DEPARTAMENTO DE ELECTRÓNICA Y AUTOMÁTICA
FACULTAD DE INGENIERÍA – UNIVERSIDAD NACIONAL DE SAN JUAN

Informe de Practica Nº2

CONTROL DE ERROR

Asignatura: TELECOMUNICACIONES II
Ingeniería Electrónica

Autores:
Avila Juan Agustín - Registro 26076
Sanguedolce, Mauricio - Registro 26577
Villafañe, Mario – Registro 27857

2º Semestre
Año 2020

1 Introducción

Se va a desarrollar a lo largo de esta práctica, la implementación en software de simulación, de las técnicas de codificación y decodificación de códigos cíclicos estudiados durante el cursado de la asignatura. Para ello utilizaremos el software de *MatLab/Simulink*

2 Control de error.

2.1 Realice la implementación de hardware

Mediante una herramienta de simulación, (por ejemplo simulink) realice la implementación de hardware correspondiente al polinomio (codificador y decodificador) que en cada caso requiera menor cantidad de componentes y emule el funcionamiento de un canal en el que se pueda introducir una cantidad de errores correspondientes a los puntos 4 y 5 y vea el funcionamiento del decodificador

Para esto, se utilizó

K=3 g1(x) : $X^{11} + X^9 + X^8 + X^7 + X^4 + X^2 + X + 1$

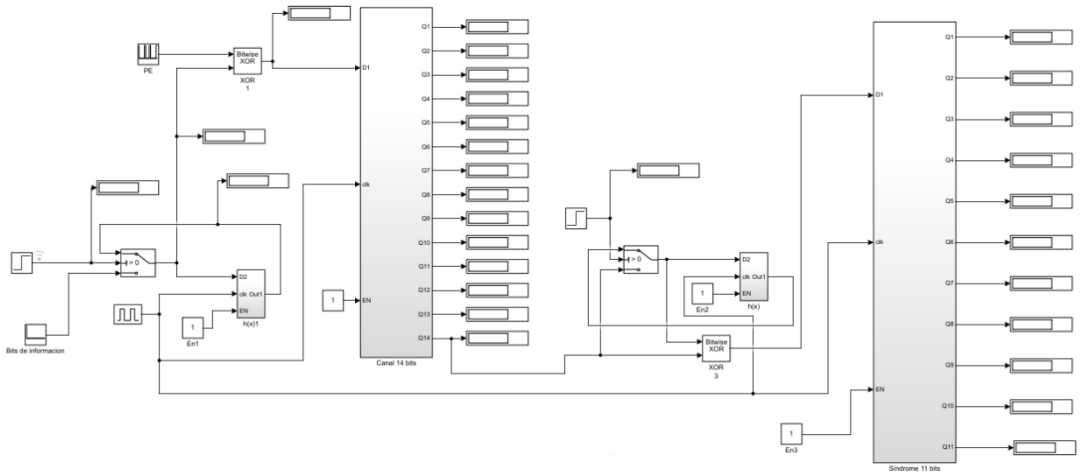

Ilustración 1 - Síntesis del circuito en simulink

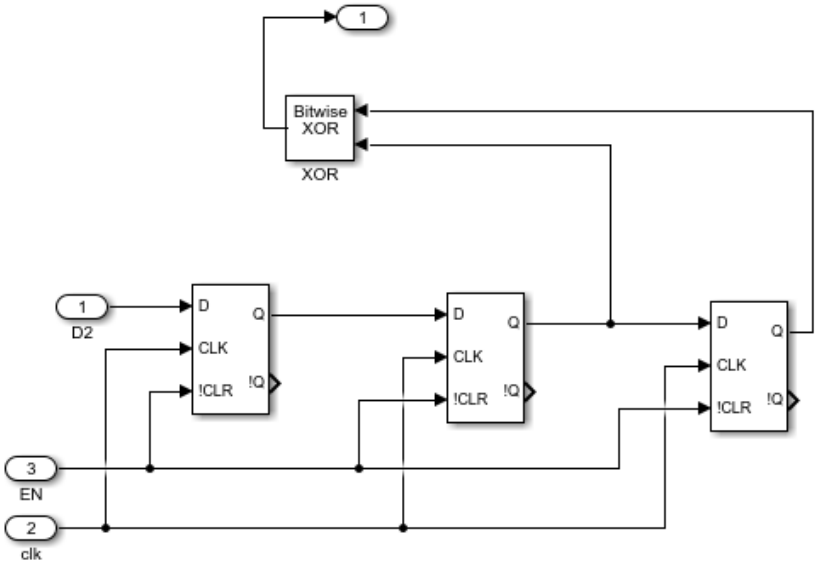

Ilustración 2 - Detalle del circuito generado con flip flops

2.2 Obtenga la distancia mínima y capacidad de corrección del código.

La distancia mínima del código es la menor diferencia entre dos palabras validas del código. Para este código en particular donde los bits de información son igual a 3 y el grado del polinomio generador es igual a 11 la distancia mínima será de 8.

