



L01. Desarrollo de un sistema embebido simple.

Sistemas Embebidos.

Integrantes:

Ángel de Jesús Alonso Márquez

Carla Iveth Velasco Nieto

22 / Agosto / 2019

Aplicación 1) Temporizador Digital

Objetivos

Estudiar y comprender el funcionamiento y los usos de un temporizador digital así como experimentar con los valores preestablecidos y sus variables para observar la respuesta del sistema.

Que se implementa

El sistema se programa en C++ haciendo uso del IDE de Arduino con un micro controlador ATMEGA328P propio de la placa Arduino Uno. El temporizador funciona utilizando el reloj interno del micro controlador que cada cierto número de pulsos determinado por el usuario, reduce en un segundo el tiempo de temporizado ingresado previamente hasta llegar al valor de cero, un segundo antes de que esto pase se acciona un buzzer que funciona como una señal de notificación al usuario de que el tiempo del temporizador se ha agotado, el sistema cuenta con un botón de reset y dos botones de entrada asignados para aumentar minutos y segundos en el temporizador respectivamente, el tiempo de temporizado será desplegado en tiempo real en una pantalla de 4 dígitos de 7 segmentos, el micro controlador se conecta a esta mediante un convertidor Serial – Paralelo 74HC595 y el buzzer alarma como métodos de salida. Las características del proceso constan de un temporizado dinámico y sin pausas al momento de agregar más tiempo al temporizado en curso, opción de reset al presionar las teclas de minutos y segundos simultáneamente, capacidad de seleccionar el digito a modificar usando una subrutina de rastreo del desplegador.

Lista de partes

1x Placa de desarrollo Arduino Uno.

1x Desplegador de 4 dígitos de 7 segmentos.

