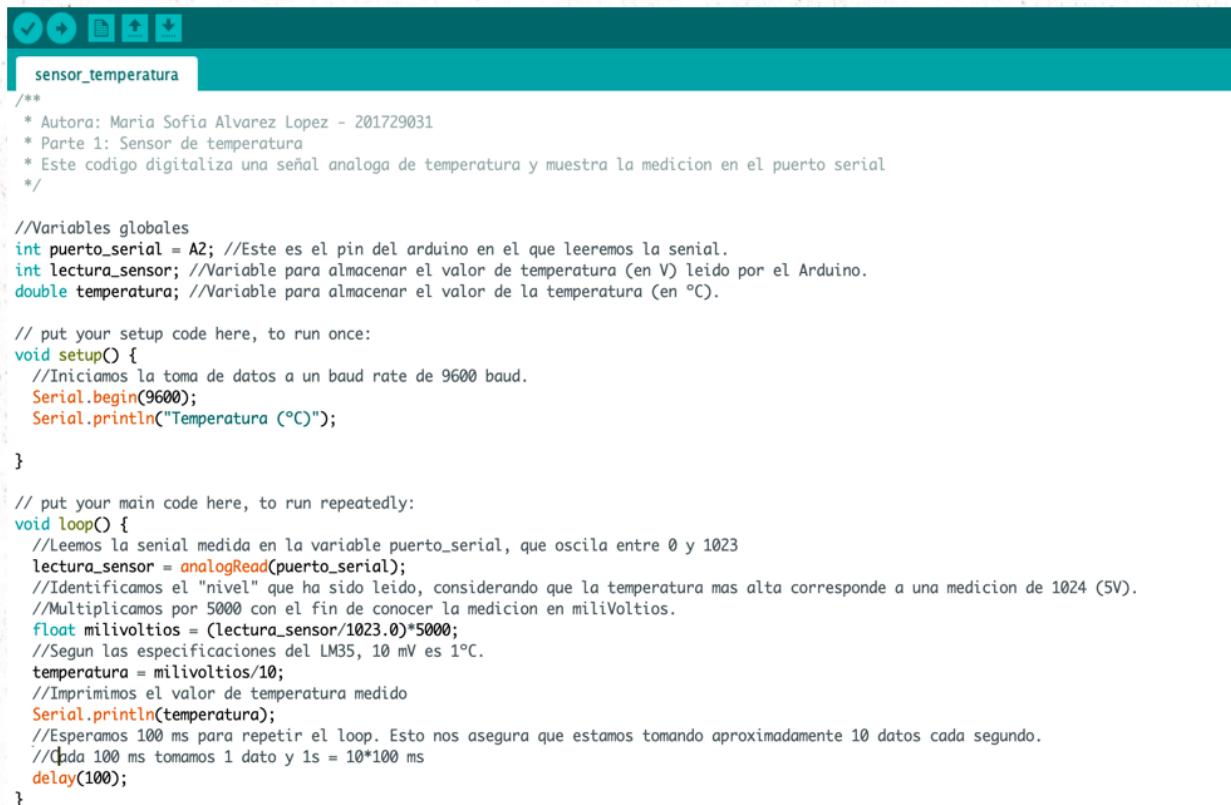


Preinforme 10: Sensores y circuitos Lógicos

Parte 1: Sensor de temperatura

1. Desarrolle un código que digitalice una señal análoga: A2 (sensor LM35), y muestre el resultado en puerto serial, haciendo la conversión adecuada para reportar el valor en grados centígrados.

