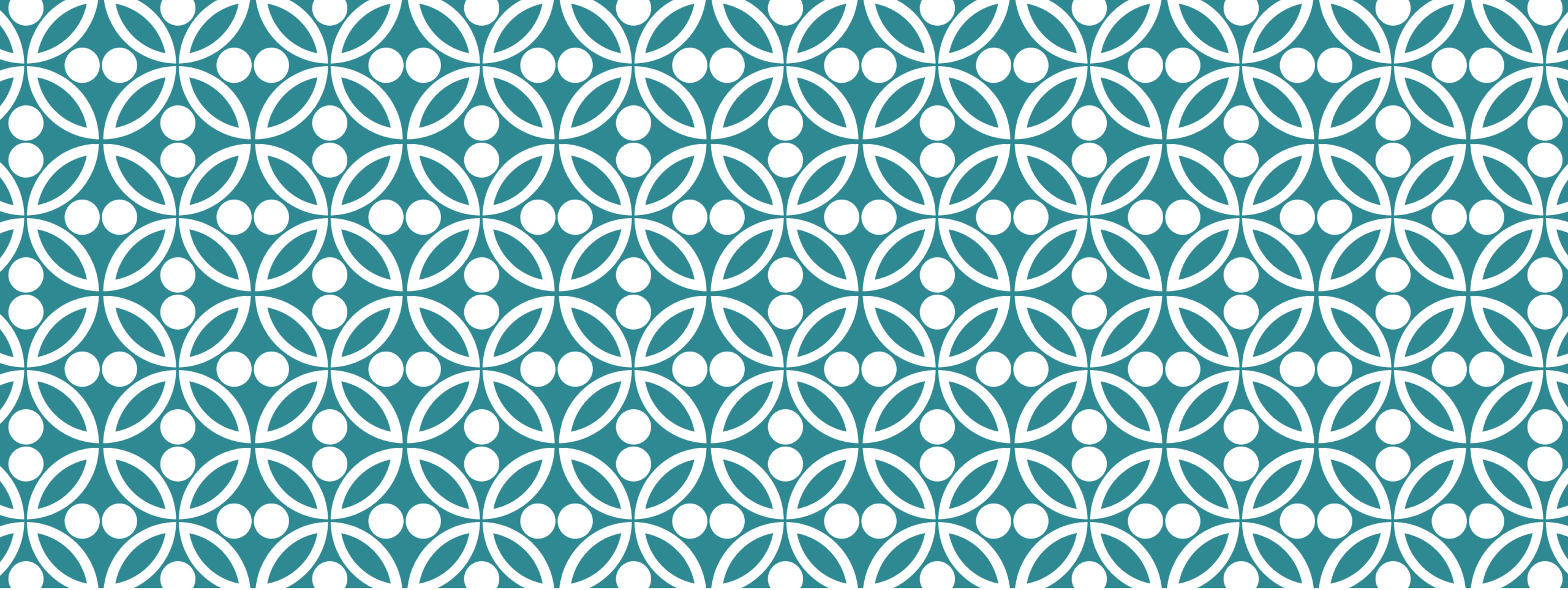
