

# XILINX Y ARDUINO

GRUPO H

*Elena Del Río Galera  
Andrea López Recio*

*Alicia Gordo Azabal  
Yolanda Lillo Mata*

# RESUMEN DEL TRABAJO

→ Xilinx:

- Tablas de Verdad
- Diseño del circuito
- Síntesis e implementación del circuito
- Programación de la FPGA

# RESUMEN DEL TRABAJO

→ *Arduino:*

- Programación*
- Simulación en Tinkercad*
- Montaje en placa*

# TABLAS DE EXCEL

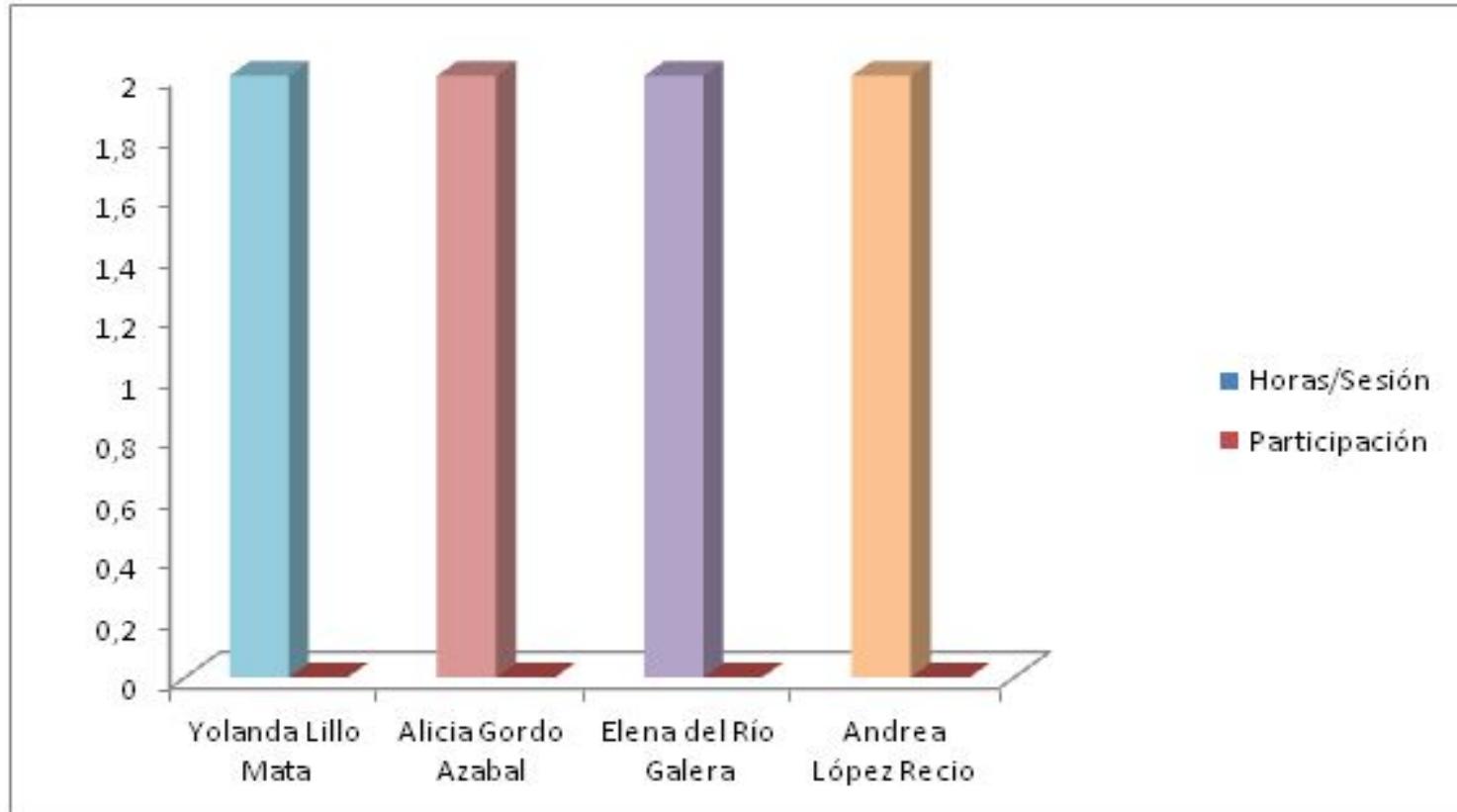
## 1) Xilinx

PARTICIPANTES	SESIÓN 1	SESIÓN 2	SESIÓN 3	SESIÓN 4	HORAS DE CADA SESIÓN	PORCENTAJE DE PARTICIPACIÓN EN CADA
Elena del Río Galera	Diseñador	Programador	Evaluador	Diseñador	2 horas	0,25% cada una
Andrea López Recio	Programador	Evaluador	Programador	Diseñador	2 horas	0,25% cada una
Alicia Gordo Azabal	Programador	Evaluador	Diseñador	Programador	2 horas	0,25% cada una
Yolanda Lillo Mata	Evaluador	Diseñador	Programador	Evaluador	2 horas	0,25% cada una
BLOQUES REALIZADOS	3,4,5 y 6	7,8 y 9	10,11,12 y 13	14 y 15		

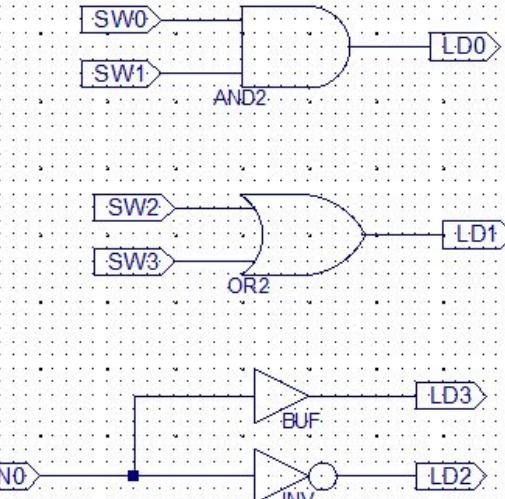
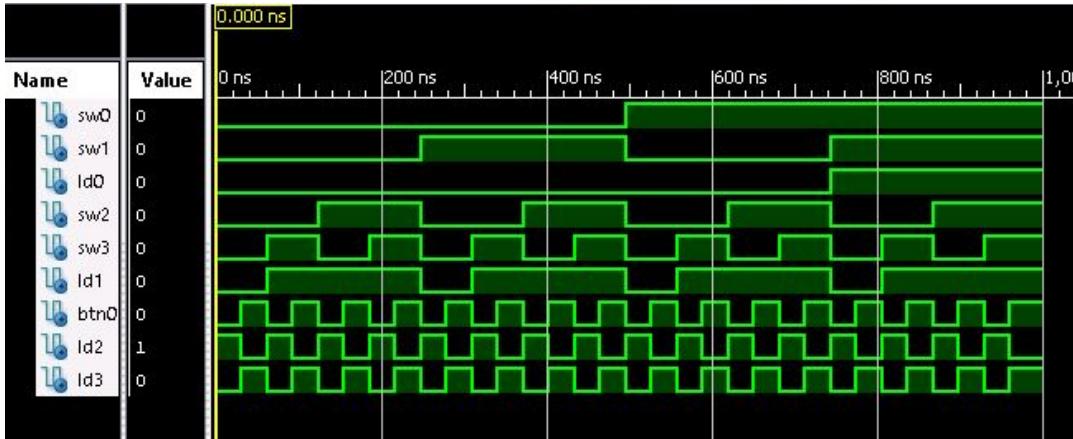
## 2) Arduino

