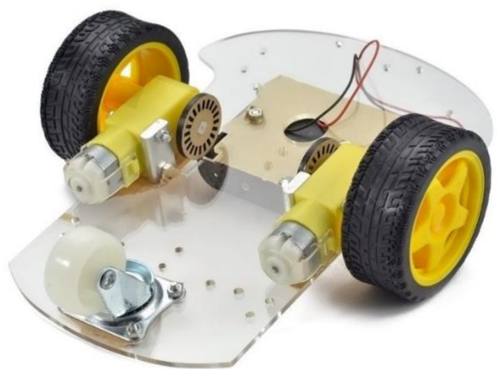
UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	FECHA	NOVIEMBRE 2016
APELLIDOS, NOMBRE	SOLUCIÓN	GRUPO-LAB.	

PRUEBA DE EVALUACIÓN INTERMEDIA 1

CUESTIÓN 1

Se propone analizar y completar el diseño de un sistema digital para implementar un robot de tracción diferencial. El procesado se realiza en una tarjeta MiniDK2 basada en el LPC1768 para el control de una plataforma móvil con 2 ruedas montadas con sendos motores DC conectados **en paralelo con un único puente en H**. Esto hace que el robot **nunca pueda girar**, siendo sólo necesario una única señal PWM para controlar la velocidad del desplazamiento, además de las necesarias para controlar el avance o retroceso.



Se proporciona el software que desarrolla la aplicación deseada en **el Anexo 1**, y parte de la conexión del hardware necesario en la Fig. 1 que es necesario **completar a partir del código proporcionado**.

Para hacer el análisis pedido se describen a continuación los elementos hardware del diseño a conectar a la MiniDK2:

- En este caso ambos motores tienen la misma excitación no lineal (y por tanto giran siempre a la misma velocidad W), a través de un único puente H controlado de forma UNIPOLAR (Ver Anexo 2) con entrada de habilitación EN por una única señal PWM y dos señales de dirección INx que deben tener valor digital inverso para no activar simultáneamente las ramas del puente: motor parado, W=0, con ciclo de trabajo D[1] de la señal PWM nulo, D=0%.
- Un encoder construido a partir de un disco ranurado y un switch óptico (ver **Anexo 3**), permite medir la velocidad y el espacio recorrido simplemente a partir Nº de pulsos recibidos y el diámetro de la rueda.
- Además, en el frontal de la plataforma se ha instalado un sensor de distancia analógico, pero con una función de variación lineal, que proporciona una tensión de salida V inversamente proporcional a la distancia al obstáculo *d* más próximo al sensor, tal que $\mathbf{d[cm]=-150\frac{(V-2)}{1.5}}$. Nota: 0<V≤2V
- Finalmente a la salida del DAC se ha conectado un amplificador de audio para generar una señal audible consistente en un tono de una determinada frecuencia.
- El funcionamiento de la plataforma se controla con un **pulsador START** que se conecta a la entrada **/EINT0**.

El funcionamiento del sistema móvil es muy sencillo, pues sólo consta de 4 estados posibles que se describen en los siguientes puntos:

- Cuando se inicia el funcionamiento (activación del pulsador START) se arranca el movimiento de avance de la plataforma a velocidad constante ***W***, recorriendo un espacio ***e***, hasta que: el sensor de distancia detecte un objeto a una distancia *d≤10cm*; o durante **30s** como máximo. Tras ambas situaciones la plataforma se para (*W=0*).

- Si la parada se produce por la **detección del objeto** se genera un tono de ***f=1000Hz*** durante **5s** mientras la plataforma continúa parada. En caso de que la parada se haya producido por la NO detección del objeto, la plataforma se mantiene también parada, en este caso, en espera de que vuelva a activarse el pulsador START para iniciar de nuevo el funcionamiento.

---

[1] $D=T_H/ T_H + T_L$, siendo T_H el tiempo que la señal PWM está a nivel alto y T_L el que está a nivel bajo.

