

12/09/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 1

Introduction.....	2
1. Conception des composants.....	3
1.1. SourceFixe.....	3
1.1. SourceAléatoire.....	3
1.2. TransmetteurParfait.....	3
1.3. DestinationFinale.....	3
2. Connexion des composants.....	4
2.1. Connexion de SourceFixe au TransmetteurParfait.....	4
2.2. Connexion du TransmetteurParfait à DestinationFinale.....	4
3. Simulation de l'émission et de la réception.....	4
3.1. Émission depuis SourceFixe.....	4
3.2. Réception de l'Information par DestinationFinale.....	4
4. Calcul du taux d'erreur binaire (TEB).....	5
4.1. Algorithme du Calcul du TEB.....	5
5. Tests.....	6
5.1. Organisation des tests.....	6
5.2. Implémentation des tests.....	7
5.3. Résultats des tests.....	7
5.4. Couverture de code avec Emma.....	7
6. Simulation d'une transmission par un canal parfait.....	9
6.1. Composants impliqués.....	9
6.2. Étapes de la simulation.....	9
6.3. Résultats attendus.....	10
6.3. Résultats obtenus.....	11
Conclusion.....	12

Introduction

L'atelier SIT213 vise à nous aider à comprendre et à simuler les concepts de la transmission numérique. On se concentre sur toutes les étapes d'une chaîne de communication, depuis la création d'un signal à la source jusqu'à sa réception à destination, en passant par les divers éléments comme les transmetteurs et les modulateurs.

Pour cette première étape du projet, notre objectif est de poser les bases de la simulation de la transmission numérique en mettant en place les composants essentiels de la chaîne de communication. On va donc travailler avec :

- La **SourceFixe**, qui est responsable de la génération du message à transmettre.
- La **SourceAléatoire**, qui génère un message aléatoire.
- Le **TransmetteurParfait**, qui transmet l'information sans la modifier.
- La **DestinationFinale**, qui reçoit l'information transmise.

Le rapport décrit les actions effectuées pour instancier ces composants, les connecter et mesurer le taux d'erreur binaire (TEB).

1. Conception des composants

1.1. SourceFixe

- **But** : La SourceFixe génère un message binaire de type `Boolean[]`, en fonction d'un String (0 ou 1).
- **Fonctionnalité** : La méthode principale `emettre()` transmet le message au composant suivant de la chaîne de communication via la méthode `connecter(DestinationInterface<T>)`.

1.1. SourceAléatoire

- **But** : La SourceAléatoire génère un message binaire de type `Boolean[]`, en fonction d'une taille et, si indiqué, d'une graine.
- **Fonctionnalité** : La méthode principale `emettre()` transmet le message au composant suivant de la chaîne de communication via la méthode `connecter(DestinationInterface<T>)`.

1.2. TransmetteurParfait

- **But** : Le TransmetteurParfait prend le message émis par la source et le transmet à la destination sans dégradation.
- **Fonctionnalité** : Le transmetteur reçoit une information via `recevoir(Information<T>)` et la retransmet avec la méthode `emettre()`.

1.3. DestinationFinale

- **But** : La DestinationFinale reçoit l'information transmise par le transmetteur et la stocke pour évaluation. Elle est capable de recevoir et de traiter les informations de type `Boolean`.
- **Fonctionnalité** : Elle stocke l'information reçue avec la méthode `recevoir(Information<T>)`, qui peut être consultée via `getInformationRecue()`.

2. Connexion des composants

2.1. Connexion de SourceFixe au TransmetteurParfait

La SourceFixe est connectée au TransmetteurParfait en utilisant la méthode `connecter(DestinationInterface<T>)`. Cela permet à la source de transmettre son message au transmetteur.

2.2. Connexion du TransmetteurParfait à DestinationFinale

Le TransmetteurParfait est ensuite connecté à la DestinationFinale. Cette connexion permet de faire passer l'information reçue par le transmetteur jusqu'à la destination.

3. Simulation de l'émission et de la réception

3.1. Émission depuis SourceFixe

Après la connexion des composants, la méthode `emettre()` de la SourceFixe est appelée. Cette méthode génère un message aléatoire de longueur définie (par exemple 100 bits) et le transmet au TransmetteurParfait.

3.2. Réception de l'Information par DestinationFinale

Le TransmetteurParfait transmet ensuite cette information à la DestinationFinale. La destination utilise la méthode `recevoir(Information<T>)` pour enregistrer le message reçu.

4. Calcul du taux d'erreur binaire (TEB)

Une fois que la source a émis et que la destination a reçu le message, le taux d'erreur binaire (TEB) peut être calculé. Le TEB est calculé en comparant le message émis par la source et le message reçu par la destination.

4.1. Algorithme du Calcul du TEB

Le calcul du TEB est basé sur la proportion de bits erronés dans le message reçu par rapport au message original. Le TEB est donné par la formule suivante :

$$\text{TEB} = \text{Nombre de bits erronés} / \text{Nombre total de bits émis}$$

Pour cela, les méthodes `getInformationEmise()` de la source et `getInformationRecue()` de la destination sont appelés et les messages sont comparés bit par bit.

5. Tests

Nous utilisons JUnit (version 4) pour effectuer les tests unitaires.

De la manière suivante :

- Les fichiers .jar sont situés dans le répertoire `lib`
- compilation des tests via le script `compile`
- Exécution des tests via le script `runTests`, utilisant la classe `AllTests`

Chaque classe et ses méthodes sont testées.

L'objectif principal des tests est de s'assurer que chaque composant du système fonctionne correctement de manière isolée, ainsi que lorsqu'il est intégré avec d'autres composants. Cela inclut la génération de sources aléatoires ou fixes, la transmission des informations via des transmetteurs parfaits, et la réception correcte des informations par les destinations. De plus, le simulateur dans son ensemble est également testé pour vérifier le flux complet de la transmission.

5.1. Organisation des tests

Une suite de tests (`AllTests`) a été configurée pour regrouper et exécuter automatiquement tous les tests des différentes classes. Voici les classes de test incluses :

1. **SourceAleatoireTest** : Teste la génération de messages aléatoires par la classe `SourceAleatoire`.
2. **SourceFixeTest** : Valide la génération et la transmission de messages fixes.
3. **TransmetteurParfaitTest** : Vérifie le comportement du transmetteur parfait, qui doit transmettre l'information sans aucune altération.
4. **DestinationFinaleTest** : Contrôle la réception de l'information par la destination et vérifie l'intégrité des données reçues.
5. **InformationTest** : Teste la structure de la classe `Information`, qui gère les informations échangées entre les différents composants.
6. **SimulateurTest** : Évalue le fonctionnement global du simulateur, en simulant une chaîne de transmission complète et en calculant le taux d'erreur binaire.

5.2. Implémentation des tests

Les tests sont implémentés en utilisant le framework JUnit. La suite de tests est définie à l'aide de l'annotation `@RunWith(Suite.class)` pour exécuter tous les tests de manière centralisée. Nous utilisons la classe `AllTests` pour lancer tous les tests.

Les tests ont pour objectifs de valider les entrées et sorties de chaque méthodes en fonction des attentes.

5.3. Résultats des tests

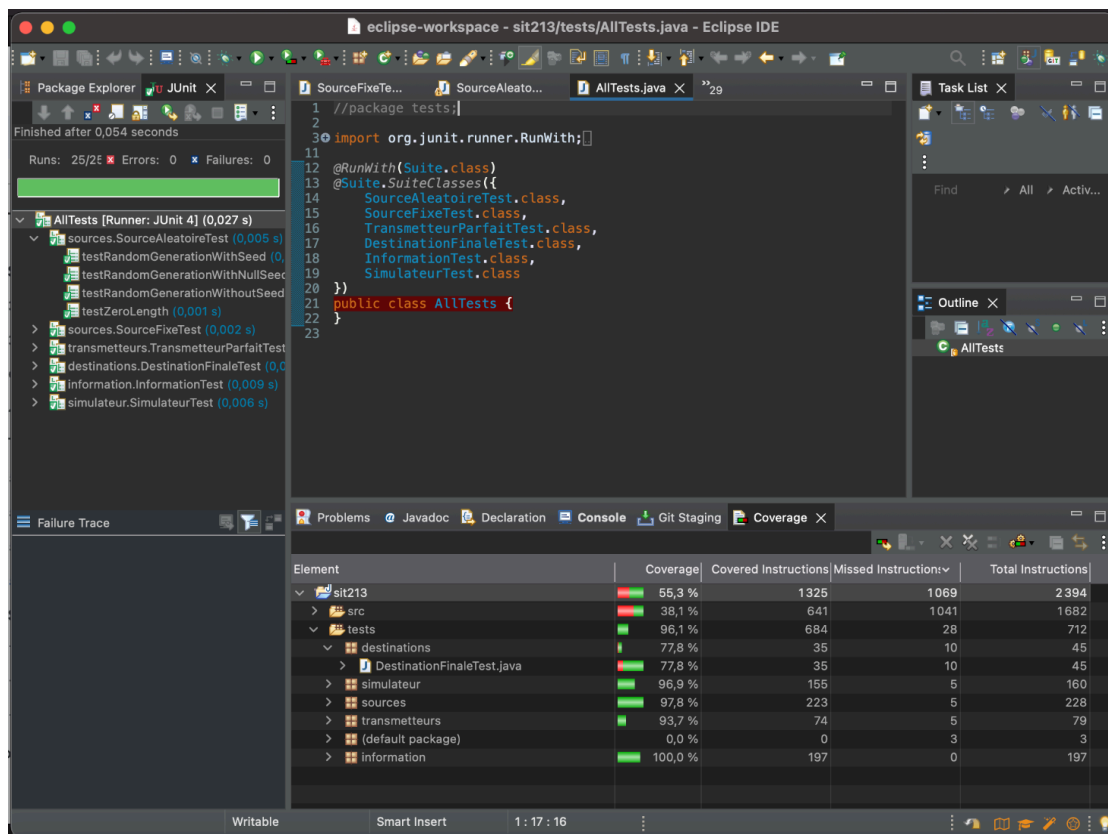
L'exécution des tests a produit les résultats suivants :

- Tous les tests ont passé avec succès, validant le comportement attendu des composants, après quelques changements dans les tests et les sources, notamment grâce à des getters.
- Le simulateur a correctement simulé la transmission d'un message à travers un canal parfait sans erreurs.
- Les classes comme `SourceFixe`, `SourceAleatoire`, `TransmetteurParfait` et `DestinationFinale` ont fonctionné comme prévu, permettant une transmission fidèle des messages avec un taux d'erreur binaire nul dans les scénarios testés.

5.4. Couverture de code avec Emma

Nous avons utilisé l'outil Emma pour mesurer la couverture de code des tests. Voici les résultats globaux et détaillés :

- **Couverture totale** : 96,1 %
- **Couverture des destinations** : 77,8 %
- **Couverture du simulateur** : 96,9 %
- **Couverture des sources** : 97,8 %
- **Couverture des transmetteurs** : 93,7 %
- **Couverture des informations** : 100 %



Ces résultats montrent que l'implémentation a été bien testée, avec une couverture très élevée dans les principaux composants du simulateur. La couverture plus faible pour les destinations (77,8 %) indique qu'il reste des scénarios spécifiques à vérifier, mais globalement, le système est largement testé et fonctionne conformément aux attentes. En effet, certains éléments spécifiques ne sont pas exécutés. Ce qui sera à corriger dans les prochaines itérations. Il manque principalement les tests sur la gestion des arguments et les visualisations, ces classes/méthodes nous ont été fournis. Dans nos classes, les sections non testées sont les méthodes main de debug.

6. Simulation d'une transmission par un canal parfait

Dans cette section, on va simuler la transmission d'un message numérique à travers un canal parfait, c'est-à-dire un canal sans bruit ni interférence. L'idée est de voir comment se comporte une transmission où les informations sont reçues exactement comme elles ont été envoyées. Ça nous permettra de mieux comprendre comment fonctionnent les composants clés de la chaîne de transmission..

6.1. Composants impliqués

La simulation s'articule autour des composants suivants :

1. **SourceFixe** : Génère un message binaire fixe composé d'une suite de bits (0 et 1). Ce message sera transmis sans aucune altération par la chaîne de transmission.
2. **TransmetteurParfait** : Ce composant représente un canal de communication parfait, c'est-à-dire qu'il n'introduit ni erreur, ni dégradation du signal au cours de la transmission. Le signal reçu par le transmetteur est donc identique au signal émis.
3. **DestinationFinale** : Reçoit l'information transmise par le canal et la stocke pour l'analyse. Elle permet de vérifier si l'information reçue correspond exactement à l'information émise par la source.

6.2. Étapes de la simulation

1. **Initialisation des composants** : Les trois composants principaux sont instanciés :
 - La source génère une séquence binaire fixe.
 - Le transmetteur parfait est initialisé pour transmettre sans erreurs.
 - La destination est prête à recevoir l'information.

2. Connexion des composants : À cette étape, les composants sont reliés entre eux via la méthode `connecter(DestinationInterface)`. La source est connectée au transmetteur, et le transmetteur est à son tour connecté à la destination.
3. Émission du signal : La méthode `emettre()` est invoquée sur la `SourceFixe`. Celle-ci génère le message binaire et le transmet au `TransmetteurParfait`, qui le relaie ensuite à la `DestinationFinale` sans altération.
4. Réception et vérification : À la fin de la transmission, la `DestinationFinale` stocke l'information reçue. Le message émis par la source est comparé au message reçu pour vérifier qu'aucune erreur n'a été introduite.
5. Calcul du Taux d'Erreur Binaire (TEB) : Le TEB) est calculé pour quantifier les éventuelles erreurs de transmission. Dans le cas d'un canal parfait, le TEB doit être égal à zéro, signifiant que tous les bits ont été correctement transmis sans erreurs.

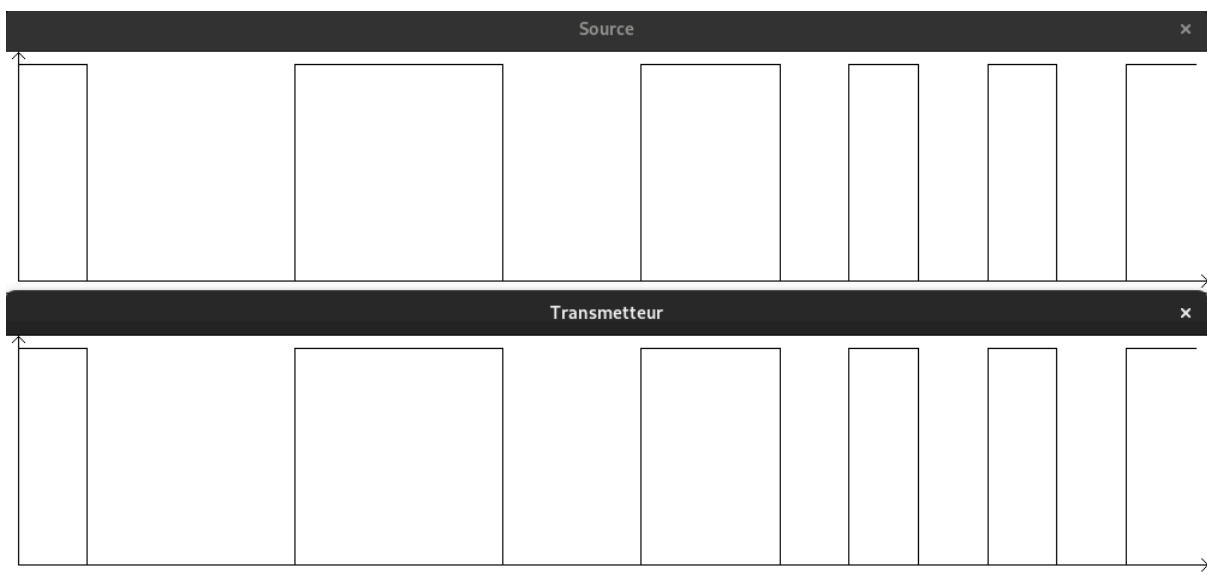
6.3. Résultats attendus

Dans une transmission via un canal parfait, l'information reçue par la destination doit être exactement identique à celle émise par la source. Cela signifie qu'aucune erreur ne doit être observée et que le TEB sera de 0 %. Ce cas constitue une référence pour les simulations futures avec des canaux bruités ou des systèmes de codage, car il permet de valider le bon fonctionnement de la chaîne de transmission dans des conditions idéales.

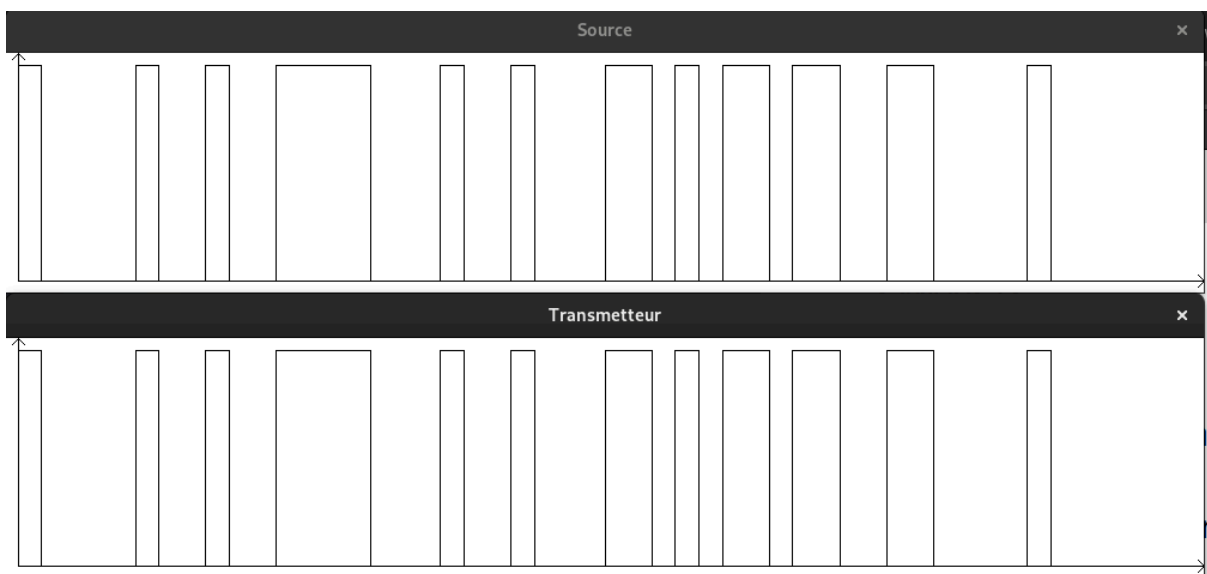
6.3. Résultats obtenus

En exécutant le simulateur avec une source fixe et avec l'affichage des sondes, on retrouve les résultats attendus (même message à la source et transmetteur/destination) correspondant au message passe en paramètres, l'affichage fonctionne avec le paramètre, et on peut utiliser un message aléatoire si indiqué.

```
~/sit213 main > ./simulateur -s -mess "10001110011010101"
java Simulateur -s -mess 10001110011010101 => TEB : 0.0
```



```
~/sit213 main > ./simulateur -s -mess 50 -seed 10
java Simulateur -s -mess 50 -seed 10 => TEB : 0.0
```



Conclusion

À la fin de cette première étape, nous avons :

- Instancier les composants essentiels de la chaîne de transmission (Source, Transmetteur, Destination).
- Connecter les composants de manière logique pour permettre la transmission de l'information.
- Simuler l'émission, la transmission, et la réception de l'information.
- Mis en place le calcul du taux d'erreur binaire (TEB), qui permet d'évaluer la performance de la transmission.
- Effectuer les tests et observer leurs couvertures.