PARTICIPANTES	SESIÓN 1	SESIÓN 2	SESIÓN 3	HORAS DE CADA SESIÓN	PORCENTAJE DE PARTICIPACIÓN EN CA
Elena del Río Galera	Programador	Diseñador	Programador	2 horas	0,25 % cada una
Andrea López Recio	Programador	Programador	Evaluador	2 horas	0,25 % cada una
Alicia Gordo Azabal	Evaluador	Programador	Diseñador	2 horas	0,25 % cada una
Yolanda Lillo Mata	Diseñador	Evaluador	Programador		
BLOQUES REALIZADOS	Iniciación a la programación a Arduino	Semáforo digital	Piano digital		
LÍNEAS EMPLEADAS		134 líneas	90 líneas		
COSTE TOTAL		33,5 euros	22,5 euros		

# GRÁFICAS DE EXCEL



# BLOQUES XILINX



Bloque 3: Puertas lógicas, interruptores y pulsadores

# BLOQUES XILINX

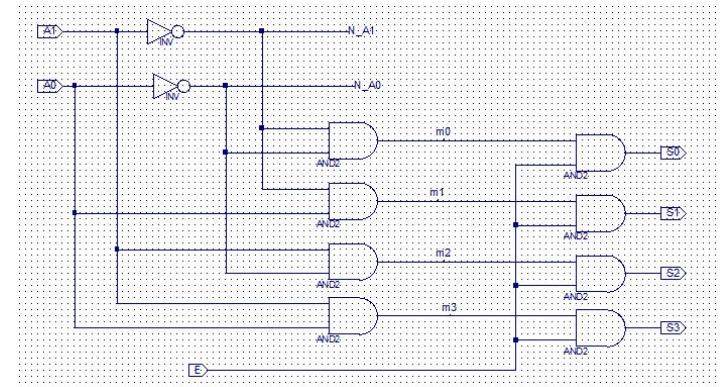
En este apartado vamos a introducir puertas lógicas, utilizando los botones e interruptores (switches) y leds de nuestra placa como entradas y salidas respectivamente. Para ello, utilizaremos puertas **AND** y **OR**, además de una puerta inversora **NOT** y un **buffer**, que electrónicamente nos retrasa un poco la señal, pero que a efectos prácticos nos sirve para poder conectar un puerto de entrada que ya tiene una salida a otra salida distinta.

- Puerta AND( $S=A \cdot B$ ): su salida LD0 únicamente será un 1 en el caso de que sus dos entradas (SW0 y SW1) tengan el valor 1. Esto ocurre cuando se encienden los switches 0 y 1 a la vez.
- Puerta OR( $S=A+B$ ): la salida LD1 sólo recibirá un 0 en el caso en que ambos switches permanezcan en off, y por tanto se activará tanto si se enciende el switch 2, como el switch 3 o los dos a la vez.

Siempre tendremos una salida activada. El Botón 0 siempre estará enviando un 0 hasta que se pulse en la FPGA. Si no se pulsa, el Led2 permanece encendido y el Led3 apagado. Cuando se pulse el botón sucederá lo contrario.

*Bloque 3: Puertas lógicas, interruptores y pulsadores*

# BLOQUES XILINX



Bloque 4: Decodificador de 2 a 4

# BLOQUES XILINX

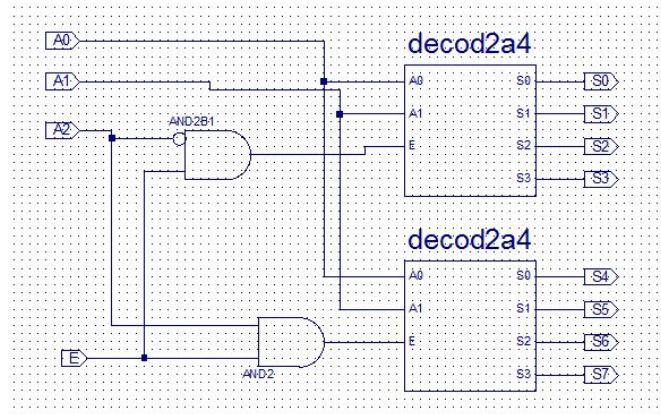
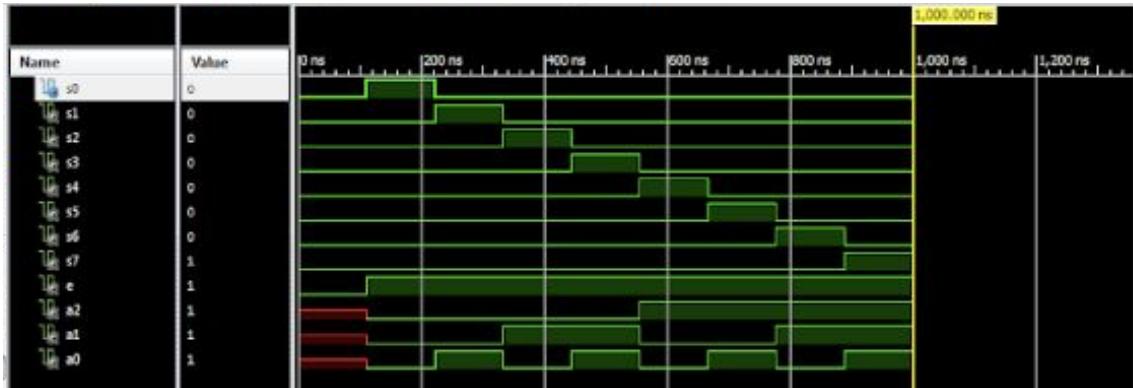
Un decodificador de 2 a 4 recibe **dos entradas: A0 y A1**, que están codificadas en binario (0 ó 1). Estas entradas tienen dos cifras binarias (formando un número del 0 al 3 ). Y tendremos **cuatro salidas (S0, S1, S2 y S3)** que se activarán dependiendo del número que se introduzca en las entradas (si se introduce 10 se activa la salida 2 del decodificador).