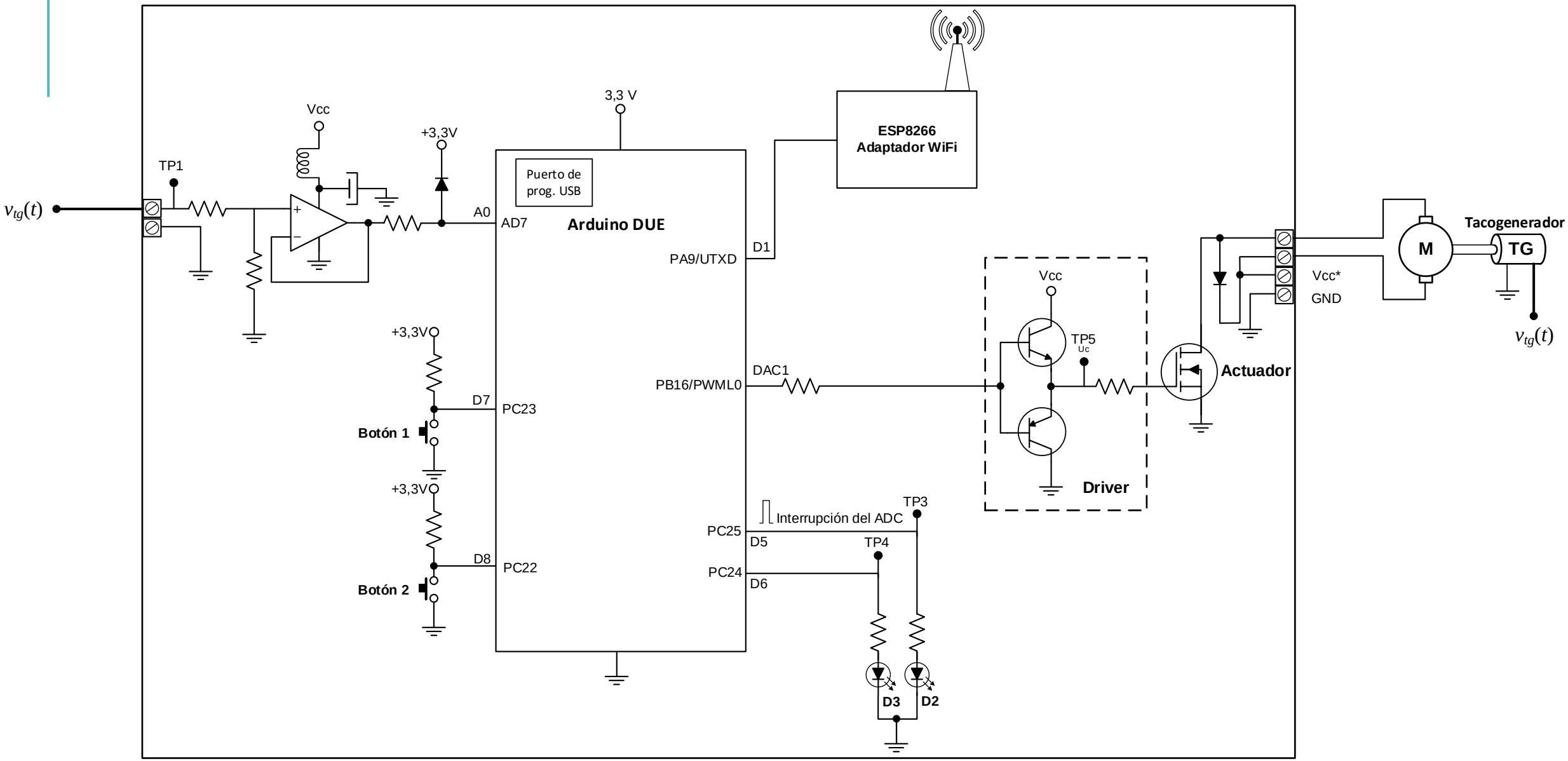


