



**Instituto Politécnico Nacional**



*Unidad Profesional Interdisciplinaria en Ingeniería y  
Tecnologías Avanzadas*

Teoría de la información

**Práctica 5**

Implementación en hardware de un codificador y decodificador de canal

**Profesor**

Jorge Rojas Beltrán

**Alumno**

Alvarado Balbuena Jorge Anselmo

**Grupo**

2TM4

27/05/2019

# Índice

<b>1. Objetivo</b>	<b>4</b>
<b>2. Descripción</b>	<b>4</b>
<b>3. Diagrama de circuito de implementación</b>	<b>5</b>
3.1. Timers utilizados . . . . .	5
<b>4. Código</b>	<b>6</b>
<b>5. Circuito lógico</b>	<b>13</b>
5.1. Coder . . . . .	13
5.1.1. Circuito lógico . . . . .	14
5.2. Decoder . . . . .	15
5.2.1. Circuito lógico síndrome . . . . .	15
5.2.2. Circuito lógico decodificador . . . . .	17
<b>6. Palabras dato y código</b>	<b>18</b>
<b>7. Síndrome vs patrón de error</b>	<b>19</b>
<b>8. Resultados simulación</b>	<b>20</b>
<b>9. Listado de asignación de terminales</b>	<b>23</b>
<b>10. Diagramas de flujo</b>	<b>24</b>
10.1. Codificador . . . . .	25
10.2. Decodificador . . . . .	26
10.3. Utilización de dispositivo . . . . .	27
<b>11. Conclusiones</b>	<b>27</b>

# Índice de figuras

1. Diagrama a bloques general del codificador a implementar. . . . .	4
2. Circuito implementado. . . . .	5
3. Timers. . . . .	5
4. Circuito lógico codificador. . . . .	14
5. Circuito lógico síndrome. . . . .	15
6. Circuito lógico decodificador. . . . .	17
7. Palabras dato y código. . . . .	18
8. Síndrome vs patrón de error. . . . .	19

---

9.	Esquema de simulación. . . . .	20
10.	Pulsos generados de la palabra código y la palabra código con error. . . . .	21
11.	Pulsos generados de la palabra código, palabra código con error, síndrome y palabra dato. . . . .	22
12.	Terminales. . . . .	23
13.	Diagrama principal. . . . .	24
14.	Codificador. . . . .	25
15.	Decodificador. . . . .	26
16.	Utilización. . . . .	27

## 1. Objetivo

Implementación en hardware de un codificador de canal de bloque mediante la utilización de dispositivos lógicos programables.

## 2. Descripción

A partir de una matriz generadora perteneciente a un CBS( $n, k$ ), se implementarán en dispositivo programable tanto la parte codificadora (coder) como la parte decodificadora (decoder) de dicho codificador, de acuerdo con el siguiente diagrama.

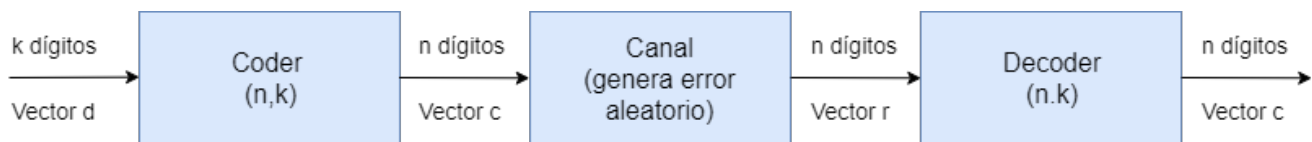


Figura 1: Daigram a bloques general del codificador a implementar.

### 3. Diagrama de circuito de implementación

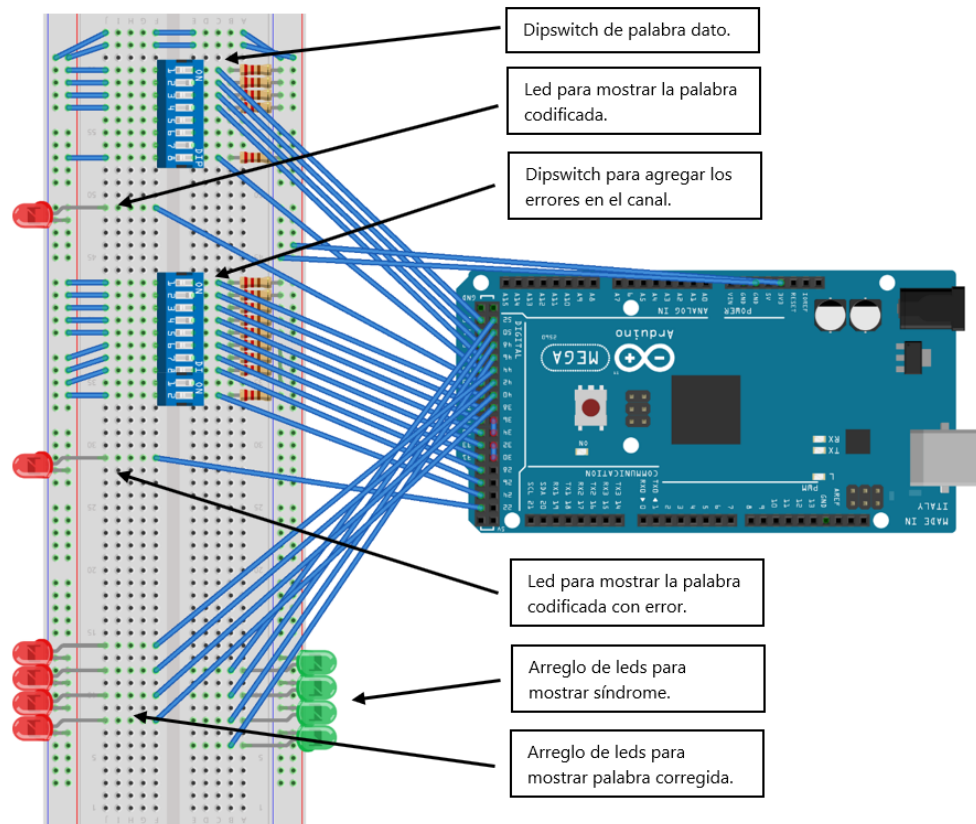


Figura 2: Circuito implementado.

#### 3.1. Timers utilizados

Timer	Frecuencia
Inicio de proceso.	2 Hz
Transmisión de fuente a canal.	1 Hz
Transmisión de canal con error a decodificador.	1 Hz
Corrección de error en el decodificador.	1 Hz

Figura 3: Timers.