1x Convertidor Serial – Paralelo 74HC595

3x Push buttom

9x Resistencia 1kΩ

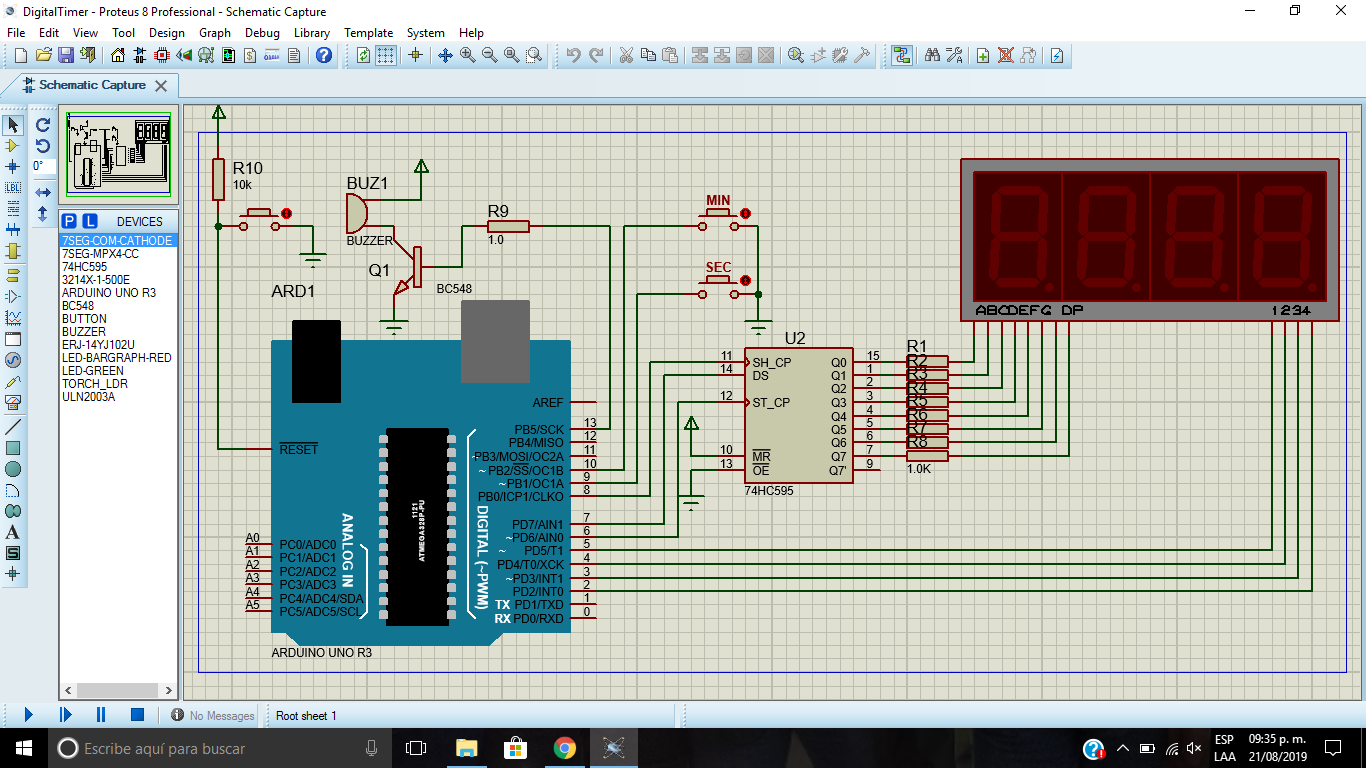
1x Resistencia 10kΩ

1x Transistor BC548

1x Buzzer

Conectores

Diagrama eléctrico



Listado del Código.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Temporizador Digital. Usa 4 dígitos de Siete Segmentos de cátodo común

// Tiene controles para Minutos y Segundos

//==============================================================================

#include <TimerOne.h>

const int Data = 7; // Conexiones entre el CI 74HC595 y la placa Arduino

const int Clock = 8; //

const int Latch = 6; //

const int SEG0 = 5; // Conexiones para manejar los cátodos

const int SEG1 = 4; //

const int SEG2 = 3; //

const int SEG3 = 2; //

const int Buzzer = 13; // Bocina o Buzzer

const int Min\_key = 10; // Teclas

const int Sec\_key = 9; //

//------------------------------------------------------------------------------

//-----[ VARIABLES ]-----

//------------------------------------------------------------------------------

int cc = 0;

char Value[4];

// Patrones para desplegar los dígitos (verificarlos):

const char SegData[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

int MIN = 0, SEC = 0, count = 10; // Variables y valores iniciales

//==============================================================================

//-----[ AJUSTES GENERALES ]-----

//==============================================================================

void setup() {

Serial.begin(9600); // Inicia comunicación serial

pinMode(Data, OUTPUT); // Terminales para manejar arreglo 7 segmentos

pinMode(Clock, OUTPUT); //

pinMode(Latch, OUTPUT); //

pinMode(SEG0, OUTPUT); //

pinMode(SEG1, OUTPUT); //

pinMode(SEG2, OUTPUT); //

pinMode(SEG3, OUTPUT); //

pinMode(Buzzer, OUTPUT); //

digitalWrite(Buzzer, LOW); // Apagado de la bocina o buzzer

pinMode(Min\_key, INPUT); // Tecla de Minutos

pinMode(Sec\_key, INPUT); // Tecla de Segundos

digitalWrite(Min\_key, HIGH); // Ajuste de "Pull-Ups" para las entradas

digitalWrite(Sec\_key, HIGH); //

cc = 0;

// Lo siguiente es un recurso proporcionado por una biblioteca: TimerOne

// Ajuste de un temporizador de 100,000 microsegundos (o de 0.1 segundo,

// o de 10 Hertz) => Un LED parpadearía 5 veces por segundo

Timer1.initialize(10000);

Timer1.attachInterrupt( timerIsr ); // Subrutina que atiende la Interrupción

}

//==============================================================================

//-----[ LAZO PRINCIPAL ]-----

//==============================================================================

void loop() {

char cMIN[4], cSEC[4];

if (count == 0) {

count = 10; // Esto es 1 segundo. 10 retardos de 100 milisegundos

//

if (MIN > 0) { // Si minutos son mayores a cero...

if (SEC == 0) { // Si se llegó al segundo "00"

MIN--;

SEC = 60;