ARDUINO DUE

Sistemas de Control 2

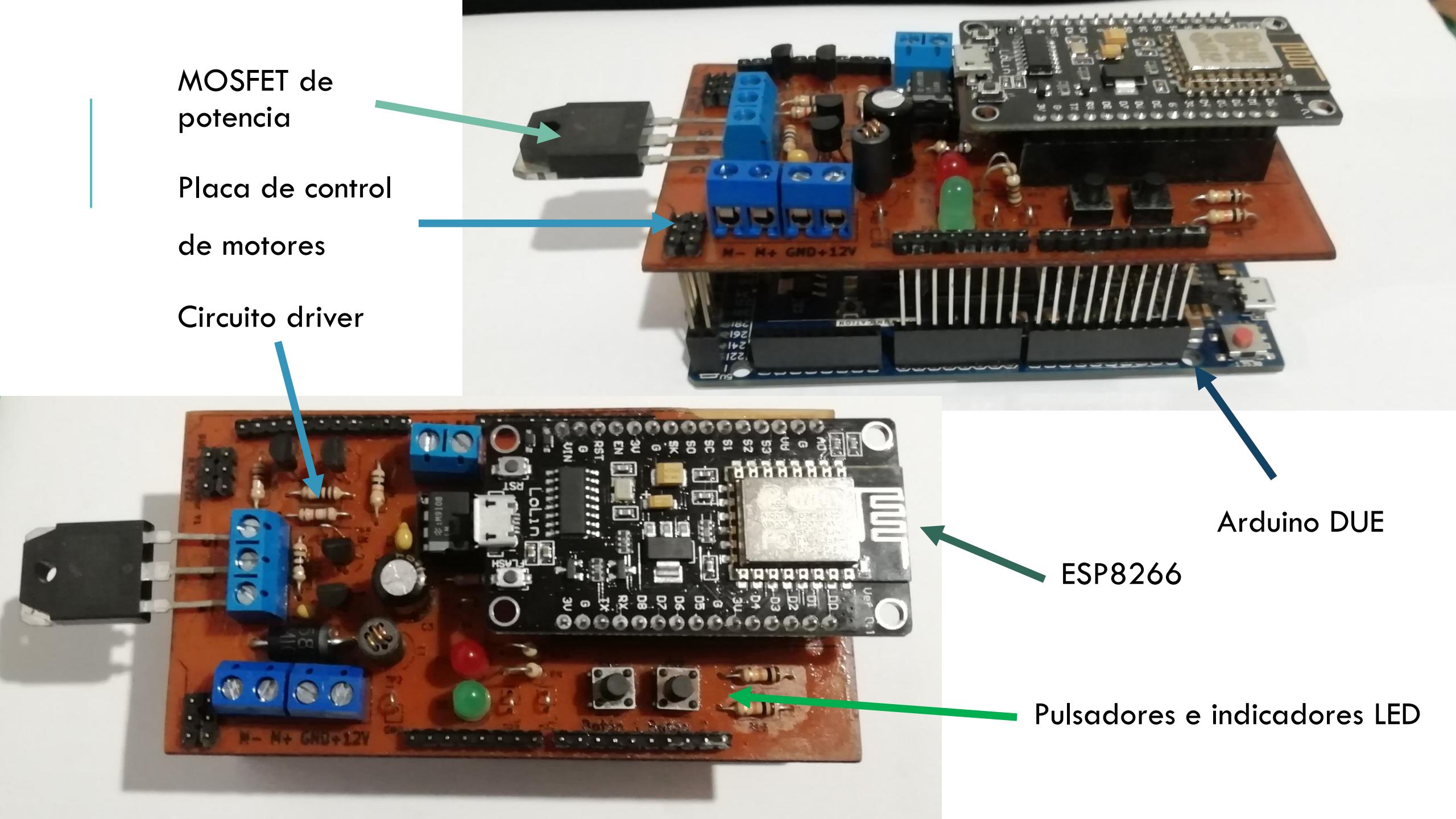
ESQUEMA SIMPLIFICADO



MOSFET de potencia

Placa de control de motores

Circuito driver



Arduino DUE

ESP8266

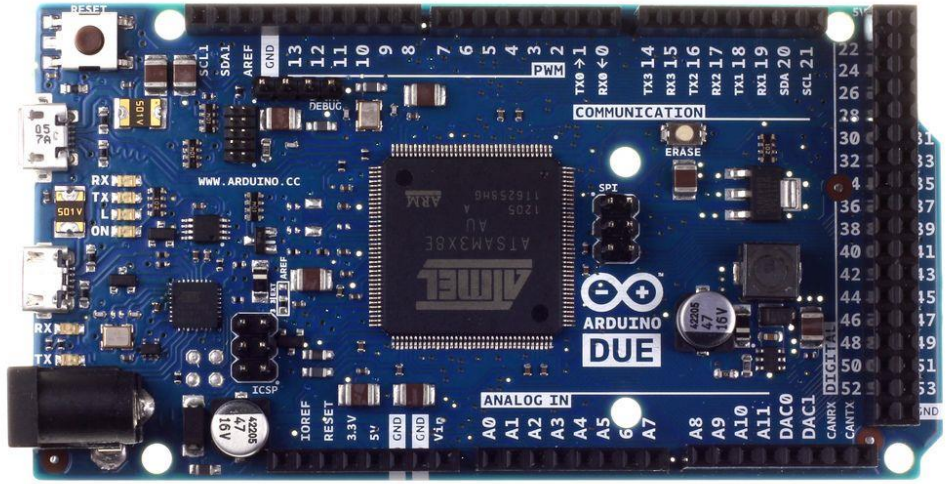
Pulsadores e indicadores LED

ARDUINO DUE

Basado en la CPU ARM de 32 bits Atmel SAM3X8E Cortex M3.