## 4. Código

```
1 // Libreria para operaciones logicas
2 #include <iso646.h>
3 // Pins para palabra dato, control de fuente y muestra
4 // de transmision
5 #define D1 53
6 #define D2 51
7 #define D3 49
8 #define D4 47
9
10 #define CD 45
11
12 #define C1 43
13 // Pins para error, control de canal y muestra
14 // de transmision
15 #define E1 41
16 #define E2 39
17 #define E3 37
18 #define E4 35
19 #define E5 33
20 #define E6 31
21 #define E7 29
22 #define E8 27
23
24 #define CE 25
25
26 #define C2 23
27 // Pins para sindrome
28 #define S1 52
29 #define S2 50
30 #define S3 48
31 #define S4 46
32 // Pins para palabra corregida
33 #define COR1 44
34 #define COR2 42
35 #define COR3 40
36 #define COR4 38
37 // Pins para canal
38 #define CCOUT1 36
39 #define CCIN1 34
40 // Pins para canal
41 #define CCOUT2 32
42 #define CCIN2 30
43
44 // Variables globales para codificacion y decodficacion
45 int d1, d2, d3, d4, cd, c1, c2;
46 int ce[8]{ 0,0,0,0,0,0,0,0 };
47 int error[8]{ 0,0,0,0,0,0,0,0 };
```

```
48 int r[8]{ 0,0,0,0,0,0,0,0 };
49 int syndrome[4]{ 0,0,0,0 };
50 int cr[8]{ 0,0,0,0,0,0,0,0 };
51
52 // Prototipo de funciones
53 void ReadPins();
54 void TransmisionCanalError(int codidgo[8]);
55 void PatronError();
56 void AgregaError();
57 void ObtieneSindrome(int error[8]);
58 void TransmisionErrorDecoder(int codidgo[8]);
59 void DecodificaPalabra();
60
61 // Funcion de inicio
62 void setup()
63 {
64     // Asignar un modo a los pines a utilizar
65     ReadPins();
66     Serial.begin(9600);
67     Serial.println("Puerto abierto");
68 }
69
70 // Funcion de proceso principal
71 void loop()
72 {
73     // Condiciones iniciales
74     delay(2000);
75     digitalWrite(COR1, 0);
76     digitalWrite(COR2, 0);
77     digitalWrite(COR3, 0);
78     digitalWrite(COR4, 0);
79     digitalWrite(S1, 0);
80     digitalWrite(S2, 0);
81     digitalWrite(S3, 0);
82     digitalWrite(S4, 0);
83
84     // Control de fuente
85     cd = digitalRead(CD);
86     while (cd == 0)
87         cd = digitalRead(CD); // Permitir transmision
88
89     // Lectura de palabra dato
90     int codigo[8]{ 0,0,0,0,0,0,0,0 };
91     d1 = digitalRead(D1);
92     d2 = digitalRead(D2);
93     d3 = digitalRead(D3);
94     d4 = digitalRead(D4);
95
96     // Generacion de palabra codigo
97     codigo[0] = d2 ^ d3 ^ d4;
98     codigo[1] = d1 ^ d2 ^ d3;
```

```
99  codigo[2] = d1 ^ d2 ^ d4;
100  codigo[3] = d1 ^ d3 ^ d4;
101  codigo[4] = d1;
102  codigo[5] = d2;
103  codigo[6] = d3;
104  codigo[7] = d4;
105
106  // Se inicia la transmision de la fuente
107  // al canal
108  TransmisionCanalError(codigo);
109
110  // Se prepara el error en el canal y se
111  // controla la transmision
112  if (digitalRead(CE))
113  {
114      // Leer los errores a agregar
115      PatronError();
116      // Agrega el error a la palabra
117      AgregaError();
118      // Se transmite la palabra codigo con error al
119      // decoder
120      TransmisionErrorDecoder(ce);
121      // Se obtiene el sindrome a partir del error
122      // del canal
123      ObtieneSindrome(error);
124      // Se corrige la palabra recibida
125      DecodificaPalabra();
126  }
127 }
128
129 // Asignacion de modo en los pines
130 void ReadPins()
131 {
132     pinMode(D1, INPUT);
133     pinMode(D2, INPUT);
134     pinMode(D3, INPUT);
135     pinMode(D4, INPUT);
136
137     pinMode(CD, INPUT);
138
139     pinMode(C1, OUTPUT);
140
141     pinMode(E1, INPUT);
142     pinMode(E2, INPUT);
143     pinMode(E3, INPUT);
144     pinMode(E4, INPUT);
145     pinMode(E5, INPUT);
146     pinMode(E6, INPUT);
147     pinMode(E7, INPUT);
148     pinMode(E8, INPUT);
149 }
```



```
150 pinMode(CE, INPUT);
151
152 pinMode(C2, OUTPUT);
153
154 pinMode(S1, OUTPUT);
155 pinMode(S2, OUTPUT);
156 pinMode(S3, OUTPUT);
157 pinMode(S4, OUTPUT);
158
159 pinMode(COR1, OUTPUT);
160 pinMode(COR2, OUTPUT);
161 pinMode(COR3, OUTPUT);
162 pinMode(COR4, OUTPUT);
163
164 pinMode(CCOUNT1, OUTPUT);
165 pinMode(CCOUNT2, OUTPUT);
166
167 pinMode(CCIN1, INPUT);
168 pinMode(CCIN2, INPUT);
169 }
170
171 void TransmissionCanalError(int codidgo[8])
172 {
173     Serial.println("Transmision fuente -> canal");
174     // Se inicia la transmision serial
175     for (auto i = 0; i < 8; ++i)
176     {
177         // Se transmite al canal
178         digitalWrite(CCOUNT1, codidgo[i]);
179         // Se lee el dato transmitido al canal
180         ce[i] = digitalRead(CCIN1);
181         // Se muestra el dato recibido
182         digitalWrite(C1, ce[i]);
183         Serial.print(ce[i]);
184         // Timer
185         delay(1000);
186     }
187     Serial.println();
188     Serial.println("Transmision terminada");
189     // Se liberan recursos
190     digitalWrite(C1, 0);
191 }
192
193 void PatronError()
194 {
195     // Se leen los errores a agregar a
196     // la palabra codigo
197     error[0] = digitalRead(E1);