}

}

SEC--; // Decremento de Segundos

// Revisar que el Temporizador está en cero

if (MIN == 0 && SEC == 1) { // Segundos mantenidos para evitar sonido

digitalWrite(Buzzer, HIGH); // Encendido del sonido

}

}

//----------------------------------------

if (MIN > 0 || SEC > 0) { // Si los Minutos y Segundos son mayores a "0"

count--; //

}

//----------------------------------------

if (digitalRead(Min\_key) == LOW) { // ¿Se oprimió la tecla "Minutos"?

delay(10); // Para evitar "rebotes"

MIN++; // Incrementa minutos

digitalWrite(Buzzer, LOW); // Apagar bocina

}

//----------------------------------------

if (digitalRead(Sec\_key) == LOW) { // ¿Se oprimió la tecla de "Segundos"?

delay(10); // Para evitar "rebotes"

SEC++; // Incrementar segundos

if (SEC > 59) { // 60 segundos - 1 minuto

MIN++; // Ajuste...

SEC = 0; // ...para reciclar

}

digitalWrite(Buzzer, LOW); // Apagar bocina o buzzer

}

//----------------------------------------

// Si se oprimieran simultáneamente ambas teclas, se Limpia el Timer

if (digitalRead(Sec\_key) == LOW && digitalRead(Min\_key) == LOW) {

MIN = 0;

SEC = 0;

digitalWrite(Buzzer, HIGH); // Prender bocina o buzzer

delay(500);

digitalWrite(Buzzer, LOW);

}

//----------------------------------------

// Muestra de conteo en el desplegador de 7 segmentos

sprintf(cMIN, "%02d", MIN); // Arreglo en ASCII

sprintf(cSEC, "%02d", SEC); // Arreglo en ASCII

Serial.println(cMIN); // Imprimir el conteo en terminal serial

// (depuración)

Value[0] = cMIN[0] & 0x0F; // Remover el "nibble" superior (AND con 0x0F)

Value[1] = cMIN[1] & 0x0F; // Por ejemplo, de 0x32 (ASCII)...

Value[2] = cSEC[0] & 0x0F; // ...sólo necesitamos el "2"

Value[3] = cSEC[1] & 0x0F;

delay(100);

}

//==============================================================================

//-----[ SUBRUTINAS ]-----

//==============================================================================

// INTERRUPCIÓN CAUSADA POR EL DESBORDAMIENTO DE TIMER 1

// (Para el desplegador de 7 segmentos)

//==============================================================================

void timerIsr() {

cc++; // Incremento del contador de "cátodos", asignados a dígitos

if (cc == 5) { // De momento sólo se tienen 4 dígitos

cc = 1;

}

Scanner(); // Realiza el rastreo

TCNT0 = 0xCC; // Reinicia "Timer 1" para ocurrencia próxima Interrupción

}

//==============================================================================

// GENERACIÓN DE LOS DÍGITOS

//==============================================================================

void DisplayDigit(char d) {

int i;

for (i = 0; i < 8; i++) { // Corrimiento, bit a bit, en...

if ((d & 0x80) == 0x80) { // ... registro de corrimiento

digitalWrite(Data, HIGH);

}

else {

digitalWrite(Data, LOW);

}

d = d << 1; // Corrimiento...

// Pulsos de Reloj

digitalWrite(Clock, LOW);

digitalWrite(Clock, HIGH);

}

// Captura del dato

digitalWrite(Latch, LOW);

digitalWrite(Latch, HIGH);

}

//==============================================================================

// FUNCIÓN DE RASTREO DEL DESPLEGADOR

//==============================================================================

void Scanner() {

switch (cc) // Selección del dígito que se actualiza

{

case 1:

digitalWrite(SEG3, HIGH);

DisplayDigit(SegData[Value[0]]);

digitalWrite(SEG0, LOW);

break;

case 2:

digitalWrite(SEG0, HIGH);

DisplayDigit(SegData[Value[1]] | 0x80); // Punto Decimal

digitalWrite(SEG1, LOW);

break;

case 3:

digitalWrite(SEG1, HIGH);

DisplayDigit(SegData[Value[2]]);

digitalWrite(SEG2, LOW);

break;

case 4:

digitalWrite(SEG2, HIGH);