La prochaine étape consistera à introduire les éléments de bruitage et de codage pour améliorer la robustesse de la transmission et à simuler différentes conditions de communication.

16/09/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 2

Introduction.....	2
1. Conception des composants.....	3
1.1 Émetteur.....	3
1.2 Transmetteur Analogique Parfait.....	4
1.3 Récepteur.....	4
2. Connexion des composants.....	5
2.1 Connexion dans la chaîne analogique.....	5
2.2 Rôles des sondes dans la connexion.....	6
2.3 Connexion dynamique dans le simulateur.....	6
4. Diagramme de classe.....	7
5. Tests.....	8
5.1 Objectifs des tests.....	8
5.2 Structure des tests.....	8
5.4 Exécution des tests.....	9
5.5 Résultats des tests.....	9
5.6 Couverture de code avec Emma.....	9
6. Simulation d'une transmission analogique par un canal parfait.....	10
6.1 Objectif de la simulation.....	10
6.2 Composants impliqués.....	10
6.3 Étapes de la simulation.....	11
6.4 Émission de l'information.....	11
6.5 Réception et analyse de l'information.....	11
6.6 Résultats attendus.....	12
6.6 Résultats obtenus.....	12
Conclusion.....	13

Introduction

Ce rapport présente la conception, la simulation et la validation d'un système de transmission numérique dans le cadre du module SIT213. Le projet se concentre sur la modélisation d'une chaîne de transmission complète, composée de plusieurs composants clés : sources d'information, transmetteurs et destinations. L'objectif principal est de simuler la transmission d'un message à travers un canal, en commençant par des scénarios idéaux avec un canal parfait, pour ensuite explorer des scénarios plus complexes tels que la transmission analogique et bruitée.

Pour cette étape, nous introduisons de nouveaux composants : l'émetteur, le récepteur, et le transmetteur analogique parfait. Ces composants permettent de moduler et démoduler des signaux analogiques afin de transmettre un message logique sous forme analogique, tout en maintenant une transmission fidèle.

L'accent est mis sur la modélisation et la simulation d'une chaîne de transmission analogique utilisant des modulations telles que NRZ, NRZT, et RZ. L'implémentation de ces modulations permet d'observer l'impact de la conversion logique-analogique et d'évaluer la transmission dans des conditions proches du monde réel

Cette étape inclut également l'utilisation de sondes pour visualiser les signaux analogiques et logiques tout au long de la chaîne de transmission. La précision de la transmission est quantifiée en termes de taux d'erreur binaire (TEB).

1. Conception des composants

1.1 Émetteur

L'émetteur prend une suite de bits logiques (représentés par des valeurs booléennes) et applique une modulation pour les transformer en un signal analogique.

Fonctionnement détaillé :

- **Paramètres de l'émetteur :**
 - A_{min} : L'amplitude du signal analogique représentant un bit logique '0'.
 - A_{max} : L'amplitude du signal analogique représentant un bit logique '1'.
 - $nbEchantillonsParBit$: Nombre d'échantillons analogiques générés pour chaque bit logique.
 - $typeModulation$: Type de modulation choisi parmi **NRZ** (Non Return to Zero), **NRZT** (Non Return to Zero Transition), et **RZ** (Return to Zero).
- **Modulation NRZ (Non Return to Zero) :**
 - Le signal analogique reste constant pendant toute la durée du bit. Un bit '1' est représenté par une amplitude égale à A_{max} , et un bit '0' par une amplitude égale à A_{min} .
 - Le signal analogique ne retourne pas à zéro entre les bits, ce qui signifie que l'amplitude reste constante tant que les bits successifs ont la même valeur.
- **Modulation NRZT (Non Return to Zero Transition) :**
 - Cette modulation introduit une transition progressive entre les bits '0' et '1'. Pour un changement de bit (de '0' à '1' ou de '1' à '0'), la transition entre A_{min} et A_{max} se fait progressivement sur une fraction du bit (le premier et le dernier tiers du bit).
 - Si deux bits consécutifs ont la même valeur, l'amplitude reste stable sans transition.
- **Modulation RZ (Return to Zero) :**
 - Le signal est à zéro au premier et dernier tiers de chaque bit. Ainsi, le second tiers d'un bit '1' est à A_{max} et le premier et troisième tiers sont à zéro. Pour un bit '0', l'amplitude reste à zéro tout au long de la durée du bit.

Processus de conversion : L'émetteur prend les bits logiques et les transforme en un signal analogique en fonction du type de modulation choisi. Pour chaque bit, il génère une série d'échantillons analogiques (dont le nombre est déterminé par $nbEchantillonsParBit$), en respectant les règles propres à la modulation sélectionnée.

1.2 Transmetteur Analogique Parfait

Le transmetteur analogique parfait relaie le signal analogique de l'émetteur vers le récepteur sans aucune modification ni altération.

1.3 Récepteur

Le récepteur a pour rôle inverse de celui de l'émetteur. Il reçoit un signal analogique, l'analyse et le convertit en une information logique (booléenne).

Fonctionnement détaillé :

- **Paramètres du récepteur :**
 - A_{min} : L'amplitude représentant un bit '0' dans le signal analogique.
 - A_{max} : L'amplitude représentant un bit '1' dans le signal analogique.
 - $nbEchantillonsParBit$: Nombre d'échantillons reçus pour chaque bit.
 - $typeModulation$: Le type de modulation utilisé pour démoduler le signal, identique à celui appliqué par l'émetteur (**NRZ**, **NRZT**, ou **RZ**).
- **Démodulation NRZ :**
 - Le récepteur collecte les échantillons du signal analogique sur la durée correspondant à un bit. Pour chaque bit, il calcule la moyenne des échantillons reçus.
 - Si la moyenne est supérieure ou égale à la moitié de la différence entre A_{min} et A_{max} , le bit est considéré comme un '1'. Sinon, il est interprété comme un '0'.
- **Démodulation NRZT :**
 - Dans le cas de la modulation NRZT, seuls les échantillons du deuxième tiers de chaque bit sont utilisés pour la démodulation. Cette méthode permet de s'affranchir des transitions progressives introduites par la modulation NRZT et de se concentrer sur la partie stable du signal pour une meilleure précision de la démodulation.
- **Démodulation RZ :**
 - Pour cette modulation, le récepteur n'utilise que la deuxième partie du signal (là où l'amplitude est supposée être maximale pour un bit '1') pour déterminer la valeur du bit. Si cette portion du signal est à A_{max} , le récepteur détermine qu'il s'agit d'un bit '1', sinon, il considère que c'est un bit '0'.

Processus de démodulation : Le récepteur reçoit une série d'échantillons analogiques et, selon le type de modulation, il calcule la valeur logique correspondante. Il génère ainsi une information booléenne qui est ensuite envoyée à la destination finale.

2. Connexion des composants

La chaîne de transmission dans le simulateur est composée de plusieurs composants interconnectés : une source de données logiques, un émetteur, un transmetteur parfait, un récepteur et une destination finale.

2.1 Connexion dans la chaîne analogique

Dans le cadre d'une transmission **analogique**, la chaîne de composants inclut des étapes supplémentaires pour la modulation et la démodulation des informations. La connexion des composants se fait comme suit :

1. **Source → Émetteur :**
 - La source génère une information logique, qui est ensuite transmise à l'émetteur.
 - L'émetteur convertit cette information logique en un signal analogique en fonction du type de modulation sélectionné (NRZ, NRZT, ou RZ).
2. **Émetteur → Transmetteur analogique parfait :**
 - Une fois le signal analogique généré par l'émetteur, celui-ci est transmis au **transmetteur analogique parfait**. Ce composant se comporte comme un relais parfait pour les signaux analogiques, garantissant qu'aucune modification ou distorsion n'est introduite pendant la transmission.
3. **Transmetteur analogique parfait → Récepteur :**
 - Le transmetteur analogique parfait envoie le signal analogique au récepteur. Ce dernier reçoit le signal et le démodule en fonction du type de modulation utilisé à l'émission.
 - Le récepteur récupère ainsi l'information logique (booléenne) correspondant au signal analogique reçu.
4. **Récepteur → Destination :**
 - Le récepteur connecte enfin la destination finale en transmettant l'information logique démodulée. Cette information correspond théoriquement à celle qui a été initialement générée par la source, à condition qu'aucune erreur ne soit survenue dans la transmission.

2.2 Rôles des sondes dans la connexion

Lorsqu'elles sont activées, elles permettent de visualiser :

- L'information logique émise par la source (sonde logique).
- Le signal analogique généré par l'émetteur et transmis par le transmetteur analogique parfait (sonde analogique).
- L'information démodulée et reçue par la destination (sonde logique).

Ces sondes permettent de vérifier visuellement que la transmission se déroule correctement à chaque étape.

2.3 Connexion dynamique dans le simulateur

La connexion des composants dans le simulateur est effectuée dynamiquement en fonction des arguments fournis par l'utilisateur lors de l'exécution du programme. Par exemple :

- Si une modulation analogique est spécifiée (-form NRZ, -form NRZT, ou -form RZ), le simulateur configure automatiquement une chaîne analogique avec émetteur et récepteur.
- Si aucun argument de modulation n'est fourni, une chaîne purement logique est configurée.
- Les sondes ne sont ajoutées que si l'option -s est activée.

5. Tests

5.1 Objectifs des tests

L'objectif principal des tests est de valider le bon fonctionnement de chaque composant du simulateur, ainsi que l'intégration globale des différents modules dans la chaîne de transmission. Les tests sont conçus pour vérifier que :

1. Chaque composant individuellement (source, transmetteur, émetteur, récepteur, etc.) fonctionne conformément à ses spécifications.
2. Les interactions entre les composants, lorsque connectés ensemble, produisent les résultats attendus, notamment en termes de transmission correcte des informations.
3. Les calculs, notamment le Taux d'Erreur Binaire, sont corrects et reflètent fidèlement la qualité de la transmission.

5.2 Structure des tests

Les tests ont été organisés en une suite de tests JUnit, permettant d'exécuter tous les tests de manière groupée. Voici les principales classes de test utilisées :

- **TransmetteurParfait** : Le transmetteur parfait doit relayer l'information sans la modifier. Les tests vérifient que l'information reçue par le transmetteur est identique à celle émise.
- **Émetteur** : Les tests de l'émetteur couvrent la conversion de l'information logique en signal analogique selon les trois types de modulation (NRZ, NRZT, RZ). Ils vérifient que l'amplitude des signaux produits correspond bien aux valeurs définies pour les bits 0 et 1.
- **Récepteur** : Le récepteur effectue la conversion inverse, du signal analogique à l'information logique. Les tests vérifient que le signal démodulé correspond à l'information logique d'origine, en prenant en compte le type de modulation utilisé.
- **Simulateur** : Chaîne de transmission analogique : Validation du bon fonctionnement de l'ensemble émetteur, transmetteur analogique parfait et récepteur. Ces tests incluent la vérification de la modulation et démodulation correcte de l'information.
- **Taux d'Erreur Binaire est correctement calculé**:
 - Transmission sans erreur : Lorsqu'aucune erreur n'est présente, le TEB doit être de 0.0.

- Transmission avec erreurs : Nous modifions l'information reçue à la destination pour introduire des erreurs et vérifions que le TEB calculé reflète bien le pourcentage de bits erronés.

5.4 Exécution des tests













Les tests ont été exécutés via la classe AllTests, en utilisant le script runTests qui regroupe l'ensemble des tests. Cette classe permet d'automatiser l'exécution des tests et de vérifier en une seule étape que tous les composants fonctionnent comme attendu.

5.5 Résultats des tests

Tous les tests ont été passés avec succès, confirmant la fiabilité des composants individuels et du système global de transmission.

5.6 Couverture de code avec Emma

L'outil Emma a été utilisé pour mesurer la couverture des tests sur l'ensemble du projet. Voici les résultats de la couverture de code :

Element	Coverage	Covered Instructions	Missed Instructions
▼ src	 84,7 %	2 308	416
▶ visualisations	 64,2 %	607	338
▶ simulateur	 89,9 %	487	55
▼ transmetteurs	 97,9 %	874	19
▶ Recepteur.java	 96,3 %	334	13
▶ Emetteur.java	 98,6 %	422	6
▶ Transmetteur.java	 100,0 %	35	0
▶ TransmetteurAnalogiqueParfait.java	 100,0 %	47	0
▶ TransmetteurParfait.java	 100,0 %	36	0
▶ information	 96,6 %	113	4
▶ destinations	 100,0 %	43	0
▶ sources	 100,0 %	184	0

La couverture légèrement plus faible de certains éléments indique que certains cas spécifiques peuvent être ajoutés dans les futures étapes pour renforcer la robustesse des tests.

6. Simulation d'une transmission analogique par un canal parfait

6.1 Objectif de la simulation

L'objectif principal de cette simulation est de valider le bon fonctionnement de la chaîne de transmission analogique à travers un canal parfait. Le canal parfait représente un modèle théorique dans lequel l'information émise par la source est transmise et reçue sans aucune altération ni distorsion. Le Taux d'Erreur Binaire doit être nul, c'est-à-dire que le message reçu par la destination est strictement identique au message émis par la source.

Cette simulation constitue une base de référence qui servira à comparer les performances du système dans des scénarios futurs, où des perturbations, comme le bruit, seront introduites.

6.2 Composants impliqués

Les composants impliqués dans cette simulation sont les suivants :

- **SourceFixe** : Elle génère une séquence de bits logiques prédéfinis. Cette source permet de vérifier la cohérence du système en émettant un message fixe et immuable.
- **Emetteur** : Ce composant prend l'information logique fournie par la SourceFixe et la convertit en un signal analogique en utilisant une modulation spécifique (par exemple, NRZ, NRZT, RZ).
- **TransmetteurAnalogiqueParfait** : Ce composant représente le canal parfait. Il reçoit le signal analogique émis par l'émetteur et le transmet sans aucune modification à la destination.
- **Recepteur** : Ce composant démodule le signal analogique reçu pour le reconvertir en une information logique. Il utilise la même technique de modulation que l'émetteur pour décoder correctement le signal.
- **DestinationFinale** : La destination reçoit l'information logique démodulée et la compare à l'information initialement émise par la source afin de vérifier l'intégrité de la transmission.

6.3 Étapes de la simulation

Initialisation des composants :

1. Une **SourceFixe** est instanciée pour générer un message logique prédéfini.
2. Un **Emetteur** est créé pour moduler ce message en un signal analogique.
3. Un **TransmetteurAnalogiqueParfait** relaie le signal sans le modifier.
4. Un **Recepteur** est mis en place pour démoduler le signal et récupérer le message logique initial.
5. Enfin, une **DestinationFinale** est configurée pour recevoir le message et en analyser l'intégrité.

Connexion des composants :

- La **SourceFixe** est connectée à l'**Emetteur**.
- L'**Emetteur** est connecté au **TransmetteurAnalogiqueParfait**.
- Le **TransmetteurAnalogiqueParfait** est relié au **Recepteur**.
- Le **Recepteur** est connecté à la **DestinationFinale**.

De plus, des sondes analogiques et logiques sont connectées pour visualiser les signaux.

Ainsi, la chaîne de transmission est complète :

SourceFixe → **Emetteur** → **TransmetteurAnalogiqueParfait** → **Recepteur** → **DestinationFinale**.

6.4 Émission de l'information

Une fois les composants connectés, la méthode `emettre()` de la **SourceFixe** est appelée. Cela envoie le message logique à l'**Emetteur**, qui le module en un signal analogique. Le signal est ensuite transmis sans altération par le **TransmetteurAnalogiqueParfait** jusqu'au **Recepteur**.

6.5 Réception et analyse de l'information

Le **Recepteur** reçoit le signal analogique, le démodule, et le convertit à nouveau en information logique. Cette information est ensuite transmise à la **DestinationFinale**, qui la compare avec le message initial émis par la **SourceFixe**. Le TEB est ensuite calculé en fonction du nombre de différences entre les deux messages.