198     error[1] = digitalRead(E2);
199     error[2] = digitalRead(E3);
200     error[3] = digitalRead(E4);
```

```
201 error[4] = digitalRead(E5);
202 error[5] = digitalRead(E6);
203 error[6] = digitalRead(E7);
204 error[7] = digitalRead(E8);
205 Serial.println("Patron error");
206 /*for (int i = 0; i < 8; ++i)
207     Serial.print(error[i]);
208 Serial.println();*/
209 }
210
211 void AgregaError()
212 {
213     // Se busca donde fue configurado el error
214     // y se agrega a la palabra codigo
215     if (error[0]) ce[0] = ce[0] xor 1;
216     if (error[1]) ce[1] = ce[1] xor 1;
217     if (error[2]) ce[2] = ce[2] xor 1;
218     if (error[3]) ce[3] = ce[3] xor 1;
219     if (error[4]) ce[4] = ce[4] xor 1;
220     if (error[5]) ce[5] = ce[5] xor 1;
221     if (error[6]) ce[6] = ce[6] xor 1;
222     if (error[7]) ce[7] = ce[7] xor 1;
223     /*Serial.println("Codigo con error");
224     for (int i = 0; i < 8; ++i)
225         Serial.print(ce[i]);
226     Serial.println();*/
227 }
228
229 void ObtieneSindrome(int error[8])
230 {
231     // Si el canal dejo de transmitir se cancela la
232     // operacion
233     if (!cr[0] && !cr[1] && !cr[2] && !cr[3] && !cr[4] && !cr[5] &&
234         !cr[6]) return;
235
236     // Se calcula el sindrome con base en la ecuaciones
237     // obtenidas
238     sindrome[0] = error[0] ^ error[5] ^ error[6] ^ error[7];
239     sindrome[1] = error[1] ^ error[4] ^ error[5] ^ error[6];
240     sindrome[2] = error[2] ^ error[4] ^ error[5] ^ error[7];
241     sindrome[3] = error[3] ^ error[4] ^ error[6] ^ error[7];
242
243     // Se muestra el sindrome en el arreglo de leds asignados
244     // en el circuito
245     digitalWrite(S1, sindrome[0]);
246     digitalWrite(S2, sindrome[1]);
247     digitalWrite(S3, sindrome[2]);
248     digitalWrite(S4, sindrome[3]);
249
250     /*Serial.println("Sindrome");
251     for (int i = 0; i < 4; ++i)
```

```
252     Serial.print(sindrome[i]);
253     Serial.println();*/
254 }
255
256 void TransmisionErrorDecoder(int codidgo[8])
257 {
258     // Se inicia la transmision serial
259     Serial.println("Transmision error -> decoder");
260     for (auto i = 0; i < 8; ++i)
261     {
262         // Se transmite al canal
263         digitalWrite(CCOUT2, codidgo[i]);
264         // Se lee el dato transmitido al canal
265         Serial.print(digitalRead(CCIN2));
266         // Se muestra el dato recibido
267         cr[i] = digitalRead(CCIN2);
268         digitalWrite(C2, cr[i]);
269         // Timer
270         delay(1000);
271     }
272     Serial.println();
273     Serial.println("Transmision terminada");
274     // Se liberan recursos
275     digitalWrite(C2, 0);
276 }
277
278 void DecodificaPalabra()
279 {
280     // Si el sindrome es 0000, significa se interrumpe la
281     // transmision o no se ha transmitido algo, por lo que se cancela
282     // la funcion
283     if (!sindrome[0] && !sindrome[1] && !sindrome[2] && !sindrome[3])
284         return;
285
286     // Con base la matriz H transpuesta se verifican la siguientes condiciones
287     // para corregir errores simples y dobles
288     Serial.println("Corrigiendo error");
289     if (sindrome[0] && !sindrome[1] && !sindrome[2] && !sindrome[3]) //1000
290     {
291         cr[0] = cr[0] xor 1;
292     }
293     else if (!sindrome[0] && sindrome[1] && !sindrome[2] && !sindrome[3]) //0100
294     {
295         cr[1] = cr[1] xor 1;
296     }
297     else if (!sindrome[0] && !sindrome[1] && sindrome[2] && !sindrome[3]) //0010
298     {
299         cr[2] = cr[2] xor 1;
300     }
301     else if (!sindrome[0] && !sindrome[1] && !sindrome[2] && sindrome[3]) //0001
302     {
```

```
303     cr[3] = cr[3] xor 1;
304 }
305 else if (!sindrome[0] && sindrome[1] && sindrome[2] && sindrome[3]) //0111
306 {
307     cr[4] = cr[4] xor 1;
308 }
309 else if (sindrome[0] && sindrome[1] && sindrome[2] && !sindrome[3]) //1110
310 {
311     cr[5] = cr[5] xor 1;
312 }
313 else if (sindrome[0] && sindrome[1] && !sindrome[2] && sindrome[3]) //1101
314 {
315     cr[6] = cr[6] xor 1;
316 }
317 else if (sindrome[0] && !sindrome[1] && sindrome[2] && sindrome[3]) //1011
318 {
319     cr[7] = cr[7] xor 1;
320 }
321 else if (!sindrome[0] && sindrome[1] && !sindrome[2] && sindrome[3]) //0101 doble
322     2,4
323 {
324     cr[1] = cr[1] xor 1;
325     cr[3] = cr[3] xor 1;
326 }
327 else if (sindrome[0] && !sindrome[1] && sindrome[2] && !sindrome[3]) //1010 doble
328     1,3
329 {
330     cr[0] = cr[0] xor 1;
331     cr[2] = cr[2] xor 1;
332 }
333
334 for (auto i = 4; i < 8; i++)
335 {
336     Serial.print(cr[i]);
337 }
338
339 // Se muestra la palabra corregida en el arrgle de
340 // leds asignados
341 digitalWrite(COR1, cr[4]);
342 digitalWrite(COR2, cr[5]);
343 digitalWrite(COR3, cr[6]);
344 digitalWrite(COR4, cr[7]);
345 Serial.println("");
346 // Timer
347 delay(1000);
348 }
```