- Finalmente, y tras la parada por la **detección de un obstáculo**, la plataforma **retrocede un espacio *e/2*** a la misma velocidad ***W*** que en el avance, **mientras emite un tono de *f=500Hz***, durante el tiempo que se prolongue ese movimiento de retroceso, hecho lo cual la plataforma vuelve a pararse (*W=0*) y queda ya en espera de que vuelva a iniciarse el funcionamiento mediante la activación del pulsador START.

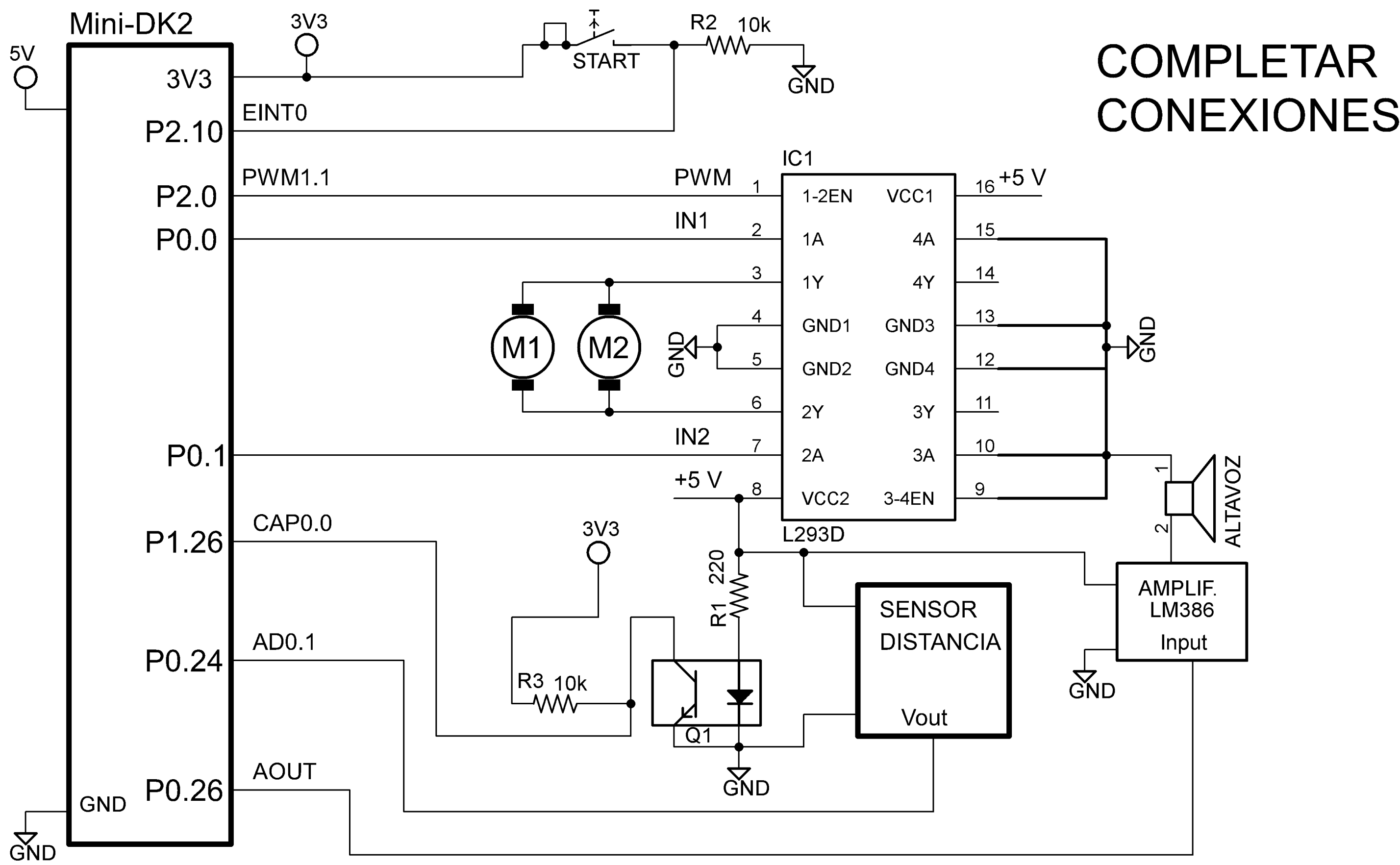


Fig. 1. Diagrama incompleto de conexión del sistema.

