

Université / U.F.R. des Sciences et Techniques

Compte-Rendu de TP Réseau

Les Codes Cycliques de Redondance

Auteur :

REDJRADJ YACINE

Année Universitaire 2025-2026

Table des matières

1	Rappel du sujet	2
2	Analyse du sujet	2
2.1	Principe mathématique	2
2.2	Fonctionnalités attendues	2
3	Choix techniques effectués	2
3.1	Architecture MVC (Modèle-Vue-Contrôleur)	2
3.2	Gestion de l’affichage des étapes (LogCapture)	3
3.3	Algorithme de division	3
4	Résultats et tests	3
4.1	Génération du CRC	3
4.2	Vérification d’un message valide	4
4.3	Détection d’erreur	5
5	Conclusion	5

1 Rappel du sujet

L'objectif de ce TP est d'implanter un programme en langage JAVA permettant de manipuler les Codes Cycliques de Redondance (CRC). Le programme doit remplir deux fonctions principales

- **Le calcul d'un CRC (Émetteur)** : À partir d'un mot binaire et d'un polynôme générateur, calculer le reste de la division binaire et former le mot complet à transmettre.
- **La vérification d'un message (Récepteur)** : À partir d'un message reçu, vérifier s'il contient des erreurs de transmission en utilisant le même polynôme générateur.

Une contrainte importante est que le programme doit permettre de visualiser les étapes intermédiaires du calcul (la division polynomiale).

2 Analyse du sujet

Le contrôle de redondance cyclique repose sur l'arithmétique polynomiale modulo 2.

2.1 Principe mathématique

Pour générer le CRC :

1. On considère le message comme un polynôme $M(x)$.
2. On multiplie ce polynôme par x^n (où n est le degré du polynôme générateur $G(x)$), ce qui revient à ajouter n zéros à la fin du message.
3. On effectue la division modulo 2 de ce message augmenté par $G(x)$. L'opération de base est le OU Exclusif (XOR).
4. Le reste de cette division constitue le CRC (FCS).

Pour la vérification, le récepteur divise le message complet reçu par le même polynôme $G(x)$. Si le reste est nul, le message est considéré comme valide. Sinon, il est erroné.

2.2 Fonctionnalités attendues

Le programme doit offrir une interface permettant à l'utilisateur de saisir des suites de bits arbitraires et de choisir son polynôme générateur[cite : 26]. Il doit traiter les chaînes de caractères pour simuler les opérations binaires.

3 Choix techniques effectués

Pour répondre aux exigences du TP, j'ai effectué les choix de conception suivants :

3.1 Architecture MVC (Modèle-Vue-Contrôleur)

J'ai structuré mon code en séparant la logique métier de l'interface graphique :

- **Package Models (CrcCalcul.java)** : Contient la logique pure. La méthode `divisionBinaire` effectue le XOR bit à bit. C'est ici que l'algorithme mathématique réside.

- **Package Views (CrcView, etc.)** : Gère l'interface graphique utilisant la bibliothèque **Java Swing** ('JFrame', 'JPanel'). Cela offre une interaction conviviale pour l'utilisateur.
- **Package Controllers** : Fait le lien entre les actions de l'utilisateur et le calcul.

3.2 Gestion de l'affichage des étapes (LogCapture)

Le sujet demandant explicitement d'afficher les étapes du calcul[cite : 18], j'ai implémenté une classe utilitaire nommée **LogCapture**. Plutôt que de modifier l'algorithme de calcul pour qu'il retourne une énorme chaîne de caractères, j'ai utilisé une redirection des flux de sortie. La classe redirige temporairement **System.out** vers un **ByteArrayOutputStream**, capture tout ce qui est affiché dans la console lors du calcul, puis l'affiche dans le **JTextArea** de l'interface graphique.

3.3 Algorithme de division

L'opération XOR est simulée sur des chaînes de caractères ('String'). J'utilise la classe 'StringBuilder' pour manipuler efficacement les chaînes (mutable) lors des décalages successifs de la division.

4 Résultats et tests

Voici les tests effectués pour valider le bon fonctionnement du programme. Nous utilisons le polynôme générateur $x^4 + x^2 + x$ (binaire : 10110).

4.1 Génération du CRC

Nous souhaitons transmettre le message 1111011101. Comme le montre la Figure 1, le programme effectue la division et trouve le CRC 1100. Le message final est bien 11110111011100. On peut voir les restes intermédiaires dans la zone de détails.

