Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio

Entregado como requisito para la obtención del título de
Licenciatura en Sistemas.
Infraestructura.

Nicolás Marino - 231142
Nicolás Sogliano - 238475
Jonathan Salaberriborda - 175034

Tutor: Tomás De Angelis.

2020

ÍNDICE

1. *Parte A*

1.1. Utilizando el emulador Logic.ly, en su versión gratuita, se desarrollarán los siguientes circuitos.

1.1.1. Contador de 0 a 5 asincrónico, de uno en uno, ascendente, circular.

Contador de 0 a 5 asincrónico, de uno en uno, ascendente, circular.

- Estado:0,1,2,3,4,5
- Entrada: 0, 1 (Porque es asincrónico.)
- Salida:
- Transiciones:
    - Circular, ascendente, 1 en 1.

Obtuvimos la tabla de verdad para s2, s1, s0, q3 n+1, q2 n+1 y q1 n+1.

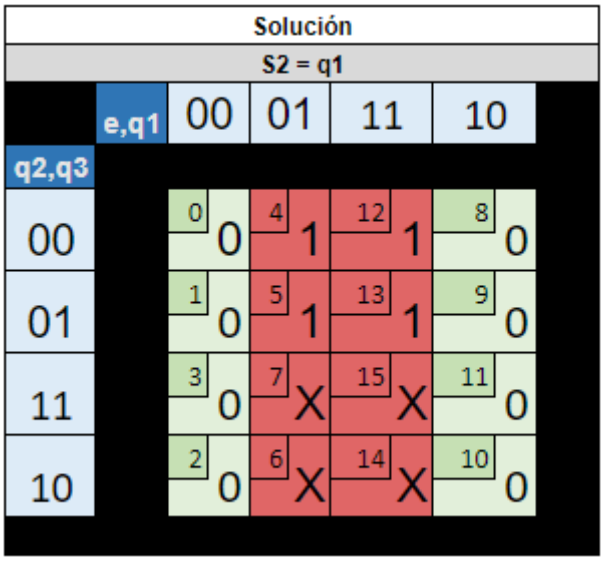
Estado:	0,1,2,3,4,5
Entrada:	0,1
Salida:	estado actual
Transciones:	circular

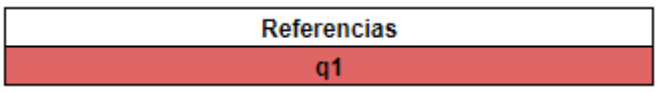
e	q3 n	q2 n	q1 n	q3 n+1	q2 n+1	q1 n+1	s2	s1	s0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	1	0	1	1
0	1	0	0	1	0	0	1	0	0
0	1	0	1	1	0	1	1	0	1
0	1	1	0	X	X	X	X	X	X
0	1	1	1	X	X	X	X	X	X
1	0	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	0	1
1	0	1	0	0	1	1	0	1	0
1	0	1	1	1	0	0	0	1	1
1	1	0	0	1	0	1	1	0	0
1	1	0	1	0	0	0	1	0	1
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

Nos enfocamos en la tabla de verdad para s2.

e	q3 n	q2 n	q1 n	s2	Indice
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	2
0	0	1	1	0	3
0	1	0	0	1	4
0	1	0	1	1	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	0	10
1	0	1	1	0	11
1	1	0	0	1	12
1	1	0	1	1	13
1	**1**	**1**	**0**	**X**	**14**
1	**1**	**1**	**1**	**X**	**15**

Introducimos los valores para s2 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para s2 es q1.

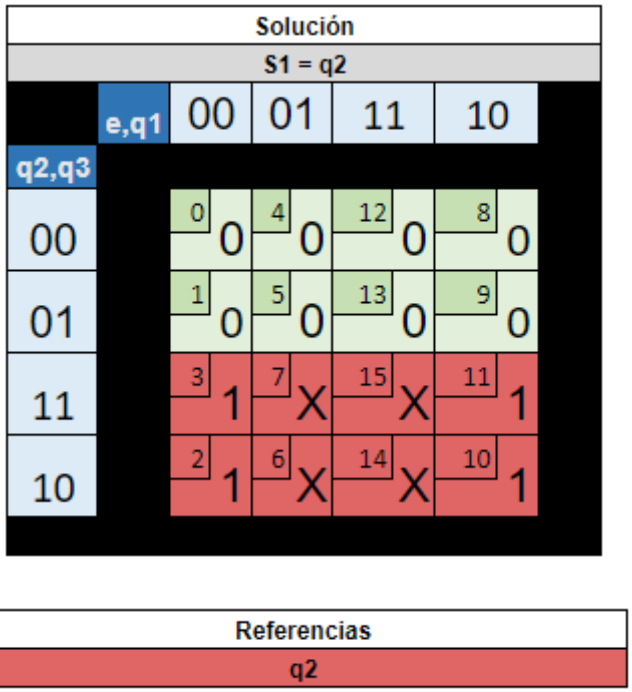




Nos enfocamos en la tabla de verdad para s1.

e	q3 n	q2 n	q1 n	s1	Índice
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	2
0	0	1	1	1	3
0	1	0	0	0	4
0	1	0	1	0	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	1	10
1	0	1	1	1	11
1	1	0	0	0	12
1	1	0	1	0	13
1	1	1	0	X	14
1	1	1	1	X	15

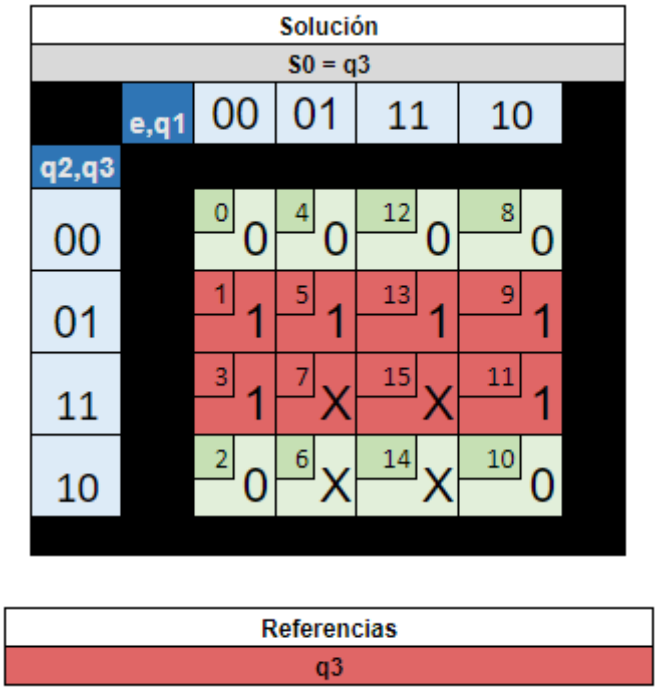
Introducimos los valores para s1 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para s1 es q2.



Nos enfocamos en la tabla de verdad para s0.

e	q3 n	q2 n	q1 n	s0	Índice
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	0	2
0	0	1	1	1	3
0	1	0	0	0	4
0	1	0	1	1	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	0	8
1	0	0	1	1	9
1	0	1	0	0	10
1	0	1	1	1	11
1	1	0	0	0	12
1	1	0	1	1	13
1	**1**	**1**	**0**	**X**	**14**
1	**1**	**1**	**1**	**X**	**15**

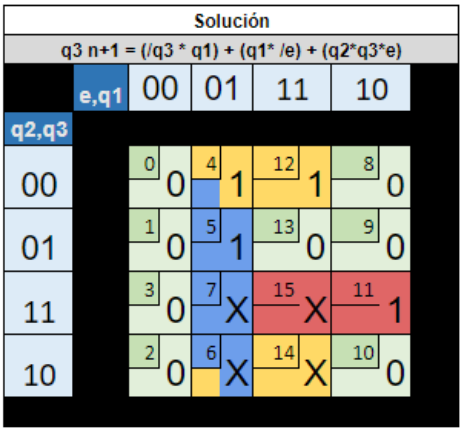
Introducimos los valores para s0 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para s0 es q3.

