.UBAfiuba
FACULTAD DE INGENIERÍA

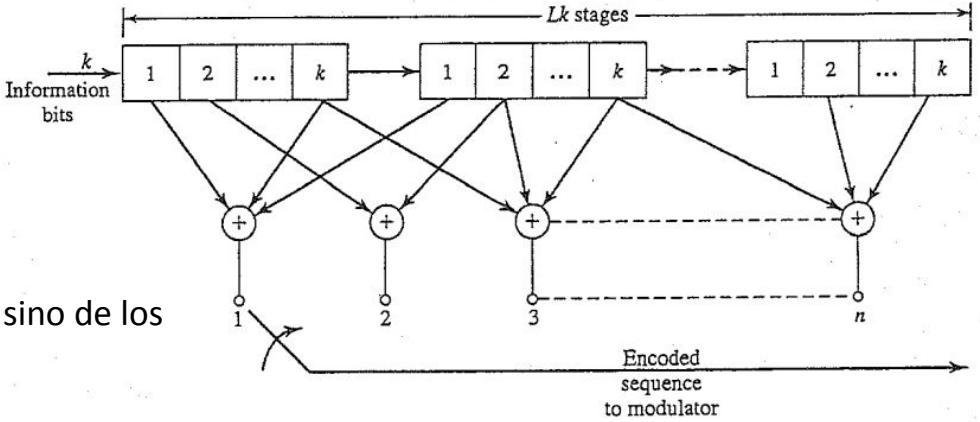
Códigos Convolucionales

Sofía Bertinat

Trabajo final, Procesamiento digital de señales

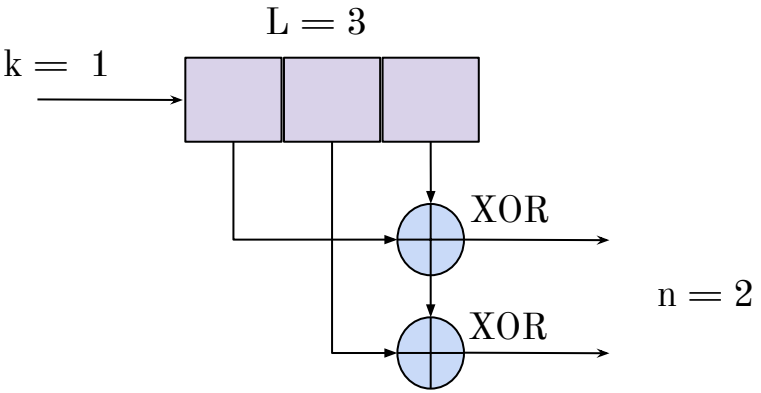
Maestría en Sistemas embebidos

Códigos convolucionales



- Su finalidad es la obtención de redundancia de bits.
- Los bits de salida dependen no solo del bit actuales sino de los bits anteriores, requiriendo de una memoria.
- Ingresan k bits por vez.
- Cada bloque de k bits es mapeado en un bloque de n bits.
- El encoder convolucional consiste en un shift register de k*L etapas, donde L es la restricción de longitud del código.
- Después de que los k bits hayan entrado en el registro de desplazamiento, se calculan n combinaciones lineales de los contenidos del registro de desplazamiento, y se utilizan para generar la forma de onda codificada.

Encoder convolucionales



- Se asume que antes del ingreso del primer bit de información el estado de los bits de memoria es el de todos ceros.
- El ratio del código, R = k/n
- Una forma de describir cómo los bits de salida del encoder dependen del contenido del shift register es mediante n vectores g1 … gn, conocidos como secuencias generadoras.