Características destacables:

- Reloj de CPU de 84 MHz.
- 96 KB de RAM (2 bancos 64KB + 32KB)
- 512 KB de memoria flash
- Aritmética de punto fijo en hardware. Punto flotante emulado
- ADC de 12 bits con 16 canales hasta 1 Msps
- DAC de 12 bits con 2 canales hasta 2 Msps
- Hasta 8 canales PWM, cada uno con 2 salidas complementarias
- 54 pines de E/S digitales
- Voltaje de operación: 3,3 V
- Voltaje de alimentación: 7 – 12 V



SETUP

```
void setup() {
  // Llamada a funciones de configuración
  GPIO_setup();
  ADC_setup();
  PWM_setup();
  Serial.begin(115200);
}
```

MAIN LOOP

```
void loop() {
  static bool bk, bkm1 = true, b2k, b2km1 = true;
  bk = PIO_Get(PIO_BOTON,PIO_INPUT,B_START);
  b2k = PIO_Get(PIO_BOTON,PIO_INPUT,B_STOP);
  if(bk == 1 && bkm1 == 0){
    // Se produjo una pulsación del botón de
arranque
    run = 1;
    start_saving = 1;
  }
  if(b2k == 1 && b2km1 == 0){
    // Se produjo una pulsación del botón de
parada
    run = 0;
  }
  b2km1 = b2k;
  bkm1 = bk;
  if(data_saved){
    data_saved = 0;
    Serial.print("start");
    Serial.print('\n');
    for(uint16_t i = 0 ; i < N_SAVED ; i++){
      Serial.print(s_ref[i],3);
      Serial.print(',');
      Serial.print(s_y[i],3);
      Serial.print(',');
      Serial.print(s_u[i],3);
      Serial.print('\n');
      delay(10);
    }
    Serial.print("end");
    Serial.print('\n');
  }

}
```

IRQ DEL ADC

```
void ADC_Handler(void){
  counter_N++;
  if(ADC->ADC_ISR & ADC_ISR_EOC7){
    adc_7 = ADC->ADC_CDR[7]; //Se lee el valor del
registro de resultado de conversión del canal 7
    if(counter_N == N_Tpwm){
      PIO_Set(PIO_LED,LED_1);
      if(run){
        ref = 1.f; // Referencia en escalón
        // Normalización
        yk = ((float) adc_7)/3276.f;
        ek = ref - yk;
        // Cálculo de la acción de control
        uk= upi_km1 + Kpi*ek + aKpi*ekm1;
        PWM_update_duty_cycle(uk);
        PIO_Clear(PIO_LED,LED_1);
        ekm1 = ek;
        upi_km1 = uk;
        // Si se dio la señal de guardado de
variables, se guardan utilizando el indice hasta
que los arreglos estén llenos
        if(start_saving && !saving){
          start_saving = 0;
          saving = 1;
          index = 0;
        }
        if(saving){
          s_ref[index] = ref;
          s_y[index] = yk;
          s_u[index] = uk;
          index++;
          if(index>=N_SAVED){
            saving = 0;
            data_saved = 1;
          }
        }
      }
      counter_N = 0; // Se limpia el contador que
genera el periodo de muestreo
    }
  }

}
```

ACCIÓN DE CONTROL ADC_HANDLER(VOID)

Definir variables necesarias para el cálculo de la acción de control.