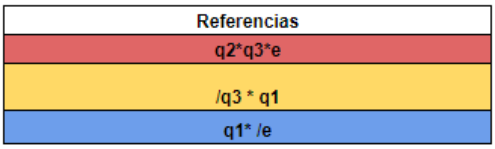


Nos enfocamos en la tabla de verdad para q3 n+1

e	q1	q2	q3	q3 n+1	Índice
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	2
0	0	1	1	0	3
0	1	0	0	1	4
0	1	0	1	1	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	0	10
1	0	1	1	1	11
1	1	0	0	1	12
1	1	0	1	0	13
1	**1**	**1**	**0**	**X**	**14**
1	**1**	**1**	**1**	**X**	**15**

Introducimos los valores para q3 n+1 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para q3 n+1 es (/q3 * q1) + (q1* /e) + (q2*q3*e).

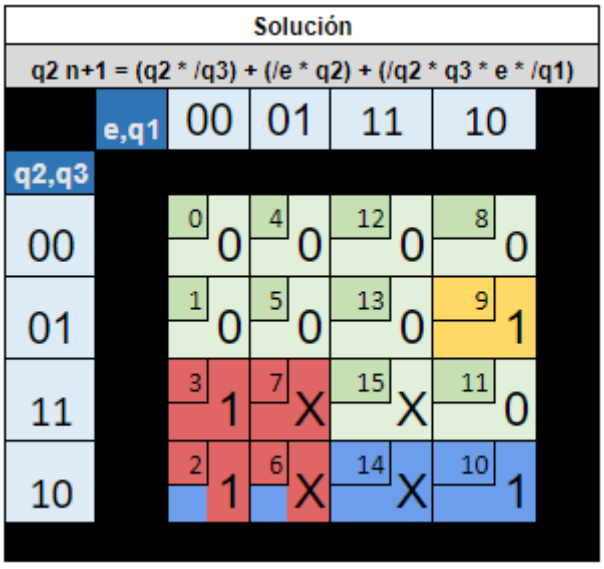


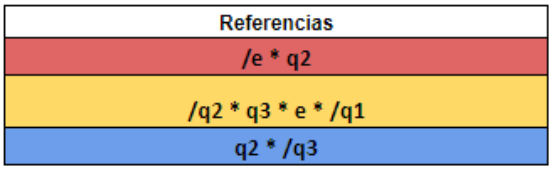


Nos enfocamos en la tabla de verdad para q2 n+1.

e	q1	q2	q3	q2 n+1	Índice
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	2
0	0	1	1	1	3
0	1	0	0	0	4
0	1	0	1	0	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	0	8
1	0	0	1	1	9
1	0	1	0	1	10
1	0	1	1	0	11
1	1	0	0	0	12
1	1	0	1	0	13
1	**1**	**1**	**0**	**X**	**14**
1	**1**	**1**	**1**	**X**	**15**

Introducimos los valores para q2 n+1 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para q2 n+1 es (q2 * /q3) + (/e * q2) + (/q2 * q3 * e * /q1).

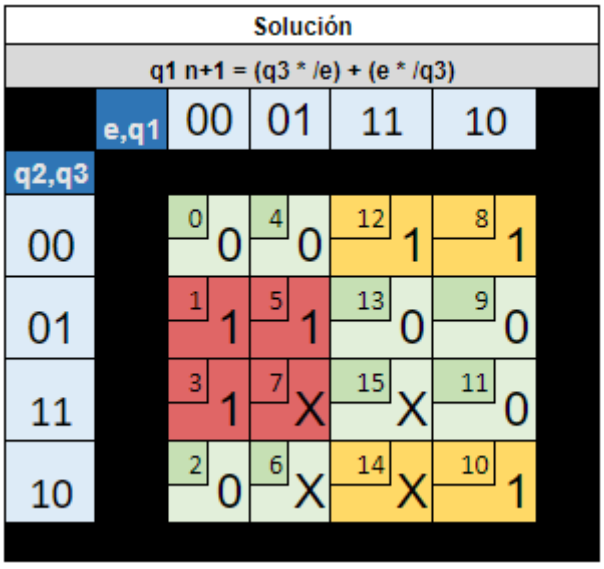


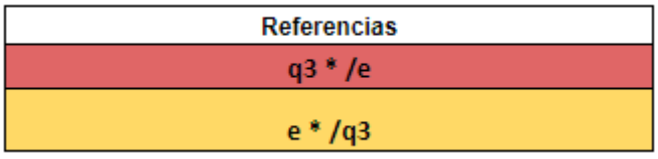


Nos enfocamos en la tabla de verdad para q1 n+1.

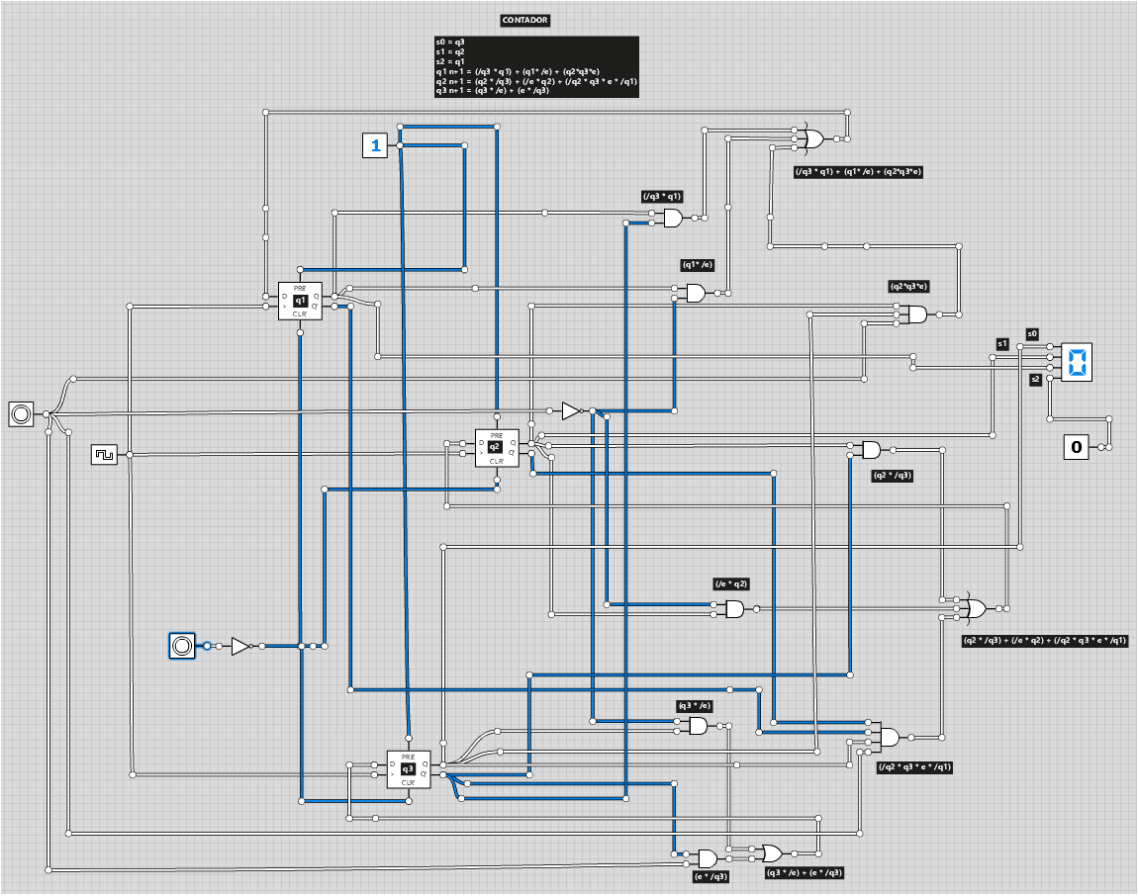
e	q1	q2	q3	q1 n+1	Índice
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	0	2
0	0	1	1	1	3
0	1	0	0	0	4
0	1	0	1	1	5
0	**1**	**1**	**0**	**X**	**6**
0	**1**	**1**	**1**	**X**	**7**
1	0	0	0	1	8
1	0	0	1	0	9
1	0	1	0	1	10
1	0	1	1	0	11
1	1	0	0	1	12
1	1	0	1	0	13
1	**1**	**1**	**0**	**X**	**14**
1	**1**	**1**	**1**	**X**	**15**

Introducimos los valores para q1 n+1 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para q1 n+1 es (q3 * /e) + (e * /q3).