Además, disponemos de una entrada **ENABLE (E)**, las salidas son minitérminos de las entradas siempre y cuando la señal E esté activada. Es por eso que implementamos el circuito con:

- Puertas AND (S=A·B): Para lograr que cuando Enable es 0 ninguna de las salidas se active.  
(EJ:  $S_0 = A_1 \cdot A_0 ; S_1 = A_1 \cdot A_0$  )

*Bloque 4: Decodificador de 2 a 4*

# BLOQUES XILINX



Bloque 5: Decodificador de 3 a 8

# BLOQUES XILINX

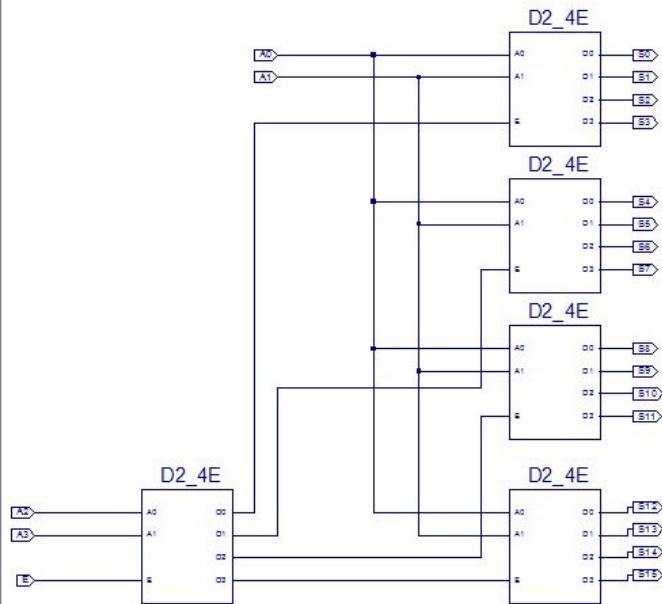
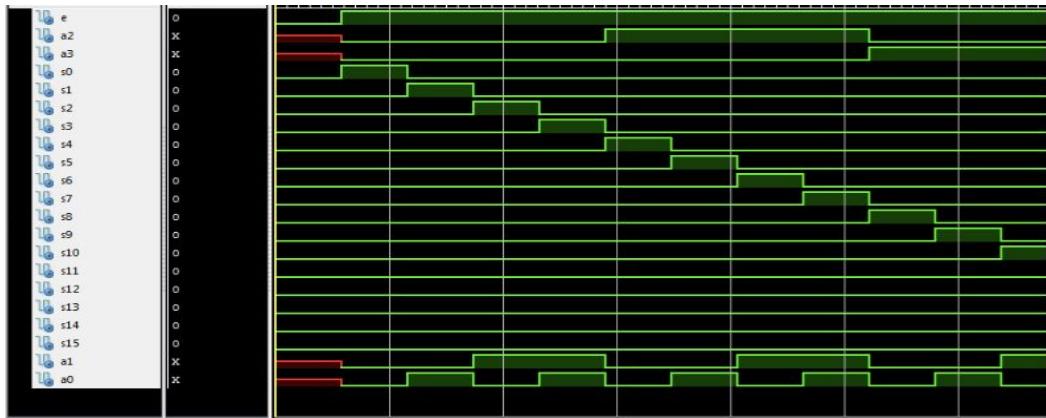
En este apartado vamos a construir un **decodificador de 3 a 8** y vamos añadir a nuestro proyecto el bloque creado en el apartado anterior, dado que realizaremos el diseño de este decodificador de 3 a 8 a partir del decodificador de 2 a 4 de manera que nos quedará un diseño mucho más simple que si eligiésemos realizarlo a través de puertas lógicas.

- En el decodificador de arriba metemos como señal de activación  $E \cdot A_2$ .
- En el decodificador de abajo metemos como señal de habilitación  $E \cdot A_2$ .

Introducimos las señales  $A_0$  y  $A_1$  como entradas de ambos decodificadores y el primero nos dará las salidas de  $S_0$  a  $S_3$  y el segundo de  $S_4$  a  $S_7$ .

*Bloque 5: Decodificador de 3 a 8*

# BLOQUES XILINX



Bloque 6: Decodificador de 4 a 16

# BLOQUES XILINX

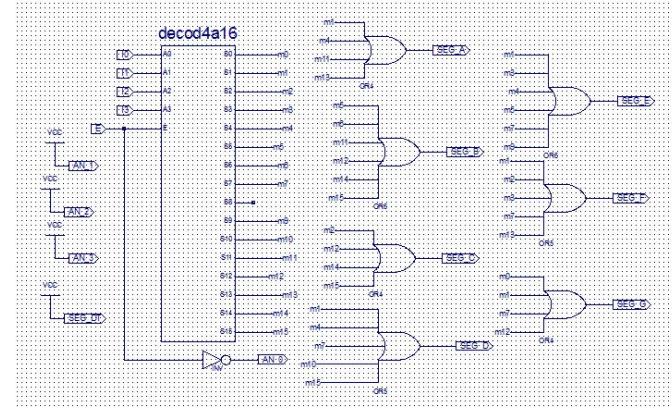
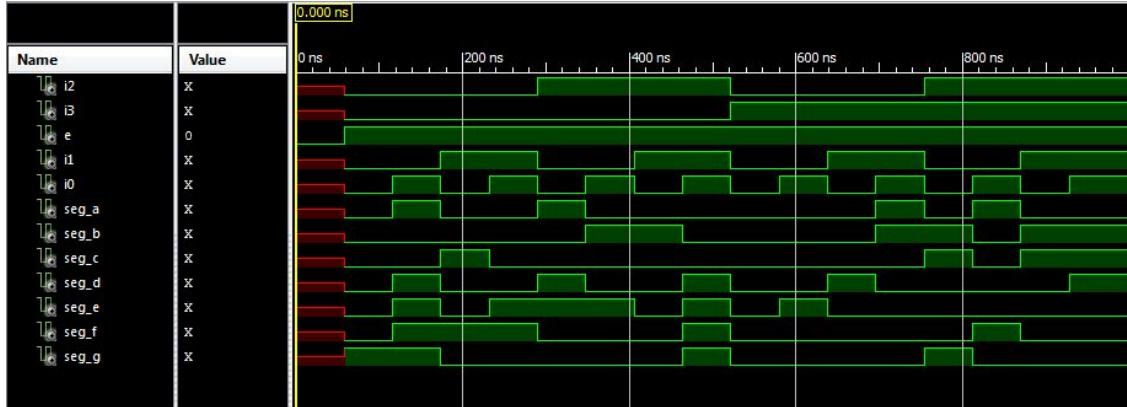
En este bloque tenemos que construir un **decodificador de 4 a 16**, implementándolo a través de los decodificadores 2 a 4 del apartado 4.