DisplayDigit(SegData[Value[3]]);

digitalWrite(SEG3, LOW);

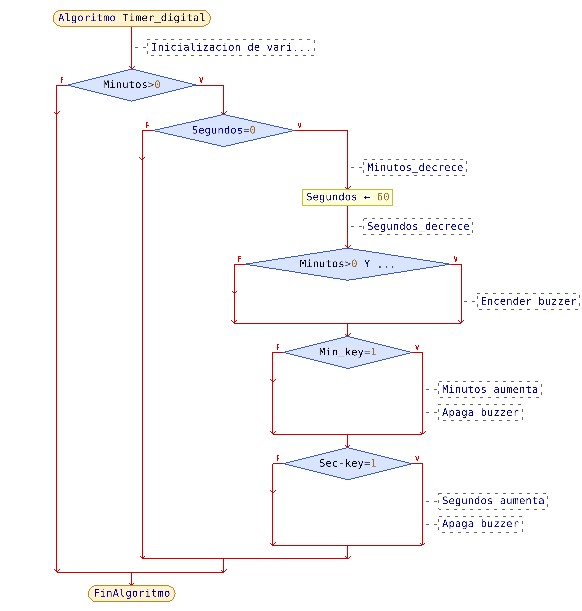
break;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Diagrama de Flujo.



Conclusión

A pesar de que un temporizador puede parecer una práctica sencilla, esta implica diversos procesos y subrutinas que hacen el sistema más complejo a nivel código pero son fundamentales para el control y operación de los dispositivos de despliegue de información como el Desplegador de 7 segmentos, por lo que es fundamental comprender y estudiar a detalle la función de cada uno de los comandos para el control de todo el proceso.

**Objetivo**

Diseñar un sistema contador de revoluciones por minuto adecuado para odómetros en bicicletas y similares, empleando un generador de pulsos para simular la entrada.

**Qué se implementa**

Un generador de pulsos el cual simula la entrada.

**Entradas**

Generador de pulsos.

**Salidas**

Display para visualizar los datos

**Procesos**

Se declaran las variables, a algunas se les asigna un valor inicial de 0 y para las salidas se establece la conexión entre un pin del circuito integrado al desplegador.

En el void setup, se hacen los ajustes generales. Se establece el manejador de interrupciones del timer; se definen las salidas. Se inicia el rastreo del desplegador en 0. Se le asigna un valor de 10000 al timer 1 (0.1 segundos).

En el void loop (lazo principal), hay un ciclo if, el cual se repite hasta que el resultado de la resta currentMillis – previousMillis, sea igual o mayor a periodo principal (MainPeriod).

Se establecen dos contadores que inician en 0.

Se calcula la frecuencia con la formula establecida en el código: Freq = 1e6 / float(\_duration).

Ahora se debe convertir la frecuencia a RPM (revoluciones por minuto)

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Para la generación de datos, se establece una variable de tipo carácter (char) y una de tipo entero (int).

Se inicia un ciclo for en donde i es igual a 0, va en aumento mientras tenga un valor menor a 8.

Dentro del ciclo for se inicia un ciclo if (d & 0x80) == 0x80), si esta condición se cumple, entonces se escribe en la salida un valor, si no se cumple esta condición, el valor queda bajo.

Se le da pulso al reloj.

Para el temporizador 1: anteriormente se inicializó la variable cc en 0, ahora esta variable irá en aumento hasta ser igual a 5.

Para el display: Se inicia un switch con 4 casos.

Para el caso 1: se escribe un valor a la salida para SEG3, mientras que SEG0 queda apagado.

Para el caso 2: ahora el valor se escribe para SEG0 y SEG1 queda apagado.

Para el caso 3: Se escribe un valor para SEG1 y SEG2 queda apagado.

Para el caso 4: Se escribe un dato para SEG2y SEG3 se apaga.

Para las interrupciones del timer: la duración será igual al valor actual menos el anterior, y en cada ciclo el valor anterior tomará el dato del valor actual y los pulsos irán en aumento.

**Características**

Se utiliza un generador de pulsos como entrada, el cual inicia en 0 y tiene una amplitud de 0.5.