Implementamos el contador en logic.ly.



1.2.Codificador con 8 entradas y 3 salidas. Código a elección.

A continuación adjuntamos el código elegido.

e0	000
e1	001
e2	010
e3	011
e4	100
e5	101
e6	110
e7	111

Obtuvimos la tabla de verdad para s2, s1 y s0.

e7	e6	e5	e4	e3	e2	e1	e0	s2	s1	s0
0	0	0	0	0	0	0	**1**	**0**	**0**	**0**
0	0	0	0	0	0	**1**	0	**0**	**0**	**1**
0	0	0	0	0	**1**	0	0	**0**	**1**	**0**
0	0	0	0	**1**	0	0	0	**0**	**1**	**1**
0	0	0	**1**	0	0	0	0	**1**	**0**	**0**
0	0	**1**	0	0	0	0	0	**1**	**0**	**1**
0	**1**	0	0	0	0	0	0	**1**	**1**	**0**
1	0	0	0	0	0	0	0	**1**	**1**	**1**

Nos enfocamos en la tabla de verdad para s2.

e7	e6	e5	e4	e3	e2	e1	e0	s2	indice
0	0	0	0	0	0	0	**1**	**0**	1
0	0	0	0	0	0	**1**	0	**0**	2
0	0	0	0	0	**1**	0	0	**0**	4
0	0	0	0	**1**	0	0	0	**0**	8
0	0	0	**1**	0	0	0	0	**1**	16
0	0	**1**	0	0	0	0	0	**1**	32
0	**1**	0	0	0	0	0	0	**1**	64
1	0	0	0	0	0	0	0	**1**	128

Introducimos los valores para s2 en el mapa de Karnaugh, nos enfocamos en los “1” (suma de productos) para los rectángulos y vemos que la solución para s2 es /e3 * /e2 * /e1 * /e0.

Para una mejor apreciación de los valores recomendamos ver el archivo adjunto Excel …..

Solución s2 = /e3 * /e2 * /e1 * /e0																	
KARNAUGH S2	e7,e6,e5,e4	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
e3,e2,e1,e0																	
0000		0 X	16 1	32 X	48 1	64 X	80 X	96 X	112 1	240 X	224 X	208 X	192 X	176 X	160 X	144 X	128 1
0001		1 0	17 X	33 X	49 X	65 X	81 X	97 X	113 X	241 X	225 X	209 X	193 X	177 X	161 X	145 X	129 X
0011		2 X	18 X	34 X	50 X	66 X	82 X	98 X	114 X	242 X	226 X	210 X	194 X	178 X	162 X	146 X	130 X
0010		3 0	19 X	35 X	51 X	67 X	83 X	99 X	115 X	243 X	227 X	211 X	195 X	179 X	163 X	147 X	131 X
0110		4 X	20 X	36 X	52 X	68 X	84 X	100 X	116 X	244 X	228 X	212 X	196 X	180 X	164 X	148 X	132 X
0111		5 X	21 X	37 X	53 X	69 X	85 X	101 X	117 X	245 X	229 X	213 X	197 X	181 X	165 X	149 X	133 X
0101		6 X	22 X	38 X	54 X	70 X	86 X	102 X	118 X	246 X	230 X	214 X	198 X	182 X	166 X	150 X	134 X
0100		7 0	23 X	39 X	55 X	71 X	87 X	103 X	119 X	247 X	231 X	215 X	199 X	183 X	167 X	151 X	135 X
1100		15 X	31 X	47 X	63 X	79 X	95 X	111 X	127 X	255 X	239 X	223 X	207 X	191 X	175 X	159 X	143 X
1101		14 X	30 X	46 X	62 X	78 X	94 X	110 X	126 X	254 X	238 X	222 X	206 X	190 X	174 X	158 X	142 X
1111		13 X	29 X	45 X	61 X	77 X	93 X	109 X	125 X	253 X	237 X	221 X	205 X	189 X	173 X	157 X	141 X
1110		12 X	28 X	44 X	60 X	76 X	92 X	108 X	124 X	252 X	236 X	220 X	204 X	188 X	172 X	156 X	140 X
1010		11 X	27 X	43 X	59 X	75 X	91 X	107 X	123 X	251 X	235 X	219 X	203 X	187 X	171 X	155 X	139 X
1011		10 X	26 X	42 X	58 X	74 X	90 X	106 X	122 X	250 X	234 X	218 X	202 X	186 X	170 X	154 X	138 X
1001		9 X	25 X	41 X	57 X	73 X	89 X	105 X	121 X	249 X	233 X	217 X	201 X	185 X	169 X	153 X	137 X
1000		8 0	24 X	40 X	56 X	72 X	88 X	104 X	120 X	248 X	232 X	216 X	200 X	184 X	168 X	152 X	136 X

Nos enfocamos en la tabla de verdad para s1.

e7	e6	e5	e4	e3	e2	e1	e0	s1	indice
0	0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	2
0	0	0	0	0	1	0	0	1	4
0	0	0	0	1	0	0	0	1	8
0	0	0	1	0	0	0	0	0	16
0	0	1	0	0	0	0	0	0	32
0	1	0	0	0	0	0	0	1	64
1	0	0	0	0	0	0	0	1	128

Introducimos los valores para s1 en el mapa de Karnaugh, nos enfocamos en los “0” (productos de suma) para los rectángulos y vemos que la solución para s1 es e3 * e2 * e7 * e6.

Para una mejor apreciación de los valores recomendamos ver el archivo adjunto Excel …..