Todos los decodificadores tendrán las mismas entradas A1 y A0. Mientras que las salidas serán diferentes dependiendo de cada decodificador:

- Primer decodificador: tendrá como salidas S0, S1, S2 y S3 y sólo estará habilitado cuando A3='0' y A2='0'.
- Segundo decodificador: estará habilitado cuando A3='0' y A2='1'.
- Y el tercer y cuarto decodificador de manera similar.

*Bloque 6: Decodificador de 4 a 16*

# BLOQUES XILINX



Bloque 7: Convertidor de hexadecimal a 7 segmentos

# BLOQUES XILINX

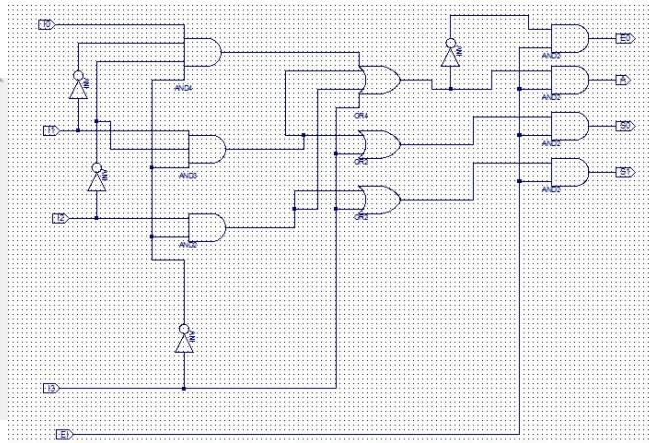
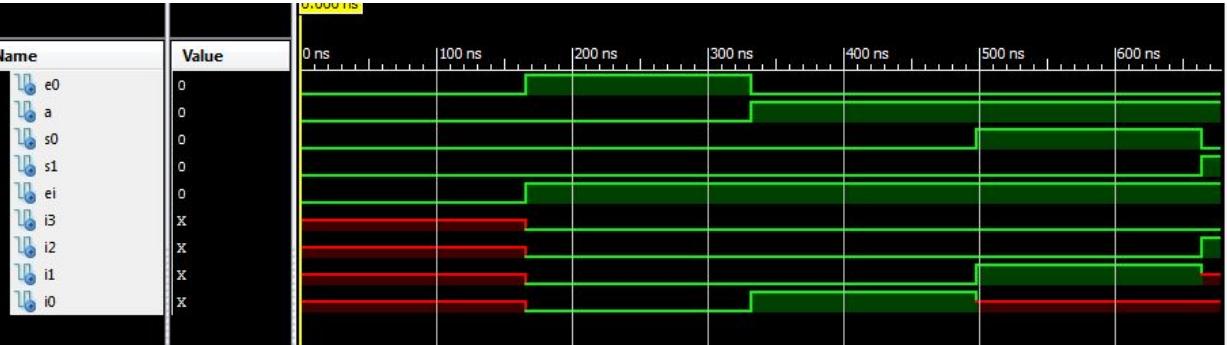
El objetivo es mostrar un número hexadecimal (4 bits) en un display de siete segmentos. Dado que es un número codificado en hexadecimal utilizando 4 bits (desde 0 hasta 15), podemos tener cifras desde el 0 hasta la F.

Disponemos de:

- La señal E. Con  $E='1'$ , se dará un 0 a aquellos segmentos que conformen el número (o letra) que tenemos representado en las entradas I0 a I3 en binario.
- Puertas OR. En las salidas del decodificador, ya que en el caso de que se tenga que encender un segmento dado, se activarán (tanto AN\_X como los segmentos), si reciben un '0' lógico '.

*Bloque 7: Convertidor de hexadecimal a 7 segmentos*

# BLOQUES XILINX



Bloque 8: Codificadores

# BLOQUES XILINX

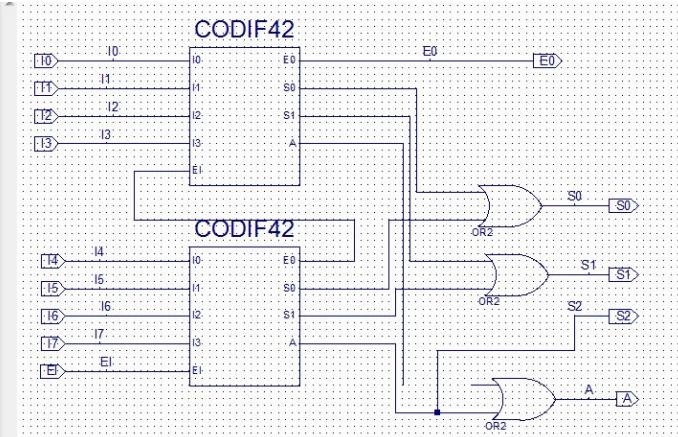
En este apartado vamos a construir un **codificador de 4 a 2 con prioridad**. Este tipo de codificadores permiten que varias entradas de las 4 disponibles estén activadas simultáneamente, y que se siga cumpliendo un comportamiento lógico. Además, incluiremos una señal visual (A) que nos aporte información acerca de si se está utilizando alguna de las entradas, es decir, si alguno de los botones ha sido pulsado.

- Para determinar las prioridades de nuestro codificador, lo que debemos hacer es incluir puertas AND, donde las entradas sean las señales con más prioridad, pondremos la prioridad más alta para el bit más significativo, de modo que si I2 e I1 estuviesen pulsados, el codificador mostrase un 10 y no un 11.
- Para la entrada I0 (BTN0), pondremos una puerta AND cuyas entradas sean I0, I1, I2 e I3, de manera que si valen '1' multiplicamos por 0 en la puerta lógica.
- Para la salida utilizaremos los leds 0 y 1 como salidas S0 y S1 (se encienden dependiendo de los botones que pulsemos).

Los codificadores con habilitación incluyen una nueva señal de salida (EO: enable out), que nos permitirá conectar varios codificadores y extender su capacidad. Esta señal nos indica si el codificador está habilitado pero no hay ninguna señal de entrada activa, nos diferencia de las ocasiones en que nuestro codificador no se encuentra habilitado.