NOTA: según la hoja de especificaciones del LM35, $10mV = 1^\circ C$ tras usar las conexiones de la figura mostrada en la guía



```

sensor_temperatura

/**
 * Autora: Maria Sofia Alvarez Lopez - 201729031
 * Parte 1: Sensor de temperatura
 * Este código digitaliza una señal analoga de temperatura y muestra la medición en el puerto serial
 */

// Variables globales
int puerto_serial = A2; //Este es el pin del arduino en el que leeremos la señal.
int lectura_sensor; //Variable para almacenar el valor de temperatura (en V) leido por el Arduino.
double temperatura; //Variable para almacenar el valor de la temperatura (en °C).

// put your setup code here, to run once:
void setup() {
    //Iniciamos la toma de datos a un baud rate de 9600 baud.
    Serial.begin(9600);
    Serial.println("Temperatura (°C)");
}

// put your main code here, to run repeatedly:
void loop() {
    //Leemos la señal medida en la variable puerto_serial, que oscila entre 0 y 1023
    lectura_sensor = analogRead(puerto_serial);
    //Identificamos el "nivel" que ha sido leído, considerando que la temperatura más alta corresponde a una medición de 1024 (5V).
    //Multiplicamos por 5000 con el fin de conocer la medición en milivoltios.
    float milivoltios = (lectura_sensor/1023.0)*5000;
    //Según las especificaciones del LM35, 10 mV es 1°C.
    temperatura = milivoltios/10;
    //Imprimimos el valor de temperatura medido
    Serial.println(temperatura);
    //Esperamos 100 ms para repetir el loop. Esto nos asegura que estamos tomando aproximadamente 10 datos cada segundo.
    //Cada 100 ms tomamos 1 dato y 1s = 10*100 ms
    delay(100);
}

```

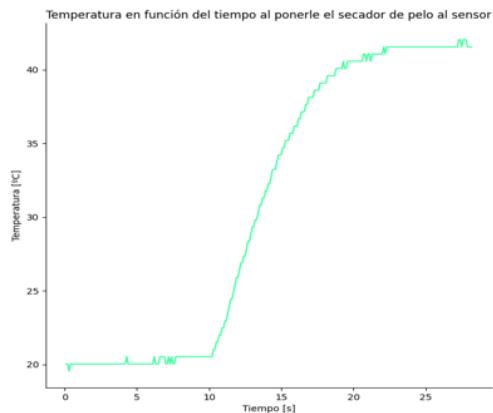
Compilado

El Sketch usa 3428 bytes (10%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 218 bytes (1%) de la memoria dinámica, dejando 1830 bytes para las variables locales. El máximo es 2048 bytes.

2. Mida y registre de forma continua, por ejemplo 10 veces por segundo, la temperatura con el sensor. Inicie su toma de datos a una temperatura estable T_0 , aplique un cambio de temperatura al sensor y finalice la toma de datos una vez se haya estabilizado de nuevo el sensor a la temperatura final, T_F . Tome suficientes datos para poder estimar el tiempo de estabilización.

Note del punto anterior que, entre cada ejecución del método `loop()`, invocamos al método `delay(100)`; lo cual corresponde a una demora de 100 ms en cada ejecución (podemos despreciar el tiempo de cálculo asociado a las instrucciones previas - al ejecutarse en tiempo $O(1)$). Esto quiere decir que, aproximadamente cada 10 datos tomados, ha transcurrido 1 segundo.

Lo anterior fue muy importante para poder graficar la temperatura del sensor (en °C), en función del tiempo, en segundos. Como podemos ver en la gráfica a continuación, después de medir la temperatura T_0 por 10 segundos, aproximadamente, encendí el secador de pelo frente al sensor, tal que la temperatura pasó de $T_0 \approx 20^\circ\text{C}$ hasta $T_F \approx 41^\circ\text{C}$ para la toma de datos una vez vi que el sensor se estabilizó en esta última temperatura.



Podemos tomar el tiempo de estabilización del sensor des de que prendí el secador hasta el último dato cuando vi que finalmente se había estabilizado. Así,

$$\tau = t_F(T_F) - t_i(T_i)$$

$$\tau = 28.2 \text{ s} - 10 \text{ s}$$

$$\tau = 18.2 \text{ s} \quad (1)$$

Es el tiempo de estabilización del sensor.