Se usa un convertidor de serie a paralelo, de 8 bits, la ventaja de este circuito integrado es poder utilizar salidas a partir de 3 pines del microcontrolador.

El valor de los pulsos del generador de señales va cambiando, por lo que las revoluciones por minuto que vemos en el display varían, van de 0 aun máximo de 58 revoluciones por minuto.

**Lista de partes**

Un Arduino uno

Convertidor analógico/digital

9 resistencias de 1K

Una LCD 7x4

**Diagrama eléctrico**

Imagen que contiene texto

Descripción generada automáticamente

**Listado del código**

// Sistema Contador de Revoluciones Por Minuto

// Adecuado para odómetros en bicicletas y similares

// Emplea un generador de pulsos para simular la entrada

//------------------------------------------------------------------------------

#include <TimerOne.h>

#define MainPeriod 100

long previousMillis = 0; // Almacena el tiempo de fin del ciclo

volatile unsigned long duration = 0; // Acumulador del Ancho del Pulso

volatile unsigned int pulsecount = 0;

volatile unsigned long previousMicros = 0;

const int Clock = 10; // Conexiones para el CI 74HC595

const int Data = 9;

const int Latch = 8;

const int SEG0 = 7;

const int SEG1 = 6;

const int SEG2 = 5;

const int SEG3 = 4;

int cc = 0;

char Value[4];

const char SegData[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};

//==============================================================================

//-----[ AJUSTES GENERALES ]-----

//==============================================================================

void setup() {

// Iniciaización de elementos que manejan el desplegador

attachInterrupt(0, myinthandler, RISING); // Manejador de Interrupción Timer 1

pinMode(Data, OUTPUT);

pinMode(Clock, OUTPUT);

pinMode(Latch, OUTPUT);

pinMode(SEG0, OUTPUT);

pinMode(SEG1, OUTPUT);

pinMode(SEG2, OUTPUT);

pinMode(SEG3, OUTPUT);

// Inicalizar el rastreo del Desplegador de Dígitos

cc = 0;

// Ajuster de Timer de un valor de 100000 microsegundos (0.1 sec - o 10Hz)

// => Un LED parpadearía 5 veces por segundo, Onn-Off.

Timer1.initialize(10000);

Timer1.attachInterrupt( timerIsr ); // Subrutina servicio a la interrupción

}

//==============================================================================

//-----[ LAZO PRINCIPAL ]-----

//==============================================================================

void loop() {

char rpm[4];

int RPM;

unsigned long currentMillis = millis();

if (currentMillis - previousMillis >= MainPeriod) {

previousMillis = currentMillis;

// need to bufferize to avoid glitches... necesita amortiguar para evitar problemas técnicos

unsigned long \_duration = duration;

unsigned long \_pulsecount = pulsecount;

duration = 0; // clear counters...contadores claros

pulsecount = 0;

float Freq = 1e6 / float(\_duration); //Duration is in uSecond so it is 1e6 / T...

//La duración es en micro Segundo, por lo que es 1e6 / T

Freq \*= \_pulsecount; //calculate F... calcular F

//Convert Freq to RPM... Convertir Freq a RPM

RPM = Freq \* 60.0 \* 1.0; //RPM = Freq \* 60 \* (Number of pulses per revolution)... Número de pulsos por revolución

//Display Freq on Segments

sprintf(rpm, "%04d", RPM); //We get ASCII array in Volt... Obtenemos la matriz ASCII en volts

Value[0] = rpm[0] & 0x0F; //Anding with 0x0F to remove upper nibble

Value[1] = rpm[1] & 0x0F; //Ex. number 2 in ASCII is 0x32 we want only 2

Value[2] = rpm[2] & 0x0F;

Value[3] = rpm[3] & 0x0F;

}

delay(200);

}

//==============================================================================

//-----[ GENERACIÓN de DATOS ]-----

//==============================================================================

void DisplayDigit(char d) {

int i;

for (i = 0; i < 8; i++) { //Shift bit by bit data in shift register... Desplazar datos bit a bit en el registro de desplazamiento

if ((d & 0x80) == 0x80) {

digitalWrite(Data, HIGH);