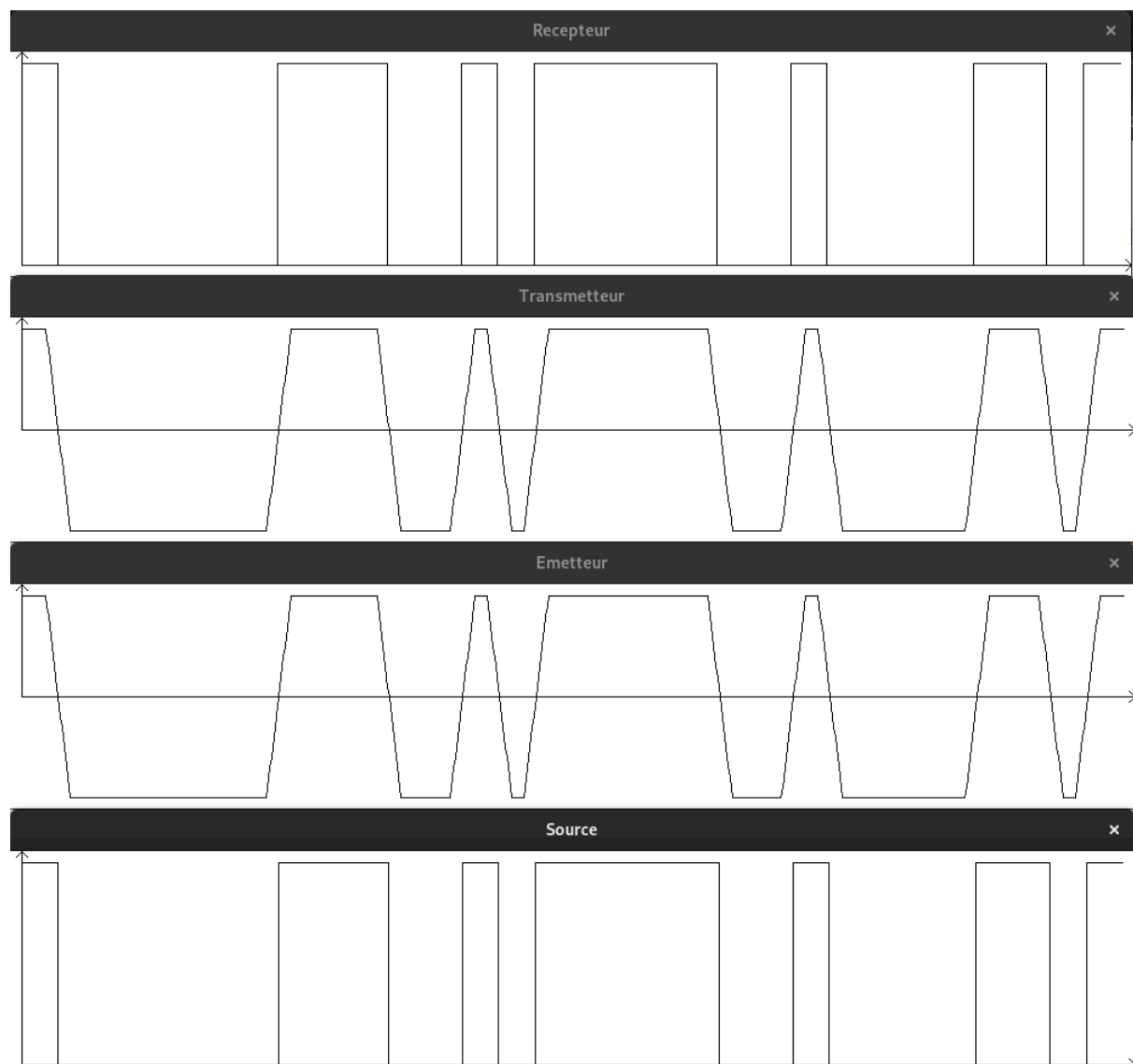
6.6 Résultats attendus

Dans cette simulation, étant donné que le canal est parfait, le message reçu à la destination doit être strictement identique à celui émis par la source. Par conséquent, le Taux d'Erreur Binaire (TEB) doit être de 0 %, indiquant qu'il n'y a aucune erreur dans la transmission.

6.6 Résultats obtenus

En exécutant le simulateur avec une source aléatoire avec seed et avec l'affichage des sondes, on retrouve les résultats attendus (même message à la source et transmetteur/destination) correspondant au message passe en paramètres, l'affichage fonctionne avec le paramètre, et on peut utiliser un message aléatoire si indiqué. Avec les différentes modulations et autres paramètres. Par exemple avec :

```
~/sit213 main > ./simulateur -s -mess 30 -form NRZT -seed 1 -nbEch 30 -ampl -1.0 1.0
java Simulateur -s -mess 30 -form NRZT -seed 1 -nbEch 30 -ampl -1.0 1.0 => TEB : 0.0
```



Conclusion

L'étape 2 du projet a permis de valider l'implémentation des composants de transmission analogique via un canal parfait. À travers la simulation, nous avons pu démontrer que la chaîne de transmission fonctionnait correctement, avec un Taux d'Erreur Binaire de 0 %, confirmant l'absence d'erreurs de transmission dans des conditions parfaites. L'intégration des modules, tels que l'Emetteur, le TransmetteurAnalogiqueParfait et le Recepteur, a été réalisée avec succès, et leur fonctionnement a été testé de manière approfondie. Cette étape constitue une base solide pour l'introduction de nouvelles perturbations dans les simulations à venir, permettant ainsi une évaluation des performances du système dans des environnements plus réalistes.

Les résultats obtenus démontrent la fiabilité de l'architecture mise en place et ouvrent la voie à des études plus approfondies pour simuler des canaux non parfaits avec du bruit et d'autres interférences.

24/09/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 3

Introduction.....	2
1. Conception des composants.....	3
1.1 TransmetteurAnalogiqueBruite.....	3
2. Connexion des composants.....	4
2.1 Connexion dans la chaîne de transmission analogique avec bruit.....	4
2.2 Rôles des sondes dans la connexion.....	5
3. Tests.....	6
3.1 Objectifs des tests.....	6
3.2 Structure des tests.....	6
3.3 Exécution des tests.....	7
3.4 Résultats des tests.....	7
3.5 Couverture de code avec Emma.....	7
4. Simulation d'une transmission avec un canal bruité (gaussien).....	8
4.1 Évolution du TEB en fonction du SNR.....	8
4.3 Analyse des résultats.....	12
Conclusion.....	14

Introduction

L'étape 3 du projet introduit un scénario de transmission plus réaliste où le canal de communication n'est plus parfait, mais bruité. Dans les systèmes de communication, les signaux sont souvent affectés par des interférences aléatoires, comme le bruit blanc additif gaussien, qui dégrade la qualité de la transmission et augmente le risque d'erreurs à la réception. L'objectif de cette étape est de modéliser un canal bruité en ajoutant ce type de bruit au signal transmis, afin d'étudier l'impact sur la transmission, en particulier en mesurant le Taux d'Erreur Binaire.

Le bruit blanc gaussien est une perturbation qui suit une distribution normale centrée et peut affecter chaque échantillon du signal analogique pendant la transmission. En fonction du rapport signal sur bruit, le bruit ajouté au signal peut rendre la détection des bits plus ou moins difficile pour le récepteur, entraînant des erreurs dans la séquence reçue. Le TEB devient un indicateur clé de la qualité de la transmission, mesurant la proportion de bits erronés par rapport au nombre total de bits transmis.

L'organisation du rapport vise à détailler chaque étape du projet, en expliquant la conception des composants, les tests réalisés, ainsi que les résultats obtenus lors des simulations.

1. Conception des composants

1.1 TransmetteurAnalogiqueBruite

Le TransmetteurAnalogiqueBruite est une extension du Transmetteur qui modélise un environnement de transmission bruité. Ce bruit est aléatoire et suit une distribution gaussienne, ce qui permet de simuler les effets des interférences présentes dans un canal de communication réel.

Fonctionnement du TransmetteurAnalogiqueBruite :

1. Réception du signal analogique :

- Le transmetteur reçoit une séquence analogique depuis le récepteur.

2. Ajout du bruit gaussien :

- Le TransmetteurAnalogiqueBruite y ajoute un bruit gaussien. Le bruit est généré aléatoirement en suivant une distribution normale centrée sur 0, avec une variance/écart-type déterminée par le SNR ($\sigma = \text{Math.sqrt}(\text{puissanceBruit})$) \Rightarrow converti à partir du E_b/N_0 en paramètre, calculé par deux méthodes `"calculerPuissanceSignal"` et `"calculerPuissanceBruit"`
- Chaque échantillon du signal analogique se voit ajouter un bruit indépendant pour simuler les interférences

3. Transmission vers le Récepteur :

- Le signal bruité est ensuite transmis au Récepteur, qui le reçoit et l'analyse. L'ajout de bruit peut entraîner des erreurs de réception, notamment une modification de la forme du signal, ce qui augmentera le TEB à la réception.

2. Connexion des composants

La chaîne de transmission dans le simulateur est composée de plusieurs composants interconnectés qui permettent la génération, la modulation, la transmission et la réception d'informations. La chaîne de transmission bruitée est activée dès que le paramètre -snrpb est ajouté (!=0).

2.1 Connexion dans la chaîne de transmission analogique avec bruit

Dans le cadre d'une transmission analogique avec bruit, la chaîne de composants inclut des étapes supplémentaires pour la modulation, l'ajout de bruit, et la démodulation. La connexion des composants se fait de la manière suivante :

- **Source → Émetteur :**

La source génère une information logique, qui peut être une séquence binaire fixe ou aléatoire, selon la configuration du simulateur. L'information logique générée est transmise à l'émetteur

- **Émetteur → Transmetteur analogique bruité :**

L'émetteur reçoit l'information logique et la convertit en un signal analogique en fonction de la modulation sélectionnée (NRZ, NRZT ou RZ) et des autres paramètres (Amin, Amax, nbEchantillonsParBit). Une fois le signal analogique généré, celui-ci est transmis au TransmetteurAnalogiqueBruite, qui modélise un canal bruité en ajoutant un bruit blanc gaussien au signal analogique en fonction du SNR par bit choisi. Ce transmetteur simule les effets du bruit sur la transmission en ajoutant des perturbations.

- **Transmetteur analogique bruité → Récepteur :**

Le signal analogique bruité est ensuite transmis au récepteur. Ce dernier reçoit le signal dégradé par le bruit et effectue la démodulation en fonction de la modulation sélectionnée à l'émission (NRZ, NRZT, ou RZ). Le récepteur convertit le signal analogique en une séquence logique (booléenne) correspondant aux informations émises

- **Récepteur → Destination finale :**

Le récepteur transmet enfin l'information logique récupérée à la DestinationFinale, qui reçoit la séquence de bits démodulée et la compare avec l'information

initialement émise par la source. La destination finale utilise ces données pour calculer le Taux d'Erreur Binaire, permettant d'évaluer la qualité de la transmission en prenant en compte les erreurs introduites par le bruit

2.2 Rôles des sondes dans la connexion

Les sondes peuvent être activées dans le simulateur pour observer :

- **Sonde logique à la source :**
Permet de visualiser l'information binaire émise par la source, avant toute modulation
- **Sonde analogique après l'émetteur :**
Permet de capturer et visualiser le signal analogique généré par l'émetteur avant l'ajout de bruit par le transmetteur. Cela permet de vérifier que la modulation est correctement effectuée
- **Sonde analogique après le transmetteur analogique bruité :**
Permet de visualiser le signal analogique une fois le bruit gaussien ajouté, afin d'observer l'impact du bruit sur la forme du signal
- **Sonde logique après le récepteur :**
Permet de visualiser l'information logique démodulée par le récepteur et de la comparer avec le signal émis par la source, ce qui facilite l'analyse des erreurs éventuelles

3. Tests

3.1 Objectifs des tests

L'objectif principal des tests est de valider le bon fonctionnement de chaque composant du simulateur, ainsi que l'intégration globale des différents modules dans la chaîne de transmission. Les tests ont été conçus pour vérifier que :

- **Chaque composant** (source, transmetteur, émetteur, récepteur, destination) fonctionne conformément à ses spécifications.
- **Les interactions entre les composants** : lorsque connectés, ils produisent les résultats attendus, notamment en termes de transmission correcte des informations, de modulation et de démodulation.
- **Le calcul du TEB** est correct et reflète fidèlement la qualité de la transmission.
- **La robustesse des composants** face à des transmissions bruitées, en analysant les erreurs introduites par le bruit gaussien.

3.2 Structure des tests

Les tests ont été organisés en une suite de tests JUnit, permettant d'exécuter tous les tests de manière groupée. Voici les principales classes de test utilisées :

- **TransmetteurAnalogiqueBruite** : Le TransmetteurAnalogiqueBruite couvre la conversion de l'information logique en signal analogique avec les différentes formes de modulation (NRZ, NRZT, RZ). Il vérifie que la conversion est correcte, que les signaux analogiques générés ont la bonne amplitude pour les bits 0 et 1, et que le bruit gaussien est correctement appliqué pour simuler un canal bruité.
- **Simulateur** : La chaîne de transmission complète est testée, de l'émission à la réception, pour valider l'interaction entre les composants (source, transmetteur, récepteur, destination). Les tests incluent la vérification que l'ensemble fonctionne correctement, même dans des environnements bruités, en calculant le TEB pour des transmissions avec et sans erreurs.

3.3 Exécution des tests

Les tests ont été exécutés via la classe AllTests, qui regroupe l'ensemble des tests unitaires et d'intégration. Cette classe permet d'automatiser l'exécution des tests et de vérifier, en une seule étape, que tous les composants fonctionnent comme attendu, aussi bien individuellement qu'en interaction les uns avec les autres.

Les tests ont été lancés à l'aide d'un script runTests qui exécute tous les tests dans un environnement contrôlé, garantissant ainsi que le code est testé systématiquement avant chaque nouvelle livraison ou itération du projet. Cela prend plusieurs dizaines de secondes.

3.4 Résultats des tests

Pour les simulations avec bruit gaussien, le TEB varie en fonction du rapport SNR par bit, comme attendu. Les tests montrent que le TEB augmente proportionnellement à la dégradation du rapport signal-bruit, reflétant bien l'impact du bruit sur la transmission. Tous les tests sont validés.

3.5 Couverture de code avec Emma

L'outil Emma a été utilisé pour mesurer la couverture des tests sur l'ensemble du projet. Voici les résultats de la couverture de code :

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ sit213	87,2 %	5453	802	6 255
▼ src	83,2 %	2 866	579	3 445
▼ transmetteurs	94,0 %	1 181	75	1 256
▶ Emetteur.java	95,4 %	436	21	457
▶ Recepteur.java	96,2 %	333	13	346
▶ TransmetteurAnalogiqueBruite.java	87,8 %	294	41	335
▶ TransmetteurAnalogiqueParfait.java	100,0 %	47	0	47
▶ TransmetteurParfait.java	100,0 %	36	0	36
▶ Transmetteur.java	100,0 %	35	0	35
▶ visualisations	65,4 %	618	327	945
▶ simulateur	81,0 %	737	173	910
▶ sources	100,0 %	174	0	174
▶ information	96,6 %	113	4	117
▶ destinations	100,0 %	43	0	43
▶ tests	92,1 %	2 587	223	2 810

Ces résultats montrent que la majorité du code a été testé de manière quasi-exhaustive. La couverture légèrement plus faible de certains éléments, indique que des cas spécifiques peuvent être ajoutés lors des futures itérations pour renforcer la robustesse des tests.

Globalement, la couverture est très satisfaisante, assurant que le projet a été largement vérifié pour détecter d'éventuelles anomalies ou régressions. Certaines classes ou méthodes n'ont pas besoin d'être testées (par exemple, la création des fichiers), ceci influe sur la couverture des tests.

4. Simulation d'une transmission avec un canal bruité (gaussien)

Dans cette section, nous explorons la transmission d'un signal analogique à travers un canal bruité modélisé par un bruit blanc gaussien. L'objectif est de mesurer l'impact du bruit sur la qualité de la transmission en analysant l'évolution du taux d'erreur binaire en fonction du rapport signal sur bruit (SNR). Les performances des différentes modulations (NRZ, NRZT, RZ) sont comparées afin de déterminer leur résistance face aux perturbations. De plus, un histogramme du bruit gaussien est généré pour vérifier la distribution du bruit ajouté au signal.

4.1 Évolution du TEB en fonction du SNR

Pour mesurer les performances du système, nous utilisons le simulateur avec le paramètre **-snrpb <float>** qui permet de spécifier le rapport signal sur bruit, calculé à partir de E_b/N_0 en paramètre. En effectuant plusieurs simulations, nous obtenons les résultats suivants.

Pour visualiser les performances des différentes modulations en fonction du bruit, nous allons faire des graphes affichant le TEB en fonction du SNR. Un meilleur TEB signifie des meilleures performances.

La classe `SimulateurTEB` permet de générer 3 fichiers CSV contenant les différents TEB pour différents SNR pour chaque type de modulation, on définit le SNR min et max ainsi que le pas, et le nombre de simulations par SNR (afin de générer une moyenne). Ensuite des boucles font les simulations pour chaque valeur de SNR.

Ensuite, nous utilisons python pour générer les courbes, elles sont lissées pour une meilleure lisibilité, en faisant une moyenne mobile sur 5 points (`np.convolve(y, np.ones(window_size) / window_size, mode='same')`) (Les librairies Matplotlib, numpy et pandas sont nécessaire).

L'axe des ordonnées indique le Taux d'Erreur Binaire de façon décroissante, un TEB de 0.0 indique qu'aucune erreur a eu lieu, donc un TEB plus faible (haut sur la courbe) indique de meilleures performances. \

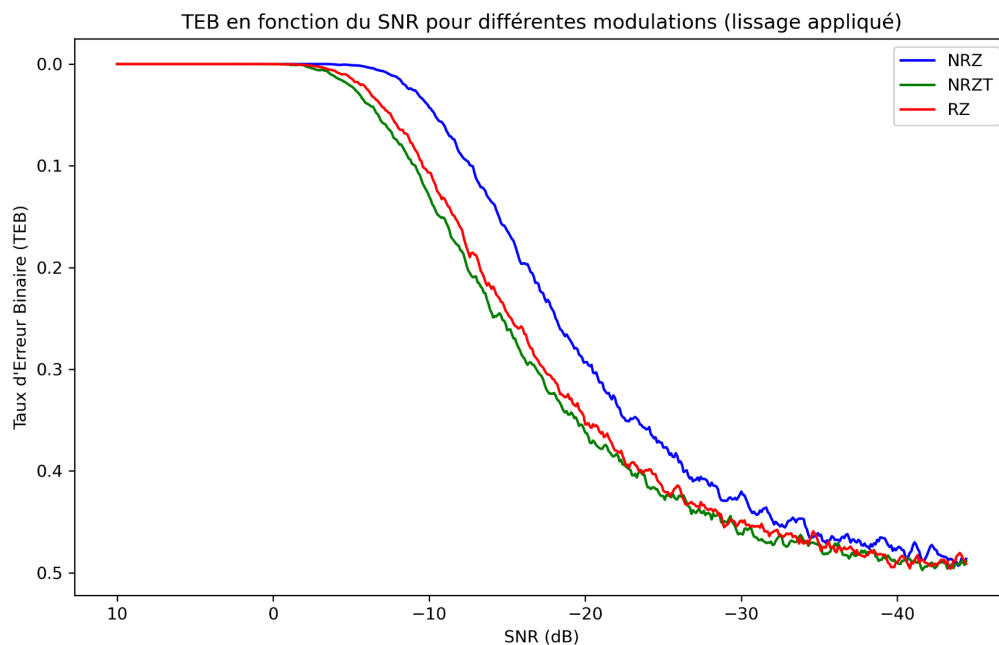
Ensuite, l'axe des abscisses indique le SNR en dB de façon décroissante, un SNR élevé (à gauche de la courbe) indique peu de bruit.

