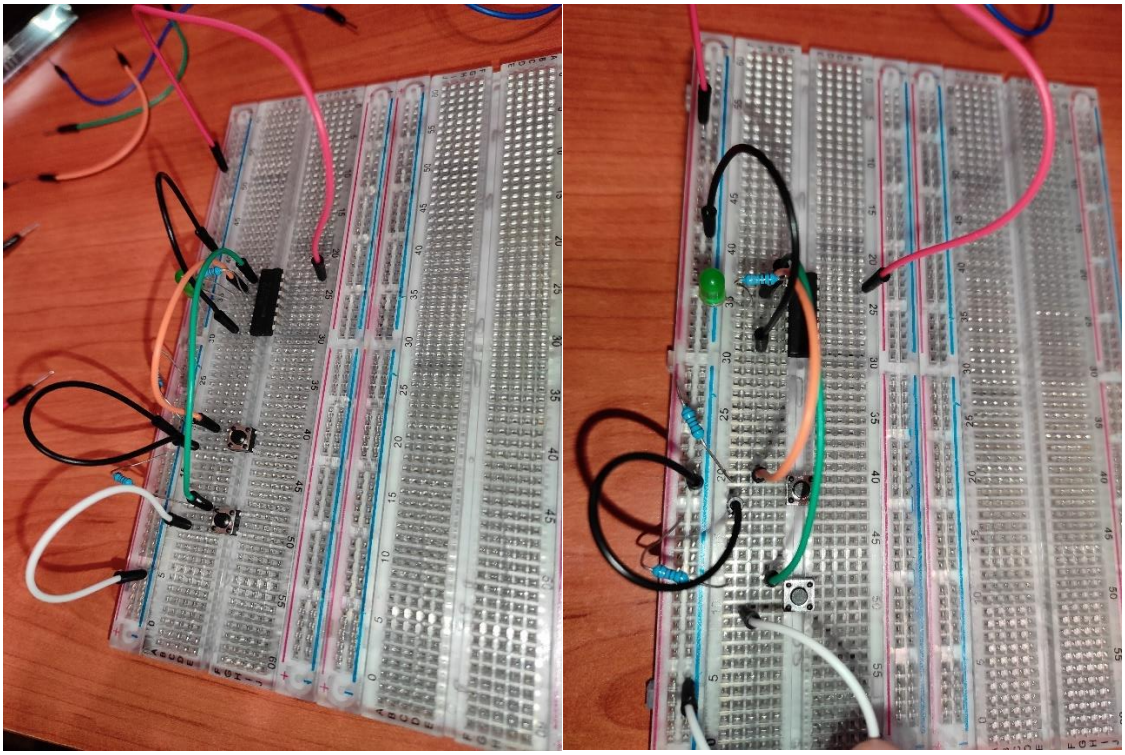
Memoria de laboratorio

Autores: Adrián Losada Álvarez y Gabriel Iglesias Sotelo

Fecha: 09/06/2022

Práctica 1:

Para la realización de esta práctica lo primero que hicimos (una vez comprendidos los componentes a usar y comprobado su correcto funcionamiento) fue realizar el montaje indicado en el guion, resultando el siguiente circuito:



Cuestiones:

a) *¿Qué propósito crees que tiene la resistencia en serie con el LED?*
La resistencia de 330 Ω colocada antes de la entrada del LED tiene como función limitar la corriente que le llegará al LED, evitando así que se dañe.

b) *¿Y las resistencias de 10k conectadas a la entrada? ¿Qué función tienen? Nota: Se suelen denominar resistencias pull-up o pull-down.*
Esta configuración de resistencias nos permite establecer voltajes de reposo para cuando el pulsador no está presionado, asegurando una lectura correcta.

c) *Analiza con el polímetro y el osciloscopio las señales de salida. Ilustra las conexiones que realizas para el proceso en la memoria de prácticas.*
(Debido a imprevistos con el osciloscopio no pudimos visualizar las señales de este)
##########

d) *Caracteriza los parámetros de tiempo de respuesta del circuito: tLH y tHL*
(Debido a imprevistos con el osciloscopio no pudimos visualizar las señales de este)

Video del funcionamiento del circuito:

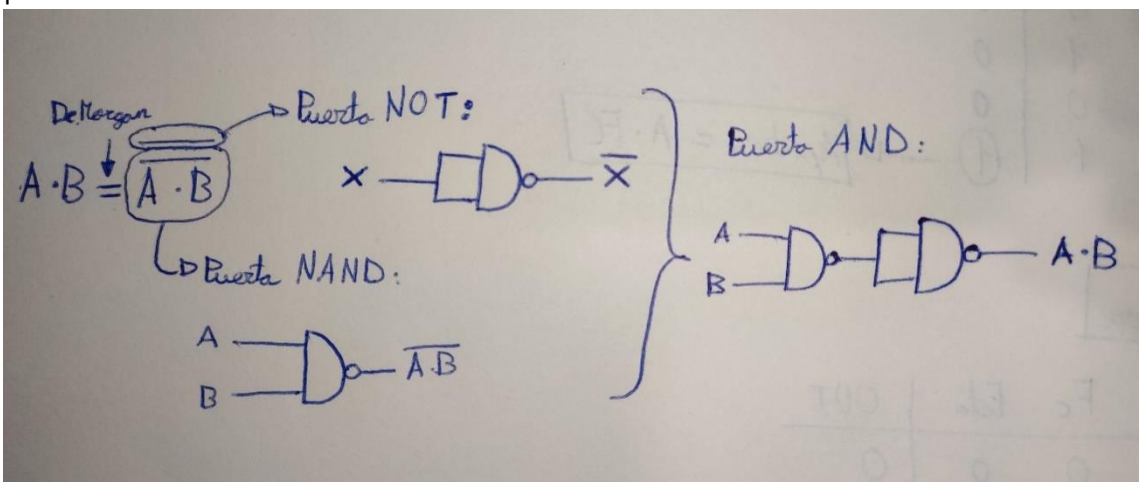
Video_practica_1.mp4

Enlace del video:
https://drive.google.com/file/d/18GQeAL1CX5fvARODO8FixSDv6me0pnUo/view?usp=sharing

Práctica 2:

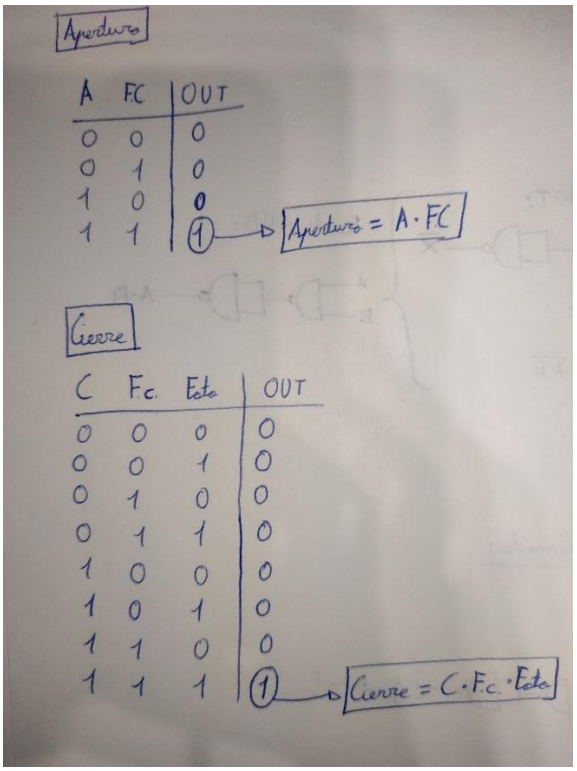
En la siguiente práctica realizamos los pasos a seguir descritos:

1) Diseñamos sobre el papel el circuito que realiza la función “AND” utilizando solamente puertas “NAND”:

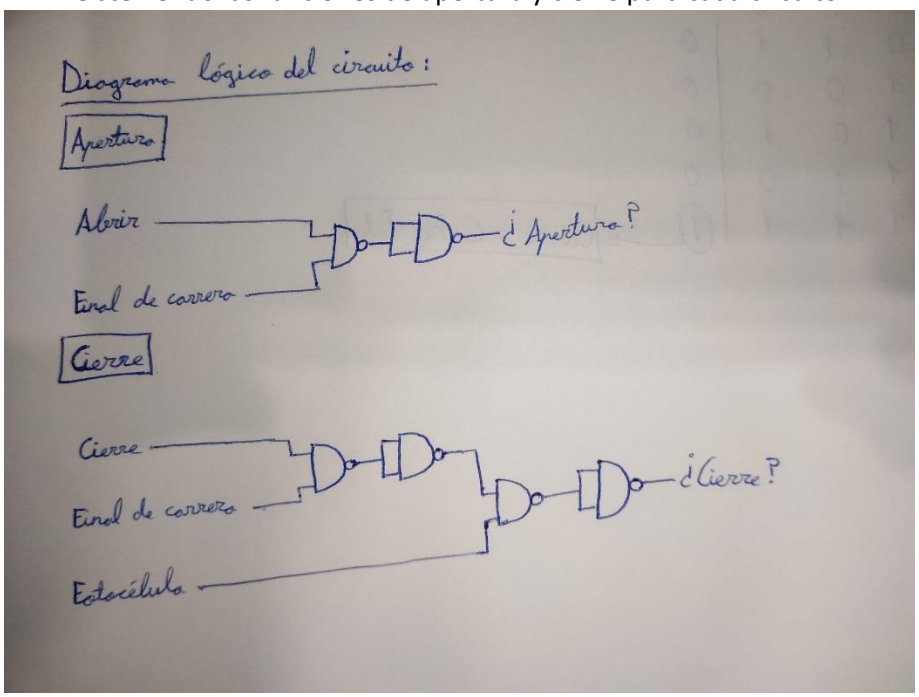


Para ello simplemente aplicamos De Morgan obteniendo así la función para finalmente juntar las puertas y obtener el equivalente a la puerta AND.

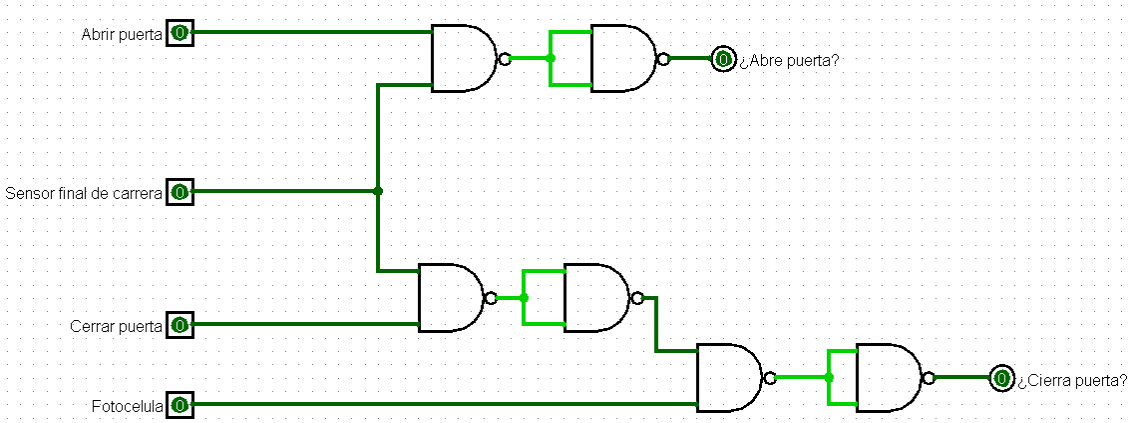
2) En este siguiente paso rellenaremos las tablas de verdad para la apertura y el cierre de la puerta:



Obteniendo las funciones de apertura y cierre para cada circuito:



3) Finalmente uniremos los 2 circuitos de apertura y cierre en uno común:



4) **Video del funcionamiento del circuito:**

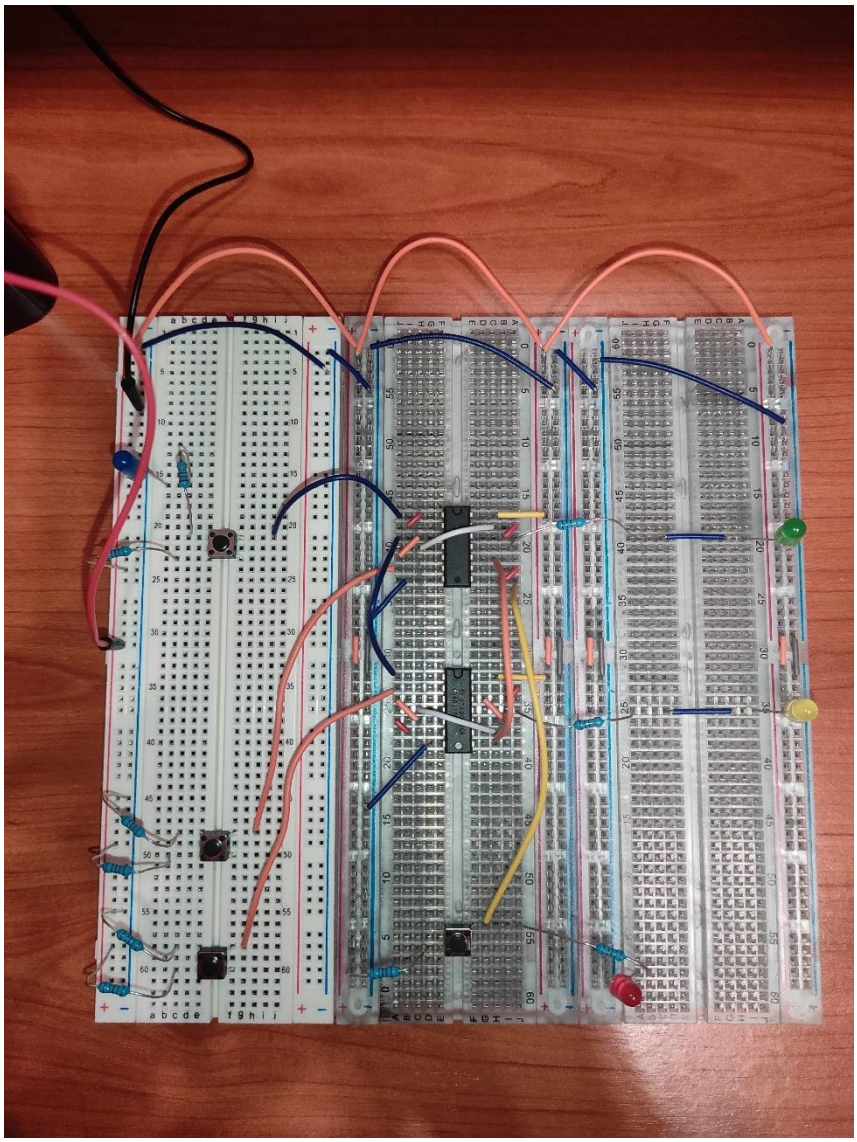
Video_practica_2.mp4

Enlace al video:
https://drive.google.com/file/d/1aGHx_u2zKaEoqqQ00aqhvjCL53GAqpiM/view?usp=sharing

NOTA: **En el video los LEDS representan:**

- **LED azul: final de carrera (común para apertura y cierre).**
- **LED verde: apertura de la puerta.**
- **LED amarillo: cierre de la puerta.**
- **LED rojo: activación de fotocélula.**

IMPORTANTE**: El circuito mostrado en el video no es exactamente igual que el del paso "3)", ya que en el simulado en Logisim las entradas "Sensor final de carrera" y "Fotocélula" no están invertidas, por lo que la puerta se abre/cierra cuando estas toman el valor HIGH, al contrario que en el video, que cuando se pulsan los botones de apertura/cierre a la vez que alguno de los sensores las salidas son LOW.**

