'examen dans un conteneur multimédia séparé. La répartition des données comme l'a prévu le standard EDF+ est représentée dans la figure 14. Au moment où l'examen est visualisé (les signaux physiologiques sont dessinés et que la vidéo est affichée), les deux fichiers sont lus simultanément. C'est l'unique point de synchronisation qui existe entre les deux fichiers. Les deux enregistrements ont ensuite chacun leur propre rythme qui peut ne pas être concordant. Il est facilement compréhensible qu'un des deux enregistrements puisse dériver au cours du temps, ce qui entraîne une perte de synchronisation entre les évènements du fichier physiologique et les images du fichier vidéo. Cette méthode d'enregistrement de l'information temporelle est inadaptée quand la relation entre la vidéo et les signaux physiologiques est importante comme dans un examen téléexpertisé. Les désavantages de cette approche sont :

- La synchronisation à la lecture ne peut pas être garantie car l'enregistrement EDF et l'enregistrement vidéo n'ont pas de base de temps commune.
- L'information physiologique est volumineuse. En effet, celle-ci n'est pas compressée ce qui impacte aussi bien la bande passante de la transmission dans le cadre d'une téléexpertise que la quantité d'espace nécessaire au stockage des examens.
- L'insertion et la suppression d'information dans un fichier EDF ne sont pas des tâches triviales qui nécessitent la plupart du temps la recréation d'un fichier.

La société BioSemi [BIOSEMI] a proposé un nouveau format de fichier (BDF/BDF+) pour augmenter la taille des échantillons de 16 à 24 bits.

**AVI**

AVI [1] [AVI] est un conteneur multimédia développé par Microsoft. Il a été développé pour transporter des données audio, vidéo et sous-titre compressés. La fréquence de ces modalités est constante tout au long du fichier. La fréquence des LDU est définie sur un mot de 32 bits contenant la fréquence du LDU. De ce fait, la fréquence maximale exprimable est $2^32$. L'erreur minimale est donc $\mathcal{E} = \frac{1}{2*2^{32}} = 0.11ns$. Cette amélioration de l'erreur étant au coup d'une place mémoire bien augmentée. Le conteneur AVI a été remplacé par le conteneur ASF.

**ASF**

ASF [2] [ASF] est un conteneur orienté objet développé par Microsoft. Les objectifs principaux du développement d'ASF sont l'efficacité pour les applications de streaming et l'indépendance à des systèmes d'exploitation ou de protocole de communication. Il supporte l'inclusion de flux de texte et de données définies par l'utilisateur. L'ASF est un conteneur à timestamps dont les valeurs sont en multiple de 100 ns. Dans le pire cas, les LDU sont placés avec une erreur de 50 ns.

**La famille MPEG**

La famille MPEG-systems (MPEG1-systems [ISO/IEC, 1993], MPEG2-systems [ISO/IEC, 2013] et MPEG4-systems [ISO/IEC, 2010]) spécifie comment des LDU encodés sont identifiés et synchronisés et comment des métadonnées sont rajoutées à un flux. Comme dit précédemment, il ne doit pas être confondu avec les standards de compression du même nom qui définissent comment les LDU sont formés.

Dans MPEG1-Systems qui est une spécification de 1988, la couche de multiplexage ne gère que la vidéo, l'audio et les sous-titres. La couche de synchronisation utilise une horloge de référence cadencée à 90 kHz. L'erreur induite est de 5,5 µs.

La spécification MPEG2-systems améliore les fonctionnalités en rajoutant la possibilité de multiplexer des données définies par l'utilisateur et en augmentant la vitesse

1. AVI : Audio Video Interleaved
2. ASF : Advanced Systems Format

de l'horloge de référence à 127 MHz. La spécification sépare également les usages à un seul flux (program stream) utilisé dans le DVD par exemple des usages multiflux (transport stream) utilisé dans la diffusion télé.

La dernière spécification MPEG4-systems permet à l'utilisateur de définir la fréquence de l'horloge système avec un maximum à $2^{32}$. La plus petite erreur du flux devient 0.12ns. La spécification se base sur une succession de paquets compris dans un ES [1]. Les paquets contiennent des LDU accompagnés d'identifiant et d'un timestamp relatif à l'horloge système. Il est à noter que la spécification des conteneurs MPEG ne prend en charge que les codecs de la spécification du même nom. Pour intégrer d'autres données, il faut passer par les types définis par l'utilisateur. Une ambiguïté est notamment présente pour les normes MPEG qui regroupent les deux aspects :

- MPEG-Systems est un ensemble de norme de multiplexage englobant les fichiers Mov, Mp4 et les flux MPEG-TS et MPEG-PS.
- MPEG1 partie 2 et partie 3, MPEG-2 partie 2, partie 3 et partie 7 et MPEG-4 partie 3, partie 10 et partie 15 qui sont un ensemble de normes de compression englobant les codeurs H264, MP3, H265, AAC et HEVC.

Nous parlons ici de la première partie de la norme.

**MKV**

Le format de fichier Matroska (MKV) est un standard ouvert de conteneur multimédia. Une documentation détaillée de ce format peut être trouvée à la référence suivante [Noé, 2007]. Il est généralement utilisé dans un fichier portant l'extension .mkv et transportant un fichier vidéo avec une piste audio et possiblement une piste de sous-titre. Cependant, celui-ci ne se limite pas à cette utilisation, pouvant être utilisé comme conteneur de fichier audio, de sous-titre ainsi que vidéo 3D ou de flux réseau. Il est important de rappeler qu'un conteneur n'est pas un format de compression vidéo ou audio. De ce fait, le conteneur MKV peut contenir des flux de données compressées ou non. Ces flux peuvent provenir de codec vidéo tel que DivX, H264, VP8, de codec audio comme MP3, AAC ou Vorbis, ou bien de format de sous-titre comme SubRip(srt) ou SubStation Alpha (ssa). MKV gère donc le transport des informations pour chacun des codecs ainsi que les données produites par les différents codecs. Il supporte les fonctionnalités des conteneurs modernes comme la possibilité de sauter dans la lecture du fichier, les entrées de chapitre, les supports des métadonnées (pour un fichier audio par exemple), diffusable sur un réseau, et les menus comme nous pouvons les avoir dans un DVD.

La structure de base utilisée par Matroska est basée sur un dérivé binaire du XML appelé EBML (Extensible Binary Meta Language). Cela permet d'avoir un format de fichier flexible pour le futur en ajoutant de nouvelles étiquettes sans casser la rétrocompatibilité avec les anciennes générations de lecteurs. MKV est une utilisation

1. ES : Elementary Stream

d'EMBL dans le cadre des données vidéo et audio mais n'est pas limité à cet usage. Tout comme dans le format XML, EBML définit un ensemble de balise pouvant contenir des données ainsi que d'autres balises EBML. C'est avec cette approche hiérarchique qu'est construit un flux multimédia MKV. Les balises sont identifiées par un identifiant unique spécifié dans le standard MKV. L'ordre des balises dans un flux MKV n'est pas important, cependant un ordre est préconisé pour faciliter l'écriture de lecteurs MKV compatibles. Nous ne rentrerons pas plus dans les détails de l'implémentation d'EBML. Cependant, la spécification complète du format peut être trouvée sur cette référence [EBML].

**SMIL**

SMIL [1] [Hoschka, 1998] est un langage de synchronisation basé sur le format XML qui est utilisé pour le format de fichier d'image SVG. Il a été développé pour les applications web par le W3C [2]. Son objectif est de fournir une description spatiale et temporelle de scène contenant des médias hétérogènes. Une description SMIL spécifie les relations entre les médias. Les données effectives sont contenues dans d'autres fichiers. Ce n'est pas un conteneur à proprement parler.

Même si SMIL représente le temps en utilisant des timestamps, les informations temporelles sont exprimées de manière relative aux autres objets de la scène. Ceci veut dire que la taille nécessaire pour exprimer les temps est en général plus petite que par rapport à un temps d'origine. Ces informations sont décrites ainsi *h :min :s.ms*. Dans cette expression le nombre de décimales des millisecondes n'est pas limité. En théorie, nous pourrions donc avoir une erreur de zéro.

Cependant, puisque le fichier est un fichier XML, son empreinte mémoire est très grande comparer à un fichier binaire. De plus, SMIL est assez difficilement extensible à de nouvelles données autres que vidéo, images, audio, et textes.

Le tableau 3 résume cette sous-section en comparant les différents formats de fichier. L'erreur minimale est la plus petite erreur atteignable du format. La caractéristique données compressées permet de savoir si le conteneur supporte la compression de données. La colonne timestamps précise si le conteneur est à échantillonnage constant (Non) ou à timestamps (Oui). La caractéristique libre de droits définit si le conteneur peut être utilisé sans royalties. La colonne agnostique permet de savoir si le conteur est capable de transporter des données dont le contenu ne lui est pas connu. Enfin, la dernière colonne, binaire, définit si le flux est un format binaire ou textuel.

Les implémentations à échantillonnage constant ne sont pas prises en considération car les méthodes pour augmenter la précision de synchronisation augmentent également le volume de données. De plus, SMIL ne rentre pas dans nos contraintes de volume de données à cause de sa représentation textuelle. Les formats ASF, MPEG4-systems et

1. SMIL : Synchronized Multimedia Integration Language
2. W3C : World Wide Web

**Tableau 3.** Résumé de la comparaison des performances et des fonctionnalités des conteneurs.

| Conteneur | Erreur minimale | Données compressées | Timestamps | Libre de droit | Agnostique | Binaire |
|---|---|---|---|---|---|---|
| EDF+ | 50fs | Non | Non | Oui | Non | Oui |
| AVI | 0.11ns | Oui | Non | Non | Non | Oui |
| ASF | 50ns | Oui | Oui | Non | Non | Oui |
| SMIL | unlimited | Oui | Oui | Oui | Non | Non |
| MPEG 1 systems | 5.5us | Oui | Oui | Non | Non | Oui |
| MPEG 2 systems | 4ns | Oui | Oui | Non | Oui | Oui |
| MPEG 4 systems | 0.1ns | Oui | Oui | Non | Oui | Oui |
| MKV | 0.5ns | Oui | Oui | Oui | Oui | Oui |

MKV sont tous de bons candidats pour l'utilisation dans un projet de télémédecine.

# 3 Compression vidéo

Cette section est consacrée à l'état de l'art de la compression vidéo. Celui-ci commence par des notions sur la compression de données pour ensuite établir les codecs d'image fixe puis les codecs vidéo.

## 3.1 Notions de compression

Dans cette sous-section, nous définirons toutes les notions utiles à la compréhension de la section et en particulier, les notions de théorie de l'information.

### 3.1.1 Systèmes communicants

Dans son article [Shannon, 2001] [1], Cloud Shannon définit un système généralisé d'une communication entre deux objets. La figure 15 schématise ce système composé d'une source d'information produisant un message à communiquer à la destination, d'un transmetteur transformant le message pour être adapté à la transmission sur le canal, d'un canal pouvant être perturbé par du bruit extérieur et transportant le message entre le transmetteur et le receveur, d'un receveur réalisant l'opération inverse du transmetteur et d'une destination à qui le message est destiné.

Ce modèle modélise bien l'application de compression, dans laquelle la source d'information est la donnée à encoder, le transmetteur est le compresseur, le canal est le réseau ou le fichier résultant de la compression, le receveur est le décodeur et la destination est la donnée décompressée. Nous considérerons, dans notre étude,

1. Réédition avec des corrections du Bell System Technical Journal, Vol. 27, pp. 379–423, 623–656, juillet, octobre, 1948.

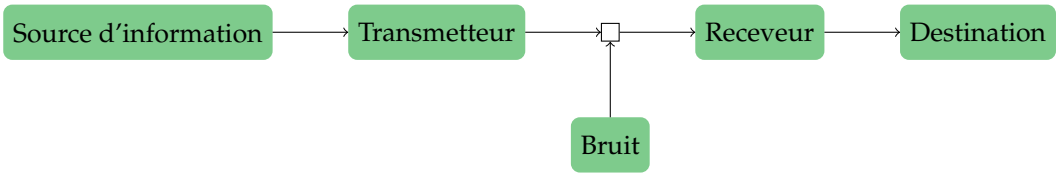



**Figure 15.** Diagramme généralisé d'un système communicant

simplement les systèmes discrets sans bruit, discrets car les données que nous étudions sont des données numériques et sans bruits car nous supposerons que les stockages ou réseaux utilisés sont sûrs.

## 3.2 Entropie au sens de Shannon

Le message envoyé par la source d'information est une succession de symboles. Les symboles constituent un ensemble $\{a_1, ..., a_{|\mathcal{A}|}\}$ appelé alphabet $\mathcal{A}$ dont la cardinalité $|\mathcal{A}|$ est le nombre de symboles possibles. À chaque symbole $a_i$ est associé une probabilité $P(X = a_i)$ d'apparition en provenance de la variable aléatoire X, où $\sum_{a_i} P(X = a_i) = 1$

D'après Shannon, la quantité d'information pour exprimer un symbole $a_i$ à l'aide des symboles d'un deuxième alphabet $\Omega$ est :

$$I(X = a_i) = -log_{|\Omega|} P(X = a_i) \tag{II.7}$$

Le deuxième alphabet $\Omega$ étant généralement composé des deux symboles $\{0, 1\}$, de manière à exprimer la quantité en bits par symbole. De cette première équation découle l'entropie d'une source définie comme la somme pondérée des quantités d'information de chaque symbole par sa probabilité d'apparition :

$$H(X) = \sum_{a_i \in \mathcal{A}} P(X = a_i) I(X = a_i) \tag{II.8}$$

Ou développée :

$$H(X) = -\sum_{a_i \in \mathcal{A}} P(X = a_i) log_{|\Omega|} P(X = a_i) \tag{II.9}$$

Cette entropie définit la longueur moyenne des symboles de l'alphabet $\Omega$ permettant d'exprimer de manière optimale un symbole de l'alphabet $\mathcal{A}$. Entre autres, l'entropie est représentative du facteur de compression moyen sans perte atteignable par un compresseur d'une source d'information connaissant, a priori, la distribution des symboles de son alphabet. C'est donc une bonne métrique pour évaluer si un prétraitement à un potentiel d'être compressé avec un bon taux de compression.

## 3.3 Le codage entropique

Le codage entropique est une procédure de conversion d'une source de symboles d'un alphabet à un autre, en exploitant les statistiques d'occurrences des symboles du premier alphabet afin de les remplacer par des symboles de longueur variable d'un deuxième alphabet. Plus un symbole apparaît souvent, plus nous lui associerons un symbole de petite longueur. Nous parlons aussi de codage à longueur variable.

Nous prenons comme exemple l'alphabet $\mathcal{A}$ et la distribution de ces symboles du tableau 4. La longueur des symboles de l'alphabet initial est $log_2(|\mathcal{A}|)$ bits, c'est-à-dire 2 bits. À partir de cette distribution, nous définissons un code binaire de longueur variable pour chaque symbole. La longueur moyenne pondérée par la probabilité d'occurrence d'un symbole est égale à $1,75$. Si le message que nous désirons encoder suit la distribution énoncée, alors le message devrait s'encoder avec moins de bits qu'avec l'alphabet initial. En utilisant le codage donné en exemple, le message *AABCAADABB* s'encode vers la suite de symbole *0 0 10 110 0 0 111 0 10 10*, les espaces étant rajoutés afin d'améliorer la lisibilité du message. En utilisant l'alphabet $\mathcal{A}$, la longueur du message est $tailleSymbole * nombreSymboles = 2 * 10 = 20bits$. En comparaison, le même message encodé est de longueur 17 bits.

**Tableau 4.** Exemple d'alphabet, sa distribution et une possibilité de codage

| Symbole | Probabilité | Code | Longueur |
|---|---|---|---|
| A | 1/2 | 0 | 1 |
| B | 1/4 | 10 | 2 |
| C | 1/8 | 110 | 3 |
| D | 1/8 | 111 | 3 |

Nous pouvons évaluer l'efficacité de l'encodage en calculant le rapport de la taille des messages d'entrée par la taille des messages de sortie. Ce rapport est appelé le facteur de compression.

$$Facteur\ de\ compression = \frac{Nombre\ de\ bits\ en\ entrée}{Nombre\ de\ bits\ en\ sortie}$$

Pour notre exemple, le facteur de compression est de $20/17 \approx 1,17$, ce qui est plus généralement écrit 20 : 17.

Il est important de noter qu'un encodage ne peut pas être toujours bénéfique, notamment quand le message à encoder ne suit pas la distribution initialement prévue. Il peut même des fois avoir un effet négatif et rallonger la taille des données. Par exemple, le message *CDCBA* s'encode ainsi *110 111 110 10 0*, ce qui nous donne un codage de 12 bits contre 10 bits pour le message initial.

#### Compression à perte

Jusque maintenant, nous n'avons parlé que de compression entropique dite compres-

sion sans perte, c'est-à-dire que les données envoyées par l'émetteur sont identiques aux données reçues par le récepteur. Cette méthode permet dans le cas de la compression d'image une reproduction parfaite pour le décodeur au prix d'un taux de compression assez faible (de l'ordre de 1 à 3). Afin d'augmenter les taux de compression [Storer, 1988], les algorithmes de compression utilisent des méthodes de dégradation sélective de l'information qui a pour but de réduire l'entropie des images encodées. Nous parlons alors de quantification. Celle-ci constitue l'étage d'un codeur ou le plus de pertes d'information sont constatées. Il est donc utile de bien choisir les informations à supprimer afin de limiter l'impact qu'elle aura lors de la reconstitution. Pour ce faire, l'information supprimée sera dépendante de la perception humaine de l'image compressée. C'est pour cette raison que l'information d'une image est prétraitée en décomposant par fréquence et par couleur la matrice de base pour ne filtrer que les informations dont l'œil humain est capable de voir. Par exemple, l'œil humain perçoit beaucoup plus les basses fréquences dans une image que les hautes fréquences. Du coup, un codeur aura tendance à conserver les basses fréquences intactes et dégrader les hautes fréquences pour maximiser l'entropie du message à encoder.

## 3.4 Codeur d'image fixe

### 3.4.1 Codeur JPEG

JPEG [Wallace, 1992][ISO/EIC, 1991] est un format d'image bien connu du grand public car très répandu comme support sur internet. Il a été développé pour compresser une large gamme de types d'images (niveau de gris, noir et blanc et couleur) avec des caractéristiques différentes (synthétique, naturelle, médicale, etc.). Il permet à des systèmes de se transférer des images en utilisant une faible bande passante. JPEG possède un mode sans perte et avec perte.

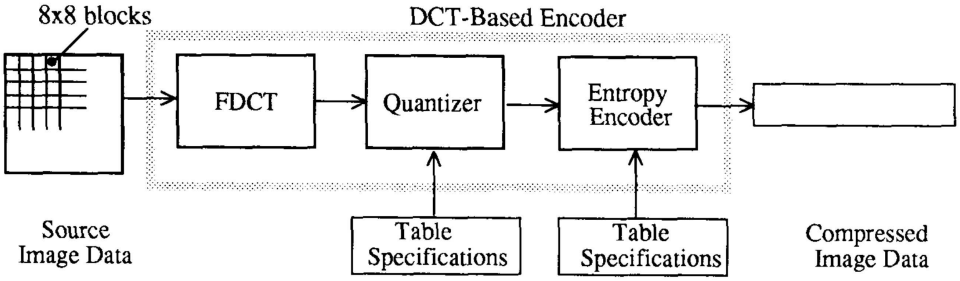



**Figure 16.** Schéma bloc du codeur JPEG. Source de l'image.

La figure 16 illustre le schéma bloc du codeur. Techniquement, la compression est basée sur une FDCT [1] effectuée sur des blocs de taille 8x8. Les coefficients de ces blocs

1. FDCT : Fast Discret Cosine Transform

sont ensuite quantifiés en fonction de leur fréquence. Les coefficients sont ensuite parcourus en zig-zag pour être encodés avec l'encodage de Huffman [Huffman, 1952] ou l'encodage arithmétique [Rissanen and Langdon, 1979].

### 3.4.2 Codeur JPEG-2000

JPEG-2000 [Christopoulos et al., 2000][ISO/IEC, 2000] est le successeur de JPEG utilisant les avancés techniques du domaine. L'évolution majeure est l'utilisation d'une transformée en ondelettes discrète et l'encodeur EBCOT[Taubman, 2000] qui est un encodeur entropique spécialement développé pour l'encodage multi résolution que produit la transformée en ondelettes. Le schéma bloc de JPEG-2000 en figure 17 est organisé de manière similaire à celui de JPEG en trois étages : transformation, quantification et encodage. Les performances de JPEG-2000 sont améliorées de 2 dB sur toute la plage de compression par rapport à JPEG-2000 [Santa-Cruz and Ebrahimi, 2000]. Il apporte également des fonctionnalités :

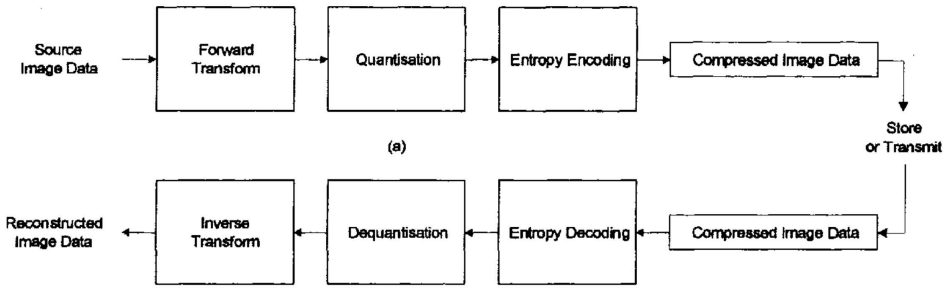



**Figure 17.** Schéma bloc du codeur JPEG-2000. Source de l'image.

- Transmission en résolution progressive dans le bitstream, permettant d'avoir une résolution par pixel de plus en plus précise en décodant le flux.
- Encodage de zone d'intérêt en utilisant une méthode de MaxShift [Tahoces et al., 2008] dissociant les plans de bits des pixels d'une zone d'intérêt des pixels du fond de la scène. Cela permet de placer la zone d'intérêt en premier dans le bitstream afin de la décoder en premier. Il est également possible de limiter le nombre de plans de bits des pixels de fond et réduire leur volume.
- L'accès à des parties de l'image de manière aléatoire, permettant de décoder qu'une zone de l'image.
- Rajout d'un canal de transparence (alpha).

Malgré toutes ces fonctionnalités et qualités, JPEG-2000 n'a pas rencontré le succès du JPEG dans le grand public et reste utilisé dans des domaines spécifiques comme l'imagerie médicale.

### 3.4.3 Codeur Waaves

Waaves [Waaves] est un codec d'images médicales développer par la société CIRA. Des études ont montré que ce codec atteignait des meilleurs taux de compression que JPEG-2000 à qualité égale. Le compresseur supporte la compression d'image sans perte et avec perte. Dans ce dernier mode, Il est capable d'atteindre des taux de compression équivalents ou meilleurs que JPEG-2000 à qualité comparable. Le graphique 18 montre que les implémentations JPEG-2000 (kakadu, jasper et acdsee) offre à taux de compression équivalent une qualité d'image meilleur.

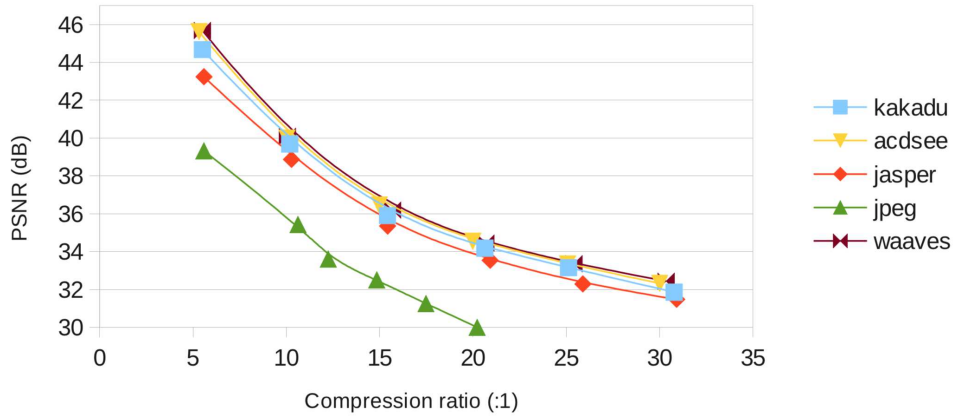



**Figure 18.** Comparaison des performances d'implémentation JPEG, JPEG-2000 et Waaves.

Le schéma bloc de la figure 19 montre le flot de données pour la compression. Nous pouvons y voir une transformation couleur qui est adaptable vers plusieurs espaces couleurs. Ensuite, les canaux de l'image sont transformés dans l'espace ondelette en utilisant une DWT. Le contrôle de flux est réalisé par une quantification uniforme. Le traitement de balayage et réorganisation adaptatif réalisé en fonction d'une métrique d'activité statistique appelée EAM [1]. Les coefficients réorganisés sont ensuite encodés avec HENUC [Öktem, 1999].

D'un point de vue de l'implémentation, le bloc de réorganisation est le plus coûteux en temps de calcul. En effet, l'algorithme demande de calculer leur EAM, de trier tous les coefficients en fonction de ce critère, de sélectionner le plus grand coefficient, puis de refaire ce traitement pour chaque coefficient jusqu'à qu'ils soient tous traités. Ce codec a fait l'objet de deux thèses au LIP6, la thèse d'Imen Medhbi [Mhedhbi et al., 2014][Mhedhbi et al., 2012] visant à adapter le codec Waaves pour compresser des séquences d'images médicales et la thèse de Yuhui Bai [Bai et al., 2014][Bai et al., 2013] portant sur une implémentation mixte matérielle/logicielle de cet algorithme afin d'accélérer ses performances. Ces derniers travaux ont

1. EAM : Expected Activity Measure

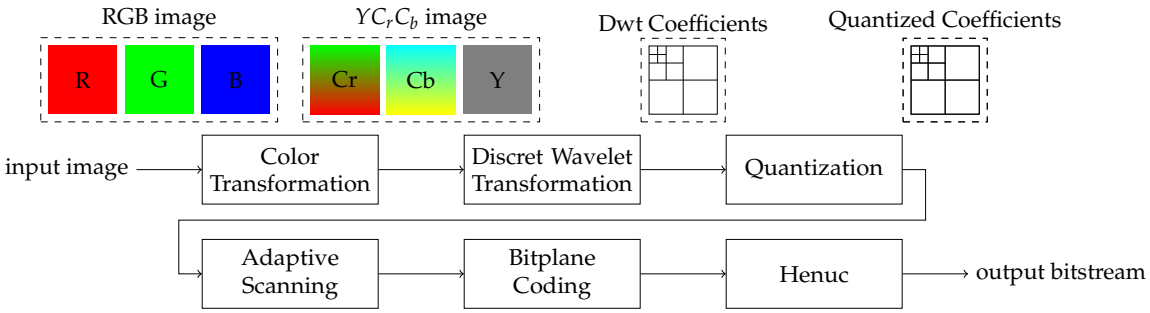



**Figure 19.** Schéma bloc du codeur Waaves.

permis une implémentation partielle de Waaves atteignant des vitesses de compression dépassant le 10 Mo/s. Waaves est certifié dispositif médical au titre CE. Cette certification favorise son utilisation dans un système devant être certifié médical. Cependant, les taux de compression atteints ainsi que la vitesse de compression ne permettent pas d'encoder et d'envoyer une vidéo à 100 images par seconde sur des serveurs distants.

## 3.5 Codeur vidéo

### 3.5.1 H264

Le codec H264 [Wiegand et al., 2003][ISO/IEC, 2003] est un algorithme de compression vidéo de 2003 développé par l'ITU-T Video Coding Experts Group et le ISO/IEC Moving Picture Experts Group dont le principal effort a été d'avoir une performance de compression optimale et être utilisable facilement sur des réseaux. La manière de compresser une image est beaucoup plus complexe que les codeurs JPEG et Waaves. Une image classique est encodée en utilisant une transformée en cosinus et un codeur adaptatif de contexte évolué par rapport au codeur arithmétique classique. En addition, des images prédictives sont encodées comme étant la résultante d'une prédiction du mouvement par rapport à une autre image du flux. Cette prédiction est l'étape de calcul très complexe de H264. À la différence de JPEG, JPEG-2000 et Waaves, H264 est un codeur asymétrique c'est-à-dire que la structuration d'encodage est différente du décodage. Entre autres, le décodage est moins demandant en calcul. La figure 20 représente le schéma bloc de l'encodeur H264.

### 3.5.2 Codeur VP8

VP8 [Bankoski et al., 2011][Wilkins et al., 2011] est un codeur vidéo développé par la société on2. La société a été rachetée par Google en 2010 afin d'intégrer son codeur dans son projet WebM visant à développer un format open source pour le web et notamment pour son service Youtube. Techniquement, le codec est assez proche dans l'ensemble de H264. Il reprend les concepts d'estimation de mouvement et de prédiction

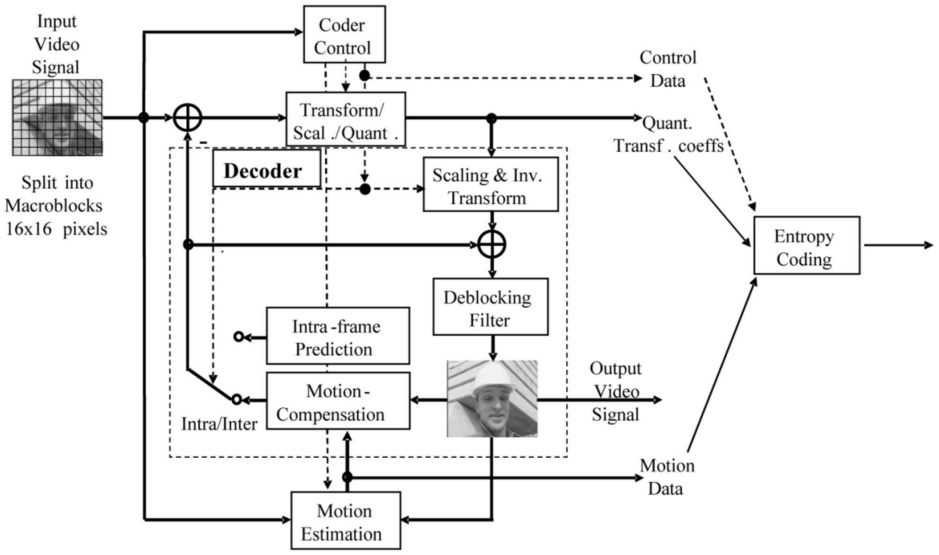



**Figure 20.** Schéma bloc du codeur H264. Source de l'image.

d'image tout en se basant sur la transformée en cosinus et un codage arithmétique. Le schéma bloc de la figure 21 met en avance la ressemblance avec le codeur H264. Un des points qui a retenu notre attention est la disponibilité d'un encodeur VP8 matériel proposé par Google sous forme d'IP. Cependant, nos demandes pour avoir ce codec n'ont pas donné suite.

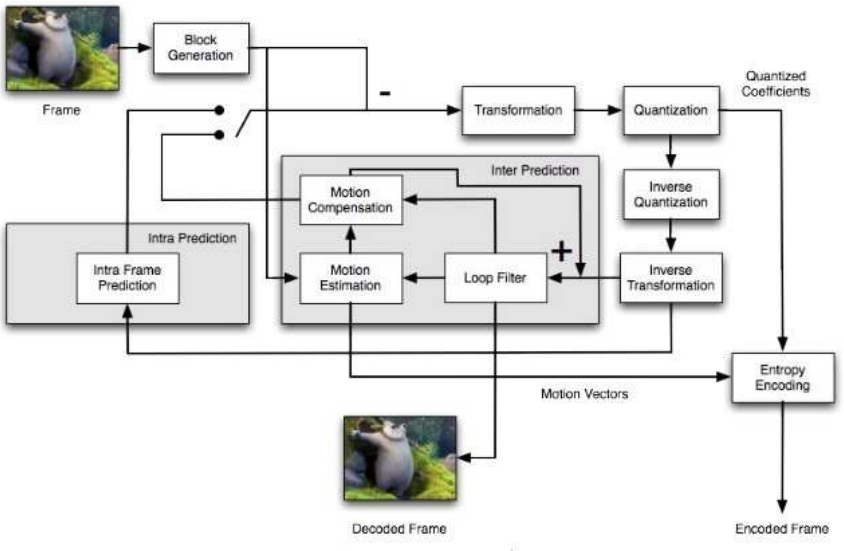



**Figure 21.** Schéma bloc du codeur VP8. Source de l'image.

**Analyse**

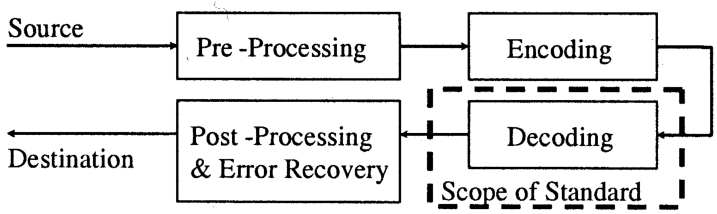



**Figure 22.** Périmètre des standards VP8 et H264. Source de l'image.

Les standards JPEG, JPEG-2000, H264 et VP8 sont des normes dont le périmètre est la spécification d'un format de fichier compressé et la manière dont il est décodé (figure 22). De ce fait, l'implémentation d'un compresseur est laissée libre d'implémentation de manière à permettre aux développeurs de réaliser des améliorations dans l'algorithme de compression sans changer le format. De ce fait, deux implémentations du même codeur peuvent donner des résultats qui ne concordent pas (qualité différente par exemple). À cause de ce point précis, il est impossible de certifier comme dispositif médical un de ces formats. Waaves est, quant à lui, un algorithme de compression dont l'unique implémentation a été certifiée dispositif médical ce qui en fait le seul codeur qui peut être utilisé légalement dans des applications de compression d'image médicales. Cependant, le codeur Waaves ne satisfait pas les besoins de performance de compression et en vitesse de compression dont nous avons besoin pour comprimer le flux vidéo à 100 images par seconde pour une utilisation en télémédecine.

# 4 Conclusion

Après analyse de l'état de l'art, nous avons présenté les dispositifs disponibles sur le marché. De ces équipements, nous avons pu conclure l'origine des problèmes que les médecins ont soulevés. La synchronisation peu précise venant du manque de déterminisme sur les équipements d'acquisition et du format de fichier ne stockant pas les informations temporelles. La qualité et la vitesse de la vidéo acquise ne sont pas assez bonnes pour réaliser des diagnostics. De ce fait, nous avons réalisé deux états de l'art. Un premier sur les aspects de synchronisation pour l'acquisition et sur les conteneurs de flux multimédia. Les méthodes de synchronisation matérielle ne permettent pas de réaliser une synchronisation déterministe en utilisant des composants du commerce. De plus, il n'existe pas de conteneur multimédia réalisant la synchronisation des données vidéo et physiologiques. Le second état de l'art sur la compression vidéo dans lequel nous avons exploré les algorithmes de compression utilisable pour la séquence vidéo. Dans ces algorithmes, aucun ne permet d'encoder un flux vidéo à 100 images par

seconde que nous pourrions certifier comme dispositif médical.

De ces deux états de l'art et de l'analyse des équipements, nous avons dégagé trois questions qui guideront notre raisonnement dans les prochains chapitres. Ces trois questions sont les suivantes :

---

Pouvons-nous définir des **mécanismes de synchronisation déterministes** conservant la synchronisation de l'acquisition des données jusqu'à la **relecture** et l'archivage d'un examen en utilisant des dispositifs d'acquisition du **marché** ?

Comment **compresser** un flux vidéo à **haute fréquence** d'acquisition en conservant une **qualité** nécessaire aux diagnostics et pouvant être **certifié** comme dispositif médical ?

Comment **implémenter** un dispositif d'acquisition d'examen d'EEG acquérant un flux vidéo à **100 images par seconde** et réalisant en **flux tendu** la **synchronisation**, la **compression** et la **transmission** pour une téléexpertise ?

---

Chapitre III

# Synchronisation



Dans ce chapitre, nous parlons des concepts introduits pour assurer la synchronisation des données de l'examen afin que celui-ci puisse être relu et analysé plus tard par un neurophysiologiste. Pour rappel, les médecins ont constaté des limitations dans l'utilisation des systèmes d'acquisition actuels sur ce point. Nous avons analysé ces systèmes et en avons fait les deux constats suivants. Premièrement, les méthodes d'acquisition des différentes modalités sont indéterministes, car basées sur des protocoles et des systèmes non temps réels (USB et Windows). De ce fait, il n'y a pas de garantie pour corréler les différentes modalités sur l'aspect temporel dès l'origine de l'acquisition d'échantillon. Deuxièmement, les formats de fichier utilisés dans les systèmes d'acquisition du commerce sont soit fermés (donc impossible à utiliser) soit ne permettent pas le stockage des informations temporelles pour la relecture de l'examen.

La structuration du chapitre suit ces deux constats. Dans un premier temps, nous donnons notre solution permettant de synchroniser l'acquisition des données au niveau matériel. Nous utilisons, pour ce mécanisme, le terme de synchronisation de bas niveau. Dans un deuxième temps, nous nous penchons sur la manière de conserver les informations temporelles établies pendant la synchronisation de bas niveau après compression des différentes modalités. Nous parlons, par opposition, de synchronisation de haut niveau.

# 1 Synchronisation bas niveau

Dans cette section, nous introduisons dans un premier temps la synchronisation de bas niveau de manière théorique et générale. Nous abordons ensuite les trois cas d'équipements d'acquisition utilisés dans cette thèse, une caméra et un système d'acquisition de signaux physiologiques et un codec audio.

**Le cas général**

Nous définissons la synchronisation de bas niveau ainsi : La synchronisation bas niveau est l'ensemble des mécanismes matériels mis en place afin d'enregistrer les *évènements* caractéristiques par des équipements d'acquisition.

Cette définition de la synchronisation est très importante car elle diverge de la vision traditionnelle que l'on peut avoir du synchronisme. En effet, la synchronisation est vue par beaucoup comme les mécanismes permettant d'harmoniser des valeurs physiques sur plusieurs systèmes en même temps. On parle plus particulièrement de synchronisation de temps quand le but est de synchroniser plusieurs horloges pour aligner leur phase et leur fréquence.

Nous faisons donc le pari de ne pas synchroniser nos équipements d'acquisition avec ce schéma traditionnel. Nous justifions cette décision, car nous avons pu observer que pour certains équipements, il est techniquement impossible de guider ou d'actionner les acquisitions d'un échantillon à un instant précis. Effectivement, beaucoup d'équipements fonctionnent dans un mode d'acquisition appelé *"free-running"*, c'est-à-dire que l'acquisition est lancée avec une commande de démarrage et l'équipement acquiert et retourne des données en continu en suivant une fréquence de consigne. Même si ce fonctionnement permet de synchroniser le départ de l'acquisition et donc en quelque sorte la phase de l'horloge, elle empêche d'avoir le contrôle sur la fréquence de cette dernière. Il est donc impossible de rétablir l'alignement en fréquence dans le cas où deux fréquences d'horloge venaient à diverger ou bien n'ayant pas exactement la même cadence d'opération.

Par exemple, deux équipements sont paramétrés avec des consignes d'acquisition à 1 Hz. Les deux équipements génèrent leur horloge interne afin d'acquérir des données périodiquement. Le deuxième équipement génère une horloge un centième fois plus lente que le premier. Donc, dans le référentiel du premier équipement, l'horloge du deuxième équipement est cadencée à 0,99 Hz. Attention, toutes ces valeurs sont relatives au référentiel du premier équipement. D'après cet exemple, au bout de cent secondes, le deuxième équipement aura un échantillon de retard. Ce qui décale la lecture des échantillons dans le cas où aucune mesure n'est mise en place pour la corriger.

Sachant que nous avons pour objectif d'utiliser des composants pris sur l'étagère [1], il

1. Composants pris sur l'étagère est un composant fabriqué en série. Il oppose les composants

est difficile d'imaginer de devoir rajouter comme contrainte, la nécessité de pouvoir déclencher les acquisitions avec une horloge externe. Cette contrainte est notamment difficile à satisfaire dans le domaine de l'acquisition de signaux physiologiques dont nous faisons partie.

Nous faisons donc le pari de n'être qu'observateur des différents périphériques plutôt que les forcer à acquérir. Attention, cela ne veut pas dire que nous n'utilisons pas les signaux de déclenchement d'acquisition si ceux si sont disponibles, mais que nous traitons ces signaux avec une vision externe comme s'ils n'étaient pas générés par notre système. De cette manière, nous avons défini un mode d'échange commun pour les équipements. Il comprend les deux interfaces suivantes (comme illustré dans la figure 23) :

- **Données** : une interface permettant de récupérer des données de l'équipement. Il n'y a pas de contrainte sur cette interface, elle peut très bien être *"push"*, c'est-à-dire que l'équipement pousse ces données, ou *"pull"*, c'est-à-dire que l'équipement attend qu'un autre équipement vienne récupérer ces données. De plus, aucune contrainte de latence ni de bande passante n'est imposée sur cette interface, tant que le périphérique peut stocker les données sans perte.
- **Déclencheur** : un signal d'acquisition booléen émit par l'équipement. Un front montant est déclenché sur ce signal quand une acquisition a lieu. Ce signal est donc soit directement l'horloge servant à déclencher l'acquisition dans l'équipement, soit un signal suffisamment proche de cette horloge pour compenser l'absence du premier signal. La proximité temporelle du front montant de ce signal avec l'occurrence de l'évènement à enregistrer définira en partie la précision atteignable par notre mécanisme.

Une règle doit être cependant suivie, il doit y avoir une relation un pour un, stricte, entre le nombre de fronts montants et le nombre de données acquises. Dans cette définition, une donnée exprime une unité d'acquisition que le système veut positionner temporellement. Par exemple, dans notre système, une image ou un échantillon de signal physiologique est une donnée acquise.

Dans notre solution, un dispositif est ajouté comme nœud central à ces périphériques d'acquisition. Ce dispositif a pour simple objectif d'enregistrer les informations de temps des évènements des périphériques d'acquisition. Nous appellerons ce dispositif l'*horodateur*. Il est constitué d'une horloge évoluant à une fréquence $f_{sys}$ et d'un compteur. Le compteur est incrémenté à chaque période de temps de l'horloge. Cette horloge sera désignée comme l'horloge système. Chaque cycle de cette horloge peut donc être identifié par la valeur spécifique du compteur à l'instant du cycle. Il y a deux hypothèses de faites pour ce dispositif. La première est que la fréquence $f_{sys}$ est supposée être supérieure à la plus grande des fréquences d'acquisition des périphériques. La deuxième

*"customs"* faits pour un projet en particulier.

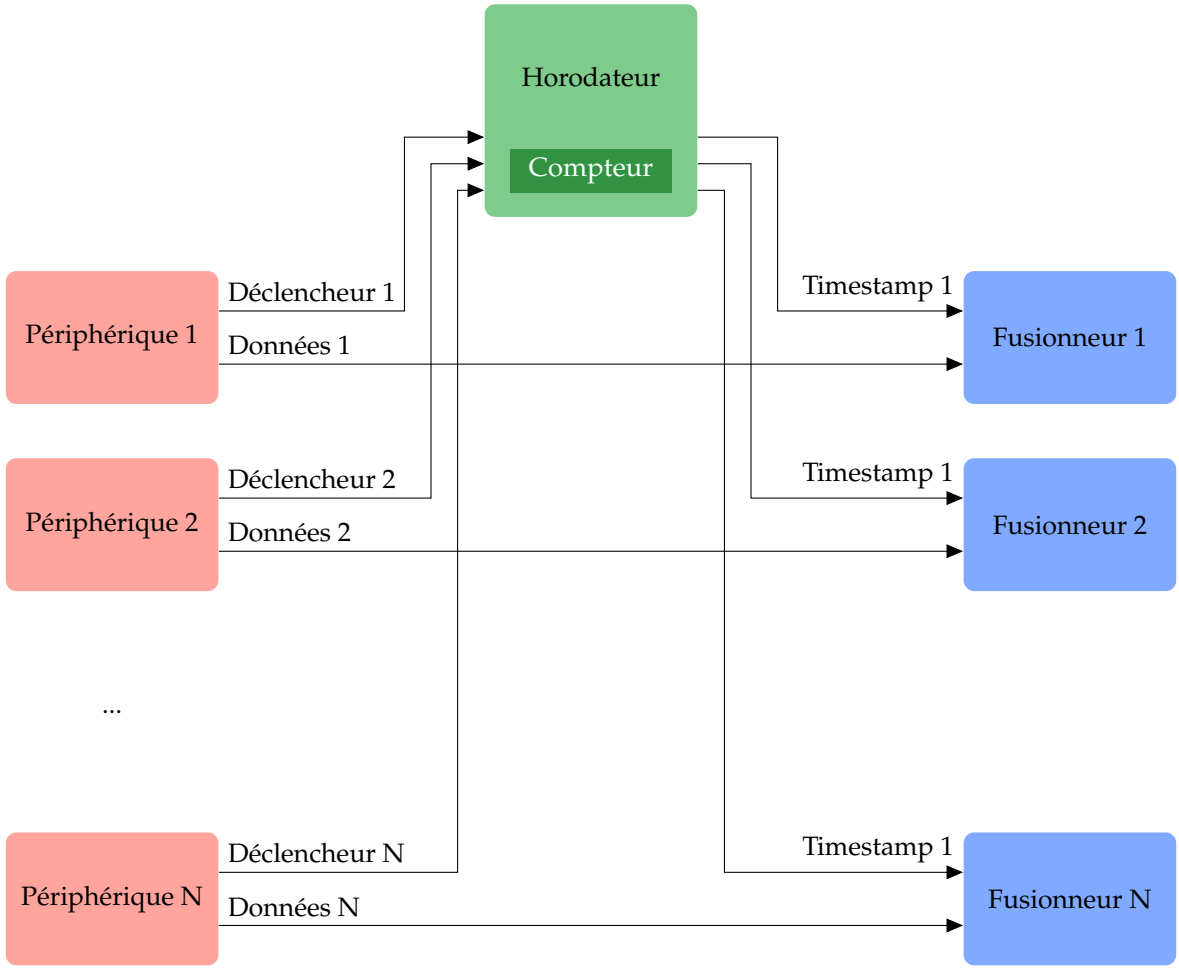



**Figure 23.** Schéma de synthèse du mécanisme de synchronisation bas niveau

hypothèse, qui relève plus de questions de commodité, est que le compteur ne peut pas prendre deux fois la même valeur. Cette hypothèse permet de garantir qu'un seul identifiant désigne un seul cycle de l'horloge système.

En pratique, cela consiste à définir un stockage suffisamment grand pour contenir des identifiants assez grands. Pour le reste du raisonnement, nous ferons l'hypothèse que l'horloge système évolue à une fréquence de 1 GHz. À l'initialisation du système, la valeur du compteur vaut zéro. Après une seconde d'opération, le compteur aura la valeur un milliard, au bout d'une minute, soixante milliards, après une heure, trois mille six cents milliards. Nous ferons l'hypothèse que la taille maximum d'un enregistrement EEG est d'une semaine ce qui correspondrait à un compteur atteignant presque la valeur six cent mille milliards ($1000000000_{nanosecondes} * 60_{secondes} * 60_{minutes} * 24_{heures} * 7_{jours} = 604800000000000_{nanosecondes}$). Cette valeur peut être stockée sur un entier non signé de 50 bits. Nous utiliserons donc un compteur supérieur à 50 bits. Pour le reste de la thèse, si rien n'est précisé, les valeurs typiques utilisées seront 1 GHz pour l'horloge système et 64 bits pour la taille du compteur. Pour finir, l'horodateur restitue les valeurs de ce compteur à chaque déclenchement, dans l'ordre où ils ont été émis. À partir

de ce moment, ces valeurs sont appelées des *timestamps*. Un dernier dispositif appelé fusionneur regroupe les données d'une modalité avec leur *timestamp*. Ce dispositif lit simplement un *timestamp* et une donnée pour les faire suivre ensemble au reste de la chaîne de traitement.

La figure 23 résume les différents éléments composants le mécanisme de synchronisation bas niveau. À droite de la figure sont représentés les périphériques d'acquisition avec leurs signaux de données et leur déclencheur. À partir des déclencheurs et de la valeur du compteur interne, l'horodateur génère des *timestamps* qu'il livre aux fusionneurs. Les fusionneurs lisent ces *timestamps* et les données du périphérique pour les transmettre ensuite au reste du système. Nous restons assez évasifs pour le moment sur l'implémentation de ces modules, mais elle sera détaillée dans le chapitre V. Nous décrivons dans les paragraphes suivants les trois périphériques utilisés dans le cadre de cette thèse.

**La caméra**

Pour nos expériences, nous utilisons une caméra industrielle (BASLER acA2000-340kc [Basler]). Cette caméra peut être pilotée par un déclencheur externe. Le déclencheur sera donc généré par nos soins et transmis à la caméra (notre équipement) et à l'horodateur. Pour récupérer les données, le modèle intègre une interface *CameraLink* pour la communication avec d'autres équipements. Cette interface nous permet de configurer la caméra ainsi que de récupérer l'image acquise à chaque déclenchement.

**L'acquisition des signaux physiologiques**

Pour l'acquisition des signaux physiologiques, nous avons utilisé les convertisseurs analogiques-numériques TI ADS1298 [ADS1298] et TI ADS1299 [ADS1298]. A contrario de la caméra, les convertisseurs ne peuvent pas être pilotés avec un signal de déclenchement pour prendre des échantillons à des instants précis. Nous utiliserons donc un signal de sortie du composant qui transmet l'acquisition d'un échantillon. Le signal le plus proche de l'acquisition d'un échantillon est le *DRDY* qui notifie la disponibilité d'une donnée dans les mémoires du composant. Ce signal n'est pas idéal car il ne signale pas l'acquisition d'un échantillon mais le composant ne nous permet pas d'avoir plus d'information interne que celle-ci.

Nous avons, dans notre système d'acquisition, six convertisseurs récupérant chacun huit voies. La question de la synchronisation de ces différents composants se pose alors. Les deux composants génèrent leur horloge d'acquisition de la même façon à partir de leur horloge (l'horloge de fonctionnement du système). La recommandation est de cadencer cette horloge à 2 MHz et est divisée par 1024 pour générer l'horloge d'acquisition. Cela donne une fréquence d'acquisition de 2 kHz environ (on remarque que l'horloge est en fait à 1,95 kHz et non à 2 kHz précisément). Nous ferons donc fonctionner tous les ADS avec la même horloge à 2 MHz. Cependant, lors du démarrage

de l'acquisition, il est spécifié que l'ADS1298 met 2308 microsecondes à commencer l'acquisition alors que l'ADS1299 ne met que 2052 microsecondes. Ce décalage aussi petit soit-il, peut être corrigé en retardant le signal de démarrage de l'ADS 1299 du delta temps indiqué dans les spécifications de ces deux composants ($2308 - 2052 = 256 microsecondes$). Synchroniser l'acquisition de ces composants permet en fait de pouvoir gérer ce parc de convertisseurs de façon uniforme, notamment la procédure de lecture des valeurs acquises par les composants peut être réalisés en une seule fois si l'on garantit qu'il n'y a aucun décalage de vitesse de fonctionnement des composants.

**L'acquisition du flux audio**

Pour l'acquisition audio, nous utilisons le composant TI TLV320AIC26 [TLV320AIC26]. Tout comme les composants ADS1298 et ADS1299, nous n'avons pas accès à l'horloge interne de ce composant. Le signal qui est donc utilisé comme déclencheur, pour contourner ce manque, est le signal *LRCLK*. Celui-ci détermine le canal audio qui est actuellement en train d'être lu du convertisseur (canal droit ou gauche). Le composant est configuré par un bus SPI. Les données sont récupérées par ce même bus.

Nous avons vu dans cette section, comment nous définissons la synchronisation de bas niveau. De cette définition, nous avons proposé un mécanisme enregistrant différents *timestamps* pour des évènements reçus. Le mécanisme proposé permet, à partir de dispositif pris sur l'étagère, de générer une information temporelle à travers des *timestamps*. Cette information permet de corréler les évènements d'une modalité avec les autres. Dans la section suivante, nous continuerons l'élaboration de notre mécanisme de synchronisation global, en expliquant comment utiliser les données générées par les mesures de bas niveau dans un flux de données haut niveau.

# 2 Synchronisation haut niveau

Notre mécanisme de haut niveau se repose sur l'utilisation d'un format de flux de données multimédia pour structurer les données acquises lors de l'examen. L'utilisation d'un fichier multimédia apporte beaucoup de fonctionnalités et de flexibilité à notre solution. Nous ne traiterons, dans cette section, que de l'apport de synchronisation que propose cette solution. Les autres aspects seront plus longuement abordés dans le chapitre VI.

## 2.1 Le flux multimédia

Afin d'intégrer le contenu d'un examen EEG, un nouveau format de fichier a été conçu à partir du format existant MKV. Nous commençons par décrire le format MKV, pour ensuite détailler notre nouveau format d'examen EEG.

### 2.1.1 Le standard MKV

Le format MKV est basé sur l'imbrication de balise. La hiérarchie de ces balises définit une structuration permettant à un lecteur de comprendre le contenu du fichier. La figure 24 représente la hiérarchie de base d'un flux MKV. Les deux balises trouvables dans le niveau 0 du flux sont l'entête ainsi que le segment. Le segment est le regroupement de toutes les données d'un flux multimédia. Un flux peut contenir plusieurs segments mais cela est fortement déconseillé puisque peu de lecteurs supportent cette fonctionnalité.

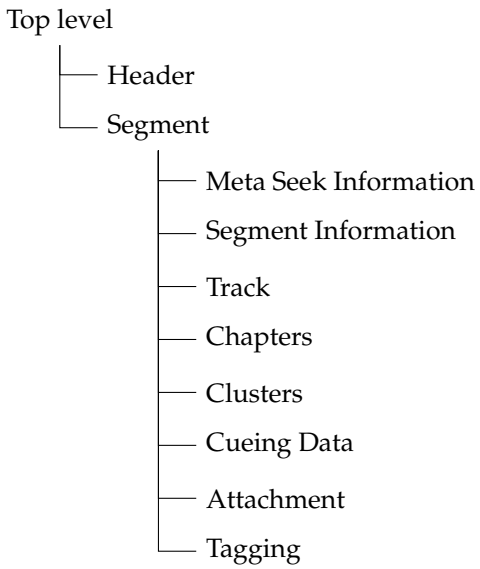



**Figure 24.** Structuration d'un flux MKV

Les informations trouvées dans l'entête du flux MKV sont les informations déclaratives permettant d'identifier le flux (Header). Nous trouvons des informations de version et de restriction de champs du flux. Nous retrouvons des balises similaires au DocType [1] trouvables dans un fichier XML. C'est dans cette balise que nous allons préciser que le flux est un flux mkv ainsi que la version du standard utilisée. La figure 25 montre un exemple d'entête pour un flux MKV.

La balise segment peut contenir 8 différentes balises dont l'ordre n'est pas obligatoirement celui exposé mais est conseillé :

1. Document Type

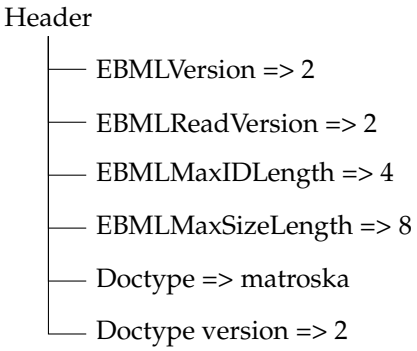



**Figure 25.** Exemple de Header d'un flux MKV

- **SegmentInfo** : contient des informations générales du segment dont le titre, la durée du média ainsi qu'un identifiant unique sur 128 bits. Cette balise contient également l'information d'échelle de temps utilisée dans ce segment (*TimecodeScale*). Il définit l'unité de mesure pour les timecodes utilisés dans ce segment. Un timecode, dans la terminologie de MKV, correspond à un *timestamp*. Cette unité s'exprime en nombre de nanosecondes. La valeur typique de ce champ est un million afin d'exprimer les timecodes en milliseconde (l'écart entre deux timecodes est un million de nanosecondes ou 1 milliseconde).
- **SeekHead** : est un index des éléments trouvés dans le niveau 1 du segment. Il permet de sauter vers une sous-section pour trouver plus rapidement d'autres informations du fichier. Il est recommandé de mettre cette balise avant toutes autres données.
- **Tracks** : contient la définition des pistes contenues dans ce segment. Chaque piste est déclarée par l'insertion d'une balise track. Dans cette dernière est contenue un identifiant, le type de piste, le codec utilisé et des informations propres à chaque type de piste.
  La figure 26 expose la hiérarchie d'une balise tracks. Celle-ci contient donc une première piste vidéo encodée avec H264. Les dimensions de la vidéo sont 1920 par 1080 à 30 images par seconde. Une seconde piste audio mono canal est déclarée. Elle est encodée en Vorbis à 22050Hz. Les types de données valides d'un flux MKV sont les suivants : vidéo, audio, complexe (vidéo et audio combinés), logo, sous-titres, bouton, contrôle.

- **Cues** : sont des informations utiles pour trouver un moment précis dans un fichier. Ces informations sont appelées des *Cuepoints* et contiennent un timecode et un pointeur sur un bloc de données. Dans une application de streaming, les cues ne peuvent pas être utilisés.
- **Cluster** : est la balise utilisée pour stocker la majorité des données du flux. Cette

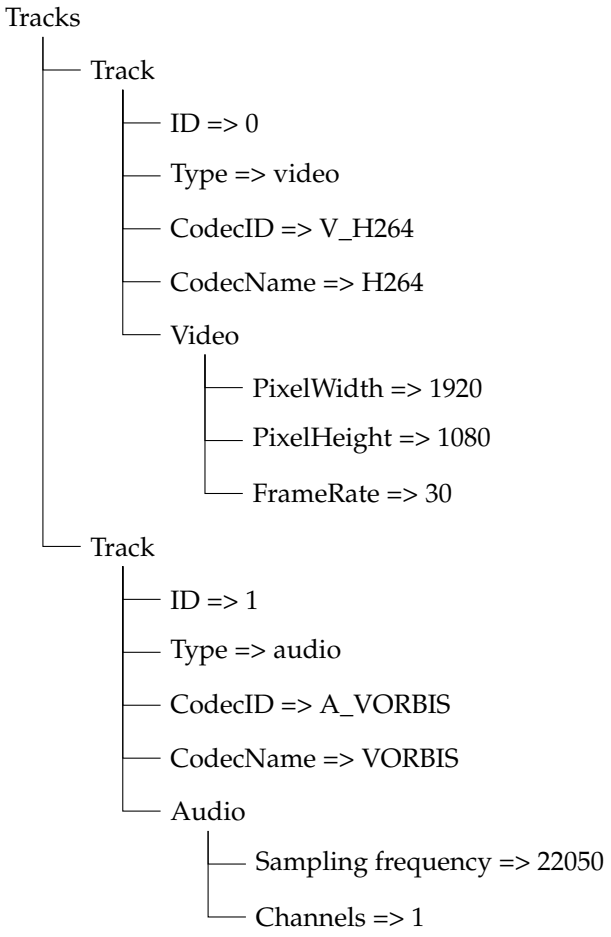



**Figure 26.** Exemple d'une balise Tracks d'un flux MKV

balise peut être (et est) répétée plusieurs fois dans un flux. Elle contient des fenêtres de temps de données. Aucune restriction n'est faite sur la durée ni la longueur d'un cluster mais les personnes implémentant le format semblent utiliser un maximum de cinq secondes ou de cinq Mégaoctets pour un cluster.
Les clusters contiennent un timecode propre et des blocs de données. Les blocs sont des structures binaires définies par la spécification de MKV. Dans ces blocs, nous trouvons le numéro de la piste dont les données font partie, un timecode propre au bloc de données et les données binaires. Un bloc contient généralement une trame de données compressées. Les blocs peuvent être soit des blocs simples contenant directement des données, soit placés dans une balise groupe de blocs permettant de rajouter des informations sur un bloc de référence. Les blocs simples sont préférés car leur empreinte mémoire est plus faible. Les groupes de blocs sont, en fait, utilisés pour les images B et P de codeur vidéo (H264, VP8).
Un exemple de contenu d'une balise *Cluster* trouvable dans la balise *Segment* est montré dans la figure 27. Ce fichier contient deux clusters, le premier placé temporellement au timecode 2300 contient quatre blocs simples et deux blocs

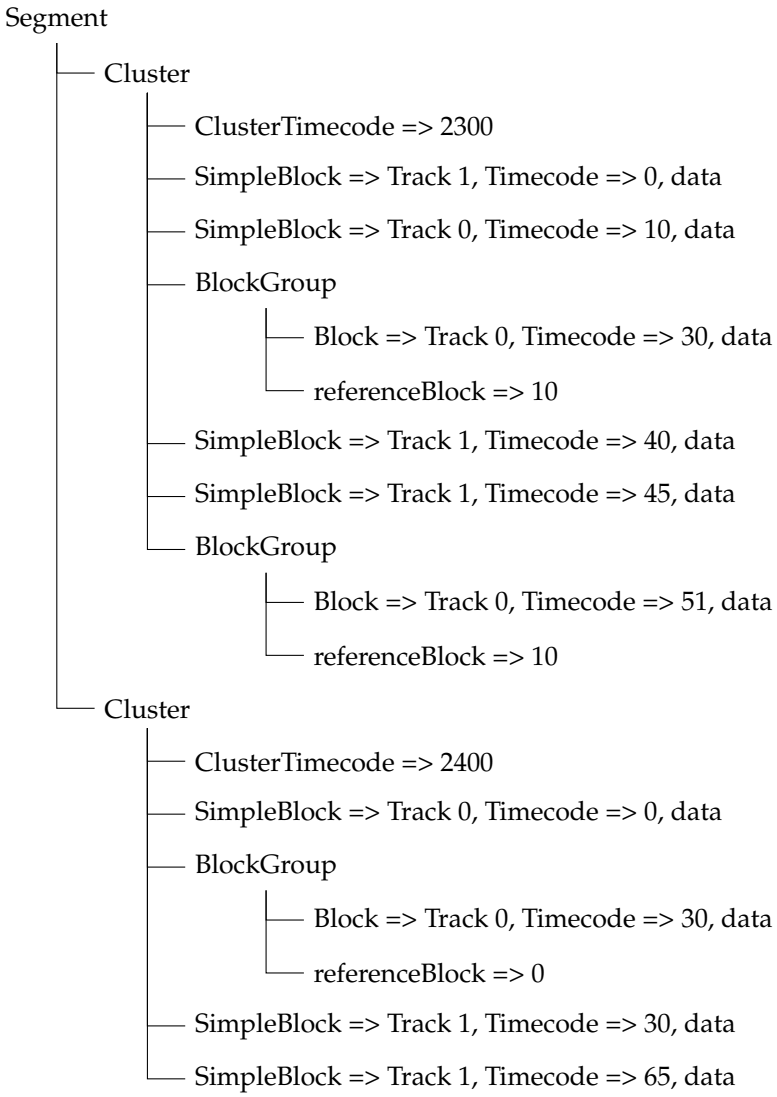



**Figure 27.** Exemple d'une balise Cluster d'un fichier MKV

complexes. Pour calculer le timecode d'un bloc, il faut utiliser la formule suivante :

$$timestamp = (timecode_{bloc} + timecode_{cluster}) * timecodescale \quad \text{(III.1)}$$

- **Chapters** : peut contenir des balises afin de créer des chapitres de ce segment. Un chapitre est simplement une portion du fichier ayant un début, une fin et un titre.
- **Attachments** : sont des données annexes au fichier multimédia. Cette balise peut par exemple contenir la couverture d'un DVD ou d'un CD.
- **Tags** : stocke des informations diverses facultatives pour la lecture. C'est une paire clé/valeur qui est attachée à une piste ou à un chapitre.