On remarque que jusqu'à un SNR de 0 dB, le TEB est stable et il n'y a aucune erreur, pour tous les types de modulations.

Ensuite, les performances diffèrent en fonction de la modulation. Le NRZ est le plus résistant au bruit puisque c'est le dernier à décroître, puis RZ et NRZT sont proches.

On atteint un plateau après -30 dB avec un TEB de 0.5, soit une chance sur deux. Le signal n'est plus du tout ce qu'il était et aucun bit n'est déterminé. On a donc une chance sur deux d'avoir le bon.

En faisant une moyenne (sur 10 fois, avec messages de 300 bits), on obtient cette courbe (axes inversés) :



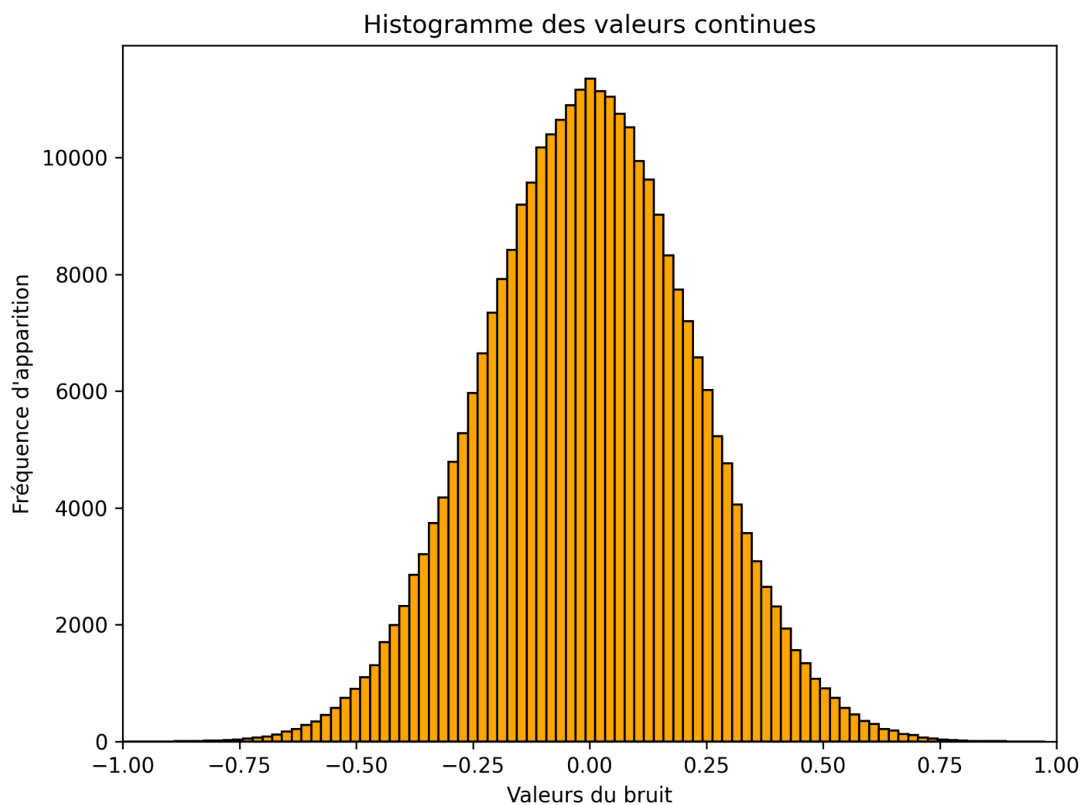
La méthode de détection peut certainement être améliorée pour avoir de meilleures performances pour tous les types de modulations.

Une autre méthode aurait pu être d'utiliser une échelle logarithmique afin de mieux visualiser et analyser les résultats, sur des zones plus précises. Les simulations n'ont pas données de résultats plus intéressants.

4.2 Histogramme du bruit

Pour générer un histogramme du bruit ajouté, nous sauvegardons le bruit généré et trié dans un fichier. Pour cela, il faut que le paramètre `genererFichierBruit` soit true. Puis, nous utilisons un script python pour générer l'histogramme (La librairie Matplotlib est nécessaire).

Voici le résultat pour un signal de 10 000 bits, avec 30 échantillons par bits et un SNR (sur tout le signal) de 10 dB :



L'histogramme montre une distribution gaussienne centrée sur zéro, confirmant la nature du bruit blanc gaussien. Cela est cohérent avec la théorie du bruit gaussien, où les valeurs sont distribuées symétriquement autour de zéro, avec une fréquence plus élevée pour les valeurs proches de la moyenne et décroissante pour les valeurs extrêmes.

L'écart type dépend bien du niveau du SNR demandé. L'écart type est la racine carrée de la puissance. Cela veut dire qu'un SNR de 10 indique une faible puissance du bruit. Les valeurs sont concentrées en -0,25 et 0,25, ce qui veut dire que le signal change très peu. Dans notre cas, Amax et Amin sont de 0 à 1. Donc avec une faible puissance de bruit.

Le fait que l'écart type et la moyenne correspondent aux attentes théoriques confirme la validité de notre modèle de bruit.

4.3 Analyse des résultats

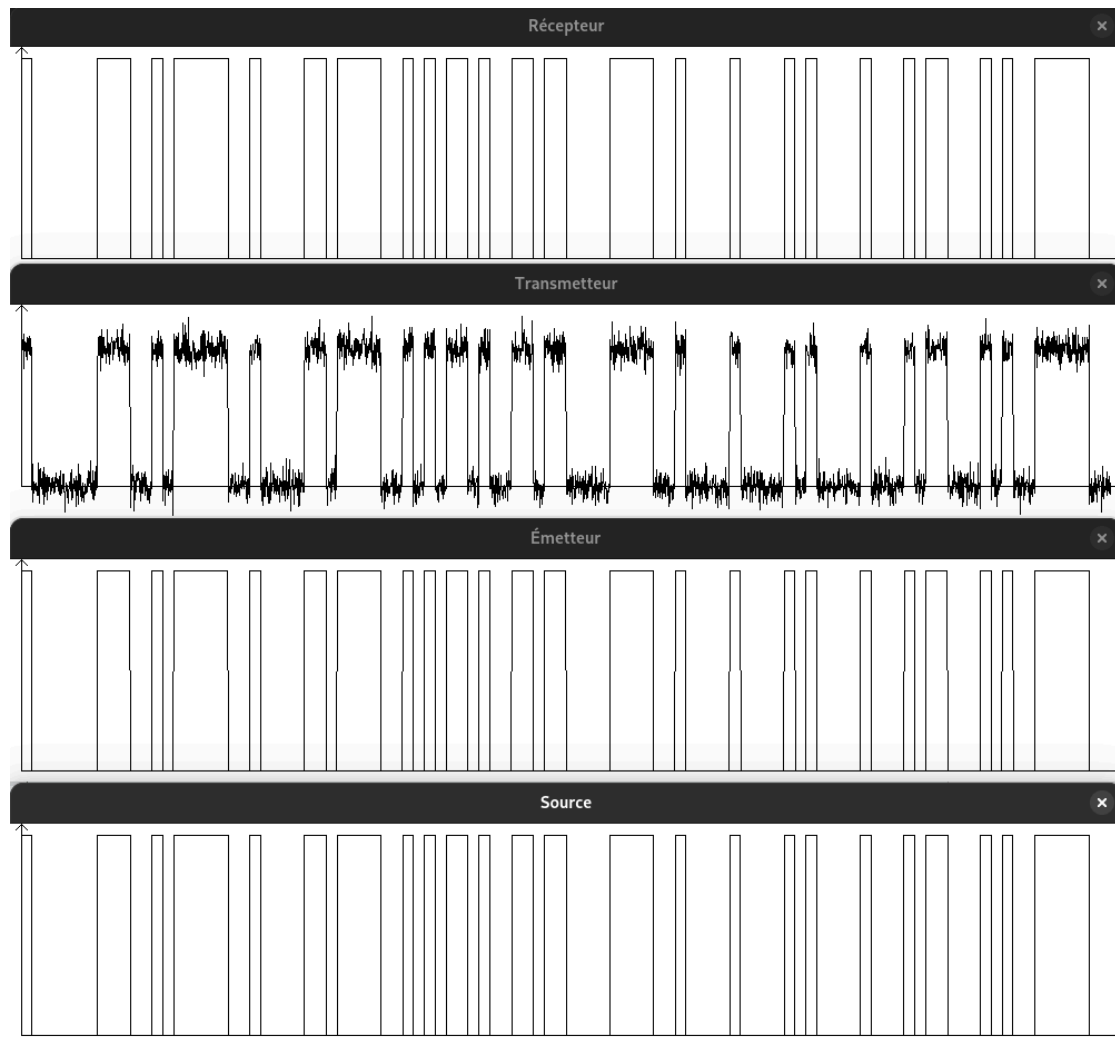
Afin de valider le fonctionnement du Simulateur avec le bruit, nous avons ajouté des mesures dans le but de vérifier les résultats avec ceux donnés ("Exemples de signaux bruités"). Il faut mettre `afficherInformations = true`.

Nos simulations ont globalement les mêmes résultats que celles fournies, ici le paramètre `snrpb` était configuré pour le SNR global et pas E_b/N_0 comme actuellement, les simulations restent les mêmes :

Pour un SNR de 20 dB, nous retrouvons les valeurs fournies

```
- Nombre de bits de la séquence : 100
- Nombre d'échantillons par bit : 30
1-> Puissance MOYENNE de la séquence de bits : 0.51
2-> Valeur de sigma (écart-type du bruit) : 0.0714
3-> Puissance moyenne du bruit : 0.0051
4-> Rapport signal-sur-bruit (S/N, en dB) : 19.9879
5-> Rapport Eb/N0 (en dB) : 31.7489
```

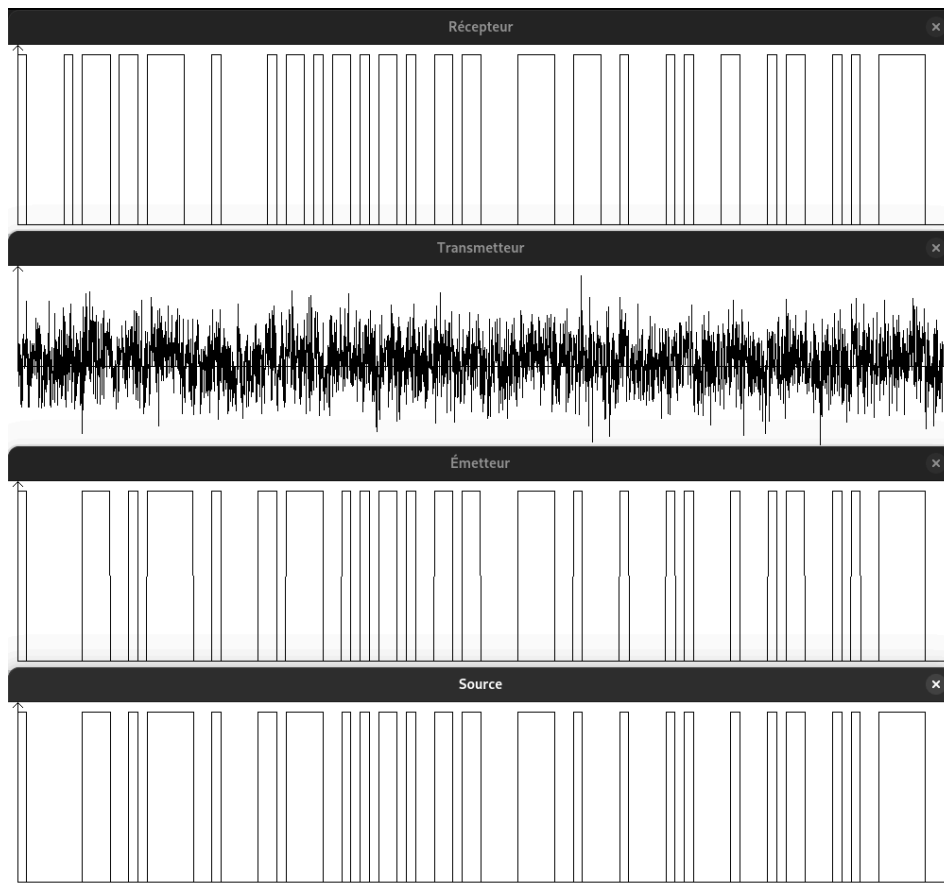
```
~/sit213 main !1 > ./simulateur -s -mess 100 -form NRZ -seed 1 -nbEch 30 -ampl 0.0 1.0 -snrpb 20
- Nombre de bits de la séquence : 100
- Nombre d'échantillons par bit : 30
1-> Puissance MOYENNE de la séquence de bits : 0.45
2-> Valeur de sigma (écart-type du bruit) : 0.0667501392915691
3-> Puissance moyenne du bruit : 0.004455581095443877
4-> Rapport signal-sur-bruit (S/N, en dB) : 20.043081611628907
5-> Rapport Eb/N0 (en dB) : 31.80399420218572
java Simulateur -s -mess 100 -form NRZ -seed 1 -nbEch 30 -ampl 0.0 1.0 -snrpb 20 => TEB : 0.0
```



Pour -10 dB, les résultats sont toujours similaires :

```
- Nombre de bits de la séquence : 100
- Nombre d'échantillons par bit : 30
1-> Puissance MOYENNE de la séquence de bits : 0.45999999999999996
2-> Valeur de sigma (écart-type du bruit) : 2.1448
3-> Puissance moyenne du bruit : 4.6261
4-> Rapport signal-sur-bruit (S/N, en dB) : -10.0245
5-> Rapport Eb/N0 (en dB) : 1.7364
```

```
~/sit213 main :1 > ./simulateur -s -mess 100 -form NRZ -seed 1 -nbEch 30 -ampl 0.0 1.0 -snrpb -10
- Nombre de bits de la séquence : 100
- Nombre d'échantillons par bit : 30
1-> Puissance MOYENNE de la séquence de bits : 0.45
2-> Valeur de sigma (écart-type du bruit) : 2.1428632359207036
3-> Puissance moyenne du bruit : 4.591862847860549
4-> Rapport signal-sur-bruit (S/N, en dB) : -10.087763940756174
5-> Rapport Eb/N0 (en dB) : 1.6731486498006394
java Simulateur -s -mess 100 -form NRZ -seed 1 -nbEch 30 -ampl 0.0 1.0 -snrpb -10 => TEB : 0.09
```



Les résultats obtenus lors de la simulation sont en accord avec les théories de transmission sur un canal bruité :

- L'analyse du TEB en fonction du SNR montre que les performances des différentes modulations se détériorent progressivement à mesure que le bruit augmente
- L'histogramme du bruit confirme la nature gaussienne du bruit généré, assurant ainsi la validité des conditions de transmission simulées
- Les valeurs mesurées sont les mêmes que théoriques et fournies.

Conclusion

L'étape 3 du projet a permis de valider l'implémentation des composants de transmission analogique via un canal bruité. À travers cette simulation, nous avons démontré que la chaîne de transmission réagit comme prévu face aux perturbations, notamment avec l'ajout d'un bruit blanc additif gaussien. Les tests ont montré que le Taux d'Erreur Binaire varie en fonction du rapport signal sur bruit, confirmant que le bruit affecte la qualité de la transmission comme attendu.

L'intégration du module TransmetteurAnalogiqueBruite a été réalisée avec succès et son fonctionnement a été testé de manière approfondie dans des environnements bruités. Cette étape constitue une avancée significative dans la simulation de scénarios plus réalistes, où le bruit et les interférences influencent la transmission des données. Les résultats obtenus démontrent la robustesse du système face aux perturbations, tout en ouvrant la voie à l'exploration d'autres types de bruit ou à l'introduction de mécanismes de correction d'erreurs pour améliorer encore les performances du système dans des environnements non parfaits.

04/10/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 4

Introduction.....	2
1. Conception des composants.....	3
1.1 TransmetteurAnalogiqueMultiTrajet.....	3
2. Connexion des composants.....	4
2.1 Connexion dans la chaîne de transmission analogique avec bruit et trajets multiples.....	4
2.2 Rôles des sondes dans la connexion.....	5
3. Tests.....	5
3.1 Objectifs des tests.....	5
3.2 Structure des tests.....	6
3.3 Exécution des tests.....	6
3.4 Résultats des tests.....	7
3.5 Couverture de code avec Emma.....	7
4. Simulation d'une transmission avec un canal à trajets multiples bruité gaussien.....	8
Conclusion.....	10

Introduction

L'étape 4 du projet introduit la simulation d'un canal de transmission avec trajets multiples combiné à un bruit blanc gaussien, si activé. Ce type de canal est caractérisé par des copies du signal émis qui arrivent à la destination avec un retard et une atténuation. Cela permet de simuler les effets des interférences ou des échos causés par des obstacles ou des réflexions dans des environnements réels. Le bruit gaussien est également ajouté (quand il est activé) au signal pour modéliser les perturbations qui affectent inévitablement la qualité de transmission.

L'objectif de cette étape est de comprendre l'impact de ces phénomènes sur la qualité de la transmission en analysant le taux d'erreur binaire, qui quantifie la proportion de bits erronés après réception. Cette simulation permet de tester la robustesse du système dans des conditions de transmission réalistes où les interférences entre symboles et le bruit sont omniprésents.

À travers la modélisation et l'implémentation d'un transmetteur analogique multi-trajet, nous avons pu observer l'effet du retard et de l'atténuation des trajets supplémentaires, ainsi que l'influence du bruit sur le signal.