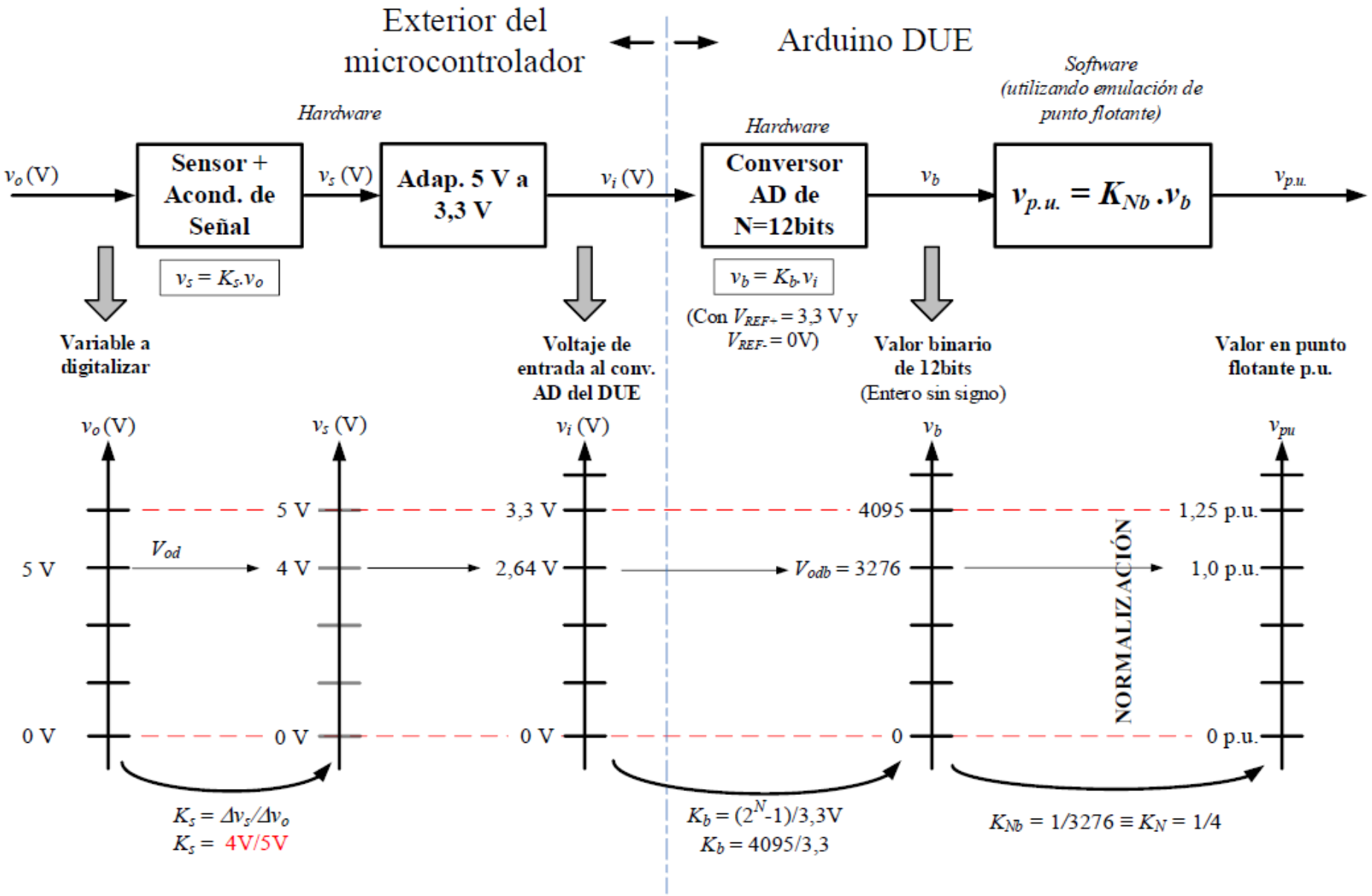
Establecer Referencia Deseada.

Escribir el código para el cálculo de la acción de control.

Actualizar variables.

Si el sistema no arrancó, o se presionó el botón de parada, se limpian las variables involucradas.

CONVERSIÓN DE ADC Y NORMALIZACIÓN



CAMBIAR FRECUENCIA DE MUESTREO ADC_SETUP(VOID)

$$\frac{1}{f_{muestreo}} = T_{muestreo} = T_{tracking} + T_{conv}$$

f_{ADC} = ADCClock = MCK/((PRESCAL + 1)*2)

Está configurado: $\mathrm{MCK} = 84\,\mathrm{MHz}$ y $\mathrm{PRESCAL} = 1 \Rightarrow f_{ADC} = \mathbf{21\,MHz}$

Según el fabricante se tardan 20 ciclos en hacer la conversión ($T_{conv} = \frac{20}{f_{ADC}} = 0{,}95\,\mu s$)

Despejan $T_{tracking} = T_{muestreo} - T_{conv}$

Configuración: $TRACKTIM = \frac{T_{tracking} \cdot f_{ADC}}{PRESCAL} - 1$

```
ADC->ADC_MR |= ADC_MR_TRACKTIM(**acá el valor**)
```

