

**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**

# Introducción

¿?????

# Control de error.

## 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) : X11 + X9 + X8 + X7 + X4 + X2 + X + 1

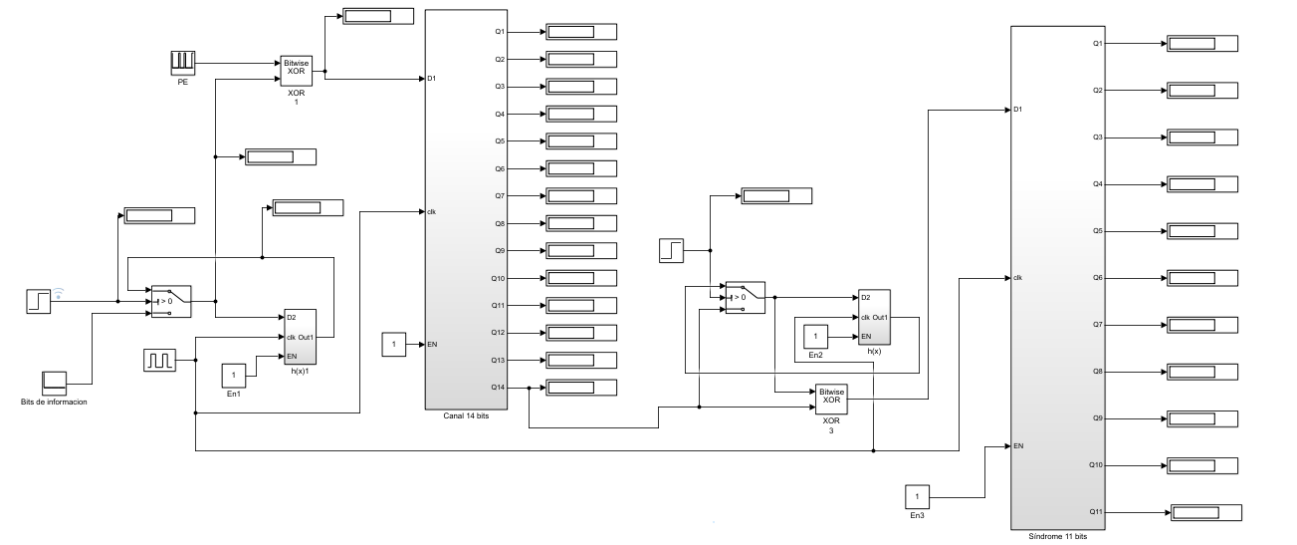


Ilustración 1 - Síntesis del circuito en simulink

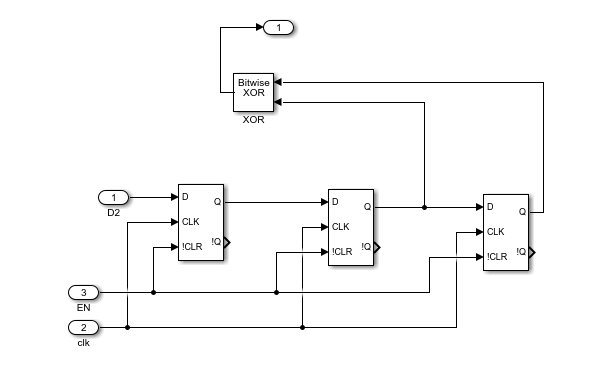


Ilustración 2 - Detalle del circuito generado con flip flops

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

La distancia mínima del código es igual a la distancia del vector nulo al generador???.

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

Distancia Mínima 8

Capacidad como detector

Capacidad de corrección de errores

## 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

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

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

### Para 1 error

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

### 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

### 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

Habria que analizar en cada caso el síndrome y dar donde están los bits de error

## 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 correccion del circuito es de 3 bits, se realiza una prueba con 4 bits errores para comprobar que el codigo falla:

### 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

# Secuencia Pseudoaleatoria

Explicar esto?

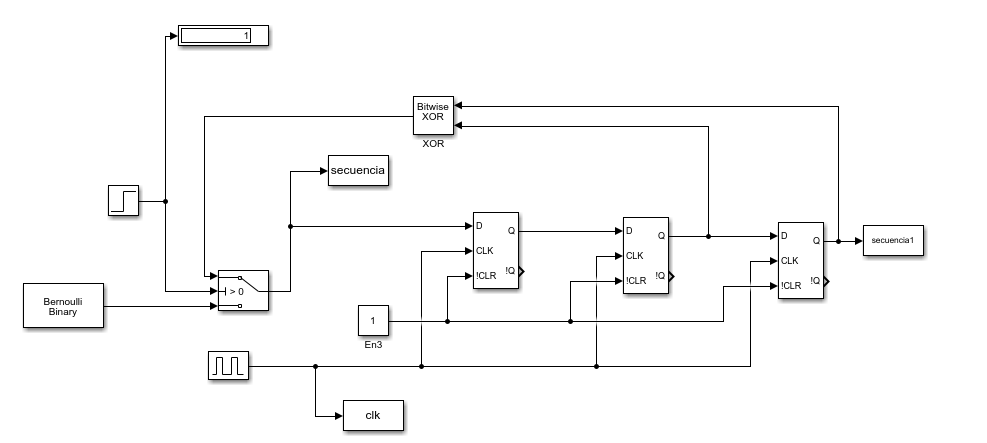


Ilustración 3 - Esquema del hardware implementado en simulink

>> 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