1. Conception des composants

1.1 TransmetteurAnalogiqueMultiTrajet

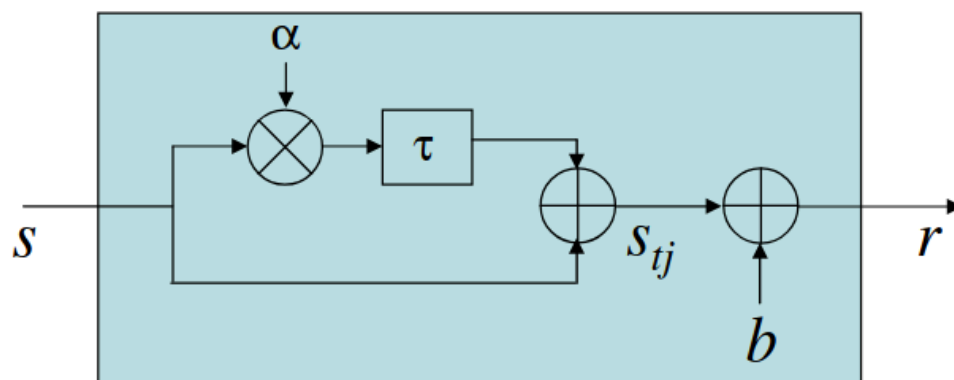
Le TransmetteurAnalogiqueMultiTrajet est une extension du Transmetteur. Ce transmetteur modélise un canal avec des copies retardées et atténuées du signal original (cinq au maximum). Les autres trajets sont caractérisés par deux paramètres principaux :

- **Tau** (variable τ) : Le retard du second trajet, exprimé en nombre d'échantillons.
- **Alpha** (variable α) : Le coefficient d'atténuation du second trajet, qui modélise la perte d'intensité du signal pour la copie retardée, entre 0 et 1.

Chaque échantillon est retardé et atténué, puis ajouté au signal global, pour chaque trajet, grâce à la méthode "ajouterTrajetsMultiples".

Il est possible de lancer le simulateur en mode multi-trajets avec ou sans bruit. De plus, nous avons ajouté un paramètre "-snr" en plus du paramètre "-snrpb", afin de définir le SNR global, snrpb indique lui le E_b/N_0 . Les deux sont fonctionnels, mais il n'est pas possible d'utiliser les deux simultanément.

Si le bruit est activé, alors nous utilisons la classe "TransmetteurAnalogiqueBruite", sur le signal avec multi-trajets. Le bruit blanc additif gaussien est généré et ajouté à chaque échantillon du signal pour simuler les perturbations naturelles du canal, en fonction du snr ou snrpb.



Signal reçu : $r(t) = s(t) + \alpha s(t - \tau) + b(t)$

2. Connexion des composants

La connexion des composants est essentielle pour permettre le passage fluide des informations depuis la source jusqu'à la destination finale, tout en tenant compte des transformations introduites par les trajets multiples et le bruit gaussien. Le simulateur suit une structure modulaire, où chaque composant est responsable d'une étape spécifique de la transmission.

2.1 Connexion dans la chaîne de transmission analogique avec bruit et trajets multiples

La chaîne de transmission comprend une source logique, un émetteur analogique, un transmetteur analogique multi-trajet (qui introduit le retard, l'atténuation et le bruit), un récepteur et des sondes. La connexion des composants se fait de la manière suivante :

- **Source → Émetteur** : La Source Fixe ou Aléatoire génère une séquence binaire qui est transmise à l'émetteur. Ce dernier modifie le signal binaire logique en un signal analogique, selon la modulation définie, le nombre d'échantillons, et l'amplitude.
- **Émetteur → Transmetteur Analogique Multi-Trajet** : Une fois que l'émetteur a modifié le signal logique en signal analogique, ce signal est transmis au TransmetteurAnalogiqueMultiTrajet. Ce composant modélise les trajets multiples. Les autres trajets sont définis par un retard (τ , dt) et une atténuation (α , ar).
- **Transmetteur Analogique Multi-Trajet → Transmetteur Analogique Bruité** : Le signal analogique, après avoir été modifié par les trajets multiples, est transmis au Transmetteur Analogique Bruité. Qui ajoute le bruit blanc Gaussien en fonction du SNR (ou E_b/N_0), et du nombre d'échantillons par bits.
- **Transmetteur Analogique Bruité → Récepteur** : Le signal analogique, après avoir été modifié par les trajets multiples et le bruit, est transmis au récepteur. Le rôle du récepteur est de démoduler ce signal analogique pour le convertir en information logique. Il utilise les amplitudes maximales et minimales du signal pour déterminer les bits correspondants (1 ou 0), en fonction de la modulation utilisée.
- **Récepteur → Destination finale** : Le récepteur transmet enfin l'information logique récupérée à la DestinationFinale, qui reçoit la séquence de bits démodulée et la compare avec l'information initialement émise par la source. Le simulateur utilise la destination finale pour calculer le Taux d'Erreur Binaire, permettant d'évaluer la qualité de la transmission en prenant en compte les erreurs introduites par les bruits.

Dans le cas où il n'y aurait pas de bruit (paramètre pas activé), le Transmetteur Analogique Multi-Trajet envoie directement les données au Récepteur.

2.2 Rôles des sondes dans la connexion

Les sondes permettent de visualiser l'évolution du signal à différentes étapes de la chaîne de transmission. Elles peuvent être placées après l'émetteur pour observer le signal analogique avant l'introduction du bruit, après le transmetteur pour visualiser l'impact des trajets multiples et du bruit, et après le récepteur pour vérifier le message logique reçu.

- **SondeAnalogique** : Capture et visualise le signal analogique à différents points (après l'émetteur, le transmetteur multi-trajet et le transmetteur bruité).
- **SondeLogique** : Visualise le message binaire émis avant modulation et reçu après la démodulation

3. Tests

3.1 Objectifs des tests

Les tests ont pour objectifs de :

- **Valider le comportement individuel de chaque composant** : vérifier que les sources, émetteurs, transmetteurs et récepteurs fonctionnent correctement et produisent les résultats attendus.
- **Tester l'intégration des composants** : s'assurer que la connexion des composants de la chaîne de transmission permet la transmission correcte du message et que le taux d'erreur binaire est calculé avec précision.
- **Vérifier l'impact des trajets multiples et du bruit** : évaluer comment le retard, l'atténuation et le bruit gaussien affectent le signal transmis et modifient la qualité de la transmission.
- **Mesurer la performance du simulateur** : calculer le TEB pour différents scénarios, en fonction des paramètres du canal (tau, alpha, SNR).

3.2 Structure des tests

Les tests ont été organisés à l'aide de JUnit, avec des tests unitaires et d'intégration pour chaque composant principal de la chaîne de transmission. Voici les principaux cas de test :

- **TransmetteurAnalogiqueMultiTrajetTest** : Cette classe de test permet de tester le comportement du transmetteur avec trajets multiples en simulant un canal avec différentes valeurs de tau (retard) et alpha (atténuation). Le test vérifie que le signal final contient à la fois le trajet direct et le trajet retardé, modifiés par le bruit si activé. Aussi, lève les bonnes exceptions s'il y a un problème, par exemple, avec des valeurs incorrectes.
- **SimulateurTest** : Cette classe de test valide l'intégration de tous les composants dans une chaîne de transmission complète. Le test vérifie que le TEB est correctement calculé en fonction du signal émis et reçu et qu'il varie en fonction du rapport signal/bruit.

3.3 Exécution des tests

Les tests sont exécutés à l'aide d'un script JUnit qui permet de lancer tous les tests de manière automatisée et groupée. L'exécution est gérée par la classe AllTests, qui regroupe l'ensemble des tests unitaires et d'intégration pour s'assurer que chaque composant fonctionne individuellement, et que l'ensemble du simulateur est stable. Pour exécuter, nous compilons avec le script `“./compile”`, puis utilisons le script `“./runTests”`

Les tests sont lancés dans différents scénarios de transmission, en ajustant des paramètres tels que le SNR, τ (retard des trajets) et α (atténuation des trajets). Ces variations permettent de vérifier que le simulateur réagit correctement à des environnements de transmission différents. De plus, des paramètres incorrects sont utilisés pour vérifier que les bonnes exceptions sont données et qu'aucun bug n'est produit.

3.4 Résultats des tests

Les résultats des tests montrent que les composants individuels (source, émetteur, transmetteur, récepteur) fonctionnent correctement selon leurs spécifications. Les tests sur le TransmetteurAnalogiqueMultiTrajet ont démontré que les valeurs de tau et alpha influencent effectivement la qualité du signal reçu et que le bruit gaussien affecte la précision du signal. Le TEB est correctement calculé et varie comme prévu avec la réduction du SNR. Dans les scénarios avec faible SNR, le TEB augmente, ce qui montre que le simulateur réagit bien aux conditions de transmission dégradées.

3.5 Couverture de code avec Emma

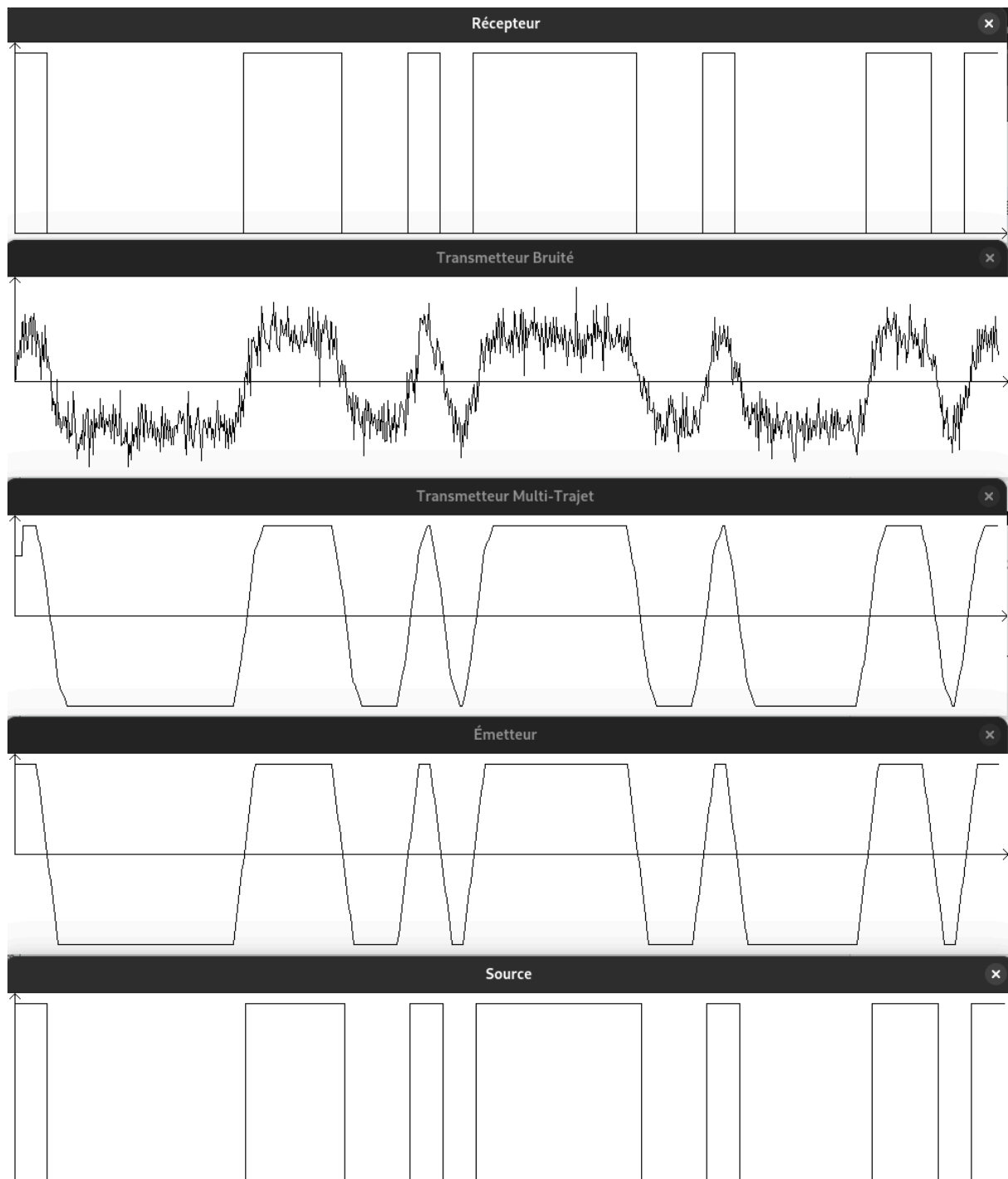
L'outil Emma a été utilisé pour mesurer la couverture des tests sur l'ensemble du projet. Les résultats sont les suivants :

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ sit213	86,4 %	8 044	1269	9 313
▶ tests	91,1 %	4 678	459	5 137
▼ src	80,6 %	3 366	810	4 176
▼ transmetteurs	94,9 %	1 442	78	1 520
▶ Emetteur.java	95,4 %	436	21	457
▶ Recepteur.java	96,2 %	333	13	346
▶ TransmetteurAnalogiqueBruite.java	87,1 %	277	41	318
▶ TransmetteurAnalogiqueMultiTrajet.java	98,9 %	278	3	281
▶ TransmetteurAnalogiqueParfait.java	100,0 %	47	0	47
▶ TransmetteurParfait.java	100,0 %	36	0	36
▶ Transmetteur.java	100,0 %	35	0	35
▶ simulateur	71,3 %	982	395	1 377
▶ visualisations	64,8 %	612	333	945
▶ sources	100,0 %	174	0	174
▶ information	96,6 %	113	4	117
▶ destinations	100,0 %	43	0	43

Les résultats montrent une couverture élevée, confirmant que l'essentiel du code a été testé et validé. La couverture légèrement plus faible de certains éléments, indique que des cas spécifiques peuvent être ajoutés lors des futures itérations pour renforcer la robustesse des tests. Globalement, la couverture est très satisfaisante, assurant que le projet a été largement vérifié pour détecter d'éventuelles anomalies ou régressions. Certaines classes ou méthodes n'ont pas besoin d'être testées (par exemple, la création des fichiers), ceci influe sur la couverture des tests.

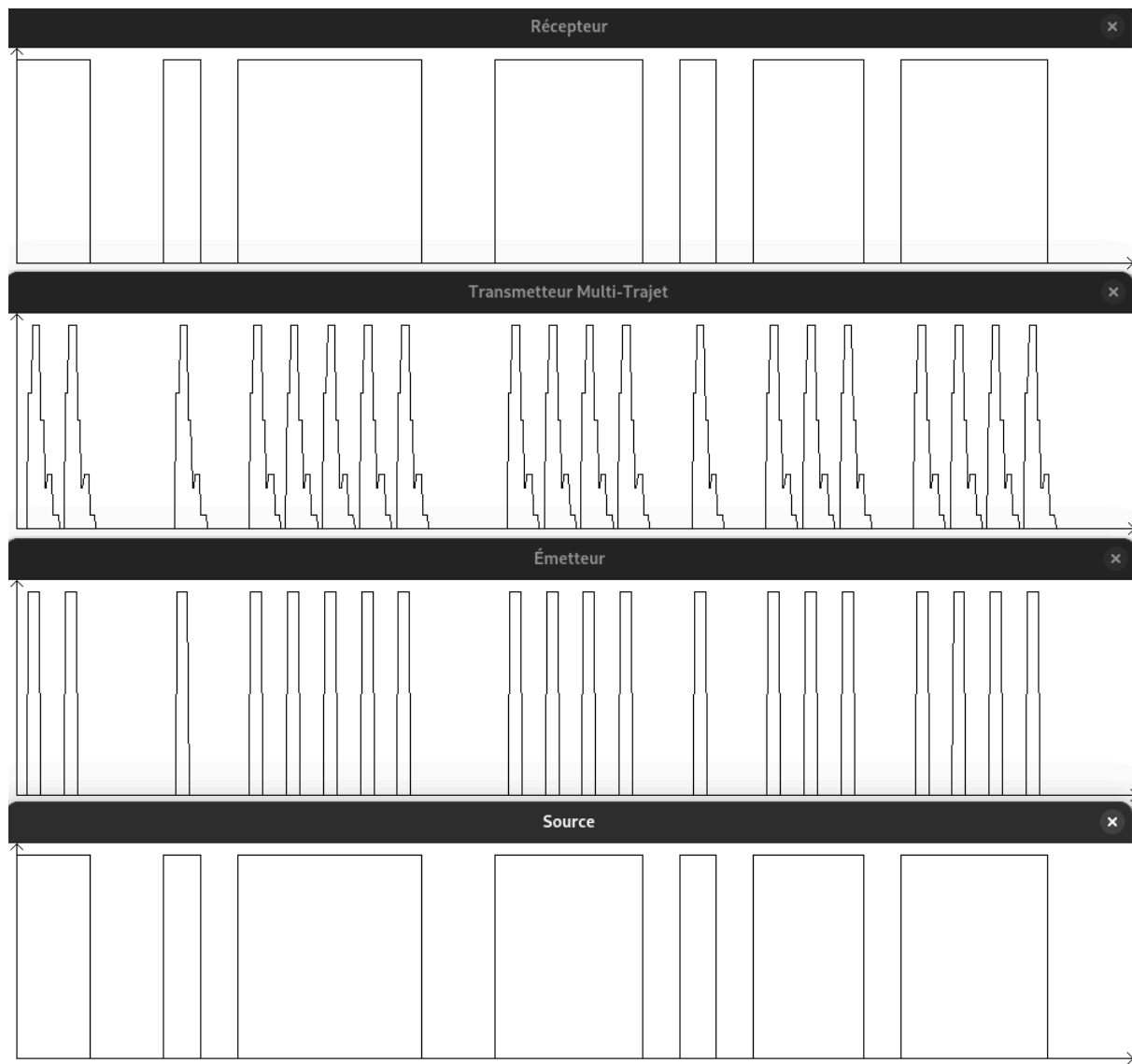
4. Simulation d'une transmission avec un canal à trajets multiples

En lançant avec du bruit et un multi trajet “./simulateur -s -mess 30 -form NRZT -seed 1 -nbEch 30 -ampl -1.0 1.0 -snrpb 20 -ti 8 0.5”, on obtient :



On observe bien une distorsion du signal amenée par le multi-trajets, le signal semble plus arrondi. En utilisant la modulation NRZT, on retrouve un problème, les quelques premiers échantillons ne sont pas bien gérés

Et, sans bruit, avec plusieurs multi-trajets `./simulateur -s -mess 30 -ti 3 0.5 10 0.3 15 0.1`



Dans cette simulation sans bruit, l'effet des composantes multi-trajets est clairement visible. Chaque trajet introduit une version retardée du signal transmis avec des niveaux d'atténuation variables.

Avec des valeurs de multi-trajet (et bruit), plus importante, le TEB serait significativement affecté négativement.

Conclusion

L'étape 4 du projet a permis de valider l'implémentation des composants de transmission analogique via un canal à trajets multiples combiné à un bruit blanc additif gaussien. À travers cette simulation, nous avons démontré que la chaîne de transmission réagit comme prévu face aux perturbations, notamment en introduisant un retard et une atténuation du signal, ainsi qu'un bruit gaussien quand spécifié.