## Bloque 8: Codificadores

# BLOQUES XILINX



Bloque 9: Extensión de la capacidad de un codificador

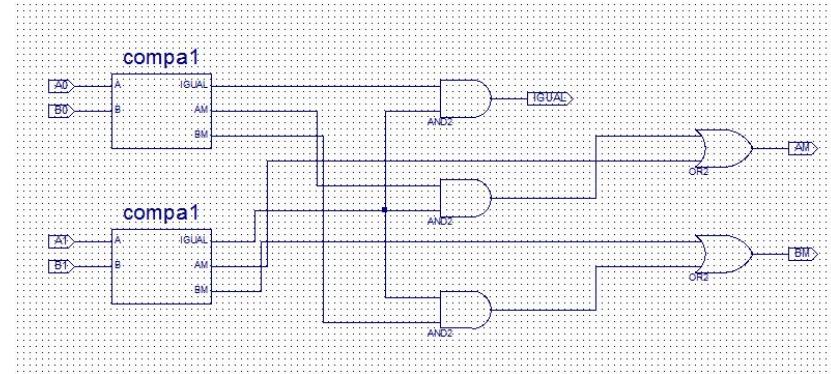
# BLOQUES XILINX

En este apartado vamos a realizar un codificador de 8 a 3 a partir de dos codificadores 4 a 2. El codificador 4 a 2 situado en la parte superior es el que se encargará de los bits menos significativos, y el otro codificador de debajo de los bits más significativos.

- Puertas OR. Para formar las salidas S0 y S1 de los codificadores.
- La señal A del codificador 4 a 2. Para obtener la tercera salida, la señal A se activará representando los bits más significativos.
- La señal A del codificador de 8 a 3 se obtiene realizando la suma lógica (OR) de las señales de activo de los codificadores de 4 a 2, ya que estará activo si cualquiera de ellos está activo.

*Bloque 9: Extensión de la capacidad de un codificador*

# BLOQUES XILINX



Bloque 10: Comparadores

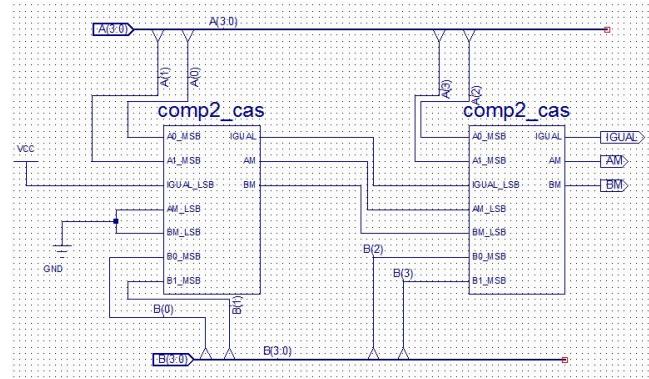
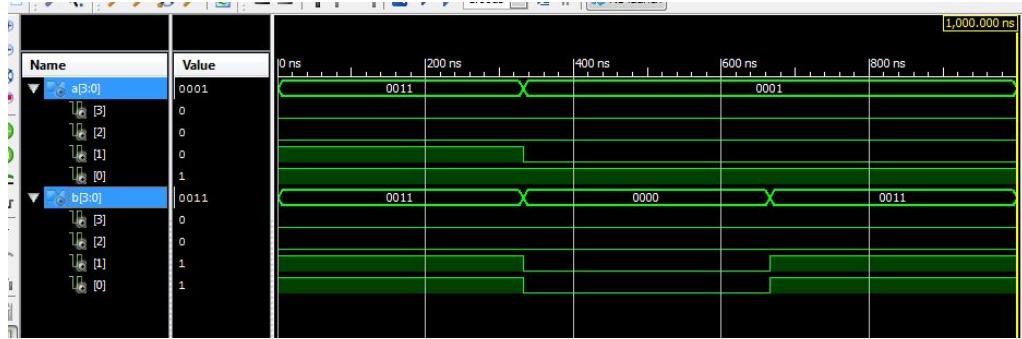
# BLOQUES XILINX

En este bloque vamos a diseñar un comparador de 2 bits. Un circuito comparador consiste en comparar las magnitudes de dos cantidades binarias para determinar la relación entre ellas ( $>$ ,  $<$  ó  $=$ ). Tenemos que tener en cuenta el bit más significativo de cada señal.

- A es igual a B cuando:  $A_1=B_1$  y  $A_0=B_0$
- A es mayor que B cuando  $A_1>B_1$  o cuando  $(A_1=B_1)$  y  $A_0>B_0$
- A es menor que B cuando  $A_1<B_1$  o cuando  $(A_1=B_1)$  y  $A_0<B_0$

*Bloque 10: Comparadores*

# BLOQUES XILINX



Bloque 11: Comparadores en cascada

# BLOQUES XILINX

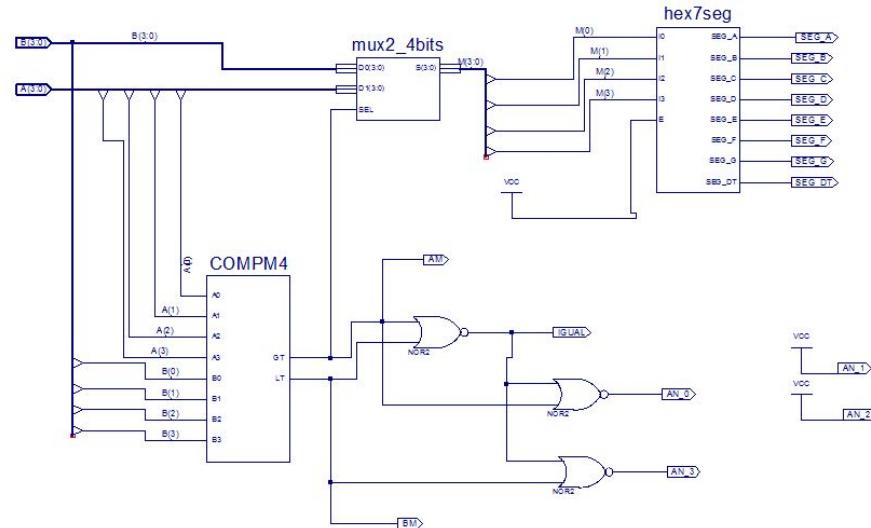
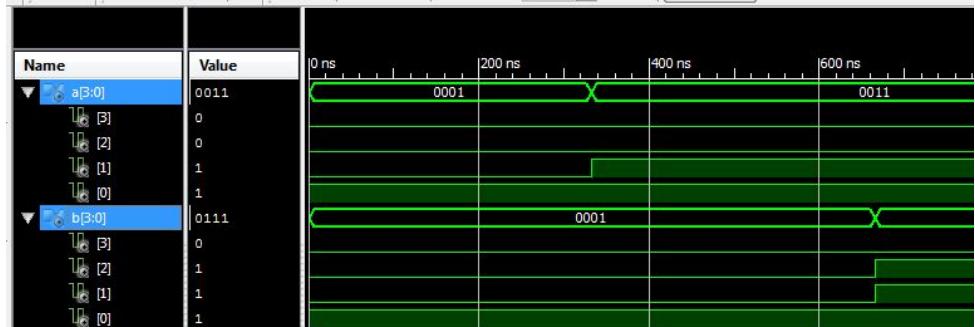
El objetivo es realizar un comparador de mayor capacidad (más bits). Tenemos 3 nuevas entradas (IGUAL\_LSB, AM\_LSB Y BM\_LSB), las cuales se corresponden con la comparación resultante de los bits menos significativos. Como resultado de la placa tendremos:

En la parte izquierda tenemos el comparador de los bits menos significativos. Para lograr que un comparador se pueda conectar en cascada tenemos que hacer que las salidas del primer comparador sean ahora las entradas del segundo comparador de bits más significativos.

Las entradas AM\_LSB y BM\_LSB del primer comparador valen '0', mientras que IGUAL\_LSB vale '1', dado que al no haber más bits antes los vamos a considerar iguales para que no influyan en la comparación posterior.

*Bloque 11: Comparadores en cascada*

# BLOQUES XILINX



Bloque 12: El multiplexor

# BLOQUES XILINX

Este bloque permite dirigir la información binaria procedente de diferentes fuentes a una única línea de salida para ser transmitida a un destino común.

Al comparar dos números binarios A y B, no sólo se encienden los leds indicando cual de los dos es mayor, sino que también aparecen dichos números en los displays, distribuidos de la siguiente forma:

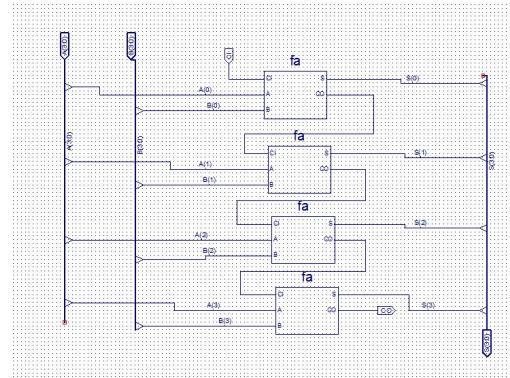
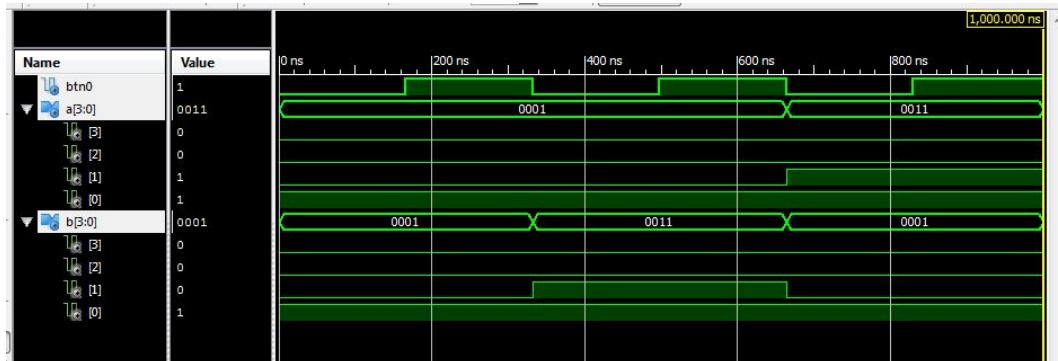
- $A > B \rightarrow$  lo mostrará por el display de la derecha (AN\_0).
- $A < B \rightarrow$  lo mostrará por el display de la izquierda (AN\_3).
- $A = B \rightarrow$  se mostrará por los dos displays anteriores.

Cada multiplexor recibe uno de los bits de las señales de datos (D0 y D1), y dependiendo de la señal de selección que se introduzca (SEL), tendremos una salida u otra (A ó B).

Por último añadimos un convertidor a 7 segmentos y un comparador al circuito.

*Bloque 12: El multiplexor*

# BLOQUES XILINX



Bloque 13: El sumador

# BLOQUES XILINX

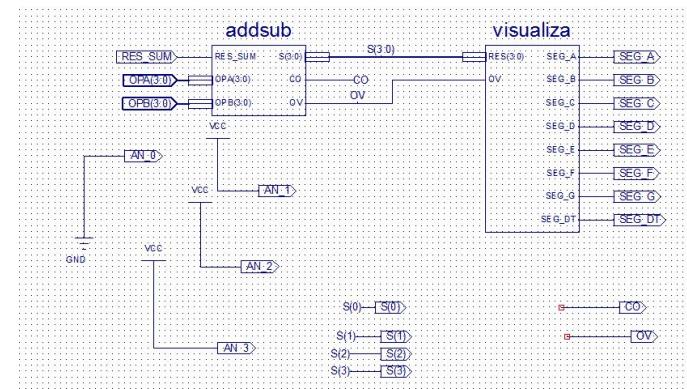
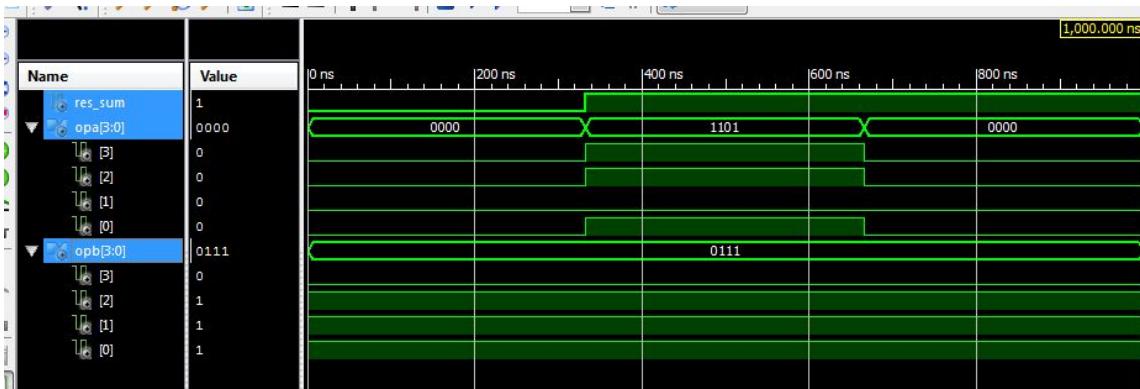
En este bloque realizamos un sumador de dos números de 4 bits codificados por medio de los interruptores de la placa. Tenemos que conectar los acarreos consecutivamente del bit menos significativo al más significativo, y conectar las entradas de A y B en paralelo, en sus sumadores correspondientes.