Solución s1 = e3 * e2 * e7 * e6																	
KARNAUGH S1	e7,e6,e5,e4	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
e3,e2,e1,e0																	
0000		0 X	16 0	32 X	48 0	64 X	80 X	96 X	112 1	240 X	224 X	208 X	192 X	176 X	160 X	144 X	128 1
0001		1 0	17 X	33 X	49 X	65 X	81 X	97 X	113 X	241 X	225 X	209 X	193 X	177 X	161 X	145 X	129 X
0011		2 X	18 X	34 X	50 X	66 X	82 X	98 X	114 X	242 X	226 X	210 X	194 X	178 X	162 X	146 X	130 X
0010		3 0	19 X	35 X	51 X	67 X	83 X	99 X	115 X	243 X	227 X	211 X	195 X	179 X	163 X	147 X	131 X
0110		4 X	20 X	36 X	52 X	68 X	84 X	100 X	116 X	244 X	228 X	212 X	196 X	180 X	164 X	148 X	132 X
0111		5 X	21 X	37 X	53 X	69 X	85 X	101 X	117 X	245 X	229 X	213 X	197 X	181 X	165 X	149 X	133 X
0101		6 X	22 X	38 X	54 X	70 X	86 X	102 X	118 X	246 X	230 X	214 X	198 X	182 X	166 X	150 X	134 X
0100		7 1	23 X	39 X	55 X	71 X	87 X	103 X	119 X	247 X	231 X	215 X	199 X	183 X	167 X	151 X	135 X
1100		15 X	31 X	47 X	63 X	79 X	95 X	111 X	127 X	255 X	239 X	223 X	207 X	191 X	175 X	159 X	143 X
1101		14 X	30 X	46 X	62 X	78 X	94 X	110 X	126 X	254 X	238 X	222 X	206 X	190 X	174 X	158 X	142 X
1111		13 X	29 X	45 X	61 X	77 X	93 X	109 X	125 X	253 X	237 X	221 X	205 X	189 X	173 X	157 X	141 X
1110		12 X	28 X	44 X	60 X	76 X	92 X	108 X	124 X	252 X	236 X	220 X	204 X	188 X	172 X	156 X	140 X
1010		11 X	27 X	43 X	59 X	75 X	91 X	107 X	123 X	251 X	235 X	219 X	203 X	187 X	171 X	155 X	139 X
1011		10 X	26 X	42 X	58 X	74 X	90 X	106 X	122 X	250 X	234 X	218 X	202 X	186 X	170 X	154 X	138 X
1001		9 X	25 X	41 X	57 X	73 X	89 X	105 X	121 X	249 X	233 X	217 X	201 X	185 X	169 X	153 X	137 X
1000		8 1	24 X	40 X	56 X	72 X	88 X	104 X	120 X	248 X	232 X	216 X	200 X	184 X	168 X	152 X	136 X

Nos enfocamos en la tabla de verdad para s0.

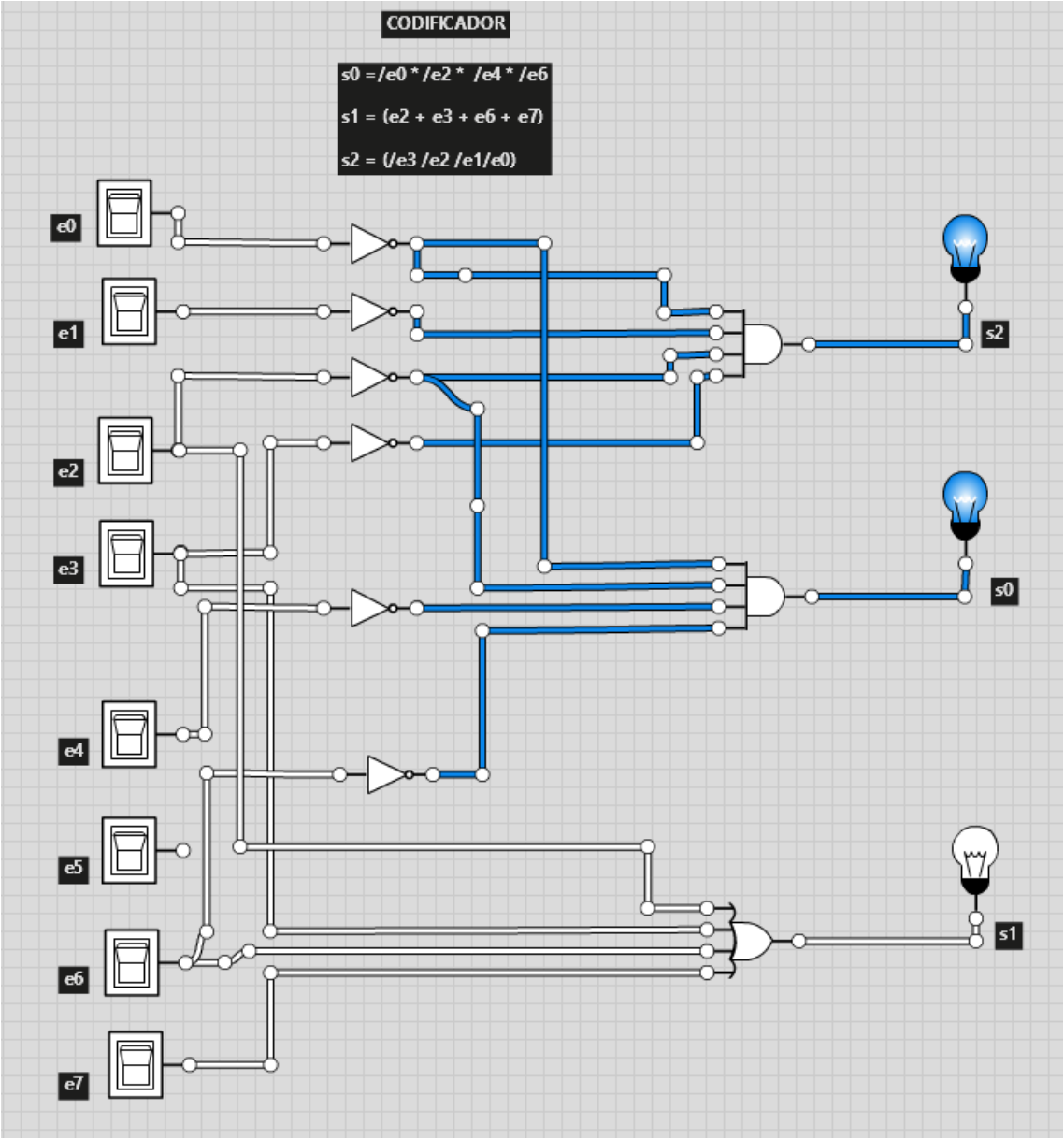
Para realizar la solución utilizamos formas canónicas, nos enfocamos en los “0” (productos de suma) y vemos que la solución para s0 es /e0 * /e2 * /e4 * /e6.

Solución								
s0 = /e0 * /e2 * /e4 * /e6								
e7	e6	e5	e4	e3	e2	e1	e0	s0
0	0	0	0	0	0	0	1	**0**
0	0	0	0	0	0	1	0	1
0	0	0	0	0	1	0	0	**0**
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	**0**
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	0	**0**
1	0	0	0	0	0	0	0	1

Una vez que listamos cada una de las soluciones vemos que queda de la siguiente manera.

Solución Final
s0 = /e0 * /e2 * /e4 * /e6
s1 = e3 * e2 * e7 * e6
s2 = /e3 * /e2 * /e1 * /e0

Implementamos del codificador en logic.ly.



1.3.Decodificador que sea la inversa del anterior.

Nos enfocamos en la tabla de verdad para las salidas s7, s6, s5, s4, s3, s2, s1 y s0.