### 2.1.2 Le format BIO-MKV

Dans cette partie, nous allons détailler les spécificités du nouveau format proposé : BIO-MKV. Pour ce faire, il convient de définir les différentes modalités présentes dans

un examen EEG afin de savoir ce qui doit être stocké dans le fichier examen. Un examen EEG est constitué de signaux physiologiques (EEG, EMG, ECG, EOG), d'une vidéo, d'une piste audio, des annotations et des métadonnées du patient et de l'examen. Il nous faut pour chaque modalité de l'examen, savoir comment l'intégrer dans un flux BIO-MKV.

**La vidéo et l'audio**

Nous traitons, dans un premier temps, le cas de la vidéo et de l'audio qui nous semblent être les plus triviaux. Nous associons à notre piste vidéo une piste prévue à cet effet dans BIO-MKV. Nous utilisons toutes les infrastructures prévues à cet effet du standard. De même pour l'audio. De ce fait, il n'y a aucune différence entre le format MKV et le format BIO-MKV, si nous utilisons des codecs déclarés dans la spécification de MKV. Cependant, nous avons utilisé des codecs (notamment un codec vidéo) qui ne sont pas prévus par le standard MKV. Nous avons donc ajouté un identifiant textuel pour identifier ce nouveau codec. Cependant, la procédure est expliquée dans ce chapitre pour un codec physiologique et est identique pour un codec vidéo.

**Les signaux physiologiques**

D'un point de vue externe, nous pouvons remarquer de nombreuses similitudes entre les signaux physiologiques et les signaux audio. Les deux modalités sont des signaux multicanaux à une dimension. Nous pourrions alors imaginer, insérer les signaux physiologiques dans BIO-MKV comme étant des signaux audio. A priori, la manipulation est possible et fonctionnerait. Cependant, cela pose quand même un problème conceptuel vis-à-vis du détournement d'un type (on parlerait de *hack*). En effet, ce type n'a pas été prévu pour transporter ces données ce qui introduirait une ambiguïté sur la sémantique des paquets audio (un paquet audio est-il vraiment audio ou physiologique). Nous avons donc éliminé cette option.

De même, utiliser le type sous-titres, logo, boutons ou contrôle pour les signaux physiologiques ne nous semblait pas une bonne solution pour la même raison que pour le canal audio ainsi que la différence de forme que présentent les signaux physiologiques avec ces modalités.

La solution que nous proposons afin d'intégrer des signaux physiologiques dans un flux BIO-MKV est de rajouter un nouveau type de piste dans la balise *Tracks*. Nous avons donc rajouté le type physio qui transporte donc des données physiologiques. Nous donnons à ce type la valeur 4, cependant nous aurions pu utiliser n'importe quelle autre valeur laissée libre par le standard (0, 1, 2 et 3 étant déjà réservé).

Une fois ce nouveau type intégré, il faut définir comment formater les données physiologiques en trames pour le flux BIO-MKV. Pour répondre à cette question, nous nous sommes demandé quels étaient les problèmes liés aux formats EDF et EDF+ utilisés couramment dans le monde hospitalier. Le point bloquant à l'utilisation de ce

format que nous avons trouvé est le manque de synchronisation avec un flux vidéo. Ce point vient principalement du manque d'informations temporelles régulières dans le flux de signaux physiologiques, ce qui empêche de resynchroniser le flux d'échantillons physiologiques à une image de la vidéo. C'est le seul point bloquant son utilisation. Il reste donc à savoir comment intégrer régulièrement des points de resynchronisation entre la vidéo et les signaux physiologiques.

La solution la plus simple mais également la plus pratique, nous a paru d'utiliser le format de fichier EDF+ comment codec dans BIO-MKV et d'éparpillé les données de ce format dans des blocs de données du flux BIO-MKV. Il ne suffit pas de couper de façon aléatoire le fichier EDF+. Nous utilisons de manière commode le découpage naturel d'un fichier EDF+ en enregistrement de données. Nous plaçons dans chaque bloc de données MKV un enregistrement EDF+. Le timecode associé à ce bloc de données est le *timestamp* du premier échantillon de l'enregistrement (comme cela peut être fait pour un bloc de données audio). Les *timestamps* des autres échantillons du bloc (qui n'ont pas de timecodes) sont ensuite extrapolés soit en utilisant la fréquence d'échantillonnage d'acquisition, soit en utilisant la différence entre deux timecodes de blocs et le nombre d'échantillons entre les deux blocs. La deuxième méthode, plus complexe, devrait atteindre des résultats plus précis.

L'entête du fichier EDF+ est placé comme tout premier bloc de données du flux BIO-MKV. Une solution alternative pour inclure les métadonnées présentes dans l'entête du fichier EDF+ aurait été de rajouter des métadonnées propres à une piste physio comme il en existe pour les pistes vidéo et audio. Il y a deux raisons pour avoir placé ce bloc tel quel dans le fichier. La première est de simplifier l'implémentation de l'encodeur BIO-MKV. En effet, il est très facile d'encoder un flux BIO-MKV à partir des signaux d'un fichier EDF+, il suffit de le recopier. La deuxième raison est d'inclure dans sa totalité le fichier EDF+ dans le flux BIO-MKV. Cette propriété permet alors à tout lecteur d'extraire un fichier EDF+ brut sans aucun traitement et donc sans risque de perte de données.

Pour conclure, le format BIO-MKV intègre les signaux physiologiques dans un nouveau type de donnée physio en les formatant en EDF+ qui est désigné par un identifiant spécifique. Celui-ci est crucial car il permet à un lecteur d'identifier la piste physiologique. BIO-MKV contient le codec *D_EDF* dans sa liste de codecs supportés.

**Les annotations**

Nous avons examiné trois pistes pour l'intégration des annotations dans un flux BIO-MKV d'examen EEG.

La première piste est d'intégrer les annotations dans les enregistrements de données EDF+ précédemment discutés. La spécification d'EDF+ permet l'intégration d'annotations en réservant de l'espace dans chaque enregistrement. Cela impose d'une part une limite sur la longueur des annotations insérables, et d'autre part la réservation d'espace

inutilisé en cas d'absence d'annotation.

Une deuxième piste est d'utiliser l'infrastructure de chapitre disponible dans MKV. Un chapitre n'est rien d'autre qu'un titre avec un début et une fin. Cela est compatible avec une annotation. Grâce à ce constat, une solution est donc, pour chaque annotation, de créer un chapitre dans le flux BIO-MKV. Cependant, une contrainte imposée par la spécification du format MKV est gênante : l'obligation de placer un chapitre dans une balise Chapters. Selon les recommandations, cette balise est placée à la fin du flux après tous les clusters, c'est-à-dire après toutes les données. Or, il est impossible lors de la création du fichier de prédire ou va s'arrêter le fichier (et donc les clusters). Il est donc aussi impossible d'écrire un index dans la balise SeekHead de manière à trouver la balise de chapitres. Il faut donc lire l'intégralité du fichier afin de pouvoir récupérer les chapitres. Cette contrainte est en fait compréhensible car dans le cas d'un fichier créé à la volée (un flux), il n'y a pas de chapitre. Cependant, elle est limitante pour notre cas d'usage.

La dernière piste est d'utiliser le canal de sous-titres. Les annotations sont des informations textuelles ayant lieu à des moments précis. De par cette définition, il y a une forte ressemblance entre ces annotations et les sous-titres d'une vidéo. Nous avons donc utilisé une piste de sous-titre pour encapsuler les annotations. Chaque annotation est donc encodée comme un texte de sous-titre et est associée à un timecode de début d'affichage et de fin d'affichage.

C'est cette dernière approche que nous avons choisie pour intégrer les annotations. Cette intégration présente plusieurs avantages :

- le premier est la dissociation des annotations des autres modalités. Cela apporte une facilité d'édition des annotations dans le flux BIO-MKV. En effet, supprimer une annotation revient à supprimer un bloc du flux BIO-MKV. Ajouter une annotation revient également à ajouter un bloc dans le flux. Nous verrons dans le chapitre V que cette facilité d'édition nous sera utile pour l'intégration de notre système finale.
- Le deuxième est l'accès direct aux annotations. Il n'y a pas besoin de décompresser un format pour accéder aux annotations. Cet avantage est vrai dans le cas où les annotations sont liées au format de stockage des signaux physiologiques comme elles pourraient l'être dans EDF+.
- Le troisième avantage est la possibilité de lire les annotations sur un lecteur MKV standard sans support du format BIO-MKV.

Un désavantage de cette méthode est de devoir lire l'intégralité du fichier pour lister les annotations. Nous nous permettons d'avoir ce désavantage car à la visualisation, les annotations pertinentes et affichées sont celles qui sont temporellement proches des données visualisées couramment. Celles-ci seront proches dans le flux BIO-MKV.

**Les métadonnées**

Les dernières informations à conserver pour un examen EEG sont les métadonnées du patient et de l'acquisition. Deux approches sont disponibles pour répondre à ce problème. La première est d'intégrer les métadonnées directement dans le flux BIO-MKV où chaque champ de métadonnées serait stocké dans une balise tag. La deuxième approche envisagée est d'utiliser ou de créer un fichier ou protocole enveloppe dans lequel un fichier BIO-MKV serait inclus. Nous allons faire une comparaison de ces deux approches.

Dans l'approche où BIO-MKV contient les métadonnées :

- **Avantages :** Les données sont unifiées dans un seul fichier qui contient toutes les données de l'examen. De ce fait, il n'y a qu'un seul format à implémenter pour un lecteur. De plus, les transactions entre organismes sont plus simples car seulement un fichier doit être envoyé.
- **Désavantages :** Le désavantage majeur est de devoir utiliser le format BIO-MKV. Ceci casse la compatibilité avec les examens créés dans d'autres formats. De ce fait, pour importer des examens dans cette solution de télémédecine, il faudrait réencoder les examens en BIO-MKV depuis EDF+.

Dans l'approche où l'on rajoute une enveloppe autour de BIO-MKV :

- **Avantages :** L'avantage certain de cette solution est que le fichier BIO-MKV devient une métadonnée de l'enveloppe utilisée. Nous pouvons donc, pour des examens enregistrés dans d'autres formats simplement remplacer le fichier BIO-MKV par cet autre format de fichier tout en gardant la même interface pour les métadonnées. De plus, les SIH utilisent déjà des enveloppes comme HL7 [HL7] pour dialoguer. Il suffit donc d'ajouter les étiquettes utiles à un examen EEG pour l'utiliser. Il faut aussi noter que dans tous les cas, il y aura une transaction HL7, avec ces métadonnées, pour dialoguer entre deux SIH quel que soit le placement des métadonnées dans le flux final.
- **Désavantages :** Le point négatif de cette approche est très certainement de devoir gérer deux flux aux formats différents. Ce désavantage est seulement valide si le SIH n'implémente pas déjà cette enveloppe ce qui est rarement le cas.

De par cette comparaison, nous avons décidé que la meilleure approche était d'exporter les métadonnées de l'examen dans une enveloppe externe à BIO-MKV. L'enveloppe qui a été préférée par les membres du projet Smart-EEG est HL7.

**Résumé de la solution choisie**

Nous allons donc résumer les différents choix faits pour formater un examen EEG. La figure 28 expose la hiérarchie d'encapsulation de l'examen. L'enveloppe générale est donc une enveloppe HL7 contenant les métadonnées du patient, de l'examen et de l'établissement de santé. Dans une des métadonnées de cette enveloppe est stockée un fichier BIO-MKV contenant les différentes données de l'examen. Quatre pistes sont

créées dans ce conteneur. Une première piste transporte le flux vidéo et une deuxième pour transporter le flux audio. La troisième piste est dédiée au transport des données physiologiques. Celles-ci sont stockées dans le format EDF+ mais elles sont éclatées par enregistrement dans différents blocs du flux BIO-MKV. Pour terminer, les annotations sont stockées dans une dernière piste de sous-titres.

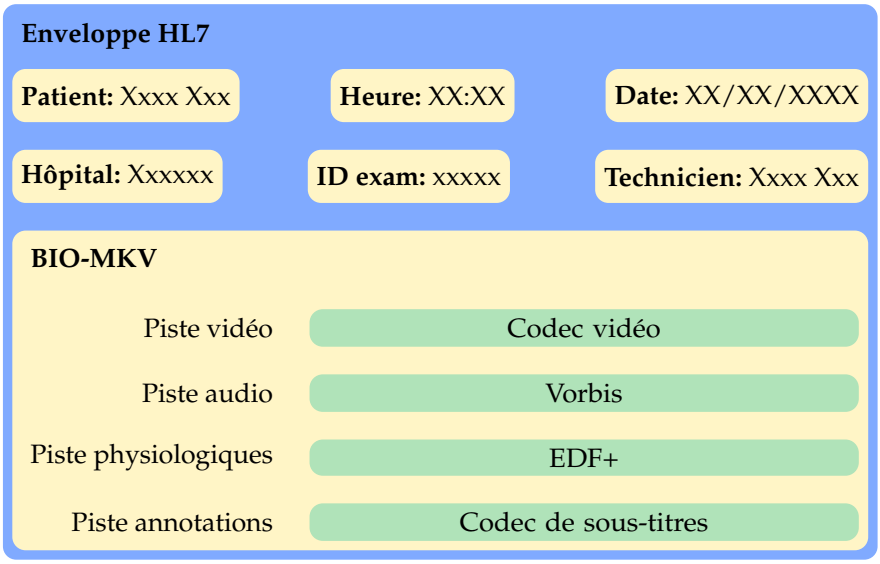



**Figure 28.** Représentation de la hiérarchie d'encapsulation d'un examen EEG

Nous précisons deux choses par rapport au choix d'encapsulation. Premièrement, le choix du codec vidéo utilisé sera détaillé dans le prochain chapitre. Deuxièmement, il faut noter que sur la base de nos spécifications et solutions proposées, les partenaires du projet Smart-EEG ont implémenté l'insertion des annotations et des métadonnées. Il est bon de remarquer que la flexibilité du format BIO-MKV nous permet de changer de codec pour chaque piste sans impacter le reste du format.

Dans la prochaine sous-section, nous détaillons la manière dont fonctionne la bibliothèque FFmpeg et expliquons comment le nouveau format est implémenté dans celle-ci.

## 2.2 La bibliothèque FFmpeg

Avant même de choisir le format MKV comme base de travail, nous avons voulu comparer des implémentations réelles, afin de mieux appréhender les différents formats de fichier disponibles. Pour faire cette comparaison, nous avons donc utilisé et étudié la bibliothèque FFmpeg [FFmpeg], contenant l'implémentation d'un nombre non négligeable de standards multimédias. Initialement lancée en 2000 avec l'implémentation des standards MPEG, elle a évolué et n'est actuellement plus restreinte au format

MPEG. Elle est utilisée comme base de bon nombre de logiciels libres ou propriétaires (VLC, Firefox, Google Chrome, Youtube). De par le fait qu'elle contienne beaucoup de formats, elle a permis d'évaluer et d'essayer les différentes options disponibles pour notre solution. Le projet est divisé en huit bibliothèques et quatre exécutables :

- **bibliothèques :**
  - **libavcodec :** bibliothèque contenant l'implémentation des codecs supportés par FFmpeg ou les bindings [1] permettant d'appeler d'autres bibliothèques ;
  - **libavformat :** bibliothèque contenant l'implémentation des muxeurs et demuxeurs [2] que supportent FFmpeg ;
  - **libavdevice :** bibliothèque contenant des bindings vers des sources d'acquisition de média (récupérer un flux d'une webcam, d'un microphone, etc.) ;
  - **libavutil :** bibliothèque d'outil permettant la portabilité de code. Elle contient entre autres des fonctions mathématiques, de chiffrement, des générateurs de nombres aléatoires, etc ;
  - **libswscale :** bibliothèque hautement optimisée destinée à réaliser des conversions de format vidéo (des conversions d'espace de couleur, de redimensionnement de taille d'image et de représentation des pixels) ;
  - **libswresample :** bibliothèque hautement optimisée destinée à réaliser des conversions de formats audio (rééchantillonage, changement du nombre de canaux, conversion de format) ;
  - **libavfilter :** bibliothèque contenant des filtres génériques audio et vidéo ;
- **logiciels :**
  - **ffmpeg :** outil pour convertir des fichiers vidéo et audio d'un format vers un autre ;
  - **ffplay :** lecteur élémentaire vidéo utilisant la bibliothèque SDL pour l'affichage ;
  - **ffserver :** serveur multimédia HTTP et RTSP permettant de fait des diffusions sur internet ;
  - **ffprobe :** outil affichant des informations sur un flux ou un fichier ;

FFmpeg propose de nombreuses fonctionnalités, mais nous n'aurons qu'à adapter que celles de compression et de multiplexage. Pour ce faire, nous nous intéressons principalement aux deux objets définis par FFmpeg : les codecs et les formats. Ces deux objets sont en fait, deux interfaces permettant d'uniformiser la compression et le multiplexage de données multimédia. Dans le but d'uniformiser ces traitements,

1. Un binding est une liaison entre deux bibliothèques. Dans le cas de FFmpeg les bindings font la conversion entre le format d'une bibliothèque tierce et le format de la bibliothèque FFmpeg. Ils permettent d"utiliser des bibliothèques tierces de façon transparente dans pour l'utilisateur de FFmpeg.

2. muxeur et demuxeur est la terminologie utilisée par FFmpeg pour parler des codes sources encapsulant des flux multimédias dans un conteneur et décapsulant des flux multimédias depuis un conteneur

FFmpeg utilise trois objets abstraits permettant de manipuler les données d'entrée des codecs, les données de sortie des codecs et les données de sortie des formats. La figure 29 clarifie ces deux étapes utilisées pour créer ou lire un fichier multimédia. Dans un premier temps, les données brutes sont encapsulées dans une trame. Cette trame est lue par un des codecs de FFmpeg afin de créer un paquet. Ce paquet est ensuite transmis à un des formats afin de générer le flux dans une sortie abstraite. Lors du décodage, les opérations inverses sont réalisées.

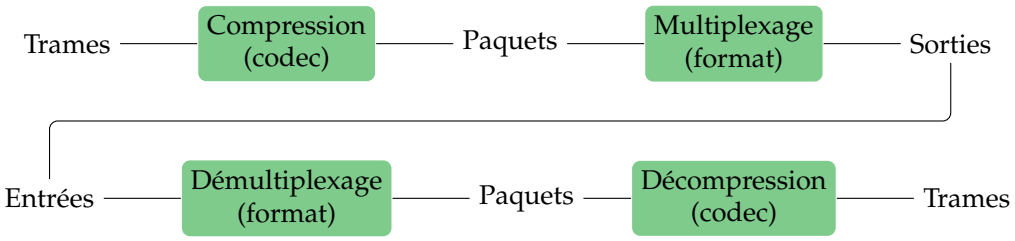



**Figure 29.** Étape de compression et d'encapsulation d'un flux multimédia

## 2.3 l'API de création de fichiers d'examen EEG

Dans cette sous-section, nous parlons du fonctionnement interne de la librairie FFmpeg lors de la génération d'un fichier d'examen (ou d'un flux réseau). Pour ce faire, nous parcourrons les différentes structures proposées par FFmpeg pour faire ce traitement. La figure 30 schématise l'organisation des structures utiles à la création d'un fichier BIO-MKV. Nous ne rentrons pas dans tous les détails de ces structures dans cette sous-section. Davantage d'informations peuvent être lues dans l'annexe A. Aussi, dans ces explications, certains noms de variable ont été changés par rapport à leur implémentation dans FFmpeg, de manière à être plus parlant. Par exemple, le nom de variable *codec* est utilisé dans deux structures différentes pour faire référence à deux objets distincts. Nous en avons renommé un des deux pour écarter l'ambiguïté.

Dans le processus de création d'un fichier, le nœud principal est l'objet *AVFormatContext* permettant de créer un contexte de multiplexage. Ce contexte, disponible à l'utilisateur, contient toutes les informations relatives à la création d'un fichier multimédia. Nous y trouvons plusieurs références vers des objets gérant les codecs, le format du flux et la sortie du flux. La première référence *format* pointe vers la variable de type *AVOutputFormat* gérant la construction d'un flux dans un format précis. Dans notre cas, *format* pointera vers notre implémentation de BIO-MKV. La deuxième référence *io* pointe vers une variable *AVIOContext* permettant de gérer les différentes formes de sortie que supporte FFmpeg. C'est à travers cet objet que les écritures vers un fichier ou que les envois de paquet sont effectués. Nous utilisons soit une sortie vers un fichier soit

une sortie réseau utilisant le protocole TCP/IP. En pratique, pour former un paquet, une fonction spécifique au format de la structure *AVOutputFormat* est appelée. Dans cette fonction, des appels aux méthodes de la structure *AVIOContext* sont faits pour écrire le flux dans sa forme finale. Les dernières références définies par la variable *streams* est un tableau de flux. Ces flux, tous décrits par une variable *AVStream*, gèrent les données propres à une piste du format BIO-MKV comme l'avancement de l'encodage, le temps courant et également un contexte de compression.

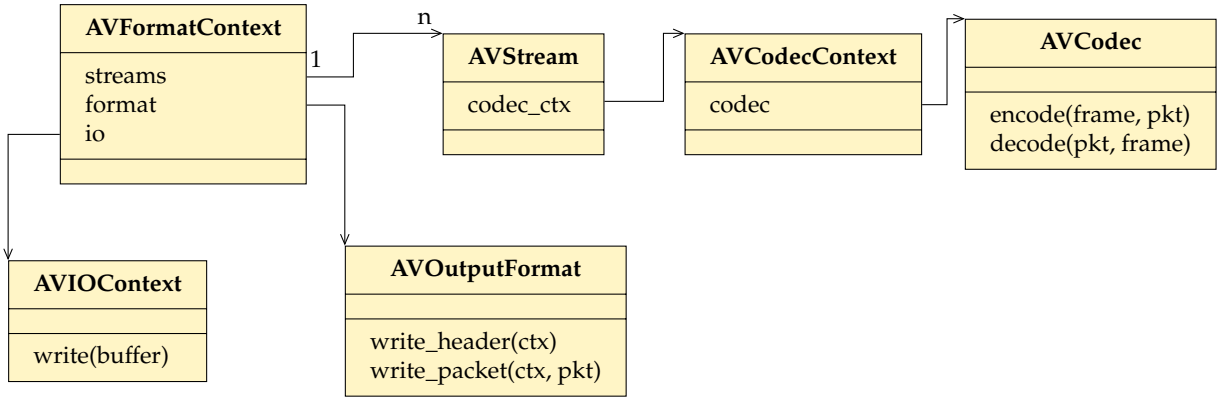



**Figure 30.** Diagramme UML des objets utiles à la création d'un fichier BIO-MKV

Le contexte de compression, matérialisé par la structure *AVCodecContext*, stocke toutes les informations relatives au codec utilisé pour la piste décrite. Elle contient entre autres le type de données traité. Dans le cas des modalités audio, vidéo et annotations, ce type est celui défini par FFmpeg comme étant *AVMEDIA_TYPE_AUDIO*, *AVMEDIA_TYPE_VIDEO* et *AVMEDIA_TYPE_SUBTITLE*. Par contre, FFmpeg ne supporte pas nativement les données physiologiques. Nous avons donc dû les intégrer dans un type de données de FFmpeg. Nous avons choisi d'utiliser le type *AVMEDIA_TYPE_DATA* disponible par défaut dans la librairie. C'est un type abstrait pour FFmpeg. De ce fait, à aucun moment dans le code de la librairie, il n'y a d'hypothèse de faite sur le contenu des paquets de ce type. Nous aurions pu intégrer un nouveau type physiologique mais cela aurait impacté beaucoup de condition et d'hypothèse que fait la librairie et aurait requis un effort de développement peu justifié. La structure *AVCodecContext* contient également l'identifiant du codec utilisé. Dans le cas des signaux physiologiques, nous avons alloué un nouvel identifiant décrivant le codec EDF+. Cette structure est présente même si FFmpeg ne gère pas la compression de la piste décrite. Dans ce cas, c'est à la charge du développeur de générer les paquets pour les écrire dans le format. Nous avons utilisé cette méthode pour l'encapsulation des signaux physiologiques en nous chargeant nous-mêmes de la création des paquets pour FFmpeg. Pour toutes les autres modalités, nous sommes passé par les implémentations de FFmpeg pour la compression. Dans ces derniers cas, la structure *AVCodecContext*

contient le champ *codec* qui pointe vers l'implémentation du codec contenu dans une structure *AVCodec*.

Dans le cas de la génération d'un fichier d'un examen EEG BIO-MKV comme décrit dans la sous-section 2.1.2, nous aurons un objet *AVFormatContext* pointant vers notre implémentation de BIO-MKV et vers une sortie fichier. Dans ce contexte seront contenues quatre pistes (signaux physiologiques, vidéo, audio et annotations). Cela implique donc que nous aurons 4 objets de type *AVStream*, chacun décrivant une piste du flux BIO-MKV. Les trois pistes vidéo, audio et sous-titre (pour les annotations) sont identifiées comme des flux et utilise les implémentations de FFmpeg pour la compression des données. La dernière piste contenant les signaux physiologiques est simplement déclarée comme un flux de type *AVMEDIA_TYPE_DATA* et encodée avec le codec EDF+ mais dont FFmpeg n'a pas d'implémentation.

Une fois cette phase d'initialisation des structures finie, l'utilisateur peut demander l'écriture de l'entête du fichier BIO-MKV par l'appel de fonction *avformat_write_header*. Des vérifications sont alors faites sur les pistes présentes dans le contexte *AVFormatContext*, pour savoir si l'implémentation du format en question les supporte. L'écriture du fichier commence par les balises d'entête et de DocType puis par l'unique segment de ce fichier. En effet, notre implémentation ne contient qu'un seul segment contenant l'intégralité de l'examen. Les informations du segment sont placées au début de celui-ci de manière à pouvoir interpréter directement le reste du segment. Elles sont donc écrites à ce moment en indiquant que l'échelle des *timestamps* est une nanoseconde, que l'heure de création du fichier est l'instant courant et que l'application écrivant le fichier est l'encodeur Smart-EEG. Ensuite, les différentes pistes du format sont déclarées dans la balise *tracks*. C'est à ce moment-là que le lien est fait entre la déclaration d'un flux dans la structure *AVFormatContext* et la déclaration de la piste dans le fichier BIO-MKV. Pour chaque flux, une nouvelle balise track est créée. Elles sont numérotées de 1 à $n$ ($n$ étant le nombre de pistes du fichier). Ce numéro correspond à l'identifiant de la piste qui est écrit dans le fichier. Le nom du codec est récupéré dans la structure *AVCodecContext* et permet au décodeur de connaître le codec utilisé pour la piste. L'écriture du type de flux pour la vidéo, l'audio, les annotations et les données physiologiques est respectivement une piste vidéo avec le type 1, une piste audio avec le type 2, un piste sous-titre avec le type 10 et pour finir une physio avec le type 17. Les informations propres à chaque modalité sont ensuite écrites (taille de la vidéo, fréquence audio, etc.). La balise track de la piste en question est ensuite fermée pour passer à la suivante. Quand toutes les pistes sont écrites dans l'entête, la balise tracks est fermée et le contenu de l'examen peut commencer à être écrit.

L'utilisateur peut à présent faire des appels à *av_interleaved_write_frame* qui est une des fonctions permettant d'écrire un paquet dans le format de sortie. Même si son nom est trompeur, la fonction prend en argument le paquet à écrire dans le fichier. La fonction prend également en argument le contexte du format contenant toutes les

informations du multiplexage. Son premier travail est de déterminer si le paquet à écrire peut être placé dans le cluster précédemment ouvert (par un ancien appel à la même fonction). Pour ce faire, elle calcule la taille et le temps accumulé du cluster courant. Si le précédent cluster dépasse 5 s ou 5 MO, alors un nouveau cluster est créé dans le fichier. Dans ce cas, elle enregistre la taille du fichier à ce moment-là pour pouvoir mesurer la taille du cluster dans les prochains appels de cette fonction et procède à la fermeture du cluster précédent et à l'écriture d'une nouvelle balise cluster dans laquelle elle écrit le *timestamp* du nouveau cluster. L'écriture du bloc en lui-même est ensuite effectuée en commençant par l'entête du bloc qui précise la taille du bloc, son identifiant de piste, son *timestamp* et si le bloc contient une trame de référence. Les données compressées du paquet sont ensuite écrites dans le fichier. Dans le cas où le bloc n'est pas un simple bloc, l'identifiant du bloc de référence est ajouté après les données. La balise du bloc peut alors être fermée. Nous pouvons remarquer que dans l'étape d'écriture du paquet, il n'y a pas de différence de traitement dû à la nature du paquet, tous les paquets sont traités de la même manière en utilisant les informations présentes dans la structure *AVCodecContext*.

Lorsque tous les paquets ont été écrits, l'utilisateur appelle la fonction *av_write_trailer* pour clôturer le fichier. Les derniers paquets qui auraient été placés en tampon sont alors écrits. Dans le cas d'un fichier, les tags , les chapitres ainsi que les pistes sont écrits et les informations présentes au début du fichier (la durée, la taille du fichier ainsi le placement des dernières balises tracks, chapters et tags) sont mises à jour. La balise de segment global ainsi que le fichier sont fermés.

Grâce à cette interface, nous pouvons écrire un programme C qui va produire un fichier examen BIO-MKV. Ce programme C réalisera des appels vers les fonctions contenues dans la librairie FFmpeg. C'est dans la librairie que la création du bitstream du fichier est implémentée. Il est à noter qu'en remplaçant la sortie du flux par une sortie de réseau TCP/IP, le fichier est envoyé à l'adresse précisée.

## 2.4 Utilisation des utilitaires pour créer un fichier Examen.

Nous venons de voir comment utiliser la librairie à travers son API pour générer un fichier examen. Cette API nous sera utile lors de l'intégration système. Cependant, pour des raisons de tests, nous avons eu le besoin de générer des fichiers examens avant que l'intégration système soit réalisée. Cette sous-section explique comment nous avons construit des fichiers examens à partir des utilitaires de la suite FFmpeg. Ce qu'il faut savoir, c'est que les utilitaires de FFmpeg utilisent l'interface précédemment décrite pour faire toutes les manipulations sur les fichiers.

Pour le cas des données vidéo, des données audio et des données d'annotations nous utilisons directement les infrastructures disponibles de FFmpeg pour créer le fichier BIO-MKV. De ce fait, les logiciels de la suite (ffmpeg, ffprobe, ffplay) sont capables

d'éditer ces fichiers qui respectent le format BIO-MKV. Pour ce faire, il suffit, par exemple, à partir d'une séquence d'images (imagesXXX.jpg), d'un flux audio (alsa [1]) et d'un fichier de sous-titres (annotations.srt), d'invoquer le programme ffmpeg afin de créer un fichier (out.bio) au format voulu :

```
$ ffmpeg -i images%d.jpg -f alsa -i hw:1 -i annotations.srt out.bio
```

L'invocation de cette commande génère alors le fichier BIO-MKV regroupant les données vidéo, audio et sous-titres des trois sources. Il est possible de relire ce fichier grâce au lecteur ffplay en utilisant la commande suivante :

```
$ ffplay out.bio
```

Pour toutes les modalités, l'élément de FFmpeg qui génère des données d'entrée a être encodées est appelé un périphérique. Un périphérique n'est rien d'autre qu'un générateur de paquet. Ces paquets sont généralement récupérés de périphérique comme une webcam ou un micro mais peuvent être générés depuis des périphériques virtuels à des fins de tests ou bien depuis des fichiers. Par défaut, il n'existe pas de périphérique gérant des paquets de données physiologiques, nous avons donc implémenté un périphérique physiologique pour générer des paquets EDF+. Ce périphérique ne fait rien d'autre que de lire un fichier EDF+ et de générer des paquets pour FFmpeg depuis ce fichier. Grâce à ce simple périphérique, il est possible, en rajoutant le périphérique physiologique comme source de données à la commande ffmpeg, de multiplexer dans un fichier BIO-MKV le contenu d'un fichier EDF+. Le périphérique physiologique développé prend en paramètre le nom du fichier EDF+ à encapsuler. Comme ce périphérique génère directement des paquets prêts à être multiplexés (qui ne nécessite pas d'encodage), il est nécessaire de dire à FFmpeg de simplement copier les données physiologiques (argument `c:d copy` (Codec :Data copy)).

```
$ ffmpeg -i images%d.jpg -f alsa -i hw:1 -i annotations.srt -f edf -i eeg.edf
c:d copy out.bio
```

Afin de valider le bon fonctionnement du multiplexeur, nous pouvons utiliser le programme ffmpeg, avec en entrée un fichier BIO-MKV produit par nos soins, pour extraire et recombiner le fichier EDF+ d'origine. Ce dernier fichier EDF+ et le fichier d'entrée EDF+ initial doivent être similaires. L'extraction est réalisée avec la commande suivante :

```
$ ffmpeg -i toto.webm -c:d copy -f data dump/data.edf
```

Cette procédure peut être réalisée pour toutes les modalités pour savoir s'il est possible de récupérer les données multiplexées. Ceci nous permet de valider que le formatage est bien réalisé et qu'aucune donnée n'est perdue par celui-ci. Nous évaluons ce format en détail dans le chapitre VI.

1. The Advanced Linux Sound Architecture [ALSA] donne des capacités audio et MIDI à Linux. Il permet de lire des données flux audio de périphérique connecté à l'ordinateur.

# 3 Conclusion

Pour conclure, ce chapitre porte sur les mécanismes de synchronisation élaborés au cours de cette thèse. Nous avons séparé ces mécanismes en deux sous-parties : synchronisation bas niveau et haut niveau. La première porte sur les mécanismes matériels de synchronisation permettant d'apporter des garanties temps réel à l'enregistrement de l'information temporelle des acquisitions. La deuxième partie explique les mécanismes logiciels permettant de stocker toute l'information acquise dans la première partie dans un flux multimédia prêt à être envoyé vers un serveur pour stockage et relecture. Ces deux niveaux de synchronisation fonctionnent conjointement pour réaliser une synchronisation au niveau système. Nous présentons nos évaluations de la synchronisation dans le chapitre VI.

Chapitre IV

# Algorithme de compression vidéo



Le format BIO-MKV détaillé lors du précédent chapitre encapsule plusieurs composantes : vidéo, audio, annotations et signaux ExG. Dans ce chapitre, nous nous intéressons à la composante vidéo d'un examen EEG. Nous décrivons dans un premier temps des spécificités de la séquence vidéo pour l'examen EEG. Nous détaillons, en suite, le codec d'image adapté pour compresser la vidéo. Nous expliquons chaque bloc du codec un par un dans le sens de compression. Il est à préciser que ce codec contient beaucoup de paramètres que nous évaluerons dans le chapitre VI. Cependant, de manière à constituer des valeurs par défaut pour ces paramètres, nous proposons des valeurs que nous avons définies empiriquement avant les évaluations. Ces valeurs seront, après évaluation, définies pour avoir le codeur final.

## 1 Vidéo d'un examen EEG

Comme établi dans le chapitre II, l'acquisition de la vidéo d'un examen EEG doit répondre à plusieurs critères permettant l'acquisition de phénomène pouvant expliquer l'activité physiologique du patient. La première analyse que nous pouvons faire de l'acquisition vidéo est de connaître au maximum les propriétés de la scène. La salle est bien éclairée et le patient est de préférence placé au centre du cadre. La scène en

question est une prise de vue fixe. De ce faite, il est possible de distinguer le patient du fond fixe de la scène. Le patient qui est un sujet pouvant avoir des mouvements brusques à cause de pathologie est au centre de l'attention pour analyser les signaux physiologiques. Cependant, les zones de fond pour certains évènements comme le claquage d'une porte peuvent être utilisés pour comprendre ces derniers. Il est donc important d'avoir à la relecture une bonne visibilité sur certaines zones du patient (comme les mains et la tête) mais nous ne pouvons pas négliger le fond de la scène en le supprimant. Cette classification en deux zones est suffisamment importante pour que nous l'exploitions dans le développement de notre solution.

D'un point de vue technique, le premier critère est la définition des images. Il fait intervenir deux paramètres, le premier est le nombre de pixels des images et le deuxième est le rapport hauteur par largeur des images que l'on appelle aussi le format. Le nombre de pixels doit être suffisamment grand pour pouvoir identifier un patient et discerner ses yeux et les doigts de ses mains. Le standard de l'industrie à ce jour est la définition Full-HD dont le format est 16/9 et la définition 1920 pixels par 1080 pixels. Le deuxième critère est la présence de couleur dans l'image. Cette information permet au praticien de visualiser un changement de teinte de la peau du patient et pouvoir la corréler avec son activité physiologique. Le troisième critère, plus novateur, est la cadence d'image. En effet, de manière à visualiser des pathologies comme les myoclonies, un praticien a besoin de voir un des mouvements rapides de doigt ou d'œil durant seulement vingt millisecondes. Nous devons alors définir l'intervalle minimum d'acquisition entre deux images pour être sûr d'acquérir ces mouvements. Pour ce faire, nous avons utilisé le théorème de Shannon [Shannon, 1949], pour déterminer qu'il nous fallait acquérir deux fois plus d'images qu'il ne peut y avoir de secousses. De ce fait, il peut y avoir au maximum 50 secousses en une seconde. Nous en déduisons qu'il nous faut 100 images en une seconde. Nous devons donc acquérir une image toutes les dix millisecondes. Cela nous contraint donc à faire une acquisition à cent images par seconde.

En conséquence, l'acquisition de la séquence vidéo doit donc respecter au minimum les caractéristiques suivantes : une vidéo Full-HD couleur à cent images à la seconde. Un problème conséquent émane de ce flux de données, la quantité de données à traiter est très importante. Un pixel couleur est stocké sur 3 octets (un par composantes). Donc une image brute entière de ce flux équivaut à $1920 * 1080 * 3 \approx 6,2\ Mo$, ce qui, une fois ramené par seconde, nous donne $6,2 * 100 \approx 620\ Mo/s$. Si nous comparons ce débit de données au débit produit par les signaux physiologiques ($40_{Canaux} * 2000_{Échantillons} * 3_{Octets} \approx 240\ ko/s$), nous nous rendons compte qu'il est 2500 fois plus important. Le même constat peut être fait pour le son qui génère seulement $44000_{Hz} * 3_{Octets} \approx 132\ ko/s$. Il y a donc un intérêt bien particulier et critique à réduire le flot de données de la modalité vidéo comparé aux autres modalités.

## 2 Algorithme ROI-Waaves

De par la spécificité de la séquence vidéo et du cadre médico-légal de l'examen, nous proposons une nouvelle chaîne de compression, appelée ROI-Waaves, basée sur des concepts du seul encodeur d'image certifié dispositif médical Waaves. Nous faisons l'hypothèse que l'approche d'utiliser des mécanismes précédemment certifiés facilitera une éventuelle homologation dans le cadre médical de notre chaîne de compression vidéo. La conception de ce compresseur a été réalisée pour répondre à trois contraintes distinctes. La première est d'être capable de réduire le volume de données de manière significative pour arriver à une quantité gérable par les systèmes d'information d'aujourd'hui. Pour rappel, un examen de routine dure entre 20 et 30 minutes ce qui pour un débit de 620 $Mo/s$ produit des examens de 744 $Go$ à $1,1$ $To$. En 2010, près de 8000 actes d'EEG ont été réalisés ce qui représente alors plus de 6000 $To$ de données par ans. La deuxième est d'avoir une perte de qualité entre la compression et la décompression la plus petite possible sur les zones d'intérêt de la séquence. La dernière, et pas la moins importante, est d'assurer une complexité d'encodage et de décodage réduite par rapport au codeur de base. La réduction de la complexité permet d'envisager une implémentation capable d'atteindre la compression en temps réel de la séquence vidéo. Il est à rappeler que l'objectif final est d'avoir un système de télémédecine embarqué dans une carte. Il est donc nécessaire de réduire au maximum le besoin de calcul et de stockage du système. Le nouveau codeur vise à garantir que la vidéo d'un examen EEG soit compressée à un faible débit de sortie tout en conservant une qualité de restitution dans des zones d'intérêt et en maintenant un codeur peu complexe pour atteindre des vitesses de compression proche de cent images par seconde. Il est important de pouvoir garantir une compression en temps réel pour différentes raisons. Premièrement, si l'examen est envoyé directement au centre relecteur, cela permet de le lire au fur et à mesure de l'acte. Deuxièmement, si la compression est réalisée en temps réel, le stockage temporaire des images brutes en attente d'être compressées n'a pas lieu d'être. Ceci simplifie donc la partie en amont de la compression. Dernièrement, cela implique que l'examen est disponible dès que le technicien arrête l'acquisition.

Une de nos solutions proposées pour atteindre ces objectifs est d'intégrer au traitement vidéo une importance à des zones d'intérêt (ROI). Ces zones sont définies sur les portions de l'image qui nécessite une qualité maximale à cent images par seconde. Ces zones d'intérêt sont des entrées du codec, en plus de la séquence vidéo à encoder. La manière dont sont définis les ROI n'est pas importante pour l'algorithme de compression. Les zones sont définies dans l'espace pixel de la vidéo et classifient essentiellement les pixels en deux groupes : les pixels d'intérêt et les pixels de fond. Cette classification est informée au codeur sous la forme d'une carte binaire de la taille

de l'image. Quand la valeur de la carte à la position d'un pixel vaut 0, le pixel est un pixel de fond et quand la valeur vaut 1, le pixel est un pixel d'intérêt. L'information importante à tirer de cette classification est que seuls les pixels de la classe d'intérêt doivent être à une qualité maximale et être rafraîchis cent fois par secondes. Nous exploitons cette information dans plusieurs étages du codec.

La chaîne de compression ROI-Waaves est un pipeline composé de six étages : le changement d'espace colorimétrique, la différence temporelle, la décomposition en ondelette, la quantification, l'extraction des plans de bits et pour finir le codeur entropique HENUC. Ces étapes sont explicitées dans le schéma bloc de la figure 31. Dans le reste de cette section, nous essayons d'expliquer le rôle et le fonctionnement de chaque bloc du codeur chacun dans une sous-section. Le codec ROI-Waaves est un compresseur relativement symétrique, c'est-à-dire que son décodeur réalise les opérations inverses du codeur. De ce fait, nous n'expliciterons principalement que l'encodage d'une image.

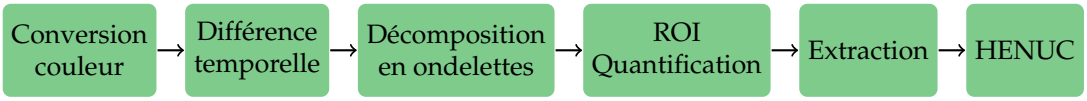



**Figure 31.** Schéma bloc du codec ROI-Waaves.

## 2.1 Conversion de l'espace colorimétrique

Le premier étage de compression de l'algorithme ROI-Waaves consiste à changer d'espaces de couleurs. L'implémentation accepte en entrée les images en nuance de gris, en RGB [1], en YCbCr [2] et en CMYK [3]. Dans le cas des nuances de gris, le codeur utilise directement les données d'entrée sans conversion pour le reste de l'encodage. Pour les autres formats (RGB, YCbCr et CMYK), ROI-Waaves propose la possibilité de convertir les images d'entrée vers un nouvel espace colorimétrique. L'intérêt de changer vers un nouvel espace est de pouvoir mieux concentrer la quantité d'information visuelle dans une composante. Cette composante contenant beaucoup d'informations pourra être compressée de manière plus légère afin d'être capable de restituer le maximum d'informations. À l'opposé, les autres composants contenant moins d'informations visuelles pourront être compressés de manière plus agressive afin de gagner en performance de compression. Ce constat peut notamment être observé lorsque qu'une image RGB, dont l'information visuelle est répartie de manière relativement uniformément sur les composantes rouge, vert et bleu, est convertie vers

1. RGB : Rouge, Vert et Bleu
2. YCbCr : Luminance, chrominance bleue et chrominance rouge
3. CMYK : Cyan, Magenta, Jaunes et Noir

un espace YCbCr dont la plus grande partie de l'information est contenue dans la composante luminance. Ce cas de figure (conversion de RGB vers YCbCr) est d'ailleurs le cas le plus couramment utilisé. Les images sont capturées en RGB pour ensuite être compressées en YCbCr. ROI-Waaves n'est pas le seul compresseur à utiliser cette espace colorimétrique comme base de compression. En effet, les codecs JPEG, JPEG-2000 ou bien même H264 pour la vidéo, l'utilisent. La formule permettant de convertir un pixel de l'espace RGB vers l'espace YCbCr, dans ROI-Waaves, est la suivante :

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} Offset \\ Offset \\ Offset \end{bmatrix} \tag{IV.1}$$

Dans laquelle, $offset$ est la moitié de la gamme de valeur d'un pixel. Pour des valeurs allant de 0 à 255, cette valeur est 128. Nous pouvons noter que les coefficients utilisés pour cette conversion ne sont pas identiques à ceux utilisés dans JPEG. La figure 32 expose la conversion de l'espace RGB vers l'espace YCbCr.

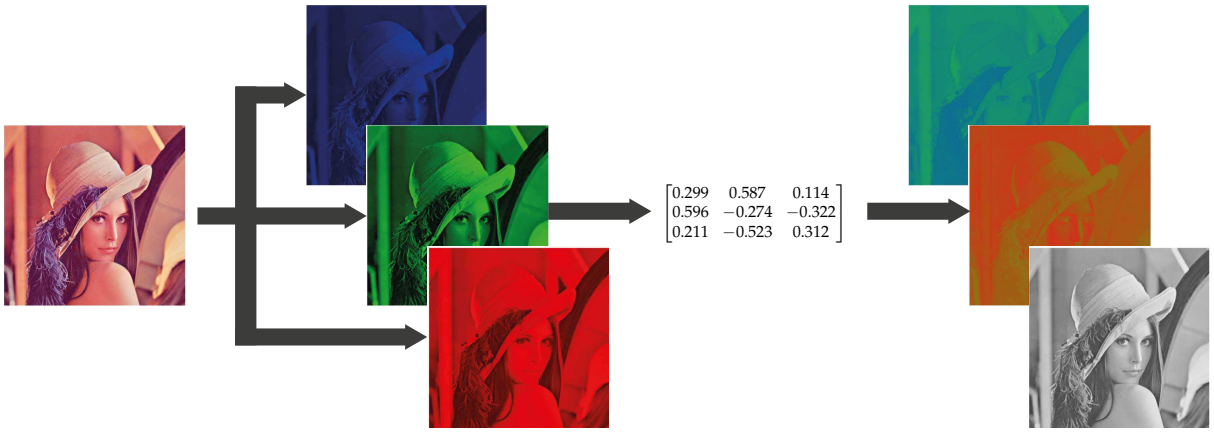



**Figure 32.** Exemple de conversion de l'espace RGB vers l'espace YCbCr.

Cette approche est valide et pratique quand les données proviennent de fichiers images ou vidéo prêts à être affichés (le RGB étant le format d'affichage le plus utilisé). En effet, ces images sont en général des fichiers dématricés en RGB. Cependant, la majorité des capteurs d'images ne capture pas les images au format RGB mais dans un schéma de Bayer [Bayer, 1976]. À la différence des formats RGB dans lesquels chaque pixel contient trois composantes, les pixels des formats Bayer ne contiennent qu'une seule composante (généralement soit rouge, verte ou bleu). Le format provient de la manière dont est réalisée la capture d'image couleur sur les capteurs numériques. La matrice de pixel composant un capteur est recouverte de filtres de couleur pour ne laisser qu'une couleur par pixel. Des groupes carrés de quatre pixels (deux lignes et deux colonnes) sont formés de deux pixels verts, d'un pixel bleu et d'un pixel rouge. L'utilisation de deux fois plus de pixels verts provient de la sensibilité de l'œil humain à

la couleur verte [Green, 1968]. Un exemple de la disposition de Bayer pour une matrice de six pixels par six pixels est illustré dans la partie gauche de la figure 33. Beaucoup de méthodes existent pour convertir le format de Bayer en format RGB [Li et al., 2008]. Cette opération, appelée dématricage, consiste à faire une interpolation des couleurs manquantes à partir des pixels voisins. C'est l'étape qui est mise en avant par les six matrices centrales de la figure 33. Par essence, il n'y a pas d'information ajoutée depuis le format Bayer au format RGB. Or, le calcul des composantes manquantes constitue une augmentation du volume de données pour une image. En effet, un pixel passe d'une composante à trois, ce qui triple la taille d'une image brute sans pour autant rajouter de l'information à l'image. La redondance créée par la dématricage n'est cependant pas inutile car elle permet de rajouter de la robustesse à l'image vis-à-vis des dégradations, notamment quand l'image est compressée à perte.

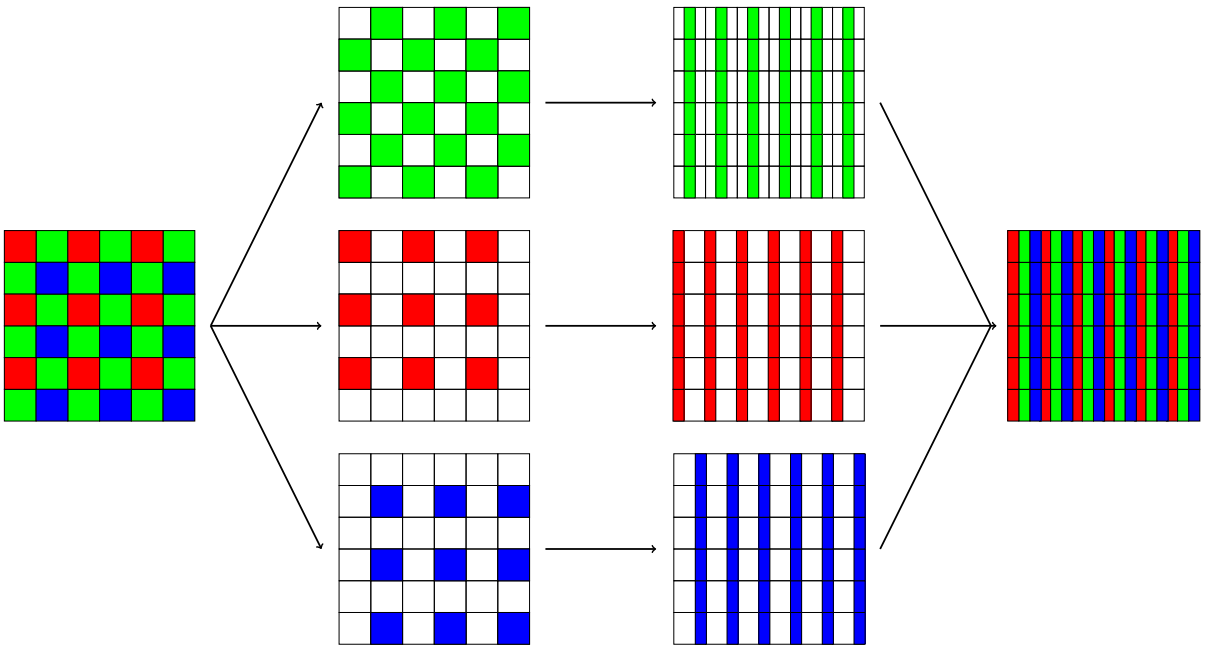


**Figure 33.** Dématricage d'une matrice six par six utilisant un format Bayer vers une matrice de pixel RGB

Au vu de ces constatations, nous avons intégré le Bayer comme un nouvel espace d'entrée. Le principal intérêt d'une telle intégration est de réduire le nombre de données à traiter, en les divisant instantanément par trois. Nous avons voulu cependant conserver l'intérêt qu'apporte le passage dans l'espace YCbCr. Ce traitement n'est pas nouveau et a déjà été réalisé dans d'autres travaux [Lee and Ortega, 2001]. Cependant, l'approche utilisée est nouvelle. En effet, l'intérêt apporté par cette approche concernait simplement la performance du codec en termes de qualité et de débit. Notre approche vis-à-vis de cette méthode est également basée sur le nombre de pixels à traiter qui est moindre par rapport à une image YCbCr classique. De ce fait, notre approche est la suivante : pour

chaque groupe de quatre pixels du format Bayer (deux lignes et deux colonnes) nous appliquons la formule suivante en utilisant les mêmes coefficients de conversion que pour passer du RGB au YCbCr.

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0 & 0.114 \\ 0.299 & 0 & 0.587 & 0.114 \\ 0.596 & -0.137 & -0.137 & -0.322 \\ 0.211 & -0.2615 & -0.2615 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G_1 \\ G_2 \\ B \end{bmatrix} + \begin{bmatrix} Offset \\ Offset \\ Offset \\ Offset \end{bmatrix} \tag{IV.2}$$

Pour les calculs des coefficients de luminance, un seul des pixels verts est utilisé pour chacun. Pour les valeurs des chrominances (bleu et rouge), la composante verte est calculée à partir de la moyenne des deux pixels verts du format Bayer. Les coefficients de luminance résultants sont ensuite fusionnés dans une matrice de pixel unique. Après cette conversion, il y a trois plans de pixels : le plan de luminance de la hauteur de l'image et de la moitié de la largeur de l'image et les deux plans de chrominances tous deux de la moitié de la hauteur et de la largeur de l'image originale. La figure 34 expose la manière dont ces trois plans sont créés à partir du format de Bayer.

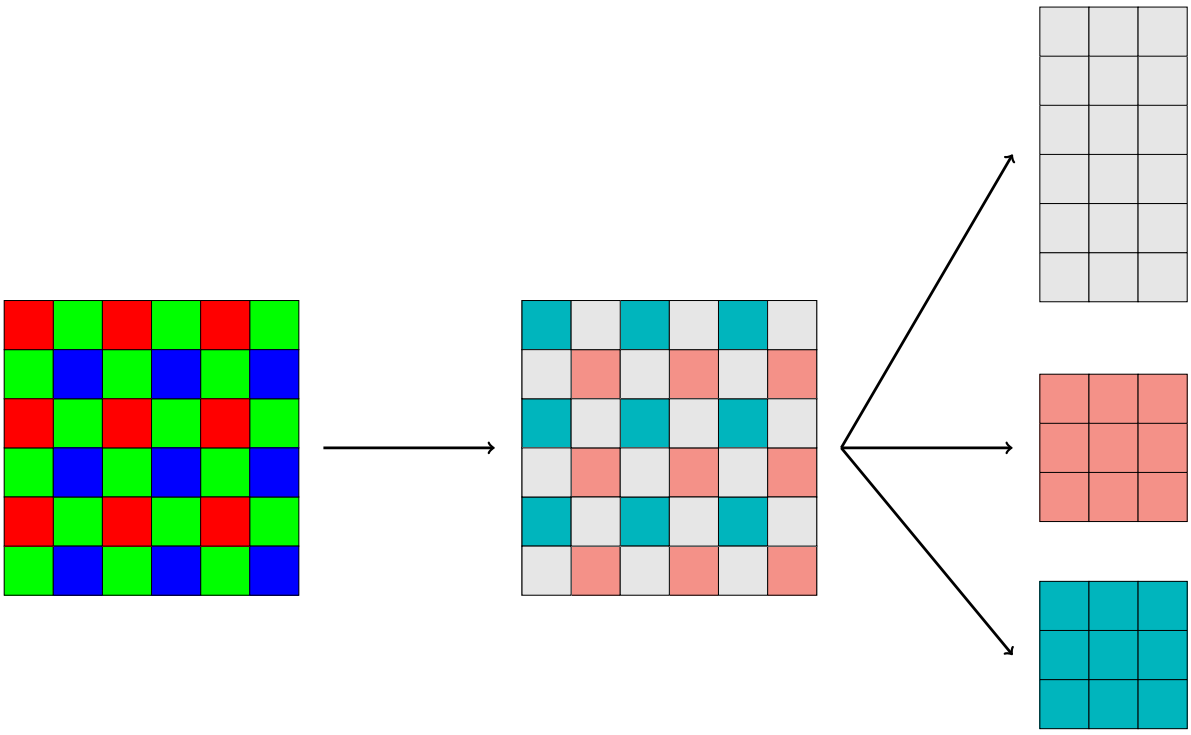


**Figure 34.** Conversion d'une matrice au format de Bayer en un plan de luminance et deux plans de chrominance.

Après avoir été convertis dans l'espace colorimétrique souhaité, les composants de l'image sont compressés séparément par le reste de la chaîne.

## 2.2 Différence temporelle

Les encodeurs vidéo utilisés dans l'industrie tels que H264, MPEG-4, etc. utilisent des méthodes d'estimation de mouvement pour réduire la **redondance temporelle** entre les images. Cette méthode est très coûteuse en temps de calcul. C'est pour cela que nous utilisons une méthode beaucoup plus simple basée sur les différences d'images. Cette méthode est viable dans notre cas d'étude car les prises de vue sont fixes. En effet, les images consécutives d'une scène auront beaucoup de similitudes et l'une des manières les plus simples et les moins coûteuses en temps de calcul pour exploiter cette redondance est de réaliser une différence pixel par pixel entre l'image courante et une autre image ressemblante. Cette image, appelée image de **référence**, est transmise avant au codeur et sert de base pour la reconstruction des images suivantes. Pour ces images, cet étage ne fait aucun traitement sur la valeur des pixels. Cependant, l'encodeur enregistre dans une mémoire cette image pour encoder les images suivantes. Dans ROI-Waaves, à un instant donné, il n'y a qu'une image de référence enregistrée au maximum.

Pour les autres images positionnées entre deux images de référence, elles sont appelées **inter**. Le traitement est simple, chaque valeur de pixel d'entrée est soustraite à la valeur du pixel de l'image de référence. Statistiquement, les valeurs de l'image inter et de l'image de référence sont très proches, ce qui entraînera des valeurs de différence relativement faibles (proche de zéro).

Dans ROI-Waaves, l'étage de différence est placé entre la conversion en couleur et la décomposition en ondelettes, afin de faire une différence entre des pixels dans l'espace spatial. Cependant, ce bloc peut être placé après la réalisation de l'étage suivante de décomposition en ondelettes afin de réaliser la différence dans le domaine spatio-fréquentiel. Afin de justifier le placement de la différence, nous évaluons l'impact sur la qualité et la performance du codec dans le chapitre VI.

Pour finir, une politique de rafraîchissement de l'image de référence doit être défini pour connaître quand une image de référence doit être remplacée par une nouvelle. Si le rafraîchissement est trop rapproché, alors beaucoup d'images sont des images de référence et nous n'exploiterions pas la redondance temporelle. Si le rafraîchissement n'est pas assez régulier, alors les images inters sont trop différentes de l'image de référence et sont plus lourdes à encoder qu'une image de référence. Dans ROI-Waaves, un seuil est fixé comme étant l'intervalle entre deux images de référence en nombre d'images. Nous évaluons l'impact de la valeur de l'intervalle sur les différentes métriques du codeur, mais la valeur typique est de rafraîchir l'image de référence toutes les dix images.

## 2.3 Décomposition en ondelettes

L'étage précédent de différence temporelle avait pour objectif d'exploiter la redondance temporelle des images. L'étage de décomposition en ondelettes a pour objectif de maximiser la **redondance spatiale** des pixels dans une image. Elle découle d'une observation simple sur les signaux naturels, ceux-ci ne sont pas stationnaires. Or les analyses aussi bien fréquentielles ou temporelles ne focalisent pas l'information de façon précise. C'est pour cela que dans les années 90, les mathématiciens ont développé leur recherche sur une nouvelle analyse temporelle/fréquentielle ou spatio-fréquentielle.

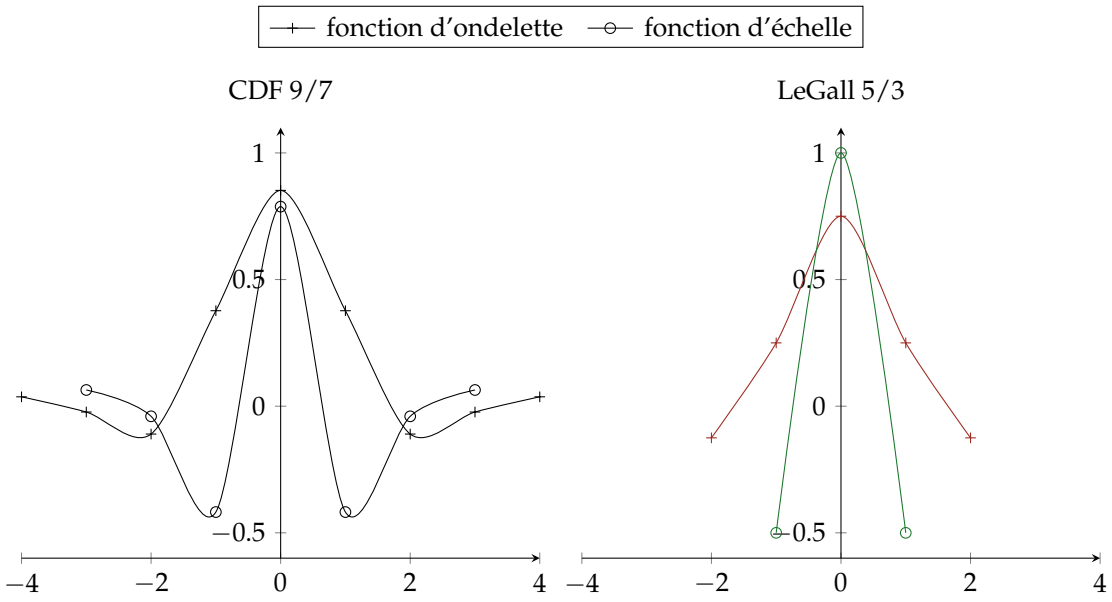
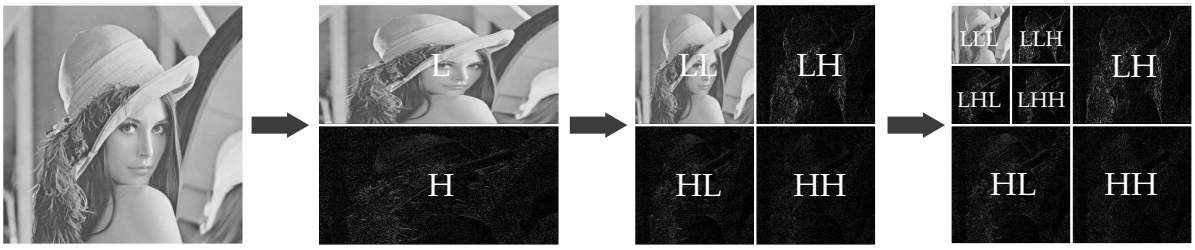



**Figure 35.** Représentation des deux filtres utilisés par ROI-Waaves pour la Décomposition en ondelette.

Chaque matrice d'une composante (soit RGB, soit YCbCr) est décomposée dans le domaine ondelette en utilisant la transformation en ondelette discrète (DWT). ROI-Waaves utilise deux ondelettes mères pour sa compression, la première est la CDF 9/7 [Antonini et al., 1992] pour de la compression avec pertes et la seconde est la LeGall 5/3 [Le Gall and Tabatabai, 1988] pour la compression sans perte. Ces deux ondelettes sont représentées dans la figure 35. Dans cette figure, les tracés rouges représentent les noyaux de convolution de la fonction d'ondelette permettant d'extraire les coefficients de basses fréquences et les tracés verts représentent les noyaux de convolution de la fonction d'échelle qui à l'inverse permet d'extraire les coefficients de hautes fréquences.