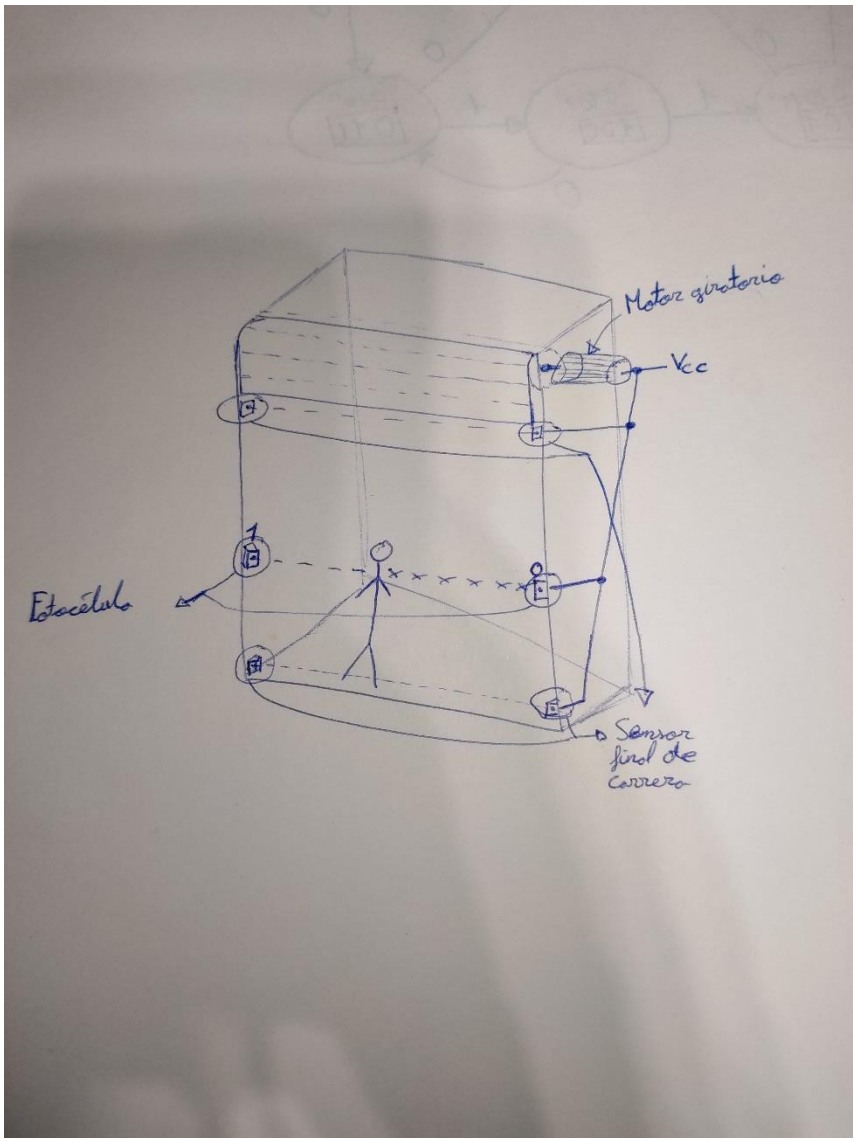


Cuestiones:

a) *¿Por qué crees que utilizamos los sensores de final de carrera como elementos N.O. (normalmente cerrados)?*
En estos dispositivos se utilizan dispositivos N.O. para que en caso de que hubiera alguna avería en dicho sensor, éste detuviera el proceso a realizar.

b) *Si la fotocélula sufriese avería y nunca diese señal quedaría automáticamente anulada la protección antiatrapamiento. ¿Qué podríamos hacer para mejorar esta situación?*
Para evitar esta situación deberíamos de utilizar el sensor de fotocélula como un elemento N.O., ya que, de esta forma si la fotocélula deja de recibir señal, la puerta se detendría inmediatamente.

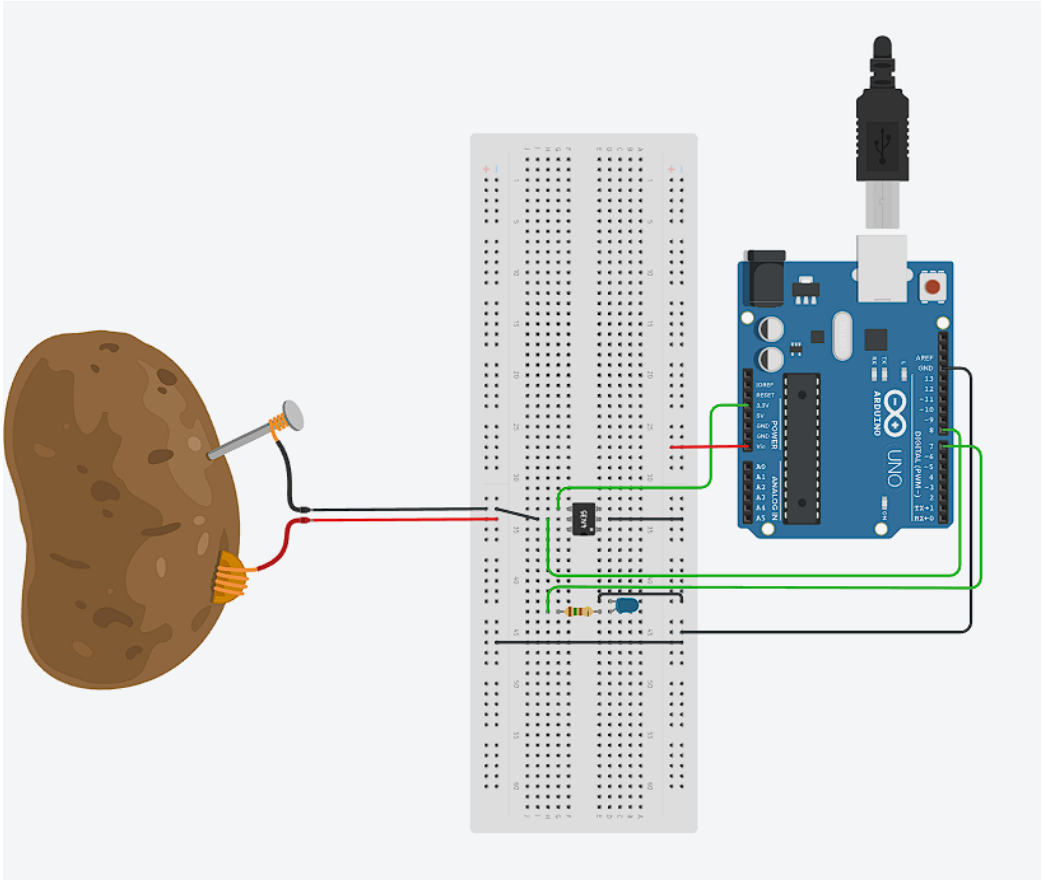
c) *Para llevar este sistema a la práctica que necesitaríamos añadir alrededor de este circuito de control. Esboza un esquema genérico de cómo podría ser el esquema completo (fuentes de alimentación, sensores, etc.).*



d) *Si los motores de la puerta fuesen de C.A, ¿qué haría falta añadir para poder actuar sobre ellos?*
Se necesitaría un conversor de corriente continua a corriente alterna.

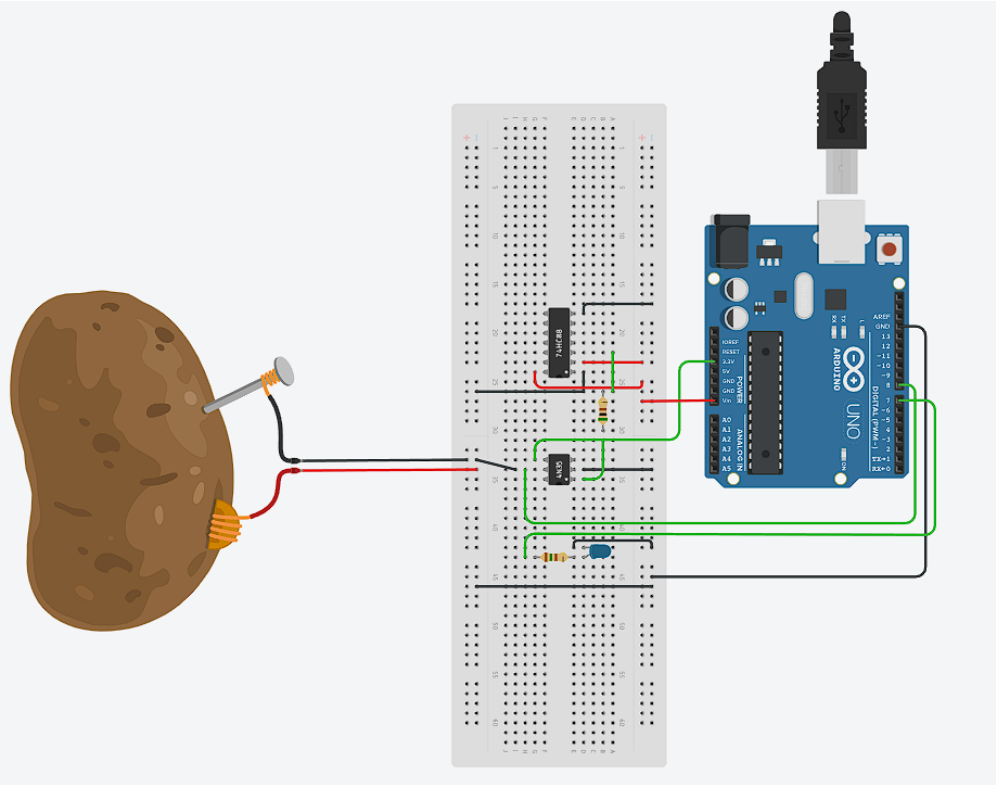
Práctica 3:

