TD2 - Calculs de code de contrôle d'erreurs

Avec corrections

Exercice 1 (Bit de parité)**.**
Pour commencer, nous considérons un code très simple. À la fin du message qu'on veut envoyer nous rajoutons un bit qui correspond à la somme modulo 2 de tous les bits du message. Si, par exemple, on veut envoyer le message 1101001, alors on envoie le message 1101001$\underline{0}$. Ainsi le receveur du message pourra tester si le message est correct en réalisant le calcul : 1+1+0+1+0+0+1=$\underline{0}$ mod 2. Si le premier bit change durant le transfert et que le message est devenu 0101001$\underline{0}$ alors la somme (0+1+0+1+0+0+1=1 mod 2) ne correspond plus au bit de parité, on sait donc qu'il y a une erreur.

1. Les messages suivant respectent-ils le code :
   10010111011101011110 10 et 11110010100010101110 11.

   le premier, oui, le deuxième, non

2. Quelles critiques pouvez-vous émettre sur ce code de vérification ?

   S'il y a deux inversions de bits, on ne peut pas le détecter.

Exercice 2 (Double-bit de parité)**.**
On se propose d'utiliser l'idée de l'exercice précédent de façon un peu plus élaborée. On commence par découper notre message en mots de taille fixe. Par exemple, disons qu'on veut envoyer le message 011101010. On décide de découper le message en mots de taille 3, on obtient 011.101.010. On place ce message dans un tableau et on calcul le bit de parité pour chaque ligne et chaque colonne comme dans le tableau suivant :

0	1	1	0
1	0	1	0
0	1	0	1
1	0	0	

On envoie alors le message : 011101010<u>001100</u>.

1. Déterminer la valeur d'un tel code pour le message 1010111010111100 en utilisant un découpage en mots de taille 4.

1	*0*	*1*	*0*	*0*
1	*1*	*1*	*0*	*1*
1	*0*	*1*	*1*	*1*
1	*1*	*0*	*0*	*0*
0	*0*	*1*	*1*	

2. Montrer que ce code permet non seulement de détecter mais aussi de corriger une erreur unique.

   S'il y a une inversion de bit dans le message, on verra quelle ligne et quelle colonne a une erreur de parité, ce qui situe la position de l'inversion

3. Lorsqu'il y a deux erreurs ce code permet-il de les corriger ?

   Non, car on a alors plusieurs lignes ou plusieurs colonnes avec erreur, et l'on ne sait alors pas précisément où corriger.

Exercice 3 (CRC)**.**
Le code CRC va utiliser une clef et réaliser un calcul pour obtenir un code similaire à celui du bit de parité mais plus robuste, c'est en fait une division de polynôme, qu'on effectue comme un genre de division posée en binaire avec xor. Regardons un exemple sur 4 bits, la clef choisie est 10011 et le message est 1001011, on ajoute au message 0 zéros pour les 4 bits qui composeront le CRC :

	1	0	0	1	0	1	1	0	0	0	0
xor	1	0	0	1	1	⋮	⋮				
		0	0	0	1	1	⋮				
xor		0	0	0	0	0	⋮				
			0	0	1	1	1				
xor			0	0	0	0	0				
				0	1	1	1				
xor				0	0	0	0	0			
					1	1	1	0			
xor					1	0	0	1	1		
						1	1	1	1		
xor						1	0	0	1	1	
							1	1	0	1	
xor							1	0	0	1	1
								1	**0**	**0**	**1**

On envoie alors le message 10010111001. À nouveau, à la réception, il nous suffira de faire le même calcul pour vérifier si le message n'a pas été modifié. En fait, on peut aussi mettre le message entier en haut (donc avec les 4 chiffres du CRC en haut à droite à la place des zéros), et constater que l'on retombe à zéro.

1. Trouver le code CRC utilisant la clef 11010 correspondant au message 0010111011.

   C'est 0100

2. Le message 0100101101 est-il correct sachant qu'on utilise un code CRC 3 bits de clef 1010 ?

   Non, on trouve comme CRC 100 au lieu de 101