Los 4 bits de las entradas A y B se corresponden con los switches del 0 al 7. En la placa se mostrará el acarreo de salida en el led 7, e introducirá el correspondiente acarreo de entrada pulsando el primer botón de la FPGA.

La salida la obtendremos tanto a través de los leds como en el display de 7 segmentos.

*Bloque 13: El sumador*

# BLOQUES XILINX



Bloque 14: Sumador/restador

# BLOQUES XILINX

Este bloque calcula el complemento a dos del sustraendo y lo suma al minuendo.

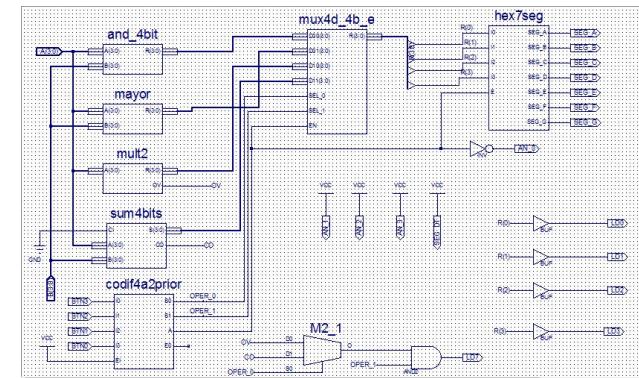
Es un Sumador/restador de dos números de 4 bits. El circuito debe sumar o restar dos números codificados en complemento a 2 con 4 bits (OPA y OPB), y cuyos valores estarán determinados por la posición de los interruptores de la placa.

El circuito funciona de la siguiente manera:

- El resultado de la operación (suma o resta) se mostrará por el primer display de 7 segmentos. (en signo magnitud)
- El minuendo (o uno de los sumandos) se codificará mediante los 4 interruptores de la derecha (de SW0 a SW3)
- El sustraendo (o el otro sumando) con los cuatro interruptores de la izquierda (de SW4 a SW7).
- El pulsador BTN0 indicará qué operación se realiza. (Pulsado=Resta, Sin pulsar=Suma)
- Para mostrar el signo negativo se utilizará el punto decimal (Encendido = Negativo)
- En caso de que haya desbordamiento (overflow) el display mostrará la letra E.
- Además los cuatro LED de la derecha (de LD0 a LD3) mostrarán el resultado directo (en complemento a 2).
- Se usará LD7 para el acarreo de salida y LD6 para el desbordamiento en la resta.

*Bloque 14: Sumador/restador*

# BLOQUES XILINX



Bloque 15: Unidad aritmético lógica

# BLOQUES XILINX

En este bloque conseguiremos realizar 4 operaciones distintas con dos números de 4 bits (suma, multiplicación por 2, mayor y la operación lógica AND). Un multiplexor escogerá las salidas de esos bloques según la operación seleccionada. Por lo que realizamos:

Un Multiplicador x2, que desplazará los números hacia la izquierda y en el bit menos significativo se pone un cero. Si hubiese un uno en el bit más significativo habría que indicar que ha habido desbordamiento.

Circuito comparador, "mayor": Si  $A > B$  (como entrada de selección de un multiplexor) valdrá '1' y escogerá A, y en caso contrario B.

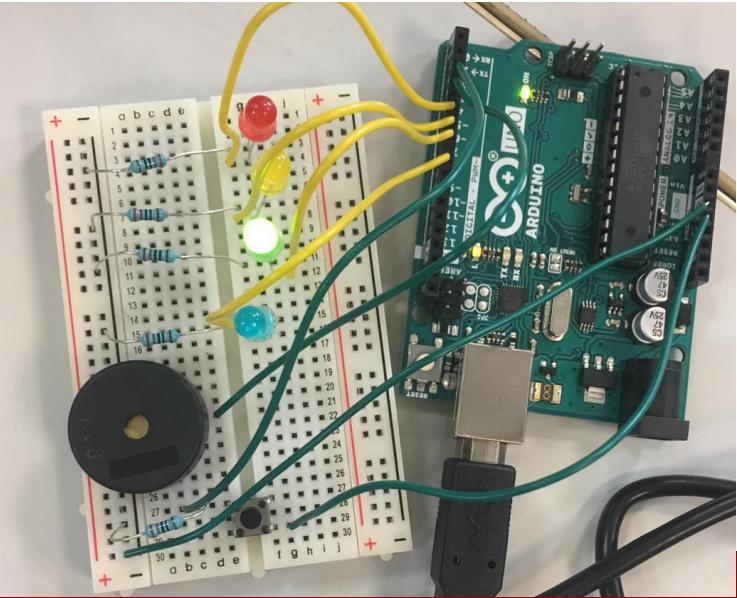
Operación lógica AND: La realizamos bit a bit multiplicando la operación a por la operación B.

Multiplexor de 4 entradas. Tendrá dos entradas de selección, cada dato tiene cuatro bits.

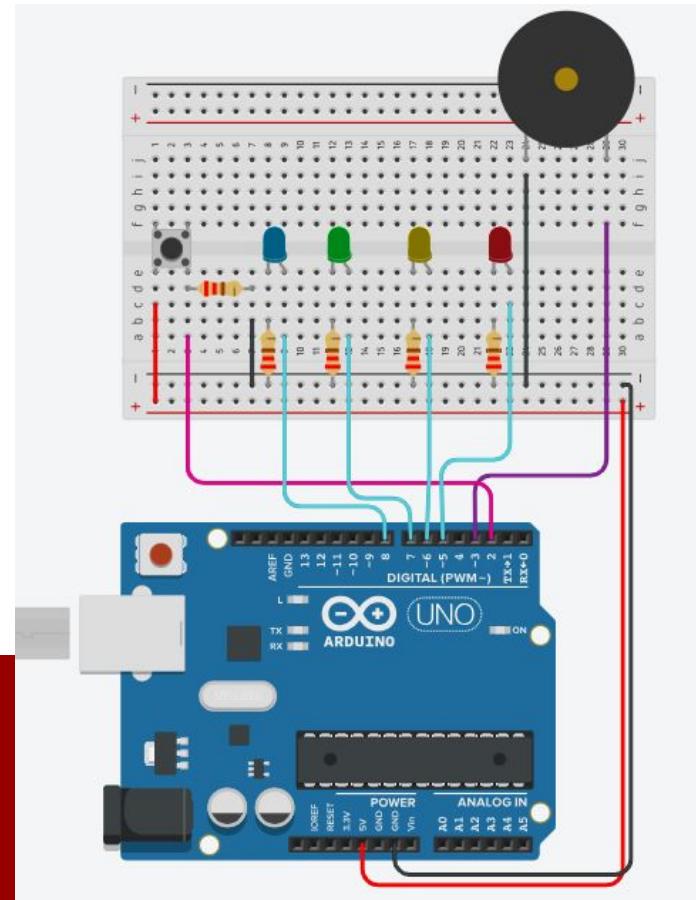
Codificador de 4 a 2 con prioridad: Para incluir el caso de que se pulsen dos botones a la vez. En este caso se pulsará el botón situado más a la derecha de nuestra placa..