## 5. Circuito lógico

### 5.1. Coder

A partir de la matriz generadora  $G$  y la siguiente ecuación se pueden obtener la palabras código.

$$\vec{c} = \vec{d}G \quad (1)$$

Donde:

- $\vec{c}$ : palabra código.
- $\vec{d}$ : palabra dato.
- $G$ : matriz generadora.

La matriz generadora es la siguiente:

$$G = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

El vector de la palabra dato es:

$$\vec{d} = [d_1 \ d_2 \ d_3 \ d_4] \quad (2)$$

Realizando la operación se podrán obtener los diferentes valores del vector  $\vec{c}$ . Las ecuaciones resultantes de esta operación son las siguientes.

$$c_1 = d_2 \oplus d_3 \oplus d_4$$

$$c_2 = d_1 \oplus d_2 \oplus d_3$$

$$c_3 = d_1 \oplus d_2 \oplus d_4$$

$$c_4 = d_1 \oplus d_3 \oplus d_4$$

$$c_5 = d_1$$

$$c_6 = d_2$$

$$c_7 = d_3$$

$$c_8 = d_4$$

Y la palabra código resultante es:

$$\vec{c} = [c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6 \ c_7 \ c_8] \quad (3)$$

### 5.1.1. Circuito lógico

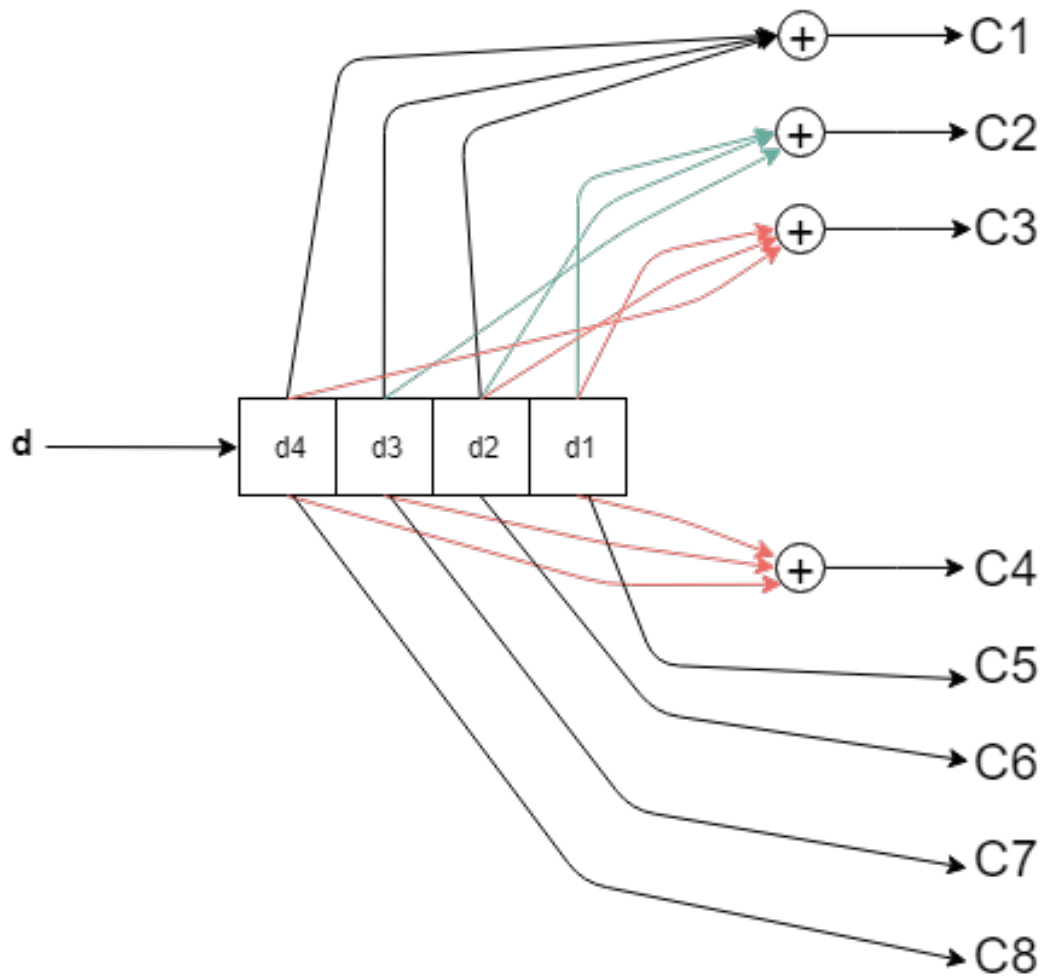


Figura 4: Circuito lógico codificador.

## 5.2. Decoder

Para el proceso de corrección y decodificación se debe de obtener la matriz  $\mathbf{H}$ .

$$H = [IP^T] \quad (4)$$

Para obtener la ecuaciones del Síndrome se calcula  $H$  transpuesta.

$$H^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Las ecuaciones del Síndrome son:

$$s_1 = r_1 + r_6 + r_7 + r_8$$

$$s_2 = r_2 + r_5 + r_6 + r_7$$

$$s_3 = r_3 + r_5 + r_6 + r_8$$

$$s_4 = r_4 + r_5 + r_7 + r_8$$

### 5.2.1. Circuito lógico síndrome

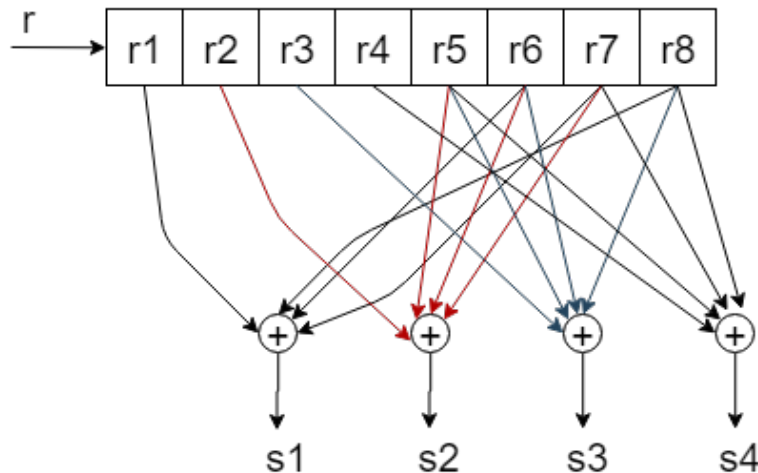


Figura 5: Circuito lógico síndrome.

Con ayuda de la misma matriz se pueden obtener las ecuaciones de error que ayudarán para la corrección de errores.

Errores simples.

$$e_1 = s_1 + \neg s_2 + \neg s_3 + \neg s_4$$

$$e_2 = \neg s_1 + s_2 + \neg s_3 + \neg s_4$$

$$e_3 = \neg s_1 + \neg s_2 + s_3 + \neg s_4$$

$$e_4 = \neg s_1 + \neg s_2 + \neg s_3 + s_4$$

$$e_5 = \neg s_1 + s_2 + s_3 + s_4$$

$$e_6 = s_1 + s_2 + s_3 + \neg s_4$$

$$e_7 = s_1 + s_2 + \neg s_3 + s_4$$

$$e_8 = s_1 + \neg s_2 + s_3 + s_4$$

Errores dobles.