Un exemple d'une telle décomposition est présenté dans la figure 36. Dans cet exemple, nous voulons calculer les coefficients dans le domaine ondelette de l'image




Une demi itération Une itération Deux itérations

**Figure 36.** Exemple de mise en œuvre de deux itérations de DWT sur l'image Lena.

Lena. Pour ce faire, colonne par colonne de l'image, nous faisons une convolution entre la série de pixels et le noyau de la fonction d'ondelette. Le résultat de cet convolution nous donne les coefficients relatifs aux basses fréquences, placé, par convention, dans la moitié haute de l'image. En parallèle, nous réalisons une deuxième convolution, en utilisant cette fois-ci, le noyau de la fonction d'échelle afin de calculer les coefficients relatifs aux hautes fréquences. Ces coefficients sont eux placés dans la moitié basse de l'image. Le résultat de cette première décomposition peut être visualisée dans la deuxième image de la figure 36. Afin d'effectuer une itération complète de la décomposition, il faut effectuer le même traitement sur chaque ligne de l'image. Toujours par convention, les coefficients de la sous-bande de haute fréquence sont cette fois-ci placés dans la partie droite de l'image alors que les coefficients de la sous-bande de basse fréquence sont placés dans la partie gauche. Le résultat de l'itération complète peut être vu dans la troisième image de la figure 36. À ce stade, l'image est séparée en 4 sous-bandes :

- La sous-bande dans le quart haut gauche concentre les basses fréquences de l'image d'origine. Nous notons cette sous-bande LL car elle contient les basses fréquences (Low frequencies) des deux demi-itérations.
- Dans le quart haut droit et le quart bas gauche sont stockés les coefficients des sous-bandes horizontale (LH, Low/High frequencies) et verticale (HL, High/Low frequencies).
- Pour finir, dans le quart bas droit est contenus les coefficients de la sous-bandes diagonale notée HH car elle contient les hautes fréquences des deux demi-itérations.

Une chose à noter dans cet exemple, les magnitudes des coefficients des sous-bandes horizontale, verticale et diagonale ont été augmentées de manière à pouvoir être dans la même dynamique que les coefficients de la sous-bande LL. En effet, leurs valeurs sont typiquement très faibles et se positionnent autour de zéro et ne seraient donc pas visibles dans la gamme de valeur allant de 0 à 255 permettant de représenter une image. Cette distribution des coefficients est exploitée de manière à compresser l'image efficacement. En effet, l'entropie des trois sous-bandes de hautes fréquences est diminuée par rapport à l'image d'origine.

Enfin, plusieurs itérations de DWT peuvent être réalisées. Les itérations suivantes sont réalisées uniquement sur la sous-bandes LL. Pour finir l'exemple, une deuxième

itération de DWT est réalisée sur la bande LL. Celle-ci donne donc quatre nouvelles sous-bandes placées dans l'ancienne bande LL. Nous pouvons noter que la surface occupée par l'unique sous-bande LL de l'image est à chaque fois divisée par quatre, ce qui est très bénéfique pour la compression. La sous-bande contenant les basses fréquences positionnées en haut à gauche de l'image est aussi appelée la sous-bande DC [1].

Dans ROI-Waaves, le nombre d'itérations réalisées est défini par l'équation suivante :

$$\text{Nombre d'itérations} = \left\lceil \frac{\log_2 \left( min(longueur, largeur) \right)}{\log_2(16)} \right\rceil \tag{IV.3}$$

Ce nombre est défini comme étant le nombre nécessaire d'itérations afin d'obtenir une sous-bande DC de taille compris entre 16 et 31 pixels. Ces valeurs proviennent du codeur Waaves original. Elles ont été choisies afin de minimiser la surface occupée par la dernière sous-bande.

## 2.4 Quantification

La quantification est un étage très important dans la conception de la chaîne de compression car c'est dans cet étage que la qualité de l'image est définie. La quantification réalisée dans ROI-Waaves est une quantification scalaire. Elle consiste à diviser tous les coefficients de la transformée en ondelette par un seul facteur de quantification. Le facteur de quantification est une entrée de l'algorithme de compression. Plus le facteur est grand et plus les coefficients vont être divisés par un grand nombre ce qui résultera à des nombres proches de zéro. Les coefficients sont ensuite arrondis à l'entier le plus proche. Le but étant de réduire la plage de valeur des coefficients de la transformée en ondelette et donc atteindre des facteurs de compression plus élevés. Lors de la reconstruction, les coefficients entiers sont remultipliés par le facteur de quantification. Cependant, une erreur est créée due à la troncation en entier des coefficients. Cette petite erreur se traduit par une reconstruction moins fidèle de l'image et donc une perte de qualité. À l'opposé, quand le facteur de quantification est proche d'un, les valeurs des coefficients reconstruits sont beaucoup plus proches des valeurs initiales et donc permet une reconstruction fidèle de l'image. Il y a donc un compromis entre les facteurs de compressions atteignable et la qualité de l'image reconstruite et ce compromis est directement lié au facteur de quantification.

Au vu de cette constatation, nous avons développé une **quantification adaptative** permettant de quantifier les zones d'intérêt de la vidéo avec un plus petit facteur de compression que les zones d'arrière-plan de la scène. Pour ce faire, nous définissons plus seulement un mais deux facteurs de compression : un facteur $Q_{ROI}$ définissons la

1. DC : direct current. L'expression fait référence à la composante continue d'un signal.

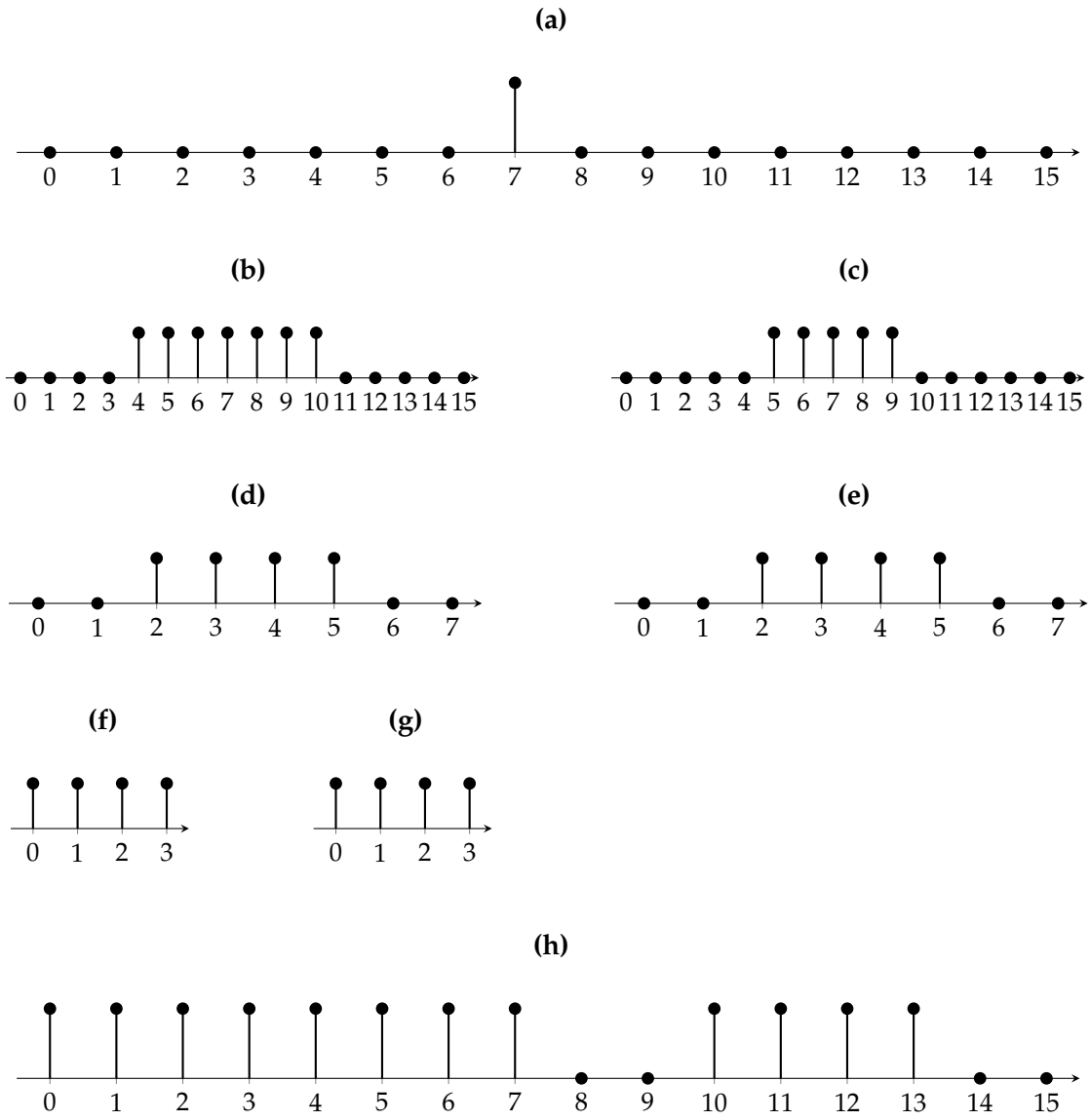



**Figure 37.** Dilatation d'un masque 1D de zone d'intérêt après la transformée en ondelettes utilisant une fonction d'ondelette de taille quatre et une fonction d'échelle de taille trois.

compression de la zone d'intérêt et un facteur $Q_{NROI}$ pour l'autre zone. Le compresseur en quantifiant les coefficients d'ondelettes utilisera soit l'un soit l'autre. Pour se faire, les coefficients sont classifiés en deux groupes : les coefficients d'intérêt et les coefficients de fond. Un coefficient est classé dans les coefficients d'intérêt s'il intervient dans le calcul d'au moins un pixel d'intérêt lors de la reconstruction de l'image. À partir du masque d'intérêt du domaine espace donné en entrée du codeur, un nouveau masque d'intérêt est calculé pour les coefficients d'ondelettes en implémentant la règle précédente.

Voyons un exemple sur une convolution 1D d'une fonction d'ondelette de largeur quatre et d'une fonction d'échelle de largeur trois sur seize coefficients dont l'unique coefficient d'intérêt est celui du milieu. La figure 37 montre la convolution de ces fonctions avec les seize coefficients. La sous-figure 37a montre le masque initial dans le domaine espace dans lequel seul le septième pixel est d'intérêt. Les sous-figures 37b et 37c montrent respectivement le résultat de la convolution avec la fonction d'ondelette et la fonction d'échelle. Nous remarquons qu'avec une itération et un noyau de convolution de taille trois, la zone d'intérêt s'est dilatée de deux pixels de chaque côté du coefficient initial, ce qui correspond à une zone de cinq coefficients, sous-figure 37c. La dilatation opérée est en fait la taille du noyau moins un coefficient. Les sous-figures 37d et 37e montrent le résultat après sous-échantillonage des résultats précédents. Enfin, les sous-figures 37f et 37g montrent le résultat de la convolution des deux noyaux et sous-échantillonage pour une deuxième itération de DWT. La dernière sous-figure 37h montre le masque final dans l'espace ondelette pour deux itérations. La transformation d'un masque 2D est équivalente mais est propagée sur deux dimensions plutôt qu'une. L'exemple d'un masque contenant deux ROI et transformé sur trois itérations est montré dans la figure 38.

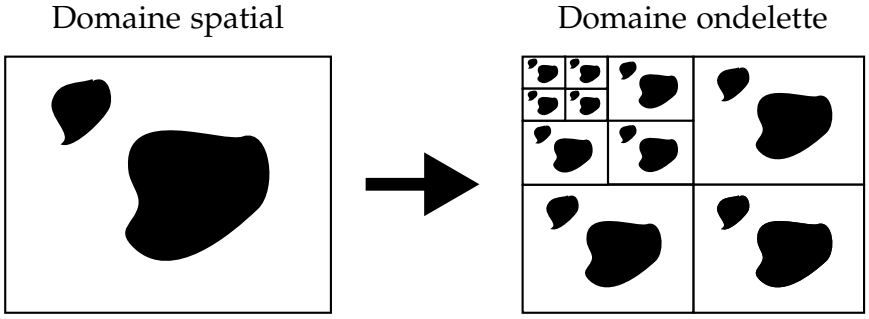



**Figure 38.** Transformation d'un masque du domaine spatial au domaine ondelette.

À partir de ce masque, il est possible par la position du coefficient de savoir à quelle classe il appartient. Les coefficients d'intérêt sont quantifiés en utilisant le facteur $Q_{ROI}$ tandis que les autres sont quantifiés par le facteur $Q_{NROI}$. Les valeurs des deux facteurs sont entrées l'utilisateur du compresseur. Cependant il est typique que $Q_{ROI}$ soit inférieur à $Q_{NROI}$ de manière à moins dégrader les zones d'intérêt. Les valeurs par défaut sont dix pour $Q_{ROI}$ et cent pour $Q_{NROI}$.

## 2.5 Extraction

L'extraction est réalisée en deux étapes : la première étape est une étape de filtrage des coefficients d'intérêt. L'objectif de cette étape est de laisser passer les coefficients d'intérêt et d'enlever les autres coefficients. La deuxième étape est une réorganisation des plans de bits à encoder.

**Filtrage de zone**

Cette étape originale se base sur l'observation que l'utilité d'avoir une cadence vidéo de cent images par seconde est pertinent que dans les zones d'intérêts. De ce fait, nous avons fait l'hypothèse qu'il serait utile de mettre à jour que les zones d'intérêt à cette cadence. Cette constatation revient en fait à dire que pour un certain nombre d'images d'une vidéo, il n'est utile de transmettre que les coefficients des zones d'intérêt. Pour ce faire, nous avons défini que les coefficients de fond ne seront mis à jour qu'en même temps que les images de référence. Donc, dans le cas d'une image de référence, l'intégralité des coefficients est envoyée à la suite de l'encodeur. La sous-bande DC est toujours transmise entièrement. Pour celle-ci, un parcours suivant la courbe de Peano représentée dans la figure 39 permet de limiter les discontinuités entre chaque ligne de l'image par rapport à un parcours ligne par ligne.

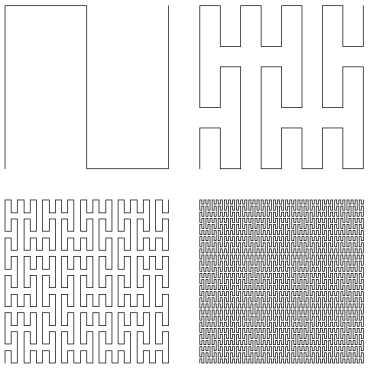


**Figure 39.** Courbe de Peano. Source de l'image

Pour le cas des images inters, les coefficients sont parcourus dans l'ordre ligne par ligne par bloc de taille $M \times M$. La valeur de $M$ est un paramètre qui vaut 128 par défaut. Nous étudierons l'impact que peuvent avoir la taille des blocs sur la performance du codeur. Une étape de filtrage est ensuite réalisée afin de ne transmettre que les coefficients des zones d'intérêt. Les coefficients sont classifiés en utilisant le même masque que pour l'étage de quantification. En pratique, un tampon d'entrée contenant tous les coefficients d'une sous-bande bloc par bloc est parcouru. Un deuxième tampon de sortie contenant les coefficients à encoder est créé vide. Quand un coefficient du tampon d'entrée est un coefficient d'intérêt, il est ajouté au tampon de sortie. À la fin du parcours de la sous-bande, le tampon de sortie contient tous les coefficients d'intérêt

de la sous-bande en question et seuls les coefficients stockés dans ce tampon sont traités dans le reste de l'algorithme.

Le fait que seulement une partie des coefficients est effectivement utilisée pour l'encodage nous permet de faire une optimisation dans le calcul de la transformée en ondelette. En effet, si les coefficients de l'image ne sont pas dans une zone d'intérêt, alors il n'y a pas d'utilité à les calculer et donc peuvent être sautés.

**Réorganisation des plans de bits**

La première étape de la réorganisation est de séparer les signes des magnitudes. L'encodeur enregistre alors les signes dans un tampon et les magnitudes dans un autre. La magnitude maximale de ces coefficients est repérée et permet de déterminer le nombre de plans de bits à encoder. Un plan de bits est l'ensemble des bits correspondant à une position dans les mots des magnitudes. Par exemple, le deuxième plan de bits est l'ensemble des bits numéro deux du tampon d'entrée. Les plans de bits sont encodés des bits de poids fort jusqu'aux bits de poids faible. Nous notons un plan de bits fils d'un autre comme le plan de bits le suivant dans l'ordre d'encodage. De même, nous définissons un plan de bits parent d'un autre comme le plan de bits le précédent dans l'encodage.

L'objectif de la réorganisation des plans de bits est de changer l'ordre des bits dans le plan courant en fonction des valeurs des bits du plan parent. Nous notons $P(x_{i,j})$ la fonction implémentant la politique permettant de savoir si le bit $x_i$ du plan $j$ est prioritaire. Cette fonction permet de classer les bits d'un plan en deux classes, les prioritaires et les non prioritaires. Les prioritaires sont alors encodés en premier pour ensuite encoder les bits non prioritaires. Pour ce faire, nous avons implémenté trois politiques différentes de réorganisation des plans de bits :

- la première politique, la plus simple, est une politique qui ne prend en fait pas en compte les valeurs des plans de bits parents. Celui-ci est le plus rapide car il ne nécessite aucun traitement.
- La deuxième politique prend en compte du plan de bits directement parent. Quand le bit du plan parent est vrai alors le bit du plan courant est prioritaire.
- La dernière politique prend en compte l'intégralité des plans de bits parents. Quand un des bits des plans parents est vrai alors le bit est prioritaire. Cela revient à faire un ou logique avec tous les bits des plans parents.

D'après nos évaluations en interne, la première méthode permet d'obtenir des résultats de compression équivalents aux autres méthodes et permet une compression plus simple en implémentation. Une fois que chaque plan de bits est extrait et réorganisé, il est envoyé au codeur entropique pour être compressé. Le dernier étage de la compression est la compression des signes mis de côté jusque-là. Tous les signes sont parcourus dans le même ordre que les magnitudes. Pour chaque signe, le codage suivant est utilisé :

- pour un signe positif, le codeur émet un bit à 1.
- Pour un signe négatif , le codeur émet un bit à 0.
- Pour un zéro, le codeur n'émet rien.

Cet encodage mise sur l'hypothèse qu'une majorité des coefficients en ondelette soit nulle.

## 2.6 Codage entropique : HENUC

Dans cette sous-section, nous détaillons la manière dont fonctionne le codeur entropique utilisé dans ROI-Waaves. Pour ce faire, nous exposons deux codeurs utilisés comme brique de base pour le codeur entropique. Nous décrivons ensuite comment sont utilisés ces deux codeurs de manière hiérarchique pour réaliser un codeur entropique parallèle.

**Codages énumératifs**

L'idée générale des codages énumératifs a été proposée par Lynch et Davisson [Lynch, 1966, Davisson, 1966] de manière séparée. Nous appelons ce codage l'encodage LD. Nous considérons que le message à encoder est divisé en vecteur de bit de taille $n$. Dans ce message, nous notons $s$ le nombre de bits vrais et $n-s$ le nombre de bits faux. En connaissant $n$ et $s$, il est possible d'organiser le vecteur de bit de $\binom{n}{s} = C_s^n = \frac{n!}{s!(n-s)!}$ manières différentes. En ordonnant toutes ces permutations en suivant un critère arbitraire, nous pouvons désigner la permutation initiale par son rang dans la liste de combinaisons. C'est ce rang que le codeur va utiliser comme code pour la compression. Il lui faut également donner le nombre de bits vrais $s$ ainsi que la taille du vecteur $n$. Généralement, $n$ est prédéfini pour ne pas avoir à l'envoyer. La taille de $C_s^n$ en bit est égale à $\lceil \log_2(C_s^n) \rceil$ bits. À cette longueur doit se rajouter la valeur de $s$ qui s'encode sur $\log_2(n+1)$.

Les deux courbes de la figure 40 représentent l'évolution de cette longueur en fonction de la valeur de $s$ pour $n = 8$ et $n = 64$. Sans encodage, le symbole de longueur $n$ coûte $n$ bits à envoyer. Ce coût est représenté par la ligne horizontale pointillée sur le schéma. De cette figure, nous pouvons déterminer pour quelles valeurs de $s$ l'encodage sera bénéfique. Nous pouvons voir que pour les symboles constitués que d'uns ou de zéros, le symbole est simplement encodé avec le nombre d'uns ou de zéros (4 bits pour un symbole de 8 bits). Nous pouvons déduire que l'encodage sera efficace quand la source produit soit beaucoup de valeurs binaires zéros ou beaucoup de valeurs binaires uns et sera inefficace quand un symbole est un mélange de zéros et d'uns.

Nous proposons un exemple d'encodage de la valeur *00101000*. La longueur du symbole $n$ est 8. Le nombre de bits vrais $s$ dans ce symbole est 2. Nous devons donc énumérer les 28 manières d'organiser 2 bits vrais dans un symbole de 8 bits. Le tableau

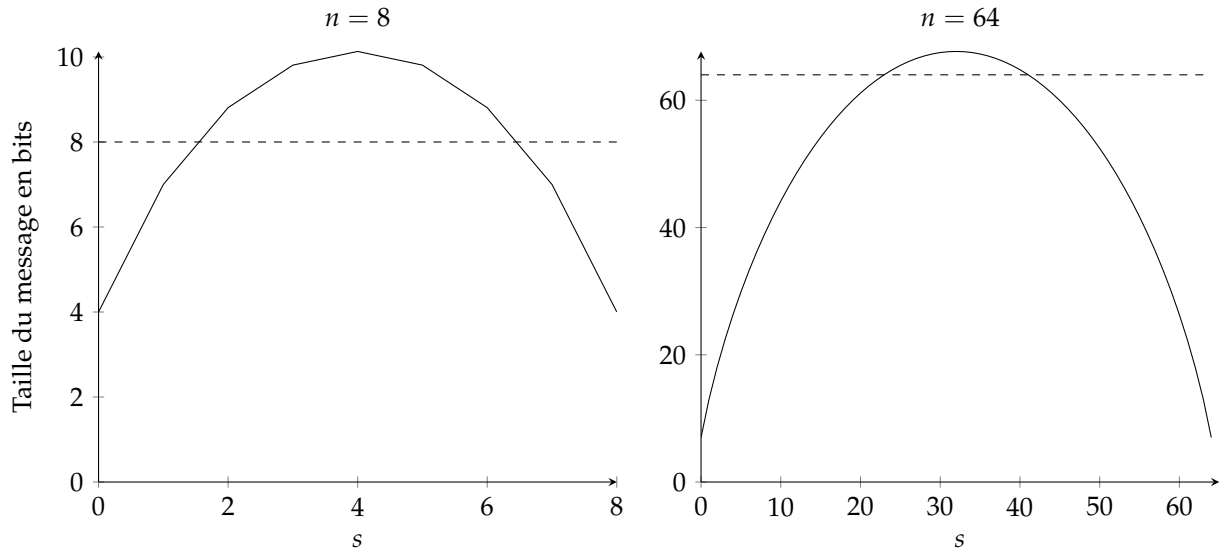



**Figure 40.** Longueur d'un symbole encodé en fonction de $s$ le nombre de bits a uns pour des symboles de longueur 8 et 64. La limite horizontale représente la longueur du symbole sans encodage.

5 les liste dans l'ordre lexicographique. Notre symbole correspond au rang 13 de cette liste. Le rang est encodé sur 5 bits ($\log_2(28)$). Pour cette taille et ce nombre de bits vrais, l'encodage est inefficace.

**Tableau 5.** Énumération des 28 combinaisons d'avoir 2 bits à un dans 8 bits dans l'ordre lexicographique.

| Rang | Code | Rang | Code | Rang | Code |
|---|---|---|---|---|---|
| 00000011 | 0 | 00100001 | 10 | 01010000 | 19 |
| 00000101 | 1 | 00100010 | 11 | 01100000 | 20 |
| 00000110 | 2 | 00100100 | 12 | 10000001 | 21 |
| 00001001 | 3 | 00101000 | 13 | 10000010 | 22 |
| 00001010 | 4 | 00110000 | 14 | 10000100 | 23 |
| 00001100 | 5 | 01000001 | 15 | 10001000 | 24 |
| 00010001 | 6 | 01000010 | 16 | 10010000 | 25 |
| 00010010 | 7 | 01000100 | 17 | 10100000 | 26 |
| 00010100 | 8 | 01001000 | 18 | 11000000 | 27 |
| 00011000 | 9 | | | | |

Le problème de l'énumération est que quand $n$ grandit et que $s$ est environ égale à $n/2$, alors le nombre de combinaisons devient très grand. Pour $n$ égal à 64, nous obtenons un nombre de combinaisons pour $s$ égal à 32 de $C_{64}^{32} = 1832624140942590534$ qui n'est pas générable de façon raisonnable sur des machines actuelles. Pour contourner cette limitation, une méthode itérative est également utilisable et décrite dans l'algorithme 1. Cet algorithme est bien plus rapide car sa complexité est linéaire avec la taille des symboles.

**Algorithme 1.** algorithme de calcul du rang sans énumération pour l'encodage LD

**Entrées :** $n$, $s$, vecteur binaire
**Sorties :** rang

```
1 Fonction CalculeRang
2 |   rang = 0;
3 |   pour i allant de 0 à n - 1 faire
4 |   |   si le bit i vaut 1 alors
5 |   |   |   rang = rang + C^s_{n-i-1};
6 |   |   |   s = s - 1;
7 |   |   fin
8 |   fin
```

Une généralisation de cet encodage est proposée par Öktem dans sa thèse [Öktem, 1999, Öktem and Astola, 1999]. Il généralise ce codage en permettant, non plus d'encoder des vecteurs de bits mais des vecteurs d'entiers positifs. Les entiers $x_i$ présents dans ce vecteur appartiennent à l'intervalle $[0, N]$. Nous appelons cet encodage l'encodage $\text{EBI}_N$ [1], avec $N$ qui est la borne maximale que peut prendre un entier dans le vecteur. Comme pour l'encodage LD, nous notons $n$ la taille du vecteur et $s$ la somme des éléments de ce vecteur. Nous définirons la fonction $f_N(p, q)$ comme le nombre de vecteurs d'entier appartenant à $[0, N]$, de taille $q$ et dont la somme des éléments est égale à $p$. Cette fonction est définie récursivement ainsi :

$$f_N(p, 1) = \begin{cases} 1, & \text{si } 0 \leq p \leq N \\ 0, & \text{sinon} \end{cases} \tag{IV.4}$$

$$f_N(p, q) = \sum_{i=p-N}^{p} f_N(i, q-1) \tag{IV.5}$$

La manière d'encoder un vecteur d'entier est décrite par l'algorithme 2. Nous remarquons que cet algorithme est capable de produire le même résultat que l'encodage de LD quand la valeur de $N$ est égale à 1. En pratique, les valeurs de la fonction $f_N$ sont précalculées et rangées dans une table de correspondances pour accélérer le processus d'encodage.

Plus de détails, sur cet algorithme d'encodage, sont trouvables dans la thèse d'Öktem [Öktem, 1999]. Une analyse que nous pouvons faire sur cet algorithme est sa complexité. Comme nous nous intéressons de savoir comment se comporte cet algorithme quand la taille de l'entrée grandit, nous analysons l'évolution de la complexité par rapport à la taille du vecteur d'entrée. Les opérations internes aux deux boucles sont effectuées en temps constant vis-à-vis de la taille du vecteur $n$ ($\mathcal{O}(1)$). La boucle ligne 4 ne dépend pas de la taille du vecteur d'entrée donc sa complexité ne change pas et est $\mathcal{O}(1)$. À la différence de la boucle interne, la boucle ligne 4 varie de manière linéaire avec le nombre

1. Enumerative Encoding of Bounded Integers

**Algorithme 2.** algorithme de calcul du rang sans énumération pour l'encodage EBI

**Entrées :** $n$, $s$, $N$, vecteur d'entiers positifs
**Sorties :** rang

```
1 Fonction CalculeRang
2 |  rang = 0;
3 |  pour i allant de 0 à n - 1 faire
4 |  |  pour pour j allant de 0 à x_i - 1 faire
5 |  |  |  rang = rang + f_N(s, n − i − 1);
6 |  |  |  s = s - 1;
7 |  |  fin
8 |  fin
```

d'éléments du vecteur. La complexité totale de l'algorithme est donc équivalente à la complexité de la boucle externe, c'est-à-dire $\mathcal{O}(n)$. En d'autres mots, le temps pour traiter un vecteur de $n$ éléments est égal à $n$ fois le temps de traiter un élément.

**Codage énumératif hiérarchique**

Le codeur énumératif hiérarchique est un schéma de compression combinant les deux codages précédemment décrits : LD et EBI. Nous avons vu que l'inconvénient de ces deux codages est la difficulté d'envoyer les paramètres d'encodage au décodeur (principalement la somme des éléments). Le codeur HEC ou HENUC [1] prévoit d'encoder de manière hiérarchique les paramètres de compression. L'algorithme commence par encoder un message en utilisant l'encodage LD. La longueur des vecteurs utilisés par l'encodage LD sera fixée et notée $n_0$. Le découpage en vecteur de longueur $n_0$ va créer des vecteurs dont la somme des valeurs sera notée $s_{0,i}$ pour l'ième vecteur. Nous parlons de cet encodage comme le niveau 0. Les sommes $s_{0,i}$ doivent être envoyées au décodeur. Pour ce faire, elles sont encodées en utilisant l'encodage $\text{EBI}_{n_0}$. L'utilisation de $\text{EBI}_{n_0}$ est possible car les sommes trouvées dans le codage LD ne peuvent pas dépasser la longueur des vecteurs. De la même manière que pour le niveau 0, la taille des vecteurs d'entiers est fixée et est notée $n_1$ et la somme de chaque vecteur est notée $s_{1,i}$. À ce niveau, nous aurons $n_1$ fois moins d'encodage qu'au niveau 0, car une somme du niveau LD est regroupé dans un vecteur de $n_1$ éléments. Récursivement, le niveau 2 encode des vecteurs de longueur $n_2$ de sommes du niveau 1. La somme maximale trouvable dans un vecteur du niveau 2 est la somme maximale du niveau 0, $n_0$ fois le nombre d'éléments regroupés au niveau 1, $n_1$. Le niveau 2 encode le vecteur de sommes du niveau 1 avec l'$\text{EBI}_{n_1 * n_0}$.

Récursivement, le niveau $j$ encode les sommes du niveau $j-1$ par vecteur de taille $n_j$. Ces sommes sont notées $s_{j,i}$. La somme maximale du niveau $j$ est $n_{j-1} * n_{j-2}$ ce qui implique que l'encodage utilisé sera $\text{EBI}_{n_{j-1} * n_{j-2}}$. Lorsqu'une seule somme est restante, l'encodage hiérarchique est fini. Cette dernière somme est alors encodée en brute pour

1. Hierarchical Enumerative Coding

pouvoir être lue par le décodeur. Les sommes encodées sont ensuite écrites encodées de la racine jusqu'aux feuilles en suivant l'ordre d'un parcours en largeur.

Nous proposons un exemple d'encodage de la chaîne binaire suivante *1000 0001 0001 0010 0101 1000 0000 0000 0001*. Cet exemple est exposé avec une configuration donnée, cependant il est possible de paramétrer un codage hiérarchique différent en utilisant une autre configuration. Les caractéristiques de l'encodeur utilisé pour cet exemple sont les suivantes : l'arbre aura une profondeur de 3 avec pour chaque niveau $n_0 = 4$, $n_1 = 3$ et $n_2 = 3$. Les feuilles sont encodées en utilisant le codage LD et les autres nœuds sont encodés en utilisant le codage EBI. En suivant ces caractéristiques, nous obtenons l'arbre de la figure 41.

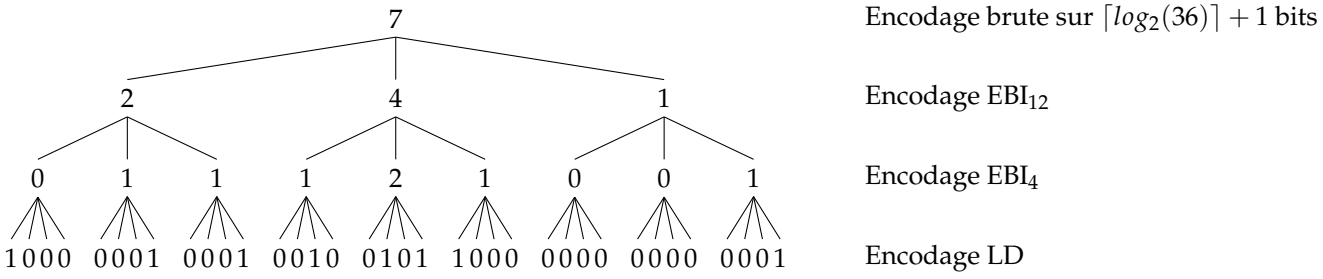



**Figure 41.** Exemple d'arbre utilisé pour l'encodage HENUC.

Nous obtenons à partir de cet arbre les différents nœuds à encoder. Nous commençons par encoder la racine 7 en binaire avec $log_2(36 + 1)$ bits. Le niveau inférieur, constitué des valeurs 2, 4 et 1, est encodé avec l'encodeur $EBI_{12}$. L'$EBI_{12}$ est utilisé car les valeurs des nœuds encodés ne peuvent dépasser la valeur 12. Le niveau suivant est également encodé avec l'encodeur EBI mais cette fois-ci avec le paramètre 4 car les nœuds prennent des valeurs de 0 à 4. Le dernier niveau, les feuilles, est encodé en utilisant l'encodage LD. Cet encodage nous donne la suite :

000111, $EBI_{12}$(2,4,1), $EBI_4$(0,1,1), $EBI_4$(1,2,1), $EBI_4$(0,0,1), LD(1000), LD(0001), LD(0001), LD(0010), LD(0101), LD(1000), LD(0000), LD(0000), LD(0001).

On remarque que chaque nœud contient la somme du nombre de bits vrais sur les feuilles de la branche à laquelle il appartient. L'exemple donné est le codeur HENUC au sens large du terme. Dans le cas du codeur ROI-Waaves, les caractéristiques utilisées pour HENUC sont les suivantes : le niveau le plus bas (les feuilles) est encodé en utilisant le codage LD en regroupant par groupe de 64 bits. Le deuxième niveau est encodé avec le codeur $EBI_{64}$ et regroupe huit valeurs du niveau inférieur. Le troisième niveau encode en $EBI_{512}$ et regroupe 4 valeurs du deuxième niveau. Au-delà du niveau trois, les nœuds sont regroupés par deux pour former un arbre binaire, où le nœud

du niveau $n$ est la somme de ces deux fils du niveau $n-1$. Ces valeurs sont encodées en binaire avec le minimum de bits possible. Dans le schéma traditionnel d'encodage, l'arbre est ensuite parcouru en largeur de la racine jusqu'aux feuilles.

De manière à pouvoir paralléliser le traitement de l'encodage de l'arbre HENUC, nous avons développé un nouveau parcours d'arbre. L'organisation et la génération de l'arbre et de ses coefficients restent identiques à la description précédente. Une fois l'arbre entièrement généré, celui-ci est découpé en plusieurs sous-arbres et en un arbre de tête. Les sous-arbres sont constitués de 2048 bits de données d'entrée encodés en 8 symboles LD, 4 symboles $\text{EBI}_{64}$ et 1 symbole $\text{EBI}_{512}$. L'arbre de tête est l'arbre contenant tous les nœuds binaires de la partie supérieure de l'arbre original. La figure 42 montre le découpage d'un arbre en sous-arbres. Dans cet exemple, la totalité de l'arbre n'est pas dessinée pour ne pas surcharger la figure. L'idée du nouveau parcours est d'encoder dans un premier temps l'arbre de tête comme dans le codeur Waaves, puis d'encoder un par un les sous-arbres. C'est sur cette différence que le nouveau parcours est intéressant. En effet, dans le codeur original, tous les symboles utilisant $\text{EBI}_{512}$ étaient encodés puis tous les $\text{EBI}_{64}$ puis tous les LD. Dans notre parcours, il y a un entrelacement des codages. Pour une implémentation matérielle, il est possible de créer un système traitant les sous-arbres en parallèle et utilisant des ressources différentes (codage différent). De ce fait, les encodeurs de plusieurs sous-arbres ont besoin de ressources différentes et peuvent donc travailler en parallèle sans conflit. Dans le cas d'une implémentation logicielle, le traitement des sous-arbres peut être réalisé en parallèle dans plusieurs threads.

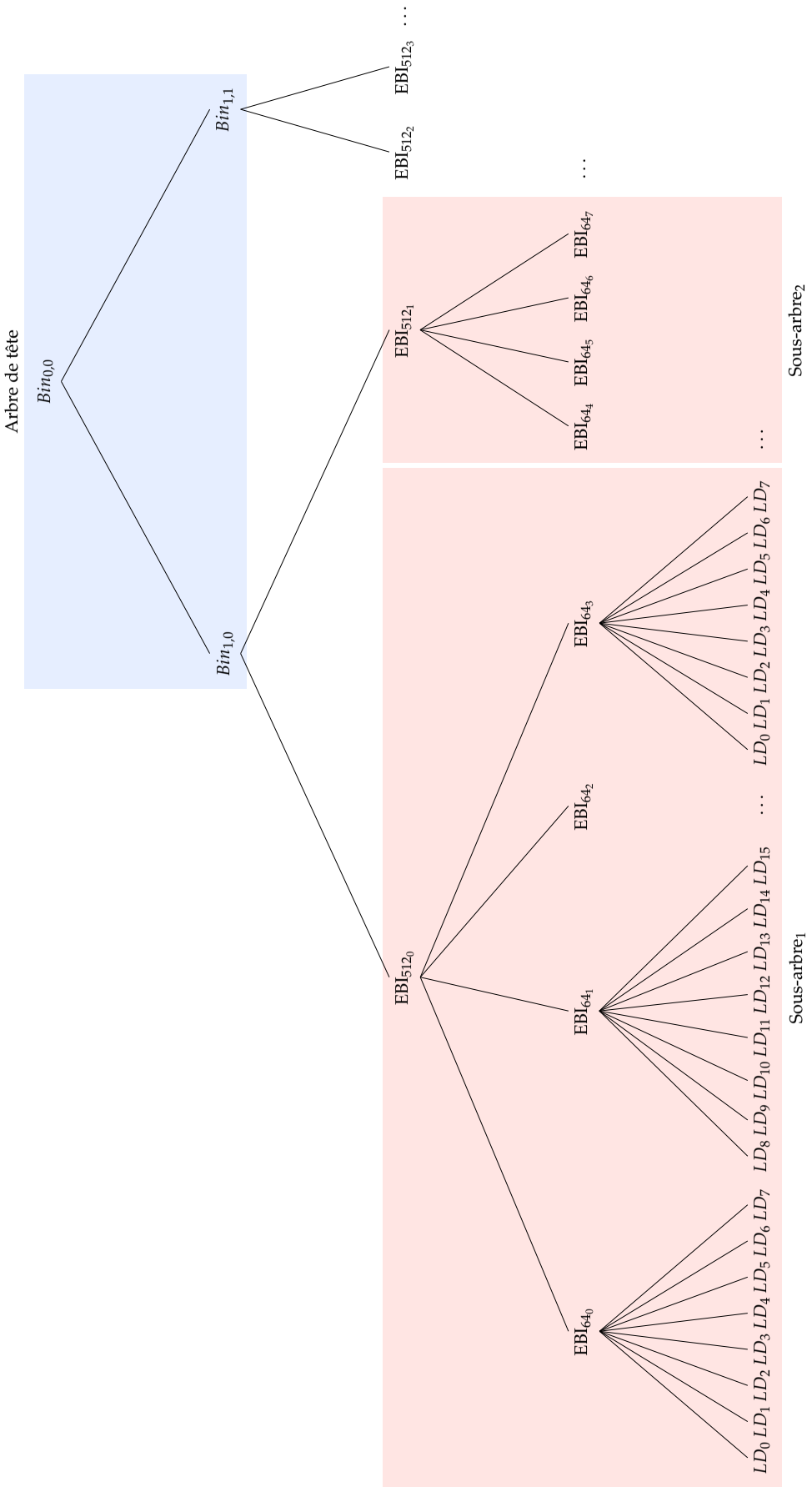



**Figure 42.** Découpage de l'arbre global en arbre de tête et en sous-arbres pour l'encodage HENUC

### 2.7 Transmission des ROI

L'une des particularités du codeur ROI-Waaves est d'encoder différemment certaines zones de l'image pour atteindre de meilleures performances. L'information de la position de ces zones dans l'image est contenue dans un masque binaire. Il est à noter que ce masque peut être changé à chaque image de référence et peut donc évoluer au cours d'une séquence vidéo. Le décodage d'une séquence fait l'hypothèse de connaître le masque des ROI afin de pouvoir décoder une image. Il est envoyé avant le début d'une image de référence. La taille du masque est proportionnelle à la taille de l'image. En brut, il est alloué un bit par pixel de l'image. Pour une image Full-HD, cela équivaut à $1920 * 1080 = 2073600\ bits = 259200\ octets$. Dans le cas d'une image au Bayer, la plus grande composante est la verte. Nous utilisons donc la taille de cette composante comme taille de masque ce qui revient à avoir $960 * 1080 = 1036800\ bits = 129600\ octets$.

Cette taille brute correspond à une grosse quantité de données que nous proposons de compresser pour gagner de la place. Cette information doit être transmise sans perte au décodeur donc il est impératif d'utiliser des codeurs entropiques. Nous avons implémenté deux méthodes différentes pour cette application. La première est d'utiliser le codeur HENUC décrit précédemment. La deuxième méthode est d'utiliser le codage RLE décrit dans l'annexe 1. L'évaluation des deux méthodes est réalisée dans le chapitre VI mais nous utiliserons par défaut la méthode utilisant HENUC.

## 3 Conclusion

Dans ce chapitre, nous avons décrit l'algorithme de compression ROI-Waaves visant à compresser la scène d'un examen EEG. Pour ce faire, nous avons exploité les spécificités de la vidéo d'entrée et de la prise de vue. Nous avons décrit les différents blocs de traitement de cet algorithme. Nous avons expliqué comment nous encodons directement les données Bayer d'une caméra pour accélérer la compression. Nous avons décrit la manière dont des zones d'intérêt sont utilisées pour quantifier adaptivement les coefficients en ondelette de l'image. De plus, nous avons utilisé ces zones pour avoir deux fréquences de mises à jour distinctes dans une séquence vidéo. Pour finir, nous avons proposé un codeur entropique HENUC qui permet de faire un encodage parallèle très adapté à l'objectif embarquabilité.

Nous comparons ROI-Waaves aux codecs de l'état de l'art dans le chapitre VI. Le codec développé comporte également de nombreux paramètres/options. Nous évaluons également l'impact de ces paramètres dans le même chapitre.

Chapitre V

# Architecture et réalisation



Les verrous techniques que nous avons pus lever dans les chapitres précédents nous permettent, à présent, de concevoir un système complet visant à acquérir, synchroniser et compresser un examen EEG en prévision d'être téléexpertisé.

Dans un premier temps, nous verrons l'architecture générale du système et en particulier ces interfaces avec les équipements voisins. Nous continuons en détaillant deux implémentations du système : une première dite logicielle dans laquelle la compression vidéo est réalisée sur processeur et une deuxième dite matérielle dans laquelle elle est implémentée dans la cible FPGA.

## 1 Architecture Système

Pour rappel, le dispositif doit acquérir des signaux physiologiques, une vidéo et un signal audio, le tout en utilisant les mécanismes de synchronisation décrits dans le chapitre III et compresser chaque modalité en utilisant l'algorithme de compression adapté notamment le codec ROI-Waaves décrit dans le chapitre IV.

## 1.1 Description globale

Le système dialogue avec une console de visualisation permettant au technicien présent de piloter l'acquisition. La figure 43 schématise sous forme de schéma bloc le dispositif au cours d'un examen. Dans cette figure, les flux de données sont représentés par les arcs rouges et les flux de contrôle par les arcs blues. Les équipements de bas niveau d'**acquisition,** électrodes et microphone sont connectés à un module de **numérisation** permettant de convertir les signaux analogiques en signaux numériques. Ces signaux numériques sont transmis au module d'acquisition. Celui-ci doit effectuer le dialogue avec les équipements et récupérer les données numériques de chaque modalité. Ces données sont ensuite transmises à deux sous-modules différents : au module de **compression** et au module de **prévisualisation** (noté **preview).** Le flux allant à la compression est destiné à être encapsulé pour former le fichier d'examen. À l'inverse, le flux allant au preview est seulement utilisé pour faire une prévisualisation des données acquises lors de l'acquisition. Les données de prévisualisation sont directement envoyées à la console pour que le praticien ait un retour. Les données compressées sont, elles, transmises au module de **synchronisation** haut niveau qui va former le fichier MKV contenant les données vidéo, physiologiques et audio compressées et synchronisées. Pour les synchroniser, ce module utilise les timestamps enregistrés acquis par le module de synchronisation bas niveau utilisant les déclencheurs du module d'acquisition. Le fichier formé est envoyé à la console. Toute l'acquisition est contrôlée par le module contrôle qui fait le relais entre la console du praticien et les différents modules du dispositif.

Dans ce dispositif, nous pouvons remarquer que les données sont envoyées deux fois, une fois par le biais du fichier et une fois par la prévisualisation. Même si cela est redonnant, il est important de comprendre pourquoi il y a cette séparation et pourquoi les deux flux cohabitent. Le flux de prévisualisation est un flux destiné à être consommé très rapidement par la console. Il n'a aucune valeur médicale et ne sert pas à faire de diagnostic. À l'inverse, le flux compressé et synchronisé sert à réaliser le diagnostic du patient. Cependant, la compression et la synchronisation sont des processus qui peuvent prendre du temps ce qui implique que si la console l'utilisait pour faire sa prévisualisation, il y aurait un temps de décalage plus ou moins long en fonction des performances du compresseur. Le découplage de la génération du fichier examen avec la prévisualisation permet alors de moins contraindre le compresseur vidéo en termes de performance. En d'autres termes, la génération du fichier peut être réalisée de manière asynchrone à l'acquisition de l'examen, sous réserve que les modules aient assez de mémoire tampon pour contenir les données de l'examen. Le fichier examen envoyé à la console contient les modalités vidéo, audio et physiologique, la console se charge de rajouter au flux BIO-MKV les annotations rentrées par le technicien.

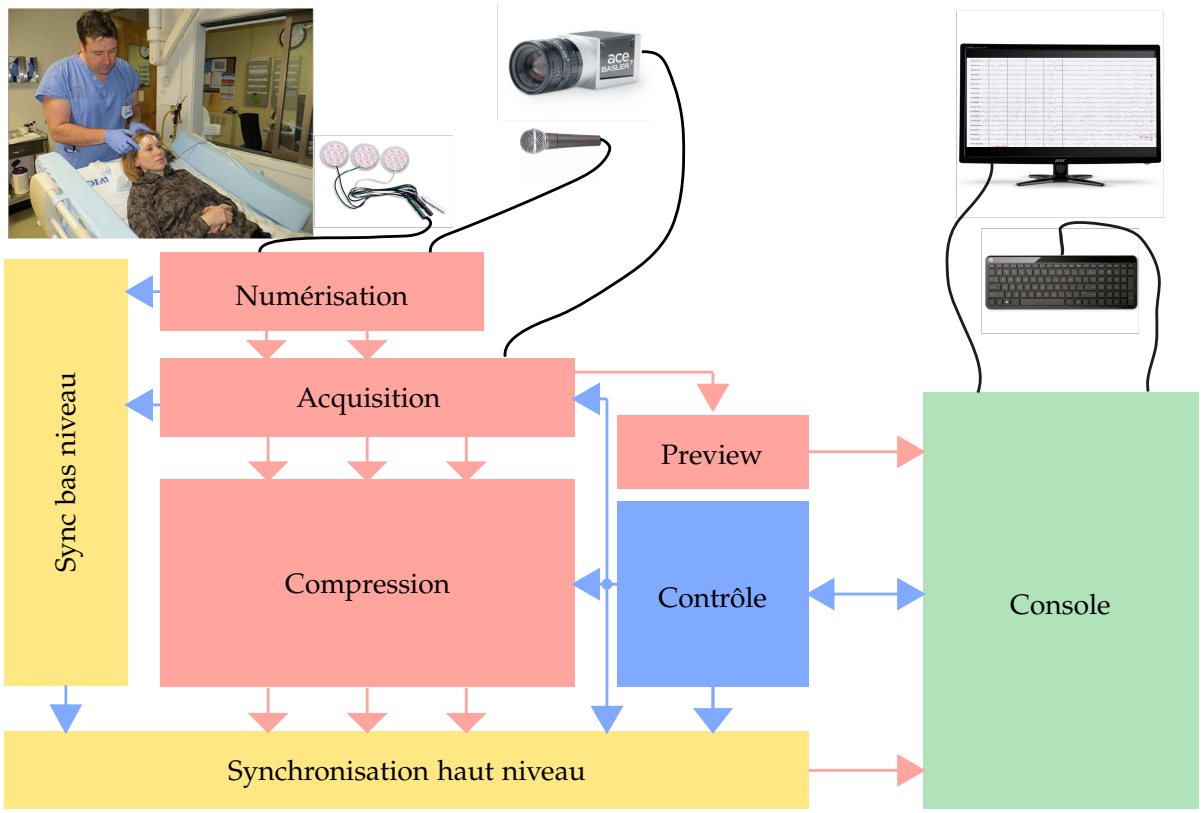



**Figure 43.** Schéma bloc du dispositif d'acquisition et de synchronisation d'examen EEG

Depuis ce schéma bloc, nous pouvons dégager les interfaces essentielles au développement de notre dispositif que nous allons détailler. En premier lieu, nous décrirons l'interface entre la caméra et l'acquisition. Ensuite, nous définissons l'interface entre le module de numérisation et l'acquisition. Pour finir, nous décrirons la liaison entre le dispositif et la console de prévisualisation.

## 1.2 Interface caméra

Pour choisir l'interface physique de la caméra, nous avons dû faire un comparatif des technologies disponibles sur les caméras du commerce. Trois grands standards sortent du lôt : *GigE*, *USB2.0*, *USB3.0* et *CameraLink*. Nous avons comparé les standards sur les critères de bande passante et de fiabilité de déclenchement. Pour rappel, l'interface choisie doit pouvoir transporter des séquences vidéo de cent images par secondes ce qui représente 200 MB/sec au format Bayer. Elle doit également pouvoir transporter un trigger pour acheminer le déclenchement d'acquisition d'une image.

Le tableau 6 résume l'analyse des trois standards.Les interfaces *GigE* et *USB2.0* ne permettent pas de transporter le flux de données que nous voulons acquérir. De plus, la procédure de déclenchement est réalisée par message sur les protocoles respectifs ce qui empêche de garantir la synchronisation de l'acquisition. L'*USB3.0* pourrait être un bon candidat cependant, la méthode de déclenchement est virtualisée sur le protocole USB ce qui rend le déclenchement indéterministe. Le tableau fait ressortir *CameraLink* comme

**Tableau 6.** Tableau comparatif des interfaces caméras du commerce

| Standard | Bande passante | Déclencheur |
|---|---|---|
| GigE | 100 MB/s | Par commande |
| USB2.0 | 60 MB/s | Par commande |
| USB3.0 | 400 MB/s | Direct virtualisé |
| *CameraLink* | 680 MB/s | Direct |

l'interface la plus adaptée pour notre utilisation, aussi bien pour acheminer la séquence vidéo de la caméra vers notre boîtier que pour être pilotée avec un déclencheur.

*CameraLink* est un standard introduit dans les années 2000 pour répondre à une problématique d'interconnexion entre les caméras industrielles et les *frame grabbers*. Pour information un *frame grabber* comme son nom l'indique est un composant récupérant des images d'une caméra. Le standard défini le câble, le connecteur ainsi que les signaux véhiculés. La connexion est réalisée sur un câble par 4 liens de 7 bits ce qui équivaut à 28 bits. 24 bits sont utilisés pour les valeurs des pixels, 3 bits pour des signaux de synchronisation verticale, horizontale, et par pixel et 1 bit pour les applications spécifiques. En plus de ces signaux, est disponible une liaison série permettant la configuration de la caméra. Le bus fonctionne à une fréquence pouvant aller jusqu'à 85 MHz. Le standard propose une configuration *Base*, *Medium* et *Full*, définissant le débit entre la caméra et le *frame grabber*. En configuration *Base*, qui est établie avec un câble *CameraLink*, la bande passante disponible est de 85 MHz $\times$ 24 bits $=$ 2040 Mbits/sec $=$ 255 MO/sec. En configuration *Medium* et *Full*, nous passons respectivement à 510 MO/sec et 680 MO/sec. Cependant, les deux dernières configurations nécessitent l'utilisation de deux câbles entre la caméra et le *frame grabber*. Dans notre cas, la configuration *Base* nous suffit pour acquérir notre flux vidéo.

## 1.3 Interface numérisation

Pour commencer, il n'y a pas d'interface standard entre une carte d'acquisition de signaux physiologique et un autre équipement car les équipements d'examen EEG sont vendus en solution complète avec un autre PC, donc les constructeurs utilisent chacun leur solution interne. Nous avons donc choisi de développer en collaboration avec ETIS une nouvelle interface pour récupérer les signaux physiologiques et maintenir la synchronisation. Dans la manière dont le projet Smart-EEG s'est déroulé et les spécificités de mise en œuvre, l'interface de numérisation est découpée en deux : une interface physique qui se caractérise par un câble et une interface logique se trouvant entre deux IP présentes sur le FPGA. Nous détaillons dans la suite les deux interfaces l'une après l'autre.

### Interface physique

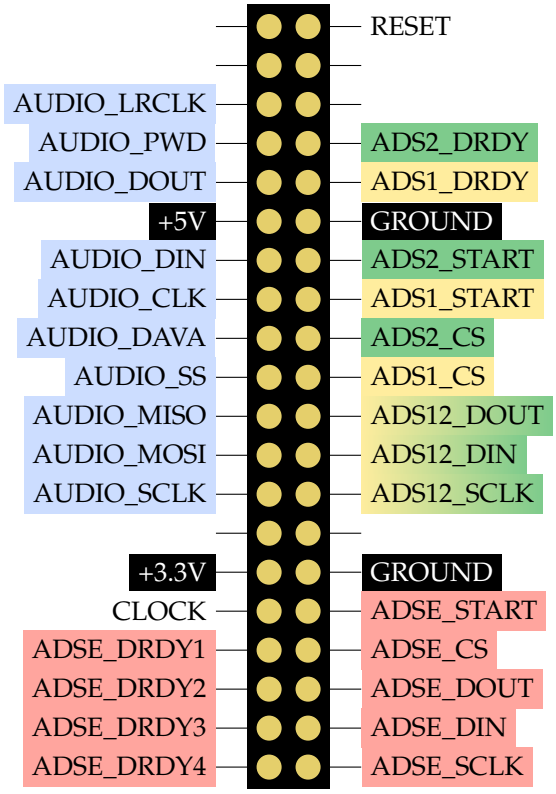



**Figure 44.** Interface physique entre la numérisation et l'acquisition

L'interface physique entre la numérisation et l'acquisition se base sur une nappe IDE de 40 pistes. Sur ces 40 pistes, 4 sont utilisées pour transporter le courant de fonctionnement de l'acquisition vers la numérisation et 30 pour les signaux de contrôle connectés aux composants d'acquisition (TI 1299, TI 1298 et TI TLV320AIC26). La figure 44 illustre le connecteur présent sur la carte de numérisation. Les signaux sont séparés en six sous-groupes :

- Les signaux de commande (en noir sur blanc) d'horloge et de réinitialisation sont reçus par tous les composants.
- Les signaux d'alimentations (en blanc sur noir). Ces signaux ne sont pas utilisés par les composants mais par la barrière d'isolation présente entre les composants et le connecteur.
- Les signaux bleus sont connectés au composant audio. Dans ces signaux, il y a deux interfaces une SPI et une I2S. La première est utilisée pour configurer le composant et la deuxième pour récupérer les échantillons du codec. Dans ces signaux, nous pouvons également remarquer le signal AUDIO_LRCLK qui nous sert de signal de déclenchement pour la synchronisation.
- Les signaux rouges sont connectés au ADS 1299 acquièrent l'EEG. Il y a quatre composants connectés en daisy-chain. Nous les configurons tous en même temps par l'interface SPI. Pour la synchronisation, nous avons la possibilité d'obtenir les signaux de déclenchement DRDY de chaque ADS indépendamment.

- Les signaux verts et jaunes correspondent à l'ADS 1299 acquérant quatre signaux ECG, deux signaux EOG et deux signaux EMG et l'ADS 1298 acquérant huit signaux ECG. Pour configurer les deux ADS, le port SPI est partagé. Pour définir avec quel composant nous désirons communiquer, nous utilisons le signal CS du composant. Tout comme pour les ADS d'EEG, nous avons les deux signaux de déclenchement DRDY pour chaque ADS.

**Interface logique**

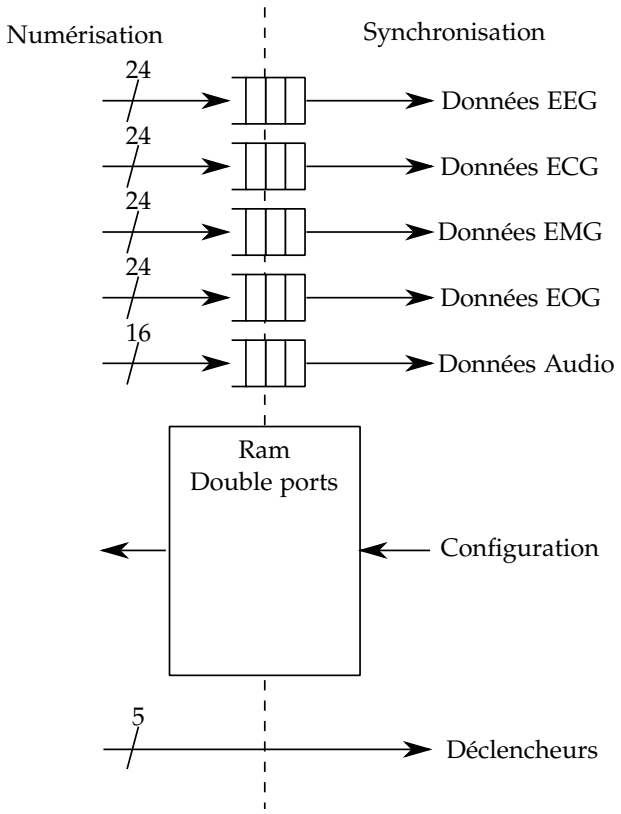



**Figure 45.** Schéma de l'interface logique entre la numérisation et la synchronisation.

La deuxième interface est une interface implémentée en logique FPGA permettant d'abstraire le contrôle des composants sous-jacent. Il est basé sur trois modules constituant l'interface. Cette interface est schématisée dans la figure 45. Le premier module est une fifo [1] bi-synchrone permettant le transport des données physiologiques et audio. Il y a une fifo par type de données transportant les échantillons bruts venant des composants. Les quatre fifo de signaux physiologiques sont de largeur 24 bits et la fifo d'audio de largeur 16 bits. Ces fifo sont assez profondes pour accueillir une période d'acquisition. C'est à dire dans le cas des signaux ECG, il y a douze canaux dont la fifo est dimensionnée avec une profondeur de 16. Le lecteur de ces fifo a donc une période

1. fifo : First In First Out, file

d'acquisition pour venir chercher les données dans la fifo au risque que les données ne soient écrasées ou perdues.

Le deuxième module est une mémoire double ports permettant de définir les paramètres de configuration des composants d'acquisition. La configuration des composants est principalement statique mais le gain des convertisseurs doit pouvoir être modifié pendant la phase de calibration de l'examen. Pour ce faire, chaque case de cette mémoire correspond à un gain d'un canal d'acquisition comme décrit dans le tableau 7.

**Tableau 7.** Adresses des valeurs de gain en fonction de la modalité

| Gain des canaux | Plage d'adresse |
|---|---|
| ECG | 0x00 - 0x0B |
| EOG | 0x0C - 0x0D |
| EMG | 0x0E - 0x0F |
| EEG | 0x10 - 0x2F |

Et le dernier module est destiné au passage des déclencheurs de synchronisation. Celui-ci se matérialise par un vecteur de signaux provenant des composants d'acquisition vers le module de synchronisation. Ce sont ces signaux qui sont utilisés pour la génération des timestamps. Il y a un déclencheur par modalité acquise c'est-à-dire cinq.

## 1.4 Interface console

Cette sous-section décrit le protocole entre le boîtier électronique et la console locale. Le boîtier est connecté avec le PC physiquement par un câble Ethernet et logiquement par des connexions TCP/IP. Il peut donc y avoir un réseau complexe entre le boîtier et le PC tant que le boîtier est adressable par IP. Nous avons préféré le protocole TCP au protocole UDP pour les garanties d'intégrité des données. Nous n'avons pas à nous soucier de savoir si les données envoyées sont bien reçues de l'autre côté car le protocole TCP le garantit. Le protocole de communication entre le boîtier et le PC repose sur trois connexions TCP. Toutes ces connexions sont établies du PC vers le boîtier.

- La connexion principale transporte les commandes du PC vers le boîtier. Toutes les commandes sont de l'initiative du PC et entraînent une réponse du boîtier. Les commandes englobent les requêtes d'authentification, de configuration du boîtier, de démarrage et d'arrêt de l'acquisition. Il y a également une commande pour connaître l'état courant du boîtier.
- La deuxième connexion transporte les données de prévisualisation du boîtier vers le PC. Sur cette connexion sont multiplexées les données physiologiques ainsi que les données vidéo sous-échantillonnées. Afin de reconnaître les données sur la connexion, un petit entête, contenant un identifiant de type de données, une longueur et le timestamp, est rajouté en début de chaque message.
- Pour finir, la dernière connexion permet au boîtier d'envoyer le fichier d'examen

en cours de production. Celui-ci est envoyé sans aucun protocole particulier au niveau applicatif.

La mise en place de la procédure d'acquisition commence par la connexion TCP principale établie par la console vers le boîtier sur le port 1234. À partir du moment où la connexion est établie, le boîtier est en attente de commande. La console peut demander une authentification du boîtier. Cette authentification demande le numéro de série du boîtier. Une fois qu'il a répondu à la console, le boîtier attend deux nouvelles connexions pour la prévisualisation et le fichier. La console peut dès ce moment se connecter sur ces deux connexions, cependant elle ne recevra des données que sur la connexion de prévisualisation. Au moment où la connexion est établie sur le canal de prévisualisation, le boîtier commence à envoyer les données récupérées par le boîtier d'acquisition. Le boîtier reste en parallèle en attente de commande sur la connexion principale. C'est notamment dans cette période que toute la configuration du boîtier est réalisée (Gain de l'acquisition, Région d'intérêt, etc.). Une fois que la configuration est terminée, le praticien peut décider de commencer l'acquisition. La console envoie alors une commande de start au boîtier. Le boîtier entame donc la génération du fichier examen à partir de ce moment et l'envoie sur la connexion du fichier. Les données de la prévisualisation sont toujours envoyées en parallèle. Quand l'examen est terminé, la console envoie une commande de stop au boîtier qui entraîne la fermeture instantanée de la connexion de prévisualisation. Les données, en cours de traitement par le boîtier pour être compressées et encapsulées dans MKV, continuent à être envoyées sur la connexion du fichier. Quand le fichier est fini, la connexion du fichier est fermée par le boîtier. La console peut terminer la connexion avec le boîtier en se déconnectant de la connexion principale.

