

Compte Rendu de TP1 Réseaux

La transmission en bande de base

REDJRADJ Yacine

22 janvier 2026

Table des matières

1	Introduction et Rappel du sujet	1
2	Analyse du sujet et Modélisation	2
2.1	Données d'entrée et de sortie	2
2.2	Logique des codages implémentés	2
3	Choix techniques et Architecture	2
3.1	Structure du code	2
3.2	Gestion du rendu dynamique	3
3.3	Robustesse et Gestion d'erreurs	3
4	Description de l'implémentation des signaux	3
4.1	Algorithme Miller (Le cas complexe)	3
5	Protocole de Test et Comportement Observé	3
5.1	Scénario nominal	3
5.2	Scénario de redimensionnement	4
5.3	Scénario d'erreur	4
6	Conclusion	4

1 Introduction et Rappel du sujet

L'objectif de ce Travail Pratique est de concevoir une application graphique en Java permettant de simuler la transmission d'un signal numérique. Le programme doit convertir une suite binaire (composée de 0 et de 1) en un signal électrique analogique selon différents protocoles de codage : **NRZ, Manchester, Manchester Différentiel et Miller**.

Le projet se concentre sur la visualisation temporelle des signaux et la gestion de l'affichage dynamique (redimensionnement de la fenêtre).

2 Analyse du sujet et Modélisation

Pour réaliser ce simulateur, nous avons décomposé le problème en deux aspects principaux : l'interface utilisateur (IHM) et le moteur de rendu graphique.

2.1 Données d'entrée et de sortie

- **Entrée** : L'utilisateur fournit une chaîne de caractères binaire (ex : "10110") et sélectionne un algorithme de codage via une liste déroulante.
- **Sortie** : Le programme dessine une courbe représentant la tension ($+V$, $0V$, $-V$) en fonction du temps.

2.2 Logique des codages implémentés

L'implémentation repose sur les définitions théoriques suivantes :

- **NRZ (Non-Return-to-Zero)** : Le niveau de tension dépend directement de la valeur du bit (Haut pour 1, Bas pour 0). C'est le codage le plus simple.
- **Manchester** : Il impose une transition au milieu de chaque bit pour assurer la synchronisation.
- **Manchester Différentiel** : La transition au milieu est toujours présente. L'information est codée par la présence (ou l'absence) de transition au *début* de l'intervalle.
- **Miller** : Ce codage réduit la bande passante en supprimant certaines transitions. Une transition est effectuée au milieu du bit pour un "1", et à la fin du bit pour un "0" (sauf si le bit suivant est un "1").

3 Choix techniques et Architecture

Le projet a été développé en langage **Java** en utilisant la bibliothèque graphique **Swing**. L'architecture respecte une séparation des responsabilités.

3.1 Structure du code

Le programme est divisé en trois classes distinctes :

1. **Main.java** : Point d'entrée de l'application. Elle utilise `SwingUtilities.invokeLater` pour garantir que l'interface graphique est créée dans le thread dédié (EDT), assurant ainsi la stabilité de l'application.
2. **Fenetre.java (L'Interface)** : Cette classe hérite de `JFrame`. Elle gère les composants d'interaction (Boutons, Champs de texte). Nous avons choisi un `BorderLayout` pour placer les contrôles en haut et la zone de dessin au centre.
3. **MyPanel.java (Le Rendu)** : Classe héritant de `JPanel`, où se déroule tout le traitement graphique via la méthode `paintComponent`.

3.2 Gestion du rendu dynamique

Un point technique important est l'adaptabilité du signal à la taille de la fenêtre. Dans la classe `MyPanel`, la largeur d'un bit n'est pas fixe mais calculée dynamiquement :

```
int bitWidth = (width - 2 * MARGIN) / Math.max(chaineBinaire.length(), 1);
```

Cela permet au signal de s'étirer ou de se compresser automatiquement si l'utilisateur redimensionne la fenêtre, garantissant une lisibilité constante.

3.3 Robustesse et Gestion d'erreurs

Pour éviter les plantages, nous avons mis en place une validation stricte des données :

- Utilisation d'une **Expression Régulière (Regex)** `[01]*` pour vérifier que la chaîne ne contient que des 0 et des 1.
- Si l'utilisateur entre une lettre ou un chiffre invalide, une exception `IllegalArgumentException` est levée et capturée pour afficher une boîte de dialogue (`JOptionPane`) explicative.

4 Description de l'implémentation des signaux

Le tracé utilise l'objet `Graphics2D` pour dessiner des lignes entre des coordonnées calculées. L'écran étant un repère où l'axe Y est inversé (0 en haut), nous avons défini trois constantes de hauteur : `yNV` (Haut), `yOV` (Milieu) et `yMinusNV` (Bas).

4.1 Algorithme Miller (Le cas complexe)

Le codage Miller a nécessité une attention particulière car il dépend de l'état précédent. Dans notre boucle de dessin, nous vérifions le bit précédent pour décider de la transition :

Si le bit courant est '0' : on effectue une transition au début du bit uniquement si le bit précédent était aussi un '0'.

Cette logique a été implémentée en accédant à `chaine.charAt(i - 1)` à l'intérieur de la boucle de tracé.

5 Protocole de Test et Comportement Observé

L'application a été testée avec plusieurs séquences binaires types pour valider le comportement.

5.1 Scénario nominal

Test : Saisie de la chaîne "10110" en mode Manchester.

Observation : Le graphique affiche bien 5 périodes de bits. Chaque période comporte une transition verticale exacte en son milieu. Les transitions correspondent aux normes implémentées (Front descendant pour 1, montant pour 0).

5.2 Scénario de redimensionnement

Test : Agrandissement de la fenêtre à la souris.

Observation : Le graphique se redessine instantanément. La grille en arrière-plan et le signal s'étirent proportionnellement pour occuper tout l'espace disponible.

5.3 Scénario d'erreur

Test : Saisie de la chaîne "10210" (contenant un 2).

Observation : Aucune modification du dessin n'est effectuée. Une fenêtre "pop-up" apparaît avec le message : *"Erreur : La chaîne ne doit contenir que des '0' et des '1'".* Le programme ne plante pas.

6 Conclusion

Ce TP a permis de consolider la compréhension des méthodes de transmission en bande de base en les traduisant par des algorithmes informatiques.

La principale difficulté rencontrée a résidé dans la gestion des coordonnées graphiques (inversion de l'axe Y) et dans la logique conditionnelle du codage Miller qui nécessite une mémoire de l'état précédent. L'utilisation du modèle objet de Java (héritage de JPanel) a permis de créer un code propre, modulaire et robuste aux erreurs de saisie.