$$e_{1,3} = s_1 + \neg s_2 + s_3 + \neg s_4$$

$$e_{2,4} = \neg s_1 + s_2 + \neg s_3 + s_4$$



### 5.2.2. Circuito lógico decodificador

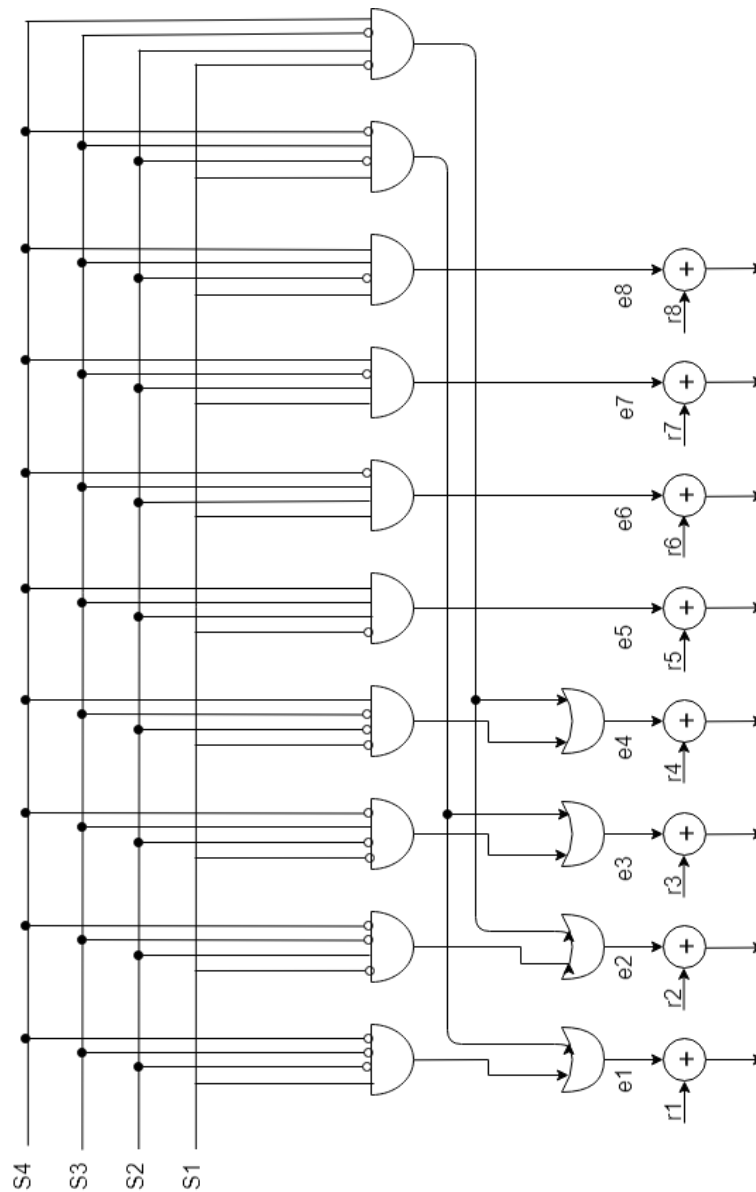


Figura 6: Circuito lógico decodificador.

## 6. Palabras dato y código

Palabra dato					Palabra código							
$d_1$	$d_2$	$d_3$	$d_4$		$c_1$	$c_2$	$c_3$	$c_4$	$c_5$	$c_6$	$c_7$	$c_8$
0	0	0	0		0	0	0	0	0	0	0	0
0	0	0	1		1	0	1	1	0	0	0	1
0	0	1	0		1	1	0	1	0	0	1	0
0	0	1	1		0	1	1	0	0	0	1	1
0	1	0	0		1	1	1	0	0	1	0	0
0	1	0	1		0	1	0	1	0	1	0	1
0	1	1	0		0	0	1	1	0	1	1	0
0	1	1	1		1	0	0	0	0	1	1	1
1	0	0	0		0	1	1	1	1	0	0	0
1	0	0	1		1	1	0	0	1	0	0	1
1	0	1	0		1	0	1	0	1	0	1	0
1	0	1	1		0	0	0	1	1	0	1	1
1	1	0	0		1	0	0	1	1	1	0	0
1	1	0	1		0	0	1	0	1	1	0	1
1	1	1	0		0	1	0	0	1	1	1	0
1	1	1	1		1	1	1	1	1	1	1	1

Figura 7: Palabras dato y código.

## 7. Síndrome vs patrón de error

Síndrome				Patrón de error								
$s_1$	$s_2$	$s_3$	$s_4$		$e_1$	$e_2$	$e_3$	$e_4$	$e_5$	$e_6$	$e_7$	$e_8$
0	0	0	0		0	0	0	0	0	0	0	0
1	0	0	0		1	0	0	0	0	0	0	0
0	1	0	0		0	1	0	0	0	0	0	0
0	0	1	0		0	0	1	0	0	0	0	0
0	0	0	1		0	0	0	1	0	0	0	0
0	1	1	1		0	0	0	0	1	0	0	0
1	1	1	0		0	0	0	0	0	1	0	0
1	1	0	1		0	0	0	0	0	0	1	0
1	0	0	1		0	0	0	0	0	0	0	1
1	0	1	0		1	0	1	0	0	0	0	0
0	1	0	1		0	1	0	1	0	0	0	0

Figura 8: Síndrome vs patrón de error.

## 8. Resultados simulación

La simulación fue realizada con ayuda del programa Proteus y la librería simulino. Estas herramientas permiten ejecutar el código realizado en un arduino virtual.

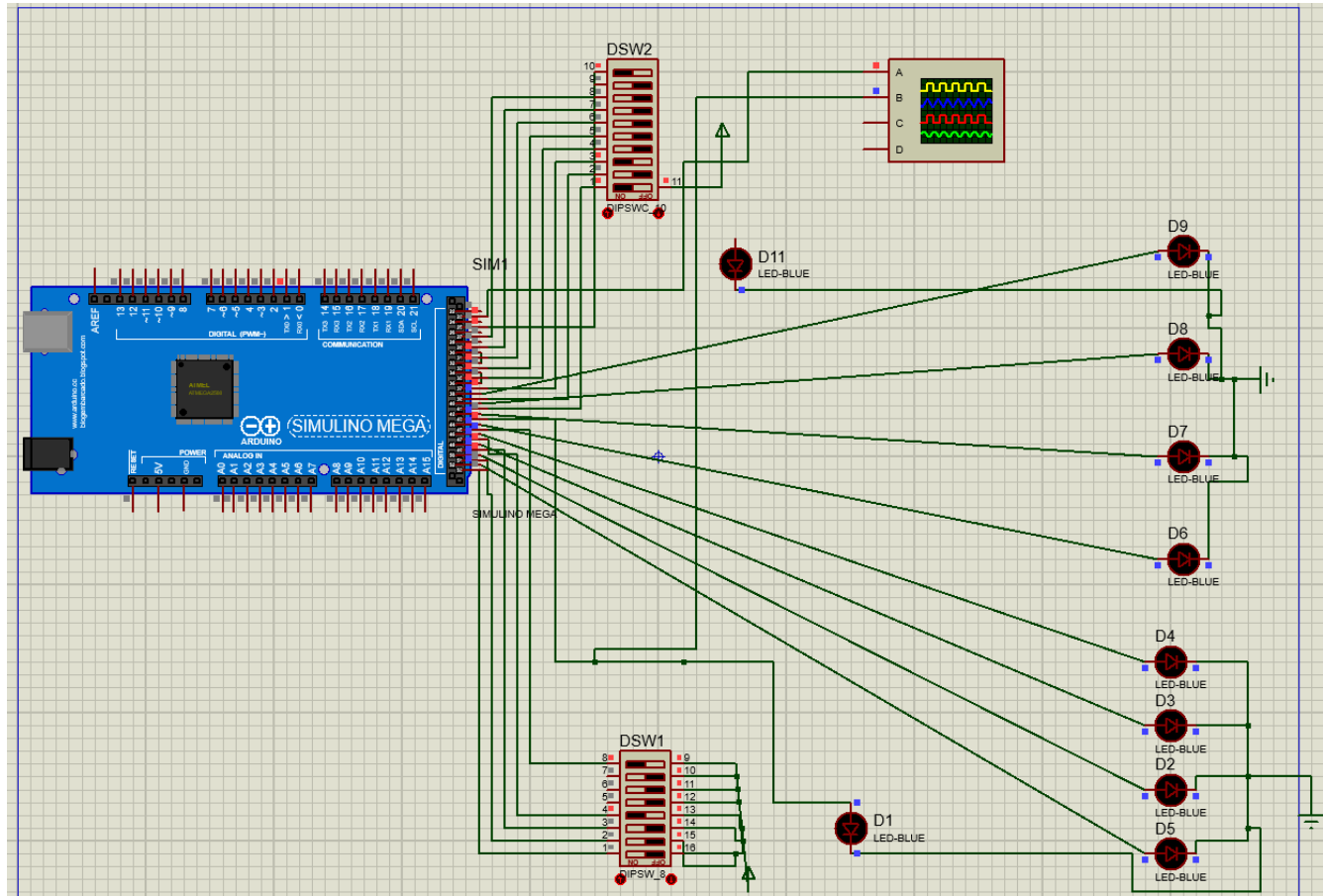


Figura 9: Esquema de simulación.