Les tests ont montré que le Taux d'Erreur Binaire varie en fonction du rapport signal sur bruit et des paramètres de retard et d'atténuation, confirmant que ces facteurs affectent la qualité de la transmission comme attendu.

L'intégration du module TransmetteurAnalogiqueMultiTrajet a été réalisée et testée avec succès. Cette étape montre que la chaîne de transmission peut être simulée dans des environnements comportant des trajets multiples et du bruit, avec une évolution du TEB conforme aux prévisions.

Ces résultats constituent une base solide pour envisager des améliorations du système, notamment par l'ajout de mécanismes de correction d'erreurs ou l'introduction de nouveaux types de bruit pour simuler des environnements de transmission plus variés.

10/10/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 5

Introduction.....	2
1. Conception des composants.....	3
1.1 CodageEmission.....	3
1.2 DecodageReception.....	3
2. Connexion des composants.....	4
2.1 Connexion dans la chaîne de transmission avec codage de canal.....	4
2.2 Rôles des sondes dans la connexion.....	6
3. Tests.....	7
3.1 Objectifs des tests.....	7
3.2 Structure des tests.....	7
3.3 Exécution des tests.....	8
3.4 Résultats des tests.....	8
3.5 Couverture de code avec Emma.....	9
4. Simulation d'une transmission avec codeur.....	10
Conclusion.....	16

Introduction

L'étape 5 du projet introduit l'intégration d'un codage de canal dans la chaîne de transmission afin de réduire le Taux d'Erreur Binaire dans des environnements bruités. Ce codage est appliqué après la source et avant la destination et permet de transformer chaque bit logique en une séquence de trois bits pour améliorer la qualité de la transmission. L'objectif est de faciliter la détection et la correction des erreurs qui peuvent survenir lors de la transmission dans des conditions bruitées.

Le codage de canal offre deux avantages majeurs: l'amélioration de la synchronisation entre l'émetteur et le récepteur, en empêchant l'envoi de séquences prolongées de bits identiques, et la correction d'erreurs simples sur les paquets de trois bits. Si une erreur affecte un seul bit dans un paquet, le décodeur est capable de la détecter et de la corriger, ce qui améliore considérablement la fiabilité de la transmission.

L'objectif de cette étape est de mesurer l'impact du codage de canal sur la qualité de la transmission, en comparant le TEB avec et sans codage dans différents scénarios, avec des variations de E_b/N_0 et des formes d'onde. À travers l'implémentation du codage, nous cherchons à démontrer l'amélioration de la robustesse du système face aux perturbations dues au bruit.

1. Conception des composants

1.1 CodageEmission

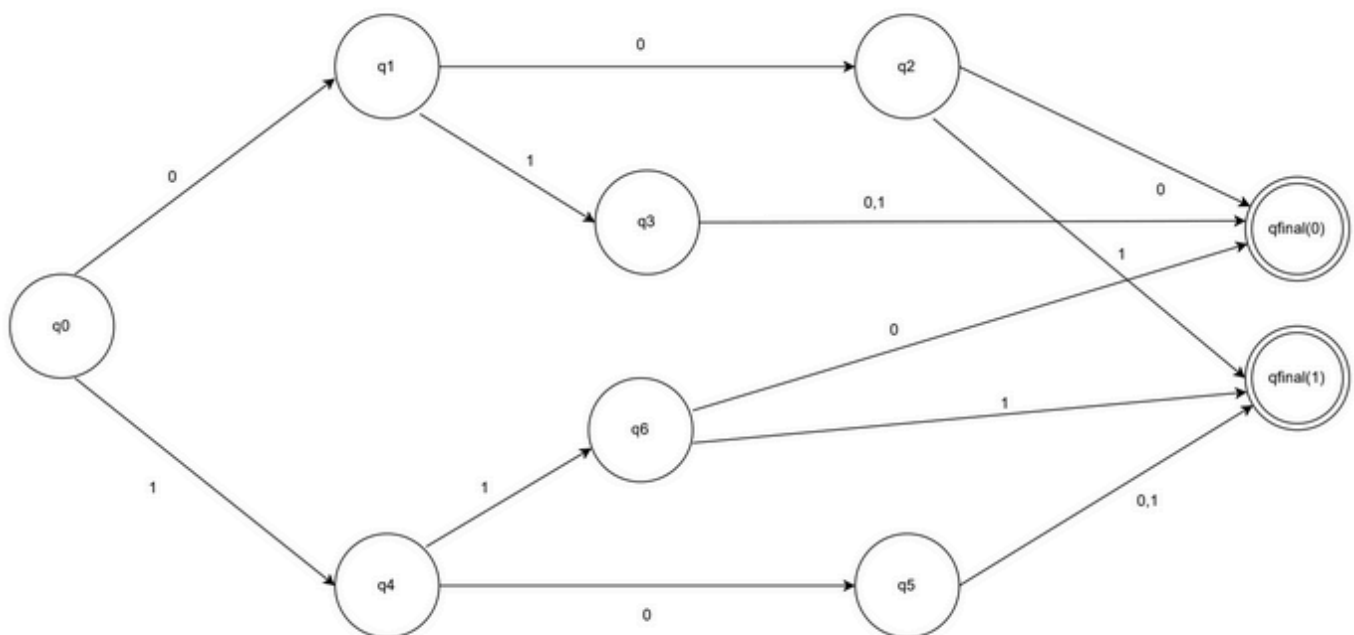
Le CodageEmission est un composant ajouté à la chaîne de transmission, juste après la source, pour transformer chaque bit logique en une séquence de trois bits. Défini dans le sujet.

Pour chaque bit **0** généré par la source, l'encodeur produit la séquence **010**. Pour chaque bit **1**, il produit la séquence **101**. Ce mécanisme ajoute une redondance qui permet au récepteur de détecter et corriger les erreurs survenant dans un paquet de trois bits.

1.2 DecodageReception

Le DecodageReception est le composant chargé de décoder les paquets de trois bits reçus en un bit unique. Ce composant est placé juste avant la destination pour corriger d'éventuelles erreurs dans les paquets reçus.

Le décodeur analyse les paquets de trois bits. Pour cela, nous utilisons un automate, créé pour être conforme aux consignes. C'est le suivant :



Il est implémenté en Java dans la méthode automate() de la classe.

2. Connexion des composants

La connexion des composants est essentielle pour permettre le passage fluide des informations depuis la source jusqu'à la destination finale, tout en tenant compte des transformations introduites par les trajets multiples et le bruit gaussien. Le simulateur suit une structure modulaire, où chaque composant est responsable d'une étape spécifique de la transmission.

2.1 Connexion dans la chaîne de transmission avec codage de canal

Dans cette étape, la chaîne de transmission inclut également un encodeur et un décodeur afin d'introduire un codage de canal pour améliorer la qualité de transmission, grâce à l'option -codeur. La connexion des composants se fait de la manière suivante, dans le cas d'une transmission bruitée avec multi-trajet (la plus complète) :

- **Source → Codage d'émission** : la Source Fixe ou Source Aléatoire génère une séquence binaire. Cette séquence est transmise à l'encodeur (CodageEmission), qui transforme chaque bit logique en une séquence de trois bits.
- **Codage d'émission → Émetteur** : l'information encodée est ensuite transmise à l'émetteur analogique, qui convertit le signal binaire en signal analogique. Ce signal est modifié selon la modulation définie (NRZ, NRZT, RZ), le nombre d'échantillons par bit, et les amplitudes maximale et minimale
- **Émetteur → Transmetteur Analogique Multi-Trajet** : une fois que l'émetteur a converti le signal logique en signal analogique, ce signal est transmis au Transmetteur Analogique Multi-Trajet. Ce composant modélise les trajets multiples, définis par un retard (τ) et une atténuation (α). Chaque trajet supplémentaire est retardé et atténué avant d'être ajouté au signal global
- **Transmetteur Analogique Multi-Trajet → Transmetteur Analogique Bruité** : après avoir été modifié par les trajets multiples, le signal analogique est transmis au Transmetteur Analogique Bruité. Ce composant ajoute un bruit blanc gaussien à chaque échantillon du signal en fonction du SNR ou du E_b/N_0 (-snr ou -snrpb, les deux sont implémentés :) spécifié dans les paramètres de simulation. Le bruit simule les perturbations présentes dans les canaux réels
- **Transmetteur Analogique Bruité → Récepteur** : le signal bruité est ensuite transmis au Récepteur Analogique, qui se charge de démoduler le signal analogique pour le convertir en information logique. Le récepteur utilise les amplitudes maximale

et minimale du signal pour déterminer les bits correspondants (1 ou 0), en fonction de la modulation utilisée.

- **Récepteur → Décodage de réception** : une fois l'information logique récupérée par le récepteur, elle est transmise au décodeur (DecodageReception). Ce composant prend les paquets de trois bits reçus et les convertit en un seul bit en détectant et corrigeant les erreurs éventuelles dans les paquets.
- **Décodage de réception → Destination finale** : après décodage, l'information binaire est transmise à la DestinationFinale, où la séquence reçue est comparée à l'information initialement émise par la source. Le TEB est ensuite calculé, permettant d'évaluer la qualité de la transmission en prenant en compte les erreurs introduites par les trajets multiples et le bruit.

Le codeur peut être utilisé sans les multi-trajets, ni le bruit, ni le passage analogique, avec le même type de fonctionnement pour chaque cas en changeant les transmetteurs entre le codeur et décodeur.

Pour avoir de meilleures performances, le récepteur pourrait être modifié, afin de détecter les blocs de trois bits et éventuellement différents décalages ou autre pour mieux détecter. Pour le moment le récepteur est le plus simpliste, il prend un seuil de moyenne en fonction de l'amplitude et décide par rapport à la moyenne du signal reçu. Avec le bit entier si la modulation est NRZ, sur le deuxième tiers du bit si RZ ou NRZT. Ce qui peut expliquer les moins bonnes performances de ces deux dernières modulations.

2.2 Rôles des sondes dans la connexion

Les sondes permettent de visualiser l'évolution du signal à différentes étapes de la chaîne de transmission. Elles fournissent des informations précieuses pour comprendre comment le signal est affecté par les différentes transformations, notamment la modulation, les trajets multiples, et le bruit gaussien. Grâce aux sondes, il est possible d'observer le comportement du système en temps réel et d'analyser les variations du signal.

Les sondes sont placées à chaque étape de la chaîne de transmission, lorsqu'elles sont activées via l'option -s. Après la source, après le codeur, après, l'émetteur, après le transmetteur multi-trajets, après le transmetteur bruité, après le récepteur, après le décodeur.

Les types de sondes sont :

- **SondeAnalogique** : Cette sonde capture et visualise le signal analogique à différents points du système, notamment après l'émetteur, après le transmetteur multi-trajet, et après le transmetteur bruité. Elle permet de suivre la déformation du signal à mesure qu'il traverse le canal de transmission.
- **SondeLogique** : La SondeLogique visualise la séquence binaire avant et après la modulation. Avant la modulation, elle montre le message logique tel qu'il est généré par la source, et après la démodulation, elle montre le message logique reçu par la destination.

3. Tests

3.1 Objectifs des tests

Les tests ont pour objectifs de :

- **Valider le comportement du codage et du décodage** : vérifier que le composant `CodageEmission` transforme chaque bit en une séquence de trois bits, et que le `DecodageReception` est capable de reconstruire le bit original tout en corrigeant les erreurs éventuelles dans les paquets de trois bits
- **Tester l'intégration des composants avec le codage** : s'assurer que le simulateur fonctionne correctement lorsque le codage est activé, que l'information est correctement encodée avant transmission, puis décodée à la réception. Le Taux d'Erreur Binaire doit diminuer grâce au mécanisme de correction d'erreurs du décodeur
- **Vérifier l'impact du bruit et des trajets multiples** : évaluer l'effet du bruit et des trajets multiples sur le signal encodé, et vérifier que le décodeur corrige efficacement les erreurs introduites lors de la transmission
- **Mesurer la performance du simulateur avec codage** : comparer le TEB avec et sans codage pour différents scénarios de transmission (SNR, trajets multiples) et mesurer l'amélioration

3.2 Structure des tests

Les tests sont organisés via JUnit, avec des tests unitaires pour valider les comportements individuels des composants de codage et décodage, ainsi que des tests d'intégration pour évaluer leur fonctionnement dans une chaîne de transmission complète.

Voici les principaux cas de test :

CodageEmissionTest :

- `testRecevoirAndEmettreValidInformation` : vérifie que l'encodage de bits se fait correctement. Par exemple, un 1 est transformé en 101 et un 0 en 010
- `testRecevoirNullInformation` et `testEmettreNullInformation` : s'assurent que des exceptions sont levées si des informations nulles ou vides sont reçues ou émises

- `testRecevoirWithMultipleDestinations` : teste que l'information encodée est correctement transmise à plusieurs destinations connectées, en s'assurant que toutes reçoivent la même séquence encodée

DecodageReceptionTest :

- `testAutomate` : teste la fonction automate, qui reconstruit le bit original à partir de trois bits encodés, en corrigeant si nécessaire
- `testRecevoirInformationValide` : vérifie que le décodeur reçoit correctement une information encodée, la traite et corrige les erreurs, si présentes
- `testRecevoirNullInformation` et `testRecevoirEmptyInformation` : s'assurent que des exceptions sont levées pour des informations nulles ou vides
- `testRecevoirInformationIndivisibleParTrois` : vérifie qu'une exception est levée si l'information reçue n'est pas divisible par trois (condition nécessaire pour le décodage)

3.3 Exécution des tests

Les tests sont exécutés via un script JUnit qui compile et exécute l'ensemble des tests. Cela garantit que chaque composant fonctionne correctement individuellement et dans le cadre d'une transmission avec codage et décodage activés. On utilise le script `./compile` pour compiler les tests, puis `./runTests` pour les exécuter, divers paramètres de transmission sont utilisés pour simuler des environnements avec du bruit et des trajets multiples, garantissant que le codage améliore la qualité de la transmission.

3.4 Résultats des tests

Les résultats des tests montrent que les composants `CodageEmission` et `DecodageReception` fonctionnent correctement selon leurs spécifications. Les tests ont confirmé que le codage transforme chaque bit logique en une séquence de trois bits, et que le décodage permet de reconstruire le bit d'origine en corrigeant les erreurs éventuelles dans les paquets reçus.

Les tests unitaires sur le `CodageEmission` ont démontré que les séquences générées pour chaque bit sont bien conformes aux attentes : 0 est transformé en 010 et 1 en 101. Les informations ont été correctement transmises et reçues par plusieurs destinations connectées.

De même, les tests du DecodageReception ont validé la capacité du décodeur à interpréter les paquets de trois bits et à corriger les erreurs d'un seul bit. Les exceptions ont été correctement levées lorsque des informations non conformes étaient reçues, telles que des messages vides ou des séquences non divisibles par trois.

3.5 Couverture de code avec Emma

L'outil Emma a été utilisé pour mesurer la couverture des tests sur l'ensemble du projet. Les résultats sont les suivants :

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ sit213	84,9 %	8 963	1 593	10 556
▶ tests	91,3 %	5 039	483	5 522
▼ src	77,9 %	3 924	1 110	5 034
▼ transmetteurs	94,9 %	1 953	104	2 057
▶ Emetteur.java	95,4 %	433	21	454
▶ DecodageReception.java	97,4 %	371	10	381
▶ Recepteur.java	96,2 %	330	13	343
▶ TransmetteurAnalogiqueBruite.java	86,0 %	277	45	322
▶ TransmetteurAnalogiqueMultiTrajet.java	98,9 %	278	3	281
▶ CodageEmission.java	92,4 %	146	12	158
▶ TransmetteurAnalogiqueParfait.java	100,0 %	47	0	47
▶ TransmetteurParfait.java	100,0 %	36	0	36
▶ Transmetteur.java	100,0 %	35	0	35
▶ simulateur	60,6 %	1 033	671	1 704
▶ visualisations	64,7 %	608	331	939
▶ sources	100,0 %	174	0	174
▶ information	96,6 %	113	4	117
▶ destinations	100,0 %	43	0	43

Les résultats montrent une couverture élevée, confirmant que l'essentiel du code a été testé et validé. La couverture légèrement plus faible de certains éléments, indique que des cas spécifiques peuvent être ajoutés lors des futures itérations pour renforcer la robustesse des tests. Globalement, la couverture est très satisfaisante, assurant que le projet a été largement vérifié pour détecter d'éventuelles anomalies ou régressions. Certaines classes ou méthodes n'ont pas besoin d'être testées (par exemple, la création des fichiers), ceci influe sur la couverture des tests.