Cette procédure est illustrée par le diagramme d'état de la figure 46. Les états de couleurs verts sont traités par la connexion principale, les rouges par la connexion de prévisualisation et les bleus par la connexion de fichier. Ce protocole, que nous avons défini lors des développements dans le cadre de cette thèse, a permis la réalisation de la connexion entre le boîtier et la console de prévisualisation qui est elle à la charge d'un partenaire du projet.

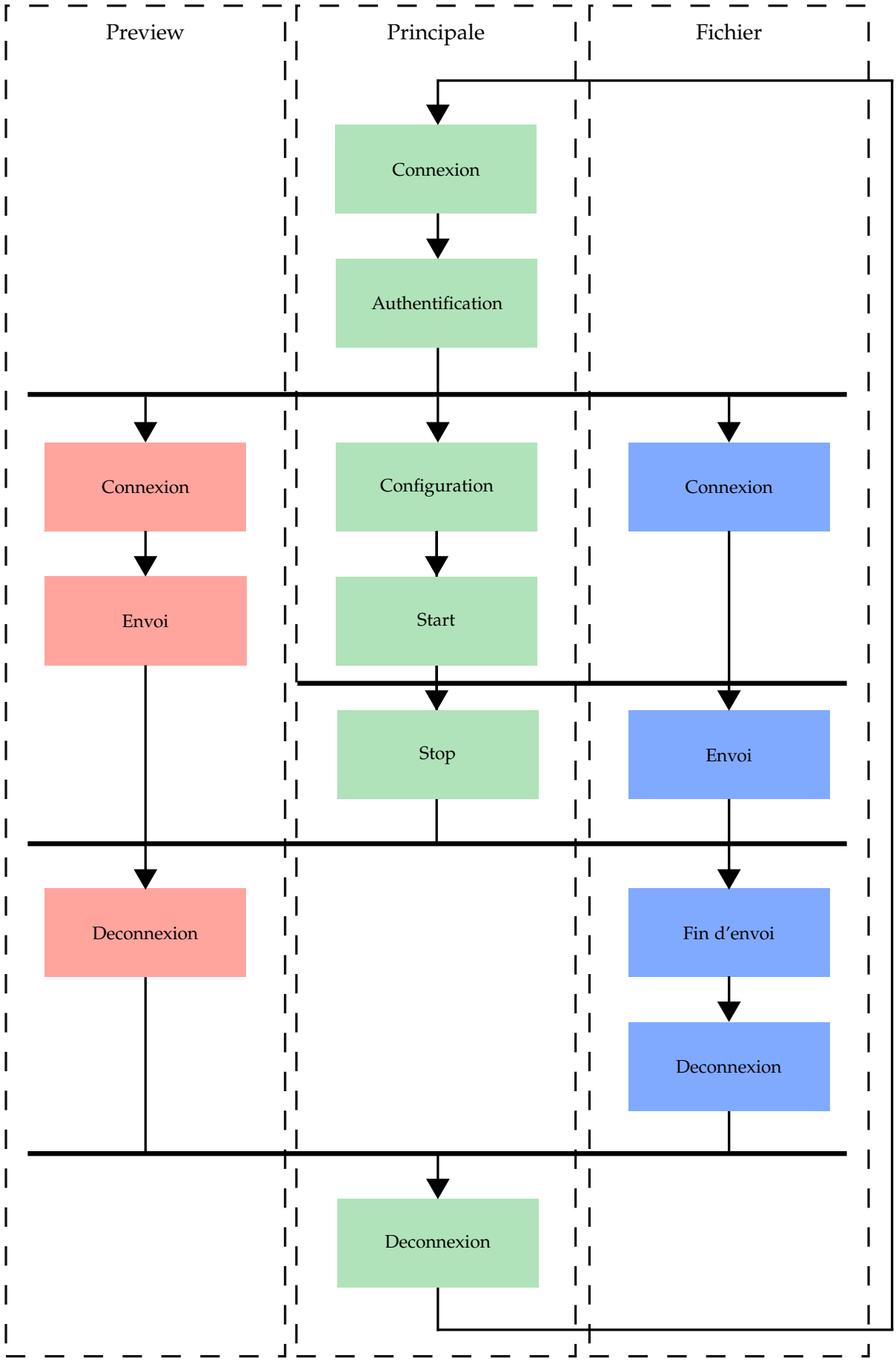



**Figure 46.** Diagramme d'état de l'interface Ethernet entre le boîtier et la console.

# 2 Implémentation logicielle

L'implémentation logicielle est réalisée en utilisant une partie FPGA et une partie sur processeur. Les traitements réalisés sur la partie FPGA sont ceux qui nécessitent des entrées/sorties spécifiques (*CameraLink*, IDE-40) et qui demandent des contraintes de temps réel (Gestion de la synchronisation et tampon d'acquisition).

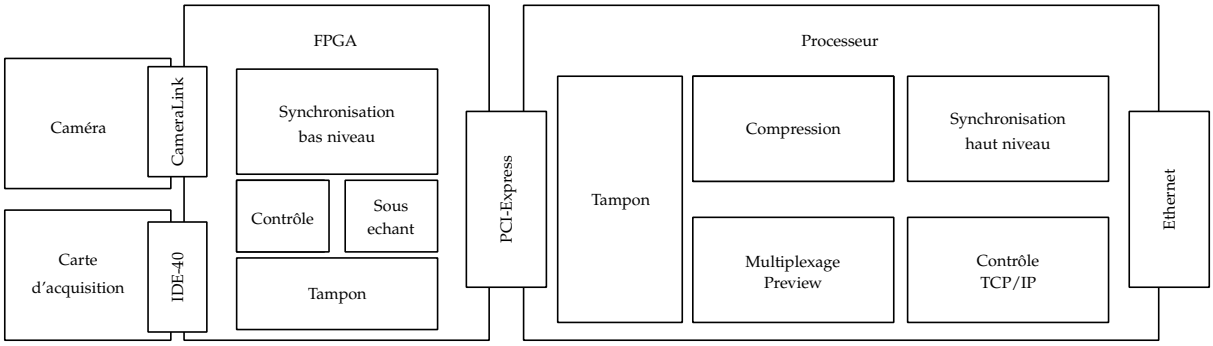



**Figure 47.** Schéma bloc du implémentation logicielle.

L'implémentation est schématisée dans le schéma bloc de la figure 47. Elle est intégrée dans un boîtier de PC contenant le FPGA et le processeur. Nous utilisons la carte Terasic DE4 comportant le FPGA Stratix IV. Cette carte est branchée par le PCI-Express à la carte mère du PC. Il y est également branché à la caméra via la carte d'adaptation Terasic HSMC-CLR permettant de lire les signaux *CameraLink* dans la cible FPGA. Le FPGA est aussi connecté à la carte d'acquisition développée par le laboratoire ETIS par le biais du cable IDE-40.

Afin de transmettre les données du FPGA, nous avons utilisé le framework *Riffa* [Jacobsen et al., 2015] dans sa version 2.1. Ce framework nous a permis de faire communiquer des données entre la carte FPGA et les applications sur le processeur. Le framework génère du côté matériel des interfaces type fifo et du côté logiciel des interfaces de type fichier. De cette façon, du côté matériel l'écriture et la lecture de données se réalisent comme dans une simple fifo et du côté logiciel il se réalise en effectuant des appels de fonction *read* et *write* comme sur un fichier. Toutes les couches PCI-Express, Driver de l'OS et appel système sont abstraits par le framework.

Pouvant échanger des données du FPGA au PC, les données sont acquises par des IP sur le FPGA le tout en enregistrant les timestamps et d'envoyer l'ensemble au PC. Le problème de cette solution est l'impossibilité de garantir la disponibilité du support PCI-Express lorsque nécessaire. En effet, l'arbitrage du réseau PCI-Express est réalisé par le maître qui est le processeur. Il peut à n'importe quel moment récupérer le réseau même en cours de transaction ce qui a pour conséquence de retarder la fin de la transaction. Il est donc impossible d'envoyer en flux tendu les données de

l'acquisition sans faire une mise en tampon sous peine d'avoir des pertes de données. Pour pallier ce problème, nous avons ajouté, dans le FPGA, un module réalisant un tampon mémoire. De plus, pour ne pas surcharger le processeur, nous avons réalisé des IP de sous-échantillonage.

Une fois les données dans la mémoire du PC, nous réalisons une deuxième mise en tampon avant que les données soient compressées. En effet, nous ne pouvons pas garantir que les données seront compressées en flux tendu donc nous devons pouvoir enregistrer les données en attendant. Pour nous prévenir du pire cas, nous voulons une solution qui permet de mettre en tampon l'intégralité de l'examen. Pour servir de tampon à un examen brut, il faut une capacité de stockage conséquente, au moins 250 GO. Nous ne pouvons pas uniquement utiliser la mémoire vive de l'ordinateur. De plus, il faut utiliser un support de stockage capable d'écrire assez vite les données de l'examen venant du FPGA, au moins 200 MO/sec. À l'heure où nous avons commencé la réalisation logicielle du systèem, la seule solution technique présente sur le marché combinant la capacité et la bande passante nécessaires est basée sur l'utilisation des disques durs SSD *NVMe* connecté à un port *PCI-Express* de la carte mère du PC . Le SSD présent dans la manipulation est un *Intel 750 series* de 1,2 TO. Les données sont alors ensuite compressées et multiplexées dans le flux BIO-MKV pour être envoyées. De même, les données sous-échantillonées par le FPGA sont multiplexées et envoyées par Ethernet à la console de prévisualisation. Les commandes quand besoin sont transmises aux IP FPGA. Entre le FPGA et le PC, il y a 5 canaux virtuels mis en place pour transporter diverses données :

- Un canal de commande dont les messages proviennent du PC et vont vers le FPGA. Le FPGA répond à ces commandes.
- Un canal de vidéo brute pour envoyer le flux d'images et leur timestamp du FPGA vers le PC.
- Un canal de vidéo preview pour envoyer le flux d'images sous-échantilloné du FPGA vers le PC.
- Un canal de données physiologiques pour envoyer les échantillons de données ainsi que leur timestamp.
- Et pour finir, un canal d'audio pour envoyer les échantillons audio avec leur timestamp.

Dans les deux sous-sections suivantes, nous rentrons dans le détail de la partie FPGA puis de la partie logicielle.

## 2.1 Partie FPGA

L'implémentation de la partie FPGA a été développée en utilisant la suite d'outils d'Altera, notamment l'intégrateur de composant QSYS en utilisant principalement le langage Verilog. La majorité des IP de cette partie sont réalisés dans QSYS avec un

fonctionnement flot de données en utilisant des interfaces de streaming/fifo pour lire et écrire des données aux autres IP. Dans les explications suivantes, nous omettrons de parler des adaptations de format que nous avons dû réaliser entre certains blocs. Par exemple, les pixels de la caméra sont lus trois par trois et font chacun 8 bits ce qui nous donne par cycle 24 bits. Par la suite, les pixels doivent être organisés en paquet de 64 bits pour être écrits en mémoire. Ces adaptations sont omises pour des raisons de simplification mais consomment évidemment des ressources logiques et mémoires.

Le haut niveau de la partie FPGA est constitué de trois pipelines, un pour chaque modalité. Les trois flots de données sont traités en parallèle par des IP distinctes. Ces pipelines ont pour objectifs de récupérer les données des composants de bas niveau, de rassembler ces données avec leur timestamp, de les mettre en tampon et de les transmettre au canal respectif de *Riffa* qui s'occupera de le transmettre à la partie logicielle. Le pipeline vidéo à la particularité de gérer deux flux vidéo, un flux brut Bayer destiné à être compressé dans la partie logicielle et un flux preview au format RGB qui a été sous-échantillonné. De ce fait, il y a quatre canaux de transmission PCI-Express instanciés dans l'IP de *Riffa* pour l'envoi de données vers le PC. Dans la figure 48, nous pouvons voir les trois pipelines en question ainsi que les différents canaux de *Riffa.* En plus, de ces IP nous pouvons voir l'utilisation de la mémoire DDR externe utilisée par les pipelines pour réaliser des tampons de haute capacité. Nous avons utilisé le contrôleur DDR d'Altera pour réaliser l'interface entre nos IP et la DDR externe. Pour finir, nous pouvons voir le module de synchronisation connecté recevant les déclencheurs et envoyant par des fifos les timestamps aux différents pipelines. Le tout est contrôlé par les commandes du PC reçus depuis le canal de commande. Un nios, pas représenté dans la figure, est également instancié pour initialiser la caméra et pour réaliser l'acquisition bas niveau physiologique et audio.

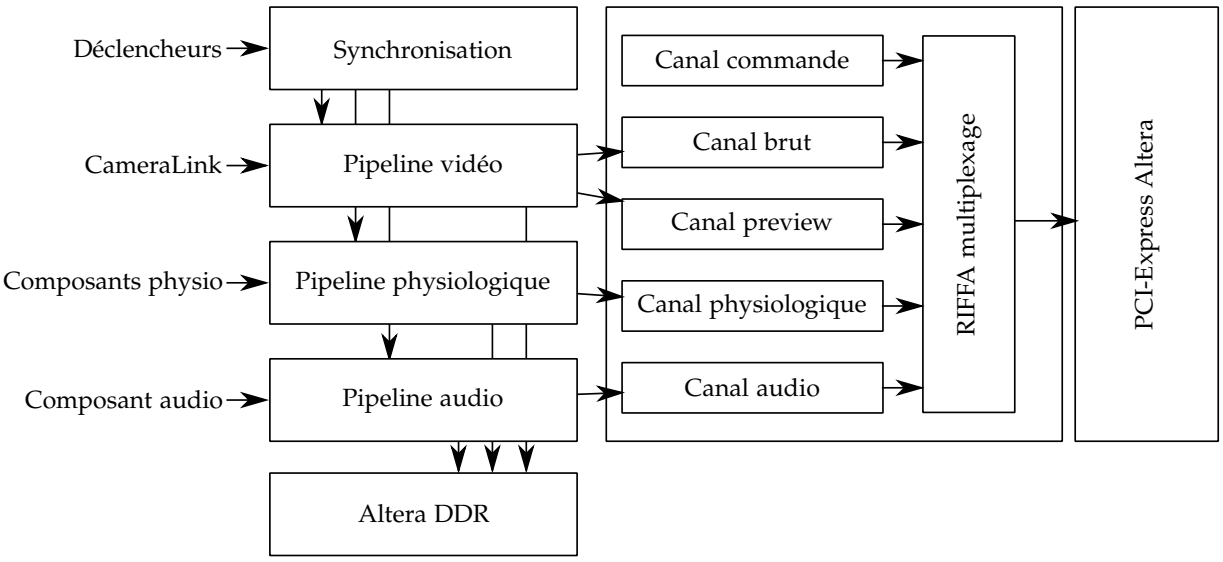



**Figure 48.** Schéma bloc de l'organisation interne des modules FPGA.

Dans la suite de la section, nous présentons les différentes parties du système : le contrôle, la synchronisation bas niveau, la partie physiologique et audio et pour finir la partie vidéo.

**Contrôle**

Le module de contrôle est l'élément permettant aux autres modules de connaître l'état de l'acquisition. Il lit les messages venant du PC, les décode, effectue les actions nécessaires puis répond au PC. Il a en entrée la fifo de commande venant du module *Riffa* et pour sortie deux signaux un bit indiquant l'état courant de l'équipement, des signaux allant à la mémoire de configuration des *ADS* et la fifo de réponse aux commandes vers *Riffa*. Les deux signaux un bit permettent aux autres modules de savoir quoi faire. Seuls trois des quatre combinaisons sont utilisées pour l'état du dispositif : *IDLE*, *CALIBRATION* et *ACQUISITION*.

Le module est basé sur une machine à état pouvant être visualisée dans la figure 49. Au reset, nous démarrons dans l'état *IDLE*. À la réception, d'un message du PC, nous basculons dans l'état *RX* afin de gérer la transaction avec le PCI-E et décoder le message reçu. Nous pouvons grâce à cela, basculer dans un des états de gestion de commande. Si le message ne correspond à aucune commande, nous basculons dans l'état *ERROR*. L'état *ACQ_START*, *CALIBRATION* et *ACQ_STOP* permet de changer l'état du système, c'est-à-dire de respectivement passer dans l'état *ACQUISITION*, *CALIBRATION* et *IDLE*. L'état *READ_CONF* permet de lire une case mémoire de la configuration des ADS. Pour pouvoir lire l'intégralité de la configuration actuelle, il faudra que le PC fasse plusieurs transactions. Le dernier état *CONF_1* est suivi d'un deuxième état *CONF_2* qui permet de réaliser l'écriture d'une configuration dans la mémoire des ADS. Il y a besoin de deux états car la procédure nécessite d'arrêter l'acquisition avant d'écrire la configuration. Quelle que soit la commande exécutée, la machine se retrouve ensuite dans l'état *TX* qui renvoie une réponse au PC pour indiquer si la commande s'est bien déroulée.

**Synchronisation**

Le module de synchronisation a deux finalités : maintenir la valeur du temps courant et acquérir les temps d'acquisition pour les différentes modalités. Pour ce faire, le module de synchronisation utilise deux petits sous-modules, le premier est le compteur et le deuxième est un générateur de timestamp.

Le module de compteur contient simplement un registre de 64 bits contenant la valeur du temps au cycle courant. Il a en entrée une horloge qui sert d'horloge pour l'incrémentation du compteur et d'un reset pour remettre à zéro le compteur. Il n'y a qu'une sortie qui est les 64 bits du compteur.

Le module GenericTS, générant les timestamps, reçoit comme entrée un déclencheur

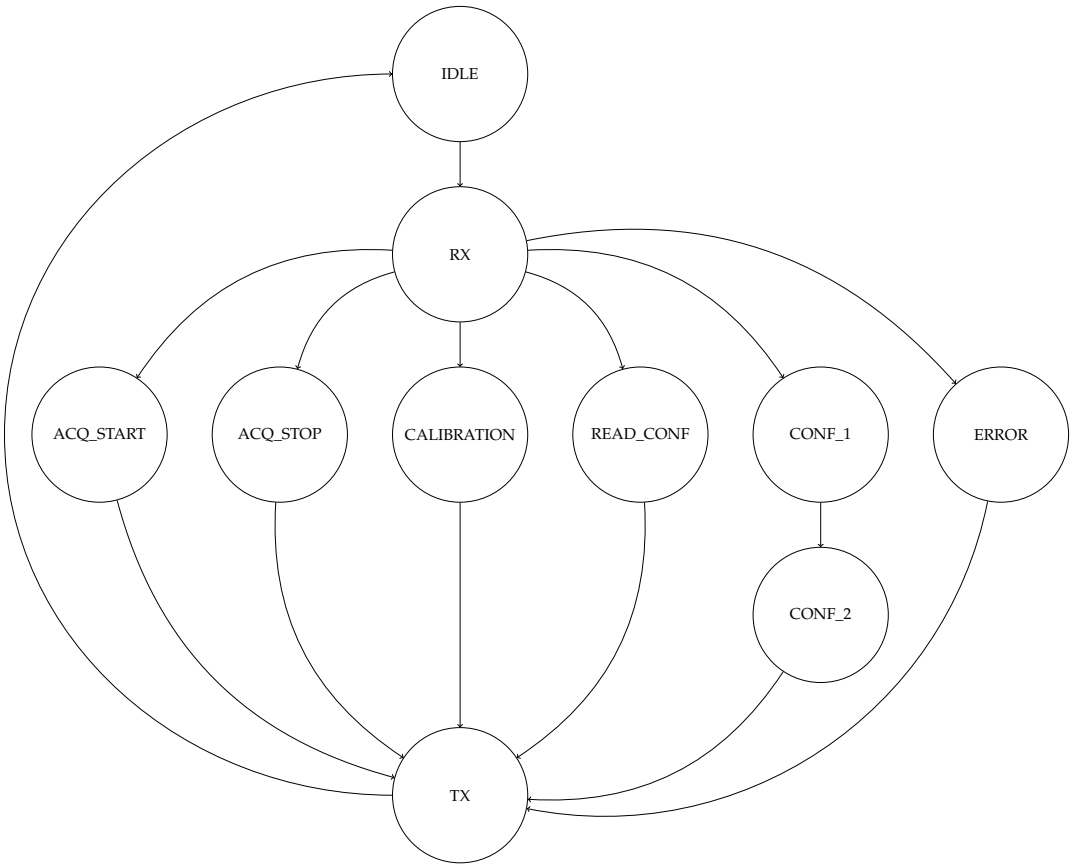



**Figure 49.** Machine à état du module de contrôle.

et le temps courant et à pour sortie une fifo. Il est simplement constitué d'un détecteur de front montant pour le déclencheur et d'une logique pour écrire le temps courant dans la fifo de sortie quand le détecteur est vrai. Un module GenericTS est instancié par modalité.

La figure 50 illustre le module de synchronisation au complet, avec à gauche le compteur du temps courant et à droite les six instances du module GenericTS.

**Pipeline physiologique et audio**

Le pipeline physiologique et le pipeline audio sont très similaires, de ce faire nous détaillons le pipeline physiologique tout sachant que le pipeline audio comporte les mêmes modules pour fonctionner. Le pipeline physiologique commence par faire la conversion entre l'interface physique et l'interface logique. Cette conversion est réalisée en utilisant un processeur softcore Nios qui gère l'interface SPI avec les six ADS. Au démarrage, le processeur programme les ADS dans une configuration par défaut présente dans la mémoire de configuration. Si la configuration change, alors le processeur redémarre et relance la configuration avec la nouvelle configuration. Pour récupérer les données, il reçoit une interruption des composants à chaque acquisition lui permettant de savoir que des données sont disponibles. Dans la routine d'interruption,

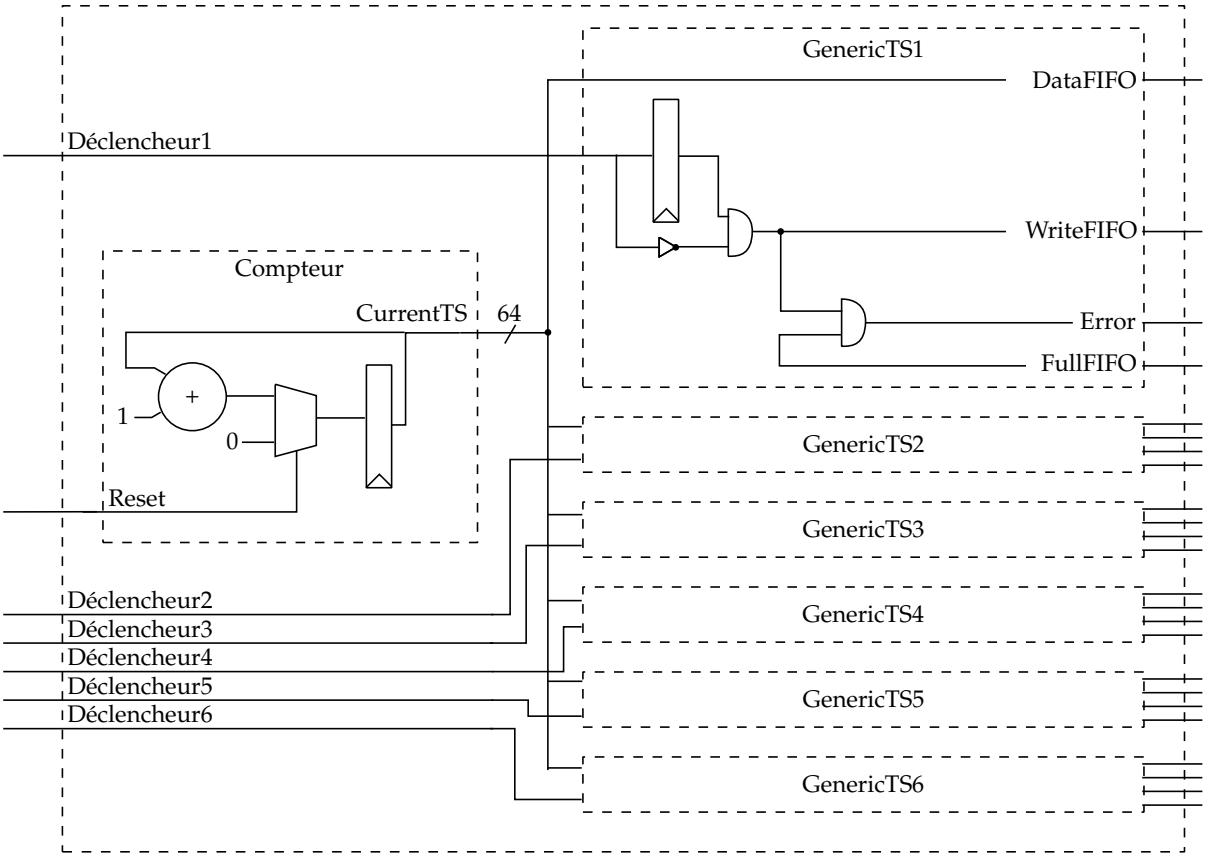



**Figure 50.** Architecture interne de la synchronisation bas niveau.

le processeur récupère les données une par une de chaque ADS et les écrit dans les fifos de la deuxième interface.

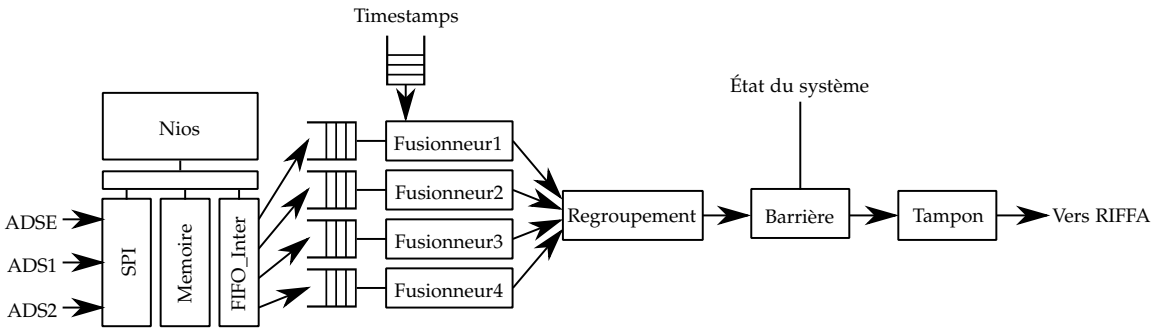



**Figure 51.** Schéma bloc du pipeline d'acquisition des signaux physiologiques.

Une fois passée l'interface logique, les données sont traitées en mode flot de données. Ce flot de données peut être vu dans la figure 51. Dans un premier temps, les différents types de données (EEG, ECG, EMG, EOG) sont chacun regroupés avec leur timestamps venant du module de synchronisation. Le module lit le timestamp puis les données venant de la fifo de l'interface logique. Les timestamps et la donnée ne sont pas mis en tampon mais sont écrits directement dans un port de sortie vers le module de regroupement. Le module de regroupement combine les quatre types de données en un seul flux de sortie. Il est constitué d'une simple machine à état qui lit cycliquement les

entrées et les écrit vers un port de sortie unique. De même que le module précédent, le flux ne met pas en tampon les données mais les transmet directement de l'entrée à la sortie. Le module suivant barrière a pour rôle de filtrer les paquets de données non voulus. Pour ce faire, il a pour entrée l'état du boîtier actuel. Quand le boîtier est en *IDLE*, toutes les données d'entrée sont jetées. À l'inverse, quand le boîtier est dans l'état *CALIBRATION* ou *ACQUISITION*, les données sont transmises au module suivant. Le dernier module du Pipeline, sert à faire la mise en tampon de paquet de données. Il est constitué de deux mémoires double ports sur puce. En alternance, une des mémoires va être écrite avec les données provenant de l'entrée puis être lue pour que son contenu soit transmis à la sortie du pipeline c'est-à-dire au canal physiologique de *Riffa*. Notre expérience a montré que ce système de double buffering permet de ne perdre aucune donnée quand le système acquiert à 2 kHz ce qui est notre fréquence maximale d'acquisition.

**Pipeline vidéo**

Le pipeline vidéo est sensiblement identique du point de vue conceptuel aux deux pipelines présents. Les trois différences sont l'interface d'acquisition (*CameraLink*) qui nécessite un module particulier pour fonctionner, le découpage en deux flux distincts pour réaliser un flux preview et un brut et la taille des transactions de la modalité. En effet, chaque transaction implique le déplacement de 2 MO.

La caméra est configurée par un port série disponible sur l'interface *CameraLink*. Nous utilisons le Nios présent dans le pipeline physiologique pour faire la configuration de la caméra. Ceci n'impacte pas le pipeline physiologique car la configuration n'est réalisée qu'une fois au reset.

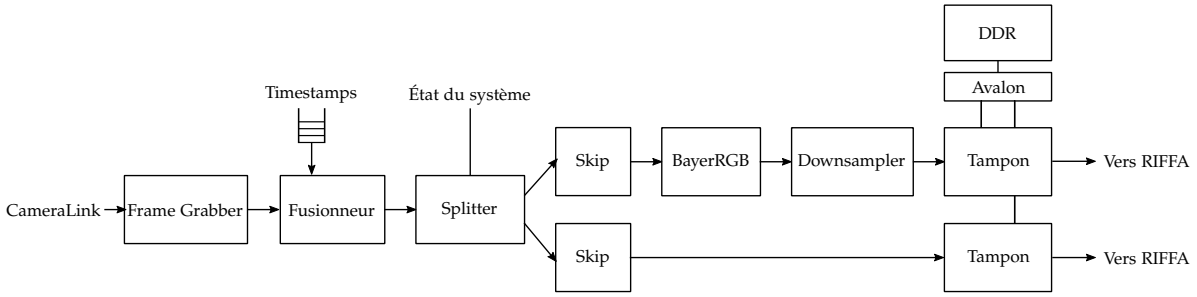



**Figure 52.** Schéma bloc du pipeline d'acquisition de la vidéo.

La figure 52 expose le pipeline utilisé pour faire l'acquisition vidéo. Le premier module sert à acquérir une donnée depuis le flux *CameraLink* vers un format flot de données/fifo. Le module suivant, Fusionneur, regroupe un timestamp avec l'image associée. À ce stade, le flux est constitué du timestamp sur 64 bits puis des pixels un par un. Le module *Splitter* va dupliquer le flot de données dans deux branches du pipeline. Il a aussi une fonction de barrière en fonction de l'état du système comme pour

le pipeline physiologique. Quand le système est en *IDLE*, le *splitter* jette les images, quand il est en *CALIBRATION*, il recopie le flux dans la branche *preview* et quand il est en ACQUISITION, il recopie le flux dans les deux branches. La branche brute est constituée de deux modules : le module *Skip* et le module Tampon. Le module Skip sert à détecter qu'une image a été bloquée quelque part dans le pipeline en amont et que l'image va être corrompue. Pour ne pas se retrouver avec un flux décalé, le module Skip s'assure que le pipeline est toujours libre pour écrire les nouveaux pixels. Quand le pipeline est bloqué, il marque l'image courante avec une erreur (métadonnée du flux fifo). Dans ce cas, le module complète l'image avec des pixels fictifs pour que le traitement puisse continuer. Il se resynchronise au début d'une image et reprend la transmission normalement. Dans la branche preview, deux modules ajoutés. Ces deux modules sont réalisés dans le FPGA pour éviter de surcharger trop le processeur de la partie logicielle. Le premier des deux modules réalise une conversion Bayer vers RGB pour que l'image soit directement utilisable par la console de prévisualisation. Le second module ajouté réalise un sous-échantillonage du flux vidéo. Il est nécessaire de réduire le débit de vidéo brute pour pouvoir l'envoyer sur la liaison Ethernet vers la console. Le module réalise un sous-échantillonage spatial en réduisant par seize le nombre de pixels (division par quatre en hauteur et en largeur) et un sous-échantillonage temporelle en réduisant par dix le nombre d'images du flux.

Pour finir, le module Tampon fonctionne comme une fifo à image d'un point de vue de l'extérieur. Cependant, la taille d'une image nous empêche de les stocker dans une fifo sur la puce. Pour pallier ce problème, nous proposons de stocker les images dans la mémoire externe de la carte de développement. Nous avons donc développé une fifo exploitant la mémoire DDR comme stockage. Le module est découpé en deux sous-blocs : un écrivain et un lecteur. L'écrivain va écrire les images dans la mémoire vive et le lecteur va relire ces images. L'écrivain décide où écrire les images reçues dans la mémoire. Nous allouons une zone mémoire à l'écrivain de manière statique lui permettant de stocker N images. Il conserve une image de la disponibilité de cette mémoire dans un tableau de registre 1 bit. Quand la case en mémoire est libre, la case associée dans le tableau est 0, quand elle est occupée, la case est 1. L'écrivain utilise la zone mémoire comme un tampon circulaire. Une fois qu'une image est écrite entièrement en mémoire par l'écrivain, il informe le lecteur qu'une image a été écrite en lui donnant l'adresse du début de l'image par une fifo. Celle-ci permet d'envoyer N adresses. Le deuxième sous-blocs à la réception d'une adresse, commence à lire l'image et la transmettre au canal de PCI-Express. Pour réaliser les accès mémoires, nous utilisons un contrôleur DMA Avalon capable de faire des rafales. Le second rôle du module est de retrouver les images marquées d'une erreur et de les supprimer. Pour ce faire, l'écrivain n'enverra jamais l'adresse de l'image corrompue au lecteur. Grâce à cela l'image en question ne sera jamais lue et sera éventuellement écrasée par une autre. Le sous-bloc lecteur notifie les images lues par une fifo de retour appelé

complété. L'écrivain grâce à cette information peut alors mettre à jour sont tableau de cases disponibles. Les deux sous-blocs ont une partie de contrôle gérée par une machine à état. La figure 53 schématise le fonctionnement de ce module.

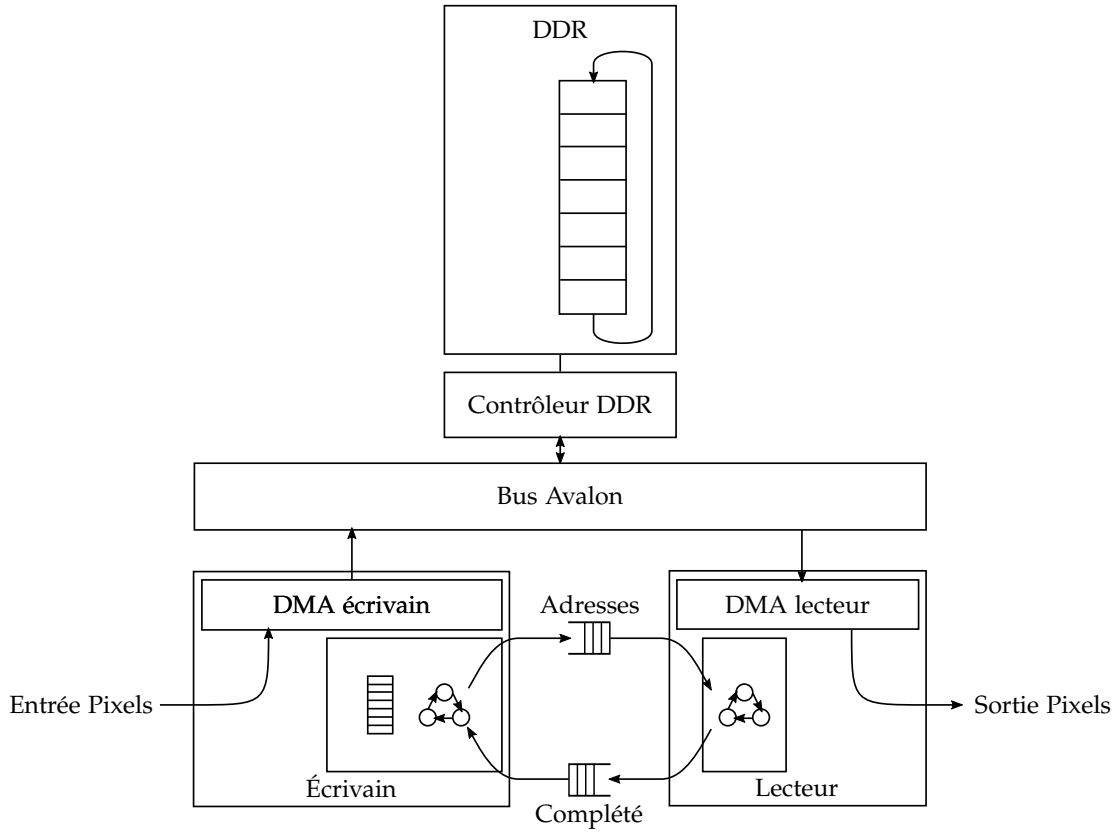



**Figure 53.** Principe de fonctionnement des deux modules DMA émulant une fifo pour les images.

Dans cette sous-section, nous avons détaillé le fonctionnement de l'acquisition et de la synchronisation bas niveau dans le module FPGA. La sous-section expliquera le fonctionnement de la partie logicielle utilisant les données transmises dans cette sous-section.

## 2.2 Partie logicielle

Pour rappel, le rôle principal de la partie logicielle est de compresser les données acquises et de les intégrer dans un format multimédia. De plus, elle doit répondre aux requêtes provenant de la console et lui envoyer les données brutes pour qu'elle les affiche. Le développement de cette partie est intégré dans une application *multi-threadée* écrite en C++11. Elle a un fonctionnement proche d'un serveur répondant aux requêtes de la console. L'aspect concurrent de l'application est indispensable pour le bon fonctionnement du prototype. En effet, l'application doit être capable de rester

en attente sur plusieurs évènements provenant soit de la console via une connexion TCP/IP soit du FPGA par l'envoie de données par les canaux *Riffa*. De ce fait pour ne pas faire d'attente active, nous avons choisi de faire une application avec plusieurs fils d'exécution. Ceci à aussi l'intérêt de pouvoir profiter du parallélisme des architectures actuelles. Nous avons préféré utiliser des *threads* plutôt que des processus afin alléger les communications entre les fils d'exécution. Les threads dialoguent avec deux moyens de communication, soit par variable globale soit par des fifos logicielles implémentées par nos soins. Les fifos sont simple écrivain et simple lecteur de ce fait nous avons de simple mutex pour les accès concurrents à celle-ci.

Le fonctionnement de l'application est représenté dans la figure 54. Chaque boite présente sur ce diagramme représente un fil d'exécution. Les fils d'exécution rouges sont les deux threads lancés au démarrage du serveur. Le thread *MAIN* est le thread principal du programme. Il s'occupe lors du démarrage d'initialiser toutes les classes partagées par plusieurs threads. Une fois démarré, il rentre dans une boucle infinie en attente de commande de la console sur la connexion *MAIN*. À chaque commande reçue, le thread la traite et répond à la console. Certaines de ces commandes peuvent impliquer envoyer des commandes au FPGA via le canal de commande. Ce thread est le seul à envoyer des commandes au FPGA. Le thread *Affichage* est optionnel et sert à visualiser l'activité du serveur et des statistiques. Les statistiques sont récupérées en lisant des variables globales gérées par le thread *MAIN*.

Lorsque le thread *MAIN* s'authentifie avec la console, il va démarrer le groupe de threads verts. Dans ces quatre threads, les threads *preview_grabber* et *physio_grabber* servent à récupérer les données brutes utiles à la génération de flux de prévisualisation pour la console. Les données sont en attente de données du FPGA. Quand des données sont présentes, le thread se débloque et les envois aux threads suivants via deux fifos différentes. Le thread *physio_grabber* doit faire suivre les données à la console de prévisualisation mais aussi au thread effectuant le fichier BIO-MKV. Pour ce dernier, le thread lit l'état du système via une variable globale du thread *MAIN* si elle doit transmettre les données physiologiques pour les inclure dans un fichier examen. Dans le groupe de quatre threads, les deux autres threads se chargent de lire les fifos et de formater les données afin les envoyer via la connexion preview à la console de prévisualisation.

Après avoir réalisé l'étape d'authentification, le boîtier est prêt à configurer le boîtier. Cette opération est entièrement réalisée dans le thread *MAIN*. Celui-ci récupère les commandes venant de la connexion TCP/IP et les transfère au FPGA. La console peut lancer une commande de *START*, pour lancer l'enregistrement d'un fichier examen. À ce moment, le thread *MAIN* démarre le groupe de threads bleus puis envoie une commande de démarrage au FPGA. Comme pour les threads preview, la récupération des données est réalisée par des threads grabber. Deux threads sont lancés pour

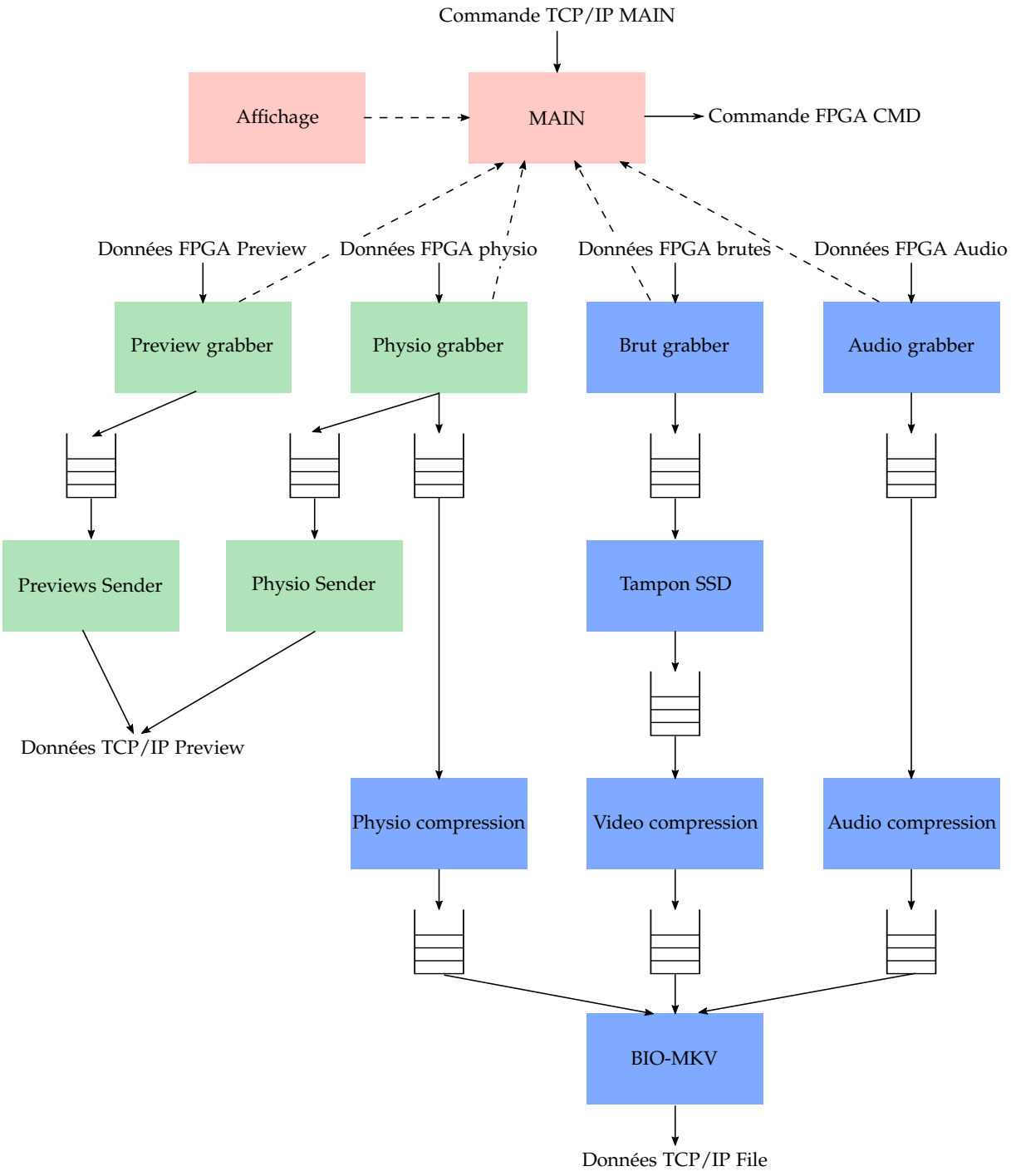



**Figure 54.** Schéma logique des fils d'exécution de la partie logicielle.

récupérer respectivement la séquence vidéo et le flux audio. Le thread récupérant les données physiologiques est déjà lancé pour effectuer le preview. Pour celui-ci, le changement d'état dans le thread *MAIN* suffit pour qu'il duplique les données pour le preview et le fichier examen. Pour chacun de ces threads, les données sont écrites dans des fifos pour être compressées par la suite. Dans le cas des données brutes physiologiques et audio, la quantité de mémoire allouée pour un examen est suffisamment petite pour que la fifo soit entièrement en mémoire vive (pour trente minutes respectivement environ 450 Mo et 160 Mo). Pour la séquence vidéo par contre, nous avons été obligés de développer un traitement spécial pour les stocker temporairement sur un SSD avant qu'elles soient compressées. Pour ce faire, un thread dédié lit les images récupérées par le grabber brut et les écrit sur le SSD. Le thread conserve le chemin vers l'image et son timestamp en mémoire et les transmet au thread de compression. C'est uniquement quand le thread de compression va avoir besoin de compresser l'image qu'il ira relire les données du disque SSD. Pour les deux autres modalités, les données sont directement compressées (ou formatées) en utilisant le compresseur dédié à la modalité (EDF+ pour le physiologique et Vorbis pour l'audio). Les données compressées sont encapsulées dans des paquets de FFmpeg qui sont écrits dans des fifos. Le dernier thread est chargé de lire ces paquets et d'effectuer la création du flux BIO-MKV en utilisant l'API de FFmpeg. Ce thread lit les paquets dans l'ordre des timestamps provenant des trois fifos. Une fois le flux BIO-MKV crée en mémoire, il est envoyé sur la connexion fichier pour la console.

Pour finir, la console envoie une commande de stop sur la connexion *MAIN*. Le thread *MAIN* transmet une commande de stop au FPGA, change son état en *STOPPING* et se bloque en attendant la fin des threads verts et blues. Les threads grabbers lisent le changement d'état, récupère les dernières données du FPGA puis envoient un message spécial dans leur fifo indiquant la fin de l'acquisition. Par cascade, les threads se ferment un après l'autre. Cependant, parceque nous utilisons des fifos, toute donnée pas encore traitée sera traitée avant le message d'arrêt. De ce fait, le fichier examen continue d'être compressé et généré pour la console de prévisualisation. Quand, toutes les données sont envoyées le dernier thread (qui est normalement le thread *BIO-MKV*) s'arrête. Le thread *MAIN* se débloque, ferme les différentes connexions et se remet dans un état *IDLE* en attente d'une nouvelle connexion.

L'application est structurée en classe gérant les différentes fonctionnalités. Nous ne détaillerons pas toutes les classes développées mais seulement de trois qui sont, à notre opinion, les plus centrales.

Premièrement, nous avons une classe *Fpga* servant à réaliser la communication entre l'application et la librairie *Riffa*. Son constructeur s'assure qu'un fpga contenant le bon nombre de canaux est connecté au PC. Elle contient les méthodes permettant aux différents threads de recevoir et envoyer des données au fpga. Toutes ces méthodes sont protégées par canal par des mutex pour empêcher deux threads de faire des

écritures ou lectures concurrentes sur le même canal (opérations qui sont interdites dans la spécification de *Riffa*). Des méthodes plus spécifiques comme le démarrage de l'acquisition sont également disponibles.

La deuxième classe est la classe utilisée par le thread *BIO-MKV* pour lire les trois flux de données compressés. Pour aider ce thread, nous avons implémenté une classe fifo_ordonne. Cette classe contient une unique méthode retirant le paquet le plus vieux d'une des trois fifos.

Dernièrement, une classe centrale de la communication entre threads est la fifo. Elle permet à un thread décrire des données et à un autre thread de lire les données. Comme chaque objet est utilisé par deux threads, il est protégé par un mutex à chaque opération. Le stockage des données est effectué en utilisant la classe vector de la librairie standard C++. Pour finir, la classe à un template permettant d'instancier une fifo d'objet quelconque. Ceci permet d'avoir qu'une classe fifo pour les différentes communications dans l'application. Nous avons implémenté les insertions et retraits de la fifo en utilisant la sémantique de déplacement de C++11 qui permet d'éliminer des allocations de données et copies inutiles.

**Profiling du codeur ROI-Waaves logiciel**

Nous proposons d'évaluer le point critique de l'implémentation logicielle qui est l'encodage vidéo utilisant ROI-Waaves. Nous avons lancé l'encodage d'une séquence d'images sans aucune autre application. La vitesse d'encodage est de l'ordre de 2,2 images par seconde. Cette vitesse d'encodage nous indique que pour encoder 20 minutes de vidéo à 100 images par seconde, il y a besoin de plus de 15 heures pour encoder un examen. Il y a donc un réel besoin, au-delà du besoin technique de tampon et d'encodage en temps réel, d'accélérer le traitement de la vidéo pour les besoins d'utilisation.

Nous avons utilisé l'outil perf nous permettant de réaliser un profiling d'une application. Le résultat de ce profiling est visible dans la figure 55. Le total séquentiel de l'encodage est d'environ 410 millisecondes. En cas de réalisation en parallèle en forme de pipeline, le temps d'encodage sera le temps de la tâche le plus long qui est 169 millisecondes. Or pour encoder à 100 images par seconde, nous ne pouvons consacrer que 10 millisecondes par images. Pour améliorer ce facteur 17, nous proposons une implémentation matérielle du codeur ROI-Waaves.

Ceci conclut la description de l'implémentation logicielle. Le principal inconvénient de cette implémentation est la compression vidéo réalisée en logiciel qui ne permet pas d'avoir un traitement en temps réel du flux. Dans la section suivante, nous détaillons une deuxième implémentation visant à régler cet inconvénient.

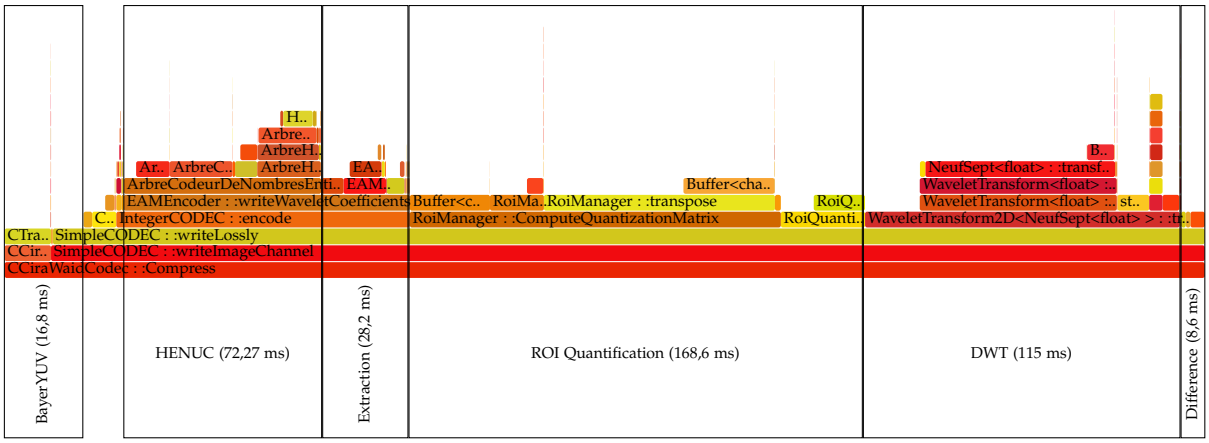



**Figure 55.** Profiling de l'encodage d'une séquence d'images en utilisant ROI-Waaves.

# 3 Implémentation matérielle

Pour rappel, la différence entre l'implémentation matérielle et l'implémentation logicielle est l'implémentation de l'encodeur vidéo dans une cible FPGA plutôt qu'en logiciel. Il s'agit de travaux préliminaires. Par manque de temps, aucune intégration complète n'a pu être réalisée. Nous présentons les spécifications du système comme il devrait être dans une implémentation finale et les blocs développés pour le constituer en laissant le développement des blocs restant et l'intégration comme perspective. Dans un premier temps, nous nous concentrons sur l'architecture générale de l'implémentation pour ensuite expliciter les développements matériels réalisés sur le compresseur vidéo ROI-Waaves.

## 3.1 Architecture générale

L'implémentation matérielle a pour objectif d'être embarquée sur une unique puce contenant à la fois une cible FPGA et un processeur hardcore. Grâce à cela, le système peut être réduit à une carte simple permettant l'encodage et la transmission d'un examen. Ceci à donc pour impact trois principales différences avec l'implémentation logicielle. Premièrement le bus de communication entre la cible FPGA et le processeur n'est plus du PCI-Express mais généralement un bus AMBA qui connecte les processeurs ARM avec des périphériques. Deuxièmement, les processeurs embarqués dans les FPGA sont généralement moins performants que les processeurs Intel trouvables dans les ordinateurs de PC. Sur ce point, nous devons être sûrs que le processeur sera capable de réaliser ses tâches attribuées. Dernièrement, la tâche la plus lourde pour l'implémentation logicielle est la compression vidéo. Celle-ci est déplacée dans la cible FPGA de manière à pouvoir suivre la cadence de compression de cent images par seconde en Full-HD.

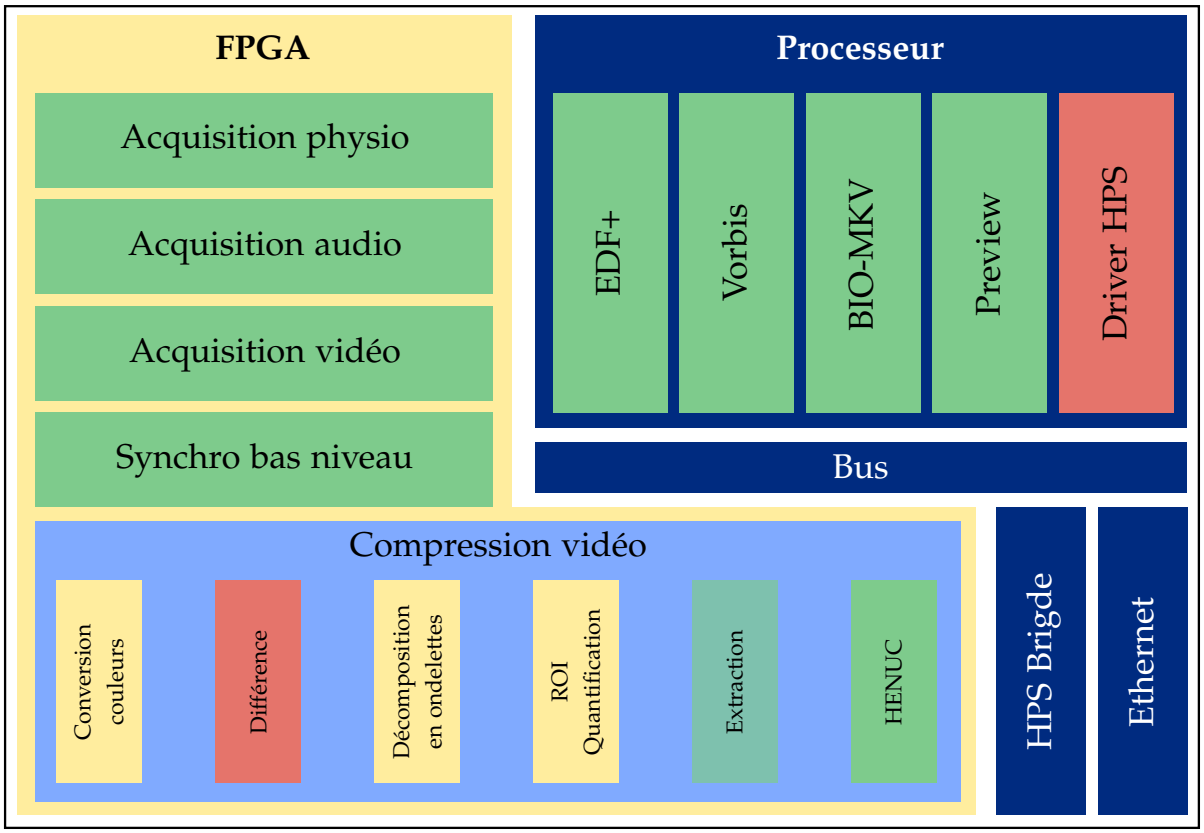



**Figure 56.** Schéma des spécifications de l'implémentation matérielle

L'architecture exposée dans la figure 56 montre la répartition des tâches du système entre le FPGA et le processeur. Nous remarquons que hormis la compression vidéo, les tâches de l'implémentation logicielle restent à la même place. Les tâches EDF+, BIO-MKV et Preview sont des tâches de mouvement de données et de formatage qui ne nécessite pas beaucoup de temps CPU mais principalement des accès mémoires. Nous évaluons d'ailleurs le pourcentage de temps CPU utilisé par BIO-MKV dans le chapitre VI. La compression Vorbis peut être plus coûteuse en temps CPU. En 2015, nous avons encadré Massime Bitam qui avait pour objectif de porter le compresseur sur une carte d'Altera SocKit contenant un ARM Cortex A9. Les évaluations de ce stage ont permis de conclure que pour une séquence audio d'une minute il fallait 11 secondes d'encodage sur le processeur ARM. Ceci représente moins de 20% du temps CPU ce qui laisse suffisamment de temps pour le reste des tâches de s'accomplir. La tâche matérielle/logicielle qui n'a pas été réalisée est l'interface de pilote entre les IP du FPGA et le logiciel tournant sur le processeur. Celle-ci remplace l'utilisation de *Riffa* de l'implémentation logicielle.

Sur le FPGA, les IP permettant l'acquisition des données et la mise en tampon en différent pipeline sont présents. Ceux-ci ne changent pas mis à part l'envoi vers le HPS. Ces IP sont donc prêtes, il ne reste que le portage sur le FPGA choisi. Le plus gros du travail à achever sur la partie FPGA est la conception du codeur ROI-Waaves.

## 3.2 Codeur matériel ROI-Waaves

Dans cette sous-section, nous décrivons les travaux réalisés sur la conception d'une architecture matérielle du codeur ROI-Waaves. D'un point de vue architectural, le codeur est organisé en un pipeline. La majorité des étages de ce pipeline utilise comme entrée/sortie des DMA lisant et écrivant des données depuis une mémoire DDR externe. La figure 57 illustre ce pipeline ainsi que les débits d'entrées/sorties des différents DMA sur la mémoire. La somme de ces débits est 4800 Mo/s. Les contraintes de bande passante sur cette mémoire sont lourdes et nous réalisons le développement dans l'idée que le goulot d'étranglement est celle-ci. Cette bande-passante est sans compter le débit de sortie du codeur et le débit nécessaire à relire le bitstream. Celui-ci sera cependant inférieur à 10 Mo/s donc nous avons choisi de l'ignorer. Les débits affichés sont donnés pour une séquence d'images Bayer de 1920 par 1024 pixels à 100 images par seconde. Pour avoir un débit suffisant, il faudrait baser la mémoire sur une mémoire DDR3 pouvant atteindre 20 Go/s théorique. Une autre solution serait de répartir sur deux bancs mémoire DDR2. Nous avons délibérément réduit l'image en hauteur de 1080 à 1024 de manière à simplifier les étages de DWT qui nécessiteraient sinon des exceptions embêtantes (dû à la division successive par 2 de 1080 qui ne tombe pas juste). Les DMA utilisés pour accéder à la mémoire ont déjà été développés pour la fifo de tampon de l'implémentation logicielle. Les modules verts de la figure 57 sont les modules développés et validés. Les modules jaunes sont ceux qui ont commencés à être développés ou qui n'ont pas totalement été validés. Les modules rouges n'ont pas été développés à ce point. Dans la suite de cette sous-section, nous remettons les étages du pipeline pour les décrire.

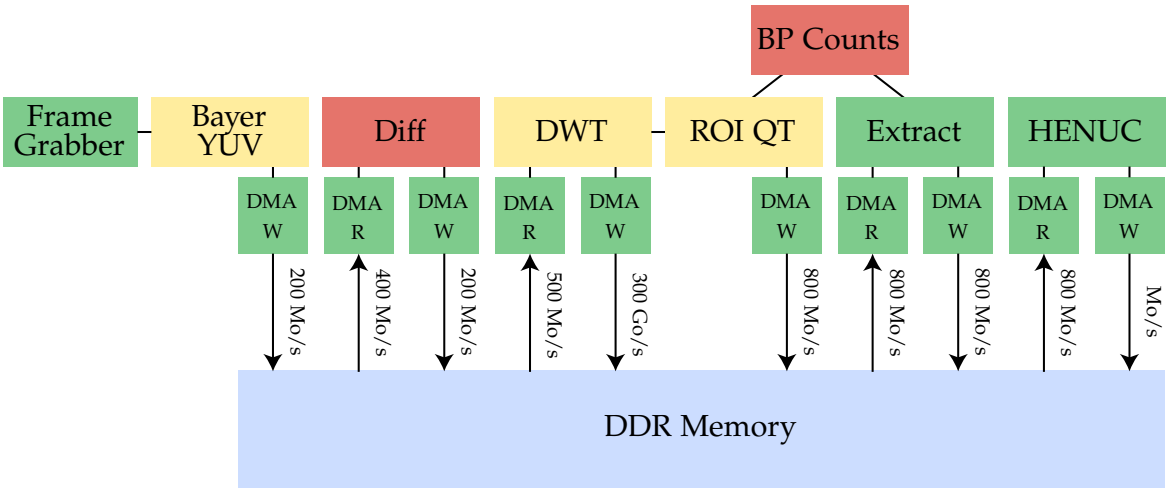



**Figure 57.** Schéma bloc de l'architecture proposée de ROI-Waaves

Le *frame grabber* utilisé pour récupérer les images est celui de l'implémentation logicielle. Il écrit pixel par pixel les données dans une fifo pour le premier étage du

codeur. Il écrit en parallèle les données dans l'ancien pipeline vidéo pour générer le flux de prévisualisation (pas représenté).

**Bayer vers YCbCr**

Le premier étage de compression est de faire passer les pixels de l'espace Bayer à l'espace YCbCr. Pour effectuer un calcul matriciel, il est nécessaire d'avoir un groupe de quatre pixels présents sur deux lignes différentes. Or, le *frame grabber* donne les pixels ligne par ligne. Pour répondre à cela, nous utilisons un *LineBuffer* qui est une mémoire circulaire de la taille d'une ligne d'une image. Les pixels rentrent un par un et ressortent dans le même ordre après avoir écrit toutes les autres cases de la mémoire. L'architecture du module est présente dans la figure 58. Le traitement est divisé en deux étapes :

- premièrement, représentés en rouge, les pixels de la ligne paire rentrent dans le *LineBuffer*.
- Deuxièmement, représentés en bleus, les pixels de la ligne impaire rentrent dans un registre temporaire. A ce moment, deux pixels de la ligne précédente sont retirés du *LineBuffer.* Le regroupement des quatre pixels du *LineBuffer* et du registre forme le groupe de quatre pixels permettant de produire les quatre pixels de composante YCbCr. Le calcul est réalisé directement et produit alors quatre pixels de sortie.