Para la simulación se utilizó la palabra dato **1010**, lo que da como resultado la palabra código **10101010**. Posteriormente se le agregó un error doble en las posiciones 2, 4. Agregando estos errores la palabra resultante con error es **11111010**.

Como se muestra en la siguiente imagen, en el canal **A** se tiene la palabra codificada y el canal **B** la palabra codificada con error.

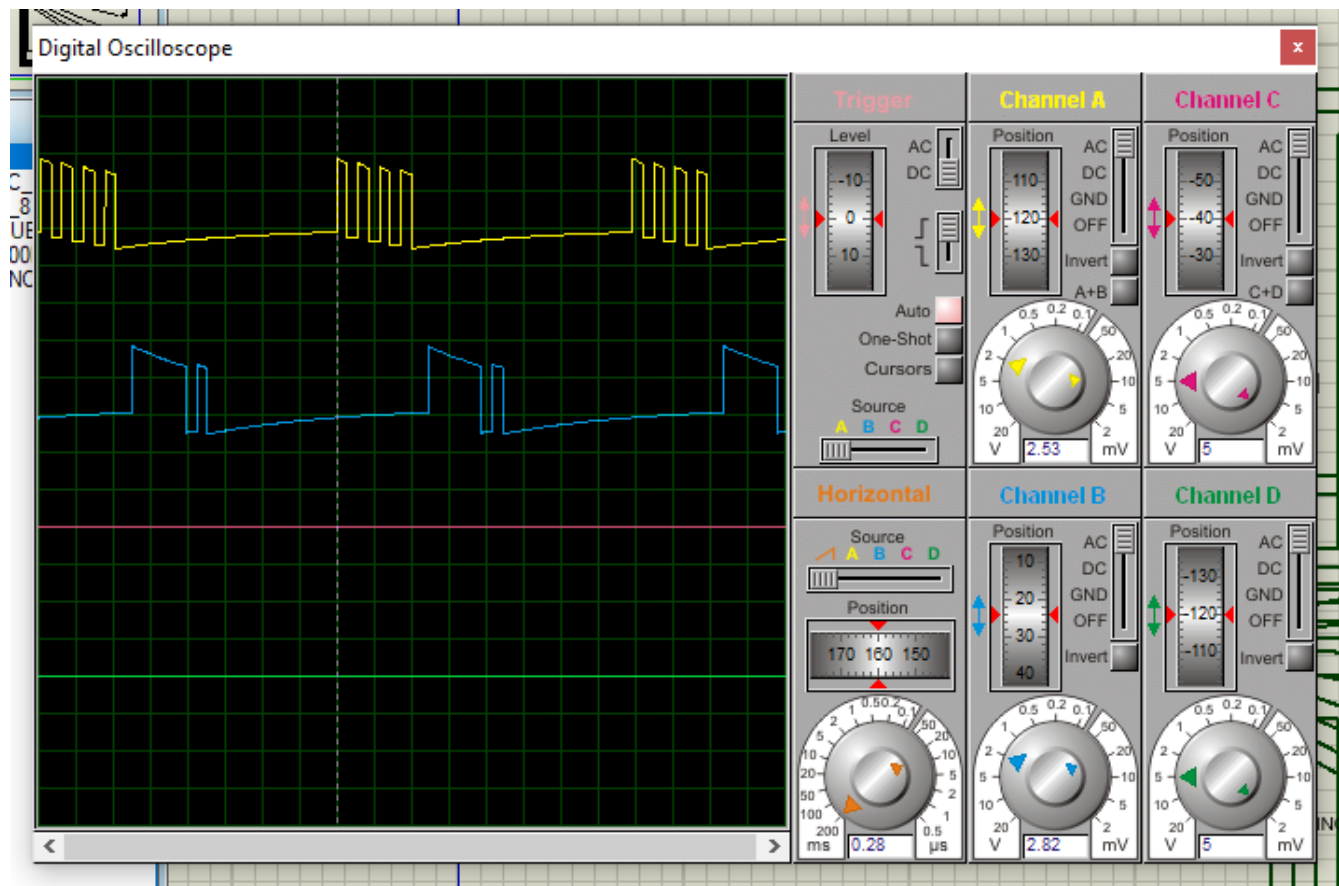


Figura 10: Pulsos generados de la palabra código y la palabra código con error.

En la última imagen de esta sección se muestra las palabra código, el síndrome y la palabra dato recuperada después de corregir el error y decodificarla, que fue **1010**.