$$g1 = 1\ 0\ 1$$
$$g2 = 1\ 1\ 1$$

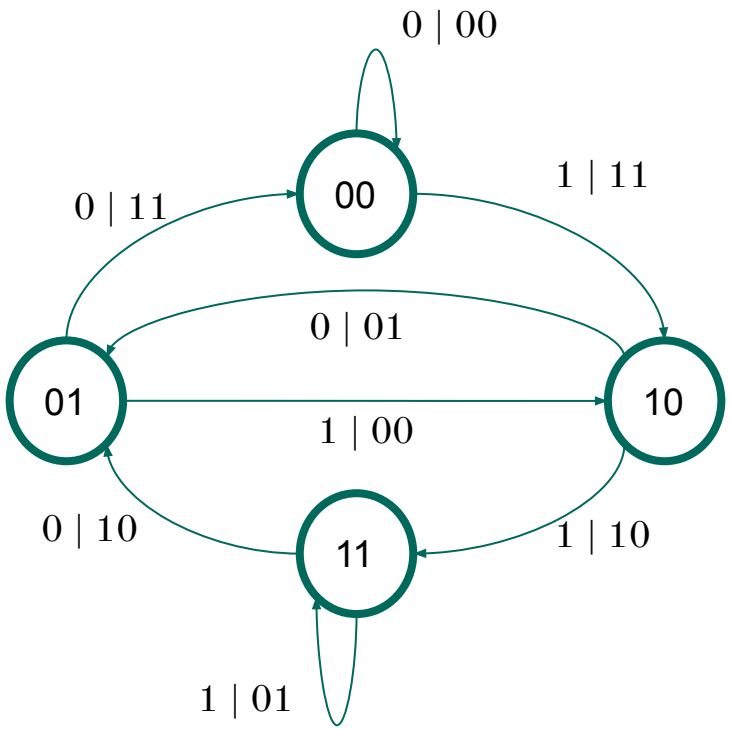
- También se puede representar en forma polinómica, asociando un polinomio generador a cada salida. Permite obtener la salida multiplicando la entrada en forma polinómica por cada uno de los polinomios generadores.

$$g1(x) = 1 + x\text{^}2$$
$$g2\ (x) = 1 + x + x\text{^}2$$

- Se tiene en memoria (L-1)* k bits, la salida depende de esta cantidad de bits anteriores.

Representación diagrama de estados

- El encoder se comporta como una máquina de estados finitos donde la entrada condiciona la salida como el próximo estado de la misma.
- Cada estado es representado por las posibles combinaciones de bits anteriores.
- La cantidad de transiciones de estados (líneas) que salen de cada estado, se determina por el número de entradas posibles al codificador en ese estado.
- Máquina de 2^((L-1)*k) estados



Caso práctico

Se realiza el código convolucional de la figura, donde L = 7, R = ½, teniendo los siguientes polinomios generadores:

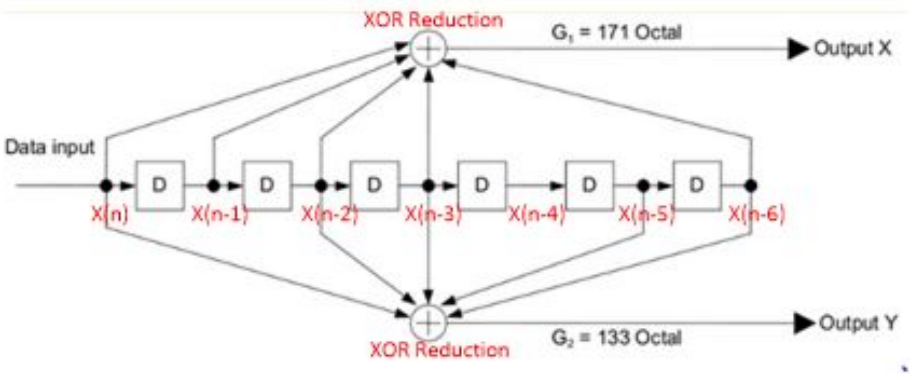
g1 = 0111 1001 = 171 OCTAL

g2 = 0101 1011 = 133 OCTAL

Reset asincrónico activo bajo

Entrada: AXI stream ancho de word 16 bits

Salida: AXI stream ancho de word 32 bits

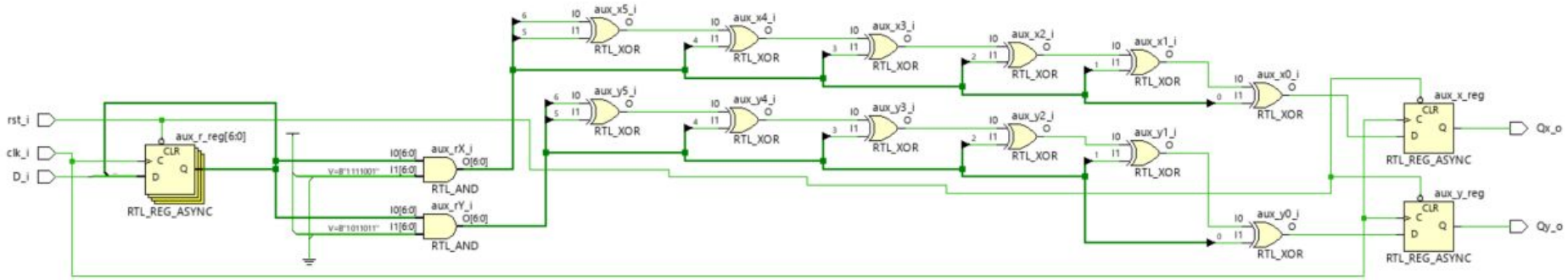


Se generan los vectores de prueba mediante python, utilizando la herramienta

Implementación

```
14  architecture encoder_arq of encoder is
15      constant C_PGENX    : std_logic_vector(6 downto 0) := ("1111001");
16      constant C_PGENY    : std_logic_vector(6 downto 0) := ("1011011");
17      signal aux_r: std_logic_vector(6 downto 0) := (others => '0');
18      signal aux_rX: std_logic_vector(6 downto 0) := (others => '0');
19      signal aux_rY: std_logic_vector(6 downto 0) := (others => '0');
20      signal aux_x: std_logic := '0';
21      signal aux_y: std_logic := '0';
22  begin
23
24      aux_rX <= aux_r and C_PGENX;
25      aux_rY <= aux_r and C_PGENY;
26
27      process(clk_i,rst_i)
28      begin
29         if rst_i = '0' then
30             aux_r <= (others => '0');
31             aux_x <= '0';
32             aux_y <= '0';
33         elsif rising_edge(clk_i) then
34              aux_r(5 downto 0) <= aux_r(6 downto 1);
35              aux_r(6) <= D_i;
36              aux_x <= aux_rX(6) xor aux_rX(5) xor aux_rX(4) xor aux_rX(3) xor aux_rX(2) xor aux_rX(1) xor aux_rX(0);
37              aux_y <= aux_rY(6) xor aux_rY(5) xor aux_rY(4) xor aux_rY(3) xor aux_rY(2) xor aux_rY(1) xor aux_rY(0);
38          end if;
39      end process;
40
41      Qx_o <= aux_x;
42      Qy_o <= aux_y;
43
44  end encoder_arq;
```