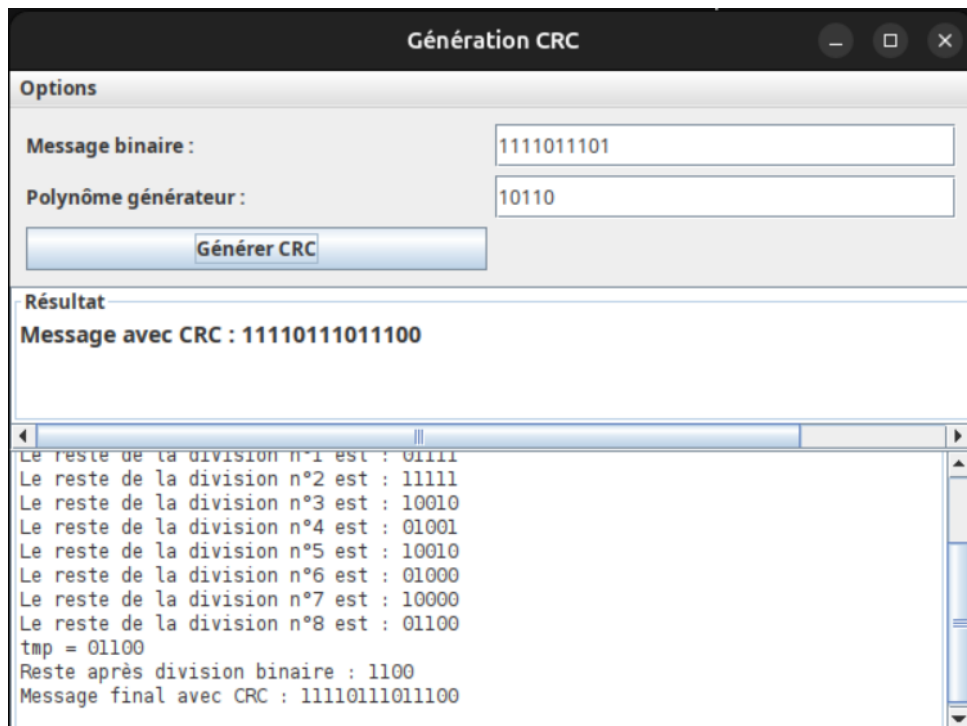


FIGURE 1 – Génération du CRC pour le message 1111011101

4.2 Vérification d'un message valide

Nous testons la vérification avec le message complet généré précédemment : 11110111011100. Le résultat (Figure 2) indique que le message est **VALIDE**, ce qui confirme que la division donne un reste nul.