Para analizar el sistema digital así implementado se pide contestar de forma justificada a las preguntas que siguen a continuación, respecto a cada uno de los periféricos usados.

a) Complete sobre la Fig. 1, ***a medida que analiza el sistema***, la conexión de los pines del LPC1768 necesarios para asegurar el correcto funcionamiento de la aplicación descrita a partir del código (**Anexo I**).

b) Complete las líneas de código indicadas directamente en el **Anexo I**.

c) Respecto al ADC, indique el modo de funcionamiento, frecuencia de reloj, canales de conversión utilizados y el tiempo de conversión.

CH1 (AD0.1), FclkADC=25e6/(24+1)= 1 MHz, Modo BURST, Tc=65 µs

d) Respecto al PWM, indique qué canales se utilizan, cuál es el periodo de la señal PWM utilizada y cuál es el valor de su ciclo de trabajo para generar el movimiento tanto de avance como de retroceso.

PWM1.1 , T=MR0*Fpck/(PR+1) = **1 ms** , D=T_H/T= MR1/MR0= 2000/5000= **0,4** (40%)

e) Respecto al pin EINT0, indique cuál es el objetivo de la instrucción **Temporizar(...);** en su ISR .

Iniciar un periodo de 30 s durante el cual el móvil puede avanzar hasta encontrar un obstáculo. La subrutina Temporizar() inicializa la variable timeout y la ISR del TIMER2 posiciona (set) dicha variable para indicar que el periodo ha transcurrido.

f) Respecto al TIMER2, indique cuál es su uso, detallando su funcionamiento.

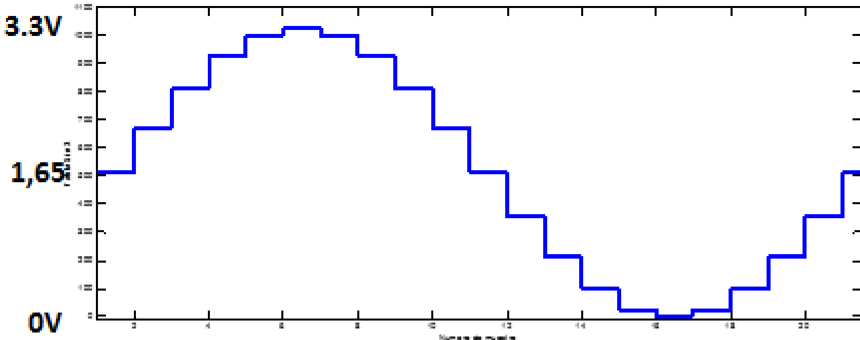
Se utiliza como temporizador. Dicho temporizador se inicializa llamando a la subrutina Temporizar, pasando como parámetro el tiempo en ms que se desea temporizar; ese tiempo inicializa el registro MR0 del TIMER2 y cuando hayan transcurrido los **ms** programados, se generará una IRQ por MATCH del TIMER2-MR0.
La ISR del TIMER2 posiciona (set) la variable timeout para indicar que el tiempo a temporizar ha transcurrido.

g) Respecto al TIMER0, indique cuál es su uso, detallando su funcionamiento en los diferentes estados del sistema.