Esta práctica no nos fue posible llevarla a cabo en el laboratorio por lo que la simulamos en Tinkercad. En el primer apartado de esta práctica se pide hacer un circuito que conecte una salida de arduino a un circuito exterior con optoacoplador. El circuito creado es el siguiente :



Para este circuito necesitamos un optoacoplador, usamos el 4N35 ya que en tinkercad no hay el 4N33, pero su funcionamiento es el mismo . Para ver que el circuito funciona correctamente conectamos un led con su resistencia.

En el segundo apartado se nos pide acoplar una señal proveniente de las puertas lógicas AND a una entrada arduino a través de un optoacoplador. Para ello creamos el siguiente circuito:



Con la pila de patata somos capaces de generar exactamente 5V como señal de entrada. Como solo usamos una protoboard, con un cable conectamos las dos tierras de la placa.
Conectamos el emisor del optoacoplador a la entrada de 3.3V del arduino y el colector a un pin del arduino, en este caso el pin nº 8 . El cátodo va a tierra y el ánodo se conecta a la resistencia que se conecta en la salida 1 de la quad-AND. Respecto a la quad-AND, la entrada 1B y la de potencia van conectadas al positivo de la placa, mientras que la entrada 1A y la de tierra van a tierra. Por último tenemos el led, al igual que en el circuito anterior, el cual conectamos a un pin del arduino, en este caso el nº 7. La función del optoacoplador es
crear puentes entre señales de corriente alterna y circuitos lógicos , como el caso de nuestra quad-AND, los cuales utilizan un voltaje menor o igual a 5v, por esto usamos la pila de patata, para lograr esos 5V.

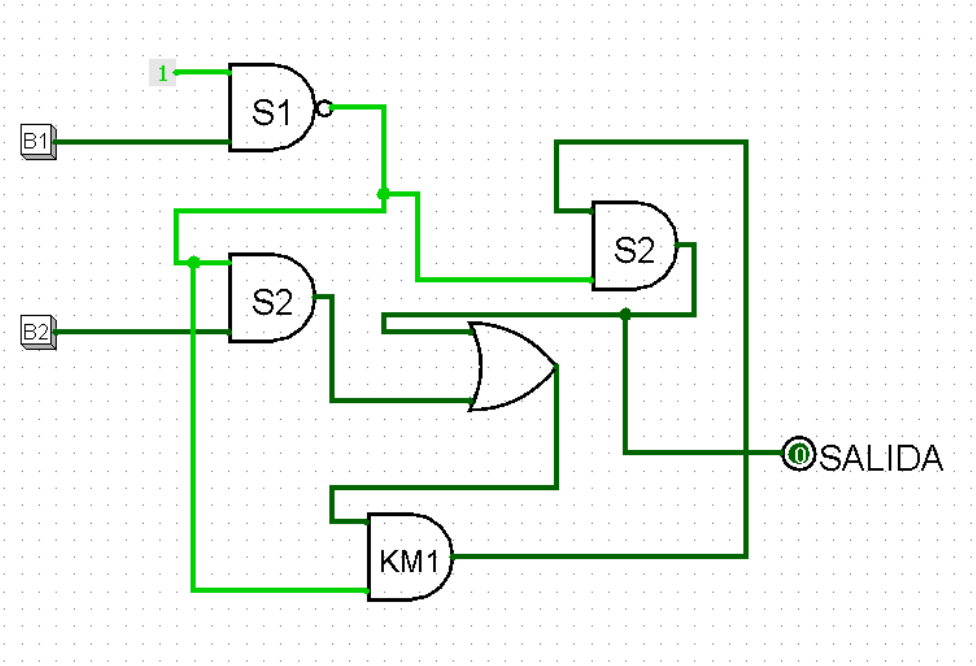
***Cuestiones*:**

a) El circuito del apartado 1) ¿genera la misma señal o la invertida? Si quisiera generar el otro caso, ¿cómo debería modificar el circuito?
El circuito del apartado 1) si que genera la misma señal. Si se quisiera la señal invertida lo único que habría que hacer sería conectar una puerta NOT antes de la CARGA.

b) analizar efectos en la velocidad de respuesta de este acoplamiento. Para ello generar a la vez una salida de cierta frecuencia desde el arduino, actuar sobre una de las entradas de una puerta AND que tenga una señal siempre activa en la otra patilla y chequear el valor de las señales de salida de la puerta AND antes y después de un optoacoplador. Nota: Una vez más, se recomienda aislar a nivel de alimentaciones ambos lados del acoplamiento.
El código propuesto para el Arduino realiza un loop en el que los pulsos de cada señal son intermitentes con un retardo de 1000 milisegundos. Este bucle pone a estado HIGH, espera ese periodo de tiempo y luego cambia a está LOW para luego esperar otros 1000 milisegundos, esto se repite indefinidamente hasta que se detenga la ejecución.

Práctica 4:

En esta práctica se nos pide diseñar un circuito de marcha-paro con puertas lógicas cualesquiera. Nos dicen que la fórmula que ejecuta el circuito es la siguiente :
KM1 = NOT(S1) AND (S2 OR KM1) , por lo cual el circuito montado es el siguiente:



De este modo construimos el circuito para que funcione de esta manera, cuando se pulse B1 el circuito dará salida 0 y esto no cambiará hasta que se pulse B2, que dará salida 1, y así sucesivamente.

***Cuestiones*:**

- Explica con tus propias palabras el funcionamiento del circuito desarrollado y los principios en los que se basa. Un circuito similar en el aspecto de la realimentación es el concepto que se utiliza para construir elementos de memoria.

La explicación del funcionamiento del circuito es la ya comentada anteriormente, el circuito funciona con señales fijas, es decir, al pulsar B1 la salida es 0 siempre y al pulsar B2 se convierte en 1 siempre hasta que se vuelva a pulsar B1.

-Incluye un análisis del concepto de enclavamiento eléctrico.

De este modo podemos decir que el enclavamiento eléctrico es simplemente un dispositivo que controla la condición de estado de cierto mecanismo para habilitar o no un accionamiento, como un interruptor de luz de casa, por así decirlo.

-**Razona en qué medida se está construyendo un elemento capaz de retener un estado.**

Aunque no es una retención de estado perfecta como podría hacer un biestable tipo D, el cual se basa en el almacenaje de estados pues para eso está diseñado , con este circuito de marcha-paro sí que estamos reteniendo estados ya que mientras no se activa uno de los dos pulsadores la señal se queda fija en 0 o 1 y no varía hasta que pulsemos el otro pulsador, por lo que si, estamos reteniendo estados.