*Bloque 15: Unidad aritmético lógica*

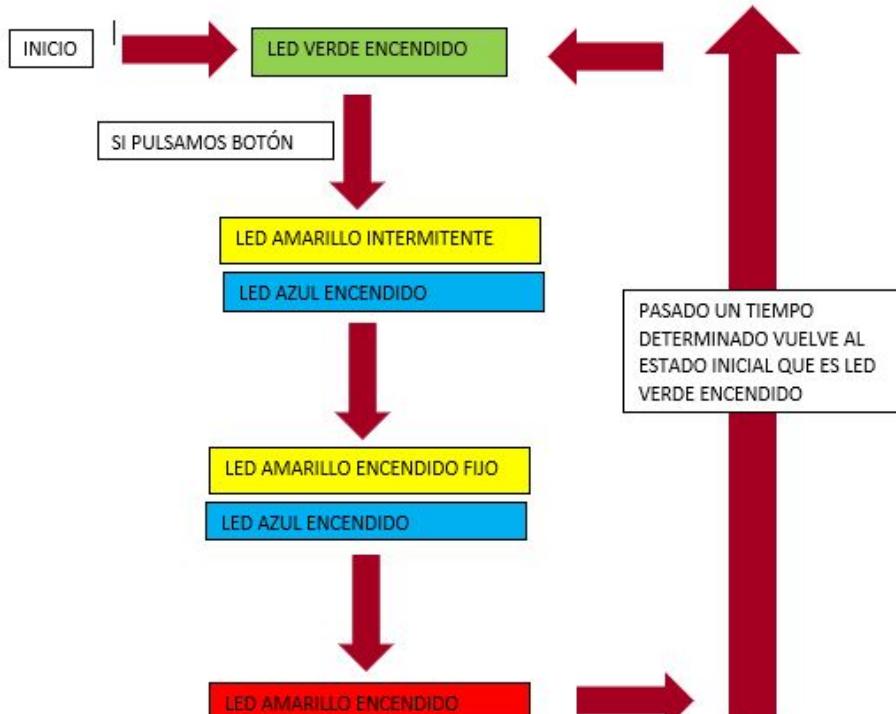
# ARDUINO



Semáforo

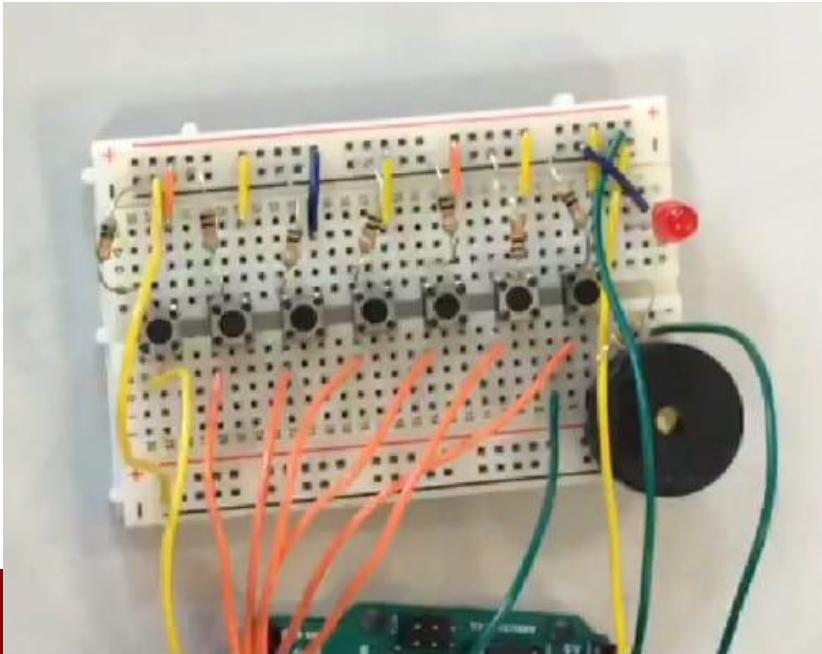


# ARDUINO

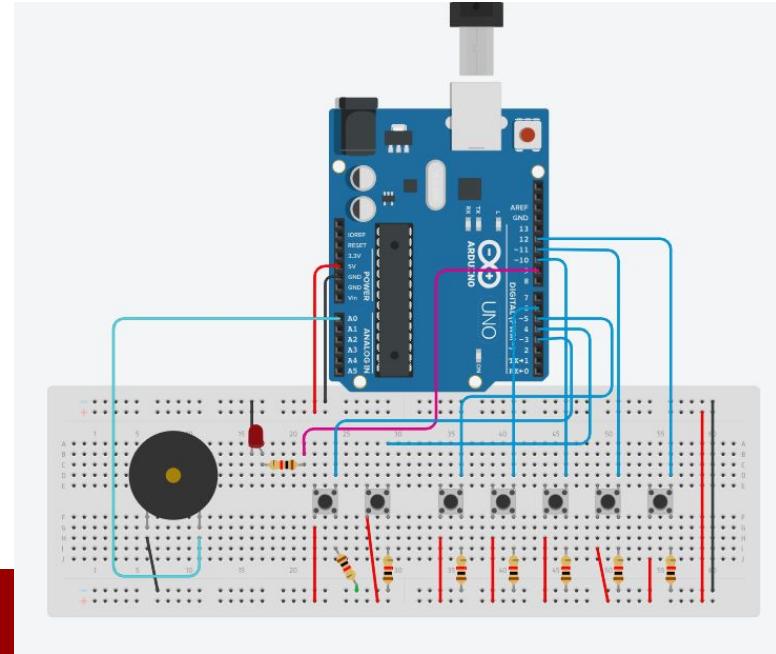


*Diagrama de flujo del semáforo*

# ARDUINO



Piano



# ARDUINO

Diagrama de flujo  
del piano.