CAMBIAR FRECUENCIA DE PWM PWM_SETUP(VOID)

$$f_{PWM} = \frac{MCK}{CPRD \cdot PRESC \cdot DIVA}$$

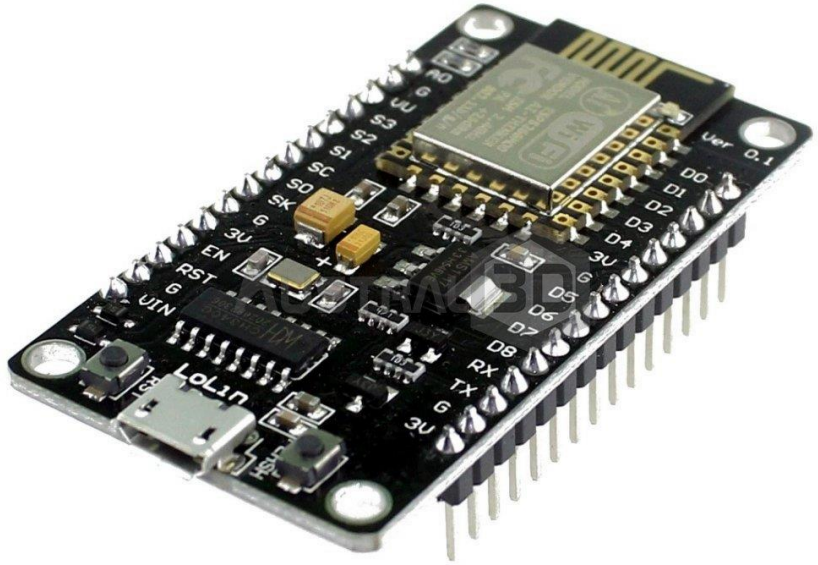
CPRD: define el número de ciclos que componen un ciclo completo de la señal PWM (valor máximo del contador).

PRESC: 2, 4, 8, 16, 32, 64, 128, etc.

DIVA: puede ser cualquier número entero, utilizado para dividir aún más la frecuencia del reloj.

Existen #define para cada uno de estos valores en el programa

ESP8266



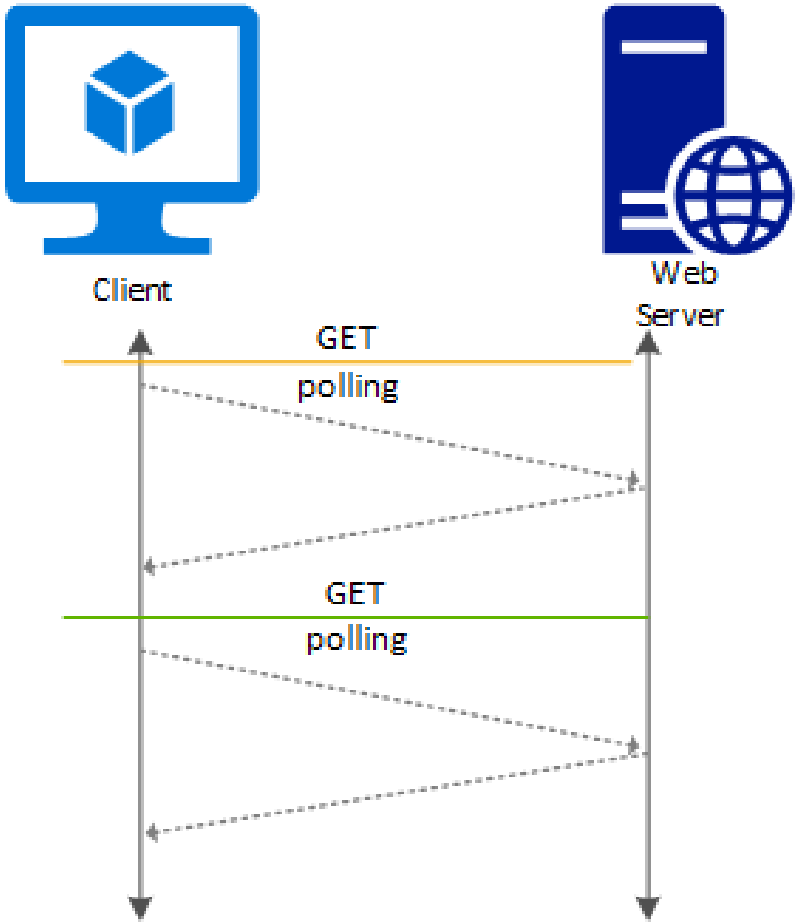
Chip de bajo costo con capacidad de comunicación WiFi, basado en un CPU RISC de 32-bit: Tensilica Xtensa LX106.