}

else

{

digitalWrite(Data, LOW);

}

d = d << 1;

//Give Clock pulse... Darle pulso al reloj

digitalWrite(Clock, LOW);

digitalWrite(Clock, HIGH);

}

//Latch the data

digitalWrite(Latch, LOW);

digitalWrite(Latch, HIGH);

}

//==============================================================================

// TIMER 1 OVERFLOW INTTERRUPT FOR DISPALY... TEMPORIZADOR 1 INTERRUMPIDO DE DESBORDAMIENTO PARA LA VISUALIZACIÓN

//==============================================================================

void timerIsr() {

cc++;

if (cc == 5) // Sólo se tienen 4 dígitos

{cc = 1;}

Scanner();

TCNT0 = 0xCC;

}

//==============================================================================

// SCAN DISPLAY FUNCTION

//==============================================================================

void Scanner() {

switch (cc) {

// Selección del dígito a actualizar

case 1:

digitalWrite(SEG3, HIGH);

DisplayDigit(SegData[Value[0]]);

digitalWrite(SEG0, LOW);

break;

case 2:

digitalWrite(SEG0, HIGH);

DisplayDigit(SegData[Value[1]]);

digitalWrite(SEG1, LOW);

break;

case 3:

digitalWrite(SEG1, HIGH);

DisplayDigit(SegData[Value[2]]);

digitalWrite(SEG2, LOW);

break;

case 4:

digitalWrite(SEG2, HIGH);

DisplayDigit(SegData[Value[3]]);

digitalWrite(SEG3, LOW);

break;

}

}

//==============================================================================

// MANEJADOR de INTERRUPCIONES del TIMER

//==============================================================================

void myinthandler(){

unsigned long currentMicros = micros();

duration += currentMicros - previousMicros;

previousMicros = currentMicros;

pulsecount++;

}

**Conclusiones**

Es una práctica completa, aunque se presta para hacer diversas modificaciones, por ejemplo, utilizar una LCD de mayor tamaño y otro dispositivo que nos permita tener una mayor numero de datos con los mismos pines del microcontrolador.

**Aplicación 3) Controlador de luces de trafico**

**Objetivos**

Analizar el funcionamiento de un sistema de luces de trafico basado en temporizado.

**Que se implementa**

El sistema se programa en C++ haciendo uso del IDE de Arduino con un micro controlador ATMEGA328P propio de la placa Arduino Uno. Mediante 4 arreglos de 3 LEDs cada uno (Verde, Amarillo y rojo) emulando los colores de un semáforo, en el tiempo cero todos los semáforos están en rojo excepto el semáforo 1 que se encuentra en verde, pasado un tiempo “x” el semáforo 1 cambia de estado de verde a amarillo durante un breve tiempo “y” y terminando este el semáforo 1 cambia a Rojo mientras que el semáforo 2 cambia a verde, en este punto se vuelve a repetir todo el proceso, el estado verde del semáforo 2 durara un tiempo “x” , luego cambiara a estado amarillo durante un tiempo “y” para finalmente volver al estado rojo y repetir el proceso con el resto de semáforos, al terminar con el semáforo 4 en rojo se repite nuevamente el proceso, es un sistema sencillo y no cuenta con un método de entrada de información aunque si es posible resetear el sistema mediante la entrada de reset propia de la placa Arduino Uno , para el despliegue de información se hace único uso de los 4 arreglos de LED’s. Sistema sencillo y automatizado que puede variar sus características como tiempo de temporizado “x” y “y” mediante software y es fácilmente expandible para dar soporte a mas arreglos de LED para operar como semáforos adicionales, de igual manera se puede modificar agregando 4 botones para para variar los tiempos “x” y “y” de manera más intuitiva para el usuario, un desplegador para visualizar los cambios y un último botón que opere como señal de guardado de información.