Esta misma respuesta (y la gráfica) la podemos ver al correr el archivo "Preinforme_10_Electrónica-para-Ciencias / Parte 1: sensor de temperatura/datos/analisis.py" disponible en Github (y más adelante). La respuesta, a continuación,

```
(base) sofiaalvarezlopez@MBP-de-Sofia ~ % python analisis.py
La grafica ha sido guardada exitosamente en la ruta: ./estabilizacion_sensor.png
El tiempo de estabilizacion del sensor es: 18.20 s
```

Código usado para el análisis de datos:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 temperaturas = np.loadtxt('./datos_tomados.txt')
5
6 #Note que la posicion i-esima corresponde a los ms transcurridos.
7 #Por ejemplo, para el decimo dato (posicion 9) significa que ha pasado 1 segundo desde que inicio la toma de datos.
8
9 tiempo = np.linspace(1/10, len(temperaturas)/10, len(temperaturas))
10
11 plt.figure(figsize=(8,8))
12 ax = plt.gca()
13 ax.spines['right'].set_visible(False)
14 ax.spines['top'].set_visible(False)
15 #plt.plot(tiempo, temperaturas, '.')
16 plt.plot(tiempo, temperaturas, color="#3bff9d")
17 plt.title('')
18 plt.xlabel(r'Tiempo [s]')
19 plt.ylabel(r'Temperatura [°C]')
20 plt.title(r'Temperatura en función del tiempo al ponerle el secador de pelo al sensor')
21 plt.savefig('./estabilizacion_sensor.png')
22
23 print('La grafica ha sido guardada exitosamente en la ruta: ./estabilizacion_sensor.png')
24 print()
25 """Podemos considerar el tiempo en que se prendio el secador como el tiempo inicial, t_i,
26 a la temperatura T_0. Este tiempo es aproximadamente a los 10 segundos.
27 Asimismo, podemos tomar el ultimo dato como el tiempo en que se estabilizo la señal, t_f, a la
28 temperatura final, t_f.
29 """
30 #Por una regla de 3 sencilla, podemos darnos cuenta que si el dato en la posicion 9 corresponde a
31 #1 segundo, el dato en la posicion 99 sera cuando hayan transcurrido 10 segundos.
32 t_i = tiempo[99]
33 t_f = tiempo[-1]
34 """
35 Definimos el tiempo de estabilizacion como el tiempo que tarda el sensor en estabilizarse,
36 desde el momento en que se empezo a cambiar la temperatura.
37 """
38 print('El tiempo de estabilizacion del sensor es: {:.2f} s'.format(t_f - t_i))
```

Parte 2: Detector de Oscuridad:

ESAI

3. Desarrolle un código que digitalice dos señales análogas: A0 (Divisor de voltaje con potenciómetro) y A1 (Divisor de voltaje R con fotocelda), y muestre sus valores de forma simultánea en puerto serial. Programme además que se active la señal AD (potenciómetro) digital D10 solo cuando la señal A0 (potenciómetro) sea mayor a A1 (fotocelda).

detector_oscuridad Arduino 1.8.13

```
Parte 2: Detector de Oscuridad
* Autora: Maria Sofia Alvarez Lopez - 201729031
* Este código digitaliza dos señales analógicas:
* A0: Divisor de Voltaje con Potenciómetro. A1: Divisor de Voltaje R con Fotocelda.
* Además, activa la señal digital D10 cuando la señal A0 (potenciómetro) sea mayor a la de A1 (fotocelda).
*/
int puerto_potenciometro = A0; //El pin en que leeremos la señal analoga del potenciómetro.
int puerto_fotocelda = A1; //El pin en que leeremos la señal analoga de la fotocelda.
int salida_digital_LED = 10; //El pin de salida que se activara si la señal del potenciómetro (A0) es mayor a la de la fotocelda (A1).
int lectura_potenciometro; //Variable para almacenar la medición del potenciómetro en el canal A0.
int lectura_fotocelda; //Variable para almacenar la medición de la fotocelda en el canal A1.