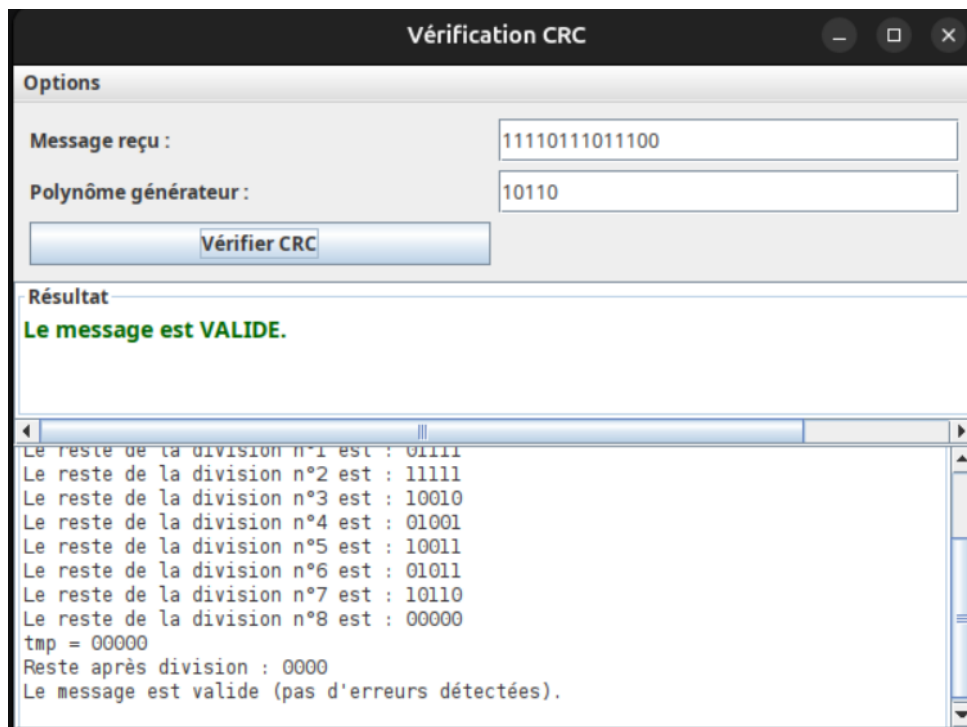


FIGURE 2 – Vérification positive d'un message correctement transmis

4.3 Détection d'erreur

Pour tester la robustesse, nous modifions le dernier bit du message : 11110111011101. Le programme détecte bien l'anomalie et affiche que le message contient des erreurs (Figure 3), car le reste de la division n'est plus nul.

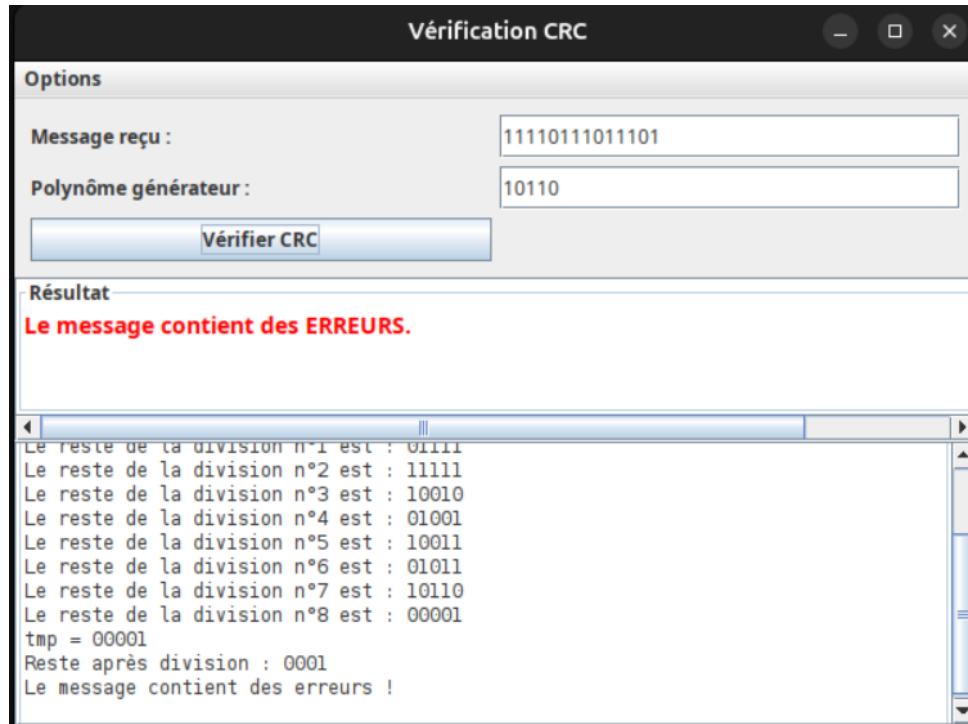


FIGURE 3 – Détection d'une erreur de transmission (1 bit modifié)

5 Conclusion

Ce TP m'a permis de comprendre en profondeur le mécanisme des Codes Cycliques de Redondance. L'implémentation en Java m'a confronté à la manipulation de bas niveau des bits via des chaînes de caractères.

Difficultés rencontrées : La principale difficulté a été de gérer correctement l'alignement lors de la division binaire (savoir quand "descendre" le bit suivant) et d'assurer l'affichage des étapes intermédiaires dans l'interface graphique sans alourdir le code du modèle. L'utilisation de la classe `LogCapture` a élégamment résolu ce problème.

Perspectives : Le programme est fonctionnel pour n'importe quel polynôme saisi manuellement. Une évolution possible serait d'intégrer des polynômes standards prédéfinis (comme CRC-16 ou CRC-32) dans un menu déroulant pour faciliter l'utilisation, et d'ajouter une validation plus stricte des entrées utilisateurs (empêcher la saisie de caractères autres que 0 et 1).