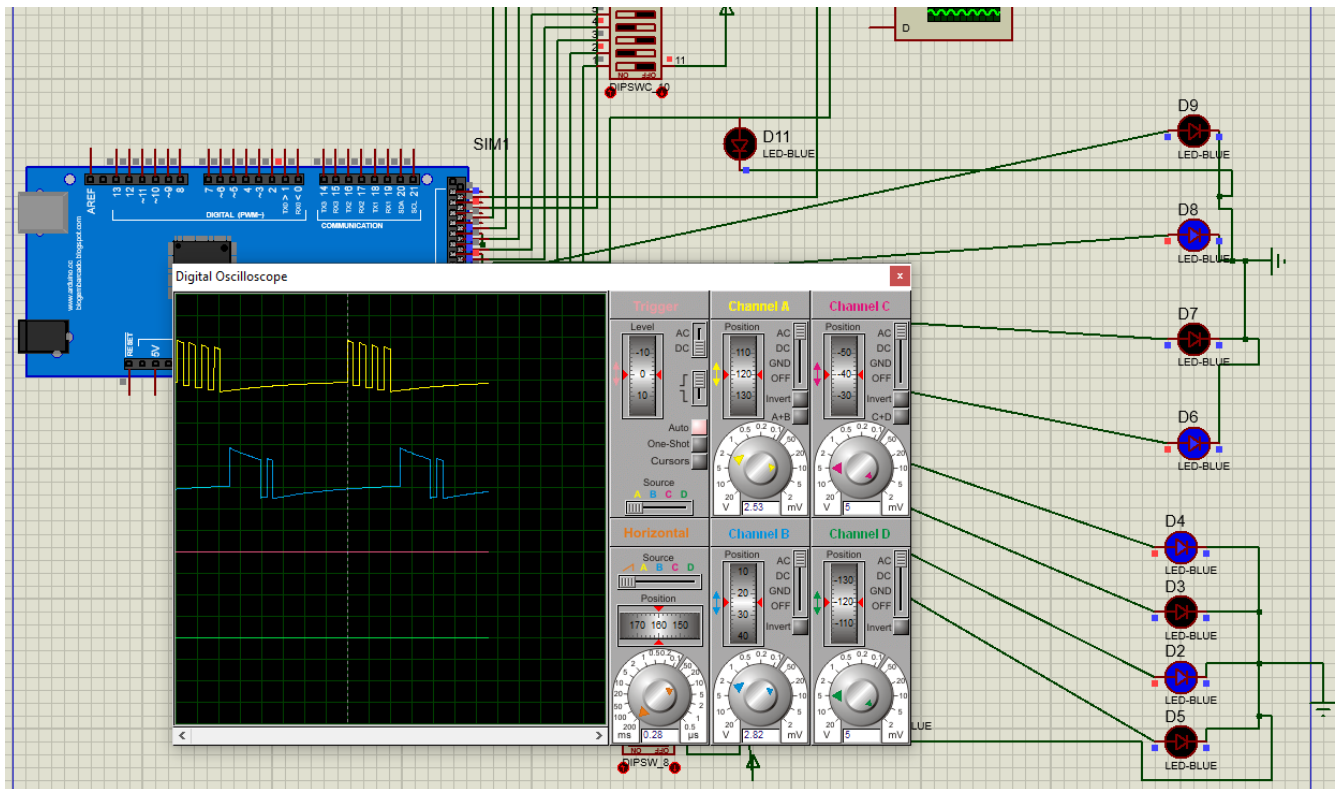


Figura 11: Pulsos generados de la palabra código, palabra código con error, síndrome y palabra dato.

## 9. Listado de asignación de terminales

Pin	Función
D1	Palabra dato posición 1
D2	Palabra dato posición 2
D3	Palabra dato posición 3
D4	Palabra dato posición 4
CD	Control de dip para la transmisión de palabra código
C1	Led para mostrar transmisión serial
E1	Patrón de error en la posición 1
E2	Patrón de error en la posición 2
E3	Patrón de error en la posición 3
E4	Patrón de error en la posición 4
E5	Patrón de error en la posición 5
E6	Patrón de error en la posición 6
E7	Patrón de error en la posición 7
E8	Patrón de error en la posición 8
CE	Control de dip para la transmisión de palabra código con error
C2	Led para mostrar transmisión serial
S1	Síndrome en la posición 1
S2	Síndrome en la posición 2
S3	Síndrome en la posición 3
S4	Síndrome en la posición 4
COR1	Palabra corregida en la posición 1
COR2	Palabra corregida en la posición 2
COR3	Palabra corregida en la posición 3
COR4	Palabra corregida en la posición 4
CCOUT1	Salida de fuente a canal
CCIN1	Entrada de canal a bloque con error
CCOUT2	Salida de bloque de error
CCIN2	Enlace de canal hacia decodificador

Figura 12: Terminales.

## 10. Diagramas de flujo

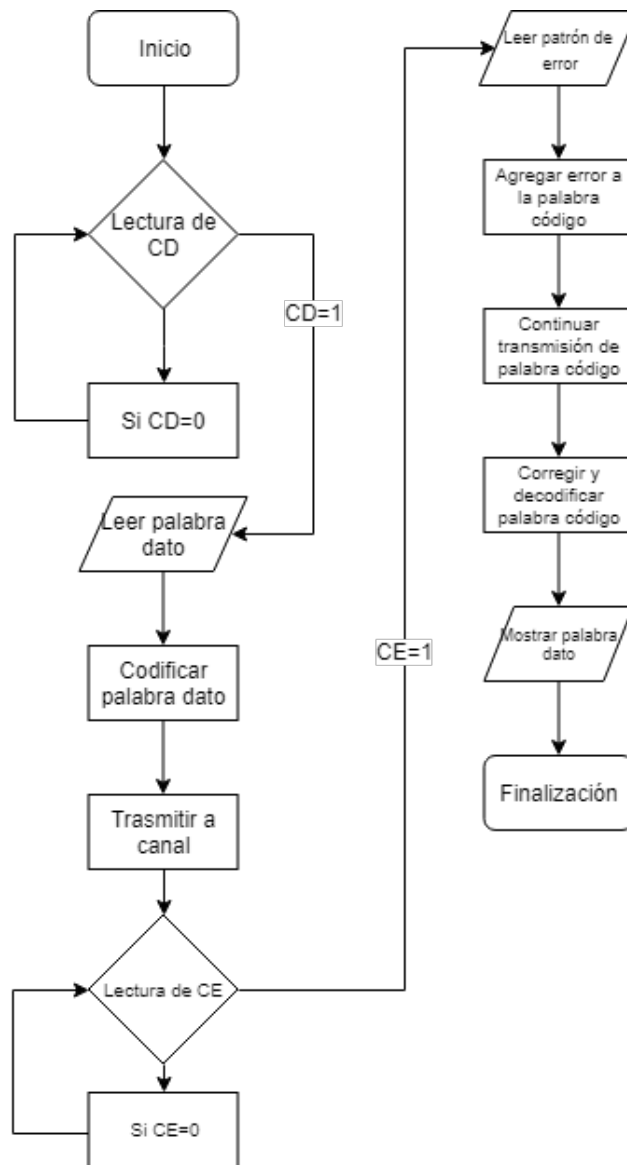


Figura 13: Diagrama principal.



### 10.1. Codificador

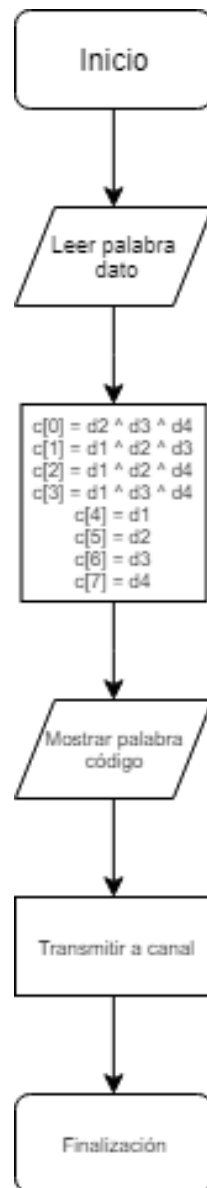


Figura 14: Codificador.