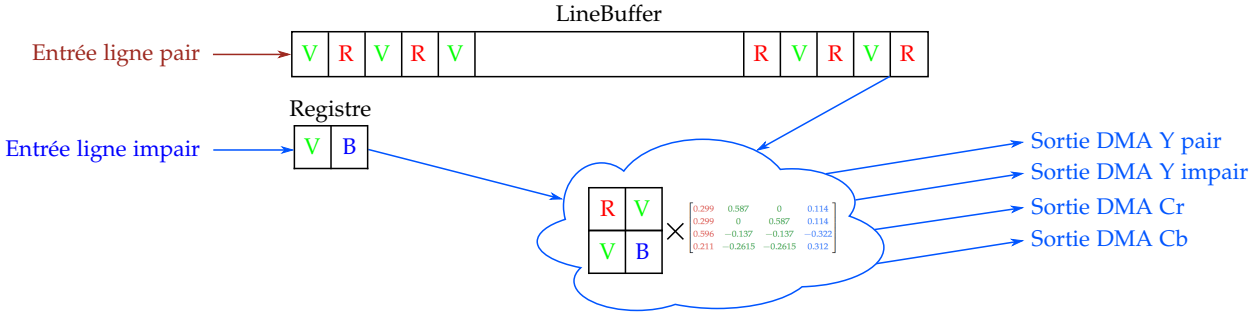



**Figure 58.** Architecture de la conversion Bayer vers YCbCr.

Le calcul matriciel demande de faire des opérations sur des nombres flottants. Pour contourner le coût du matériel pour faire ces opérations, nous convertissons les opérations en virgule fixe. Les coefficients de la matrice conversion sont dans une dynamique très faible entre 0 et 1, nous devons donc les mettre à l'échelle. Nous avons réalisé une évaluation, présente dans la figure 59, de l'erreur moyenne induite en fonction de la valeur d'échelle utilisée. À partir d'une échelle de 1024, les valeurs ont une erreur inférieure à 0,5 %. Au-delà de 1024, l'erreur ne semble pas diminuer. Nous utilisons donc 1024 comme valeur d'échelle. Nous convertissons les valeurs des pixels entrées de 8 bits en entier sur 32 bits et les multiplions par le facteur d'échelle. Les

coefficients sont statiquement mis à l'échelle. Le résultat de ce calcul est ensuite divisé par 1024*1024 pour retrouver la dynamique de valeur entre 0 et 256.

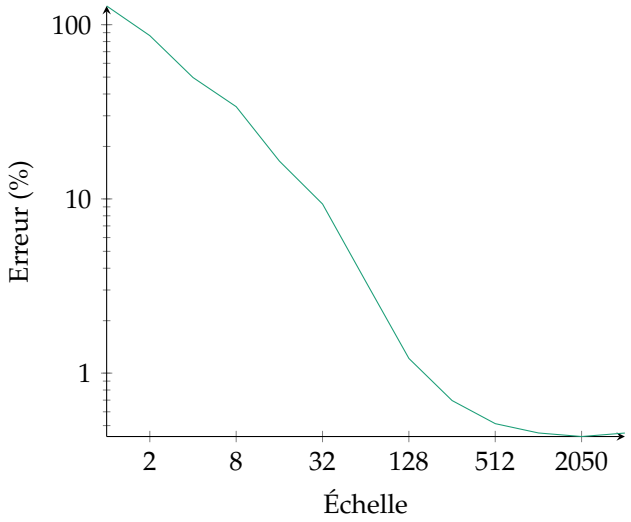



**Figure 59.** Impact de la conversion en nombre à virgule fixe en fonction du rapport d'échelle.

Les pixels sont ensuite écrits en utilisant des DMA. Comme le calcul produit deux pixels Y présents sur deux lignes différentes, nous avons besoin de deux DMA différents écrivant les pixels à une ligne d'intervalle dans la mémoire. Nous avons également besoin de deux DMA pour écrire les pixels Cb et Cr car ils sont placés dans des composantes séparées pour la compression.

**Différence**

Pour réaliser la différence, il aurait théoriquement pu être possible de faire le traitement au vol avec le flux de pixel venant du convertisseur YCbCr. Cependant, la différence est réalisée entre une image de référence et l'image courante. Nous ne pouvons pas garantir que le DMA lira l'image à temps pour faire la différence en temps avec le flux de pixel venant de la caméra. De ce fait, nous réalisons une mise en tampon avant la différence de manière à garantir l'intégrité de l'image dans la mémoire. Dans le cas où le DMA n'est pas capable d'écrire l'image en mémoire (car le pipeline est plein ou que la mémoire est occupée), elle est jetée. De plus, il aurait fallu dupliquer par quatre l'IP de différence pour traiter les quatre flux venant de l'IP précédente.

L'IP de différence est relativement simple, elle est constituée de deux DMA de lecture lisant l'image de référence et l'image courante à la même vitesse. Les pixels sont simplement soustraits un à un puis réécrits en mémoire par un DMA. Les images de référence sont sautées (pas lues) par ce module car elles ne requièrent pas de traitement. Cependant, l'adresse de l'image est mise à jour pour les futures images.

Une optimisation que nous n'avons pas implémentée est réalisable en mutualisant la lecture de l'image de référence avec plusieurs images de différence. Dans ce cas, il faudrait rajouter un DMA par images de différence rajoutées. C'est un compromis entre la surface de l'IP et la bande-passante en mémoire.

**Décomposition en ondelettes**

Le travail d'implémentation et de conception de la décomposition en ondelettes est le fruit de la thèse de Mohammed Shaaban Ibraheem [Ibraheem, 2017]. Nous ne rentrerons pas dans les détails de l'implémentation. Simplement, pour comprendre la bande-passante nécessaire, nous expliquerons les entrées/sorties.

- Premièrement, les pixels de l'image sont lus par un DMA ce qui correspond à $\sim 200\ Mo/s$. Au premier traitement, les pixels sont convertis en 32 bits afin de réaliser les calculs de la décomposition. L'IP est capable de réaliser une itération de DWT en ne lisant qu'une fois l'image. À la fin de l'itération, les trois quarts de l'image sont calculés et est écrits dans une fifo pour l'IP de quantification. Le reste de l'image est réécrit en mémoire pour la deuxième itération de DWT ($\sim 200\ Mo/s$).
- Lors de la deuxième itération, la zone de l'image lue est divisée par quatre mais chaque coefficient est stocké sur 32 bits ce qui correspond à $\sim 200\ Mo/s$. Après calcul, la zone réécrite en mémoire est $\sim 50\ Mo/s$.
- Pour la troisième itération, l'IP relu les $\sim 50\ Mo/s$ que l'itération précédente à généré et écrit un quart de cela c'est-à-dire $\sim 12,5\ Mo/s$.

On retrouve alors pour les lectures :

$$207,3 + 207,3 + 51,8 + 12,9 + 3,2 + 0,81 + 0,2 = 483,7\ Mo/s \tag{V.1}$$

et pour les écritures :

$$207,3 + 51,8 + 12,9 + 3,2 + 0,81 + 0,2 = 276,4\ Mo/s \tag{V.2}$$

L'IP génère près de 800 Mo/s de trafic sur la mémoire.

**ROI Quantification**

L'IP de quantification lit les coefficients à partir d'une fifo remplie par l'IP de DWT. Les coefficients arrivent des sous-bandes de hautes fréquences vers les basses fréquences en suivant le parcours de la figure 60 pour chaque itération. Pour chaque coefficient, il faut savoir de quelle classe il fait partie pour qu'il soit divisé par $Q_{ROI}$ ou $Q_{NROI}$. Pour cela, pour chaque niveau d'itération, une mémoire appelée mémoire des zones contenant cette information est instanciée. Elle est dimensionnée de manière à être de la taille d'une sous-bande de ce niveau et contenir un bit par coefficient. La taille des mémoires

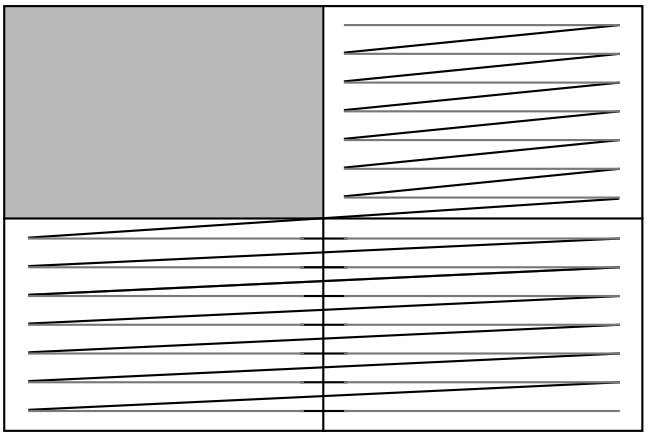


**Figure 60.** Ordre de réception des coefficients d'ondelettes par l'IP de Quantification

pour chaque niveau est présentée dans le tableau 8. Ces mémoires sont initialisées par les commandes reçues de la prévisualisation. Si elles ne sont pas initialisées alors elles sont par défaut initialisées à 1 (toute l'image est une zone d'intérêt).

**Tableau 8.** Taille des mémoires en fonction du niveau d'itération

| Itération | Taille en bits |
|---|---|
| 1 | $480 \times 512 = 245760$ |
| 2 | $240 \times 256 = 61440$ |
| 3 | $120 \times 128 = 15360$ |
| 4 | $60 \times 64 = 3840$ |
| 5 | $30 \times 32 = 960$ |
| 6 | $15 \times 16 = 240$ |

Des compteurs de ligne et de colonne sont utilisés pour suivre l'évolution des coefficients entrant dans l'IP. À partir de ces compteurs, il est possible de déterminer l'adresse de la valeur à récupérer dans la mémoire des zones. Chaque case est lu trois fois pour les trois sous-bandes de chaque itération. À partir de la valeur en mémoire, le coefficient est divisé par $Q_{ROI}$ ou $Q_{NROI}$. Un DMA écrit le résultat de l'opération en mémoire.

#### BP Counts

Cette IP est un module simple permettant de connaître le nombre de plans de bits présent dans chaque sous-bande. Pour ce faire, il se connecte à la sortie du module précédent et enregistre la plus grande valeur de chaque sous-bande. Le module détermine le nombre de bits pour représenter ce maximum. Cette information est ensuite utilisée par le module suivant pour connaître le nombre de plans de bits à traiter.

### Extraction

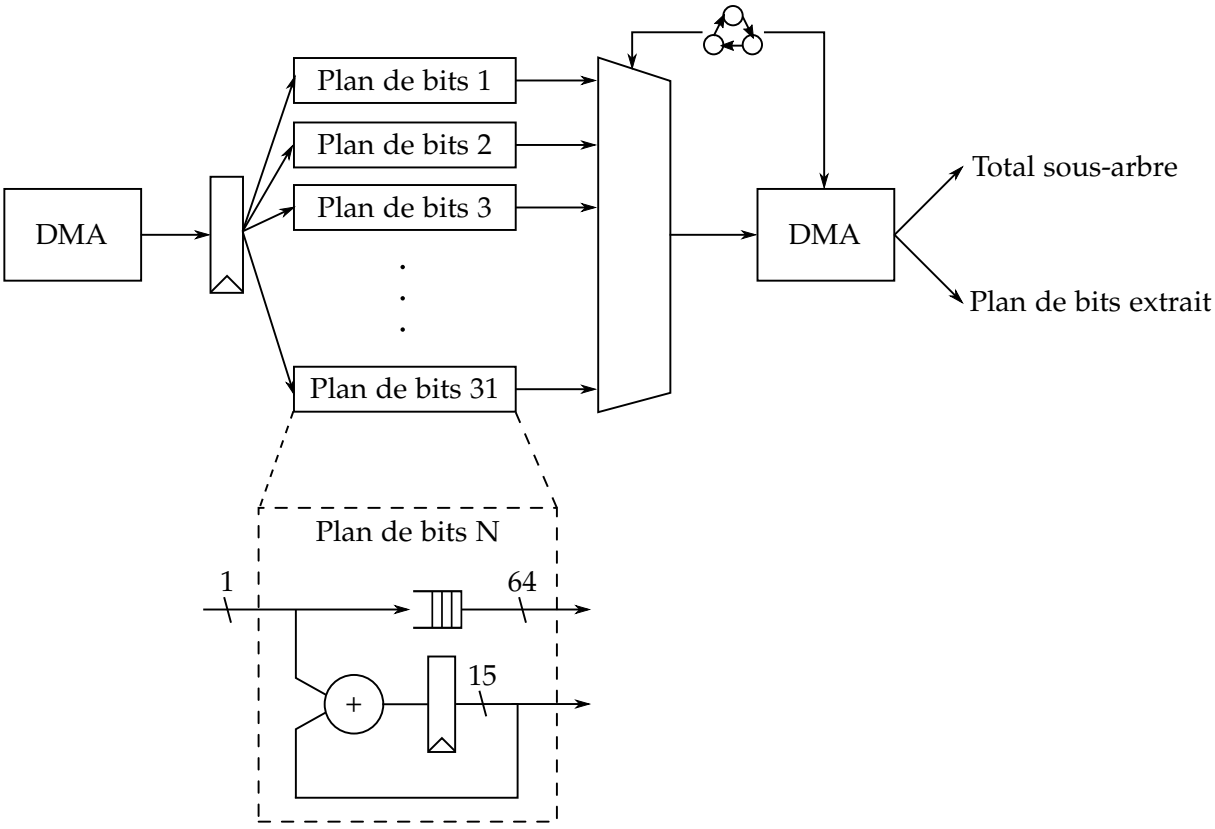



**Figure 61.** Architecture du module d'extraction.

L'IP d'extraction a trois rôles à remplir, lire les matrices en suivant le parcours bloc par bloc, séparer les plans de bits de la matrice à encoder et générer les valeurs des nœuds racines des sous-arbre de l'arbre HENUC. La première partie de l'IP est réalisée en utilisant un DMA intelligent jouant avec les adresses de lecture de la sous-bande traitée. Les coefficients sont lus en groupe de 16384 pour être traités. Ce nombre provient du découpage en sous-arbre de la version parallèle de HENUC. Pour chaque coefficient, le signe est séparé de la magnitude et est écrit directement dans une fifo. À partir du nombre de plans de bits du module *BP Counts*, l'IP sépare les coefficients sur 31 bits en flux de 1 bit. Dans le pire des cas, le nombre de plans de bits est 31. Pour se conformer à ce pire cas, il y a 32 flux (en comptant les bits de signe) de 1 bit d'instancié dans l'IP. Ceci consomme beaucoup de logique et de mémoire de dupliquer 32 fois mais cela permet de pouvoir traiter en une seule fois chaque coefficient. Il est alors nécessaire de lire chaque coefficient qu'une fois et donc permet d'alléger la mémoire. Les 32 flux rentrent dans de grandes fifos capables d'accueillir les 16384 bits du sous-arbre. À la volé le nombre de bits à vrai est compté dans un compteur unique par plan de bits. Ce total de bits à vrai est alors utilisé comme feuille de l'arbre de haut-niveau alors que le flux de bits est utilisé comme feuille des sous-arbres. Un par un, les flux sont traités puis écrits par un DMA en mémoire dans des tampons séparés. La figure 61 permet de visualiser cette architecture.

**HENUC**

Le dernier module HENUC permet de finaliser la compression. Cette IP est découpée en trois sous-modules :

- un sous-module pour encoder l'arbre de tête.
- Un sous-module duplicable pour encoder les sous-arbres.
- Un sous-module d'alignement mémoire pour générer le flux final.

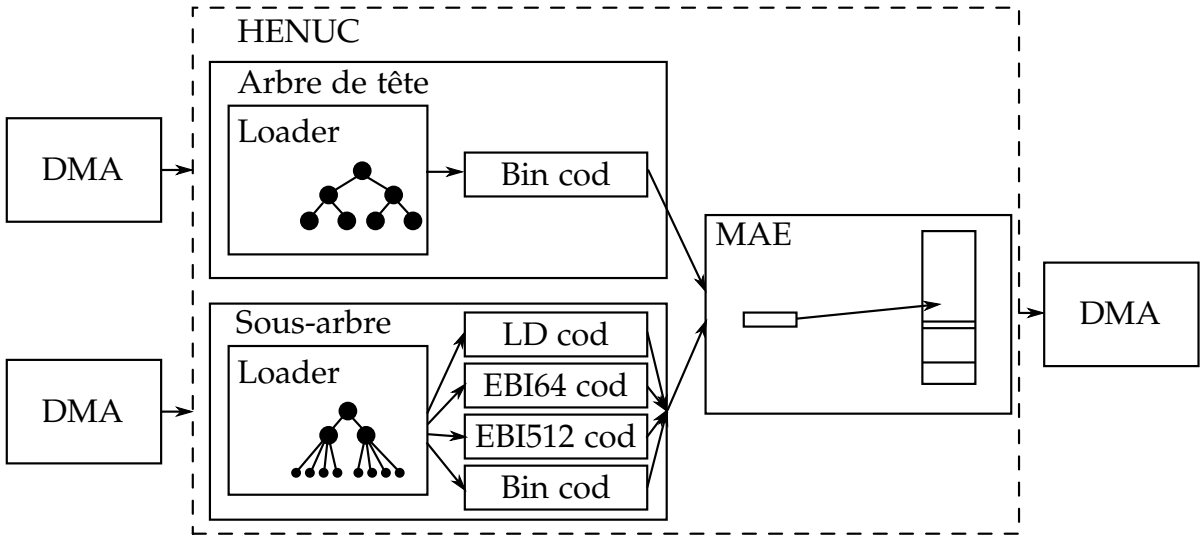



**Figure 62.** Architecture système du codeur HENUC.

Les connexions et architectures internes de chaque sous-module sont présentes dans la figure 62. Le sous-module Arbre tête lit les totaux écrits par le module d'extraction en utilisant un DMA dédié. Il stocke toutes ces valeurs dans une mémoire afin de représenter l'arbre de tête. Une fois tous les coefficients des feuilles de l'arbre de tête lu et écrit dans la mémoire interne de l'IP, le reste des nœuds de l'arbre sont calculés. À partir de ces nœuds, l'encodeur binaire peut lire ces valeurs et les encoder. Pour ce faire, elle ne génère pas directement de bitstream mais simplement les valeurs des symboles binaires sur 64 bits complétés avec des bits quelconques pour les bits inutilisés ainsi que le nombre de bits utilisé sur ces 64 bits. Ces deux valeurs sont transmises à l'unité suivante MAE qui se charge de la génération du bitstream à partir du flux de symboles généré par le codeur. [1]. L'IP sous arbre fonctionne de la même façon que l'IP d'arbre de tête à l'exception que les différentes couches de l'arbre sont encodées en utilisant des encodeurs différents. Ceci est géré par une machine à état qui aiguille les nœuds du sous-arbre vers le codec approprié. Pour finir, le MAE qui est aussi géré par une machine à état lit dans un premier temps les symboles venant de l'arbre de tête puis lit un par un les sous-arbres. Il assemble le bitstream en bloc de 4096 bits et les écrit ensuite en utilisant un DMA d'écriture.

1. Memory Alignement Engine

Nous verrons ensuite que le codeur HEC est un des goulots du codeur, pour améliorer sa bande passante, il est possible de multiplier le nombre de codeurs de sous-arbre afin d'en traiter plusieurs en parallèle. Le codeur MAE aura alors à lire cycliquement les différents codeurs de sous-arbre pour reconstituer le bitstream de l'image. Ceci est possible grâce à la modification algorithmique que nous avons réalisée sur le codeur HENUC en modifiant l'ordre des coefficients dans le bitstream.

Ceci termine la description du codeur vidéo et de l'implémentation matérielle. Dans cette implémentation, la majeure partie du travail restant sont des tâches de validation et d'intégration entre les IP du codeur vidéo et le portage des modules fonctionnels de l'implémentation logicielle vers cette nouvelle plateforme.

# 4 Conclusion

Dans ce chapitre, nous avons dans un premier temps décrit l'architecture globale du système contenant des modules de compression, de synchronisation et de contrôle. Les interfaces avec les différents modules extérieurs ont été spécifiées et implémentées par nos soins et ont été détaillées dans ce chapitre. Ensuite, deux versions d'implémentations ont été présentées.

La première permettant d'avoir un prototype fonctionnel rapidement, effectue les tâches nécessitant de manière indispensable du temps réel sur une plateforme FPGA et les autres tâches en logiciel. Ce prototype est à ce jour fonctionnel et permet l'acquisition de données physiologiques en parallèle d'un enregistrement de flux vidéo le tout synchronisé et compressé avec les mécanismes décrits dans les deux chapitres précédents.

Le second visant une plateforme plus embarquée à base de puce contenant un FPGA et un processeur a été décrit. Il vise à régler l'un des problèmes de l'implémentation logicielle qui est le temps de traitement de la compression vidéo. Nous avons donc utilisé les solutions algorithmiques proposées dans le chapitre IV afin de proposer un codeur ROI-Waaves entièrement matériel dont le but est d'être capable d'encoder un flux Full-HD à 100 images par seconde.

Les différents aspects de ces architectures et implémentations sont évalués dans le chapitre VI.

Chapitre VI

# Évaluation



Dans ce chapitre, nous évaluons les différentes solutions présentées dans les trois chapitres précédents. De ce fait, nous avons découpé cette évaluation en trois sections reprenant ces chapitres un par un.

## 1 Synchronisation

Dans cette section, nous commençons par évaluer la qualité de la synchronisation de bas niveau en précisions de temps. Nous élaborons ensuite les qualités de BIO-MKV pour ensuite analyser quantitativement son impact sur le flux et sur son besoin en temps de calcul.

### 1.1 Synchronisation bas niveau

Nous commençons l'analyse de la synchronisation par faire son évaluation en termes de précision. Pour mesurer cela, nous avons déployé un banc d'essai afin de stimuler deux modalités de manière à repérer si cette stimulation est concordante sur les deux signaux. La stimulation de base que nous utilisons est un signal carré servant de commande à une D.E.L. [1]. L'illumination de cette D.E.L. est enregistrée par la caméra afin de servir de premier signal évalué. La commande de cette D.E.L. est en parallèle

1. D.E.L. : Diode électroluminescente

acquise par une des 48 voies de signaux physiologiques qui constituent le deuxième signal évalué. La figure 63 schématise le montage effectué. Le système d'acquisition est configuré pour générer un fichier examen BIO-MKV contenant les deux modalités stimulées. Un script relit le fichier généré et extrait les images de la vidéo ainsi que les tensions de la borne physiologique stimulée. La vidéo est traitée afin de détecter l'état de la diode à chaque image. La tension enregistrée est également traitée pour connaître l'état de la commande à chaque échantillon. De ces deux courbes d'états sont découlés les moments de transition entre l'état haut et l'état bas. C'est sur cette base que les évaluations ont été réalisées. Le signal de commande pour alimenter la DEL est délivré par un GBF par un signal carré à une fréquence de 1 Hz. Il est bien à noter que l'expression du temps de cette mesure est relative à l'horloge système utilisée pour enregistrer les timestamps.

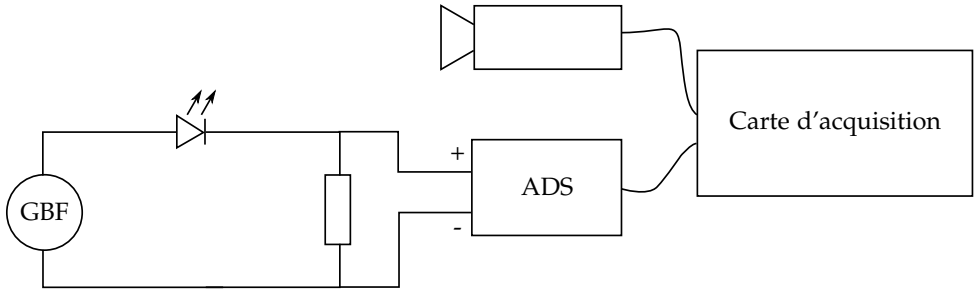



**Figure 63.** Schéma du banc d'essai pour évaluer la synchronisation bas niveau

Nous avons fait exprès de démarrer la commande dix secondes après le début de l'acquisition de manière à être sûrs que l'acquisition ne soit pas décalée d'une période exacte. Si cela avait été le cas, nous n'aurions pas pu détecter ce décalage avec un signal périodique. L'expérimentation a été effectuée sur une période de 20 minutes (durée d'un examen). Nous ne pouvons visualiser les 20 minutes d'acquisition car nous ne verrions rien sur le graphique, nous proposons alors de les visualiser par fenêtre de 20 secondes comme le ferait un expert. Nous présentons dans les figures 64, 65, 66 la visualisation des fronts montants détectés sur la vidéo et sur le signal physiologique.

À l'œil nu, nous ne pouvons pas détecter de décalage entre les deux modalités ce qui est déjà un résultat satisfaisant car les praticiens analysent les traces de manière visuelle. Cette constatation reste vraie sur les 20 minutes de l'acquisition.

Cependant, du fait que l'acquisition vidéo et l'acquisition physiologique ne soient pas synchronisées, nous savons qu'il y a un décalage entre les valeurs des timestamps des fronts détectés. Pour mesurer ce décalage, nous avons représenté graphiquement la différence un à un des valeurs de timestamps des fronts détectés entre la vidéo et le

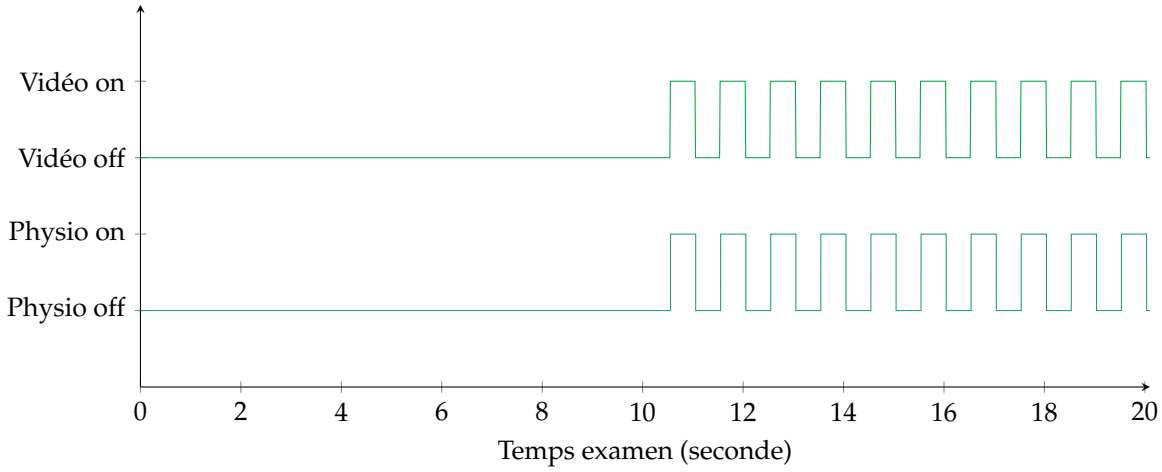



**Figure 64.** Visualisation des fronts détectés sur la vidéo et sur le signal physiologique lors des 20 premières secondes d'acquisition

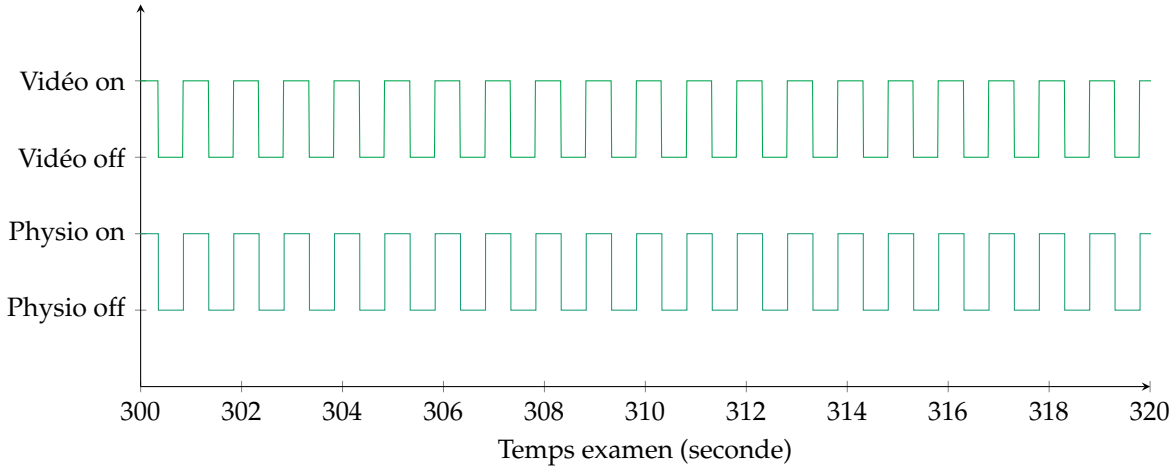



**Figure 65.** Visualisation des fronts détectés sur la vidéo et sur le signal physiologique après 10 minutes d'acquisition

signal physiologique. Les abscisses représentent les numéros des fronts détectés et les ordonnées la différence de synchronisation. Cette représentation peut être observée dans la figure 67. Nous remarquons que le décalage entre la vidéo et le signal physiologique est réparti entre 0 et 17 millisecondes.

Dans la figure 68, nous représentons les intervalles mesurés entre deux fronts détectés sur le signal physiologique. Nous voyons qu'il y a une variation sur la détection du front montant de 5 millisecondes. Dans la figure 69, nous effectuons la même représentation pour le signal vidéo. Nous constatons que la variation dans ce cas est plus grande de l'ordre de 20 millisecondes. Il faut comprendre qu'il y a une variation d'une image avant et une image après lors de la détection du front montant.

Sur ces deux schémas, nous avons représenté la moyenne de la variation nous

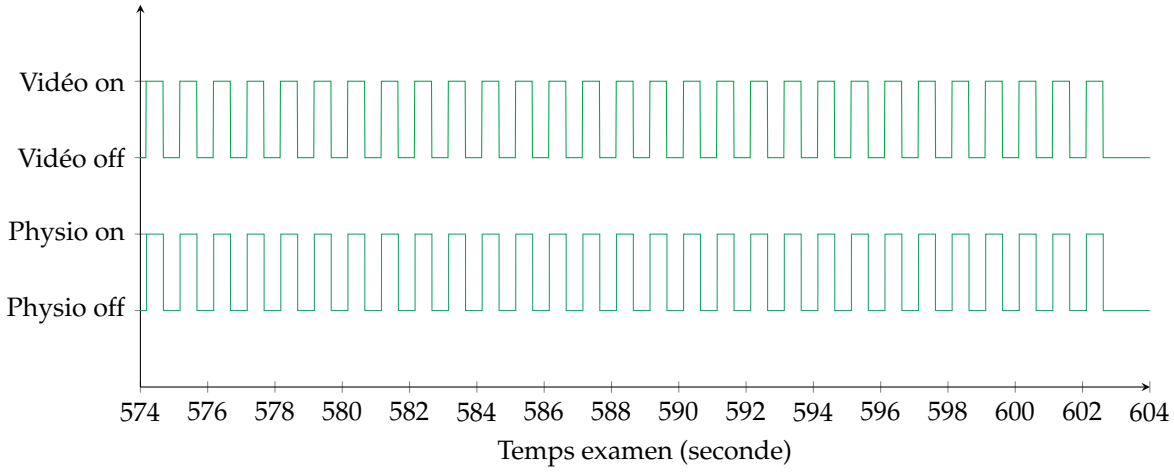



**Figure 66.** Visualisation des fronts détectés sur la vidéo et sur le signal physiologique après 20 minutes d'acquisition

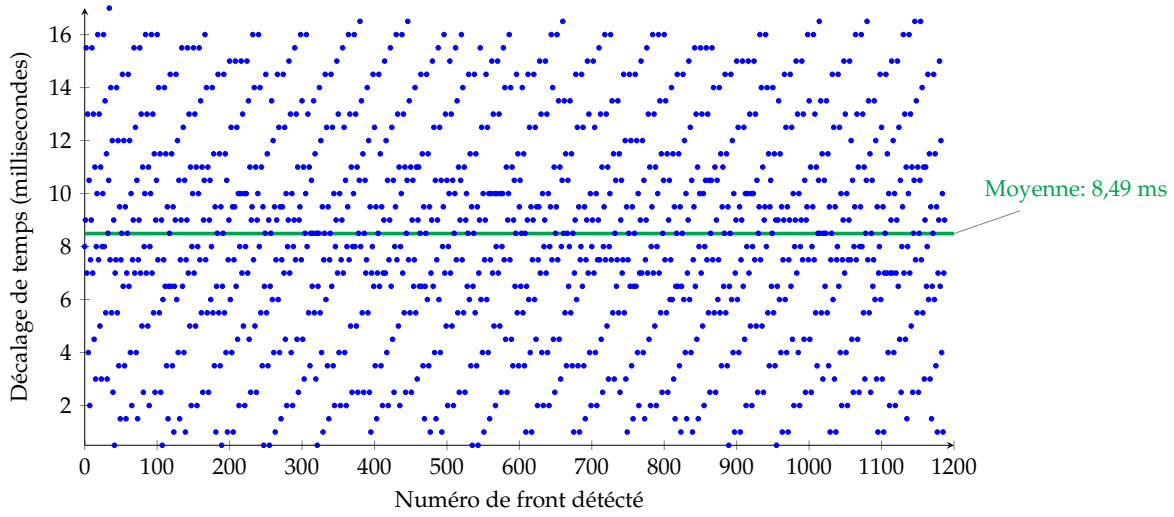



**Figure 67.** Visualisation des différences un à un entre les fronts détectés de la vidéo et du signal physiologique.

donnant la demi-période du signal mesuré. Nous pouvons voir que malgré deux fréquences d'acquisition différentes et des variations différentes de détection, nous obtenons une demi-période similaire pour les deux modalités. Nous pouvons alors dire que l'acquisition est synchronisée en fréquence avec des variations allant de 0 à 17 millisecondes.

Dans l'exemple exposé, nous pouvons voir que notre horloge système et l'horloge du GBF ne sont pas synchronisées. En effet, notre mesure montre que le GFB produit à intervalle de 498,78 millisecondes alors que nous attendions 500 millisecondes. Cette expérience montre que les systèmes d'acquisition qui font des hypothèses de fréquence

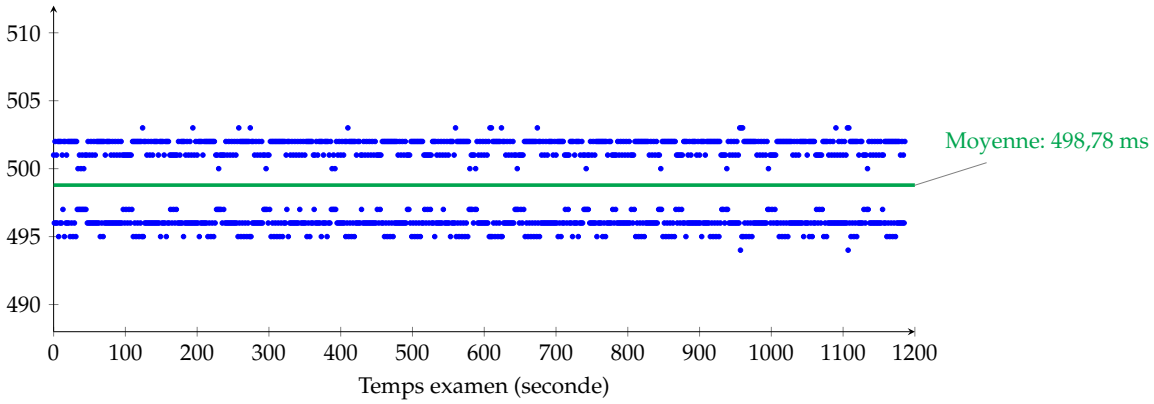



**Figure 68.** Temps mesuré entre deux fronts détectés sur le signal physiologique

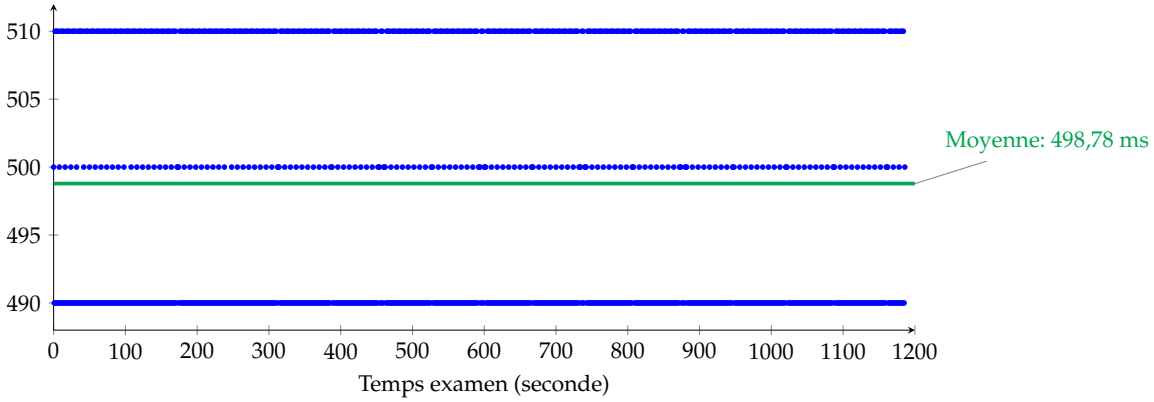



**Figure 69.** Temps mesuré entre deux fronts détectés sur la vidéo

sans synchronisation ne peuvent pas générer des acquisitions synchrones. Dans cet exemple, notre système se serait décalé de 1.22 milliseconde * 1200 = 1464 millisecondes, c'est-à-dire 1,5 seconde environ. Ceci montre l'intérêt d'une telle synchronisation de bas niveau.

Un point à ne pas oublier, la synchronisation a été réalisée avec des composants pris sur l'étagère. Ceci diffère en termes de précision avec d'autres mécanismes de synchronisation dédiés que des sociétés, comme NI, mettent en place pour leur système PXI que nous ne pouvions pas nous permettre pour cause de coût de production.

## 1.2 Synchronisation haut niveau

**Analyse qualitative**

Pour l'analyse qualitative de la synchronisation bas niveau, nous pouvons dire que le mécanisme mis en place permet d'attribuer des *timestamps* de manière fiable. En effet, le mécanisme ne se repose que sur des composants ne comportant pas d'indéterminisme, de ce fait il ne peut y avoir de variation quant à l'acquisition de l'évènement (encore une fois seulement si le signal de déclenchement est suffisamment fiable).

L'utilisation de BIO-MKV comme format de fichier pour transporter des données physiologiques et la vidéo de l'examen apporte un certain nombre d'avantages.

- **Resynchronisation** : comme toutes les données sont estampillées par un *timestamp* nous sommes capables de recaler le flux vidéo avec les flux de signaux physiologiques même si les fréquences d'acquisition ne sont pas multiples de l'un et de l'autre.
- **Compression** : parce que le cadre de l'étude est la télémédecine, l'utilisation d'algorithme de compression pour les signaux physiologiques est une option à prendre en compte même si dans notre étude ceux-ci ne sont pas compressés. Rien dans le format BIO-MKV n'empêche de le faire pour de prochaine version.
- **Édition** : en utilisant un conteneur multimédia, l'édition du flux que ce soit pour l'insertion ou la suppression de données est simplifiée puisque chaque piste et paquet sont indépendants. Il est aussi assez simple de rajouter une piste de données si besoin. De plus, extraire une partie de l'examen pour l'isoler à des fins d'archivage est également faisable et ne nécessite aucun réencodage.
- **Données non continues** : l'abstraction introduite par BIO-MKV permet d'avoir des données qui ne se suivent pas forcement temporellement. Cette fonctionnalité est facilement intégrable et n'implique pas de surcoût. Par exemple pour les annotations, des paquets sont insérés quand il y a besoin et ne consommant de l'espace que quand nécessaire.

De plus, comme nous utilisant EDF comme codec à l'intérieur de BIO-MKV, extraire un fichier EDF du flux est simple car suffit d'extraire les données physiologiques et les concaténer. Ceci veut dire que l'on est capable de retrouver un format compatible avec différents logiciels de lecture. Grâce à cela, il est alors possible de relire les examens avec le logiciel d'un constructeur des systèmes d'acquisition physiologique que l'on pourrait trouver dans les hôpitaux.

**Analyse quantitative**

Nous voulons évaluer le surcoût que le conteneur BIO-MKV ajoute par rapport aux données seules. Pour ce faire, nous rappelons que le format BIO-MKV contient un entête général en début de fichier décrivant le contenu du flux et une entête par paquet décrivant le contenu du paquet en question. Nous avons donc effectué deux évaluations, la première pour mesurer l'impact de l'entête général et la deuxième pour mesurer l'impact des entêtes de paquets.

Pour faire ces évaluations, nous avons développé une application génératrice de paquets. Elle permet à partir d'une configuration de taille de paquets et d'un nombre de paquets de générer un fichier BIO-MKV. À partir de ce flux, nous pouvons déterminer la proportion de données propre à BIO-MKV et les données contenues dans les flux. Cette métrique que nous appelons surcoût est le rapport de la taille des données de BIO-MKV et de la taille du flux complet.

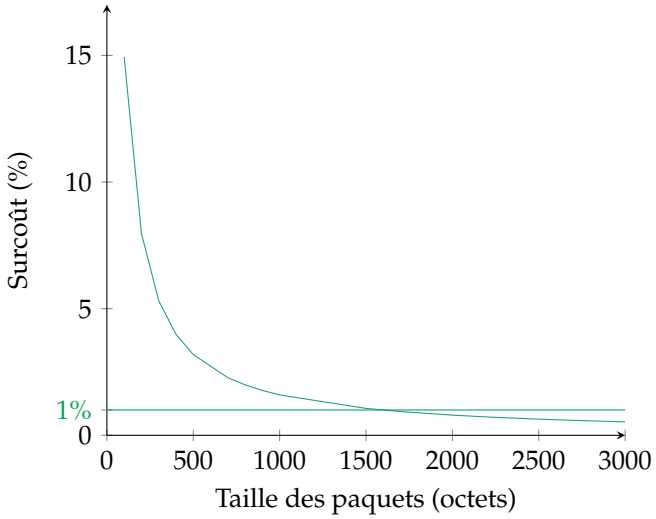



**Figure 70.** surcoût de BIO-MKV en fonction de la taille des données multiplexées.

Dans un premier temps, nous voulons évaluer l'impact de l'entête général. La proportion du fichier relative à l'entête général sera la plus haute quand il n'y aura très peu de paquets dans le flux (avec un maximal à zéro paquet), et plus il y aura de paquets dans le flux plus son impact sera réduit. Nous avons donc généré des flux en variant le nombre de paquets du flux mais en gardant fixe la taille des paquets fixe (10 ko). La figure 70 montre l'évolution du surcoût avec l'évolution du nombre de paquets dans le flux. On remarque que à partir de treize paquets dans le flux le surcoût est inférieur à $0,3\%$. Dans un examen de vingt minutes et en ne prenant en compte que la vidéo, il y a 120 000 paquets générés, ce qui fait passer le surcoût de l'entête général en dessous des $0,15\%$. Nous pouvons donc considérer que dans un cas d'examen classique cet entête est très largement négligeable.

Pour évaluer le conteneur, nous avons mesuré l'impact que les entêtes de chaque paquet ont sur la taille du flux. Le graphique 71 montre cette mesure en fonction de la taille des paquets de données. Nous pouvons voir sur ce graphique que plus la taille des paquets est grande plus le surcoût est petit. Ce résultat montre que le surcoût de la solution apportée est acceptable au-dessus de 1500 octets (moins de 1%).

Des deux mesures effectuées et exposées ci-dessus, nous pouvons remarquer que l'utilisation de BIO-MKV comme conteneur multimédia n'engendre pas de gros surcoût par rapport aux données compressées sans conteneur. De plus, son utilisation permet de multiplexer des données compressées ce qui en réalité permet de faire un gain de place par rapport aux données brutes.

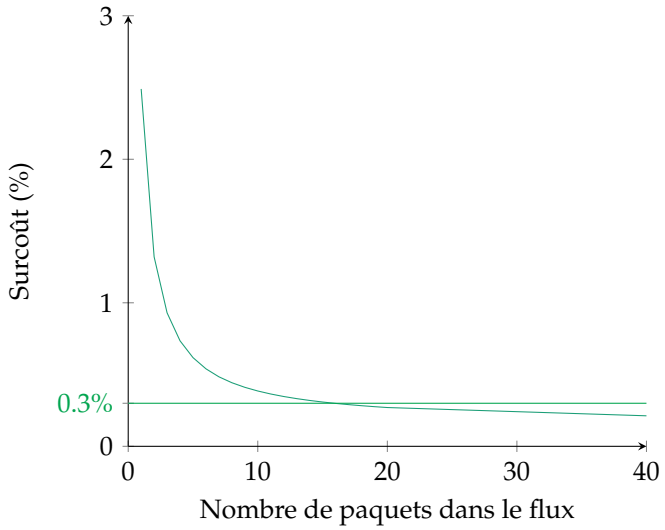



**Figure 71.** surcoût de BIO-MKV en fonction du nombre de paquets multiplexés.

Nous désirons maintenant savoir si la puissance de calcul nécessaire à construire le flux BIO-MKV est réalisable en temps réel (c'est-à-dire au moins aussi vite que l'arrivée des données). Pour ce faire, nous utilisons toujours le même programme de génération de paquet qui nous permet de multiplexer des paquets sans faire de compression. Le temps d'exécution du programme est égal au temps utile à multiplexer le nombre de paquets demandé au générateur. Dans un premier temps, nous avons mesuré le temps d'exécution du programme en fonction du nombre de paquets. La figure 72 montre cette évolution dont nous remarquons la linéarité ce qui est une bonne chose. Nous constatons aussi que pour multiplexer 200 000 paquets, qui constituent un examen d'une vingtaine de minutes, il faut environ 7 secondes ce qui est très négligeable.

Pour mettre en relation le temps passé à multiplexer comparé au temps de l'examen, nous calculons, à partir du temps d'exécution, le rapport du temps de multiplexage par le temps de l'examen. Cela nous donne le pourcentage de temps qu'il faudrait passer pour effectuer le multiplexage du fichier. Dans la figure 73, nous illustrons ce rapport en fonction du nombre de minutes de l'examen. Nous remarquons par cette courbe que le temps nécessaire est inférieur à 0,8%. Ceci est très bien car cela laisse 99,2% de temps pour faire le reste des traitements.

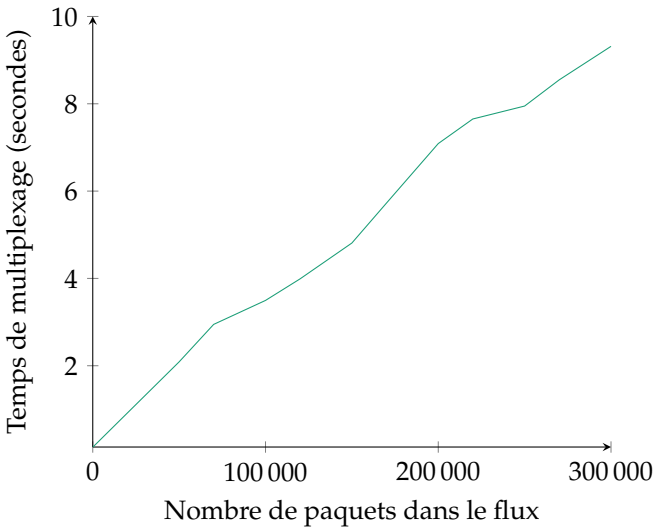



**Figure 72.** Temps de multiplexage de BIO-MKV en fonction du nombre de paquets multiplexés.

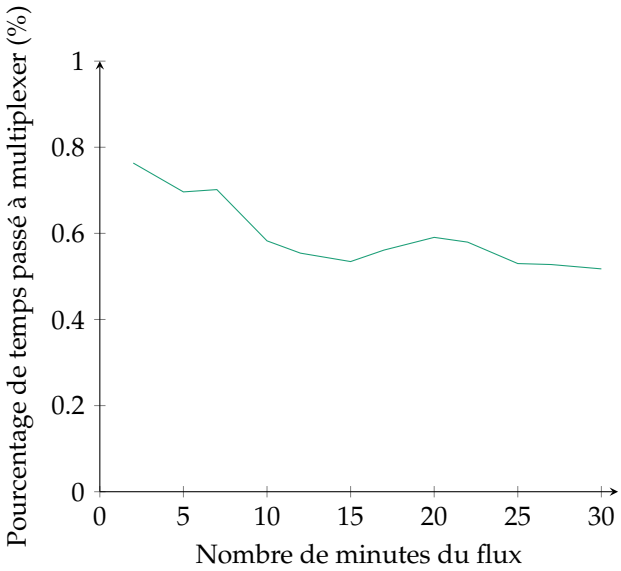



**Figure 73.** Pourcentage de temps réel passé à multiplexer BIO-MKV en fonction de la longueur du flux.

# 2 Compression

Dans cette section, nous proposons d'évaluer l'algorithme de compression ROI-Waaves. Pour ce faire, nous proposons d'évaluer l'algorithme sur les critères de qualité de taux de compression et de vitesse d'encodage/décodage. Dans un premier temps, nous réalisons une étude théorique en complexité de l'algorithme et nous la comparons aux codecs de l'état de l'art. Ensuite, nous définissons comment nous réalisons l'évaluation et calculons chaque critère évalué. Pour finir, nous présentons pour plusieurs séquences d'images les résultats des évaluations.

## 2.1 Analyse de complexité du codeur

La complexité en temps d'un algorithme est le nombre d'étapes permettant de finir un algorithme. Celui est en fait fonction de la taille d'entrée de l'algorithme et permet de connaître le comportement asymptotique de l'algorithme quand cette taille tend vers l'infinie. La complexité d'un algorithme est notée $\mathcal{O}(n)$ pour un algorithme dit linéaire (dont le temps de calcul est proportionnel à la taille des données d'entrée). La complexité suit les règles de composition suivantes :

$$\mathcal{O}(K \times f(n)) = \mathcal{O}(f(n)), \text{ quand } K \text{ est une constante} \tag{VI.1}$$

$$\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n)), \text{ quand } \lim_{n \to +\infty} \mathcal{O}(f(n)) >= \lim_{n \to +\infty} \mathcal{O}(g(n)) \tag{VI.2}$$

La première règle permet de simplifier n'importe quel membre constant d'une expression et la deuxième règle définit que le résultat de la somme de deux complexités est la complexité la plus importante des deux. Par exemple, l'expression suivante $\mathcal{O}(3 \times n) + \mathcal{O}(6 \times n \times log(n))$ se simplifie en $\mathcal{O}(n \times log(n))$ car le membre $\mathcal{O}(3 \times n)$ est toujours inférieur à $\mathcal{O}(6 \times n \times log(n))$ quand n est positif. Finalement, le membre constant 6 peut être simplifié ce qui nous donne la complexité $\mathcal{O}(n \times log(n))$

.

Afin de déterminer la complexité du codec globale, nous allons étudier la complexité de chaque étage. Nous noterons $n$ comme étant le nombre d'éléments à traité dans l'étage courant et qui est toujours fonction du nombre de pixels de l'image.

- **Conversion couleur :** La conversion couleur est une étape dans laquelle chaque pixel est multiplié par une matrice pour donner une autre valeur de pixel. Cette opération s'exécute en temps constant et est répétée $n$ fois. La complexité de l'étage est donc $\mathcal{O}(n)$.
- **Différence temporelle :** De la même manière que pour la conversion couleur, la différence temporelle est une opération en temps constant qui est réalisée une fois

par pixel. La complexité de la différence est donc la même que pour la conversion couleur c'est-à-dire $\mathcal{O}(n)$.

- **Transformée en ondelette :** La complexité de la transformée en ondelette est un peu plus compliquée que pour les deux premières. La transformée en ondelette est une convolution. La complexité d'une convolution est $\mathcal{O}(n \times m)$, dans laquelle $n$ est le nombre d'éléments de la fonction et $m$ le nombre de coefficients du noyau de convolution. Dans notre cas, le noyau de convolution est une constante donc nous pouvons simplifier $m$. Ainsi, la complexité d'une itération de convolution est $\mathcal{O}(n)$. Cette opération est itérativement répétée sur la sous-bandes DC. La sous-bandes DC est réduite par quatre à chaque itération. L'équation suivante exprime le nombre d'opérations d'une transformée complète en fonction du nombre de pixels :

$$n + \frac{n}{4} + \frac{n}{4^2} + ... + \frac{n}{4^{NB_{iteration}}} = n \times \sum_{i=0}^{log_4(n-256)} \frac{1}{4^i} \tag{VI.3}$$

  La deuxième forme de cette équation nous sert ci-dessous pour déterminer le nombre d'itérations de transformée à réaliser. Celui dépend du nombre de pixels donc doit être pris en compte pour l'analyse de la complexité de la transformée en ondelette. En utilisant la formule précédente, nous déterminons que la complexité de la transformée complète :

$$\begin{aligned} \mathcal{O}(DWT) &= \mathcal{O}(n) + \mathcal{O}(\tfrac{n}{4}) + \mathcal{O}(\tfrac{n}{4^2}) + \cdots + \mathcal{O}(\tfrac{n}{4^{NB_{iteration}}}) \\ \mathcal{O}(DWT) &= \mathcal{O}(n \times log_4(n-256)) \\ \mathcal{O}(DWT) &= \mathcal{O}(n \times log(n)) \end{aligned} \tag{VI.4}$$

  Nous déduisons alors que la complexité de cet étage est $\mathcal{O}(n \times log(n))$

- **ROI Quantification :** La quantification est réalisée en effectuant une opération de complexité constante par pixel. La complexité de l'étage est donc $\mathcal{O}(n)$.
- **Extraction :** L'extraction de plans de bit effectue pour chaque pixel de chaque plan de bit une opération. De ce fait, la complexité dépend du nombre de pixels et du nombre de plans de bits. La complexité de l'étage est $\mathcal{O}(n \times m)$ dans laquelle $n$ est le nombre de pixels et $m$ est le nombre de plans de bits maximum. En prenant en compte que $m$ est borné par la valeur 32 et ne dépend pas du nombre de pixels d'entrée, la complexité de l'étage redescend à $\mathcal{O}(n)$.
- **HENUC :** Finalement, HENUC encode $n$ pixels dans la couche zéro. Chaque opération d'encodage dans l'arbre est effectuée en temps constant. La hauteur de l'arbre est variable en fonction de la configuration de l'arbre mais peut-être bornée par le pire cas qui est que chaque niveau regroupe les pixels par deux. Dans ce cas, la hauteur de l'arbre est $log(n)$. La complexité de l'algorithme est donc la hauteur multipliée par la largeur de l'arbre et donc $\mathcal{O}(n \times log(n))$.

De toutes les complexités précédemment établies, il est possible d'établir l'équation suivante pour l'algorithme général :

$$\begin{aligned} \mathcal{O}(algorithm) &= \mathcal{O}(Conversion\ couleur)+ \\ &\quad \mathcal{O}(Diffrence\ temporelle)+ \\ &\quad \mathcal{O}(DWT)+ \\ &\quad \mathcal{O}(ROI\ Quantification)+ \\ &\quad \mathcal{O}(Extraction)+ \\ &\quad \mathcal{O}(HENUC) \end{aligned} \tag{VI.5}$$

$$\begin{aligned} \mathcal{O}(algorithm) &= \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n \times log(n))+ \\ &\quad \mathcal{O}(n) + \mathcal{O}(n) + \mathcal{O}(n \times log(n)) \end{aligned}$$

$$\mathcal{O}(algorithm) = \mathcal{O}(n \times log(n))$$

En simplifiant, nous déterminons que les deux étages les plus complexes de ROI-Waaves sont les blocs de DWT et le codeur entropique HENUC. Nous déterminons de cette analyse que la complexité de l'algorithme est $\mathcal{O}(n \times log(n))$.

Pour rappel, Waaves réalise un tri des coefficients après chaque sélection de coefficient. Donc cela se traduit par réaliser un tri à chaque fois que l'on traite un coefficient. Le tri utilisé est un tri par panier [Corwin and Logar, 2004] qui a une complexité de $\mathcal{O}(n)$. Cette opération est répétée pour chaque pixel donc la complexité est $\mathcal{O}(n) \times \mathcal{O}(n) = \mathcal{O}(n^2)$. Cette complexité est supérieure au nouveau codeur proposé qui est de $\mathcal{O}(n \times log(n))$. Nous avons donc effectivement réduit la complexité du codeur. Attention, réduire la complexité ne veut pas dire que les performances seront meilleures, elle permet simplement de dire que l'algorithme sera plus performant quand la taille de l'image grandira. En négligeant les constantes, l'analyse ne permet pas de déterminer le comportement pour des tailles d'images plus petites.

## 2.2 Performance du codec

**Protocole d'évaluation**

Afin d'évaluer le codec proposé, nous prenons en considération les points suivants :

- le débit que nous déterminons en mesurant le nombre d'octets par secondes produit par le codeur. Ceci permet de connaître le taux de compression que le codeur est capable d'accomplir.
- La qualité de l'image restituée que nous mesurons grâce à l'indice SSIM [1][Wang et al., 2004]. Cet indice est un score de similarité entre deux images.

1. SSIM : Structural SIMilarity

Nous l'utilisons pour comparer la similarité d'une image compressée à son image d'origine.

- La vitesse de compression que nous évaluons en images par seconde. Plus le codec est capable d'encoder un grand nombre d'images par seconde, plus il est rapide.

Pour mesurer ces trois critères, nous avons développé un banc de tests automatisé afin d'évaluer les différentes configurations de codec pour une séquence d'images. Dans ce banc sont testées plusieurs configurations de ROI-Waaves, mais également Waaves, JPEG-2000, H264 et VP8. À noter que l'implémentation de ces derniers codeurs sont celles présentes dans FFmpeg. L'entrée de ce banc de tests est la séquence d'images à encoder au format Bayer. Le codec ROI-Waaves lit directement ces images, les encode et les décode pour produire les images visibles par le relecteur d'examen. Le bitstream entre l'encodage et le décodage permet de mesurer la bande passante utilisée par le codeur. L'image d'entrée convertie en RGB ainsi que l'image de sortie permet de mesurer le SSIM. Le calcul du SSIM est réalisé pour chaque image du flux vidéo. Les temps passés à encoder et à décoder sont mesurés. Toutes ces données sont enregistrées dans un fichier JSON qui est un fichier structuré permettant d'exploiter les résultats par la suite. Les autres codecs sont évalués en utilisant le même procédé à la différence que la conversion RGB est réalisée en entrée du codec et non en sortie. La même méthode de conversion Bayer vers RGB est utilisée dans les deux cas. Pour chaque codec et chaque configuration de codec, un paramètre de qualité (qui est différent pour chaque codec) est utilisé pour pouvoir faire varier la qualité du flux encodé.

Afin de tracer les courbes, les valeurs de SSIM sont moyennées pour chaque configuration et paramètre de codec. Le même procédé est réalisé pour le débit. Cette paire de valeur nous permet de déterminer un point d'une configuration d'un codec. En combinant tous les points d'une configuration, nous obtenons une courbe qui représente la qualité d'une séquence vidéo en fonction du débit allouée. Les vitesses de compression et de décompression sont simplement moyennées pour chaque configuration de codec. Ces vitesses exprimées en FPS sont à prendre avec précaution car elles sont mesurées sur machine qui sont à pleine charge (avec d'autres encodages tournant en parallèle). Nous ne prendrons alors compte que des vitesses relatives et non absolues. Le processus de validation est mis en avant par la figure 74.

Nous avons réalisé l'évaluation sur cinq séquences d'images utilisant des plans fixes comme critère. Les images montrées dans cette section sont les images brutes visant à être encodées :

- la vidéo *toddler* (figure 75) [Toddler] est une vidéo 4K mise à disposition par Netflix. C'est une vidéo d'un enfant qui court dans des fontaines. Cette séquence est a priori plus compliquée que les vidéos que celles d'examen. Elle nous permettra de définir la portée du codeur développé. Nous avons visé l'enfant comme zone d'intérêt.

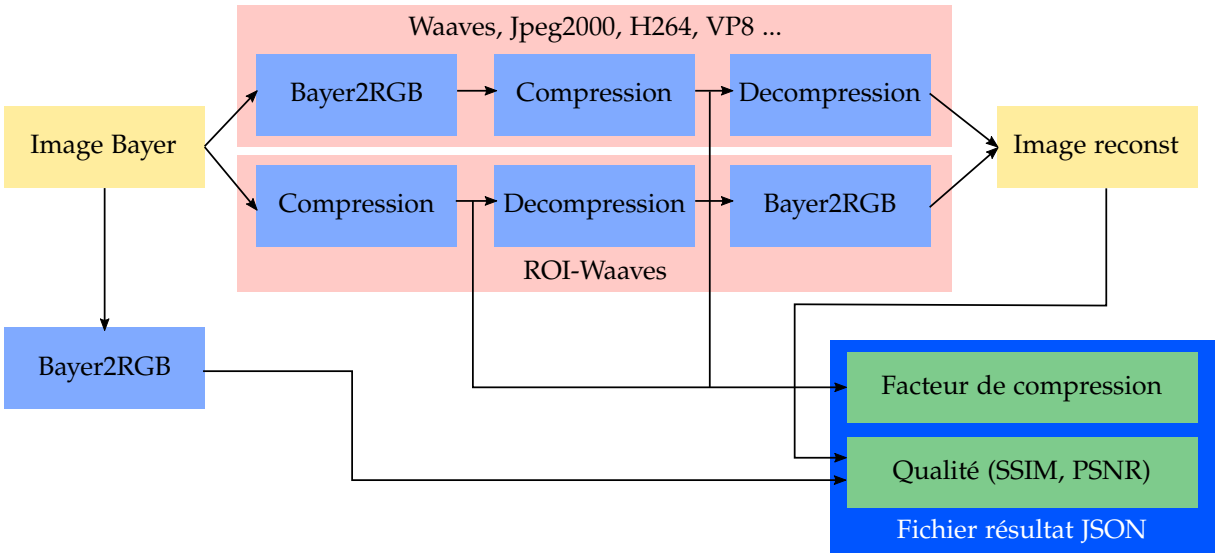



**Figure 74.** Schéma du processus d'évaluation des codeurs.

- La séquence *news* (figure 76) [News] est une reproduction 4K de la séquence news de la base de données Xiph media. Dans cette scène deux présentateurs parlent et une vidéo d'un ballet est montrée dans le fond. La zone d'intérêt sélectionnée sur cette vidéo est le présentateur de gauche. Il y a donc des éléments bougeant en dehors des zones d'intérêt.
- La séquence *suzie* (figure 77) [Suzie] est également une séquence reproduite 4K de la séquence suzie de la base de données Xiph media. Cette scène est un plan rapproché d'une femme au téléphone. La zone d'intérêt de cette séquence représente la plus grande partie de l'image.
- La séquence *wide* (figure 78) est une séquence vidéo prise par l'implémentation logicielle. Elle simule un patient lors d'un examen. Nous avons constitué à partir de cette séquence deux séquences avec des zones d'intérêt différentes : la première sur le visage et la deuxième sur les mains.
- La séquence *close* (figure 79) est identique à la séquence *wide* mais nous avons utilisé un objectif avec une focale plus courte se qui rapproche le patient. Nous avons également réalisé deux zones d'intérêt sur cette séquence en faisant en sorte de les resserrer le plus possible.