4. Simulation d'une transmission avec codeur

Utilisation du codeur :

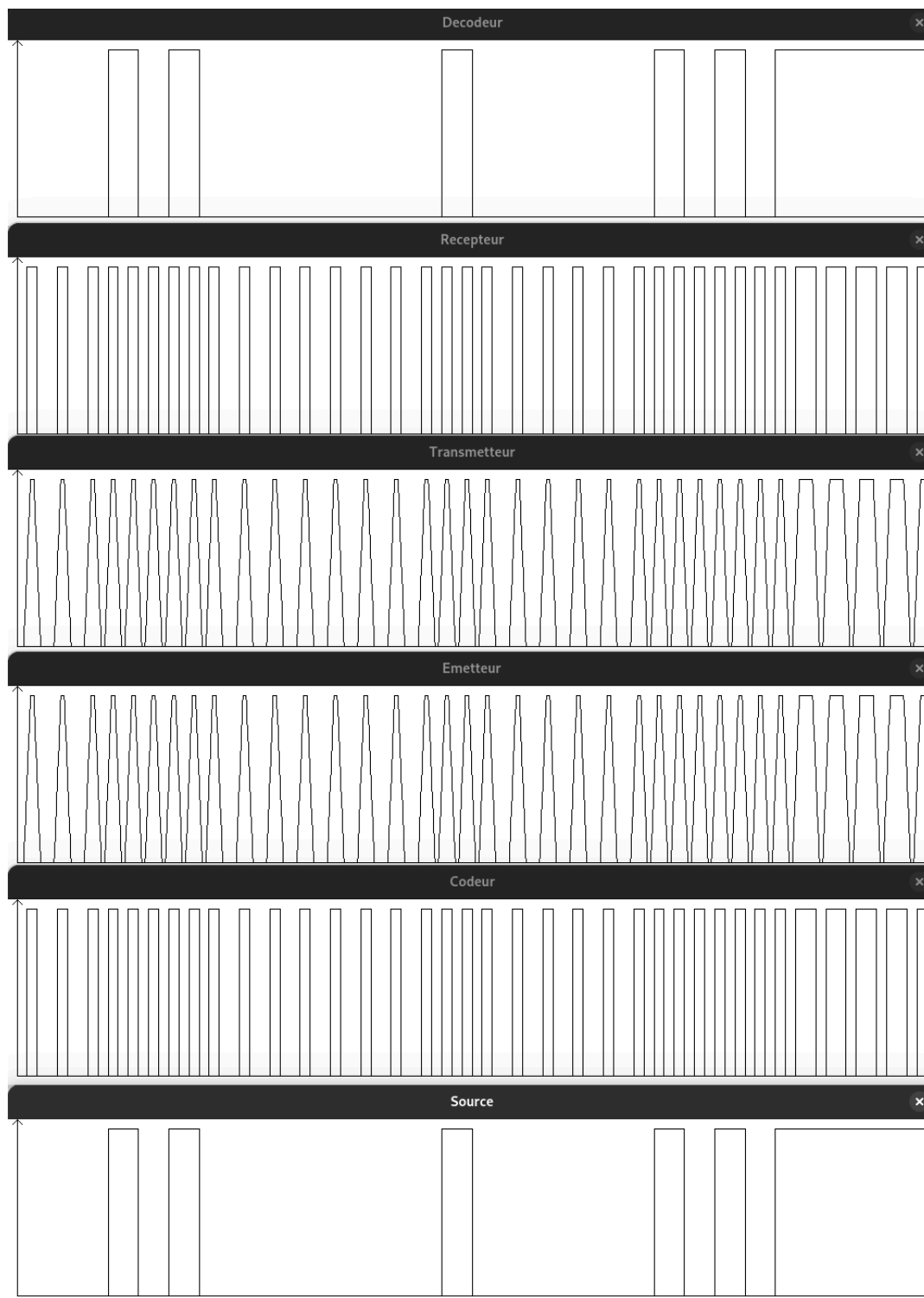
Transmission logique

```
~/sit213 main > ./simulateur -s -mess 30 -codeur  
java Simulateur -s -mess 30 -codeur => TEB : 0.0
```



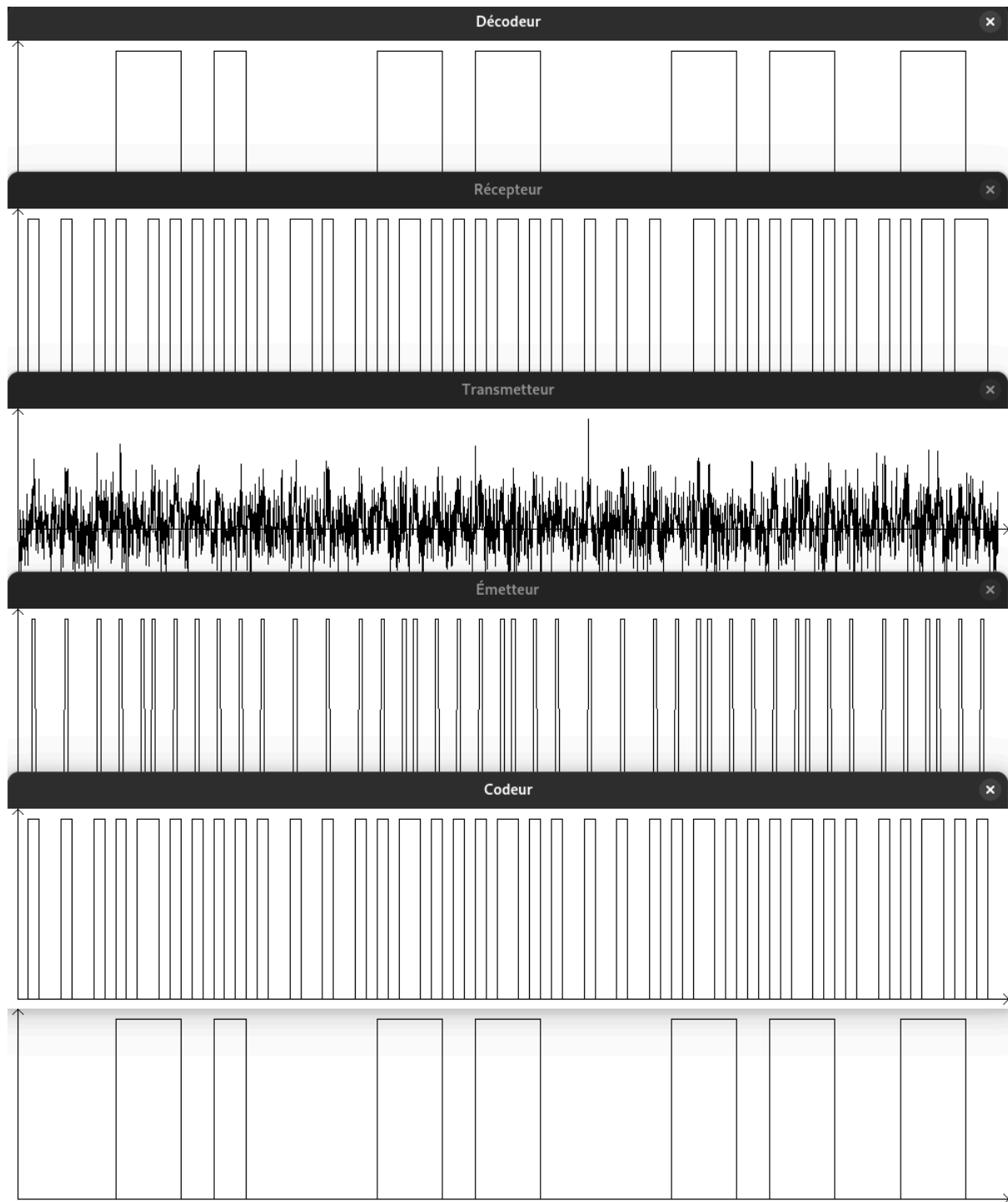
Transmission analogique parfaite

```
~/sit213 main > ./simulateur -s -mess 30 -codeur -form NRZT
java Simulateur -s -mess 30 -codeur -form NRZT => TEB : 0.0
```



Transmission analogique bruitée

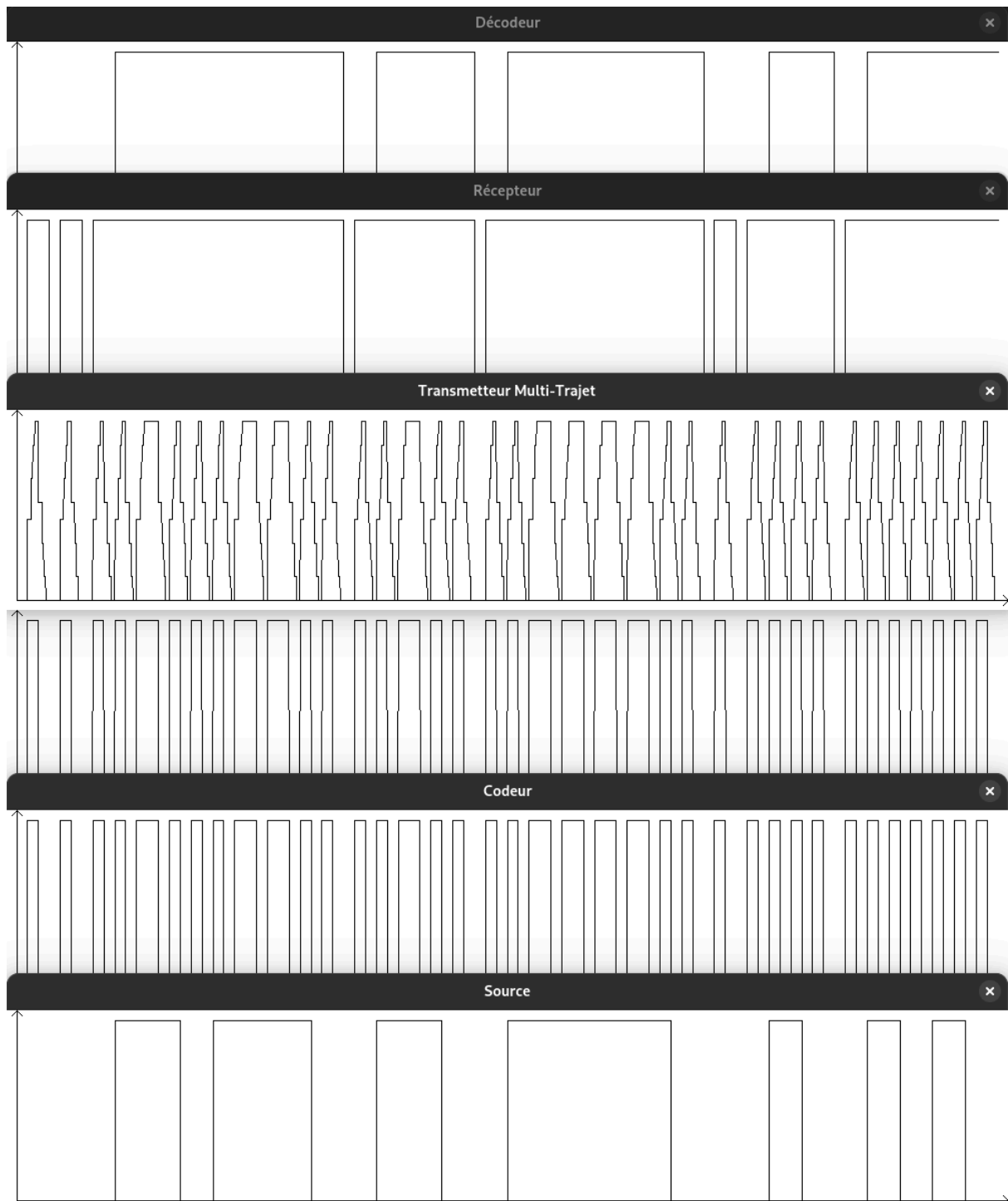
```
~/sit213 main > ./simulateur -s -mess 30 -codeur -form RZ -snrpb 6  
java Simulateur -s -mess 30 -codeur -form RZ -snrpb 6 => TEB : 0.0
```



On remarque bien la correction d'erreur, le codeur et récepteur sont légèrement différents. Cependant, il n'y a aucune erreur à la destination.

Analogique multi-trajets sans bruits

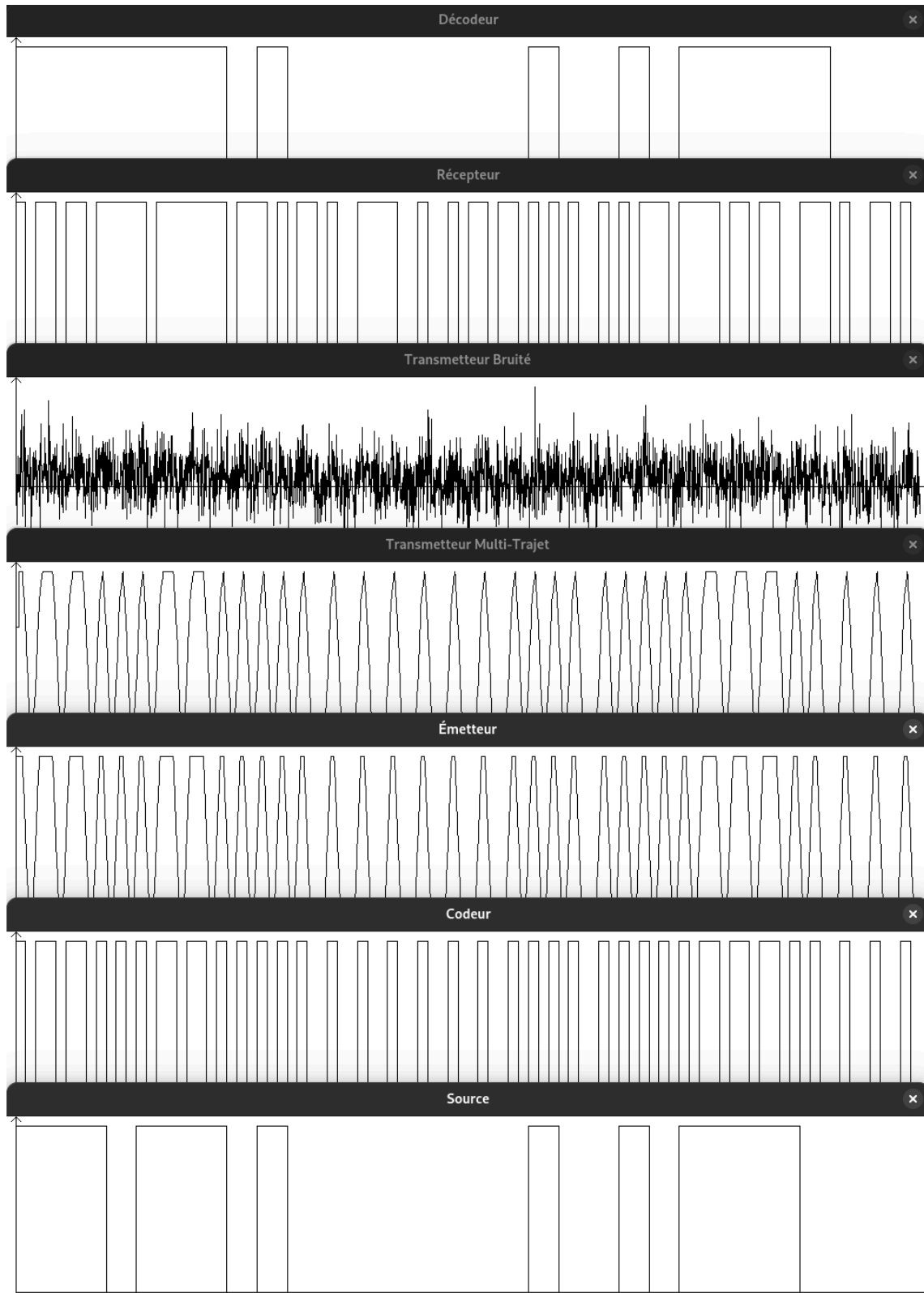
```
~/sit213 main > ./simulateur -s -mess 30 -codeur -form NRZ -ti 10 0.5 15 0.4 20 0.3
java Simulateur -s -mess 30 -codeur -form NRZ -ti 10 0.5 15 0.4 20 0.3 => TEB : 0.2
```



Certaines erreurs sont corrigées, mais pas toutes.

Analogique multi-trajets avec bruits

```
~/sit213 main > ./simulateur -s -mess 30 -codeur -form NRZT -ti 10 0.5 -snrpb 5
java Simulateur -s -mess 30 -codeur -form NRZT -ti 10 0.5 -snrpb 5 => TEB : 0.06666667
```

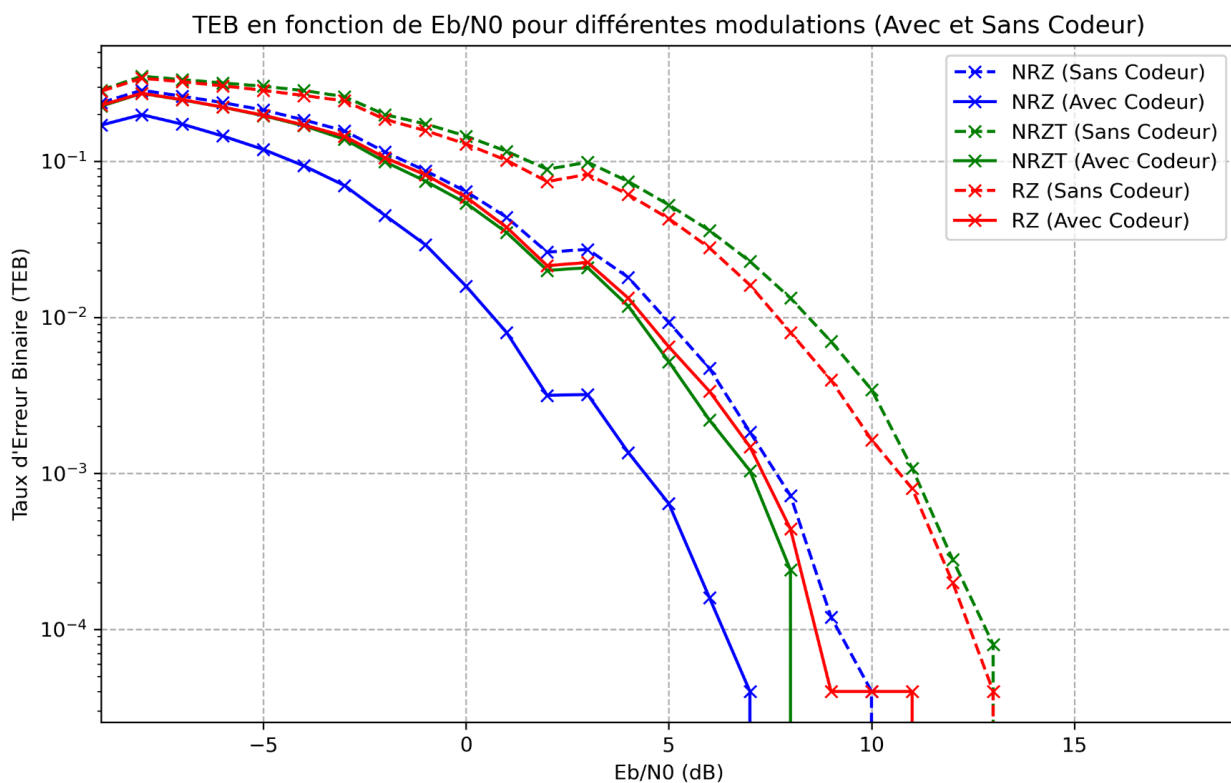


Deux erreurs, malgré le codeur et récepteur différents.

Pour vérifier le fonctionnement du codage et son efficacité, nous faisons des courbes de TEB en fonction de E_b/N_0 . Comme dans l'étape 3, nous faisons évoluer le E_b/N_0 et regardons le SNR pour chaque type de modulations, cette fois avec et sans codage.

Nous affichons le Taux d'Erreur Binaire sous échelle logarithmique et E_b/N_0 en échelle linéaire. Cela est différent de l'affichage à l'étape 3 qui n'était pas adapté et moins lisible. De même, un léger lissage est effectué (moyenne mobile) mais moins efficace, car moins de mesures.

Sur des messages de 5000 bits (la simulation a pris une heure).