Esquemático



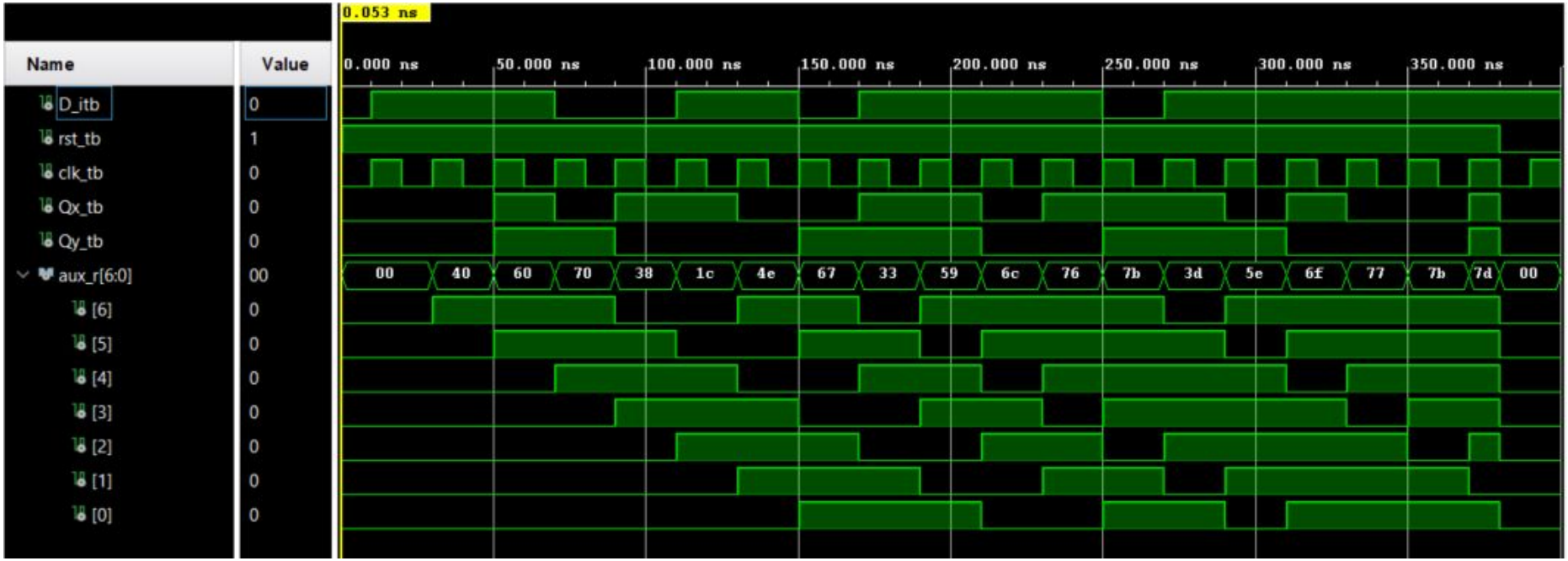
Commpy

testEncoder > test_encoder.py > ...

```python
import numpy as np
import commpy.channelcoding.convcode as cc
import commpy.modulation as modulation

#Simular el codigo VHDL y comparar la salida obtenida con resultado en python

N = 16
message_bits = np.random.randint(0, 2, N) # mensaje, Stream of bits to be convolutionally encoded.
print(message_bits)
np.savetxt('message_bits.txt', message_bits, fmt="%5i")

generator_matrix = np.array([[0o5, 0o7]]) # G(D) = [1+D^2, 1+D+D^2],  Generator matrix G(D) of the convolutional encoder
memory = np.array([2]) # Number of memory elements per input of the convolutional encoder.

trellis = cc.Trellis(memory, generator_matrix) # Trellis structure, classTrellis(memory, g_matrix)

coded_bits = cc.conv_encode(message_bits, trellis)

print(coded_bits)
np.savetxt('coded_bits.txt', coded_bits, fmt="%5i")
```

```
(studysession) PS C:\Users\sofi_\Desktop\MSE\PDS\TPF\testEncoder> python test_encoder.py
[1 0 1 1 0 1 1 1 0 0 1 0 0 0 0 0]
[1 1 0 1 0 0 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0]
```

Simulación


0.053 ns
Name
Value
0.000 ns
50.000 ns
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
350.000 ns
D_itb
0
rst_tb
1
clk_tb
0
Qx_tb
0
Qy_tb
0
aux_r[6:0]
00
00
40
60
70
38
1c
4e
67
33
59
6c
76
7b
3d
5e
6f
77
7b
7d
00
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0

Bibliografía

- Communication Systems Engineering, Second Edition
- https://commpy.readthedocs.io/en/latest/channelcoding.html#convolutional-codes
- https://github.com/veeresht/CommPy
- https://github.com/veeresht/CommPy/blob/master/commpy/channelcoding/README.md