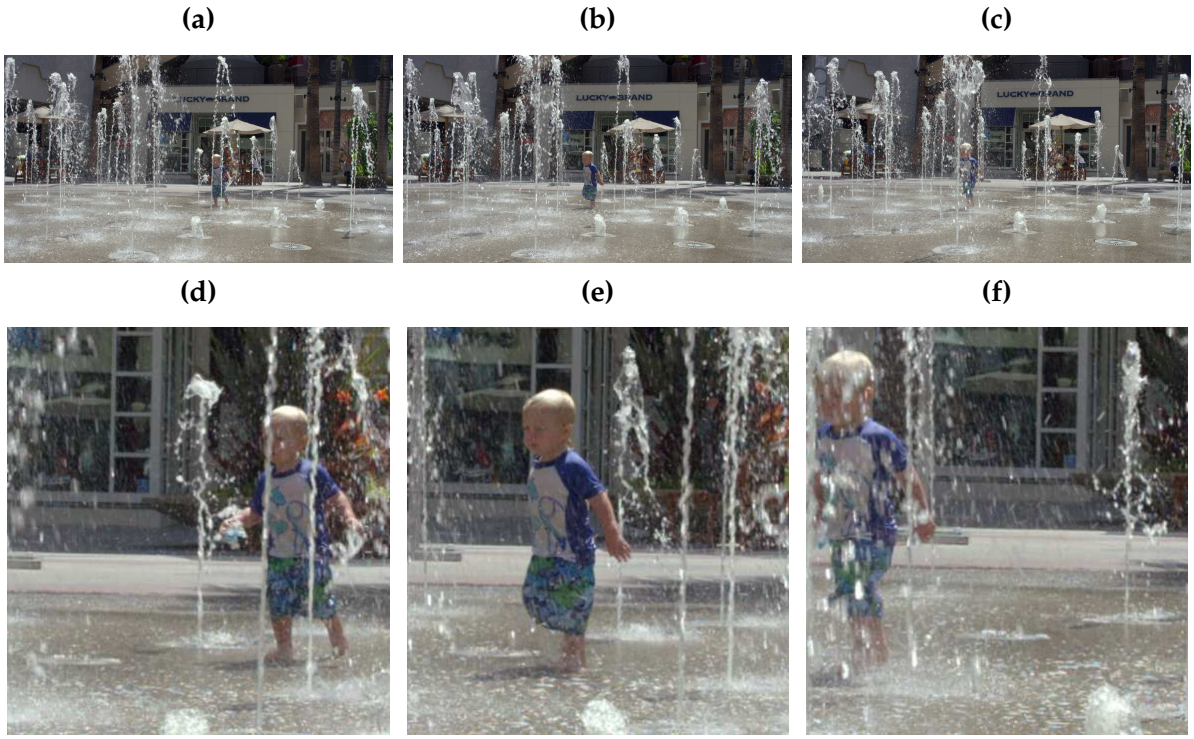



**Figure 75.** Extrait de la séquence d'image *toddler* et sa zone d'intérêt (image 1, 50 et 100)

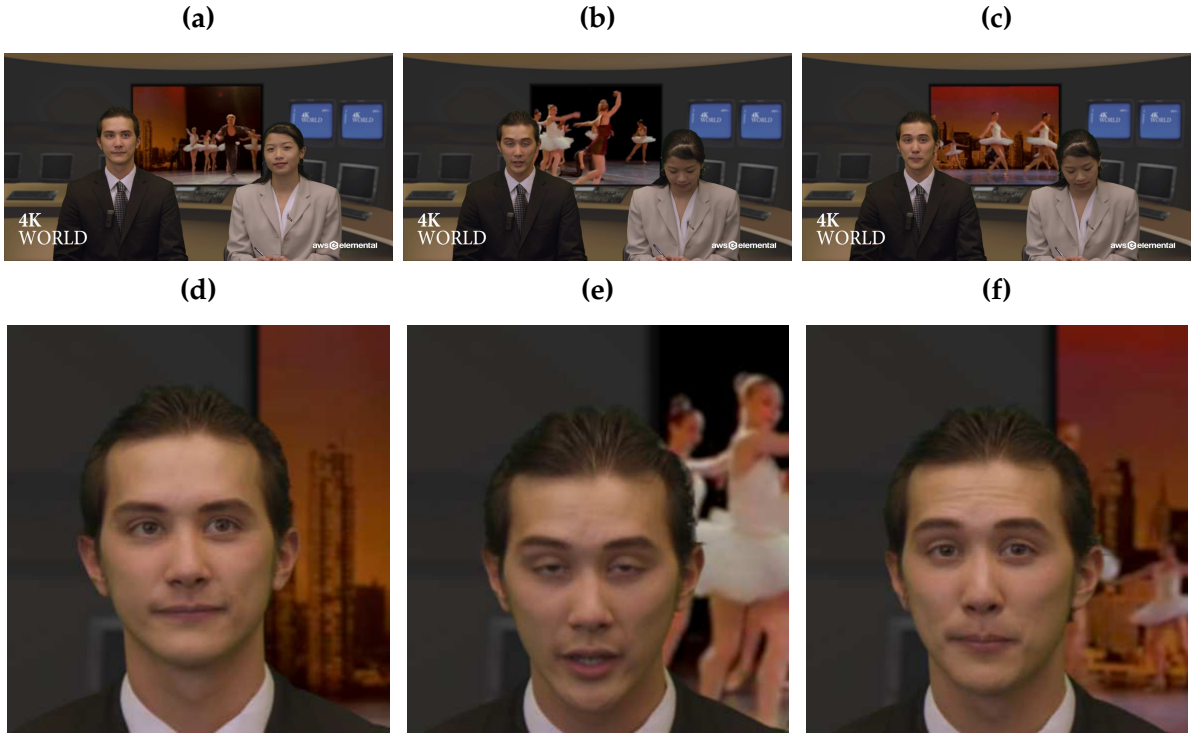



**Figure 76.** Extrait de la séquence d'image *news* et sa zone d'intérêt (image 1, 100 et 200)

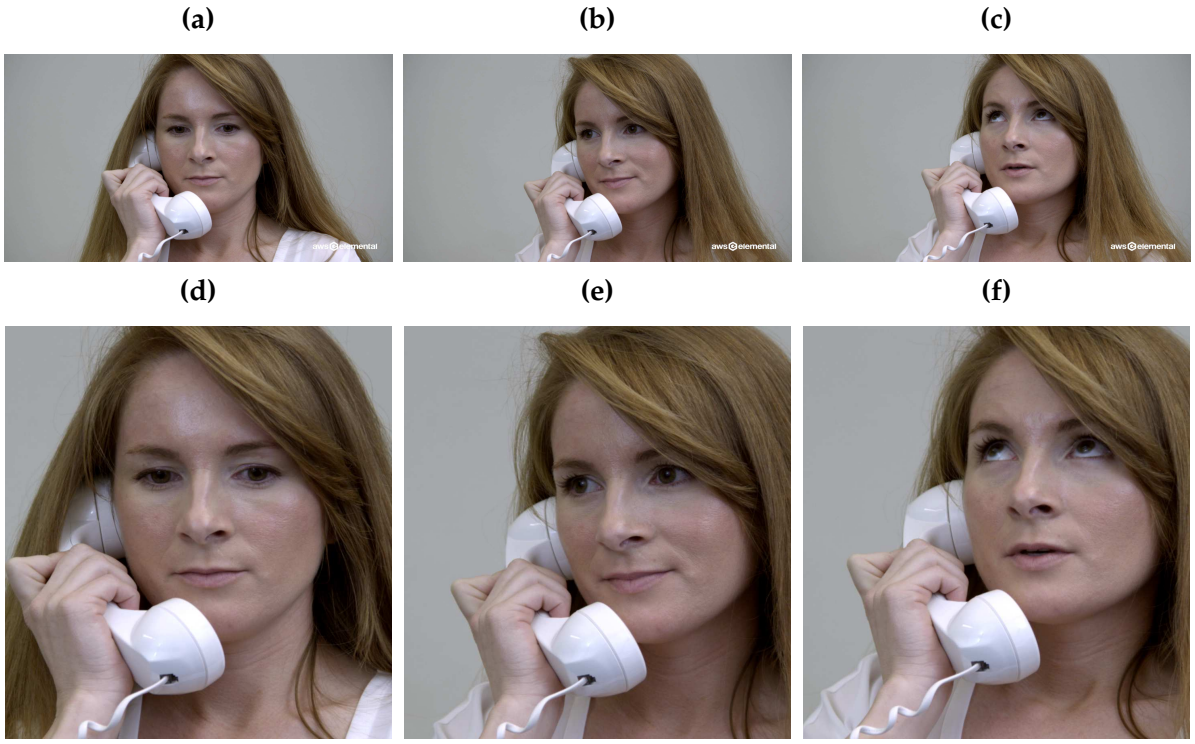



**Figure 77.** Extrait de la séquence d'image *suzie* et sa zone d'intérêt (image 1, 150 et 300)

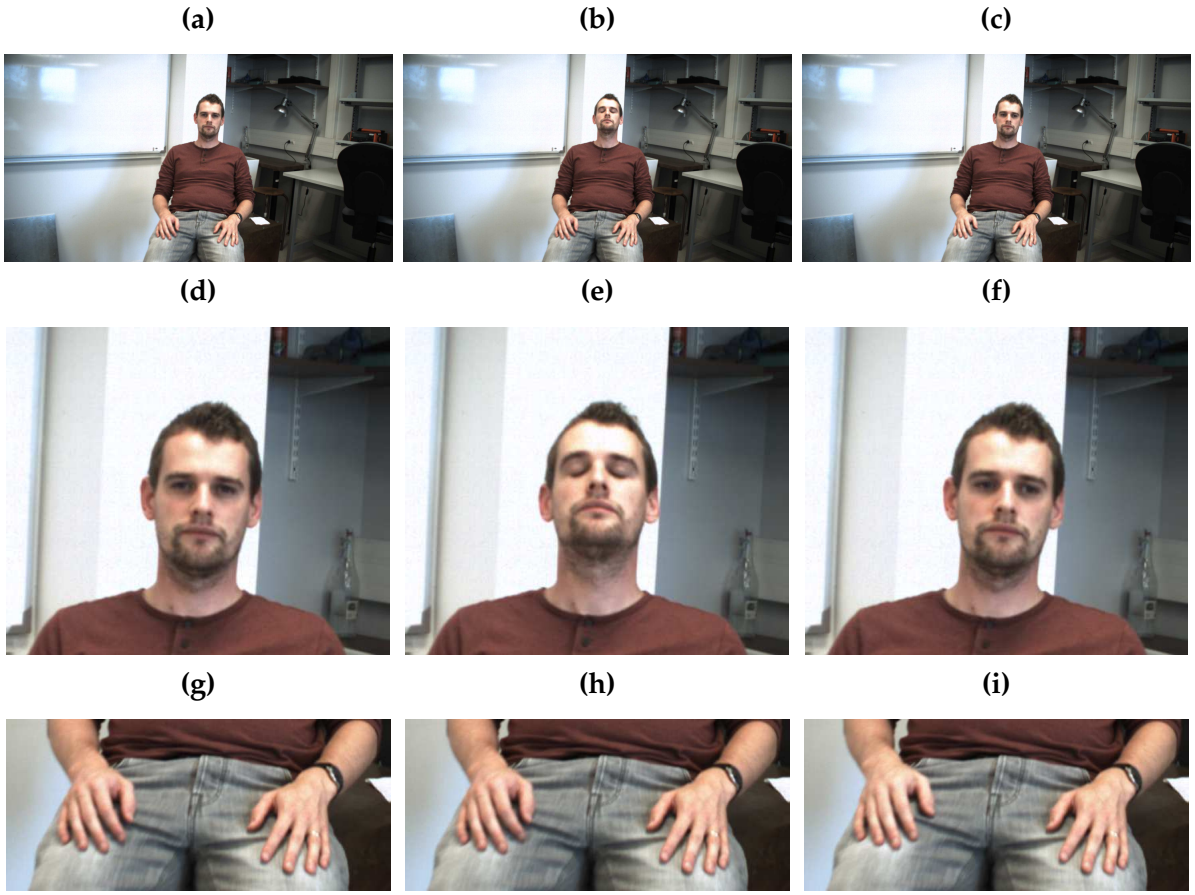



**Figure 78.** Extrait de la séquence d'image *wide* et ses deux zones d'intérêt (image 1, 250 et 500)

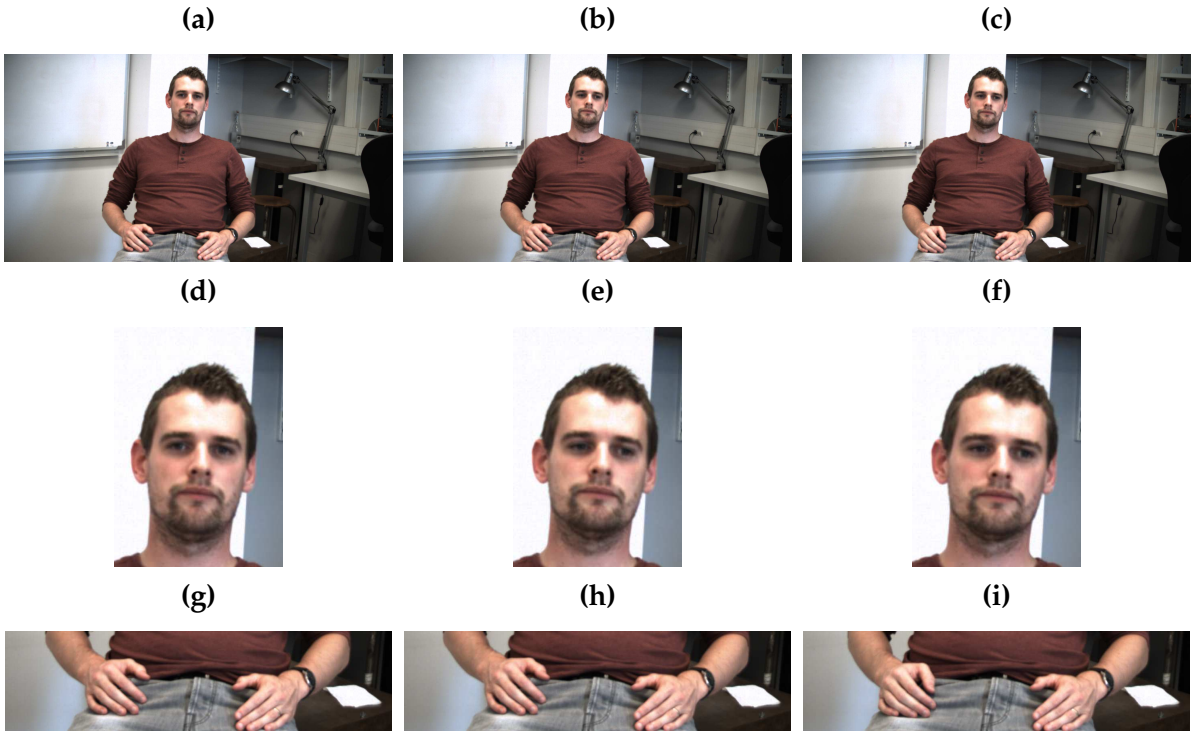



**Figure 79.** Extrait de la séquence d'image *close* et ses deux zones d'intérêt (image 1, 250 et 500)

Les vidéos au format 4K sont redimensionnées dans un format 1920 par 1024. Nous les avons ensuite converties en Bayer avant de démarrer les expérimentations. De cette façon, toutes les séquences d'images de base sont des images brutes de taille 1920 par 1024 au format Bayer. Nous regroupons les séquences en deux groupes, les séquences généralistes (toddler, news et suzie) et les séquences examens (wide et close). Les séquences examens sont réalisées avec le matériel de l'implémentation et essayent de représenter au mieux les séquences que nous pourrions avoir dans un examen classique. Pour ces deux raisons, nous donnerons plus de poids au résultat trouvé sur ces séquences.

Dans un premier temps, nous étudierons les paramètres du codeur pour ensuite comparer ROI-Waaves aux codeurs de l'état de l'art. Pour rappel, nous voulons étudier le placement de la différence dans le codeur, l'influence du pas de rafraîchissement des images de référence, des méthodes d'encodage de la zone d'intérêt et de la taille des blocs pour le parcours des coefficients. Toutes les courbes d'évaluation seront présentes dans l'annexe C. Seules les courbes que nous voulons souligner seront reportées dans cette section.

**Évaluation des paramètres de ROI-Waaves**

Dans un premier temps, nous voulons évaluer l'impact de la position de la différence sur l'algorithme. En analysant les courbes comparant la différence avant et après, nous remarquons deux tendances. La première tendance est vue sur les séquences généralistes qui montrent que sur l'image entière, nous obtenons de meilleures performances quand

la différence est réalisée après la transformée en ondelettes avec des performances comparable sur les ROI. La deuxième tendance (figure 80) est vue sur les séquences venant de la caméra dans laquelle nous remarquons que les performances sont globalement meilleures notamment dans les bas débits quand nous réalisons la différence avant la DWT. Les vitesses d'encodage des deux configurations sont disponibles dans la figure 81. Comme attendu, les deux configurations ne présentent pas de différence de temps de traitement car il s'agit d'un déplacement d'un traitement et non d'une modification. De ce paramètre, nous concluons que la différence avant la DWT est meilleure sur nos séquences d'images mais pas nécessairement pour toutes. Nous utilisons pour la suite la différence avant la DWT.

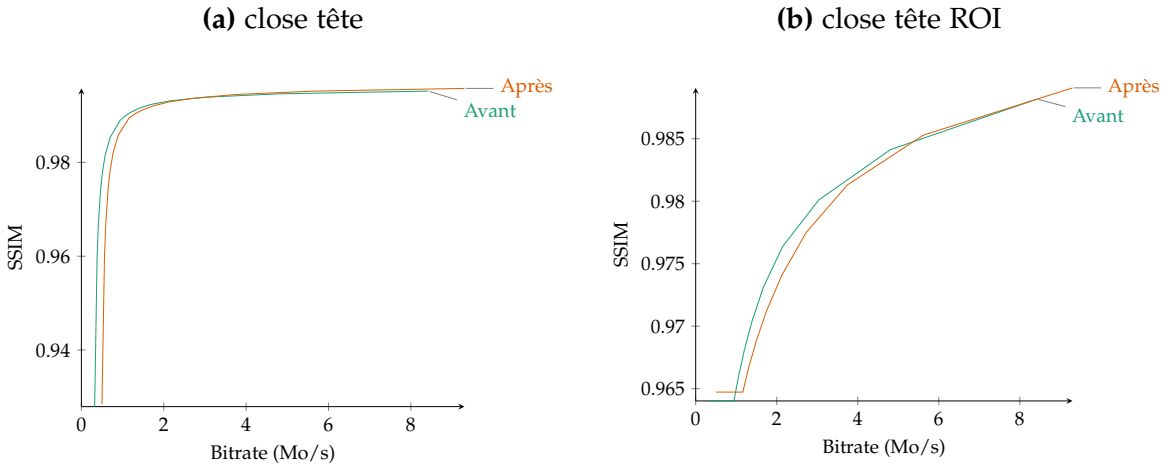



**Figure 80.** Évaluation de la position de la différence sur la séquence close avec comme zone d'intérêt la tête


**Figure 81.** Vitesse d'encodage de ROI-Waaves avant et après la différence

Dans un second temps, nous désirons évaluer l'impact du pas de rafraîchissement entre deux images de référence. Pour ce faire, nous avons fait varier le pas en utilisant les valeurs 1, 5, 10, 15 et 20. En faisant ainsi, nous avons remarqué que le résultat était différent sur les séquences généralistes et les séquences examens. Pour les séquences généralistes, comme le montrent les résultats de la séquence suzie (figure 82), les deux pas donnant les meilleures performances sont le pas 1 et 5 (1 à haut débit et 5 à bas débit). Cela est dû au grand changement entre deux images entraînant une augmentation de l'entropie et donc une augmentation du débit. Au contraire sur les

séquences examens (figure 83), le résultat est plus serré. Seul le pas de 1 est moins bon que le reste. Pour les vitesses de compression (figure 84), hormis le pas 1, toutes les configurations sont comparables. Le pas 1 est plus rapide car il y a moins de traitements spéciaux à réaliser. Avec cette analyse, nous proposons de faire un compromis en utilisant le pas de 5 par défaut.

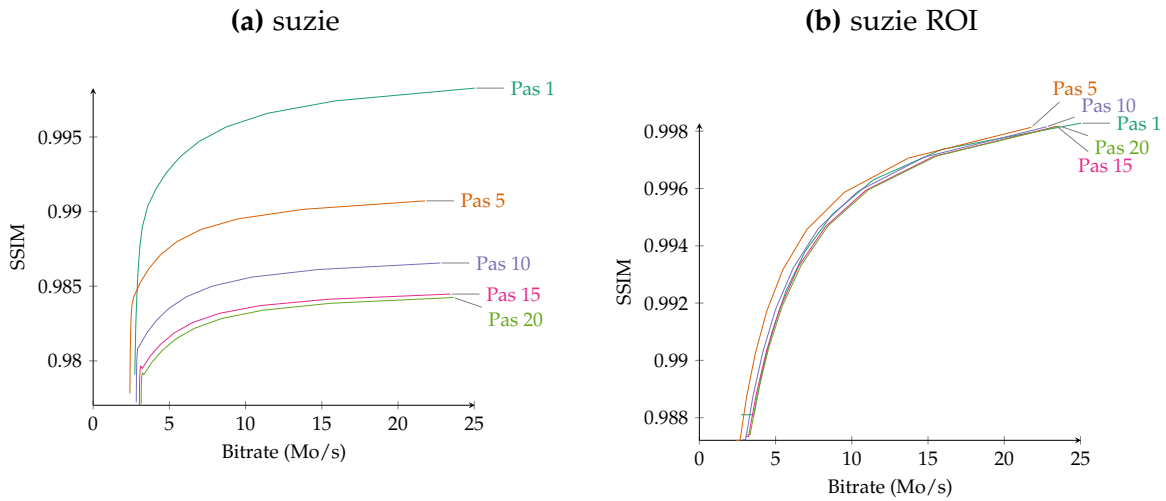



**Figure 82.** Évaluation du pas de rafraîchissement sur la séquence suzie

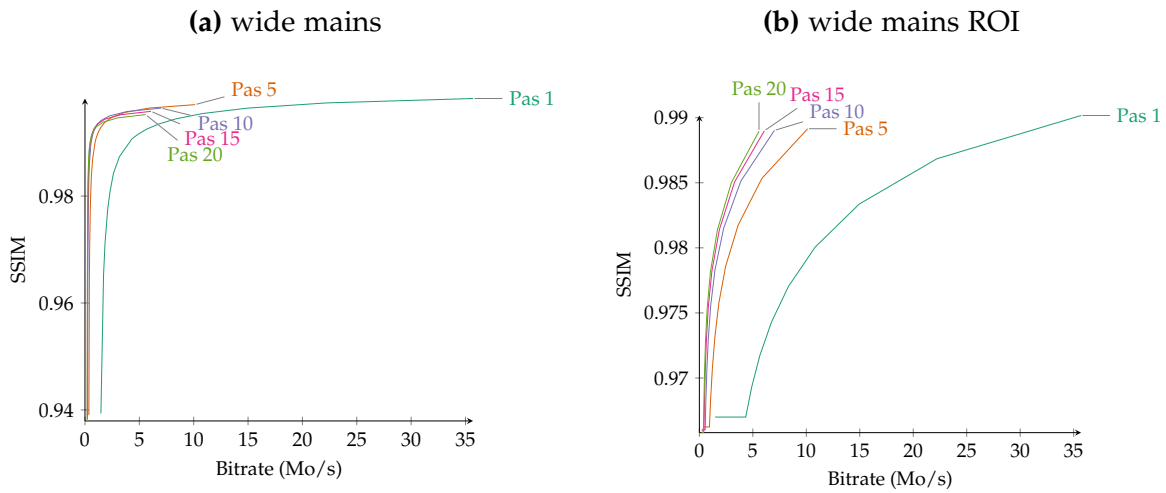



**Figure 83.** Évaluation du pas de rafraîchissement sur la séquence wide avec comme zone d'intérêt les mains

Dans un troisième temps, nous voulons déterminer quel est le meilleur parcours en faisant varier la taille des blocs parcourus. La conclusion de cette étude est plus difficile à porter. En ne regardant que les résultats sur la zone d'intérêt, la plus petite


**Figure 84.** Vitesse d'encodage de ROI-Waaves en fonction du pas de rafraîchissement

taille de blocs (8 par 8) est la meilleure. Cependant, en regardant sur l'image entière, les résultats sont très comparables pour les séquences examens et montre la taille 128 par 128 meilleure sur les séquences généralistes par exemple sur *news* (figure 85). En comparant, les vitesses de compressions (figure 86), nous remarquons que les grandes tailles de blocs sont plus rapides à compresser que les petites. Nous pensons que cela vient d'effet de cache ou la lecture en mémoire est plus efficace sur des grandes lignes. Cependant, en gardant à l'esprit une implémentation matérielle du codeur, les transactions mémoires en rafale se font sur des transactions de 512 octets. La taille idéale d'une ligne pour cette taille de transactions est 16 par 16 (car 16 pixels de 32 bits égale 512 bits). En considérant cela et en comparant la taille 16 par 16 avec la taille 8 par 8, nous avons remarqué que le gain en performance n'était pas assez significatif pour faire une grande différence et pouvait par contre apporter une meilleure utilisation des transactions mémoires en matériel. La taille que nous conservons est 16 par 16.

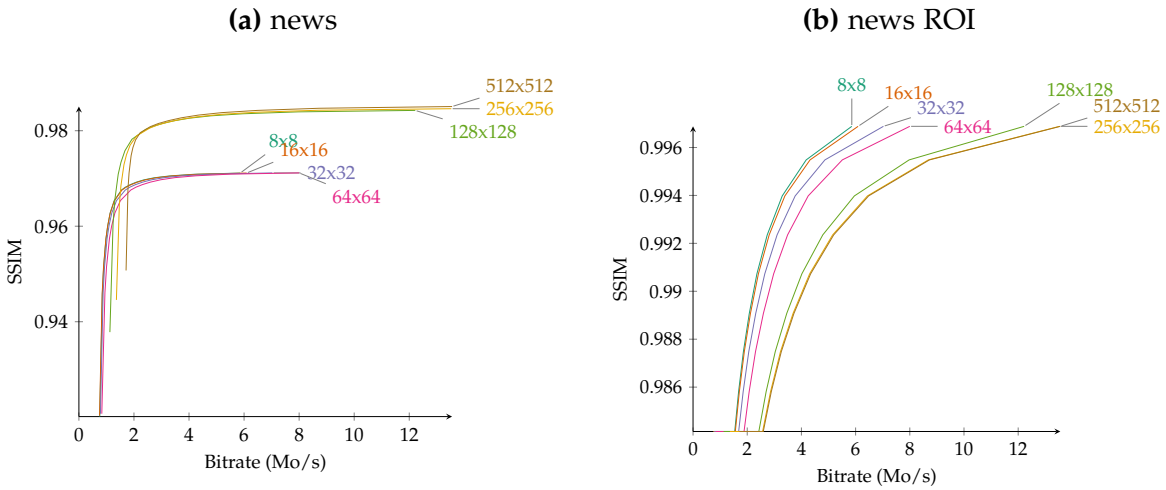



**Figure 85.** Évaluation de la taille des blocs de parcours sur la séquence news

Dans un dernier temps, la méthode d'encodage des masques de ROI impacte que très légèrement les performances d'encodage. Nous remarquons que la différence entre

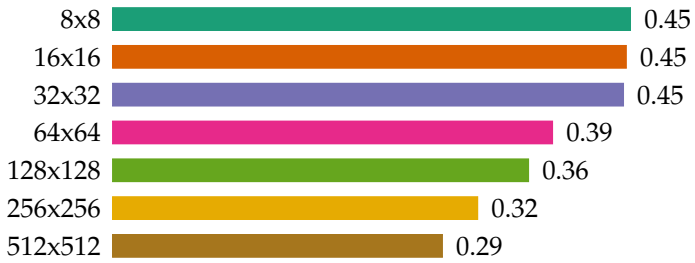
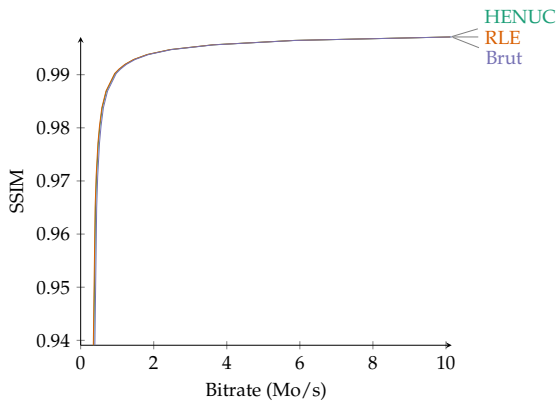



**Figure 86.** Vitesse d'encodage de ROI-Waaves pour différentes tailles de bloc

HENUC et RLE est très faible (figure 87). En vitesse d'encodage cependant, la méthode HENUC est relativement plus lente que l'encodage brut et RLE (figure 88). Nous concluons que nous utilisons l'encodage RLE qui fait le bon compromis entre les deux critères. Cependant, le choix de la méthode ne change que très peu les performances du codeur.

**(a)** wide mains




**(b)** wide mains ROI

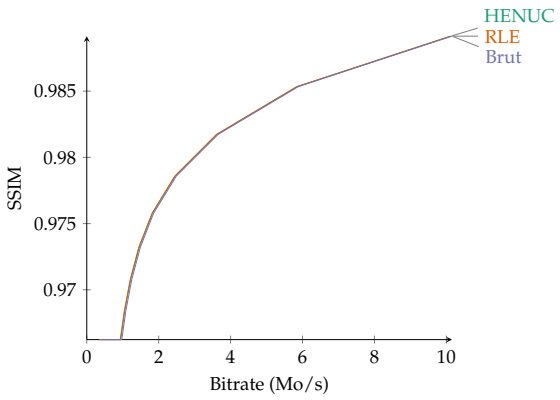



**Figure 87.** Évaluation de la méthode d'encodage du masque de ROI sur la séquence wide avec comme zone d'intérêt les mains

**Évaluation de ROI-Waaves avec les codeurs de l'état de l'art**

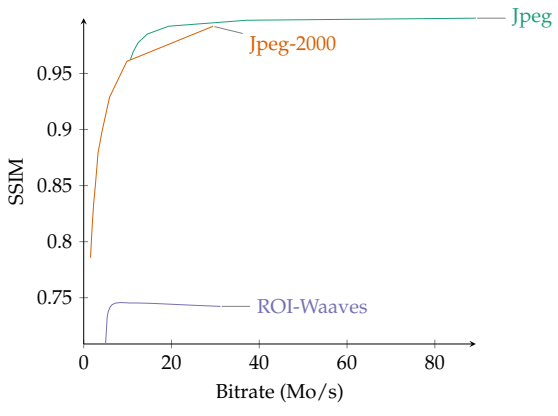
Nous avons séparé notre analyse en trois étapes, une première étape dans laquelle nous comparons ROI-Waaves aux codeurs d'image fixe, une seconde étape dans laquelle nous comparons ROI-Waaves aux codeurs vidéo et une dernière étape dans laquelle nous comparons ROI-Waaves à Waaves. Pour les deux premières étapes nous ne


**Figure 88.** Vitesse d'encodage de ROI-Waaves pour différentes méthodes d'encodage de la ROI

comparerons pas les vitesses de compression car elles ne sont pas significatives. En effet, les implémentations des codeurs standards sont beaucoup plus optimisées et ne sont pas comparables aux performances de Waaves et ROI-Waaves.

Premièrement, ROI-Waaves se montre plus mauvais que JPEG et JPEG-2000 ou équivalent quand nous réalisons une analyse de qualité sur l'image entière. Ce phénomène est amplifié sur les séquences généralistes, notamment sur la séquence *toddle* (figure 89), quand nous regardons les performances en qualité sur l'image entière. Cependant, quand l'analyse est concentrée sur la zone d'intérêt nous remarquons que la qualité est meilleure que les codeurs JPEG et JPEG-2000. De plus, sur les séquences examens, comme nous pouvons le voir sur la figure 90, les performances sur l'image entière sont supérieures et encore améliorées sur la zone d'intérêt. Nous pouvons de plus constater sur les deux figures de résultat des zones d'intérêt des courbes caractéristiques du codeur ROI-Waaves. En effet, celles-ci possèdent un palier dans ces bas débits équivalent au gain apporté par la compression du reste de l'image en conservant la qualité de la zone d'intérêt constante.

**(a)** toddler




**(b)** toddler ROI

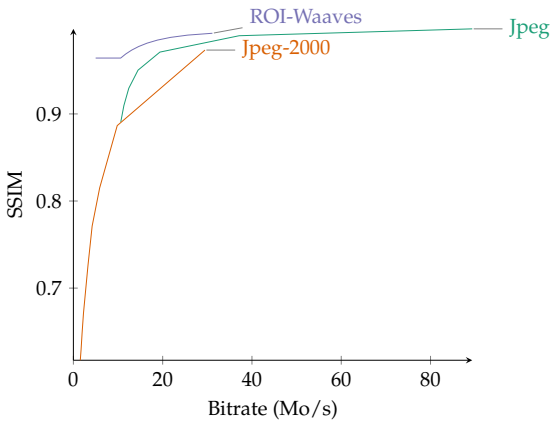



**Figure 89.** Comparaison des performances de JPEG, JPEG-2000 et ROI-Waaves sur la séquence toddler

Deuxièmement, en comparant ROI-Waaves aux codeurs H264 et VP8, nous pouvons

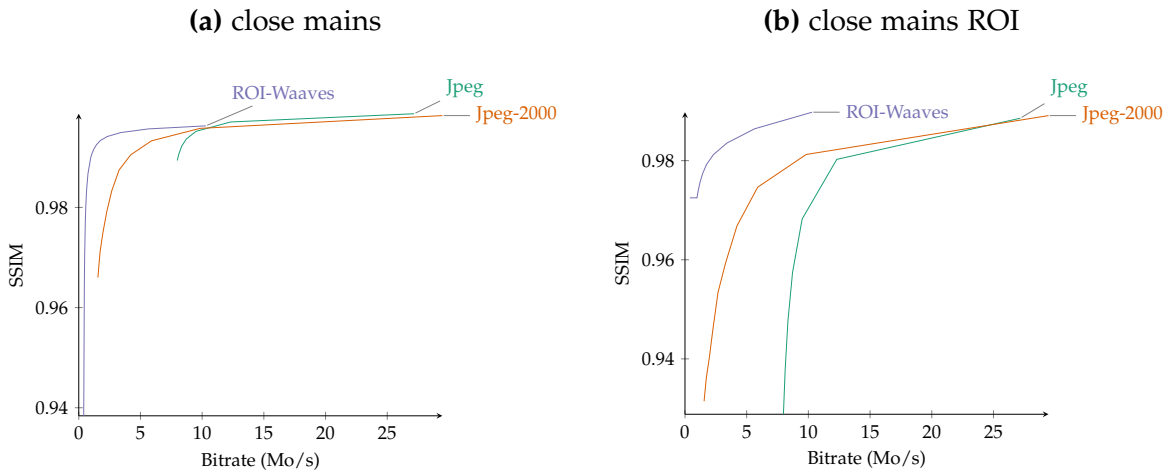



**Figure 90.** Comparaison des performances de JPEG, JPEG-2000 et ROI-Waaves sur la séquence close avec comme zone d'intérêt les mains

voir que ces derniers sont meilleurs sur toute l'image et sur toutes les séquences. Ces résultats sont attendus dans ce sens. La différence est d'ailleurs aggravée quand la séquence présente des grands mouvements comme dans *toddler* (figure 91). Cependant, nous nous intéressons maintenant à la qualité de la zone d'intérêt pour savoir si la dégradation est acceptable ou non. Sur une majorité des séquences notamment sur la séquence *close hand* (figure 92), le codeur ROI-Waaves est comparable dans les bas débits grâce au palier introduit par la zone d'intérêt. Il est même, à notre plus grande surprise, meilleur dans la zone d'intérêt sur la séquence *toddler* sur laquelle il a été le plus mauvais en qualité sur le reste de l'image.

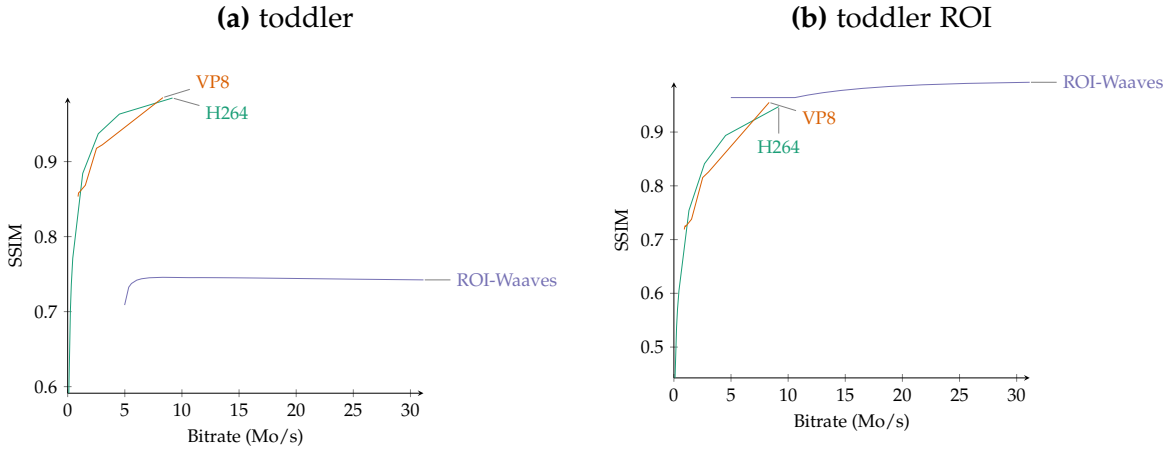



**Figure 91.** Comparaison des performances de H264, VP8 et ROI-Waaves sur la séquence toddler

Troisième et dernièrement, nous voulons comparer ROI-Waaves à Waaves son petit frère. Là encore, nous obtenons des résultats divergents entre les séquences généralistes

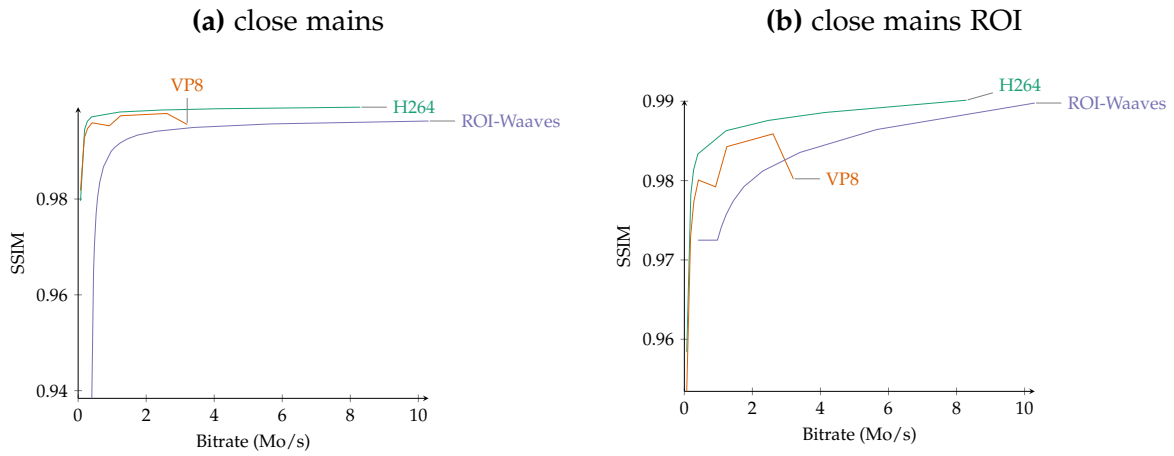



**Figure 92.** Comparaison des performances de H264, VP8 et ROI-Waaves sur la séquence close avec comme zone d'intérêt les mains

et les séquences examens. Pour les séquences généralistes, par exemple *news* (figure 93), nous pouvons voir que la qualité générale de l'image est moins bonne pour ROI-Waaves que pour Waaves. Le constat est inversé quand nous regardons uniquement la zone d'intérêt. Nous pouvons alors conclure que les zones d'arrière-plan sont compressées de manière plus agressive sur ROI-Waaves afin de faire bénéficier les zones d'intérêt. Pour les séquences examens (figure 94), le constat est que ROI-Waaves propose des meilleures performances sur toute l'image. Dans la zone d'intérêt, ROI-Waaves est capable de conserver jusqu'à 0.1 point de SSIM à débit équivalent. Pour finir, la figure 95 montre les vitesses de compression en images par seconde des deux codeurs. De cette figure, nous pouvons établir que la nouvelle chaîne de compression est 3,5 fois plus rapide à compresser que la chaîne Waaves en n'intervenant que sur l'aspect algorithmique.

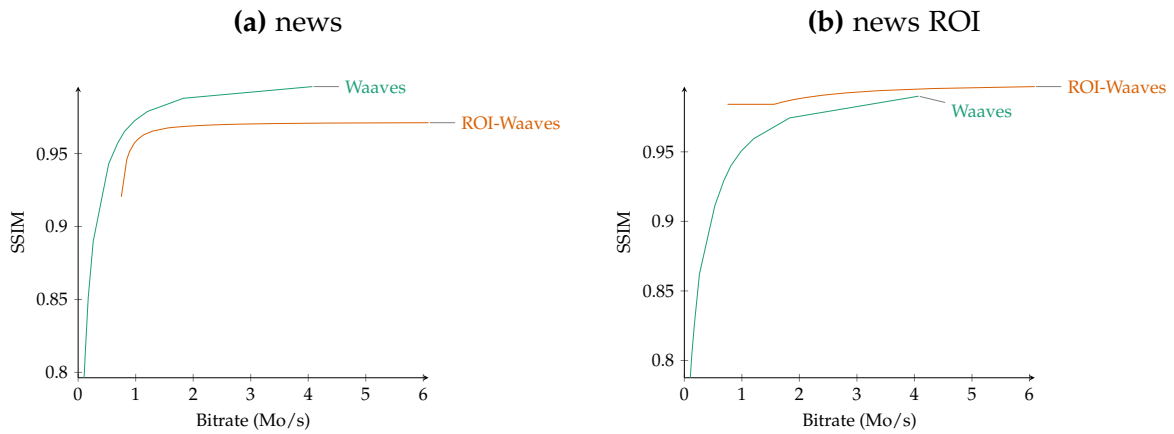



**Figure 93.** Comparaison des performances de Waaves et ROI-Waaves sur la séquence news

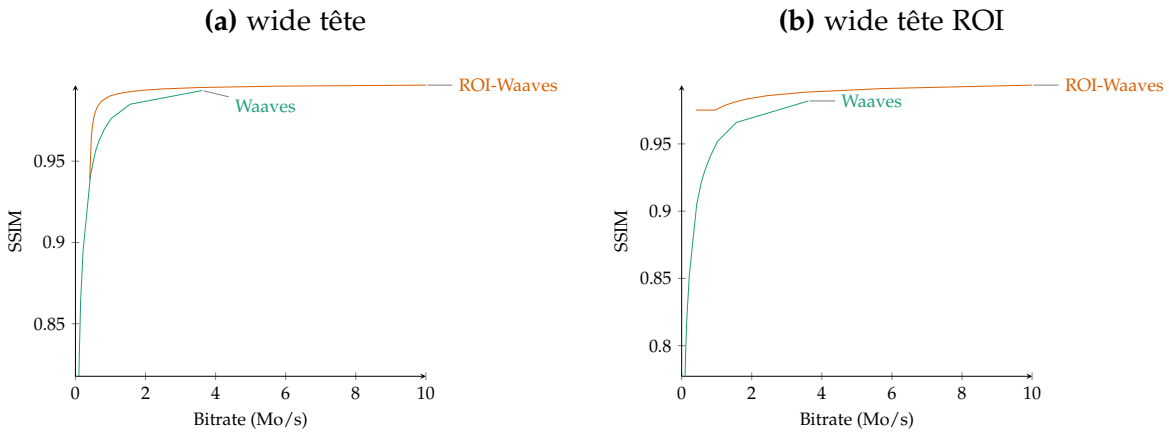



**Figure 94.** Comparaison des performances de Waaves et ROI-Waaves sur la séquence wide avec comme zone d'intérêt la tête


**Figure 95.** Vitesse d'encodage de ROI-Waaves et de Waaves

**Conclusion**

Dans cette section, nous avons déterminé que la complexité du codeur ROI-Waaves était $\mathcal{O}(n \times log(n))$. En comparaison, le codeur Waaves est de complexité $\mathcal{O}(n^2)$. La réduction de la complexité est de plus en plus importante que la taille des images augmente. Ensuite, nous avons déterminé par l'évaluation les paramètres que nous utiliserons pour le codeur ROI-Waaves, la différence avant la DWT, le pas de rafraîchissement de 5, la méthode d'encodage RLE du masque de la ROI et la taille de 16 par 16 des blocs de parcours. Pour finir, nous avons montré toujours par l'évaluation que ROI-Waaves était la plupart du temps meilleur que JPEG, JPEG-2000 et Waaves et toujours meilleur sur les zones d'intérêt. Nous avons également déterminé que dans certains cas, il est capable de reproduire des zones d'intérêt de même qualité à des débits équivalents que H264 et VP8.

# 3 Système

Dans cette dernière section d'évaluation, nous évaluons l'intégration système des deux démonstrateurs présentés. Nous évaluerons principalement l'aspect quantitatif des deux implémentations du point de vue des ressources nécessaires dans le FPGA, des performances d'encodage et du coût de l'implémentation.

## 3.1 Implémentation logicielle

D'un point de vue qualitatif, l'implémentation logicielle est entièrement fonctionnelle. Elle émule entièrement le boitier final et permet de faire l'acquisition de données d'un examen EEG par une console de prévisualisation afin de l'envoyer à des fins de téléexpertise. Elle a été testé par le LNE[1] dans le but d'obtenir les certifications électriques et électromagnétiques afin de pouvoir être utilisé dans un cadre clinique. Cette procédure nous a permis de nous assurer que notre boitier était transparent pour un utilisateur lambda (le technicien du LNE). Elle permet aussi de valider le bon fonctionnement du boitier dans des conditions plus ou moins extrêmes.

Pour l'évaluation quantitative, nous commençons par évaluer les ressources utilisées par les différentes IP intégrées dans le FPGA. Les chiffres présents dans cette évaluation sont les résultats de la synthèse de l'implémentation envoyée au LNE. La synthèse a été effectuée sous Quartus 14.1. Ces résultats sont présents dans le tableau 9.

**Tableau 9.** Résultat de synthèse des IP FPGA pour l'implémentation logicielle

| Composant | Logiques (ALUT) | Registres | Mémoires (bits) |
|---|---|---|---|
| *Riffa* | 16990 (7,45%) | 17576 | 1064366 (6,06%) |
| Nios | 3317 (1,45%) | 2709 | 1025664 (5,84%) |
| Video | 12567 (5,51%) | 15714 | 380975 (2,17%) |
| Physio | 999 (0,43%) | 905 | 286656 (1,63%) |
| Audio | 118 (0,05%) | 85 | 106854 (0,60%) |
| PCI-E | 819 (0,35%) | 503 | 32 (0%) |
| Timestamp | 143 (0,06%) | 427 | 1024 (0%) |
| Autre | 845 (0,37%) | 592 | 0 (0%) |
| Total | 35798 (15,7%) | 38511 | 2865571 (16,33%) |

L'IP occupant la majeure partie du FPGA est le module *Riffa* permettant la communication avec le processeur du PC. Ensuite vient l'îlot NIOS qui utilise à lui seul plus de 5,5% des mémoires sur puce du FPGA. Cette utilisation de mémoires sur puce est due à l'utilisation de mémoire sur puce comme mémoire vive du NIOS (960 Kbits). Cette utilisation pourrait être supprimée en utilisant la mémoire DDR2 présente sur le kit de développement. Cependant, par simplicité et comme cela n'était pas bloquant, nous avons conservé cette implémentation. Nous trouvons ensuite les IP du pipeline vidéo composé principalement de logique. La mémoire utilisée par cette IP vient des fifos de tampon entre les blocs ainsi que des LineBuffer présents dans les IP de conversion Bayer vers RGB. Près de la moitié des ALUT utilisées de ce pipeline proviennent du contrôleur DDR d'Altera (4933 ALUT). Le pipeline physio utilise peu de logique mais fait intervenir des blocs mémoires afin de réaliser les mémoires tampons. De même que pour les mémoires du NIOS, nous aurions pu basculer ces tampons en mémoire vive. Le pipeline le plus léger est celui gérant l'audio car le besoin en tampons est plus faible

1. LNE : Laboratoire national de métrologie et d'essais

que pour les deux autres modalités. Les IP PCI-Express d'Altera et de timestamp sont négligeables dans le design. Pour finir, d'autres IP sont dans le design afin de gérer le ventilateur de la carte, les resets, les PLL etc.

Au final, la totalité des ressources utilisées sur le FPGA représente moins de 16% des ALUT et moins de 16,5% des mémoires sur puce du FPGA Stratix IV. Cette utilisation de blocs mémoires pourrait être plus diminuée en basculant les IP vers la mémoire DDR2. Nous avons évalué que faire cette optimisation pourrait économiser 6% de blocs mémoires sur le Stratix IV.

**Tableau 10.** Tableau des coûts des composants de l'implémentation logicielle

| **Composant** | **Prix** |
|---|---|
| PC | 700 € |
| SSD Intel P3600 | 2700 € |
| Caméra | 1110 € |
| Objectifs | 99 € |
| Câble *CameraLink* | 175 € |
| FPGA Altera DE4 | 2,500 € |
| Carte d'adaptation *HSMC/CameraLink* | 210 € |
| Carte d'adaptation HSMC/GPIO | 46 € |
| Total | 7540 € |

D'un point de vue du coût matériel utilisé pour faire l'implémentation logicielle, nous avons fait une liste des composants dans le tableau 10. Cette implémentation coûte 7540 €. Le poste le plus onéreux est le PC à cause de l'achat d'un SSD Intel de 1,2 To. Ceci renforce l'idée d'avoir un encodeur matériel de ROI-Waaves capable de traiter en temps réel le flux vidéo car cela permettrait d'enlever cette dépense.

## 3.2 Implémentation matérielle

Nous rappelons avant tout que l'implémentation matérielle n'est pas complète et ne prend pas en compte tous les aspects de l'intégration. De ce fait, il est possible que certaines valeurs soient plus basses qu'une implémentation complète, notamment sur les utilisations de ressources ou plus hautes pour les performances. Les performances d'un système complet sont trop aléatoires pour pouvoir être estimées à l'avance. Nous évaluerons principalement la version matérielle du codeur ROI-Waaves afin d'évaluer s'il peut, après une intégration complète, suivre la cadence de 100 images par seconde et tenir dans un FPGA.

Le tableau 11 met en avant l'utilisation de ressource de la synthèse des IP composant le codeur ROI-Waaves. Cette synthèse a également été réalisée sur le logiciel Quartus 14.1 d'Altera. Il faut bien prendre en compte que cette synthèse n'est pas celle d'un système complet et fonctionnel. De ce fait il manque certains blocs notamment des DMA. Par ailleurs, il est a noter que les blocs les plus coûteux constituant le codeur

final sont présents dans cette synthèse. Pour les IP de BayerYUV et de Quantification, elle utilise très peu de ressources que ce soit mémoire ou logique. La mémoire utilisée par BayerYUV correspond au linebuffer de 1920*8 = 15360 bits. La DWT, développée dans le cadre de la thèse de Ibraheem Mohammed Shaaban [Shaaban, 2017], utilise beaucoup de ressources pour réaliser le traitement des pixels en parallèle et beaucoup de mémoire pour pouvoir stocker une bonne partie de l'image et donc traiter en une fois la passe verticale et horizontale de la transformée en ondelettes.

**Tableau 11.** Résultat de synthèse des IP FPGA de ROI-Waaves

| Composant | Logiques (ALUT) | Registres | Mémoires (bits) |
|---|---|---|---|
| BayerYUV | 586 (0,25%) | 233 | 23 008 (0,13%) |
| DWT | 20 671 (9%) | 21 489 | 1 254 560 (7,1%) |
| Quantification | 3 548 (1,5%) | 1 804 | 0 (0%) |
| Extraction | 44 027 (19,3%) | 28 940 | 632 440 (3,6%) |
| HENUC | 6 638 (2,9%) | 3 760 | 1 459 208 (8,3%) |
| Total | 75 470 (32,95%) | 56 235 | 3 369 216 (19,13%) |

L'IP d'extraction est principalement composée des 32 plans de bits extraits. Comme dit dans le chapitre V, l'utilisation de cette IP provient de la mise en parallèle de 32 logiques similaires pour extraire les 32 flux de plan de bits en parallèle. Dans le cas, où cette utilisation deviendrait trop importante pour une implémentation, il y a la possibilité, au coût de lecture mémoire plus fréquente, d'extraire moins de plans de bits à la fois. Il est par exemple possible de n'extraire que 8 plans de bits à la fois ce qui divise par 4 les ressources utilisées par l'IP. Cela revient à faire un compromis entre la vitesse de traitement et la surface de silicium. Dans notre cas, nous considérons que cette utilisation reste acceptable. De plus, ces ressources ne dépendent pas de la taille de l'image ce qui veut dire que l'IP ne changerait pas de taille si la résolution de l'image venait à grandir. Dernièrement, l'IP d'HENUC est la plus grande en utilisation de mémoires sur puce. Ce sont les précalcules des valeurs utilisées par $EBI_{512}$, par $EBI_{64}$ et par LD avec respectivement 565248, 286720 et 573440 bits.

À titre de comparaison, dans la thèse de Yuhui [Bai, 2014], deux IP ont étés complètement implémentées ASWD et HENUC. Les algorithmes ayant changés légèrement, il faut prendre cette comparaison avec prudence. Nous pouvons d'ailleurs que comparer l'IP HENUC car l'ASWD n'est plus présent dans notre codeur. L'IP HENUC implémenté dans la thèse de Yuhui utilise 7 298 ALUT, 2 196 registres et 1 779 872 bits mémoires. Avec la nouvelle version de l'algorithme et la nouvelle implémentation, nous avons donc pu réduire de 9% la logique, de 41% les registres et de 18% les bits mémoires. C'est donc une réduction non négligeable de ressource du FPGA. Il est cependant assez dur de savoir si cette réduction est due au changement de l'algorithme ou à la nouvelle implémentation.

Au total, l'estimation du composant ROI-Waaves représente près de 33% des

ressources logiques et 20% de la mémoire du Stratix IV. En rajoutant les IP d'acquisition et de synchronisation et en enlevant les IP de *riffa* et de PCI-E car plus utiles, cela porte l'utilisation sur un Stratix IV a 40,85% (environ 91200 ALUT) de logique et 29.4% (environ 4Mbits) de mémoires sur puce ce qui reste largement gérable pour les outils de placement routage. À titre d'exemple, la cible ARRIA V ST Soc contient 174 000 ALUT et 22Mbits de mémoires sur puce. Il serait donc suffisant de porter le projet sur cette cible.

D'un point de vue des performances des IP, nous avons mesuré individuellement, pour chaque IP développée, le temps de traitement d'une image de 1920 par 1080. De ce temps, nous sommes capables d'en déduire le nombre d'images par seconde que l'IP peut encoder. Nous avons ensuite, à partir de ce nombre d'images par seconde, déduit le débit de traitement en Mo/s que l'IP est capable de réaliser. Pour finir, nous avons rajouté à titre indicatif les fréquences de fonctionnement de chaque composant. Le tableau 12 résume nos évaluations sur les différents composants. Pour rappel, pour encoder 100 images par seconde il faut traiter une image en moins de 10 millisecondes ou avoir un débit supérieur à 207 Mo/s. Dans ce tableau, les composants remplissant la contrainte de la cadence de fonctionnement sont en verts et ceux qui ne la remplissent pas sont en rouge. Nous colorons en orange ceux qui nécessitent davantage de développement mais qui ont un bon potentiel pour atteindre la cadence.

**Tableau 12.** Résultat de timing des IP FPGA de ROI-Waaves

| Composant | Temps par image (ms) | FPS | Débit (Mo/s) | Fréquence (MHz) |
|---|---|---|---|---|
| Nios SW | 174,80 | 0,005 | 0,0112 | 150 |
| Intel | 454,5 | 2,2 | 4,32 | 2200 |
| BayerYUV [1] | 7,5 | 131,6 | 258,8 | 258,87 |
| DWT | 8,31 | 120,2 | 236,3 | 612 |
| Quantification [1] | 9,33 | 107,1 | 210,7 | 210,7 |
| Extraction | 3,75 | 266,8 | 524 | 137,4 |
| HENUC | 19,39 | 51,6 | 101,4 | 143,9 |
| Total séquentiel | 48,28 | 20,71 | 42,9 | - |
| Total pipeline | 19,39 | 51,6 | 101,4 | - |

Dans la première ligne, nous avons le résultat d'encodage d'une image entière sur un processeur NIOS et un processeur Intel. Aucun des deux processeurs ne suit la cadence de traitement nécessaire. Ensuite, hormis l'IP HENUC, tous les composants du codeur ROI-Waaves sont capables d'encoder un flux vidéo à la cadence de 100 images par seconde.

L'implémentation de HENUC proposée dans cette thèse ne remplit pas le critère de vitesse d'encodage. Dans cette version, il est possible d'encoder avec une vitesse de plus de 50 images par seconde. Cependant, nous avons développé dans la constitution même

1. Estimation sans DMA.

de HENUC une manière de paralléliser l'encodage d'un arbre HENUC en plusieurs sous-arbres. En exploitant deux ou trois fois ce parallélisme, nous pouvons espérer atteindre la cadence de 100 images par seconde. Les développements nécessaires à la réalisation d'un tel codeur restent à faire de manière à estimer la cadence en exploitant ce parallélisme. Nous plaçons donc ce composant en orange.

Nous avons combiné ces composants en deux versions : une première ou une seule IP fonctionne à la fois qui constitue un total séquentiel et un ou toutes les IP travaillent en parallèle qui constitue un total pipeline. Le total séquentiel est capable d'encoder à 20 images par seconde alors que le total pipeline bridé par HENUC encode à 51,6 images par seconde. Cette dernière version est limitée par l'IP la plus lente, dans ce cas HENUC. Si l'hypothèse de parallélisation de HENUC se trouvait être vraie, alors l'encodeur total sera capable d'encoder à 100 images par seconde.

Toujours en comparaison avec le codeur Waaves développé dans le cadre de la thèse de Yuhui, l'implémentation est capable d'encoder en 21,45 millisecondes une image de 512 par 512 en niveaux de gris. Ceci correspond à 12,2 Mo/s de débit. Nous avons gagné un facteur 10 et un potentiel facteur 20 grâce à des améliorations algorithmiques et une nouvelle implémentation matérielle. Par rapport à l'implémentation logicielle sur Intel nous avons amélioré par un facteur 23 la vitesse d'encodage.

**Tableau 13.** Tableau des coûts des composants de l'implémentation matérielle

| **Composant** | **Prix** |
|---|---|
| Caméra | 1110 € |
| Objectifs | 99 € |
| Câble *CameraLink* | 175 € |
| FPGA Arria V SoC | 2897 € |
| Carte d'adaptation *HSMC/CameraLink* | 210 € |
| Carte d'adaptation *HSMC/GPIO* | 46 € |
| Deux cartes d'adaptation *HSMC/FMC* | 654 € |
| Total | 5191 € |

Pour finir, nous réalisons une estimation du prix de l'implémentation matérielle comme nous l'avons fait pour le l'implémentation logicielle. Cette estimation est détaillée dans le tableau 13. Le coût de la solution matérielle est 30 % moins cher que l'implémentation logicielle. Ceci est principalement dû à l'absence du PC et de SSD dans cette nouvelle solution. En contrepartie de ce gain sur le PC, le FPGA lui est plus coûteux et il y a un nouveau besoin de deux nouvelles cartes d'adaptation permettant de convertir un port HSMC en port FMC. Ces deux cartes sont utilisées car la carte de développement Arria V ne possède que des ports FMC comme connectique. Les cartes permettraient alors de continuer à utiliser les cartes d'adaptation *CameraLink* et GPIO comme pour l'implémentation logicielle.

# 4 Conclusion

Dans ce chapitre, nous avons premièrement montré que notre solution de synchronisation bas niveau est capable de synchroniser un examen en fréquence sur une période de 20 minutes. De cette expérimentation, nous avons trouvé que cette synchronisation est nécessaire car les fréquences de fonctionnement des appareils ne sont pas exactes et a entraîné un décalage de 1,5 seconde entre les modalités à la fin de l'examen. Le mécanisme proposé corrige intrinsèquement ce décalage. Nous avons également mesuré que le décalage entre deux modalités est en moyenne de 8,49 millisecondes. Nous avons ensuite exposé les avantages d'utiliser un conteneur tel que BIO-MKV pour un examen multimodal, en termes de synchronisation, d'édition et de stockage de données non continues et compressées. Nous avons montré également que le surcoût engendré par BIO-MKV est négligeable sur des examens de 20 minutes et est très léger à multiplexer.

Nous avons ensuite évalué ROI-Waaves en commençant par analyser sa complexité de $\mathcal{O}(n \times log(n))$. Ceci est une amélioration non négligeable par rapport au $\mathcal{O}(n^2)$ de Waaves dans une ère ou les tailles d'image ne cessent d'augmenter. Les évaluations des paramètres de ROI-Waaves nous ont permis de définir les valeurs idéales pour le codeur. Grâce à cette configuration, nous avons montré que ROI-Waaves reproduisait des séquences de meilleure qualité que JPEG, JPEG-2000 et Waaves dans les zones d'intérêts. Nous avons également montré que ROI-Waaves pouvait être comparé à H264 et VP8 pour la reproduction des zones d'intérêt dans les bas débits. Finalement, nous avons mesuré que la chaîne ROI-Waaves compressait 3,5 fois plus rapidement que Waaves.

Finalement, nous avons exposé les utilisations en ressources des deux implémentations ainsi que leur coût. Pour l'implémentation logicielle, celle-ci a été testée avec succès par le LNE afin d'avoir des certifications pour l'utilisation en cadre médical. Pour finir, nous avons montré que le développement du codeur matériel ROI-Waaves est théoriquement capable d'encoder 50 images par seconde en full-HD avec une possibilité en dupliquant l'IP HENUC d'atteindre 100 images par seconde.

# Conclusions et Perspectives

## Conclusions

La téléexpertise d'examen EEG reste, à ce jour, une problématique majeure dans le développement de la qualité des soins en France. Dans ses travaux de thèse, nous avons tout d'abord cherché à comprendre les besoins des médecins dans ce domaine afin de pouvoir y répondre au mieux. Les discussions avec les experts du domaine ont permis de remonter des limitations sur les équipements d'acquisition utilisés dans les centres de soins, notamment sur la synchronisation de l'acquisition vidéo avec les signaux physiologiques et sur la qualité de l'examen, spécifiquement la vidéo, en relecture.

Nous avons établi qu'aucun équipement de l'état de l'art ne répondait au problème de synchronisation car ces équipements étaient architecturés avec des composants et des protocoles non déterministes et que les résolutions et fréquences vidéo proposées ne suffisaientt pas aux médecins pour réaliser des diagnostiques de pathologies comme les myoclonies. De plus, les formats de fichier utilisé ne permettaient pas de conserver les informations temporelles nécessaires pour resynchroniser les différents flux de données lors de la relecture. Finalement, nous avons constaté que les algorithmes de compression utilisés pour encoder l'enregistrement vidéo ne répondaient pas aux attentes de certification médicale. De plus, les codeurs certifiés dispositif médical ne proposait pas des taux de compression et des temps de traitement suffisants pour l'application.

Afin de proposer une solution au problème de synchronisation, nous avons développé une méthode combinant une synchronisation bas niveau et une synchronisation haut niveau. La synchronisation complètement matérielle permet l'horodatage des données lors de leur acquisition permettant ainsi d'avoir un repère temporel déterministe par données. Les données acquises sont ensuite encapsulées dans le conteneur développé BIO-MKV permettant la synchronisation et le multiplexage des données. Le fichier BIO-MKV constitue la base du fichier examen pouvant être envoyé à un centre de diagnostic ou archivé chez un hébergeur de santé. Par l'expérimentation, nous avons montré que la combinaison de synchronisation permettait d'éliminer la dérive d'horloge d'acquisition. De plus, nous avons mesuré que le surcoût en espace de BIO-MKV est inférieur à 1% dans les cas d'usage classique et que la proportion de temps à constituer un flux était de moins de 0.8% par un processeur.