Práctica 5:

Esta práctica es únicamente simulada en Logisim, a continuación, explicaremos el funcionamiento de esta:

Primeramente, nada más abrir el archivo .circ veremos 2 displays 7 segmentos de 3 dígitos, el superior corresponderá a la temperatura actual y el inferior a la temperatura de consigna (temperatura a elegir), para modificar estos dos valores podremos hacerlo a través de los botones situados a la izquierda: en primer lugar pulsaremos el botón START/RESET para empezar con los valores iniciales, posteriormente podremos aumentar o decrementar el número mostrado en los displays en 0.5 alternando entre la suma y la resta (botón "Suma/Resta", un LED indicará si se está sumando o restando), y pulsando el botón con etiqueta "0.5" para agregárselo/extraérselo al número visualizado.

Siempre que se quiera se puede volver a reiniciar el circuito pulsando los dos botones "START/RESET".

Por otro lado, un LED nos indicará si el bombeo de vapor está activado siempre y cuando este esté encendido (botón "ON/OFF"). Todos los ajustes de temperatura tendrán que realizarse a mano.

Práctica 6:

Tareas:

1) *Probar, para empezar, exclusivamente el módulo de testeo (panel izquierdo). Escribir código en él utilizando funciones del sistema como $display, asignaciones a variables y retardos de tiempo. Revisar el ejemplo https://www.edaplayground.com/s/example/352. Verifica que se graban los diseños (opción "save").*
   Código empleado:

   (testbench.sv)

```
// Ejemplos de displays.
module test_display;

  reg [8*200:0] mi_cadena;
```

```
 reg [15:0] mi_numero;
 real mi_tiempo;

 initial begin
  $display("Hola mundo!");
  mi_cadena = "Esta es mi cadena.";
  $display("Mi cadena: %0s", mi_cadena);

  mi_numero = 8'h1a;
  $display("Mi numero en decimal: %0d", mi_numero);
 end

endmodule
```

2) *Abrir y analizar el ejemplo incorporado del flip-flop tipo D incluído en los ejemplos EDA Playground. Entender para que se usa $dumpfile("dump.vcd"); (ver documentación incorporada en EDA) Fijarte en la estrategia que se usa para tener un "log" adecuado por pantalla llamando a una "task" y así verificar el funcionamiento del diseño. Probar la salida de formas de onda usando el módulo EPWAVE.*

$dumpfile("dump.vcd"); se usa para abrir un fichero para almacenar la información de las ondas del circuito.

Código empleado:

(testbench.sv)

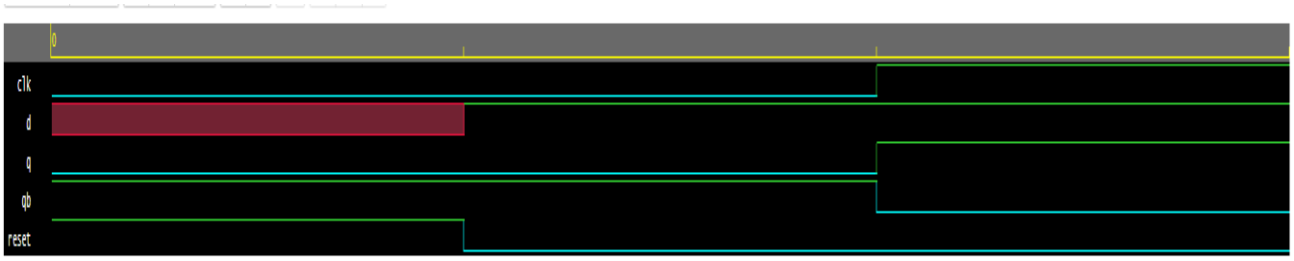
```
module test;

 reg clk;
 reg reset;
 reg d;
 wire q;
 wire qb;

 dff DFF(.clk(clk), .reset(reset), .d(d), .q(q), .qb(qb));

 initial begin
  //$dumpfile("dump.vcd"); abre un archivo para almacenar la información de la
forma de onda.
  //Lo que nos permite usar EPWave para visualizar la salida.
  $dumpfile("dump.vcd");
  $dumpvars(1);

  $display("Reset flop.");
  clk = 0;
  reset = 1;
  d = 1'bx;
  display;
```

```
    $display("Release reset.");
    d = 1;
    reset = 0;
    display;

    $display("Toggle clk.");
    clk = 1;
    display;
  end

  //Los task funcionan de manera equivalente a una función en un lenguaje de
programación.
  //En este caso se utiliza para obtener una salida determinada cada vez que se
llama a "display".
  task display;
    #1 $display("d:%0h, q:%0h, qb:%0h", d, q, qb);
  endtask

endmodule
```

(design.sv)

```
//Flip-flop tipo D
module dff (clk, reset, d, q, qb);
  input      clk;
  input      reset;
  input      d;
  output     q;
  output     qb;

  reg        q;

  assign qb = ~q;

  always @(posedge clk or posedge reset)
  begin
    if (reset) begin
      q <= 1'b0;
    end
    else begin
      q <= d;
    end
  end
endmodule
```

Si lo ejecutamos obtenemos las siguientes ondas:



Respectivamente:

Clk

d

q

qb

reset

Análisis:

Como podemos observar, inicialmente *d* toma el valor "x" cuanto el *reset* está en HIGH ya que, independientemente de la entrada (0 o 1) la salida de *q* va a ser 0. A continuación ponemos *reset* en LOW y la entrada *d* en HIGH, como podemos ver, *q* sigue siendo 0, esto se debe a que el circuito aún no ha tenido ningún flanco de subida por parte del *clk*. Finalmente, *clk* da un flanco de subida mientras *d* es 1 por lo tanto *q* también será 1 y su inversa (*qb*) 0.

3) *Modificar el código del ejemplo para hacer un flip-flop tipo JK y probar el resultado elaborando un testbench adecuado.*

   Código empleado:

   (testbench.sv)

```
module JK;
  reg reset;
  reg j;
  reg k;
  wire q;
  wire qb;
```

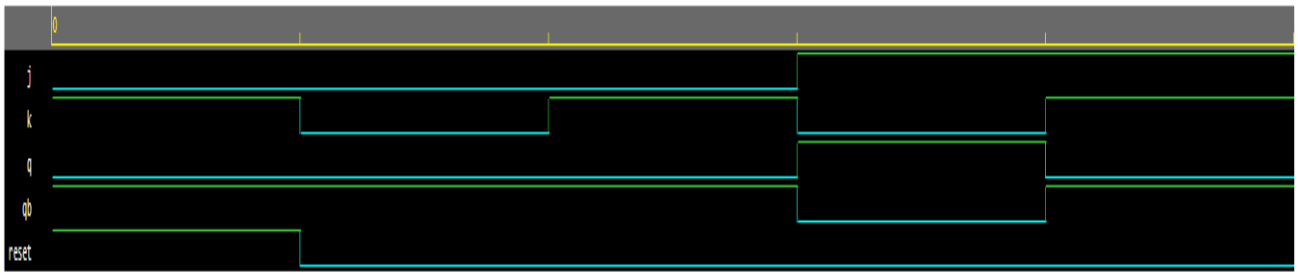
```
 jk JK(.reset(reset), .j(j), .k(k), .q(q), .qb(qb));

 initial begin
  $dumpfile("dump.vcd");
  $dumpvars(1);

  //Probamos todas la posibilidades del JK.
  $display("1");
    reset = 1;
  j = 0;
  k = 1;
  display;

  $display("2");
    reset = 0;
  j = 0;
  k = 0;
  display;

  $display("3");
  j = 0;
  k = 1;
  display;

  $display("4");
  j = 1;
  k = 0;
  display;

  $display("5");
  j = 1;
  k = 1;
  display;
 end

 task display;
  #1 $display("reset:%0h, j:%0h, k:%0h, q:%0h, qb:%0h", reset,j,k,q,qb);
 endtask

endmodule
```

(design.sv)

```
// Biestable flip-flop JK
module jk (reset, j, k, q, qb);
 input     reset;
 input     j;
 input     k;
 output    q;
```

```
    output     qb;

    reg        q;
    assign qb = ~q;

    always @(posedge j or posedge k or posedge reset)
    begin
      //Asignamos el valor de "q" dependiendo de las entradas.
      if (reset) begin
         q <= 0;
      end
      else if (j == 0 && k == 0) begin
        q <= q;
      end
      else if (j == 0 && k == 1) begin
         q <= 0;
      end
      else if (j == 1 && k == 0) begin
         q <= 1;
      end
      else if (j == 1 && k == 1) begin
         q <= qb;
      end
    end
endmodule
```