**Lista de partes**

1x Placa de desarrollo Arduino Uno

1x Resistencia 10kΩ

1x Push Buttom

4x Led Verde

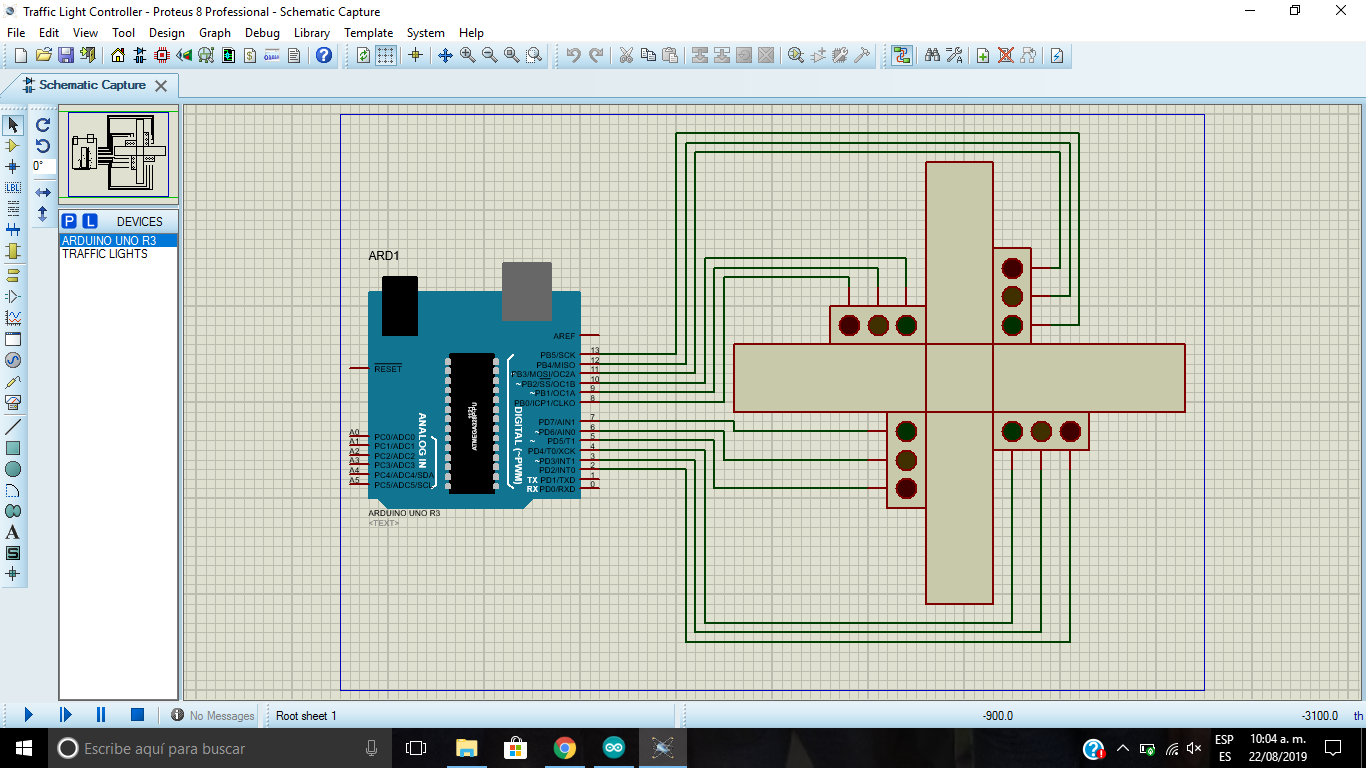
4x Led Rojo

4x Led Amarillo

12x Resistencia 220Ω

Conectores

**Diagrama eléctrico**



**Listado del Código**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Controlador de Semáforo

// Especificaciones simples

//

//==============================================================================

// Conexiones para las luces de tráfico (de momento con LEDs)

const int R\_1 = 11;

const int Y\_1 = 12;

const int G\_1 = 13;

const int R\_2 = 8;

const int Y\_2 = 9;

const int G\_2 = 1;

const int R\_3 = 5;

const int Y\_3 = 6;

const int G\_3 = 7;

const int R\_4 = 2;

const int Y\_4 = 3;

const int G\_4 = 4;