Nous avons ensuite introduit le codeur ROI-Waaves proposant une solution à la

compression de la vidéo de l'examen. Celui-ci exploite les caractéristiques propres de la scène et de l'acquisition afin de proposer une compression adaptée. Le codeur exploite notamment des zones d'intérêt pour optimiser la compression ainsi que le temps de traitement du codeur. De cette manière, nous avons montré que nous avions amélioré la complexité du codeur par rapport à Waaves de $\mathcal{O}(n^2)$ à $\mathcal{O}(n \times log(n))$. De plus, l'analyse réalisée sur cinq séquences vidéo a permis de démontrer que le codeur était plus performant en taux de compression à qualité égale que les codeurs JPEG, JPEG-2000 et Waaves et qu'il proposait des performances comparables à H264 et VP8 sur la reproduction des zones d'intérêts.

Pour finir, nous avons proposé deux implémentations permettant l'acquisition d'examen EEG et sa télétransmission. Les deux implémentations utilisent les mécanismes de synchronisation et de compression décrits précédemment. La première implémentation est constituée d'un FPGA pour réaliser les interfaces matérielles et la synchronisation et d'un PC pour réaliser la compression et l'encapsulation dans BIO-MKV. L'acquisition est effectuée en flux tendu pour ensuite utiliser un SSD comme tampon. La compression est alors effectuée en logiciel à partir des images de ce tampon. La vitesse de compression de 2 images par seconde demande une optimisation. Nous avons donc proposé une deuxième implémentation basée sur un SoC contenant un FPGA et un processeur ARM. Cette nouvelle implémentation reprend les mécanismes du premier en basculant l'encodage vidéo dans une IP du FPGA. Nous avons avancé que cette implémentation était capable d'encoder à plus de 50 images par seconde avec la possibilité de dupliquer une partie du codeur pour atteindre 100 images par seconde grâce à notre innovation de sous-arbre sur l'encodeur entropique HENUC. Il est à noter que l'implémentation logicielle est en cours de certification par le laboratoire national de métrologie et d'essais pour son usage en milieu médical expérimental.

Ces travaux de recherche ont été publiés deux journaux (JRTIP et ERT), quatre conférences (EMBC, BHI et FTFC), une conférence nationale (JETSAN) et deux groupes de recherche nationaux (GDR-SoCSiP). Un dépôt logiciel sur l'encodeur ROI-Waaves a été accepté et un autre dépôt est en cours de publication.

## Perspectives

Nous avons découpé cette section en perspectives à court terme et perspectives à moyen/long terme.

**Court terme :**

La certification de l'implémentation logicielle nous permettrait alors de commencer à faire des essais cliniques sur des patients afin d'évaluer la solution apportée par le projet Smart-EEG. Ces essais cliniques demandent un suivi et un accompagnement du

personnel hospitalier afin qu'il n'y ait pas de problème ou dans le cas échéant il soit réglé le plus rapidement possible. Ces essais sont primordiaux et permettront de fermer la boucle de ces travaux de recherche.

Nous voulons également investiguer les performances de la synchronisation bas niveau pour comprendre les variations et décalage que nous observons lors des évaluations. Nous émettons l'hypothèse que les résultats obtenus sont des conséquences de temps de délai de la caméra et du composant ADS 1298. Cependant, pour en être sûrs, nous voulons utiliser d'autres composants pour confirmer cette hypothèse.

**Moyen/long terme :**

À moyen terme, nous désirons finaliser l'implémentation matérielle afin de pouvoir améliorer la qualité de service de la solution. Ceci implique dans un premier temps de définir la cible/carte pour l'implémentation et de porter l'implémentation logicielle sur celle-ci. Dans un deuxième temps, il sera nécessaire d'intégrer les IP du nouveau codeur ROI-Waaves développé afin de faire un pipeline complet. Cette tâche permettra d'évaluer beaucoup plus facilement le codeur ROI-Waaves car les temps de compression seront bien réduits.

Une tâche qui a été commencée par un stagiaire de l'équipe SYEL consistait à implémenter le codeur Waaves sur GPU. Ce stage a montré des résultats très encourageants sur les performances de la DWT sur GPU et nous pensons que l'implémentation du codeur entier Waaves puis ROI-Waaves sera bénéfique pour les temps de traitement et plus portable qu'une implémentation FPGA. Ceci permettra également de comparer les versions GPU et FPGA.

Un autre aspect que nous avons survolé dans ce mémoire est la décompression d'un flux ROI-Waaves. Une solution de décodage doit pouvoir être mise en place afin de pouvoir décoder le flux ROI-Waaves de manière efficace sur un PC personnel. Ce point rejoint la tâche précédente car une implémentation GPU pourrait aider dans cette optique.

Un autre aspect que nous voudrions explorer est la certification du codeur ROI-Waaves. En effet, même si le codeur comporte des similarités avec Waaves, il y a également de grandes différences dans la chaîne de traitement. Un travail de certification pour sa légalisation est donc nécessaire.

Finalement, nous pensons que la solution proposée peut être utilisée dans un cadre plus large que pour l'examen EEG simple. Des applications de polysomnographie, dont les signaux acquis sont très proches de ceux de l'examen EEG, pourraient bénéficier de la solution. Nous pensons aussi que le dispositif pourrait être utilisé dans le cadre d'analyse de performance pour des athlètes ou dans des sessions d'entraînement de militaires ou d'astronautes.

# Publications

## Journaux

M. S. Ibraheem, K. Hachicha, S. Z. Ahmed, L. Lambert et P. Garda : *"High-throughput parallel DWT hardware architecture implemented on an FPGA-based platform."* Journal of Real-Time Image Processing, pp. 1-15, 2015.

L. Lambert, J. Despatin, I. Dhif, I. Mhedhbi, M. Ibraheem, S.-Z. Ahmed, B. Granado, K. Hachicha, A. Pinna, P. Garda, F. Kaddouh, M. TEROSIET, A. Histace, O. Romain, C. Bellet, F. Durand, J. P. Commes, S. Hochberg, D. Heudes, P. Lozeron et N. Kubis : *"Telemedecine, electroencephalography and current issues. SMART-EEG : An innovative solution"*, European Research in Telemedicine / La Recherche Européenne en Télémédecine, vol. 4 (3), pp. 81-86 , 2015.

## Conférences internationales

L. Lambert, S. Z. Ahmed, K. Hachicha, A. Pinna et P. Garda, *"High frame rate medical quality video compression for tele-EEG"*, Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Orlando, USA, pp. 5392-5396, 2016.

I. Dhif, M. Ibraheem, L. Lambert, Kh. Hachicha, A. Pinna, S. Hochberg, I. Mhedhbi et P. Garda : *"A novel approach using WAAVES coder for the EEG signal compression"*, International Conference on Biomedical and Health Informatics, (BHI), Las Vegas, USA, pp. 453-456, 2016.

L. Lambert, K. Hachicha, S. Zahid Ahmed, A. Pinna et P. Garda : *"Synchronizing Physiological Data and Video in a Telemedicine Application : a Multimedia Approach"*, Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Milan, Italy, pp. 181-185, 2015.

S. Z. Ahmed, Y. Bai, I. Dhif, L. Lambert, I. Mehdbhi, P. Garda, B. Granado, Kh. Hachicha, A. Pinna, F. Ghaffari, A. Histace et O. Romain : *"SmartEEG : a multimodal tool for EEG signals"*, Faible Tension Faible Consommation, Monaco, Monaco, pp. 1-4, 2014.

## Conférences nationales

L. Lambert, A. Pinna, K. Hachicha et P. Garda : *"Un format de fichier pour l'examen vidéo-EEG"*, GDR-SOCSIP, Nantes, France, Juin 2016.

L. Lambert, I. Dhif, M. Shaaban Ibraheem, B. Granado, Kh. Hachicha, A. Pinna, P. Garda, N. Kubis, F. Kaddouh, D. Heudes, M. Terosiet, A. Histace, O. Romain, S. Hochberg, I. Mhedhbi : *"Smart-EEG : a Tele-medicine System for EEG Exams"*, JETSAN 2015, Compiègne, France, 2015.

L. Lambert, I. Dhif, M. Shaaban Ibraheem, S. Zahid Ahmed, B. Granado, K. Hachicha, A. Pinna et P. Garda : *"Smart-EEG : A New Platform for Tele-Expertise of Electroencephalogram"*, GDR-SOCSIP, Paris, France, Juin 2014.

## Dépôt logiciel

L. Lambert, Kh. Hachicha, A. Pinna, P. Garda : *"Waaves-ROI"*, IDDN.FR.001.250003.000.S.A.2016.000.21000, 9 Juin 2016

L. Lambert, Kh. Hachicha, A. Pinna, P. Garda : *"BIO-FFMPEG"*, en cours de publication.

# Bibliographie


[ADS1298] Convertisseur analogique-numérique TI ADS 1298. http://www.ti.com/product/ADS1298. Accessed : 2017-05-11.

[ALSA] Audio Linux Sound Architecture. https://www.alsa-project.org/main/index.php/Main_Page. Accessed : 2017-05-09.

[André-Obadia et al., 2014] André-Obadia, N., Sauleau, P., Cheliout-Heraut, F., Convers, P., Debs, R., Eisermann, M., Gavaret, M., Isnard, J., Jung, J., Kaminska, A., et al. (2014). Recommandations françaises sur l'électroencéphalogramme. *Neurophysiologie Clinique/Clinical Neurophysiology*, 44(6) :515–612.

[Antonini et al., 1992] Antonini, M., Barlaud, M., Mathieu, P., and Daubechies, I. (1992). Image coding using wavelet transform. *IEEE Transactions on image processing*, 1(2) :205–220.

[ASF] Microsoft, Windows Media Video. http://www.microsoft.com/en-us/download/details.aspx?displaylang=en&id=14995.

[AVI] Microsoft, Audio Video Interleaved. https://msdn.microsoft.com/en-us/library/windows/desktop/dd318189(v=vs.85).aspx.

[Bai, 2014] Bai, Y. (2014). *Compression temps réel de séquences d'images médicales sur les systèmes embarqués*. PhD thesis, Cergy-Pontoise.

[Bai et al., 2013] Bai, Y., Ahmed, S. Z., and Granado, B. (2013). Design and analysis of an FPGA based encoder SoC for locally stationary image source. In *DASIP*, pages 271–278. IEEE.

[Bai et al., 2014] Bai, Y., Ahmed, S. Z., and Granado, B. (2014). Accelerating Heap-Based Priority Queue in Image Coding Application Using Parallel Index-Aware Tree Access. In Goehringer, D., Santambrogio, M. D., Cardoso, J. M. P., and Bertels, K., editors, *ARC*, volume 8405 of *Lecture Notes in Computer Science*, pages 37–48. Springer.

[Bankoski et al., 2011] Bankoski, J., Wilkins, P., and Xu, Y. (2011). Technical overview of VP8, an open source video codec for the web. In *2011 IEEE International Conference on Multimedia and Expo*, pages 1–6.

[Basler] Caméra Basler acA2000-340kc. https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2000-340kc/. Accessed : 2017-05-11.

[Bayer, 1976] Bayer, B. E. (1976). Color imaging array. US Patent 3,971,065.

[BIOSEMI] BIOSEMI. http://www.biosemi.com/faq/file_format.htm. Accessed : 2017-09-08.

[Bird and Murphy Jr, 1974] Bird, K. T. and Murphy Jr, R. L. (1974). Telediagnosis : A New Community Health Resource. *American Journal of Public Health*, 64(2).

[Blakowski and Steinmetz, 1996] Blakowski, G. and Steinmetz, R. (1996). A media synchronization survey : Reference model, specification, and case studies. *IEEE journal on selected areas in communications*, 14(1) :5–35.

[Brown et al., 1991] Brown, P., Thompson, P., Rothwell, J., Day, B., and Marsden, C. (1991). Axial myoclonus of propriospinal origin. *Brain*, 114(1) :197–214.

[Campos et al., 2012] Campos, C., Caudevilla, E., Alesanco, A., Lasierra, N., Martinez, O., Fernández, J., and García, J. (2012). Setting up a telemedicine service for remote real-time video-EEG consultation in La Rioja (Spain). *International journal of medical informatics*, 81(6) :404–414.

[Chacko and McCullagh, 2014] Chacko, S. and McCullagh, P. (2014). Home based mobile solution for video ambulatory EEG monitoring. In *Intelligent Environments (IE), 2014 International Conference on*, pages 107–110. IEEE.

[Christopoulos et al., 2000] Christopoulos, C., Skodras, A., and Ebrahimi, T. (2000). The JPEG2000 still image coding system : an overview. *IEEE Transactions on Consumer Electronics*, 46(4) :1103–1127.

[Coates et al., 2012] Coates, S., Clarke, A., Davison, G., and Patterson, V. (2012). Tele-EEG in the UK : a report of over 1000 patients.

[Corwin and Logar, 2004] Corwin, E. and Logar, A. (2004). Sorting in linear time-variations on the bucket sort. *Journal of computing sciences in colleges*, 20(1) :197–202.

[Davisson, 1966] Davisson, L. D. (1966). Comments on" Sequence time coading for data compression". *Proceedings of the IEEE*, 54(12) :2010–2010.

[De Luca, 1998] De Luca, L. (1998). TeleEEG : A telemedical software package for EEG. *Future Generation Computer Systems*, 14(1-2) :61–66.

[EBML] Spécification du format EBML. https://github.com/Matroska-Org/ebml-specification/blob/master/specification.markdown. Accessed : 2017-04-25.

[Eidson and Lee, 2002] Eidson, J. and Lee, K. (2002). IEEE 1588 standard for a precision clock synchronization protocol for networked measurement and control systems. In *Sensors for Industry Conference, 2002. 2nd ISA/IEEE*, pages 98–105. IEEE.

[FFmpeg] FFmpeg tool. https://ffmpeg.org/. Accessed : 2017-05-12.

[GMEPRT] Guide méthodologique pour l'élaboration du programme régional de télémédecine. http://solidarites-sante.gouv.fr/IMG/pdf/guide_methhodologique_elaboration_programme_regional_telemedecine.pdf. Accessed : 2017-09-08.

[Green, 1968] Green, D. G. (1968). The contrast sensitivity of the colour mechanisms of the human eye. *The Journal of physiology*, 196(2) :415.

[HL7] ISO/HL7 10781 :2015, HL7 Electronic Health Records-System Functional Model, Release 2 (HER FM). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57757. Accessed : 2017-05-12.

[Holzer, 1974] Holzer, W. (1974). Telemedicine : New Application of Communications Technology. *IEEE Transactions on Communications*, 22(5) :685–688.

[Hoschka, 1998] Hoschka, P. (1998). Synchronized Multimedia Integration Language (SMIL) 1.0 Specification.

[House, 1991] House, M. (1991). The Canadian experience : Using telemedicine for the support of medical care at remote sites.

[House et al., 1987] House, M., Keough, E., Hillman, D., Hillman, E., Bwibo, N., Meme, J., Wafula, E., MacLeod, S., and McCullough, N. (1987). Into Africa : the telemedicine links between Canada, Kenya and Uganda. *CMAJ : Canadian Medical Association Journal*, 136(4) :398.

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9) :1098–1101.

[Ibraheem, 2017] Ibraheem, M. S. (2017). *Logarithmic Discrete Wavelet Transform for High-Quality Medical Image Compression*. PhD thesis, Université Pierre et Marie Curie (UPMC). Type : Thèse de Doctorat – Soutenue le : 2017-03-29 – Dirigée par : Garda, Patrick – Encadrée par : HACHICHA Khalil.

[ISO/EIC, 1991] ISO/EIC (1991). 10918-1. Digital Compression and Coding of Continuous-tone Still Images (JPEG). *CCITT Recommendation T*, 81.

[ISO/IEC, 1993] ISO/IEC (1993). 11172-1 :1993 - Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 1 : Systems. Technical report, RFC Editor.

[ISO/IEC, 2000] ISO/IEC (2000). JPEG2000, ISO and FCD15444, IEC : 2000. *Annex H, March*.

[ISO/IEC, 2003] ISO/IEC (2003). recommendation and final draft international standard of joint video specification (ITU-T Rec. H. 264| ISO/IEC 14496-10 AVC). *Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050*, 33.

[ISO/IEC, 2010] ISO/IEC (2010). 14496-1 :2010 - Information technology – Coding of audio-visual objects – Part 1 : Systems. Technical report, RFC Editor.

[ISO/IEC, 2013] ISO/IEC (2013). 13818-1 :2013 - Information technology – Generic coding of moving pictures and associated audio information – Part 1 : Systems. Technical report, RFC Editor.

[Jacobsen et al., 2015] Jacobsen, M., Richmond, D., Hogains, M., and Kastner, R. (2015). RIFFA 2.1 : A reusable integration framework for FPGA accelerators. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 8(4) :22.

[Jasper, 1958] Jasper, H. H. (1958). The ten-twenty electrode system of the International Federation. *Electroencephalography and Clinical Neurophysiology*, 10(10) :371 – 375.

[Kemp et al., 1992] Kemp, B., Värri, A., Rosa, A. C., Nielsen, K. D., and Gade, J. (1992). A simple format for exchange of digitized polygraphic recordings. *Electroencephalography and Clinical Neurophysiology*, 82(5) :391 – 393.

[Lasierra et al., 2009] Lasierra, N., Alesanco, A., Campos, C., Caudevilla, E., Fernandez, J., and Garcia, J. (2009). Experience of a real-time tele-EEG service. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 5211–5214.

[Le Gall and Tabatabai, 1988] Le Gall, D. and Tabatabai, A. (1988). Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques. In *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*, pages 761–764. IEEE.

[Lee and Ortega, 2001] Lee, S.-Y. and Ortega, A. (2001). A novel approach of image compression in digital cameras with a Bayer color filter array. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 482–485. IEEE.

[Lewandowski et al., 1993] Lewandowski, W., Petit, G., and Thomas, C. (1993). Precision and accuracy of GPS time transfer. *IEEE Transactions on Instrumentation and Measurement*, 42(2) :474–479.

[Li et al., 2008] Li, X., Gunturk, B., and Zhang, L. (2008). Image demosaicing : A systematic survey. In *Electronic Imaging 2008*, pages 68221J–68221J. International Society for Optics and Photonics.

[Lindsay et al., 1987] Lindsay, E. A., Davis, D. A., Fallis, F., Willison, D. B., and Biggar, J. (1987). Continuing education through Telemedicine for Ontario. *CMAJ : Canadian Medical Association Journal*, 137(6) :503.

[Lynch, 1966] Lynch, T. J. (1966). Sequence time coding for data compression.

[Mhedhbi et al., 2012] Mhedhbi, I., Hachicha, K., Garda, P., Bai, Y., Granado, B., Topin, S., and Hochberg, S. (2012). Towards a Mobile Implementation of Waaves for Certified Medical Image Compression in E-Health Applications. In *MobiHealth*, pages 79–87.

[Mhedhbi et al., 2014] Mhedhbi, I., Kaddouh, F., Hachicha, K., Heudes, D., Hochberg, S., and Garda, P. (2014). Mask Motion Adaptive medical image coding. In *IEEE BHI conference*.

[Mills, 1991] Mills, D. L. (1991). Internet time synchronization : the network time protocol. *IEEE Transactions on Communications*, 39(10) :1482–1493.

[MistarDoc] Mitsar EEG documentation. http://www.mitsar-medical.com/download/manual/Mitsar_EEG_202L31_OM_ENG_2016-10-12.pdf. Accessed : 2017-09-08.

[News] News vidéo. http://hwcdn.net/j9t9v3v5/cds/News_ProRes.mov. Accessed : 2017-09-08.

[NI] National Instrument. http://www.ni.com/. Accessed : 2017-09-08.

[Noé, 2007] Noé, A. (2007). Matroska File Format. *Jun*, 24 :1–51.

[Öktem, 1999] Öktem, L. (1999). *Hierarchical enumerative coding and its applications in image compression*. Tampere University of Technology.

[Öktem and Astola, 1999] Öktem, L. and Astola, J. (1999). Hierarchical enumerative coding of locally stationary binary data. *Electronics Letters*, 35(17) :1428–1429.

[Preston, 1995] Preston, J. (1995). Texas Telemedicine Project : a viability study. *Telemedicine Journal*, 1(2) :125–132.

[Rissanen and Langdon, 1979] Rissanen, J. and Langdon, G. G. (1979). Arithmetic coding. *IBM Journal of research and development*, 23(2) :149–162.

[Santa-Cruz and Ebrahimi, 2000] Santa-Cruz, D. and Ebrahimi, T. (2000). A study of JPEG 2000 still image coding versus other standards. In *2000 10th European Signal Processing Conference*, pages 1–4.

[Sauleau et al., 2016] Sauleau, P., Despatin, J., Cheng, X., Lemesle, M., de Villepin, A. T., the Tich, S. N., and Kubis, N. (2016). National French survey on tele-transmission of EEG recordings : More than a simple technological challenge. *Neurophysiologie Clinique/Clinical Neurophysiology*, 46(2) :109 – 118.

[Shaaban, 2017] Shaaban, I. M. (2017). *Étude d'une arithmétique logarithmique appliqué à la compression des images médicales*. PhD thesis, Université Pierre et Marie Curie.

[Shannon, 1949] Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, 37(1) :10–21.

[Shannon, 2001] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1) :3–55.

[Storer, 1988] Storer, J. (1988). *Data compression*. Elsevier.

[Suzie] Suzie vidéo. http://hwcdn.net/j9t9v3v5/cds/Suzie_ProRes.mov. Accessed : 2017-09-08.

[Tahoces et al., 2008] Tahoces, P. G., Varela, J. R., Lado, M. J., and Souto, M. (2008). Image compression : Maxshift ROI encoding options in JPEG2000. *Computer vision and Image understanding*, 109(2) :139–145.

[Taubman, 2000] Taubman, D. (2000). High performance scalable image compression with EBCOT. *IEEE Transactions on image processing*, 9(7) :1158–1170.

[TLV320AIC26] Codec audio TI TLV320AIC26. http://www.ti.com/product/TLV320AIC26. Accessed : 2017-05-11.

[Toddler] Toddler. https://media.xiph.org/video/derf/Chimera/Netflix_ToddlerFountain_4096x2160_60fps_10bit_420.y4m. Accessed : 2017-09-08.

[Vaz et al., 1991] Vaz, F., Pacheco, O., and da Silva, A. M. (1991). A telemedicine application for EEG signal transmission. In *Engineering in Medicine and Biology Society, 1991. Vol. 13 : 1991., Proceedings of the Annual International Conference of the IEEE*, pages 466–467. IEEE.

[Waaves] Waaves compression algorithm. http://www.waaves.com. Accessed : 2016-11-04.

[Wallace, 1992] Wallace, G. K. (1992). The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1) :xviii–xxxiv.

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment : from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4) :600–612.

[Wiegand et al., 2003] Wiegand, T., Sullivan, G. J., Bjontegaard, G., and Luthra, A. (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7) :560–576.

[Wilkins et al., 2011] Wilkins, P., Xu, Y., Quillio, L., Bankoski, J., Salonen, J., and Koleszar, J. (2011). VP8 Data Format and Decoding Guide. RFC 6386.

Annexe A

# Structures internes de FFmpeg

## 1 Les objets abstraits

Les données manipulées par FFmpeg sont diverses et variées, allant de données brutes à des données encapsulées et compressées. FFmpeg traite de façon native trois types de médias : la vidéo, l'audio et les sous-titres. De manière à interpréter les diverses formes que peuvent prendre ces données, FFmpeg définit trois abstractions lui permettant de gérer ce polymorphisme : les trames, les paquets et les entrées/sorties. Nous détaillerons dans la suite comment sont implémentées ces trois structures de données.

### 1.1 Les trames

Une trame est un objet contenant des données brutes dans FFmpeg. Elle peut contenir des données vidéo (images) ou des données audio (échantillons). Chaque trame est représentée par un objet de type *AVFrame* contenant les données brutes d'une unité de données ainsi que les métadonnées permettant de l'interpréter. L'information la plus importante dans cet enregistrement est le format précisant le type d'encodage utilisé pour les données contenues. Ces valeurs proviennent soit de l'énumération de *AVPixelFormat* (Bayer, RGB, YUV, etc.) pour les formats de pixels vidéo, soit de l'énumération de *AVSampleFormat* (huit bits, seize bits, flottant simple précision, double précision, etc.) pour les formats d'échantillons audio. Quelles que soient les données contenues (vidéo ou audio), l'enregistrement contient une zone mémoire pour stocker les données brutes, ainsi qu'un champ de taille, *linesize*, permettant de stocker le nombre de pixels ou d'échantillons d'une trame. Enfin le champ *pts* (Presentation TimeStamp) est un champ de timestamp contenant la position temporelle de la donnée, il précise le temps auquel la donnée doit être présentée (affichée ou jouée). L'enregistrement est ensuite complété par des données spécifiques à un type de média comme la taille de la vidéo par les champs *width* et *height* ou bien le nombre d'échantillons et le nombre de canaux d'une trame audio pas les champs *nb_samples* et *channels*.

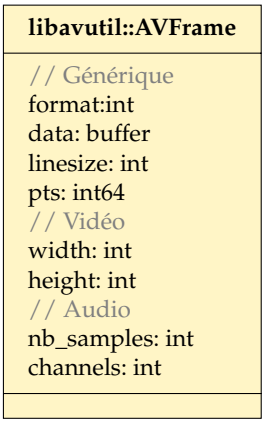



**Figure 96.** Diagramme de classe Uml de l'enregistrement *AVFrame* de FFmpeg contenant une trame de données.

## 1.2 Les paquets

Au moment de la compression, les trames (données d'entrées) sont consommées pour produire des paquets (données de sorties). Ces paquets, tous comme les trames, sont capables de contenir des données de n'importe quel type de média (vidéo, audio, sous-titre). Ils sont définis par des enregistrements de type *AVPacket* dont la composition est montrée dans la figure 97. La plupart du temps, un paquet contient une donnée compressée et n'est plus qu'une succession d'octets sans signification pour FFmpeg. Il contient un champ *data* pouvant stocker cette donnée ainsi qu'un champ *size* précisant la taille de cette donnée. Le paquet est ensuite attaché à un flux par son champ *stream_index* ce qui permet, entre autres, de connaître les paramètres du codec utilisé. Il y a ensuite deux métadonnées de temps : le *pts* et *dts*. Le *pts* est similaire au *pts* utilisé par l'enregistrement *AVFrame*. Celui-ci peut, cependant, avoir été mis à l'échelle par rapport aux contraintes de l'encodeur ou du format utilisé. Le *dts* (Decoding Timestamp) peut lui différer et prendre la valeur d'un autre moment. Ce timestamp est surtout utilisé à titre informatif pour le décodeur. En effet, pour certains codecs, l'ordre des paquets reçus par le décodeur n'est pas l'ordre dans lequel les données sont ensuite visualisées (c'est notamment le cas pour les codecs MPEG et VPX). Pour informer le décodeur que les paquets ne sont pas dans l'ordre, FFmpeg différencie les valeurs de *pts* et *dts*.

La figure 98 illustre la différence entre le *dts* et le *pts*. Dans cet exemple, le paquet un (pkt 1) est placé en premier dans le flux de données et doit être décodé en premier car son *dts* vaut un. Cependant, son *pts* est quatre et le décodeur n'a pas encore lu les paquets allant de un à trois. De ce fait, le décodeur sait qu'il doit continuer à lire les paquets suivants (de deux à quatre). Ceux-ci peuvent être décodés et présentés. C'est seulement après avoir présenté le paquet quatre que le décodeur peut afficher le paquet

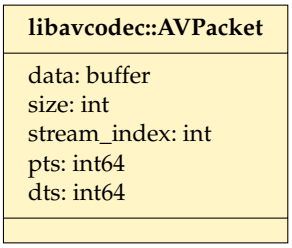



**Figure 97.** Diagramme de classe Uml de l'enregistrement *AVPacket* de FFmpeg contenant un paquet de données

un, mis en attente. Lorsque le *dts* et le *pts* d'un paquet sont égaux, alors celui-ci peut être décodé et présenté sans attente.

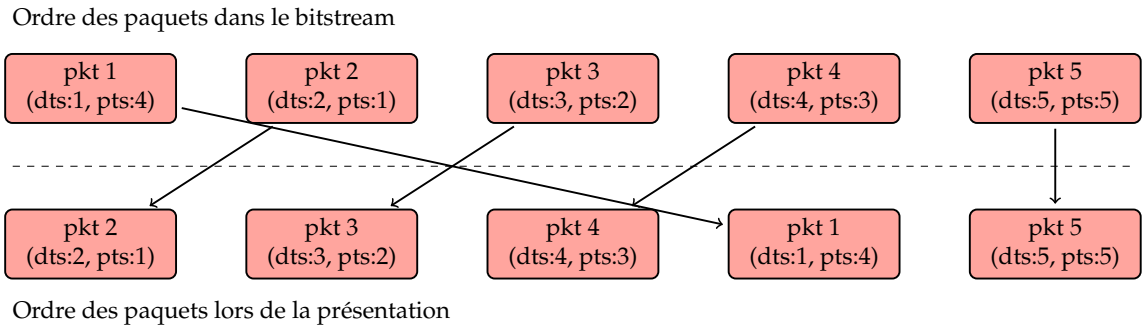



**Figure 98.** Illustration du changement d'ordre des paquets

## 1.3 Les entrées/sorties

Pour finir, FFmpeg doit gérer les multiples entrées/sorties que peut utiliser un flux multimédia. En effet, même si un flux multimédia peut être supposé écrit dans un fichier, il arrive souvent qu'un flux soit envoyé sur un réseau (RTP, TCP, UDP) ou que l'utilisateur veuille rajouter une couche de chiffrement. Pour gérer cela, FFmpeg définit un enregistrement *AVIOContext* contenant les données et méthodes propres à chaque type d'entrée/sortie. La figure 99 présente les différents champs de cet enregistrement. Celui-ci contient des champs servant à décrire l'entrée/sortie (Possibilité de se positionner dans le flux, la taille max du fichier, la taille maximum d'un paquet, etc) ainsi que trois méthodes permettant de faire des opérations. La première opération ,*read_packet*, est la lecture de données permettant de lire *size* octets du flux dans un buffer. La deuxième opération, *write_packet*, analogue, permet d'écrire *size* octets dans un flux. Et pour finir, l'opération *seek* permettant de se déplacer dans le flux si le type d'entrée/sortie en question le permet. Les noms utilisés pour ces opérations ne sont pas forcément adaptés car ils laissent transparaître qu'elles écrivent où lisent des paquets

alors que ces méthodes travaillent sur des tampons de mémoires bruts.

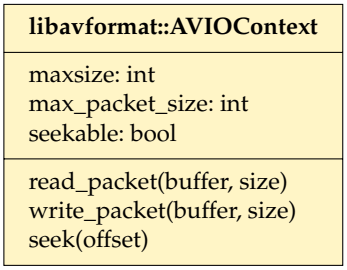



**Figure 99.** Diagramme de classe Uml de l'enregistrement *AVIOContext* de FFmpeg.

Maintenant que nous avons décrit les trois types de données manipulés par FFmpeg, nous allons détailler comment la bibliothèque passe d'un type à l'autre en utilisant des codecs et des formats.

## 2 Les codecs

Les codecs sont déclarés en instanciant des objets de type AVCodec dont le patron est visible dans la figure 100. Cet objet contient les informations spécifiques de ce codec, en particulier, son nom, le type de données que le codec peut traiter, la norme qu'implémente ce codec (spécifiée par un identifiant unique dans FFmpeg), ainsi que les fonctionnalités/caractéristiques de l'implémentation. Ces fonctionnalités peuvent être diverses et variées comme pouvoir permettre l'encodage sans perte d'une donnée, l'encodage de manière parallèle, l'encodage en deux passes, l'encodage assisté par du matériel, etc. En addition à ces champs, l'objet contient cinq méthodes permettant d'utiliser ce codec. Toutes ces méthodes ont en paramètre le contexte servant à garder l'évolution d'une instance de compression ou de décompression.

- La méthode *init* est exécutée une seule fois au début de la compression ou de la décompression et permet d'initialiser le contexte du codec.
- La méthode *encode2* est utilisée pour encoder des données brutes en données compressées. Cette méthode ne retourne pas toujours de paquet. Comme dit précédemment, les compresseurs tels que h264 ou vp8 ne compriment pas forcément les images directement. Ils stockent une séquence d'images de manière à réorganiser le flux de manière optimale. De ce fait lors de la phase de mise en tampon, lorsque la fonction *encode2* est exécutée, le compresseur va mettre en tampon l'image courante en attendant les prochaines exécutions de la fonction *encode2* pour recevoir davantage d'images. De la même manière, pour encoder un

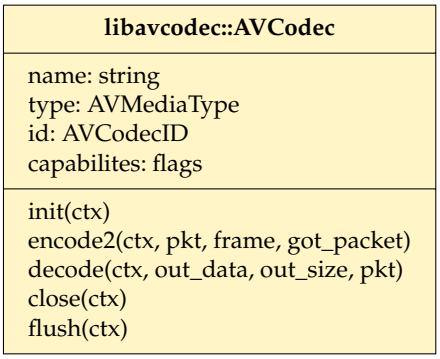



**Figure 100.** Diagramme de classe Uml de l'enregistrement *AVCodec* de FFmpeg.

flux audio, certains codecs attendent d'avoir un minimum d'échantillons afin de pouvoir créer un paquet entier.

- La méthode *decode* est utilisée pour faire l'opération inverse c'est-à-dire transformer un paquet en données brutes. Pour les mêmes raisons que la méthode *encode2*, la méthode *decode* peut ne pas retourner de trame. Dans ce cas la taille des données retournées sera zéro.
- La méthode *close* est exécutée une dernière fois à la fin d'un contexte de compression ou de décompression. Elle permet de récupérer les ressources allouées lors de l'initialisation.
- La méthode *flush* permet à l'utilisateur de vider toutes les données mises en attente par le codec. Cette fonction est utilisée principalement lorsque le flux de sortie est réseau et qu'il a besoin de garantir des propriétés de temps réel.

Cet objet, unique à chaque implémentation de codec, est déclaré comme étant une variable globale à la bibliothèque. Cette variable globale est ensuite utilisée lors de la phase d'initialisation de FFmpeg, dans la fonction *avcodec_register_all*, afin d'enregistrer le codec dans sa base de codecs. De ce fait, quand FFmpeg a besoin d'un codeur ou d'un décodeur, il va chercher l'implémentation du codec dans sa base de codecs. Une fois le codec trouvé, il initialise un contexte pour ce codec et initialise ensuite le codec en lui-même grâce à la méthode *init*. Il peut ensuite demander au codec de compresser une donnée avec la fonction *encode2* ou de décompresser une donnée avec la fonction *decode*.

## 3 Les formats

À la différence des codecs, la gestion des formats est scindée en deux, les formats d'entrée et les formats de sortie. Un format d'entrée aussi appelé demuxeur permet de lire un flux d'entrée et de séparer les différents médias afin de fournir des paquets. L'enregistrement servant à contenir les informations d'un format d'entrée est *AVInputFormat*. De façon analogue, un format de sortie appelé muxeur entrelace et écrit des

paquets dans un flux de sortie et est contenu dans l'enregistrement *AVOutputFormat*. Tout comme pour les codecs, les formats nécessitent un contexte permettant de stocker les données utiles à la réception ou à l'émission d'un flux, ce contexte est passé en paramètres aux méthodes de l'enregistrement. Nous détaillerons dans un premier temps la gestion des formats d'entrée puis la gestion des formats de sortie.

Les formats d'entrées sont déclarés via une variable de type AVInputFormat. Ce type est représenté par la figure 101. Cet enregistrement ne contient que deux membres, le nom du format et des flags indiquant les capacités de l'implémentation comme le support de la lecture sur réseau, la possibilité de se déplacer dans le flux (*seek*).

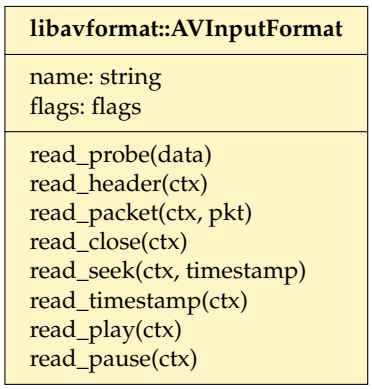



**Figure 101.** Diagramme de classe Uml de l'enregistrement *AVInputFormat* de FFmpeg.

Les méthodes associées à l'enregistrement sont :

- la méthode *read_probe* est un peu particulière car elle ne prend pas en paramètre le contexte mais un buffer de données contenant le début d'un flux. La méthode doit déterminer si les données correspondent au format implémenté. Cette méthode est utilisée pour déterminer le type de flux quand celui-ci n'est pas connu.
- La méthode *read_header* permet de lire l'entêtes du flux d'entrée ainsi que l'initialisation du contexte de lecture. Cette méthode est très importante car elle permet de connaître toutes les informations contenues dans un flux et déclenche l'initialisation des codecs utiles à la décompression du flux.
- La méthode *read_packet* est la méthode principale pour récupérer des paquets d'un flux. Cette méthode renvoie séquentiellement les paquets arrivant d'un flux. Ces paquets peuvent ensuite être décompressés en fonction de leur codec. Les paquets sont retournés dans l'ordre de compression.
- La méthode *read_close* ferme un flux d'entrée et libère les ressources allouées lors de la lecture de l'entête.
- La méthode *read_seek* déplace le curseur interne de l'implémentation vers un nouveau paquet, permettant ainsi de se positionner dans le flux à une position bien précise. La position vers laquelle le flux se positionne est définie par le

timestamp passé en paramètre. Cette méthode peut être utilisée que sur des flux donc tout le contenu est disponible comme un fichier.
- La méthode *read_timestamp* renvoie le prochain timestamp dans le flux.
- La méthode *read_play* lance la lecture d'un flux. Cette méthode est utilisée lorsque le flux est un flux réseau.
- La méthode *read_pause* met en pause la lecture d'un flux. Cette méthode est utilisée lorsque le flux est un flux réseau.

De manière similaire, les formats de sorties sont eux déclarés par une variable de type *AVOutputFormat*. L'enregistrement *AVOutputFormat* est illustré par la figure 102. Il contient les mêmes membres que *AVInputFormat* contenant le nom et les capacités du format, tout en rajoutant trois nouveaux membres contenant les identifiants des codecs utilisés par défaut pour le format.

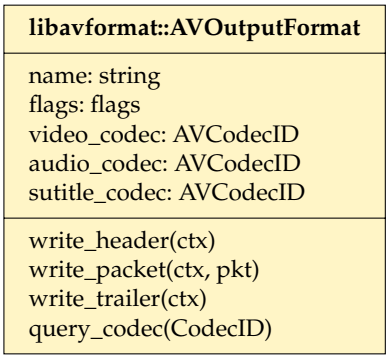



**Figure 102.** Diagramme de classe Uml de l'enregistrement *AVOutputFormat* de FFmpeg.

Les méthodes pour cet enregistrement sont très similaires aux méthodes utilisées pour la lecture de flux. Elles sont :
- la méthode *write_header* initialise le contexte du flux de sortie et écrit les entêtes du format dans le flux.
- La méthode *write_packet* écrit un paquet dans le flux de sortie. Le paquet en question peut être mis en tampon le temps d'avoir plus de données à écrire.
- La méthode *write_trailer* permet d'écrire la fin de fichier ainsi que de libérer les ressources allouées pour l'écriture du flux.
- La méthode *query_codec* permet de savoir si un codec est supporté par le format en question.

Chaque format a donc deux variables globales afin de gérer l'écriture et la lecture d'un flux, une variable AVInputFormat et une variable AVOutputFormat. Les différents muxeurs et demuxeurs sont enregistrés dans le code de FFmpeg dans la fonction *av_register_all*. Un format peut n'avoir que le muxeur ou que le demuxeur implémenté en fonction du support pour le format en question.

Annexe B

# Les codages entropiques

## 1 Le codage RLE

Le codage RLE [1] est un codage plus simple que le codage de Huffman. Il a été développé de manière à encoder des sources de symbole suivant un processus de Markov stable [2]. Il est notamment très bon pour les textes numérisés en noir et blanc (deux valeurs) car la probabilité qu'un pixel voisin soit d'une couleur différente est très faible.

Le principe est d'encoder les successions de symboles plutôt que les symboles seuls. L'encodeur compte donc le nombre de fois qu'un symbole apparaît, et écrit ce nombre ainsi que le symbole en lui-même. Par exemple, le message *AAACCDEEEEE* est encodé en *3A2C1D5E*. L'avantage de ce codage est sa simplicité de mise en œuvre. Cependant, dans le cas où le message est instable, l'encodage sera plus long que le message initial. Par exemple, le message *ACDBEADBBE* sera encodé en *1A1C1D1B1E1A1D2B1E* résultant en une perte de compression. Ce schéma pathologique est très souvent rencontré dans la compression d'image car les valeurs de pixel voisin, même si proche, sont toujours différentes ceci dû aux textures des objets de la scène ou au bruit d'acquisition.

## 2 Le codage de Huffman

Le codage de Huffman [Huffman, 1952] est probablement le codeur entropique le plus connu grâce à sa simplicité et son efficacité. Il est utilisé dans des compresseurs connus comme JPEG ou le MP3. Il produit, à partir de statistique préalablement définie, des codes à longueur variable pour représenter les symboles de la source de données. De ce fait, une première étape de génération de la correspondance entre chaque symbole de la source et son encodage est faite. Une fois cette correspondance établie (celle-ci peut être réalisée dynamiquement ou statiquement), l'encodeur se retrouve simplement à lire chaque symbole et d'écrire le code correspondant comme sortie. Pour trouver le

1. RLE : Run Length Encoding
2. Nous définirons un processus de Markov stable comme un processus dont la probabilité de changer d'état est bien plus faible que de rester dans le même état. Il est dit stable car il ne change pas souvent d'état.

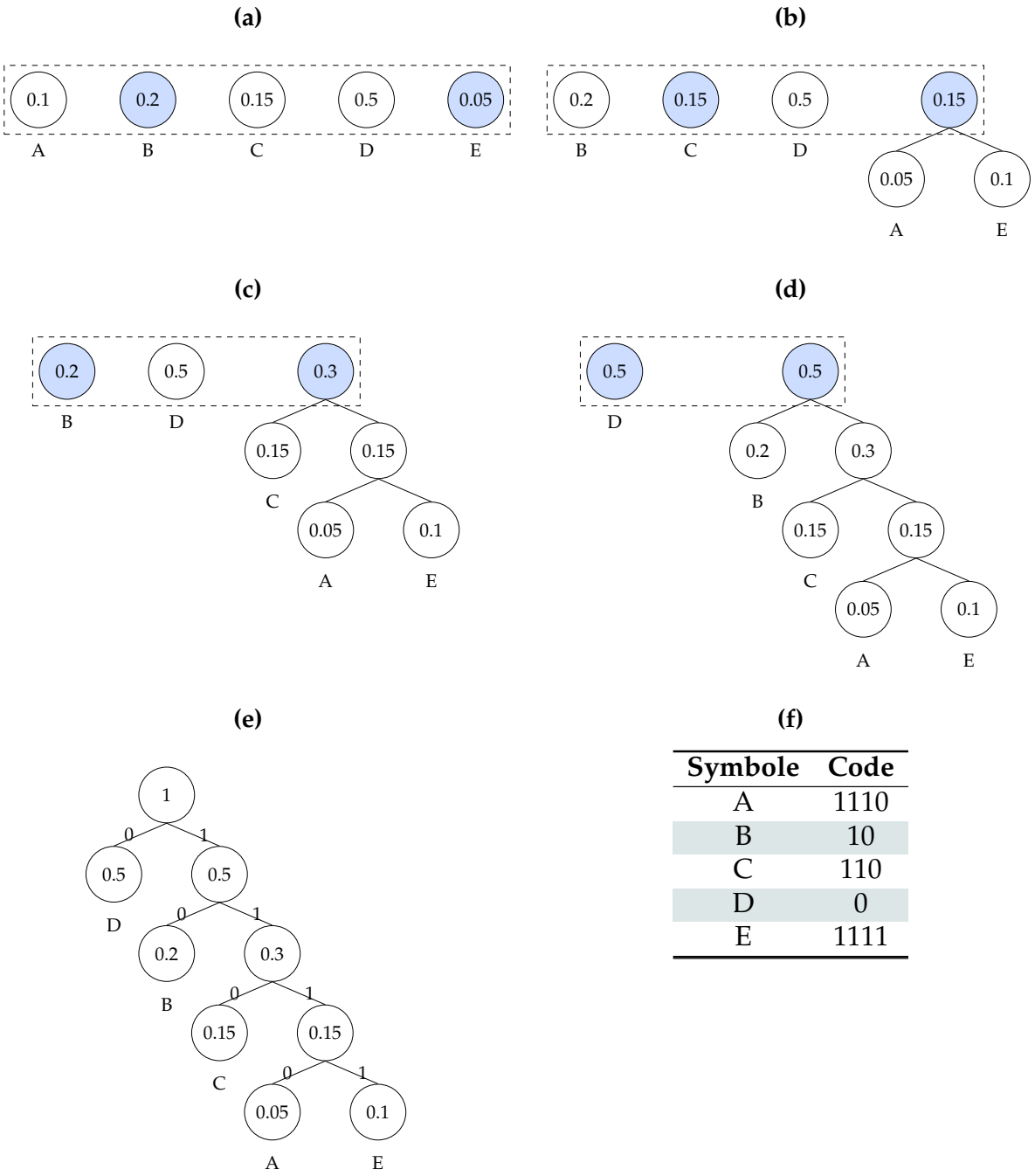
code correspondant, il est possible d'utiliser une table de correspondance simple ou un tableau associatif en fonction de la forme des symboles d'entrée. La particularité de ce codage est dans l'établissement de cette correspondance. Un arbre de codage (aussi appelé arbre de Huffman) est utilisé pour établir la correspondance entre les différents symboles et leur code. Cet arbre est un arbre binaire et il est construit en suivant l'algorithme 3.

**Algorithme 3.** Construction de l'arbre de Huffman

**Entrées :** L : Liste d'arbres à un nœud pour chaque symbole contenant le symbole ainsi que sa probabilité d'apparition
**Sorties :** A : Arbre de Huffman

```
1 Function ContruireArbreHuffman
2     tant que la liste contient plus d'un nœud faire
3         Rechercher les deux arbres a_1 et a_2 dont les probabilités sont les plus faibles;
4         Retirer a_1 et a_2 de la liste L;
5         Créer un nœud N ayant comme probabilité la somme des probabilités des
            arbres a_1 et a_2;
6         Définir comme fils de N les arbres a_1 et a_2;
7         Insérer ce nouvel arbre L;
8     fin
9     retourner L'arbre restant de la liste
```

De manière à illustrer l'algorithme de construction de cet arbre, nous proposons un exemple en utilisant l'alphabet suivant A, B, C, D, E avec les probabilités correspondantes 0.1, 0.2, 0.15, 0.5, 0.05. La sous-figure 103a représente la liste d'arbres qui sont initialement composés de seulement un nœud. Les nœuds entourés d'un rectangle pointillé sont les nœuds racines des arbres présents dans la liste d'arbres. Les deux nœuds colorés A et E qui ont les probabilités les plus faibles sont utilisés pour faire un nouvel arbre. Cet arbre est constitué d'un nouveau nœud donc la probabilité est la somme des probabilités de ces fils et des deux nœuds comme fils de ce nouveau nœud. Le résultat de cette opération est visible dans la sous-figure 103b. Les deux nouveaux nœuds sélectionnés sont maintenant la racine de l'arbre qui vient d'être créé et le nœud C. Un nouvel arbre est créé avec une probabilité de 0.3 et comme fils le nœud C et l'ancien arbre. La sous-figure 103c synthétise cette étape. L'algorithme continue ainsi jusqu'à n'avoir qu'un nœud dans la liste d'arbres. Ce nœud est la racine de l'arbre de Huffman. Le résultat de l'algorithme est donc l'arbre visible dans la sous-figure 103e. De manière à déduire les codes pour chaque symbole, il faut parcourir en profondeur cet arbre de la racine vers la feuille du symbole en question. Quand nous effectuons une descente vers la branche gauche, nous associerons la valeur binaire 0 et pour une descente vers la branche droite une valeur binaire 1. Toutes les valeurs binaires concaténées constituent le code. De ce fait, pour avoir le code du symbole C, nous effectuons deux descentes à droite




(f)

| Symbole | Code |
|---|---|
| A | 1110 |
| B | 10 |
| C | 110 |
| D | 0 |
| E | 1111 |

**Figure 103.** Exemple d'élaboration d'un arbre de Huffman. Les sous-figures 103a,103b,103c et 103d montre les étapes de construction de l'arbre de Huffman. La sous-figure 103e montre l'arbre final et la sous-figure 103f expose la correspondance entre les symboles et les codes d'après le codage de Huffman.

puis une descente à gauche ce qui correspond au code 110. Le tableau 103f montre la correspondance entre les symboles de l'alphabet et ses codes. Nous remarquons de ce codage que les symboles les plus probables sont les plus courts. Le décodage est réalisé en suivant l'algorithme 4.

**Algorithme 4.** Décompression d'un message compressé avec le codage de Huffman

**Entrées :** Suite binaire
**Sorties :** Message décodé
**Données :** Tampon binaire

```
1 Function décoder Huffman
2     tant que il reste des bits à lire faire
3         Lire un bit;
4         Ajouter le bit au tampon;
5         si tampon = un des codes de Huffman alors
6             Écrire le symbole correspondant dans la sortie;
7             Vider tampon;
8         fin
9     fin
```

Annexe C

# Évaluation complète de ROI-Waaves

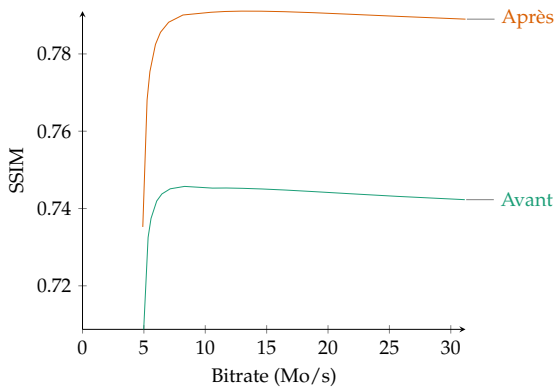
Pour toutes les évaluations, les sous-figures sont organisées suivant le tableau 14

**Tableau 14.** Récapitulatif de l'ordre des séquences évaluées

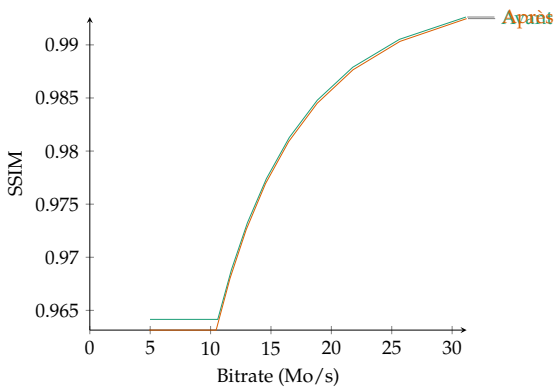
| Sous-figure | Séquence |
|---|---|
| (a) | toddler image entière |
| (b) | toddler ROI seulement |
| (c) | news image entière |
| (d) | news ROI seulement |
| (e) | suzie image entière |
| (f) | suzie ROI seulement |
| (g) | wide tête image entière |
| (h) | wide tête ROI seulement |
| (i) | wide mains image entière |
| (j) | wide mains ROI seulement |
| (k) | close tête image entière |
| (l) | close tête ROI seulement |
| (m) | close mains image entière |
| (n) | close mains ROI seulement |

## 1 Différence

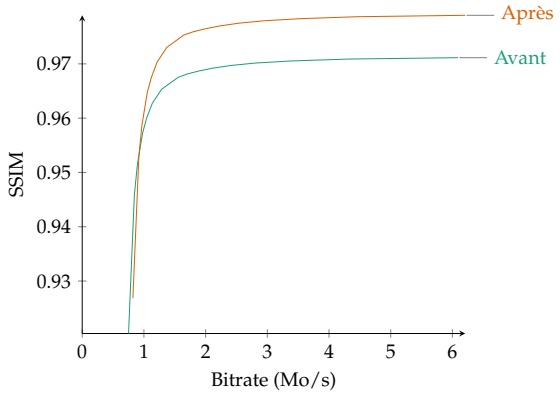
**(a)** toddler



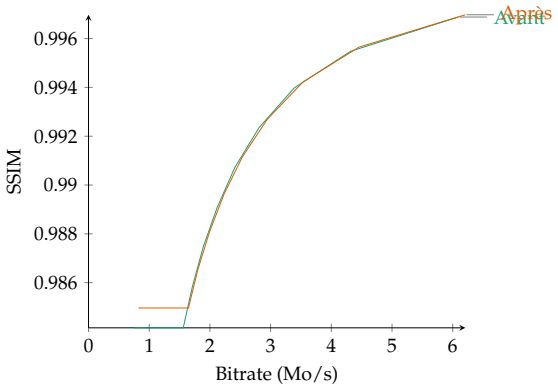

**(b)** toddler ROI



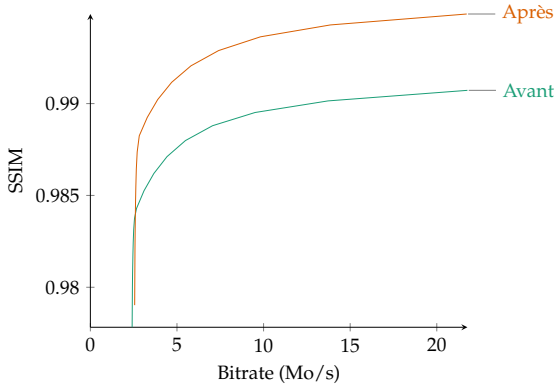
**(c)** news



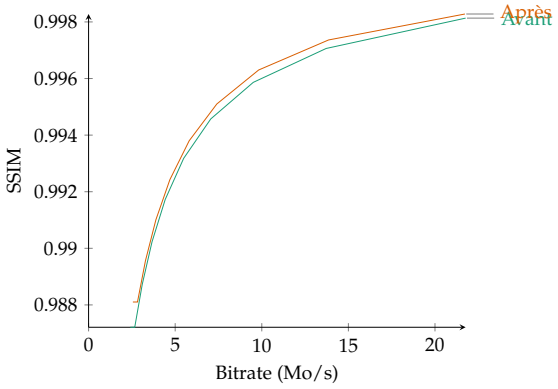

**(d)** news ROI



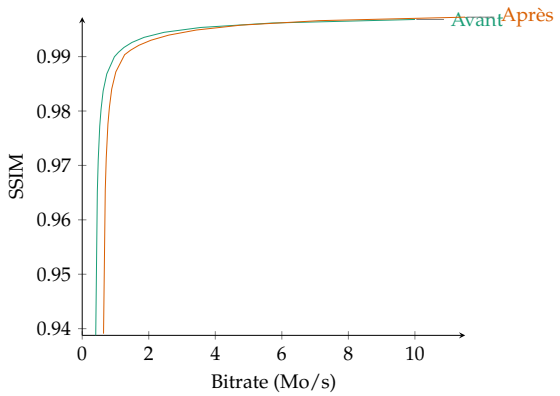

**(e)** suzie



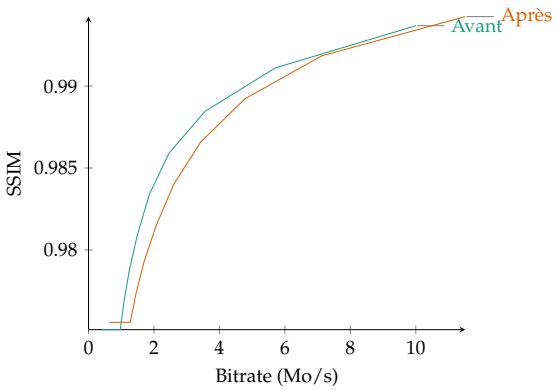

**(f)** suzie ROI



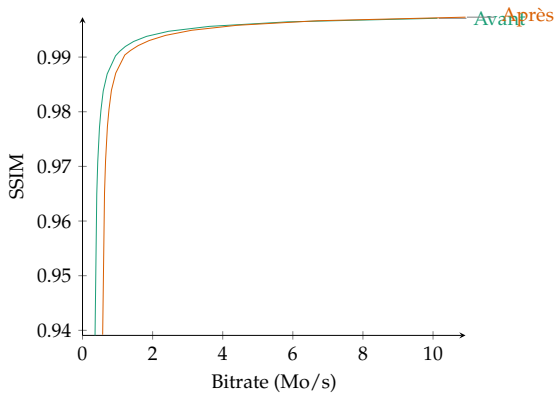

**(g)** wide tête



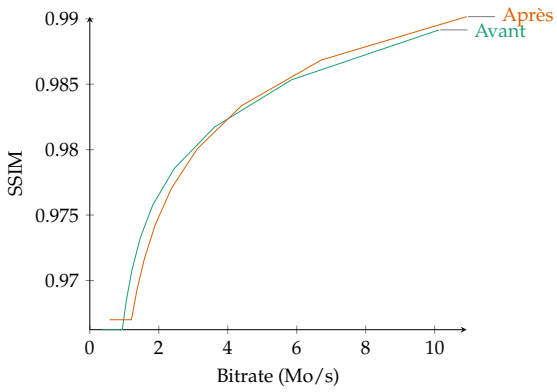

**(h)** wide tête ROI



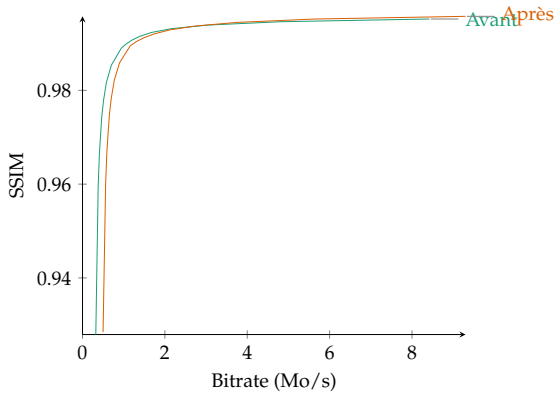
**(i)** wide mains



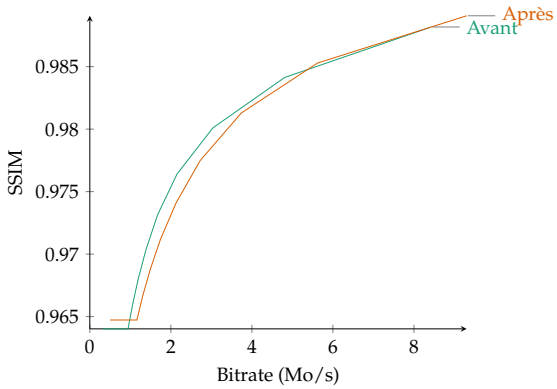

**(j)** wide mains ROI



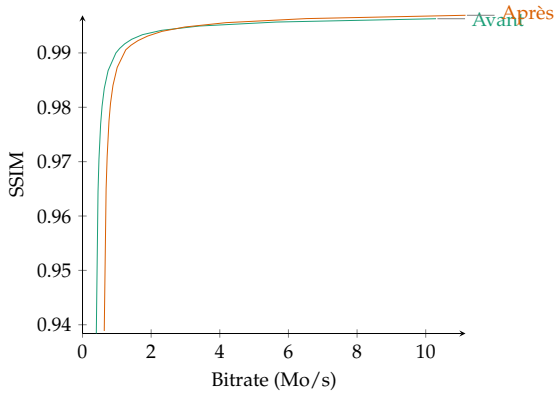

**(k)** close tête




**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI

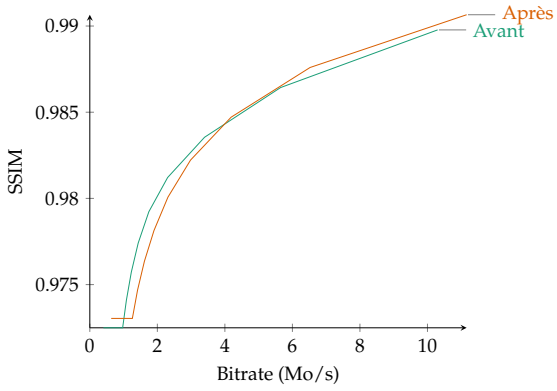
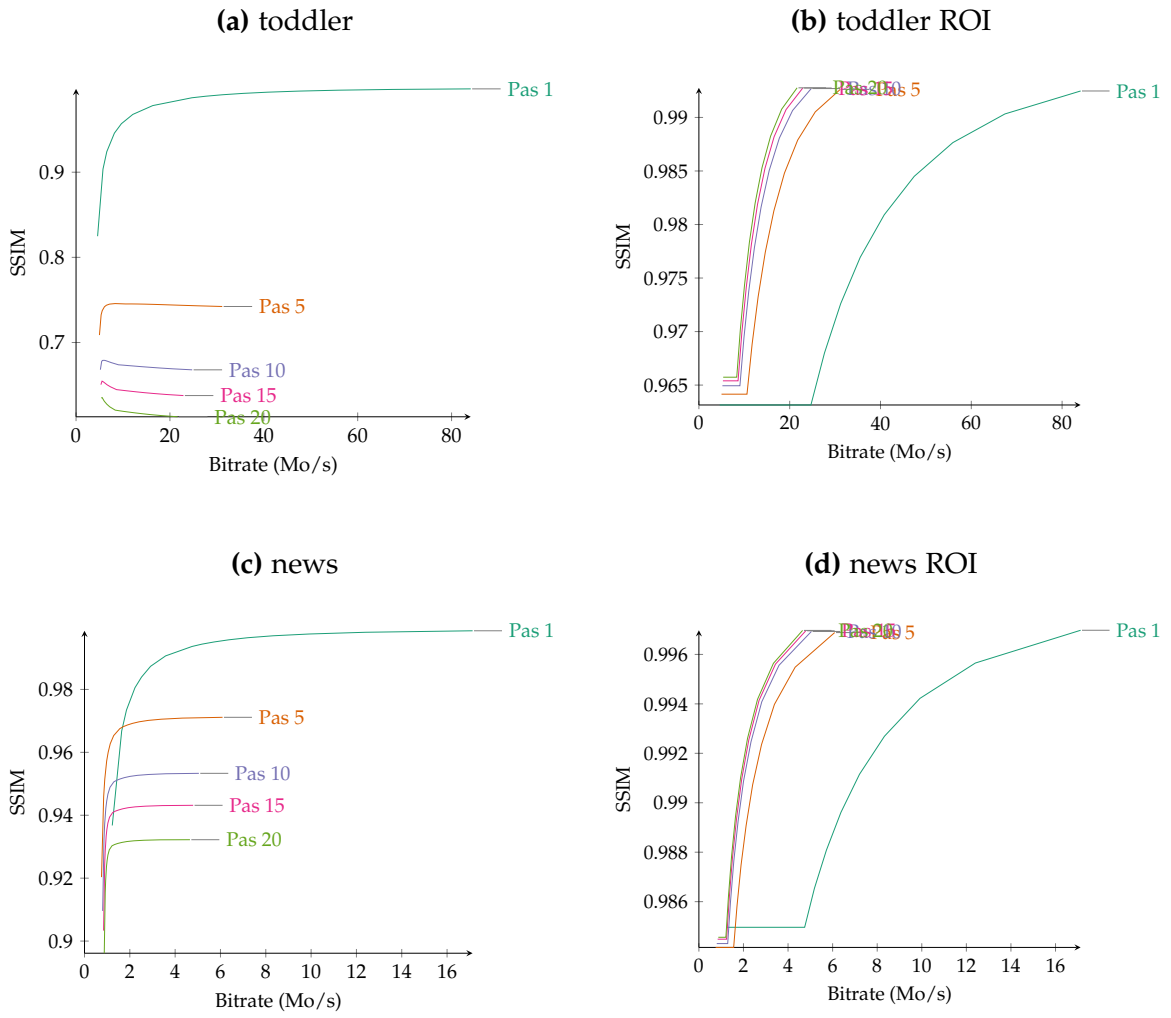
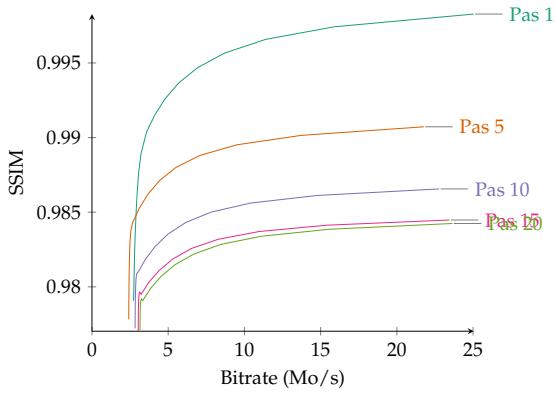