// put your setup code here, to run once:
void setup() {
    //Iniciamos la toma de datos a un baud rate de 9600 baud.
    Serial.begin(9600);
    //Configuramos cada uno de los pines: ¿cuales señales son de entrada y cuales de salida?
    pinMode(puerto_potenciometro, INPUT); //Serial de entrada del potenciómetro.
    pinMode(puerto_fotocelda, INPUT); //Serial de entrada de la fotocelda.
    pinMode(salida_digital_LED, OUTPUT); //Serial de salida para encender/apagar el LED.
    //Imprimimos el encabezado
    Serial.print("V(A0) - Pot");
    Serial.print(F("\t")); //Separamos el texto con \t -una tabulación- para 'serial plotter'
    Serial.print("V(A1) - Foto"); //el '3' indica 3 cifras decimales
    Serial.print(F("\n")); //la ultima lectura debe terminar con un final de linea \n para 'serial plotter'
}

// put your main code here, to run repeatedly:
void loop() {
    //Leemos la señal medida en cada uno de los puertos seriales.
    //Primero, para el potenciómetro.
    lectura_potenciometro = analogRead(puerto_potenciometro);
    //Ahora, para la fotocelda
    lectura_fotocelda = analogRead(puerto_fotocelda);
    //De manera similar al caso anterior, debemos convertir las señales recibidas a voltios.
    //Debemos identificar el "nivel" que ha sido leido, considerando que el voltaje mas elevado corresponde a 1024.
    //Multiplicamos por 5 con el fin de conocer la medición en Voltios y dividimos entre 1023.0, haciendo una regla de 3.
    //Primero para el potenciómetro.
    float voltaje_potenciometro = lectura_potenciometro*5.0/1023.0;
    //Ahora, para la fotocelda.
    float voltaje_fotocelda = lectura_fotocelda*5.0/1023.0;
    //Imprimimos los valores obtenidos.
    Serial.print(voltaje_potenciometro,3); //el '3' indica 3 cifras decimales
    Serial.print(F("\t")); //Separamos los valores con \t -una tabulación- para 'serial plotter'
    Serial.print(voltaje_fotocelda,3); //el '3' indica 3 cifras decimales
    Serial.print(F("\n")); //la ultima lectura debe terminar con un final de linea \n para 'serial plotter'

    //Debemos encender el LED (i.e. activar la señal del pin D10 del Arduino) cuando la señal del potenciómetro sea mayor a la de la fotocelda.
    if (voltaje_potenciometro > voltaje_fotocelda){
        digitalWrite(salida_digital_LED, HIGH); //Como enviamos una señal HIGH (alta) por el pin digital, el LED se encenderá (o se mantendrá encendido).
    }
    else{
        digitalWrite(salida_digital_LED, LOW); //Como enviamos una señal LOW (baja) por el pin digital, el LED se apagará (o se mantendrá apagado).
    }
    //Esperamos 100 ms para repetir el loop. Esto nos asegura que estamos tomando aproximadamente 10 datos cada segundo, pues cada 100 ms tomamos 1 dato y 1s = 10*100 ms
    delay(100);
}