Recordamos también la codificación para cada una de las entradas.

	e2	e1	e0	s7	s6	s5	s4	s3	s2	s1	s0
1	0	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	1	0
3	0	1	0	0	0	0	0	0	1	0	0
4	0	1	1	0	0	0	0	1	0	0	0
5	1	0	0	0	0	0	1	0	0	0	0
6	1	0	1	0	0	1	0	0	0	0	0
7	1	1	0	0	1	0	0	0	0	0	0
8	1	1	1	1	0	0	0	0	0	0	0

e0	000
e1	001
e2	010
e3	011
e4	100
e5	101
e6	110
e7	111

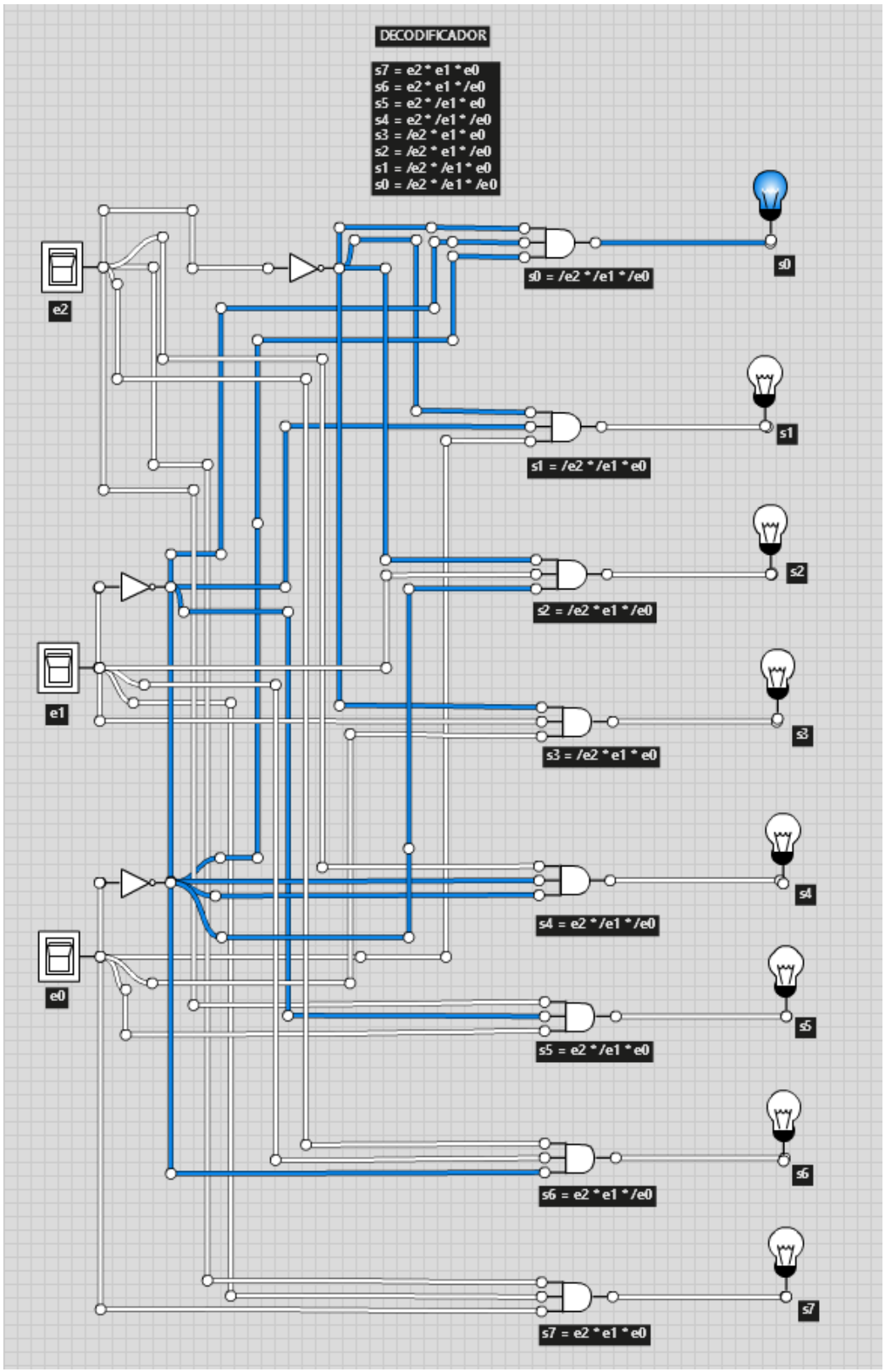
Detallamos cada una de las entradas cada vez que el bit de la salida correspondiente esta encendido “1”.

1	s0	000
2	s1	001
3	s2	010
4	s3	011
5	s4	100
6	s5	101
7	s6	110
8	s7	111

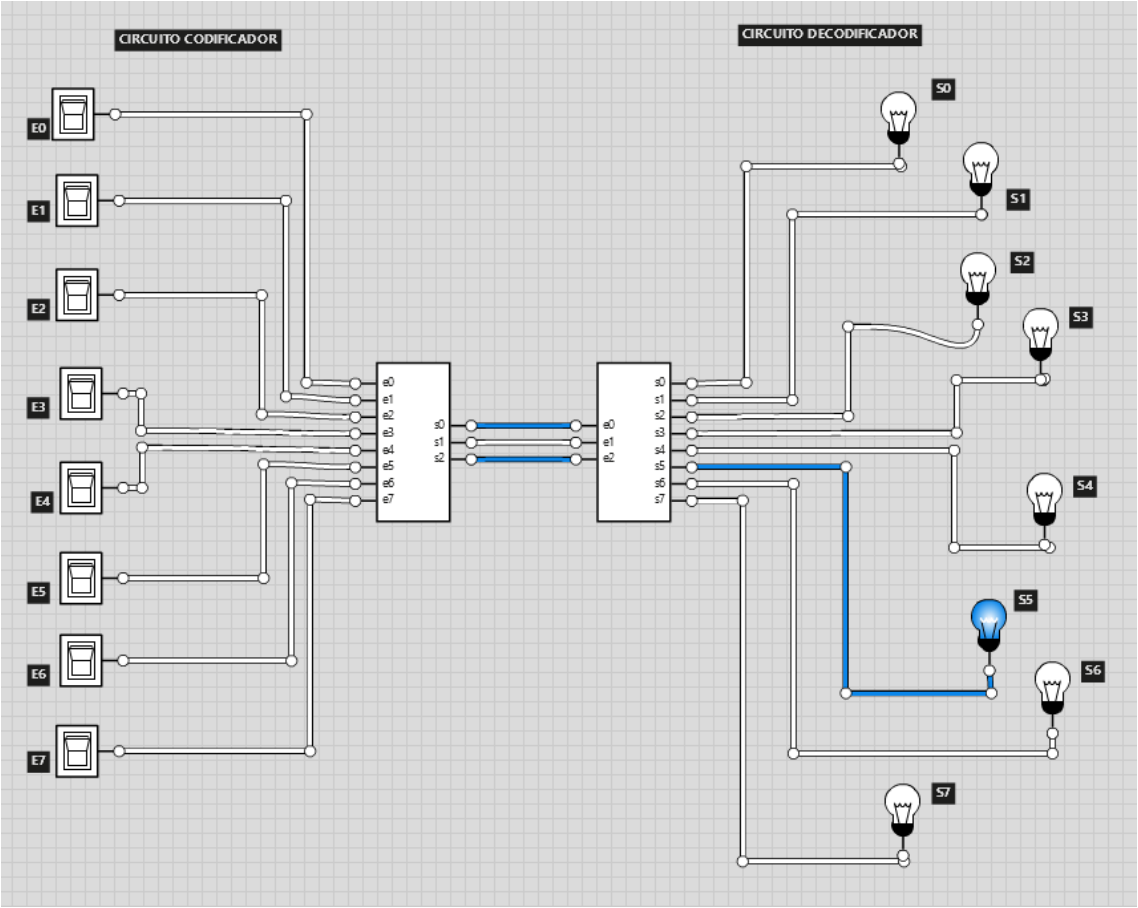
Para realizar la solución utilizamos formas canónicas, nos enfocamos en los “1” (suma de productos). Siendo la solución final la que adjuntamos a continuación.

Solución Final
s7 = e2 * e1 * e0
s6 = e2 * e1 * /e0
s5 = e2 * /e1 * e0
s4 = e2 * /e1 * /e0
s3 = /e2 * e1 * e0
s2 = /e2 * e1 * /e0
s1 = /e2 * /e1 * e0
s0 = /e2 * /e1 * /e0

Implementamos el decodificador en logic.ly.



Implementamos los circuitos del codificador y decodificador en logic.ly.