**Figure 105.** Vitesse d'encodage de ROI-Waaves avant et après la différence

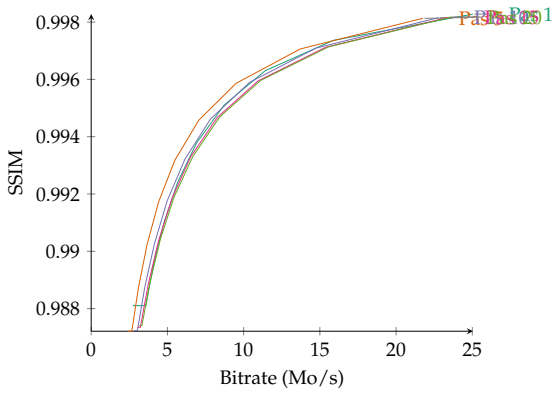
# 2 rafraîchissement de l'image de référence



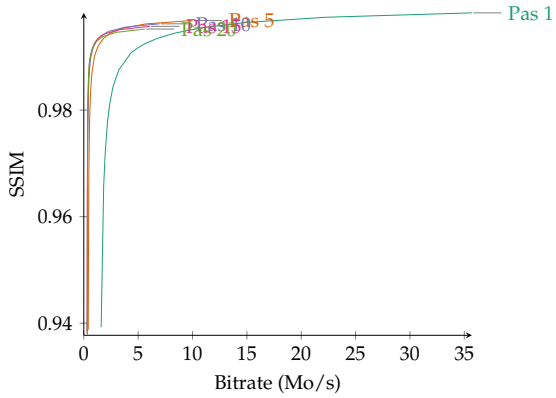
**(e)** suzie



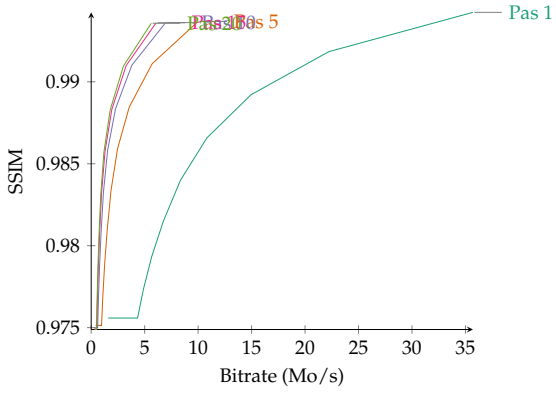

**(f)** suzie ROI



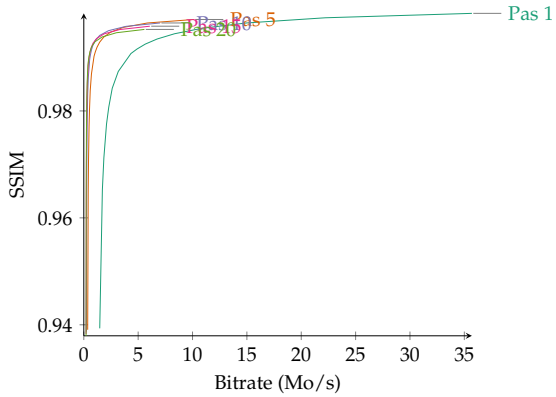

**(g)** wide tête



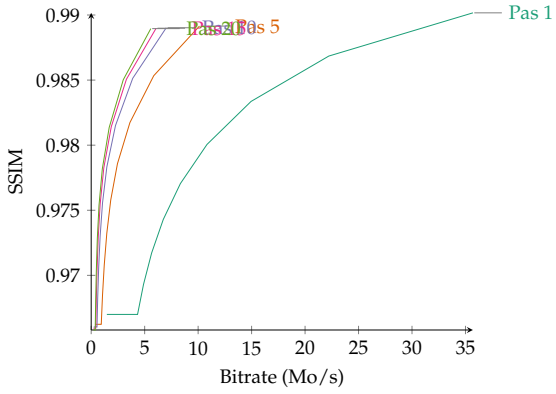

**(h)** wide tête ROI



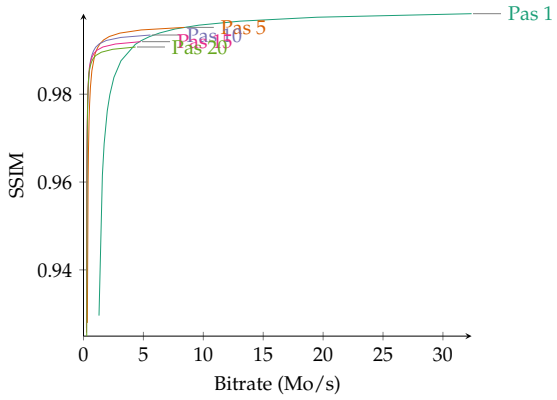

**(i)** wide mains



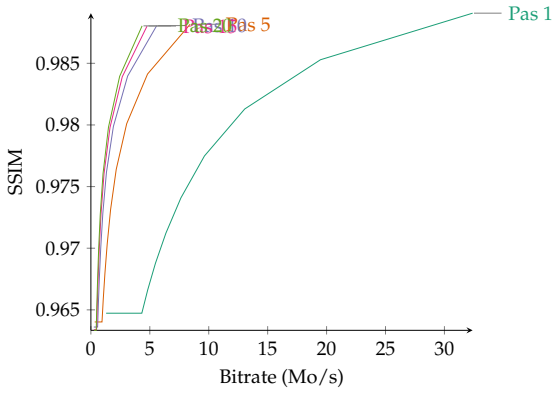

**(j)** wide mains ROI



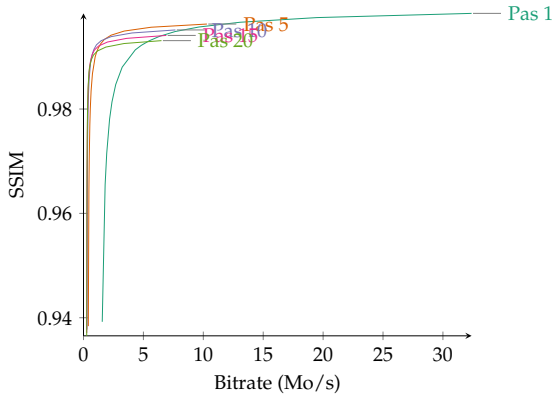
**(k)** close tête




**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI

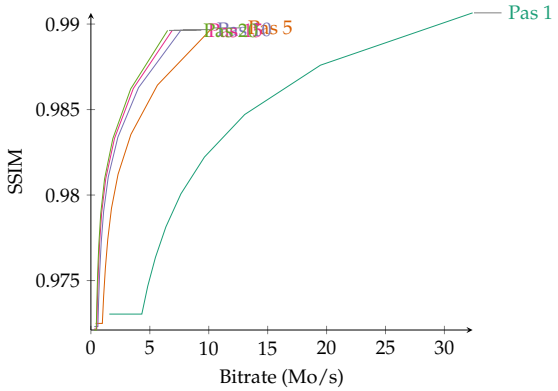
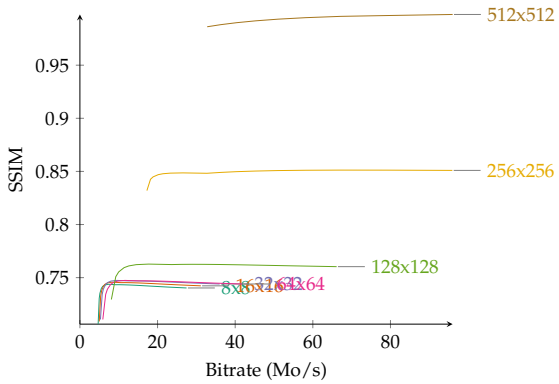




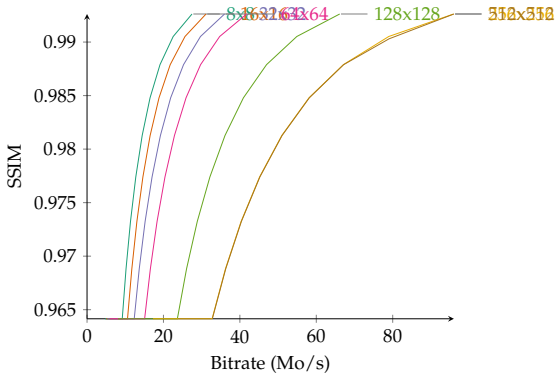
**Figure 107.** Vitesse d'encodage de ROI-Waaves pour différents pas de rafraîchissement

# 3 Taille bloc de parcours

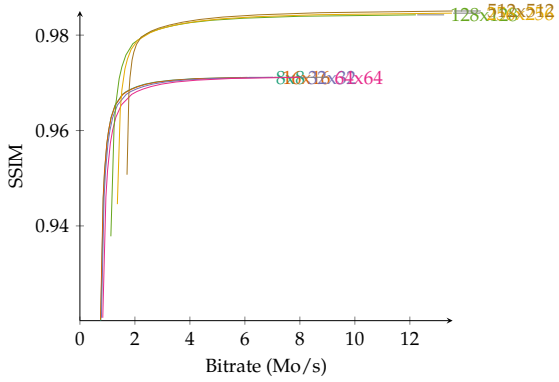
**(a)** toddler



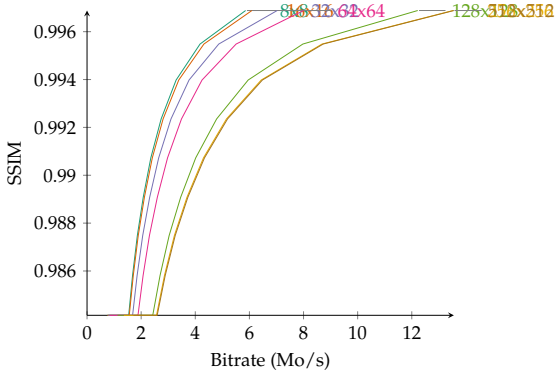

**(b)** toddler ROI



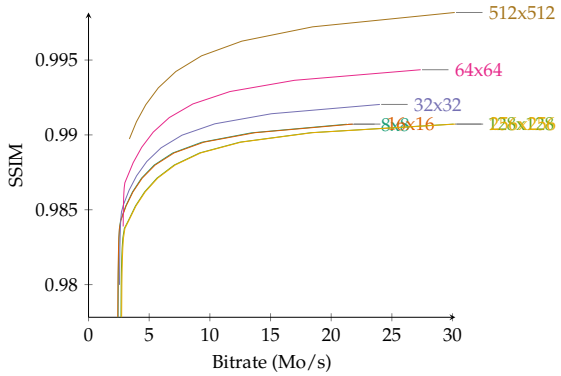

**(c)** news



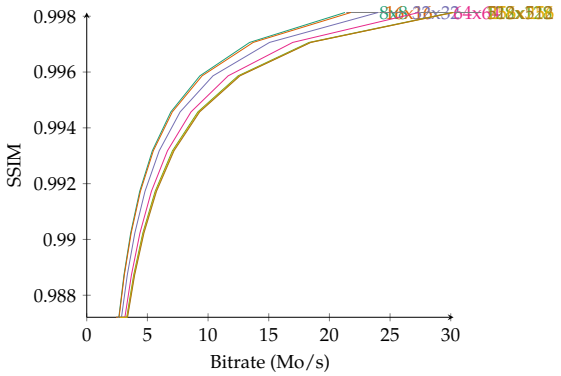

**(d)** news ROI



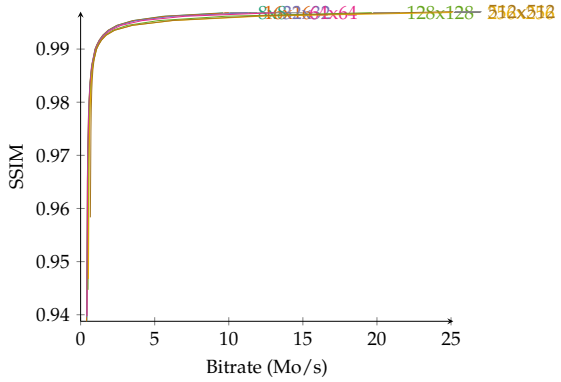
**(e)** suzie



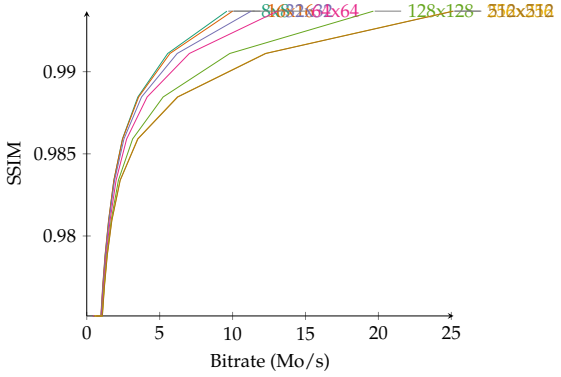

**(f)** suzie ROI



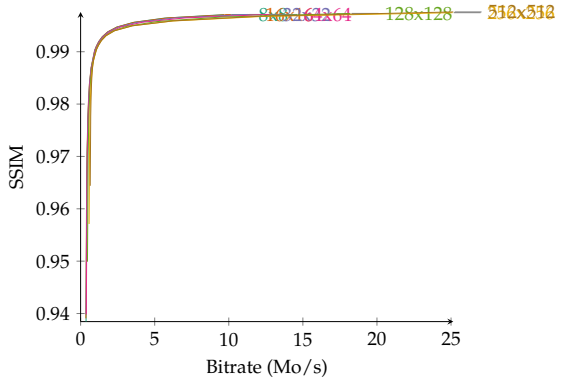

**(g)** wide tête



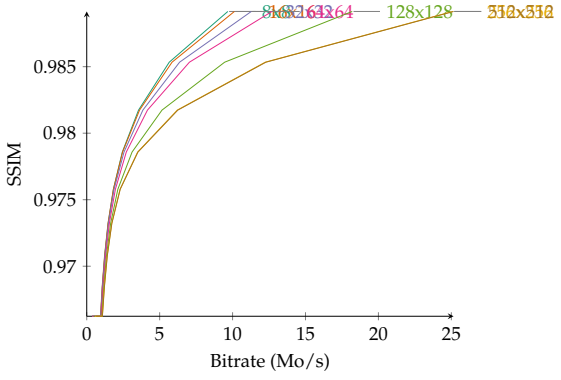

**(h)** wide tête ROI



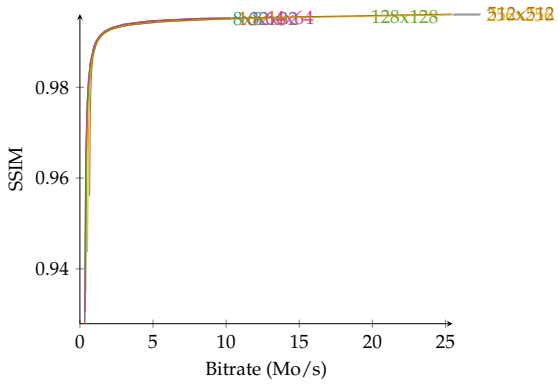

**(i)** wide mains



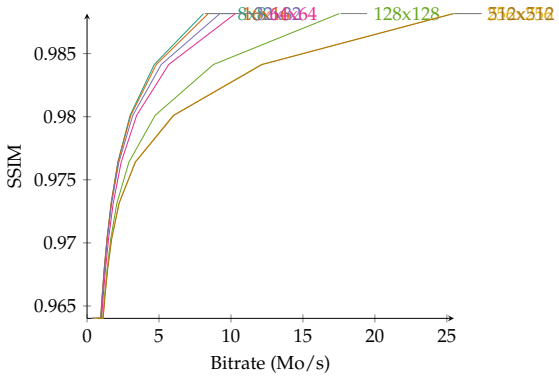

**(j)** wide mains ROI



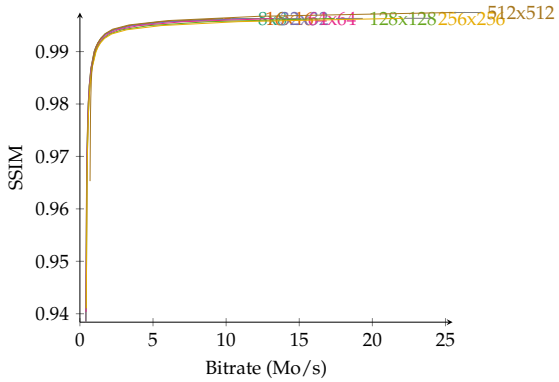
**(k)** close tête



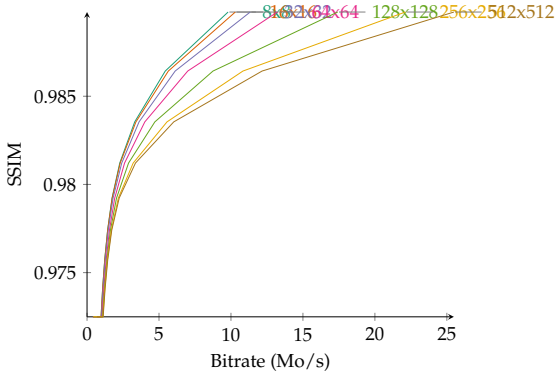

**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI



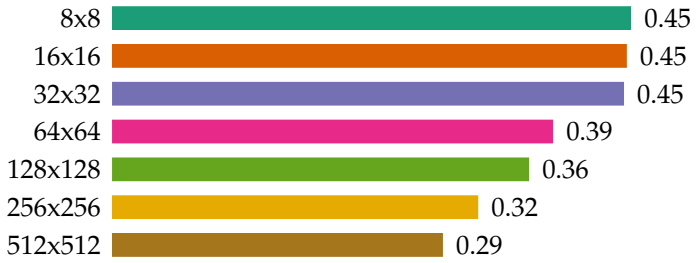
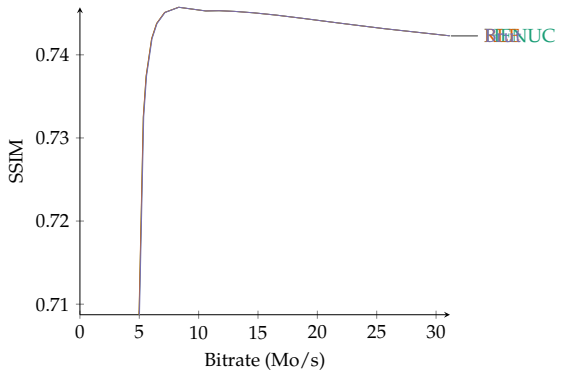



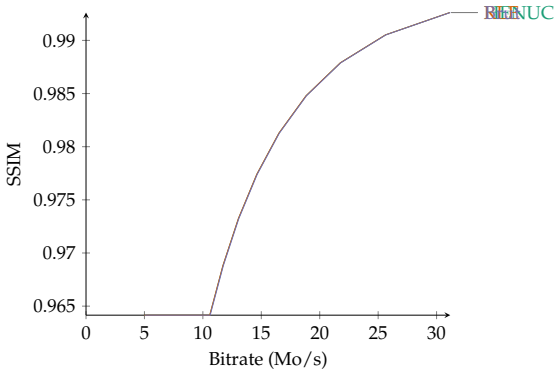

**Figure 109.** Vitesse d'encodage de ROI-Waaves pour différentes tailles de bloc

# 4 Encodage ROI

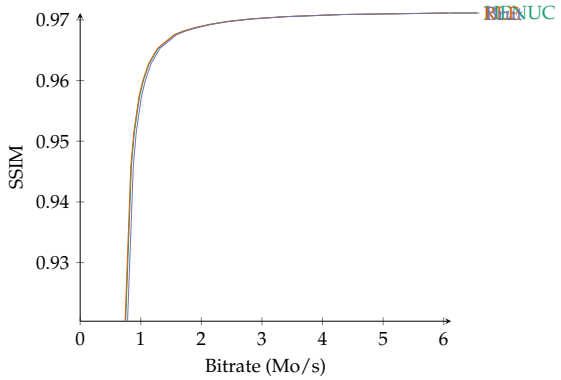
**(a)** toddler



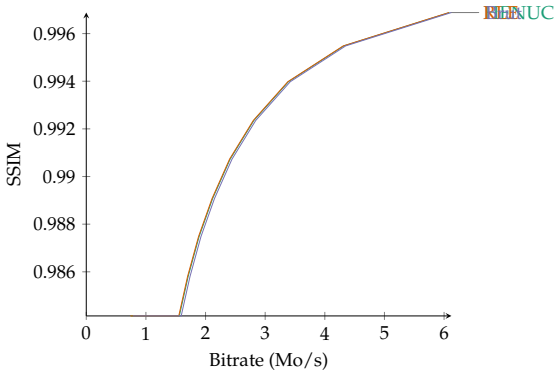

**(b)** toddler ROI



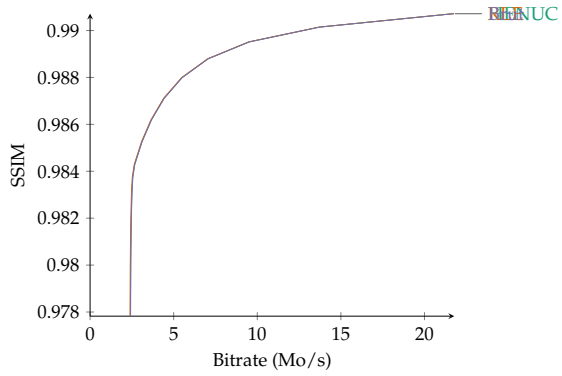

**(c)** news



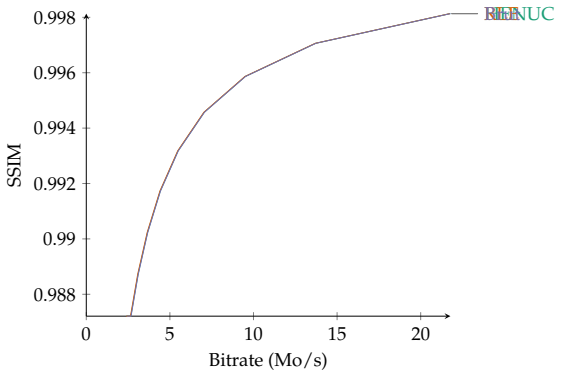

**(d)** news ROI



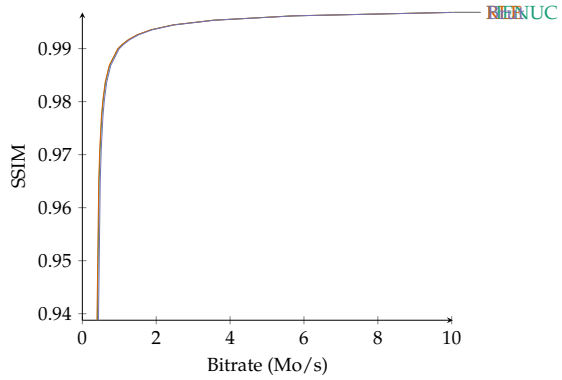
**(e)** suzie



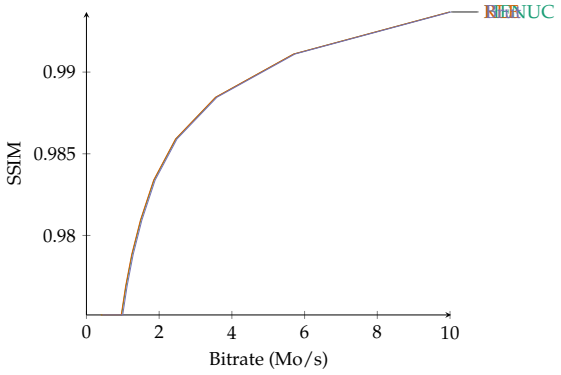

**(f)** suzie ROI



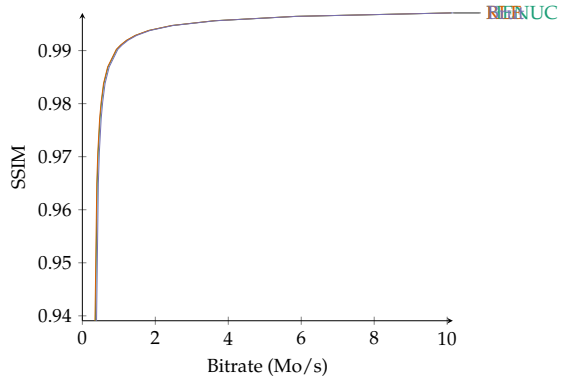

**(g)** wide tête



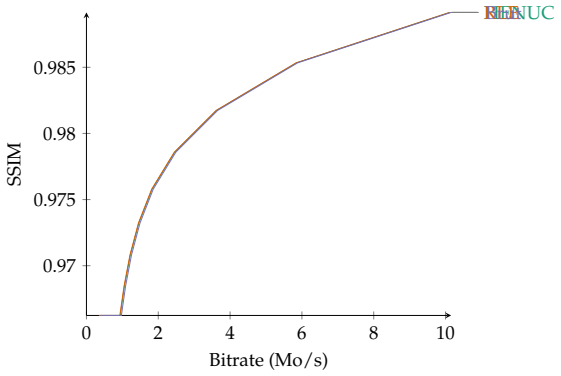

**(h)** wide tête ROI



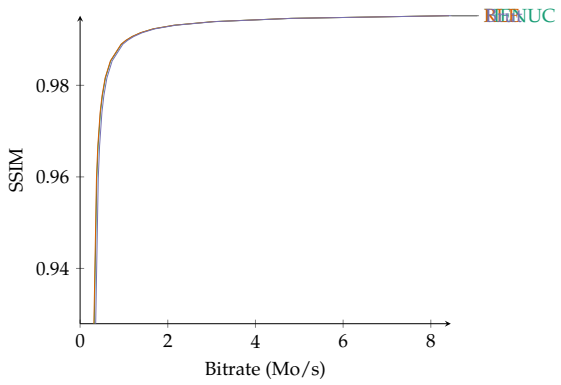

**(i)** wide mains



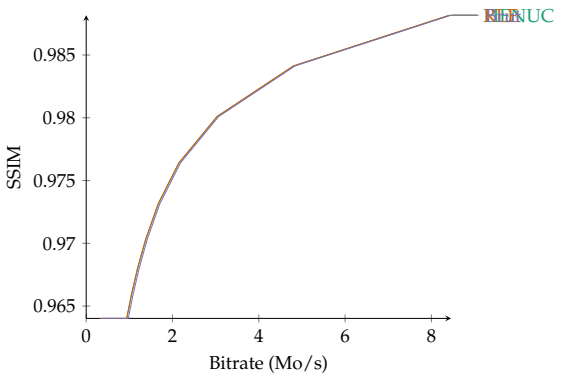

**(j)** wide mains ROI



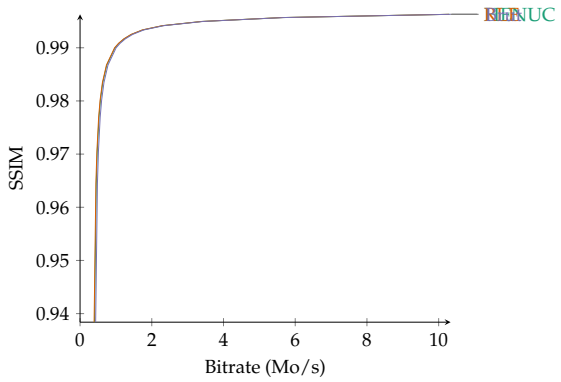
**(k)** close tête




**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI

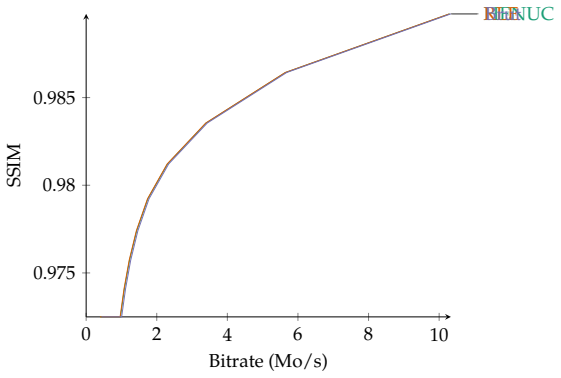
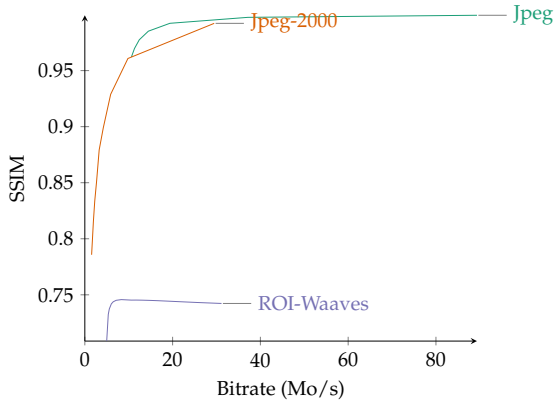




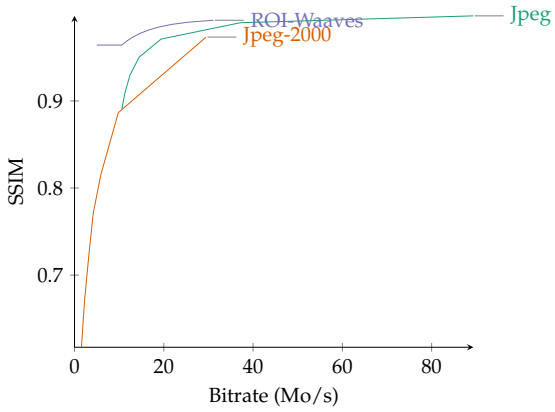
**Figure 111.** Vitesse d'encodage de ROI-Waaves pour différentes méthodes d'encodage de la ROI

# 5 Comparaison JPEG, JPEG-2000, ROI-Waaves

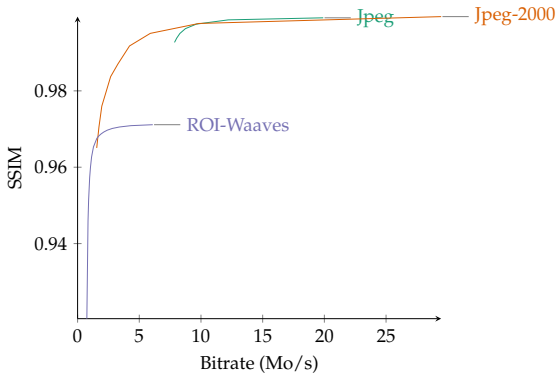
**(a)** toddler



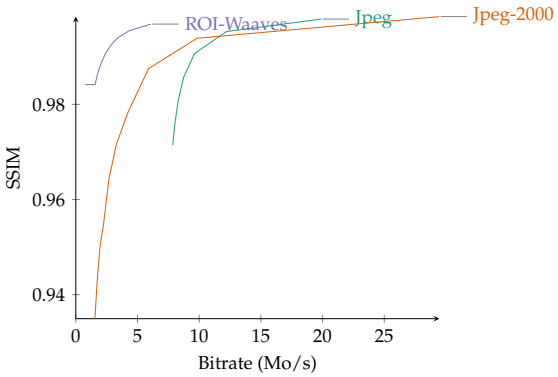

**(b)** toddler ROI



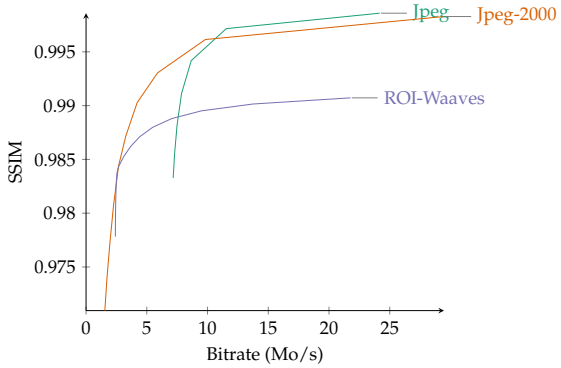

**(c)** news



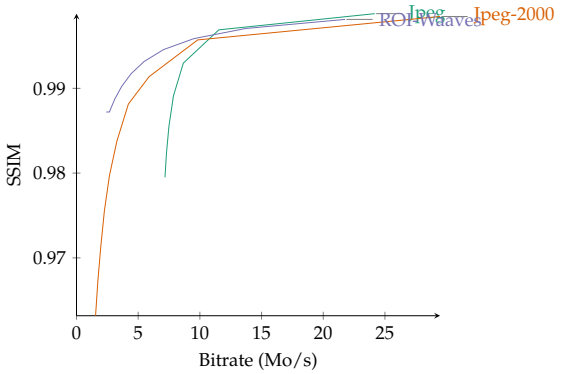

**(d)** news ROI



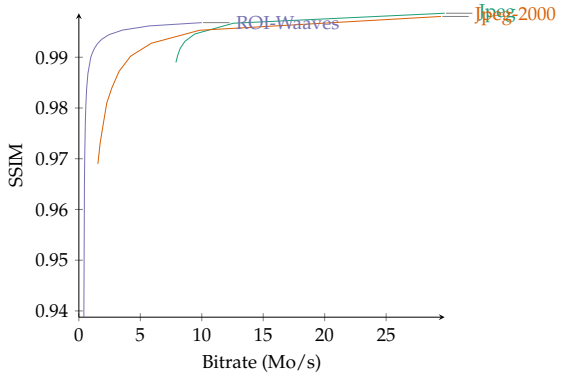
**(e)** suzie



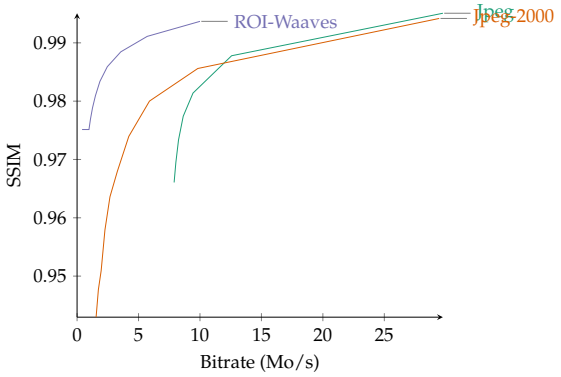

**(f)** suzie ROI



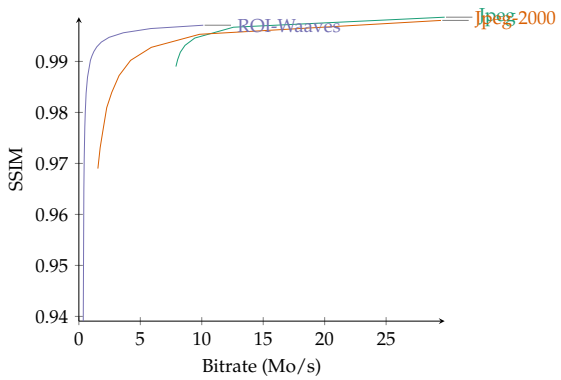

**(g)** wide tête



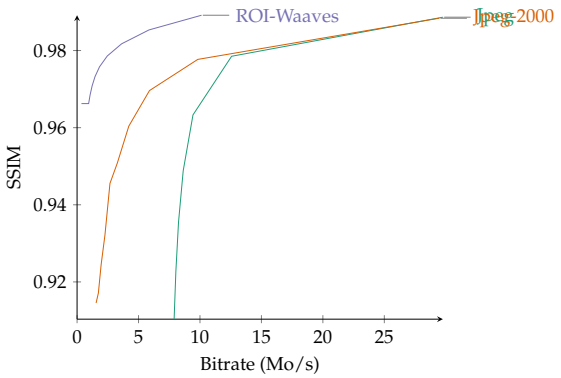

**(h)** wide tête ROI



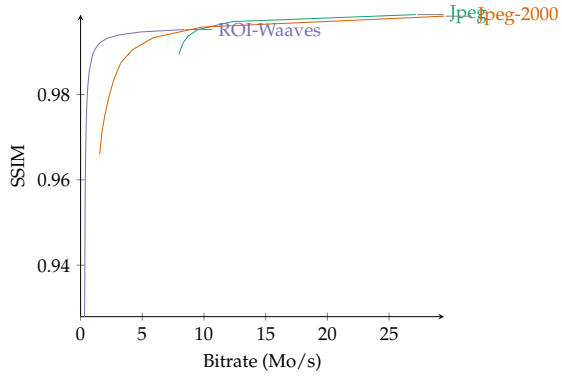

**(i)** wide mains



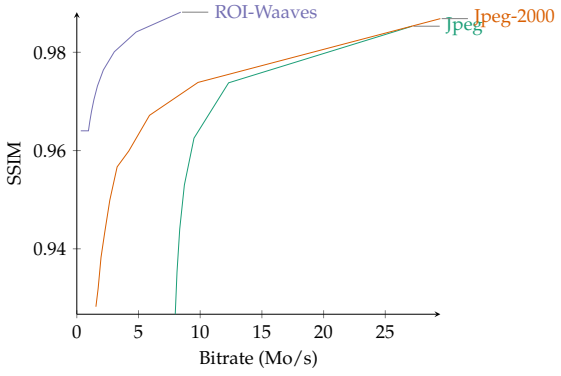

**(j)** wide mains ROI



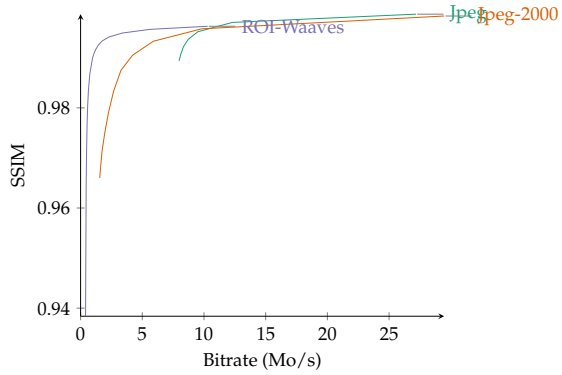
**(k)** close tête




**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI

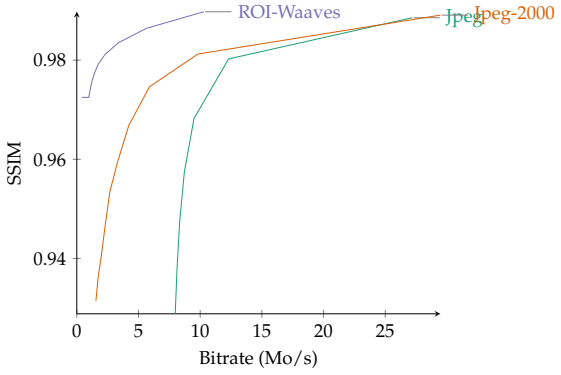
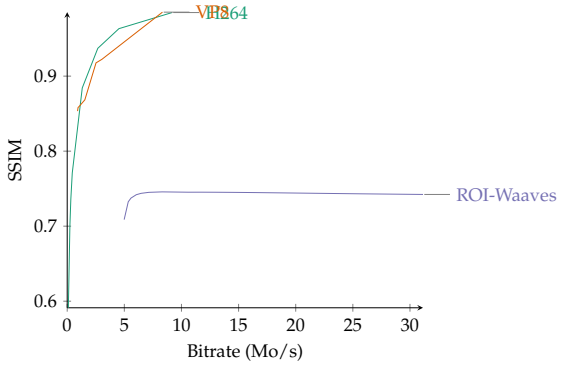




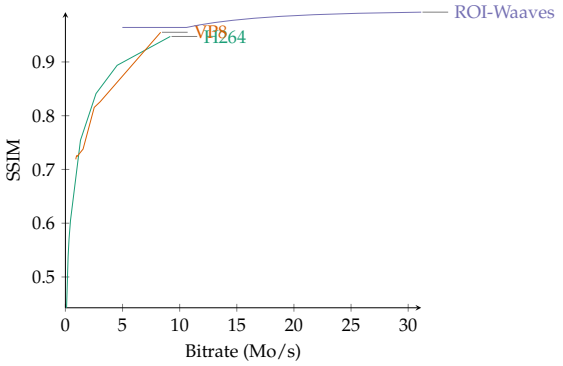
**Figure 113.** Vitesse d'encodage de ROI-Waaves, de Jpeg et de Jpeg-2000

# 6 Comparaison H264, VP8, ROI-Waaves

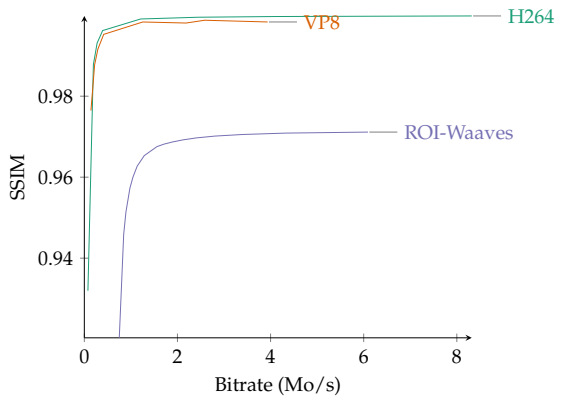
**(a)** toddler



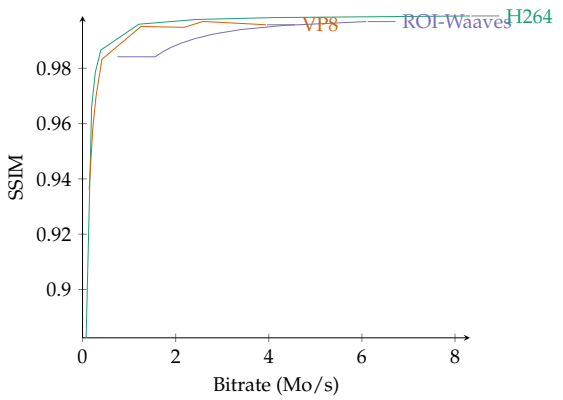

**(b)** toddler ROI



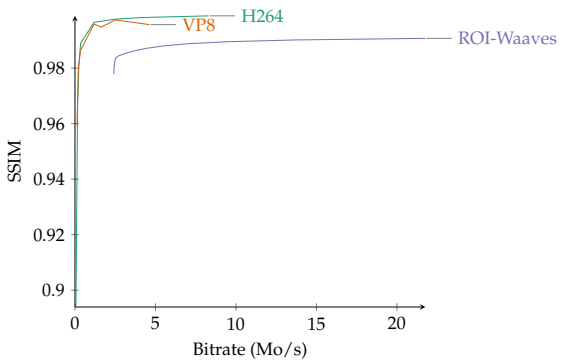

**(c)** news



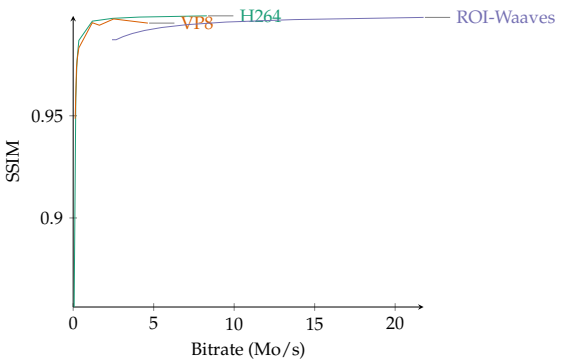

**(d)** news ROI



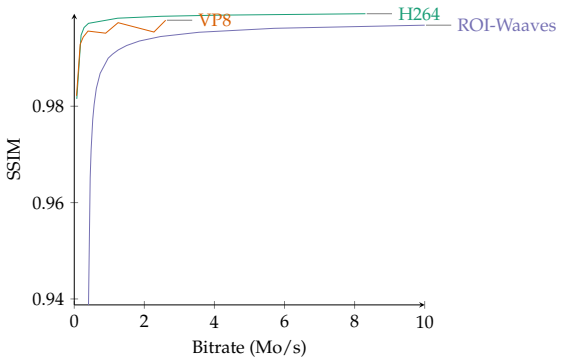

**(e)** suzie



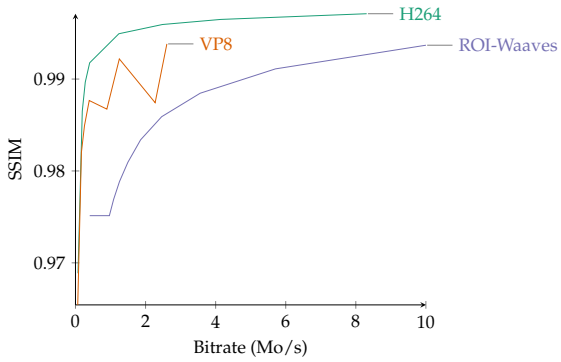

**(f)** suzie ROI



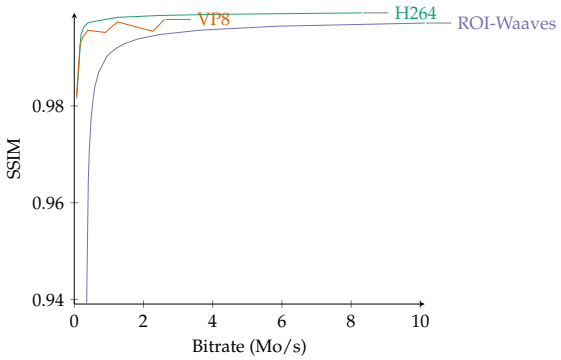
**(g)** wide tête



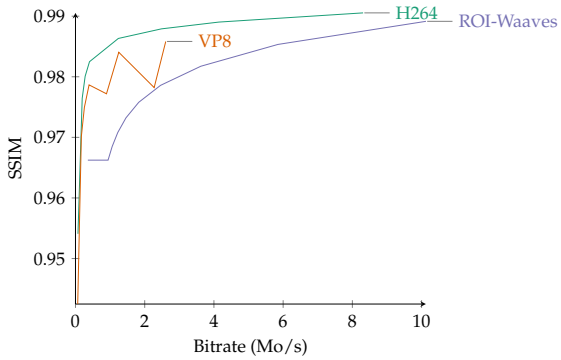

**(h)** wide tête ROI



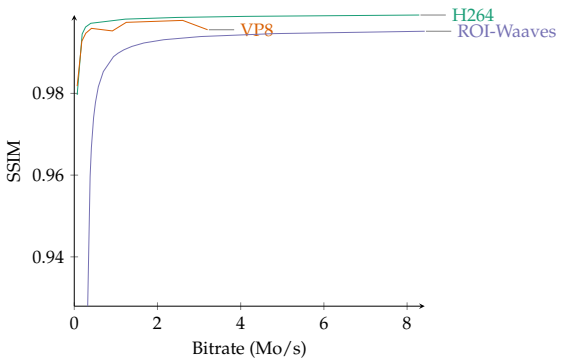

**(i)** wide mains



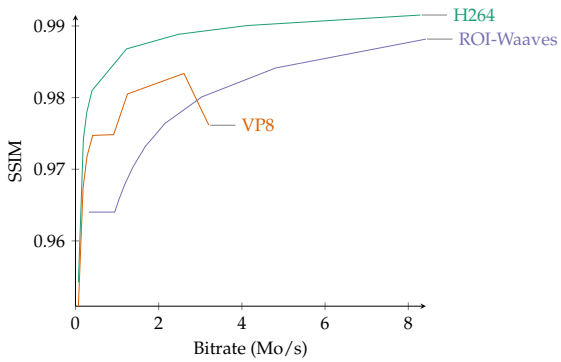

**(j)** wide mains ROI



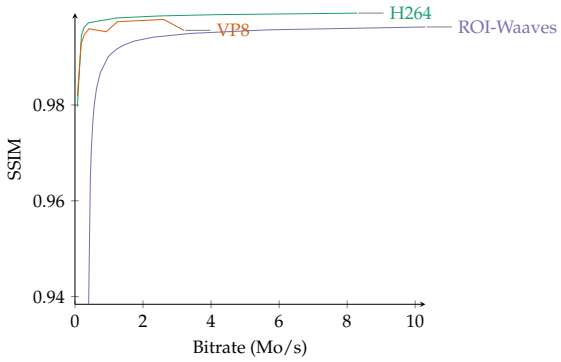

**(k)** close tête



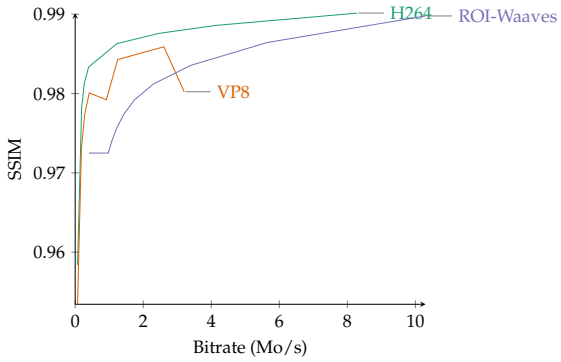

**(l)** close tête ROI



**(m)** close main




**(n)** close mains ROI



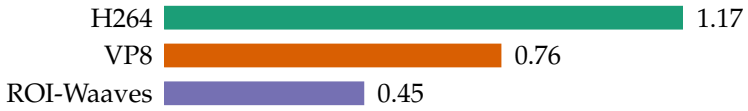
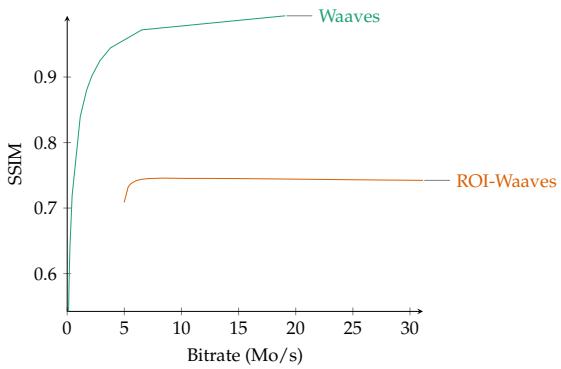



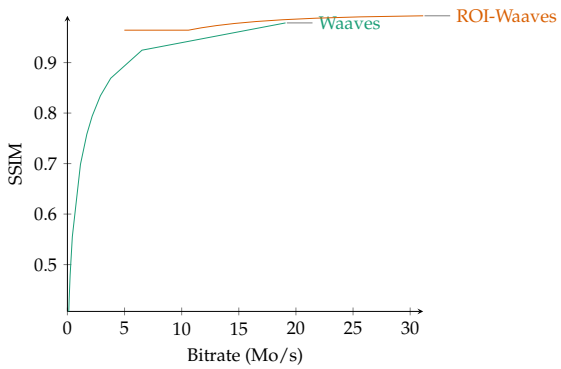

**Figure 115.** Vitesse d'encodage de ROI-Waaves, de H264 et de VP8

# 7 Comparaison Waaves, ROI-Waaves

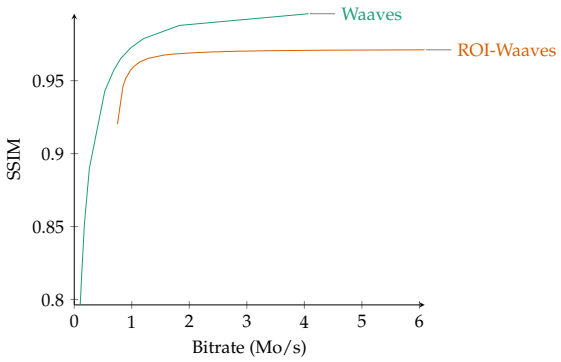
**(a)** toddler



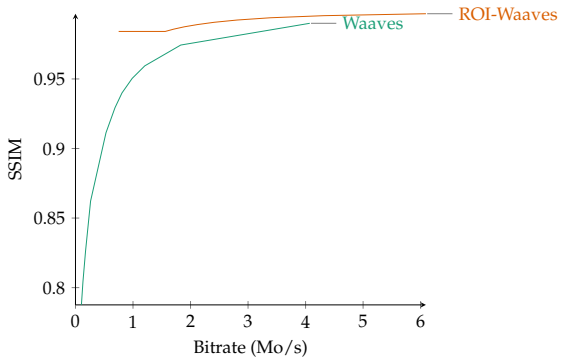

**(b)** toddler ROI



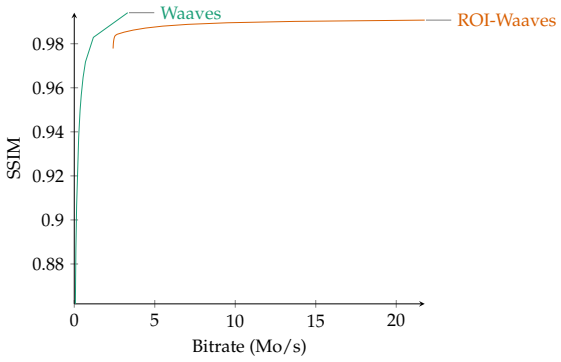
**(c)** news



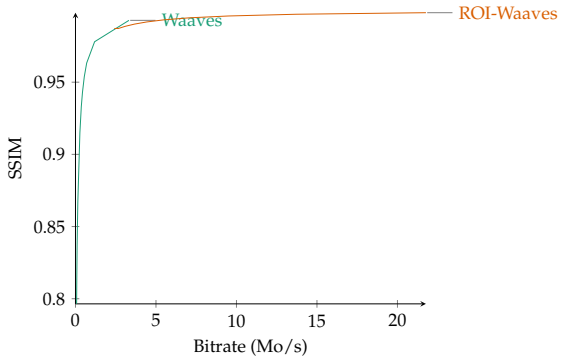

**(d)** news ROI



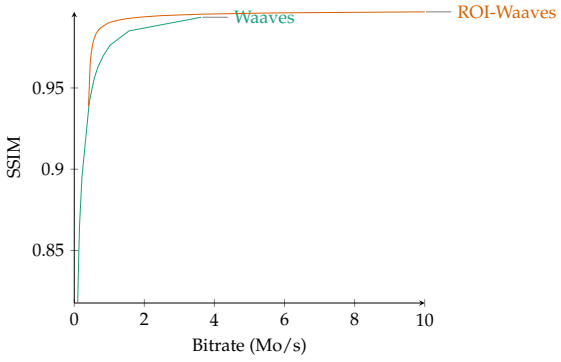

**(e)** suzie



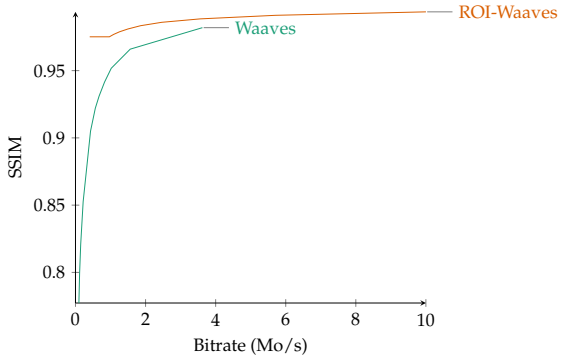

**(f)** suzie ROI



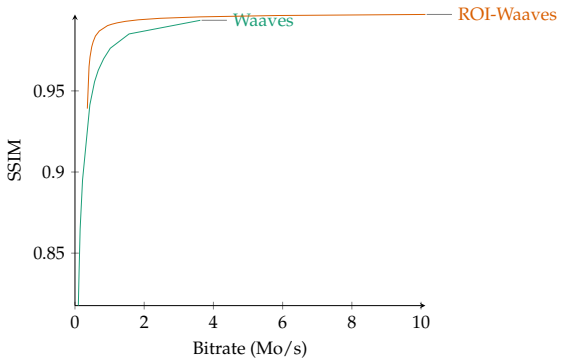

**(g)** wide tête



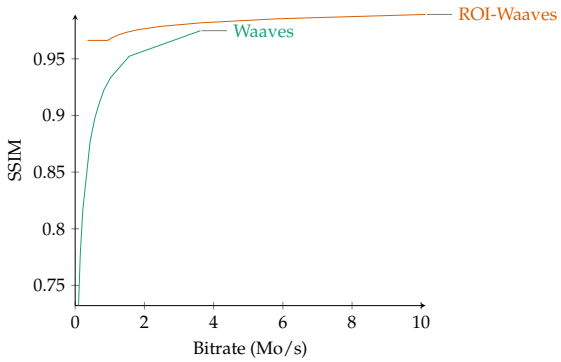

**(h)** wide tête ROI



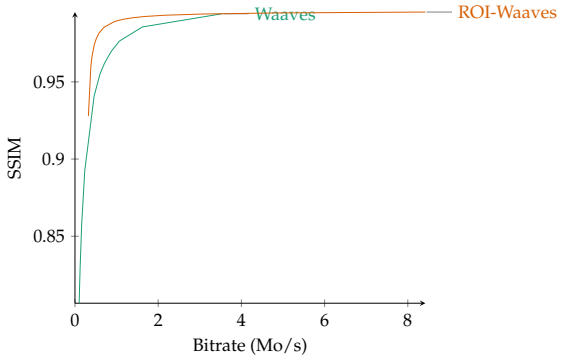
**(i)** wide mains



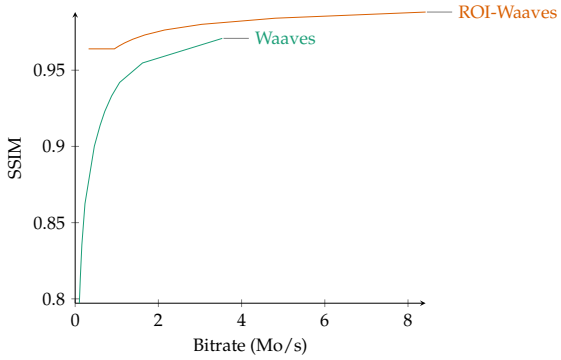

**(j)** wide mains ROI



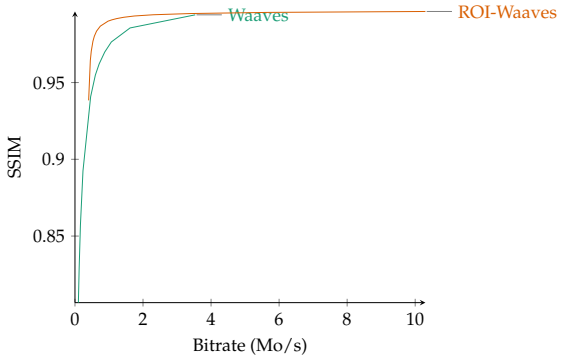

**(k)** close tête



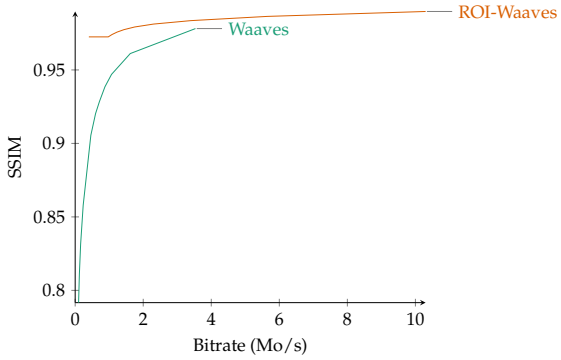

**(l)** close tête ROI




**(m)** close main




**(n)** close mains ROI



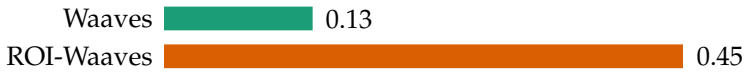




**Figure 117.** Vitesse d'encodage de ROI-Waaves et de Waaves

# Résumé

L'examen EEG est un examen fonctionnel essentiel de la médecine moderne. Il consiste dans l'acquisition de signaux physiologique d'un patient afin de diagnostiquer ses troubles neurologiques. De manière à compléter ces signaux, une vidéo du patient est également fournie. À cause d'une mauvaise répartition des experts et d'une diversification des spécialités de neurophysiologie, la téléexpertise s'impose comme une solution pour apporter des soins pour tous.

Cette thèse aborde la problématique de téléexpertise d'un examen EEG tout en repoussant les limitations des équipements d'EEG actuelle. Ces limitations, observées par les médecins, sont l'imprécision de la synchronisation entre les signaux physiologiques et l'enregistrement vidéo et la faible qualité de cet enregistrement vidéo.

La première limitation nous a conduit à développer un mécanisme de synchronisation matérielle déterministe ainsi que de définir un nouveau format de fichier capable de stocker les signaux physiologiques ainsi que la vidéo de manière uniforme. Grâce à ces deux mécanismes, nous pouvons garantir une synchronisation lors de la relecture de l'examen EEG.

La seconde limitation a mené en la définition d'un nouvel algorithme de compression nommé ROI-Waaves. Cet algorithme, utilisant la transformée en ondelettes et le codeur entropique HENUC, est capable d'encoder des zones de l'image avec une qualité supérieure afin de conserver les détails. Ces zones sont définies comme les mains et la tête du patient permettant aux médecins de faire des diagnostics correctes.

Finalement, nous avons développé deux implémentations mettant en place les solutions proposées et permettant de réaliser un examen EEG synchronisé et compressé. Les examens produits par ces dispositifs sont complets et peuvent être utilisés pour relecture ou archivage. De plus, nous avons proposé une architecture matérielle compressant un flux vidéo à 100 images par seconde en temps réel en utilisant ROI-Waaves.

---

# Abstract

The EEG exam is an essential functional exam in modern medicine. It involves the acquisition of biological signals of a patient to diagnose his neurological disorders. In order to supplement those signals, a video recording of the patient is also provided. Because of a poor spread of experts and diversification of neurophysiological specialties, remote expertise imposes itself as a solution to bring health care to most people.

The thesis approaches the problematic of remote diagnosis of EEG exams while pushing back the current EEG device limitations. Those limitations, observed by doctors, are the imprecise synchronization between biological signals and the video recording and the low quality of this video recording.

The first limitation has led us to develop a hardware determinist synchronization as well as to define a novel file format to store biological signals alongside the video. Thanks to those two mechanisms, we guarantee the correct synchronization when reviewing a EEG file.

The second limitation has driven us to define a new compression algorithm named ROI-Waaves. The algorithm uses wavelet transform and the HENUC coding scheme and encodes one or many regions of an image with higher quality to preserve details. Those regions are defined as the hands and head of the patient and help doctors to perform correct diagnoses.

Lastly, we have developed two implementations which use the proposed solutions and allowing to carry out a synchronized and compressed EEG exam. The produced exams by these devices are complete and can be used for review or long-time storage. In addition, we have proposed a hardware architecture that compresses 100 frames per second video stream in real time using ROI-Waaves.