Ejecutándolo obtenemos las siguientes ondas:



Respectivamente:

j

k

q

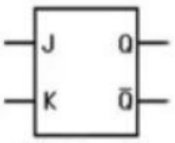
qb

reset

Análisis:

Siguiendo la tabla de verdad de los flip-flop's tipo JK:

J	K	Q_t	$\overline{Q_t}$
0	0	Q_{t-1}	$\overline{Q_{t-1}}$
0	1	0	1
1	0	1	0
1	1	$\overline{Q_{t-1}}$	Q_{t-1}



Flip flop J-K

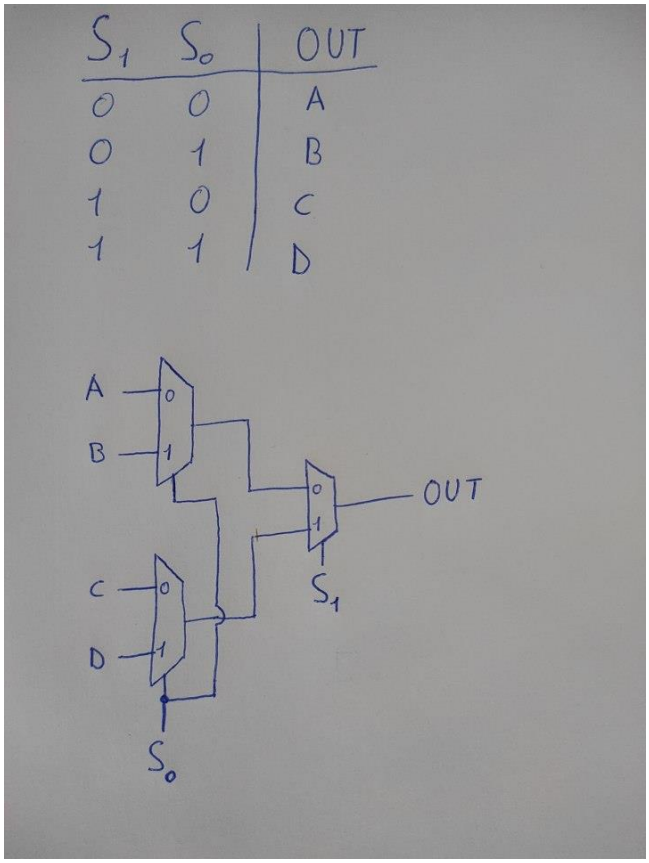
Podemos observar que, cuando *j* es 0 y *k* es 0 o 1, la salida *q* es siempre 0, ya que cuando *k* = 0 la salida es el valor anterior, es decir, *q* = 0 y cuando *k* = 1, el valor de *q* también es 0 ya que k actúa como un reset.

Por otro lado, cuando *j* = 1, si *k* = 0 la salida *q* será 1, mientras que si *k* = 1, la salida *q* será el valor anterior de *q* pero invertido.

Práctica 7:

Tareas:

1) *Diseñar sobre el papel, a nivel de bloques, el circuito mux 4:1 a partir de bloques 2:1.*



2) *Crear y verificar en EDA un bloque funcional MUX 2:1. El bloque ha de ser genérico, con sus variables de entrada y salida, de forma que posteriormente pueda ser insertado como subcircuito en otras partes del diseño. Existe la opción de crearlo como task pero en este caso es preferible que simplemente sea un módulo separado. Nota: En un fichero pueden convivir varias definiciones de módulo. Aunque así es una práctica habitual definirlos en ficheros diferentes. Si se desea pueden añadirse más módulos a un proyecto EDA, añadiendo pestañas tanto en el lado de diseño como en el de test.*

Código empleado:

(testbench.sv)

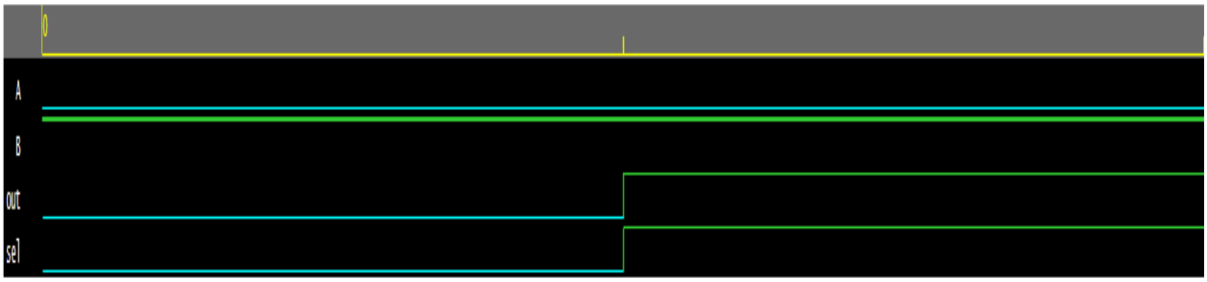
```
module mux2_1;
 reg A, B, sel;
 wire out;

 mux2x1 mi_mux2(.A(A), .B(B), .sel(sel), .out(out));

 initial begin
  $dumpfile("dump.vcd");
  $dumpvars(1);

  A = 0;
  B = 1;
```

```
  for(int i=0;i<2;i=i+1) begin
    sel = i;
    print;
  end
 end

 task print;
  #1 $display("Selector=%d, A=%d, B=%d, salida=%d", sel, A, B, out);
 endtask

endmodule
```

(design.sv)

```
module mux2x1(A, B, sel, out);
 input A;
 input B;
 input sel;
 output out;

 assign out = (A&(!sel))|(sel&B);

endmodule
```

Si lo ejecutamos obtenemos las siguientes ondas:



Respectivamente:
A
B
sut
sel

Análisis:
Siendo A = 0 y B = 1, podemos comprobar que mientras el bit de selección (*sel*) está en LOW la salida va a ser igual a *A* y cuando *sel* cambia a HIGH la salida va a ser *B*.

3) *Crear un segundo diseño donde probaremos el módulo 4:1. Ver las notas generales de apoyo a esta práctica más abajo para conocer el procedimiento de instanciado. Generar un fichero de ondas y analizarlo para asegurar el buen funcionamiento del mismo.*

Código empleado:

(testbench.sv)

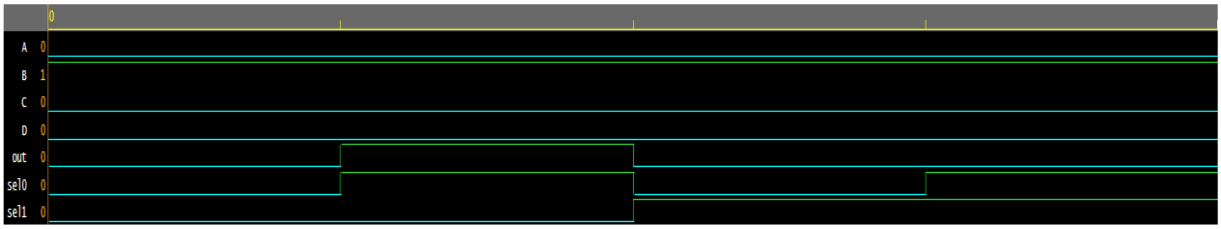
```
module mux4_1;
 reg A, B, C, D, sel1, sel0;
 wire out;

 mux4x1 mi_mux4(.A(A), .B(B), .C(C), .D(D), .sel1(sel1), .sel0(sel0), .out(out));

 initial begin
  $dumpfile("dump.vcd");
  $dumpvars(1);

  A = 0;
  B = 1;
  C = 0;
  D = 0;

  for(int i=0;i<4;i=i+1) begin
   {sel1, sel0} = i;
   print;
  end
 end

 task print;
  #1 $display("Selector=%d%d, A=%d, B=%d, C=%d, D=%d, salida=%d", sel1, sel0, A, B,
C, D, out);
 endtask

endmodule
```