2. PARTE B

Se desarrollará, en Lenguaje Java, un simulador de un sistema operativo multitarea multiusuario.

Sobre tal sistema, ejecutarán un conjunto de programas, concurrentemente, y se deberá simular su ejecución compartiendo un recurso procesador, además del resto de los recursos.

Habrá un conjunto de recursos, los que deberán ser arbitrados, en cuanto a permisos, entre un conjunto de usuarios.

Concretamente:

- Habrá al menos 3 usuarios u1, u2, u3.
- Habrá al menos hasta 3 procesos concurrentes, p1, p2, p3.
- Habrá al menos 3 recursos, r1, r2 y r3.
- Cada proceso consistirá en la ejecución de un código, línea a línea, y dicho código puede incluir la solicitud, devolución o uso de los recursos.
- Los recursos se accederán bajo mutua exclusión.

El simulador deberá simular:

- El reparto del procesador.
- La ejecución concurrente.
- La política de permisos con respecto a usuarios y recursos.
- El pedido y devolución de recursos.
- La existencia eventual de deadlocks.
- La política de scheduling.
- El alojamiento del código en memoria

Contemplando las bases anteriores, cada grupo alineará a su elección el resto de los detalles.

Respecto a la salida, podrá usarse una interfaz que muestre un estado de las cosas, o un log (basado en texto) de la ejecución.

Se entregará un informe completo de lo realizado, incluyendo código, pruebas, decisiones de diseño, conclusiones, etc.

2.1.Decisiones de diseño

2.1.1. Clases

2.1.1.1.Clase Program

Atributos:

- name(String): Nombre del programa.
- processList(List<Process>): Lista de procesos que contiene el programa.

Métodos: Posee únicamente métodos de acceso y modificación (getters y setters).

2.1.1.2.Clase Process

Atributos:

- name(String): Nombre del proceso.
- status(Status): Estado del proceso.
- executionTimeout(Integer): Tiempo de ejecución necesario para timeout.
- availableTimeout(Integer): Tiempo de ejecución disponible del proceso.
- taskList(List<Task>): Lista de tareas que contiene el proceso.
- permission(Permissions): Permiso que requiere el proceso.
- actualResource(Resource): Recurso actual del proceso.

Métodos:

Además de los métodos de acceso y modificación (set y get), se declararon los siguientes métodos:

- isAvailable(): Devuelve true si el estado del proceso es ‘Available’.
- isRunning(): Devuelve true si el estado del proceso es ‘Running’.
- getTaskById(Integer pos): Devuelve una tarea en base un índice dado.
- run(): Setea el status del proceso en ‘Running’.

- terminate(): Modifica el timeout del proceso y setea su estado en ‘AVAILABLE’.
- validateActionPermission(User user): Devuelve true si el rol del usuario tiene acceso a este proceso.
- validateResourcesPermission(User user): Devuelve true si el rol del usuario tiene acceso a este recurso.
- sortTaskListByExecutionTime(): Ordena la lista de tareas por tiempo de ejecución.

2.1.1.3.Clase Task

Atributos:

- name(String): Nombre de la tarea.
- executionTime(Integer): Tiempo de ejecución de la tarea.
- resource(Resource): Recurso que solicita la tarea.

Métodos:

Contiene únicamente métodos de acceso y modificación (setters y getters).

2.1.1.4.Clase Resource

Atributos:

- name(String): Nombre del recurso.
- status(Status): Estado del proceso. Puede ser AVAILABLE, LOCKED o RUNNING.

Métodos:

- isAvailable(): Devuelve true si el status es ‘AVAILABLE’.
- equals(): Redefinimos el método equals para realizar comparaciones.

2.1.1.5.Clase User

Atributos:

- name(String): Nombre del usuario.
- password(String): Contraseña del usuario.
- role(Role): Rol que posee el usuario.

Métodos:

Contiene únicamente métodos de acceso y modificación (getters y setters).

2.1.1.6.Clase Role

Atributos:

- name(String): Nombre del rol.
- permissionActionList(List<Permissions>):
- permissionResourceList(List<Resource>):

Métodos:

Contiene únicamente métodos de acceso y modificación (getters y setters).

2.1.2. Funciones

A continuación se detallan todas las funcionalidades del programa.

Se cuenta con un menú principal donde el usuario elige qué simulación ejecutar. Se tiene un método por cada una de estas opciones. Además, hay métodos definidos que operan en conjunto para permitir el correcto funcionamiento de las simulaciones.

2.1.2.1.Menú principal

Al iniciar el programa, se muestra en pantalla un menú con 8 opciones; las primeras 7 corresponden cada una a un caso de prueba distinto. Estas opciones ejecutan cada una su método correspondiente, donde se realiza la simulación. La opción 8 es para salir del menú y finalizar la ejecución del programa.

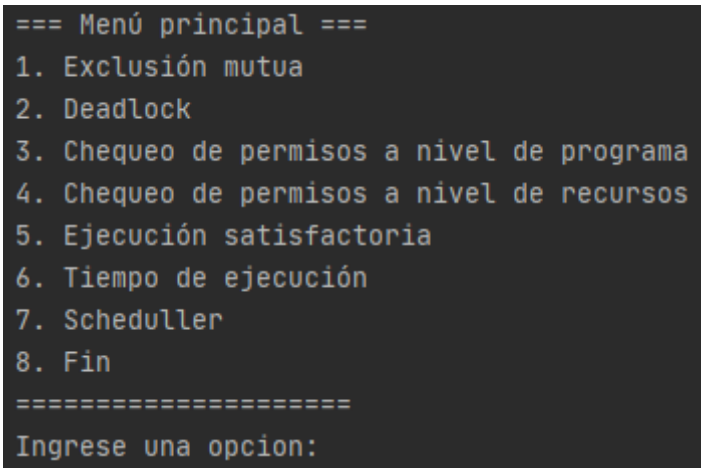


Imagen del menú principal

Cada vez que se selecciona una opción, se realiza la simulación y se vuelve a mostrar el menú para que el usuario pueda continuar seleccionando opciones.