Por lo tanto, se observan las siguientes propiedades del código:

Distancia Mínima 8

Capacidad como detector $Dmin - 1 = 7$

Capacidad de corrección de errores $\frac{Dmin-2}{2} = 3$

2.3 Encuentre las matrices generadora y chequeadora.

Para generar las matrices, se utilizó el siguiente código en matlab:

```
tb=0.06;
n=14; k=3;
G=[1 0 1 1 1 0 0 1 0 1 1 1];
P=zeros(3,14);
for i=0:2
    X=[1 zeros(1,(n-k+i))]
    [q,r]=deconv(X,G);
    r=abs(r)
    P(3-i,:)=[zeros(1, 2-i) r+X];
end
P=P(:,4:size(P,2)) %matriz generadora

H=[P' eye(n-k)] %Matriz chequeadora

%% polinomio chequeador
X=[1 zeros(1,13) 1];
[q,r]=deconv(X,G);
h=abs(q)    %polinomio chequeador
```

Y los resultados obtenidos fueron:

P =

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

H =

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

2.4 Genere errores en las palabras transmitidas dentro de la capacidad del código y verifique los síndromes obtenidos.

Teniendo en cuenta que la capacidad de corrección de errores era de 3, se realizan pruebas introduciendo entre 1 y 3 errores:

2.4.1 Para 1 error

- Bits de info: 0 0 0
- Pe: 0 0 0 0 1 0 0 0 0 0 0 0 0
- Canal: 0 0 0 0 1 0 0 0 0 0 0 0 0
- Síndrome: 0 0 0 0 0 0 0 0 0 1 0

2.4.2 Para 2 errores

- Bits de info: 0 0 0
- Pe: 0 0 0 0 0 0 0 0 0 0 0 0 1 1
- Canal: 0 0 0 0 0 0 0 0 0 0 0 0 1 1
- Síndrome: 1 1 0 0 0 0 0 0 0 0 0

2.4.3 Para 3 errores

- Bits de info: 0 0 0
- Pe: 0 0 0 0 1 0 0 0 1 0 0 0 0 1

- Canal: 0 0 0 0 1 0 0 0 1 0 0 0 0 1
- Sindrome: 1 0 0 0 0 1 0 0 0 1 0

Para el primer caso en el cual se presenta un solo error, el síndrome nos devuelve un vector que es igual a la quinta columna de la matriz H. Esto dice que el error es simple y que se encuentra en el quinto bit, tal cual se ve en el Vector PE

Para el segundo caso el síndrome es igual a la combinación lineal de las últimas dos columnas. Esto indica la presencia de un error doble en los últimos bits del mensaje.

Por último el sistema nos devuelve un síndrome que es igual a la combinación lineal de la quinta, novena y última columna de H. Esto representa a un error triple en dichos bits del mensaje. Con esto se comprueba que el código puede detectar y corregir hasta tres errores

2.5 Genere errores en las palabras transmitidas fuera de la capacidad del código y compruebe que la decodificación se puede interpretar como otra palabra valida.

Teniendo en cuenta que la capacidad de corrección del circuito es de 3 bits, se realiza una prueba con 4 bits errores para comprobar que el código falla:

2.5.1 Para 4 errores

- Bits de info: 0 0 0
- Pe: 1 0 0 0 1 0 0 0 1 0 0 0 0 1
- Canal: 1 0 0 0 1 0 0 0 1 0 0 0 0 1
- Sindrome: 1 0 0 0 0 1 0 0 0 1 0

3 Secuencia Pseudoaleatoria

El siguiente sistema tiene como propósito generar para un mismo conjunto de bits de información, diferentes mensajes respetando una misma longitud. La longitud del mensaje vendrá determinada por la siguiente formula.