Características destacables:

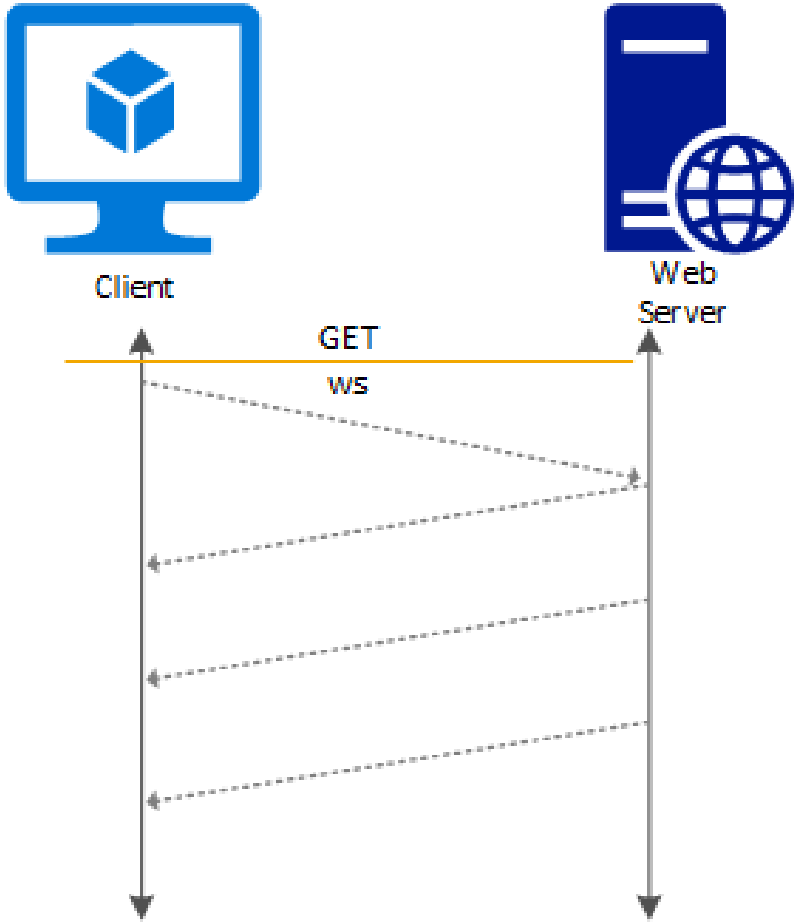
- Conexión WiFi IEEE 802.11 b/g/n : permite generar un servidor web desde el propio dispositivo.
- Reloj de CPU de 80 MHz
- 64 KB de RAM de instrucciones y 96 KB de RAM para datos.
- Memoria flash QSPI de 4 MB para almacenamiento de programa y archivos de la página web.

WEBSOCKET

Protocolo de intercambio de datos en red de baja latencia full duplex. Permite comunicación en tiempo real.



HTTP



WebSocket

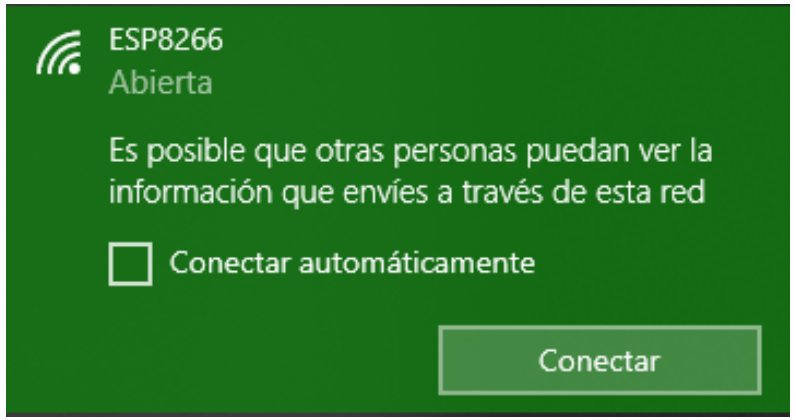
SETUP

```
void setup(void){
  Serial.begin(115200);

  if(!LittleFS.begin()) // Monta la partición de la memoria Flash que almacena la página web
  {
    Serial.println("\nAn error has occurred while mounting LittleFS");
  }
  Serial.println("\nLittleFS mounted successfully");

  ConnectWiFi_AP();
  WiFi.softAPIP();  // Inicializa la red WiFi en modo AP

  InitWebSockets(); // Inicia el servidor de WebSockets
  InitServer();     // Inicia el servidor web
}
```