2.1.2.2.Exclusión mutua

En este caso, se ejecuta la exclusión mutua y se puede ver en el log cómo el proceso ‘Imprimir documento Word’ e ‘Imprimir documento Excel’ comienzan a ejecutarse. El sistema va ejecutando las tareas pidiendo acceso a los recursos que las mismas solicitan desde el usuario Santiago. Llega un momento que dos tareas solicitan la impresora, por lo que una de las dos queda en estado de espera hasta que la impresora se libere, para luego ejecutarse y finalizar correctamente. En este caso podemos apreciar claramente como se hace uso de la exclusión mutua y además que cada recurso es pedido, usado y devuelto.

```
Ingrese una opcion:
4
=== Exclusión mutua iniciada ===
Se crea el proceso: imprimir documento, referido al programa: Word, pertenenciente al usuario: Santiago.
Se crea el proceso: imprimir documento, referido al programa: Excel, pertenenciente al usuario: Santiago.
Se crea el proceso: ver video, referido al programa: Video Player, pertenenciente al usuario: Santiago.
Empezando ejecución programa: Video Player, por parte del usuario: Santiago.
Empezando ejecución programa: Word, por parte del usuario: Santiago.
Empezando ejecución programa: Excel, por parte del usuario: Santiago.
Usuario: Santiago, pide acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Santiago, obtiene acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Ejecutando tarea: visualizar video, por el usuario: Santiago, en el marco del programa: video player.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: leer archivo, en el marco del programa: Word.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: leer archivo, en el marco del programa: Word.
Ejecutando tarea: leer archivo, por el usuario: Santiago, en el marco del programa: word.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: leer archivo, en el marco del programa: Excel.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: leer archivo, en el marco del programa: Excel.
Ejecutando tarea: leer archivo, por el usuario: Santiago, en el marco del programa: excel.
El proceso: imprimir documento, devolvió el recurso: Ram, y terminó de ejecutar la tarea: leer archivo, en el marco del programa: Word.
El proceso: imprimir documento, devolvió el recurso: Ram, y terminó de ejecutar la tarea: leer archivo, en el marco del programa: Excel.
El proceso: ver video, devolvió el recurso: Monitor, y terminó de ejecutar la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Santiago, pide acceso al recurso: Printer, para la tarea: imprimir archivo, en el marco del programa: Word.
Usuario: Santiago, obtiene acceso al recurso: Printer, para la tarea: imprimir archivo, en el marco del programa: Word.
Ejecutando tarea: imprimir archivo, por el usuario: Santiago, en el marco del programa: word.
Usuario: Santiago, pide acceso al recurso: Printer, para la tarea: imprimir archivo, en el marco del programa: Excel.
Usuario: Santiago, no puede acceder al recurso: Printer, en el marco del programa: Excel, dado que el mismo se encuentra en uso.
El proceso: imprimir documento, devolvió el recurso: Printer, y terminó de ejecutar la tarea: imprimir archivo, en el marco del programa: Word.
Usuario: Santiago, pide acceso al recurso: Printer, para la tarea: imprimir archivo, en el marco del programa: Excel.
Usuario: Santiago, obtiene acceso al recurso: Printer, para la tarea: imprimir archivo, en el marco del programa: Excel.
Ejecutando tarea: imprimir archivo, por el usuario: Santiago, en el marco del programa: excel.
El proceso: imprimir documento, devolvió el recurso: Printer, y terminó de ejecutar la tarea: imprimir archivo, en el marco del programa: Excel.
=== Exclusión mutua finalizada ===
```

2.1.2.3.Deadlock

En este método se ejecuta el caso de Deadlock, se ejecutan tres procesos: imprimir documento Word, imprimir documento Excel, ver video. Se observa que se produce un deadlock dado que, dos procesos quedan llamando al mismo recurso entre sí y uno de ellos está en ejecución, en este caso validamos estados a nivel de proceso y no de recurso. El recurso que quieren acceder es la impresora y cómo se aprecia en (2.1.4 Recursos), este recurso no es de acceso compartido entonces da error el chequeo de si está en estado disponible. Al dar deadlock devolvemos los recursos que estábamos utilizando y terminamos el proceso que abrimos y dio deadlock dejando al sistema en un estado disponible (todos los procesos y recursos están en estado disponible).

```
Ingrese una opcion:
2
=== Deadlock iniciado ===
Se crea el proceso: Imprimir documento word, referido al programa: Word, pertenenciente al usuario: Santiago.
Se crea el proceso: Imprimir documento excel, referido al programa: Excel, pertenenciente al usuario: Santiago.
Se crea el proceso: ver video, referido al programa: Video Player, pertenenciente al usuario: Santiago.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: Leer archivo word, en el marco del programa: Word.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: Leer archivo word, en el marco del programa: Word.
Ejecutando tarea: leer archivo word, por el usuario: Santiago, en el marco del programa: word.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: Leer archivo excel, en el marco del programa: Excel.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: Leer archivo excel, en el marco del programa: Excel.
Ejecutando tarea: leer archivo excel, por el usuario: Santiago, en el marco del programa: excel.
Usuario: Santiago, pide acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Santiago, obtiene acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Ejecutando tarea: visualizar video, por el usuario: Santiago, en el marco del programa: video player.
El proceso: Imprimir documento word, devolvió el recurso: Ram, y terminó de ejecutar la tarea: Leer archivo word, en el marco del programa: Word.
El proceso: Imprimir documento excel, devolvió el recurso: Ram, y terminó de ejecutar la tarea: Leer archivo excel, en el marco del programa: Excel.
El proceso: ver video, devolvió el recurso: Monitor, y terminó de ejecutar la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Santiago, pide acceso al recurso: Printer, para la tarea: Imprimir archivo word, en el marco del programa: Word.
Usuario: Santiago, obtiene acceso al recurso: Printer, para la tarea: Imprimir archivo word, en el marco del programa: Word.
Ejecutando tarea: imprimir archivo word, por el usuario: Santiago, en el marco del programa: word.
Hubo un error de deadlock al intentar ejecutar el proceso: Imprimir documento excel, en el marco del programa: Excel.
El proceso: Imprimir documento word, devolvió el recurso: Printer, y terminó de ejecutar la tarea: Imprimir archivo word, en el marco del programa: Word.
El proceso: imprimir documento excel, no pudo ejecutar la tarea: imprimir archivo excel, por deadlock, debido a esto fue cancelado.
=== Deadlock finalizado ===
```

2.1.2.4.Chequeo de permisos por programa

En esta opción el usuario Roberta la cual tiene el rol (Guest) solicita ejecutar el proceso ‘Imprimir documento word’, sin poder ejecutarlo por falta de permisos dado que el programa tiene permisos de ‘TO_PRINT’ y este rol como se observa en (2.1.4 Recursos) no tiene acceso a este permiso entonces finaliza sin poder ejecutar la tarea ‘Imprimir archivo word’, dejando el estado del proceso en disponible nuevamente.

```
Ingrese una opcion:
3
=== Chequeo de permisos por programa iniciado ===
Empezando ejecución programa: Word, por parte del usuario: Roberta.
El usuario: Roberta, no tiene permisos sobre el proceso: Imprimir documento word, en el marco del programa: Word.
El proceso: imprimir documento word, no pudo ejecutar la tarea: imprimir archivo word, por permisos, debido a esto fue cancelado.
=== Chequeo de permisos por programa finalizado ===
```

2.1.2.5. Chequeo de permisos por recurso

En esta opción el usuario Pedro la cual tiene el rol (Admin) intenta ejecutar el proceso ‘Sacar y guardar foto’ no pudiéndo hacerlo por falta de permisos sobre el recurso ‘Camera’ dado que el permiso que tiene este rol el cual es ‘READ’. Para finalizar, se cancela el proceso por no poder ejecutar la tarea ‘Sacar foto’.

```
Ingrese una opcion:
4
=== Chequeo de permisos por recurso iniciado ===
Empezando ejecución programa: WebCamToy, por parte del usuario: Pedro.
El usuario: Pedro, no tiene permisos sobre el recurso: Ram, al ejecutar el proceso: Sacar y guardar foto, en el marco del programa: WebCamToy.
El proceso: sacar y guardar foto, no pudo ejecutar la tarea: sacar foto, por permisos, debido a esto fue cancelado.
=== Chequeo de permisos por recurso finalizado ===
```

2.1.2.6. Ejecución satisfactoria

Se realiza la ejecución del proceso Reproducir video de forma exitosa. El usuario Roberta solicita acceso a los recursos necesarios para ejecutar cada tarea, el sistema valida que los tenga, se ejecutan las tareas y finaliza el proceso sin problemas. Siempre en cada tarea se pide se usa y se devuelven los recursos de forma satisfactoria.

```
Ingrese una opcion:
5
=== Ejecución satisfactoria iniciando ===
Se crea el proceso: reproducir video, referido al programa: Video Player, pertenenciente al usuario: Roberta.
Empezando ejecución programa: Video Player, por parte del usuario: Roberta.
Usuario: Roberta, pide acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Roberta, obtiene acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Ejecutando tarea: visualizar video, por el usuario: Roberta, en el marco del programa: video player.
El proceso: reproducir video, devolvió el recurso: Monitor, y terminó de ejecutar la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Roberta, pide acceso al recurso: Speakers, para la tarea: escuchar video, en el marco del programa: Video Player.
Usuario: Roberta, obtiene acceso al recurso: Speakers, para la tarea: escuchar video, en el marco del programa: Video Player.
Ejecutando tarea: escuchar video, por el usuario: Roberta, en el marco del programa: video player.
El proceso: reproducir video, devolvió el recurso: Speakers, y terminó de ejecutar la tarea: escuchar video, en el marco del programa: Video Player.
=== Ejecución satisfactoria finalizando ===
```