(design.sv)

```
module mux2x1(A, B, sel, out);
 input A;
 input B;
 input sel;
 output out;

 assign out = (A&(!sel))|(sel&B);

endmodule


module mux4x1(A, B, C, D, sel1, sel0, out);
 input A, B, C, D, sel1, sel0;
 output out;
 wire mux1, mux2;
```

```
  mux2x1 mux_1(.A(A), .B(B), .sel(sel0), .out(mux1));
  mux2x1 mux_2(.A(C), .B(D), .sel(sel0), .out(mux2));
  mux2x1 mux_3(.A(mux1), .B(mux2), .sel(sel1), .out(out));

endmodule
```

Si lo ejecutamos obtendremos el siguiente diagrama de ondas:



Respectivamente:

A = 0

B = 1

C = 0

D = 0

out

sel0

sel1

Análisis:

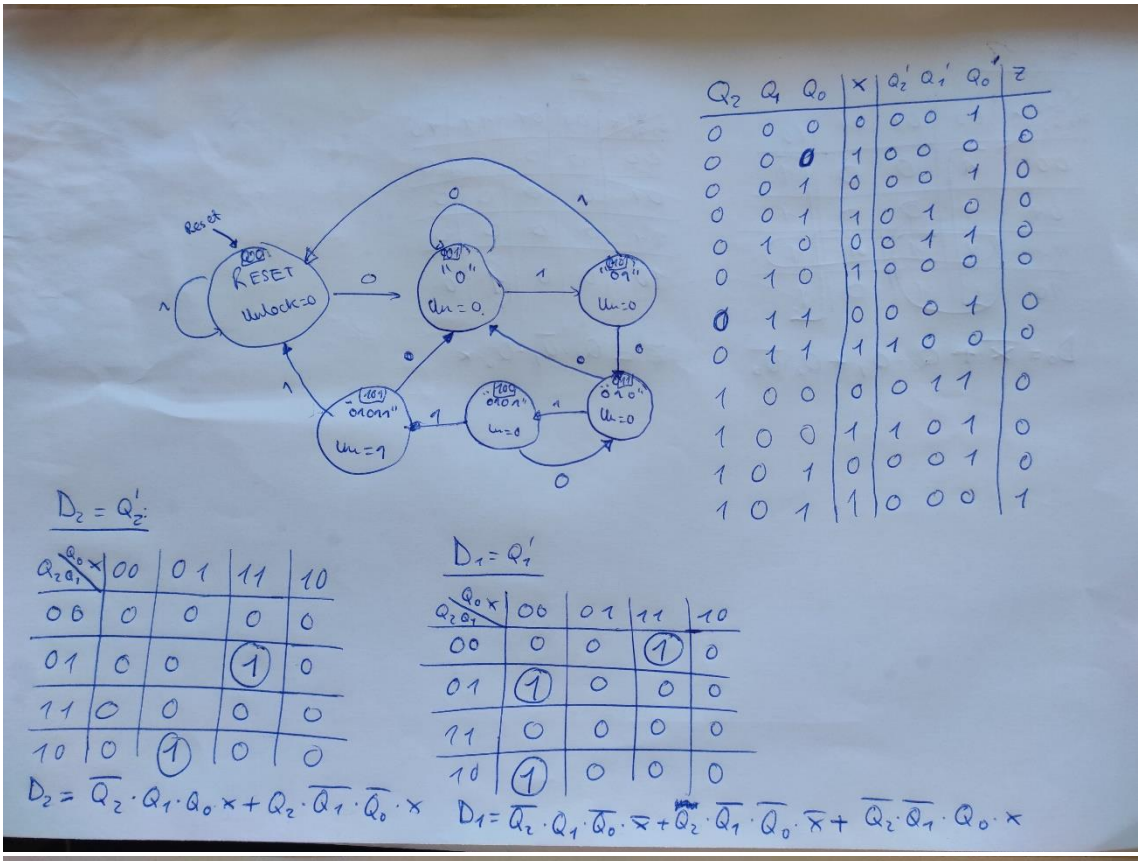
Dependiendo de los bits de selección (*sel1 y sel0*) obtendremos las salidas A, B, C o D:

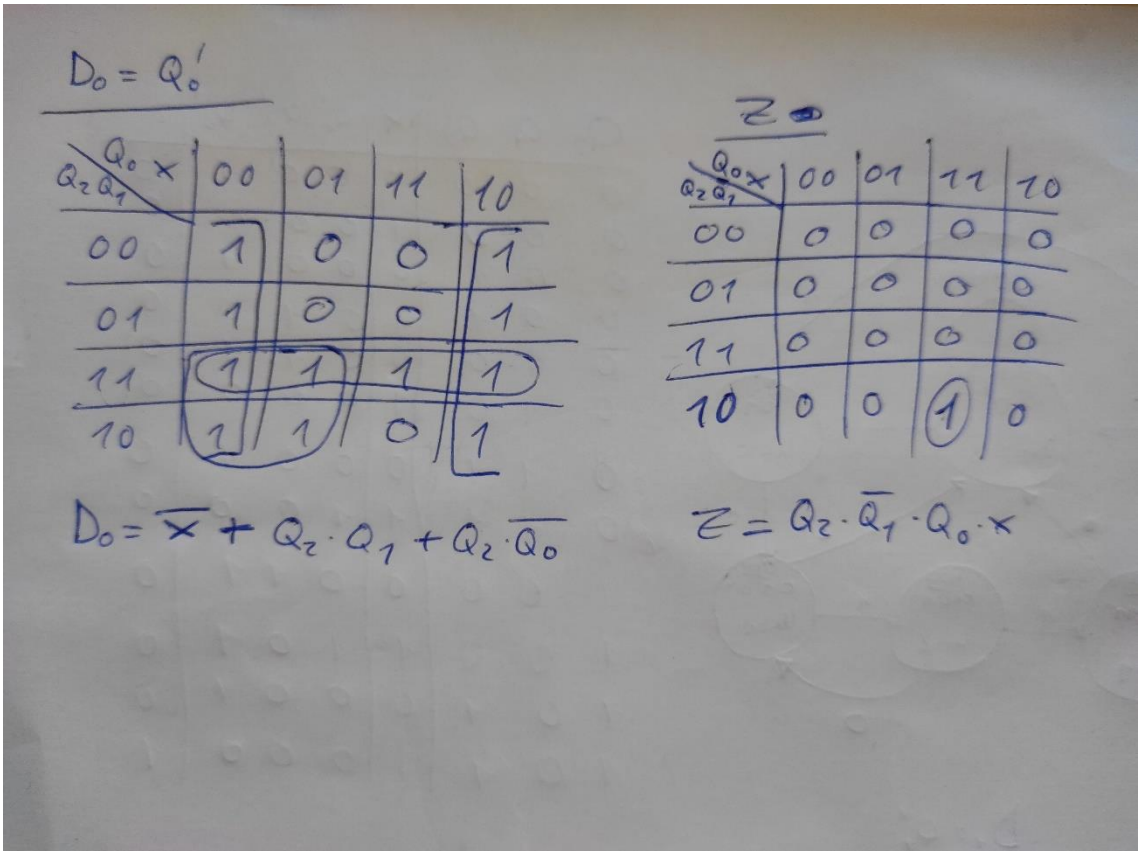
sel1	**sel0**	**out**
0	0	0
0	1	1
1	0	0
1	1	0

Práctica 8:

Tareas:

1) Analizar el ejemplo indicado y comprender su funcionamiento.