## 10.2. Decodificador

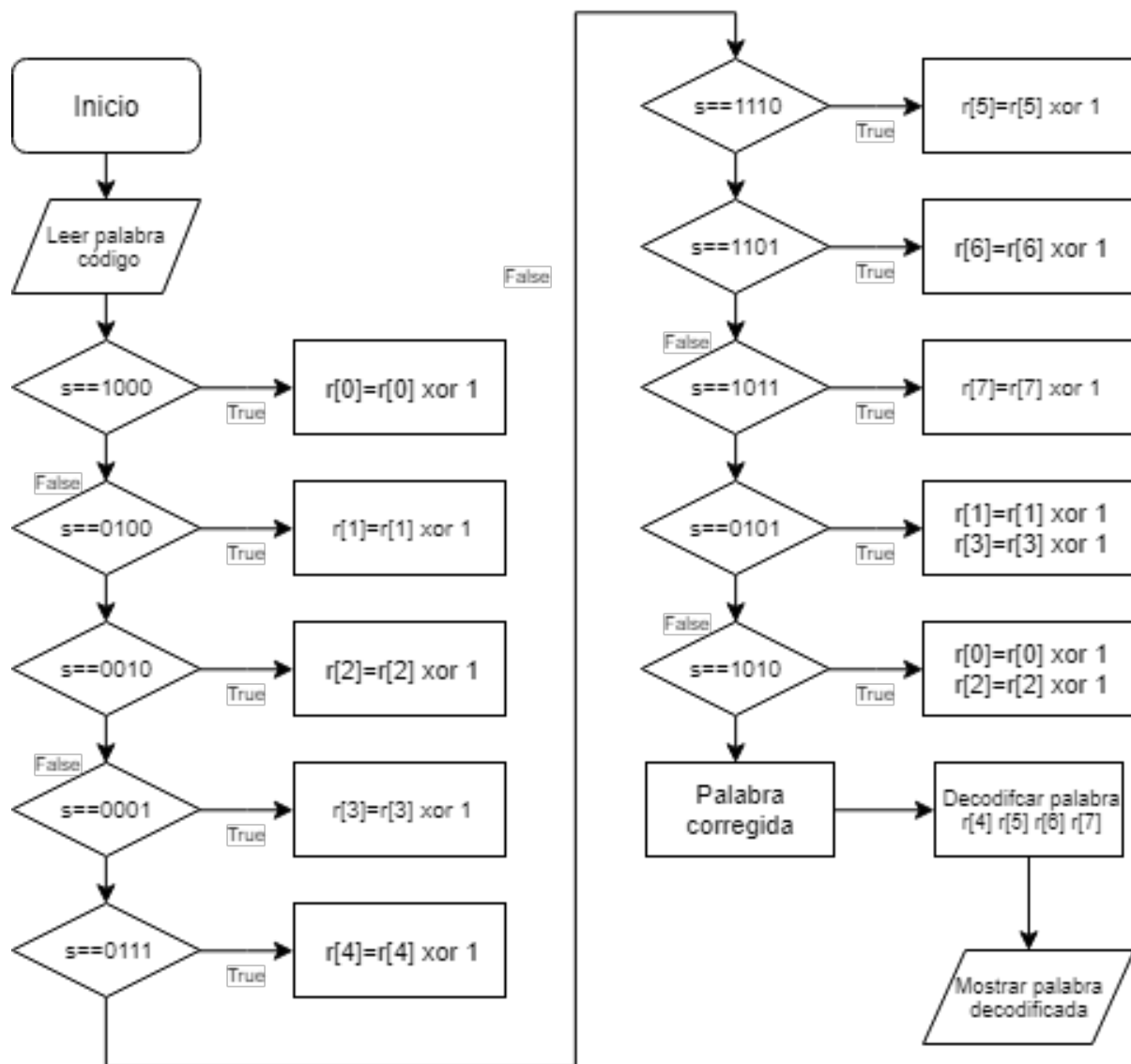
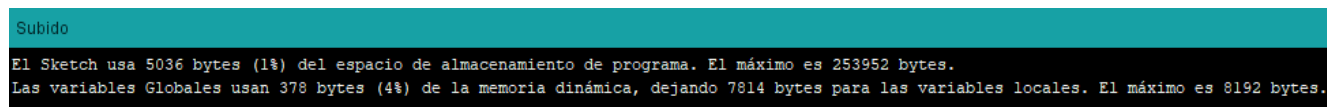


Figura 15: Decodificador.

### 10.3. Utilización de dispositivo

De acuerdo con información por el IDE de arduino, la utilización es la siguiente.



```
Subido
El Sketch usa 5036 bytes (1%) del espacio de almacenamiento de programa. El máximo es 253952 bytes.
Las variables Globales usan 378 bytes (4%) de la memoria dinámica, dejando 7814 bytes para las variables locales. El máximo es 8192 bytes.
```

Figura 16: Utilización.

## 11. Conclusiones

Al comienzo de la realización de la práctica fue necesario empezar con el análisis del caso particular asignado. Este análisis permitió conocer las ecuaciones que debían ser implementadas para el desarrollo de la práctica. La matriz generadora asignada fue particular, ya que el componente de paridad se encontraba invertido, lo que al inicio fue algo confuso. Pero después de investigar acerca de esta característica, se encontró la solución, la cual no distaba mucho de lo aprendido en clase.

Aparte del punto mencionado anteriormente, el análisis para encontrar las ecuaciones para la codificación, para obtener el síndrome y las ecuaciones de error, se desarrolló normalmente. Cabe mencionar que para este caso se debía de contemplar dos pares de errores dobles en las posiciones [1,3] y [2,4].

En la implementación se requirió que la comunicación entre los bloques del codificador, el canal y el decodificador fueran en serie, lo que supuso una dificultad. Para sortear esto surgieron dos ideas, controlar la transmisión del codificador y del canal o sincronizar los datos transmitidos. Se optó por controlar la transmisión del codificador y del canal, ya que permitía tener un mejor control de los componentes y eventos dentro del programa. La comunicación serial en este caso fue realizada mediante dos puentes dentro del mismo Arduino y con un pin extra de salida para mostrar los datos transmitidos.

En la etapa de pruebas se introdujeron las combinaciones para una palabra dato de 4 bits. Así mismo, se probaron los errores simples en todas las posiciones disponibles y los pares de errores dobles. Posteriormente se probaron errores dobles en diferentes posiciones e incluso errores triples. Como resultado de esta última etapa de pruebas para el decodificador, mientras que realizaba algo, no necesariamente era correcto. Esto debido a que fue diseñado para casos específicos de errores dobles con la capacidad de corregir errores simples. En algunos casos podría decirse que corrigió correctamente, pero esta respuesta debió clasificarse como falso positivo, ya que la entrada no era la correcta.

Con la realización de esta última práctica se entendió mejor la manera en que la corrección de errores es implementada en un sistema de comunicación. Es importante señalar que la etapa de análisis

es fundamental y debe de revisarse y comprobarse en busca de errores que puedan afectar en la implementación. También se debe de contemplar las limitaciones que el diseño y la implantación tenga, para dejar claro lo que es capaz de realizar.