2.1.2.7.Timeout

Se simula una ejecución que da error por timeout al ejecutar la tarea ‘Escuchar video’, pero luego vuelve a ejecutarla para finalizar correctamente. Esto lo logramos gracias a que cada tarea tiene su executionTime(tiempo de ejecución) y el proceso tiene un executionTimeout(tiempo en el que se acaba su ejecución), entonces antes de ejecutar cada tarea preguntamos si nos queda tiempo disponible para poder ejecutar la misma, en caso de que no damos un error que es el que sea aprecia en la imagen y en el código en caso de ejecutarlo.

```
Ingrese una opcion:
6
=== Timeout iniciando ===
Se crea el proceso: reproducir video, referido al programa: Video Player, pertenenciente al usuario: Roberta.
Empezando ejecución programa: Video Player, por parte del usuario: Roberta.
Usuario: Roberta, pide acceso al recurso: Monitor, para la tarea: ver video, en el marco del programa: Video Player.
Usuario: Roberta, obtiene acceso al recurso: Monitor, para la tarea: ver video, en el marco del programa: Video Player.
Ejecutando tarea: ver video, por el usuario: Roberta, en el marco del programa: video player.
El proceso: reproducir video, devolvió el recurso: Monitor, y terminó de ejecutar la tarea: ver video, en el marco del programa: Video Player.
Hubo un error de timeout al intentar ejecutar la tarea: escuchar video, del proceso: reproducir video, ejecutado por el usuario: roberta, en el marco del programa video player.
Usuario: Roberta, pide acceso al recurso: Speakers, para la tarea: escuchar video, en el marco del programa: Video Player.
Usuario: Roberta, obtiene acceso al recurso: Speakers, para la tarea: escuchar video, en el marco del programa: Video Player.
Ejecutando tarea: escuchar video, por el usuario: Roberta, en el marco del programa: video player.
El proceso: reproducir video, devolvió el recurso: Speakers, y terminó de ejecutar la tarea: escuchar video, en el marco del programa: Video Player.
=== Timeout finalizando ===
```

2.1.2.8. Scheduller

A la hora de realizar el scheduller, utilizamos la planificación por tiempo más corto ; esto lo logramos dado que nuestras tareas tienen un executionTime, entonces a nivel de proceso podemos recorrer su lista de tareas y ordenarlas por tiempo de ejecución más corto(Esto dado que los recursos están hardcodeados en el código y sabemos que no se van a pisar entre ellos, porque es un simulador, si quisiéramos hacerlo de otra forma tendríamos que chequear aparte del tiempo de ejecución, chequear que las tareas que encolemos no tengan el mismo recurso). Se ejecutan dos procesos con sus respectivas tareas, y lo que hace el método es ordenar las tareas por tiempo de ejecución, para luego ejecutarlas en éste orden, cumpliendo con la planificación por tiempo más corto.

```
Ingrese una opcion:
7
=== Schedulling planificado por tiempo mas corto iniciando ===
Se crea el proceso: escribir documento, referido al programa: Bloc de notas, pertenenciente al usuario: Santiago.
Se crea el proceso: guardar como, referido al programa: Bloc de notas, pertenenciente al usuario: Santiago.
Se crea el proceso: ver video, referido al programa: Video Player, pertenenciente al usuario: Santiago.
Empezando ejecución programa: Bloc de notas, por parte del usuario: Santiago.
Empezando ejecución programa: Video Player, por parte del usuario: Santiago.
Empezando ejecución programa: Bloc de notas, por parte del usuario: Santiago.
Empezando ejecución programa: Video Player, por parte del usuario: Santiago.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: escribir documento, en el marco del programa: Bloc de notas.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: escribir documento, en el marco del programa: Bloc de notas.
Ejecutando tarea: escribir documento, por el usuario: Santiago, en el marco del programa: bloc de notas.
Usuario: Santiago, pide acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Usuario: Santiago, obtiene acceso al recurso: Monitor, para la tarea: visualizar video, en el marco del programa: Video Player.
Ejecutando tarea: visualizar video, por el usuario: Santiago, en el marco del programa: video player.
El proceso: ver video, devolvió el recurso: Monitor, y terminó de ejecutar la tarea: visualizar video, en el marco del programa: Video Player.
El proceso: escribir documento, devolvió el recurso: Ram, y terminó de ejecutar la tarea: escribir documento, en el marco del programa: Bloc de notas.
Usuario: Santiago, pide acceso al recurso: Ram, para la tarea: mostrando pantalla guardar como, en el marco del programa: Bloc de notas.
Usuario: Santiago, obtiene acceso al recurso: Ram, para la tarea: mostrando pantalla guardar como, en el marco del programa: Bloc de notas.
Ejecutando tarea: mostrando pantalla guardar como, por el usuario: Santiago, en el marco del programa: bloc de notas.
El proceso: guardar como, devolvió el recurso: Ram, y terminó de ejecutar la tarea: mostrando pantalla guardar como, en el marco del programa: Bloc de notas.
Usuario: Santiago, pide acceso al recurso: Hdd, para la tarea: guardar documento, en el marco del programa: Bloc de notas.
Usuario: Santiago, obtiene acceso al recurso: Hdd, para la tarea: guardar documento, en el marco del programa: Bloc de notas.
Ejecutando tarea: guardar documento, por el usuario: Santiago, en el marco del programa: bloc de notas.
El proceso: guardar como, devolvió el recurso: Hdd, y terminó de ejecutar la tarea: guardar documento, en el marco del programa: Bloc de notas.
=== Schedulling planificado por tiempo mas corto finalizando ===
```

2.1.3. Permisos

Tanto para la asignación de permisos a los procesos, como para la otorgación de permisos para los distintos roles de usuarios, decidimos crear una clase enum ‘Permissions’ con los tres permisos existentes: ‘READ’, ‘WRITE’ y ‘TO_PRINT’.

En la clase Role, se tiene un atributo permissionsActionList, que es una lista de tipo Permissions, donde se almacenan los permisos que el Role tiene asignado.

Por otro lado, en la clase Process contamos con un atributo permission, que define qué permiso debe tener el rol al que pertenece el usuario que ejecuta el proceso.

Tenemos métodos de validación de permisos a lo largo del código que realizan comparaciones basándose en los atributos mencionados anteriormente, de forma de poder validar distintas acciones, como por ejemplo la ejecución de un proceso por un usuario en específico.

2.1.4. Recursos:

Para realizar los casos de prueba, definimos seis recursos, de los cuales algunos son compartidos (pueden ser utilizados por más de un proceso o tarea simultáneamente) y otros no:

- Printer: no compartido.
- Camera: no compartido.
- Monitor: no compartido.
- Speakers: compartido.
- Hdd: compartido.
- Ram: compartido.

2.1.5. Roles y Usuarios

Para efectuar los casos de prueba, definimos tres roles y tres usuarios:

Roles:

- Admin: posee los permisos: ‘READ’, ‘WRITE’ y ‘TO_PRINT’ en todos los recursos.
- User: posee los permisos: ‘READ’ y ‘TO_PRINT’ en Monitor, Printer, Camera y Speakers.
- Guest: posee el permiso ‘READ’ en Monitor y Speakers.

Usuarios:

- Santiago, posee el rol Admin.
- Pedro, posee el rol User.
- Roberta, posee el rol Guest.