//-----[ AJUSTES GENERALES ]-----

void setup() {

// Todas las terminales de LEDs son salidas

pinMode(R\_1, OUTPUT);

pinMode(Y\_1, OUTPUT);

pinMode(G\_1, OUTPUT);

pinMode(R\_2, OUTPUT);

pinMode(Y\_2, OUTPUT);

pinMode(G\_2, OUTPUT);

pinMode(R\_3, OUTPUT);

pinMode(Y\_3, OUTPUT);

pinMode(G\_3, OUTPUT);

pinMode(R\_4, OUTPUT);

pinMode(Y\_4, OUTPUT);

pinMode(G\_4, OUTPUT);

// Encendido de los LEDs ROJOS

digitalWrite(R\_1, HIGH);

digitalWrite(Y\_1, LOW);

digitalWrite(G\_1, LOW);

digitalWrite(R\_2, HIGH);

digitalWrite(Y\_2, LOW);

digitalWrite(G\_2, LOW);

digitalWrite(R\_3, HIGH);

digitalWrite(Y\_3, LOW);

digitalWrite(G\_3, LOW);

digitalWrite(R\_4, HIGH);

digitalWrite(Y\_4, LOW);

digitalWrite(G\_4, LOW);

}

//-----[ LAZO PRINCIPAL ]-----

void loop(){

int YellowTime=2000; // 2 segundos

int GreenTime=30000; // 30 segundos

// Etapa 0.

digitalWrite(R\_1,LOW);

digitalWrite(Y\_1,LOW);

digitalWrite(G\_1,HIGH);

delay(GreenTime);

// Etapa 1.

digitalWrite(R\_1,LOW);

digitalWrite(Y\_1,HIGH);

digitalWrite(G\_1,LOW);

delay(YellowTime);

digitalWrite(R\_1,HIGH);

digitalWrite(Y\_1,LOW);

// Etapa 2.

digitalWrite(R\_2,LOW);

digitalWrite(Y\_2,LOW);

digitalWrite(G\_2,HIGH);

delay(GreenTime);

digitalWrite(R\_2,LOW);

digitalWrite(Y\_2,HIGH);

digitalWrite(G\_2,LOW);

delay(YellowTime);

digitalWrite(R\_2,HIGH);

digitalWrite(Y\_2,LOW);

// Etapa 3.

digitalWrite(R\_3,LOW);

digitalWrite(Y\_3,LOW);

digitalWrite(G\_3,HIGH);

delay(GreenTime);

digitalWrite(R\_3,LOW);

digitalWrite(Y\_3,HIGH);

digitalWrite(G\_3,LOW);

delay(YellowTime);

digitalWrite(R\_3,HIGH);

digitalWrite(Y\_3,LOW);

// Etapa 4.

digitalWrite(R\_4,LOW);

digitalWrite(Y\_4,LOW);

digitalWrite(G\_4,HIGH);

delay(GreenTime);

digitalWrite(R\_4,LOW);

digitalWrite(Y\_4,HIGH);

digitalWrite(G\_4,LOW);

delay(YellowTime);

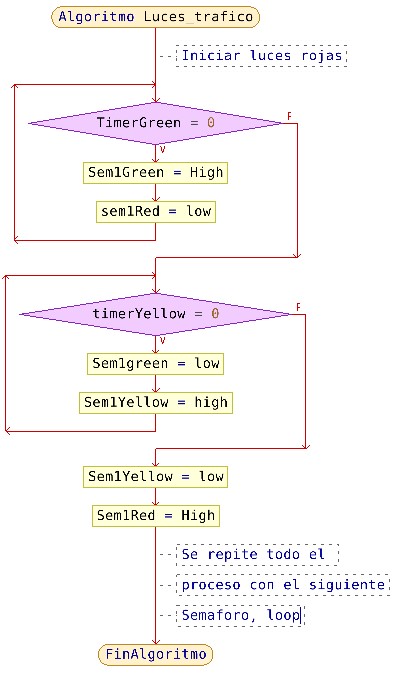
digitalWrite(R\_4,HIGH);

digitalWrite(Y\_4,LOW);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Diagrama de flujo**



**Conclusión**

Es una práctica sencilla de la cual se puede rescatar como se construyen los sistemas automatizados haciendo uso de slots de tiempo sincronizados entre los semáforos que forman parte de por ejemplo, una ciudad.