MAIN LOOP

```
void loop(void){
  webSocket.loop(); // Revisa el estado del servidor de WebSockets y acepta conexiones nuevas
  String message;
  bool send_data = false;
  while(Serial.available()>0){  // Si existe comunicación serial entrante guarda el mensaje
    message += String((char)Serial.read());
    send_data = true;
  }
  if(send_data){    // Envía el mensaje guardado como texto desde el servidor de WebSocket
    webSocket.broadcastTXT(message);
    send_data = false;
    }
}
```

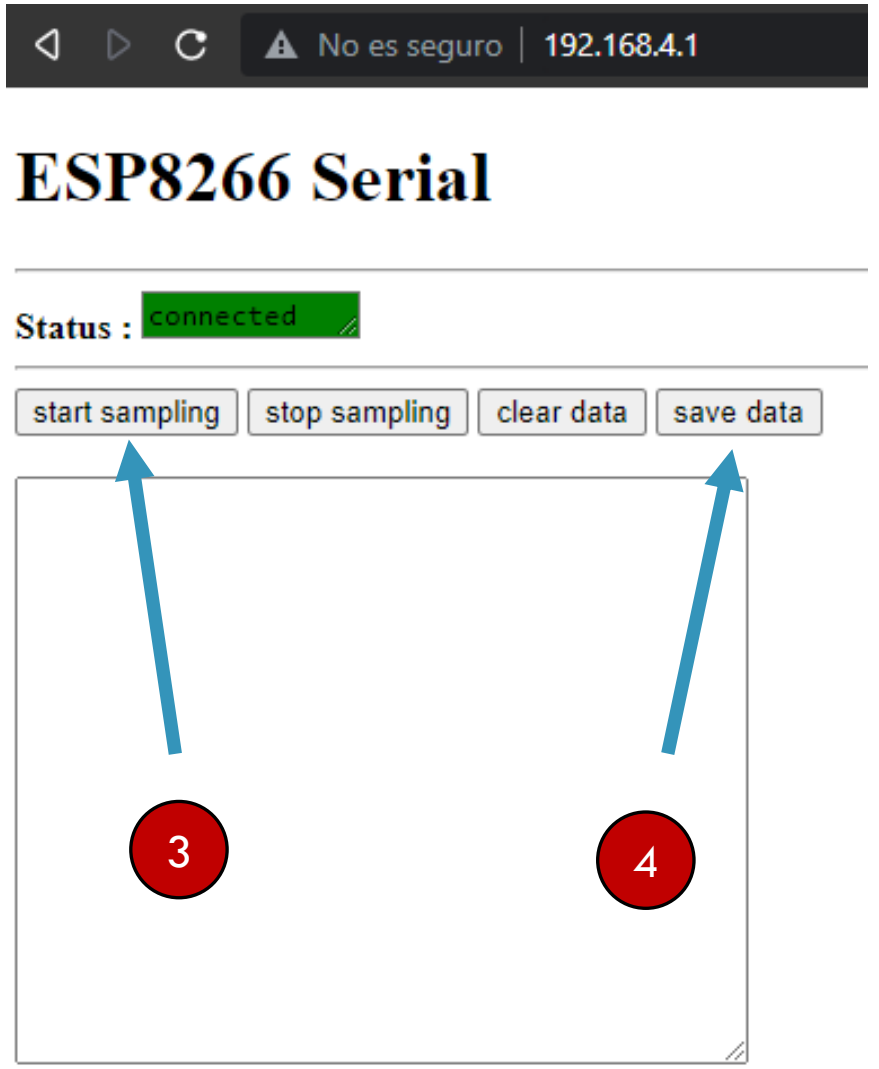
CONEXIÓN Y RECEPCIÓN DE DATOS



1

2

192.168.4.1



PUNTOS DE PRUEBA