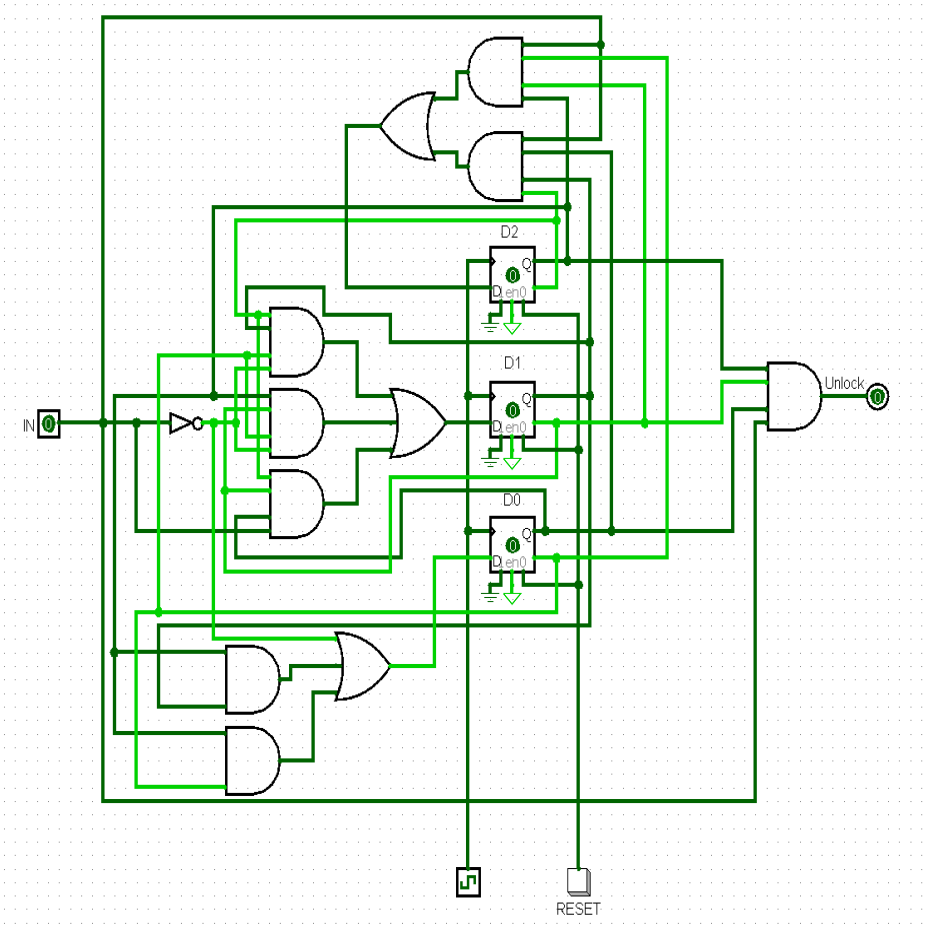


2) Como ejemplo vamos a construir una cerradura electrónica simplificada, capaz de abrir una puerta con un código introducido por teclado. Supondremos que el número de dígitos posibles del código es solamente 2 (un teclado binario que solo tiene las teclas 0 o 1), y que la longitud total del codigo es 5. Tambien tendremos una tecla RESET para borrar el código introducido en caso de error. Si se teclea el código correcto se dará un señal para abrir la puerta. Lo realizaremos para un código fijo. En la máquina de estados que se presenta aquí se utiliza el código 01011.

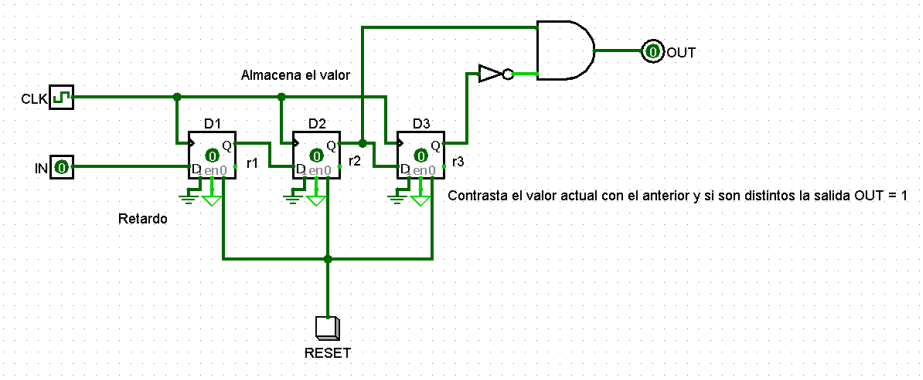
    a) Analizar el sistema secuencial a partir de la siguiente máquina de estados

    b) Esbozar la arquitectura del sistema con los bloques funcionales necesarios

    Pag. 3 / 422



c) El teclado debe funcionar de una manera tal que al mantener pulsado un botón no introduzca constantemente ese código. Esto requiere de algunos trucos a nivel electrónico. Se propone para ello el siguiente circuito. Probar su funcionamiento en Logisim e interpretar qué operación hace (incluir la explicación en la memoria de prácticas).



d) Una vez tengamos clara la arquitectura comenzar a diseñarlo en EDA Playground basándonos en el ejemplo que hemos analizado anteriormente, junto con su banco de pruebas.

Código empleado:

(testbench.sv)

```
//Creamos el modulo para probar el funcionamiento de la máquina.
module test;

 //Definimos las variables.
 reg clk, reset, in, contador;
 wire out;
 reg[4:0] secuencia;
 integer i;

 maquina_moore prueba(clk, reset, in, out);

 initial
   begin
     clk = 0;
     reset = 1;
     secuencia = 5'b11010; //Escribimos la secuencia de derecha a izquierda
     #5 reset = 0;
     print; //Mostramos el estado inicial (IDLE).

     //Recorremos la secuencia escrita, mostrando en cada paso el estado
actual, la entrada y la salida.
     for (i = 0; i <= 4; i = i + 1)
       begin
         in = secuencia[i];
         #2 clk = 1;
         #2 clk = 0;
```

```
        print;
      end
    end

  task print;
    $display("Nº teclas pulsadas = %0d Entrada = %b Estado actual = %b Salida =
%b", prueba.contador, in, prueba.estado, out);
  endtask

endmodule
```

(design.sv)

```
//Creamos el módulo para la máquina de Moore dada.
module maquina_moore(clk, reset, in, out);

  //Definimos las variables (cables) de entrada y salida.
  input clk, reset, in;
  output out;

  //Definimos las variables: contador de teclas, estado y salida.
  reg [7:0] contador = 0; //Variable que cuenta el número de teclas pulsadas
(capacidad de 8 bits).
  reg [2:0] estado;
  reg out;

  //Escribimos todos los posibles casos en cada estado y su respectiva salida.
  //Además en cada caso, independientemente de la entrada (in), sumamos 1
al contador de teclas pulsadas.
  always @(posedge clk, posedge reset)
    begin
      if (reset)
        begin
          contador = 0;
          estado <= 3'b000;
          out <= 0;
        end
      else
        begin
          case (estado)
            3'b000:
            begin
               contador = contador + 1;
              if (!in)
              begin
                estado <= 3'b001;
                out <= 0;
              end
              else
```

```
  begin
    estado <= 3'b000;
    out <= 0;
  end
end

3'b001:
  begin
    contador = contador + 1;
    if (in)
    begin
      estado <= 3'b010;
      out <= 0;
    end
    else
    begin
      estado <= 3'b001;
      out <= 0;
    end
  end

3'b010:
  begin
    contador = contador + 1;
    if (!in)
    begin
      estado <= 3'b011;
      out <= 0;
    end
    else
    begin
      estado <= 3'b000;
      out <= 0;
    end
  end

3'b011:
  begin
    contador = contador + 1;
    if (in)
    begin
      estado <= 3'b100;
      out <= 0;
    end
    else
    begin
      estado <= 3'b001;
      out <= 0;
```

```
            end
          end

        3'b100:
          begin
            contador = contador + 1;
            if (in)
            begin
              estado <= 3'b101;
              out <= 1;
            end
            else
            begin
              estado <= 3'b011;
              out <= 0;
            end
          end

        3'b101:
          begin
            contador = contador + 1;
            if (in)
            begin
              estado <= 3'b000;
              out <= 0;
            end
            else
            begin
              estado <= 3'b001;
              out <= 0;
            end
          end
      endcase
    end
  end

endmodule
```

Ejecutándolo nos mostrará la siguiente salida por terminal:

```
Nº teclas pulsadas = 0 Entrada = x Estado actual = 000 Salida = 0
Nº teclas pulsadas = 1 Entrada = 0 Estado actual = 001 Salida = 0
Nº teclas pulsadas = 2 Entrada = 1 Estado actual = 010 Salida = 0
Nº teclas pulsadas = 3 Entrada = 0 Estado actual = 011 Salida = 0
Nº teclas pulsadas = 4 Entrada = 1 Estado actual = 100 Salida = 0
Nº teclas pulsadas = 5 Entrada = 1 Estado actual = 101 Salida = 1
```