On retrouve ainsi les trois types de modulations avec et sans codage, les transmissions avec codage sont toutes meilleures que celles sans. Le NRZ avec codeur est le plus performant, suivi du NRZT codeur, RZ codeur et NRZ sans codeur proches.

On gagne 3 dB en moyenne.

Nous remarquons que pour $E_b/N_0 = 2$ dB et $E_b/N_0 = 3$, TEB reste stable ou augmente légèrement pour toutes les modulations avec ou sans codage. Nous ignorons pourquoi. Avec plus de points, la courbe aurait pu être lisible.

Conclusion

En conclusion, le mécanisme de codage et de décodage mis en place permet d'améliorer la transmission des données en ajoutant la transformation des bits logiques en séquences de trois bits. Le CodageEmission et le DecodageReception fonctionnent de manière complémentaire pour encoder les bits à l'envoi et corriger les erreurs à la réception.

Les tests réalisés ont validé l'efficacité du système, notamment face au bruit et aux multi-trajets. L'intégration du codeur dans une chaîne de transmission complète avec modulation et bruit a montré des améliorations dans le Taux d'Erreur Binaire (TEB), notamment avec la modulation NRZ, où un gain de 3 dB a été observé.

Malgré les bonnes performances générales, certaines erreurs persistent dans des conditions de transmission bruitée et avec multi-trajets.

10/10/2024

BODIN Noé

COLIN Guillaume

DOUANT Antoine

LE COQ Justine

Rapport SIT213

Atelier Logiciel

Simulation d'un système de transmission

Étape 6

Introduction.....	2
Améliorations.....	3
Environnement 1.....	4
Environnement 2.....	8
Conclusion.....	11

Introduction

Ce rapport vise à répondre aux exigences d'un client qui souhaite déployer un réseau de capteurs câblés dans deux environnements distincts. Le premier environnement concerne un canal de transmission de type bruit blanc additif gaussien (BBAG), alors que le second implique un canal de propagation à trajets multiples. L'objectif est de garantir des performances en termes de taux d'erreur binaire (TEB) tout en minimisant la consommation énergétique, et de garantir un TEB en fonction d'un débit.

Au cours de cette étape, nous avons également apporté des améliorations à notre simulateur, ce qui a permis de rendre les simulations beaucoup plus rapides et plus efficaces.

Ce rapport détaille les simulations effectuées et les résultats obtenus pour chaque environnement, en tenant compte des améliorations apportées au simulateur.

Améliorations

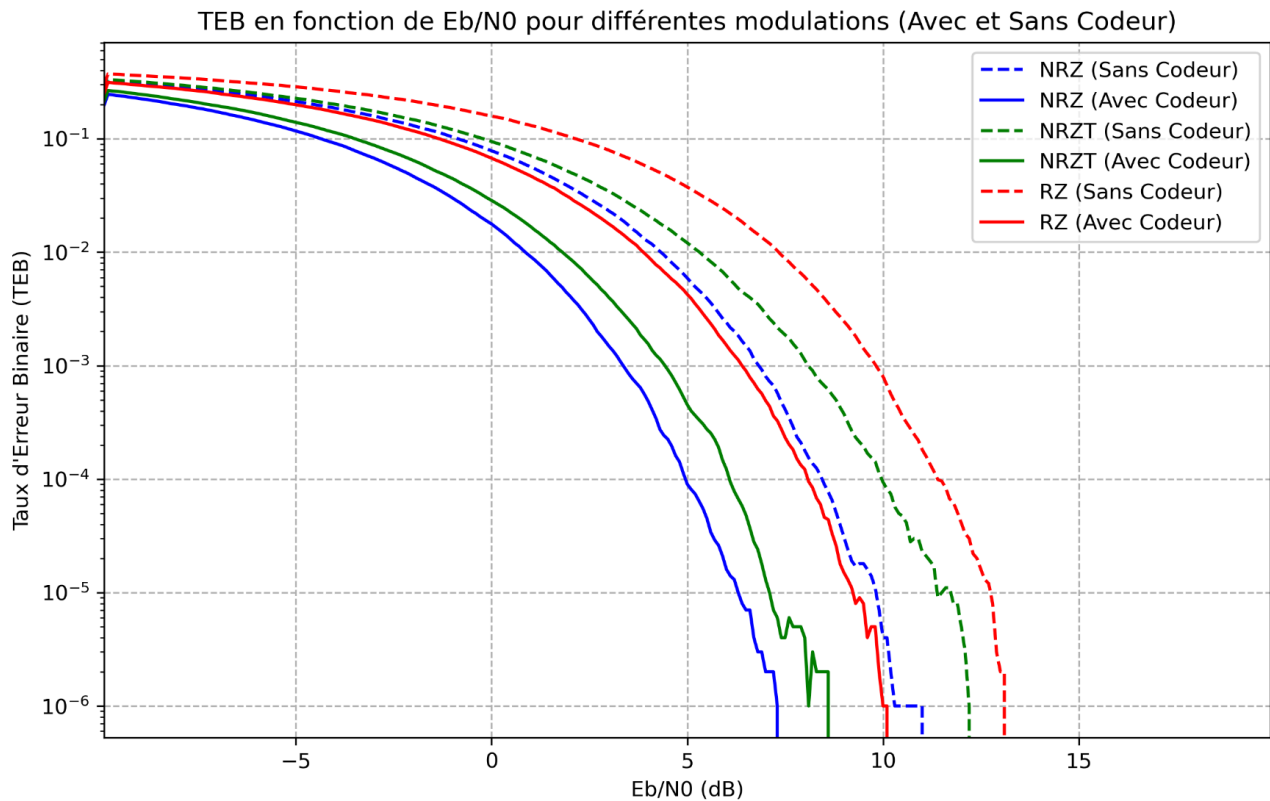
Dans un premier temps, nous avons apporté des améliorations à notre simulateur :

- Optimisation en utilisant les ArrayList a la place des LinkedList (initialement présent dans le code) jusqu'à 50 000 fois plus rapide, une simulation de 7 heures devient 0,5s. Il aurait fallu utiliser des itérateurs pour être performants avec LinkedList, ce qui aurait provoqué de leurs changements.
- Remplacements du type double en Double pour permettre null en valeur de SNR et SNRpb, et vérification du bruit ou non (avant, il était impossible d'avoir 0 dB de bruit)
- Améliorations dans le récepteur
- Ajouts de tests du simulateur

Tout cela améliore nos courbes de TEB en fonction de E_b/N_0 et les changent légèrement (surtout pour NRZT).

Environnement 1

Sur des messages de 200000 bits, d'amplitude -1 et 1, 30 échantillons par bits, nous avons ces courbes de TEB en fonction de E_b/N_0 (de -10 à 20 dB, tous les 0.1 dB) :



Paramètre	Valeur
Type de bruit	Blanc additif gaussien
Bruit N_0	-80 dBm/Hz
Atténuation totale α	-40 dB
Seuil TEB	10^{-3}
Consommation	Minimale
Capacité de la batterie	3 J
Volume de données utiles	10^6 bits/jour

La courbe de TEB en fonction de E_b/N_0 a été refaite après les améliorations citées précédemment, les valeurs changent légèrement, mais les calculs n'ont pas été refaits, il y a donc un léger décalage (moins de 1 dB, NRZ : 6.8 dB au lieu de 7.7 dB).

Choix de la forme d'onde :

D'après nos courbes, pour un TEB de 10^{-3} , les valeurs de E_b/N_0 requises sont :

- NRZ : 7.7 dB
- RZ : 10 dB
- NRZT : 8.5 dB

NRZ est la forme d'onde la plus économe en énergie, car elle nécessite le moins de puissance pour atteindre le TEB cible. Nous allons donc utiliser les valeurs du codage NRZ pour la suite.

Calcul de la puissance d'émission (Pt) :

Conversion de N_0 en Watts/Hz :

$$N_0 \text{ (dBm/Hz)} = -80 \text{ dBm/Hz}$$

$$N_0 \text{ (W/Hz)} = 10^{(N_0(\text{dBm/Hz})/10)} * 10^{-3} = 10^{-11} \text{ W/Hz}$$

Calcul de E_b (énergie par bit) :

$$E_b/N_0 \text{ (dB)} = 7.7 \text{ dB (pour NRZ)}$$

$$E_b/N_0 \text{ (linéaire)} = 10^{(E_b/N_0(\text{dB})/10)} = 5.89$$

$$E_b = E_b/N_0 \text{ (linéaire)} * N_0 = 5.89 * 10^{-11} \text{ J/bit}$$

Calcul de la puissance reçue (P_r) :

$$P_r = E_b * R_b$$

où R_b est le débit binaire. Pour maximiser la durée de vie de la batterie, nous voulons minimiser la puissance, donc nous allons choisir le débit minimal qui satisfait les contraintes du client.

Le client transmet 10^6 bits/jour, soit environ 11,57 bits/seconde. Nous allons arrondir ce chiffre à 12 bits/seconde pour avoir une marge de sécurité.

$$P_r = 5.89 \cdot 10^{-11} \text{ J/bit} \cdot 12 \text{ bits/s} = 7.07 \cdot 10^{-10} \text{ W}$$

Calcul de la puissance d'émission (P_t) :

P_t dépend de l'atténuation A

$$P_t = P_r \cdot 10^{(A/10)} = 7.07 \cdot 10^{-10} \text{ W} \cdot 10^{(40/10)} = 7.07 \cdot 10^{-6} \text{ W} = 7.07 \mu\text{W}$$

Calcul de l'autonomie de la batterie :

Énergie totale de la batterie : 3 J

Consommation quotidienne :

$$P_t \cdot 24 \text{ heures} \cdot 3600 \text{ secondes/heure} = 7.07 \cdot 10^{-6} \text{ W} \cdot 86400 \text{ s} = 0.61 \text{ J/jour}$$

Autonomie de la batterie :

Énergie totale / Consommation quotidienne = 3 J / 0.61 J/jour = 4.9 jours (cela reviendrais à 6 jours avec les nouvelles simulations)

Calcul de la nouvelle puissance d'émission (P_t) avec codeur :

Calcul de E_b (énergie par bit) avec codeur :

$$E_b/N_0 \text{ (dB)} = 4.5 \text{ dB (pour NRZ avec codeur)}$$

$$E_b/N_0 \text{ (linéaire)} = 10^{(E_b/N_0(\text{dB})/10)} = 2.82$$

$$E_b = E_b/N_0 \text{ (linéaire)} \cdot N_0 = 2.82 \cdot 10^{-11} \text{ J/bit}$$

Calcul du débit binaire (R_b) avec codeur :

Le codeur transforme 1 bit en 3 bits, donc le débit binaire est multiplié par 3.

$$R_b = 12 \text{ bits/s} \times 3 = 36 \text{ bits/s}$$

Calcul de la puissance reçue (P_r) avec codeur :

$$P_r = E_b \times R_b = 2.82 \times 10^{-11} \text{ J/bit} \times 36 \text{ bits/s} = 1.015 \times 10^{-9} \text{ W}$$

Calcul de la puissance d'émission (P_t) avec codeur :

$$P_t = P_r \times 10^{(A/10)} = 1.015 \times 10^{-9} \text{ W} \times 10^{(40/10)} = 1.015 \times 10^{-5} \text{ W} = 10.15 \text{ } \mu\text{W}$$

Calcul de l'autonomie de la batterie avec codeur :

$$\text{Consommation quotidienne avec codeur : } P_t \times 24 \text{ heures} \times 3600 \text{ secondes/heure} = 10.15 \times 10^{-6} \text{ W} \times 86400 \text{ s} = 0.88 \text{ J/jour}$$

$$\text{Autonomie de la batterie avec codeur : } \text{Energie totale} / \text{Consommation quotidienne} = 3 \text{ J} / 0.88 \text{ J/jour} = 3.4 \text{ jours}$$

1.

La solution proposée est d'utiliser la forme d'onde NRZ avec un débit binaire de 12 bits/s. La puissance d'émission nécessaire est de 7.07 μW .

2.

Avec cette solution, la batterie tiendra environ 4.9 jours avant d'être déchargée.

3.

Non, la batterie ne tiendra pas plus longtemps pour le même TEB cible avec le code correcteur d'erreurs. La consommation énergétique est plus élevée avec le codeur, ce qui diminue l'autonomie de la batterie.

Environnement 2

De même que pour l'environnement précédent, les améliorations du code ne sont pas pris en compte dans les calculs...

Paramètre	Valeur
Type	Multi Trajet
Seuil TEB	10^{-2}
Nombre de trajet	2
Durée entre les trajets	10 microseconde
Amplitude trajet 1	1
Amplitude trajet 2	0,5
Seuil Eb/N0	< 15dB

On sait que :

$$D = F_e/N$$

$$F_e = 1/T_{\text{symbole}}$$

Avec F_e fréquence d'échantillonnage, N le nombre d'échantillons par bits et T_{symbole} la durée d'un bit (symbole).

On considère ici $F_e = 10$ MHz on pourrait choisir n'importe quelle valeur de F_e , celle-ci est la plus adaptée a nos simulations, car ne nécessite pas un grand nombre d'échantillons par bits pour simuler les débits.

On fait varier le nombre d'échantillons par bits ce qui fait varier le débit, plus N est faible plus le débit est grand. On prend $E_b/N_0 = 15$ dB.

Nous simulons donc avec les paramètres suivants, ceux non spécifiés prennent les valeurs par défaut.

-mess 2000

-form : NRZ et NRZT (car plus performants)

-snrpb : 15

-nbEch : varie en fonction du débit -> $\text{nbEch} = \text{Fe}/D$

-ti : nombre d'échantillons varie en fonction de Fe -> $\text{delta}_t * \text{Fe} = 10\mu\text{s} * 10\text{MHZ}$, ici 100 0.5

En moyenne, nous retrouvons ces valeurs de TEB :

Les simulations sont faites à partir de la classe SimulateurNbEch. Voici les résultats intéressants de 5 à 380 tous les 15 échantillons :

NbEch	Débit bit/s	TEB NRZT	TEB NRZ
5	2000000	0.1255	0.129
20	500000	0.1245	0.1285
35	285714,2857	0.121	0.0975
50	200000	0.132	0.1255
65	153846,1538	0.09	0.07
80	125000	0.1115	0.0865
95	105263,1579	0.1215	0.105
110	90909,09091	0.12	0.078
125	80000	0.1015	0.0495
140	71428,57143	0.0845	0.0315
155	64516,12903	0.0685	0.0185
170	58823,52941	0.0445	0.0125
185	54054,05405	0.033	0.008
200	50000	0.034	0.007
215	46511,62791	0.0315	0.002
230	43478,26087	0.0205	0.0035
245	40816,32653	0.0175	0.0025
260	38461,53846	0.02	5.0E-4
275	36363,63636	0.0165	0.0025
290	34482,75862	0.013	0.002
305	32786,88525	0.014	0.0
320	31250	0.011	5.0E-4
335	29850,74627	0.012	5.0E-4
350	28571,42857	0.01	0.001
365	27397,26027	0.0065	0.001
380	26315,78947	0.0075	5.0E-4

Nous devons obtenir un $TEB < 10^{-2}$

On utilise donc un nbEch autour de 185 en NRZ

Soit un débit de 54kbit/s

Pour un $TEB < 10^{-3}$ avec le même débit, on utilise un codeur pour améliorer les performances, toujours en NRZ.

Cette fois, le débit utilisé : $D = F_e/N/3$. Car 3 bits réels pour représenter un bit utile.

Donc $N = F_e/3 \cdot D \rightarrow 10000000/3 \cdot 54000 \approx 62$

On lance donc une simulation avec un nbEch de 62 et le paramètre codeur d'activé.
Avec 1000 bits

On obtient un $TEB =$ variant entre $1.0E-4$ et $5.0E-4$, donc bien inférieur à 10^{-3}

Sur 10000 bits, 0.0022 en NRZT et 0.0 en NRZ :

```
~/sit213 main !1 ?2 > ./simulateur -mess 10000 -form NRZ -snrpb 15 -ti 100 0.5 -codeur -nbEch 62  
java Simulateur -mess 10000 -form NRZ -snrpb 15 -ti 100 0.5 -codeur -nbEch 62 => TEB : 0.0
```

```
~/sit213 main !1 ?2 > ./simulateur -mess 10000 -form NRZT -snrpb 15 -ti 100 0.5 -codeur -nbEch 62  
java Simulateur -mess 10000 -form NRZT -snrpb 15 -ti 100 0.5 -codeur -nbEch 62 => TEB : 0.0022
```

4.

Nous disposons des résultats de simulations qui montrent les TEB en fonction du nombre d'échantillons par bit (NbEch) pour des formes d'onde NRZ et NRZT.

Selon ces résultats, pour un $TEB < 10^{-2}$, en NRZ, il faut utiliser environ 185 échantillons par bit, ce qui correspond à un débit de 54 kbit/s, pour $F_e = 10\text{MHz}$.

5.

Pour un $TEB < 10^{-3}$ avec un débit identique, l'utilisation d'un codeur est nécessaire.

D'après les résultats, on voit que pour atteindre un $TEB < 10^{-3}$, un codeur doit être utilisé en NRZ. Le débit effectif est réduit par un facteur de 3 car le codeur représente 1 bit utile par 3 bits transmis.

Avec un débit de 54 kbit/s et un codeur activé, le nombre d'échantillons par bit est ajusté à 62, et le TEB obtenu varie entre 0.0 et $5.0E-4$, ce qui est bien inférieur au seuil de 10^{-3} .

Conclusion

En conclusion, les simulations effectuées ont permis de répondre aux exigences du client dans les deux environnements.

Dans l'environnement 1, où la minimisation de la consommation d'énergie était cruciale, la forme d'onde NRZ a été identifiée comme la solution la plus économe, permettant d'atteindre un TEB inférieur à 10^{-3} avec une puissance d'émission de $7,07 \mu W$ et une autonomie de la batterie de 4,9 jours. L'utilisation d'un codeur correcteur d'erreurs a amélioré le TEB, mais au prix d'une consommation énergétique plus élevée, réduisant ainsi l'autonomie à 3,4 jours.

Dans l'environnement 2, pour un canal à trajets multiples, nous avons montré qu'un débit maximal de 54 kbit/s peut être atteint en utilisant la modulation NRZ avec un TEB inférieur à 10^{-2} . L'ajout d'un codeur permet d'atteindre un TEB inférieur à 10^{-3} avec un même débit, répondant ainsi aux exigences de fiabilité accrues du client.