Compilado

El Sketch usa 4098 bytes (12%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 224 bytes (10%) de la memoria dinámica, dejando 1824 bytes para las variables locales. El máximo es 2048 bytes.
```

Parte 3: Compuestas Simples

9. Construya la tabla de verdad para cada una de las compuestas simples:

*Not:

La compuesta NOT negra la señal de entrada. Retorna lo contrario a la señal recibida. Si la señal se recibe en A y la salida es en Y,

A	Y
1	0
0	1

(Tabla 1)

Su ecuación booleana es: $Y = \bar{A}$. Esto quiere decir que si la entrada A es cierta, la salida Y es falsa (y viceversa)

*And:

La compuesta AND hace que, solamente si las entradas A y B son ciertas, la salida Y es cierta.

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(Tabla 2)

Su ecuación booleana está dada por: $Y = A \cdot B$

*Or:

La compuesta OR hace que solamente si A ó B son ciertas, la salida Y es cierta.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(Tabla 3)

Su ecuación booleana está dada por: $Y = A + B$

5. Desarrolle un código en Arduino que implemente las siguientes compuertas simples

ESAI

- $D4 = \text{NOT } D2$
- $D5 = D2 \text{ AND } D3$
- $D6 = D2 \text{ OR } D3$

Hay 2 formas de hacer esto: usando condicionales o usando expresiones booleanas

Opción 1:

The screenshot shows the Arduino IDE interface with the code for implementing simple logic gates using conditional statements. The code includes comments explaining the implementation of NOT, AND, and OR logic using pins D2, D3, D4, D5, and D6.

```
expresiones_condicionales
/*
 * Autora: Maria Sofia Alvarez Lopez - 201729031
 * Parte 3: Compuertas Simples (Version 1).
 * Este codigo implementa las compuertas simples NOT, AND, OR usando condicionales.
 */

//Variables globales
int D2 = 2; //Representa el pin de entrada D2.
int D3 = 3; //Representa el pin de entrada D3.
int D4 = 4; //Representa el pin de salida D4.
int D5 = 5; //Representa el pin de salida D5.
int D6 = 6; //Representa el pin de salida D6.
int A; //Variable para almacenar la lectura del pin digital D2.
int B; //Variable para almacenar la lectura del pin digital D3.

// put your setup code here, to run once:
void setup() {
    //Decimos cuales pines son señales de entrada y cuales de salida.
    pinMode(D2, INPUT);
    pinMode(D3, INPUT);
    pinMode(D4, OUTPUT);
    pinMode(D5, OUTPUT);
    pinMode(D6, OUTPUT);
}

// put your main code here, to run repeatedly:
void loop() {
    //Invocamos a cada una de las funciones con sus pines de entrada y salida.
    A = digitalRead(D2);
    B = digitalRead(D3);
    NOT(A,D4);
    AND(A,B,D5);
    OR(A,B,D6);
}

//Implementacion del NOT para una entrada A y una salida Y.
void NOT(int A, int Y){
    A == HIGH ? digitalWrite(Y,LOW) : digitalWrite(Y,HIGH);
}

//Implementacion del AND para dos entradas A,B y una salida Y.
void AND(int A, int B, int Y){
    A == HIGH && B == HIGH ? digitalWrite(Y, HIGH) : digitalWrite(Y, LOW);
}

//Implementacion del OR para dos entradas A,B y una salida Y.
void OR(int A, int B, int Y){
    A == HIGH || B == HIGH ? digitalWrite(Y, HIGH) : digitalWrite(Y, LOW);
}
```

Compilado

El Sketch usa 1032 bytes (3%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 13 bytes (0%) de la memoria dinámica, dejando 2035 bytes para las variables locales. El máximo es 2048 bytes.

Opción 2:

The screenshot shows the Arduino IDE interface. At the top, there are standard file navigation icons: back, forward, new, open, save, and upload/download. Below the toolbar, the title bar displays the sketch name: "expresiones_booleanas". The main code area contains the following C++ code for implementing simple logic gates using boolean expressions:

```
* Autora: Maria Sofia Alvarez Lopez - 201729031
* Parte 3: Compuertas Simples (Version 1).
* Este codigo implementa las compuertas simples NOT, AND, OR usando expresiones booleanas.
*/
//Variables globales
int D2 = 2; //Representa el pin de entrada D2.
int D3 = 3; //Representa el pin de entrada D3.
int D4 = 4; //Representa el pin de salida D4.
int D5 = 5; //Representa el pin de salida D5.
int D6 = 6; //Representa el pin de salida D6.
int A; //Variable para almacenar la lectura del pin digital D2.
int B; //Variable para almacenar la lectura del pin digital D3.
// put your setup code here, to run once:
void setup() {
    //Decimos cuales pines son seniales de entrada y cuales de salida.
    pinMode(D2, INPUT);
    pinMode(D3, INPUT);
    pinMode(D4, OUTPUT);
    pinMode(D5, OUTPUT);
    pinMode(D6, OUTPUT);
}

// put your main code here, to run repeatedly:
void loop() {
    //Leemos ambos valores en el pin digital.
    A = digitalRead(D2);
    B = digitalRead(D3);
    //Convertimos los valores leidos a booleanos.
    boolean boolean_A = convertir(A);
    boolean boolean_B = convertir(B);
    //NOT
    //D4 != D2; //D4 es la negacion de D2.
    NOT(boolean_A, D4);
    //AND
    //D5 = D2 && D3; //D5 = D2 AND D3.
    AND(boolean_A, boolean_B, D5);
    //OR
    //D6 = D2 || D3; // D6 = D2 OR D3.
    OR(boolean_A, boolean_B, D6);

}
//Implementacion del NOT para una entrada A y una salida Y.
void NOT(boolean A, int Y){
    digitalWrite(Y, !A);
}
//Implementacion del AND para dos entradas A,B y una salida Y.
void AND(boolean A, boolean B, int Y){
    digitalWrite(Y, A && B);
}
//Implementacion del OR para dos entradas A,B y una salida Y.
void OR(boolean A, boolean B, int Y){
    digitalWrite(Y, A || B);
}
//Convierte los valores de HIGH y LOW a TRUE o FALSE
boolean convertir(int A){
    boolean boolean_val = A == HIGH ? true : false;
    return boolean_val;
}
```

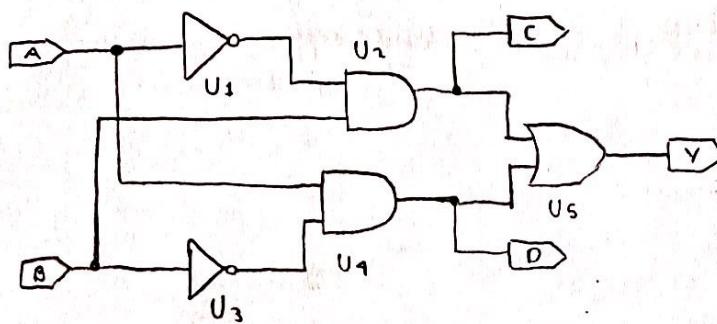
Below the code, a "Compilado" (Compiled) section displays the memory usage report:

El Sketch usa 1016 bytes (3%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 11 bytes (0%) de la memoria dinámica, dejando 2037 bytes para las variables locales. El máximo es 2048 bytes.

2

parte 4: compuertas compuestas

6. Construya las tablas de verdad de las salidas de los puntos C, D y Y del circuito lógico de la figura en función de las entradas A y B.



(Circuito 1)

Hagamos primero la tabla de verdad en el punto C; considerando que U_1 es un NOT y U_2 es un AND.

A : $U_1 = \bar{A}$	B	$C = \bar{A} \cdot B$
1	0	0
1	1	0
0	1	0
0	1	1

(Tabla 4)

Ahora, hagamos la tabla de verdad en el punto D, considerando que U_3 es un NOT y U_4 es un AND.

B : $U_3 = \bar{B}$	A	$D = A \cdot \bar{B}$
0	1	1
1	0	1
0	1	0
1	0	0

(Tabla 5)

Con esto, ya podemos construir la tabla de verdad para Y, considerando que U_5 es un OR.

D	C	Y
1	0	1
0	0	0
0	0	0
0	1	1

(Tabla 6).

Desarrolle el código para implementar el circuito.

Opción 1:

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Shows the sketch name: "compuertas_compuestas_opcion_2".
- Code Area:** Displays the C++ code for the sketch. The code implements a logic gate circuit using digital pins D2 through D9. It includes setup() and loop() functions, as well as helper functions for each logic gate (punto_C, punto_D, punto_Y) and a conversion function (convertir). The code uses boolean variables A, B, C, D, and Y to represent inputs and outputs.
- Compile Output:** A section titled "Compilado" displays memory usage statistics:
 - Sketch size: 1020 bytes (3%) of program storage used.
 - Global variables: 11 bytes (0%) of dynamic memory used, leaving 2037 bytes for local variables.
 - Max memory used: 32256 bytes.
- Page Number:** The bottom left corner shows the page number 33.

Opción 2:

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Shows the file name "compuertas_comuestas".
- Toolbar:** Standard Arduino IDE icons for file operations.
- Code Area:** The main code area contains the following C++ code for a logic gate implementation:

```
compuertas_comuestas

/**
 * Autora: Maria Sofia Alvarez Lopez - 201729031
 * Parte 4: Compuertas compuestas
 * Este codigo implementa el circuito de la figura 10.8 de la guia
 */

//Variables globales
int D2 = 2; //Representa el pin de entrada D2 (Entrada A).
int D3 = 3; //Representa el pin de entrada D3 (Entrada B).
int D7 = 7; //Representa el pin de salida D7 (Salida C).
int D8 = 8; //Representa el pin de salida D8 (Salida D).
int D9 = 9; //Representa el pin de salida D9 (Salida Y).
int A; //Variable para almacenar lo medido en A.
int B; //Variable para almacenar lo medido en B.

// put your setup code here, to run once:
void setup() {
    //Decimos cuales pines son seniales de entrada y cuales de salida.
    pinMode(D2, INPUT);
    pinMode(D3, INPUT);
    pinMode(D7, OUTPUT);
    pinMode(D8, OUTPUT);
    pinMode(D9, OUTPUT);
}

// put your main code here, to run repeatedly:
void loop() {
    A = digitalRead(D2);
    B = digitalRead(D3);
    int C = punto_C(A,B,D7);
    int D = punto_D(A,B,D8);
    punto_Y(C,D,D9);
}

//Implementacion de la compuerta en C
int punto_C(int A, int B, int C){
    //Lo primero que hacemos es negar A
    int not_A = A == HIGH ? LOW : HIGH;
    //Ahora hacemos not_A AND B
    not_A == HIGH && B == HIGH ? digitalWrite(C, HIGH) : digitalWrite(C, LOW);
    return not_A && B;
}

//Implementacion de la compuerta en D
int punto_D(int A, int B, int D){
    //Lo primero que hacemos es negar B
    int not_B = B == HIGH ? LOW : HIGH;
    //Ahora hacemos not_B AND A
    not_B == HIGH && A == HIGH ? digitalWrite(D, HIGH) : digitalWrite(D, LOW);
    return not_B && A;
}

//Implementacion de la compuerta en Y
void punto_Y(int C, int D, int Y){
    C == HIGH || D == HIGH ? digitalWrite(Y, HIGH) : digitalWrite(Y, LOW);
}


```

Compilado

El Sketch usa 1080 bytes (3%) del espacio de almacenamiento de programa. El máximo es 32256 bytes.
Las variables Globales usan 13 bytes (0%) de la memoria dinámica, dejando 2035 bytes para las variables locales. El máximo es 2048 bytes.

Finalmente, puede ser más sencillo ver la tabla compuesta en una sola.

Veamos,

A	U1: A'	B	U3: B'	C= A' · B	D= A · B'	Y=C+D
1	0	0	1	0	1	1
1	0	1	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	1	0	1

TODO el código está disponible en el siguiente repositorio de GitHub:

https://github.com/sofiaalvarezlopez/Preinforme_10_Electronica_para_Ciencias