$$2^k - 1 = 2^3 - 1 = 7$$

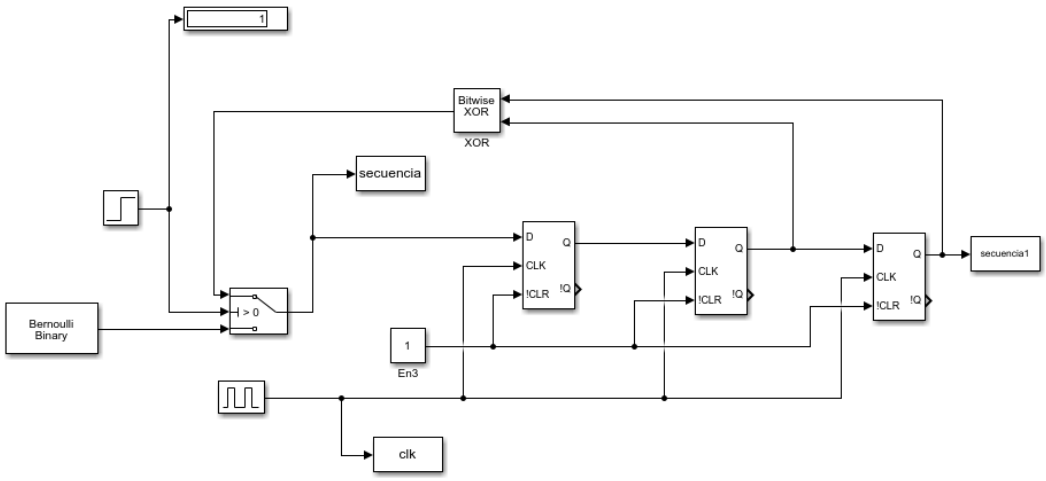

Ilustración 3 - Esquema del hardware implementado en simulink

A continuación, veremos diferentes ejemplos para varias ternas de bits de información, inclusive repitiéndola y generando diferentes palabras de información.

```
>> secuencia

   0   1   1

>> secuencia1

   1   1   1   0   0   0   0

>> secuencia

   0   0   0

>> secuencia1

   0   1   1   0   0   1   1

>> secuencia

   0   0   0

>> secuencia1

   0   1   1   0   0   1   1

>> secuencia

   0   0   0

>> secuencia1

   0   1   1   0   0   1   1

>> secuencia

   1   1   1

>> secuencia1

   0   1   1   1   1   1   1

>> secuencia

   1   1   1

>> secuencia1

   0   0   0   1   1   0   0
```

4 Demostraciones de las propiedades de los códigos CRC-16 y CRC-CCIT

Estos códigos son capaces de detectar:

-Todos los errores con un número impar de bits

La paridad par es un caso particular de los códigos CRC donde el polinomio generador es x+1. Dado que estos polinomios tienen como factor x+1, lo que se logra es un bit adicional, que configura una

paridad del código resultante de otro factor del polinomio. Dado que, si hay un número impar de errores, la paridad cambia, entonces estos códigos detectan todos los errores impares.

--Todos los errores a ráfagas de longitud 16 o menor

A una ráfaga de longitud r puede ser representada por el polinomio xi*(xr-1 +…+1), donde los coeficientes entre r-1 y 0 pueden ser 1 o 0. Una ráfaga de 16 bits seria xi*(x15 +…+1)
Dado que, CRC-16 y CRC-CCITT no son factor de xi, y (x15 +…+1) es de menor orden que estos polinomios y por lo tanto no pueden ser factor de los mismos, una ráfaga de 16 bits será detectada en el 100% de los casos.

-99,997% de los errores a ráfagas de 17 bits

En el casado de una ráfaga de 17 bits, esta sería representada por xi*(x16 +…+1).
Dado que CRC-16 y CRC-CCITT son factor de ese polinomio solo en el caso en que (x16 +…+1) sea igual a estos polinomios (al polinomio usado en esa codificación) y que estas existen 215 posibles polinomios-ráfaga de 17 bits, la probabilidad de que estas ráfagas sean detectadas es de:

$$\left(1-\frac{1}{2^{15}}\right)*100\% = 99.9969.\,\%. \cong 99{,}997\%$$

-99,998% de los errores a ráfagas de 18 o más bits.
En el casado de una ráfaga de 18 bits o más bits, esta sería representada por B(x)=xi*(xn +…+1)
Si los polinomios usados en la codificación son factores de (xn +…+1), entonces estos pueden ser expresados de la siguiente forma: g(x)*p(x), siendo g(x) el polinomio usado en la codificación.
P(x) no es cualquier polinomio, sino que debe cumplir con ciertas restricciones.
Dado que P(x) es de orden n-r (r es el orden de g(x)) el termino pn-r debe ser 1 y dado que el termino independiente del polinomio (xn +…+1) no es nulo, p0 también debe ser 1.
Por lo tanto, existen 2n-r-2 polinomios p(x) que cumplen con estas reglas y, por lo tanto, 2n-r-2 polinomios b(x) que poseen como factor a g(x).
Existen 2n-2 posibles polinomios b(x), por lo tanto, la probabilidad de que sea detectada una ráfaga de 18 bits o más es:

$$\left(1-\frac{2^{n-r-2}}{2^{n-2}}\right)*100\% = \left(1-\frac{1}{2^{r}}\right)*100\%$$

En este caso, donde r=16:

$$\left(1-\frac{1}{2^{16}}\right)*100\% = 99{,}9984\ldots\% \cong 99{,}998\%$$

5 Conclusión

A través de Simulink, se ha comprobado la eficiencia del polinomio codificador y decodificador, donde se ha encontrado las matrices generadora y chequeadora, la distancia mínima, y verificado la correcta capacidad de detección y corrección de errores a través de diferentes pruebas realizadas a lo largo del presente práctico. Por otro lado, se implementó en Simulink la simulación de un hardware generador de secuencias Pseudoaleatoria, la cual se comprobó con diferentes secuencias de entrada, que efectivamente generaba diferentes palabras de información, comprobando finalmente, los conocimientos adquiridos en el cursado de la materia.