Funciona en **modo contador de pulsos**, contando los flancos de subida de CAP0.0 que está conectado al encoder óptico de las ruedas del móvil.
Mientras el móvil avanza, el **TIMER0** cuenta los pulsos del encoder para conocer la distancia avanzada.
Cuando el móvil se para, se configura una interrupción por MATCH con la **mitad de los pulsos contados** hasta que el móvil se detuvo, es decir el valor T1TC/2. La interrupción se generará cuando el móvil haya retrocedido, por tanto, la mitad del espacio previamente avanzado.
Cuando se produzca la interrupción el móvil se detiene y el TIMER0 se resetea.

h) Si se conectara un osciloscopio al pin P0.26 del LPC1768 cuando se detecta un obstáculo: Dibuje de forma aproximada la señal que se observaría. ¿Qué líneas del programa modifican el estado de dicho pin? ¿Qué registro(s) modifican la frecuencia de salida?

20 muestras/ciclo



P0.26 es la salida del DAC. El valor de dicho pin se modifica con las líneas de programa

```
LPC_DAC->DACR = …
```

La frecuencia de salida de la señal del DAC depende de TIMER1. En cada interrupción por MATCH del TIMER1 se actualiza **LPC_DAC->DACR**, así que para cambiar la frecuencia de salida del DAC hay que modificar el periodo del MATCH de TIMER1, y esto se hace con:

```
LPC_TIM1->MR0 = …
```

i) Explique qué recurso utilizaría y como lo configuraría para reducir la carga de CPU durante la generación de la señal de alarma.

Evitaríamos la ejecución de la interrupción del Timer3 empleando el DMA en modo Linked con el propio canal (ej. Canal 0), para que al finalizar la transferencia se inicie de nuevo sin que sea necesaria la interrupción del DMA.

LPC_GPDMACH0->DMACCSrcAddr = **(uint32_t) &Tabla_SinX[0]**; // Fuente Memoria
LPC_GPDMACH0->DMACCDestAddr = **(uint32_t) &(LPC_DAC->DACR)**; //Destino Periférico
Transfer Size=**20**; Tamaño transferencia= **16 bits**; **Incrementa Fuente**; **No incremento Destino**
Sin interrupción (ojo, modo Linked)
LPC_DAC->DACCNTVAL = (F_pclk/20/1000) -1; // Frecuencia transf. DMA

j) En la rutina **SysTick_Handler(void)**:

- ¿Cuál es el periodo de interrupción?

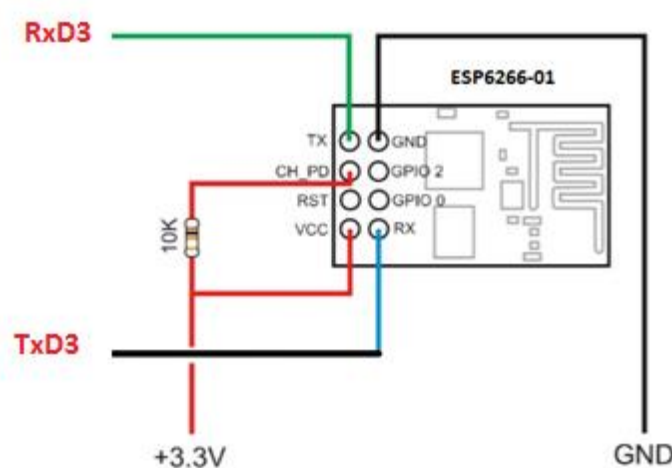
  Periodo de interrupción del SysTick = **1 ms → Tcpu*(Fcpu/1000)=1/1000**

- Calcula el periodo **máximo** de interrupción que podríamos tener.

  (1/Fcpu)* 2^{24}= 0,16777 seg.

k) Se desea controlar el robot mediante una conexión WIFI, para lo cual se conecta con la UART3 el módulo ESP8266.

- Complete el diagrama de conexión con el módulo WIFI.
- Escriba la función de configuración del puerto serie y **realice los cálculos** para configurar la velocidad a 115200 baudios.
- Calcule la velocidad real obtenida y el tiempo que tarda en recibirse un comando.



Consideramos FR=1

DL_{16}= 25e6/(16*11520)*FR = 13,56 → 14

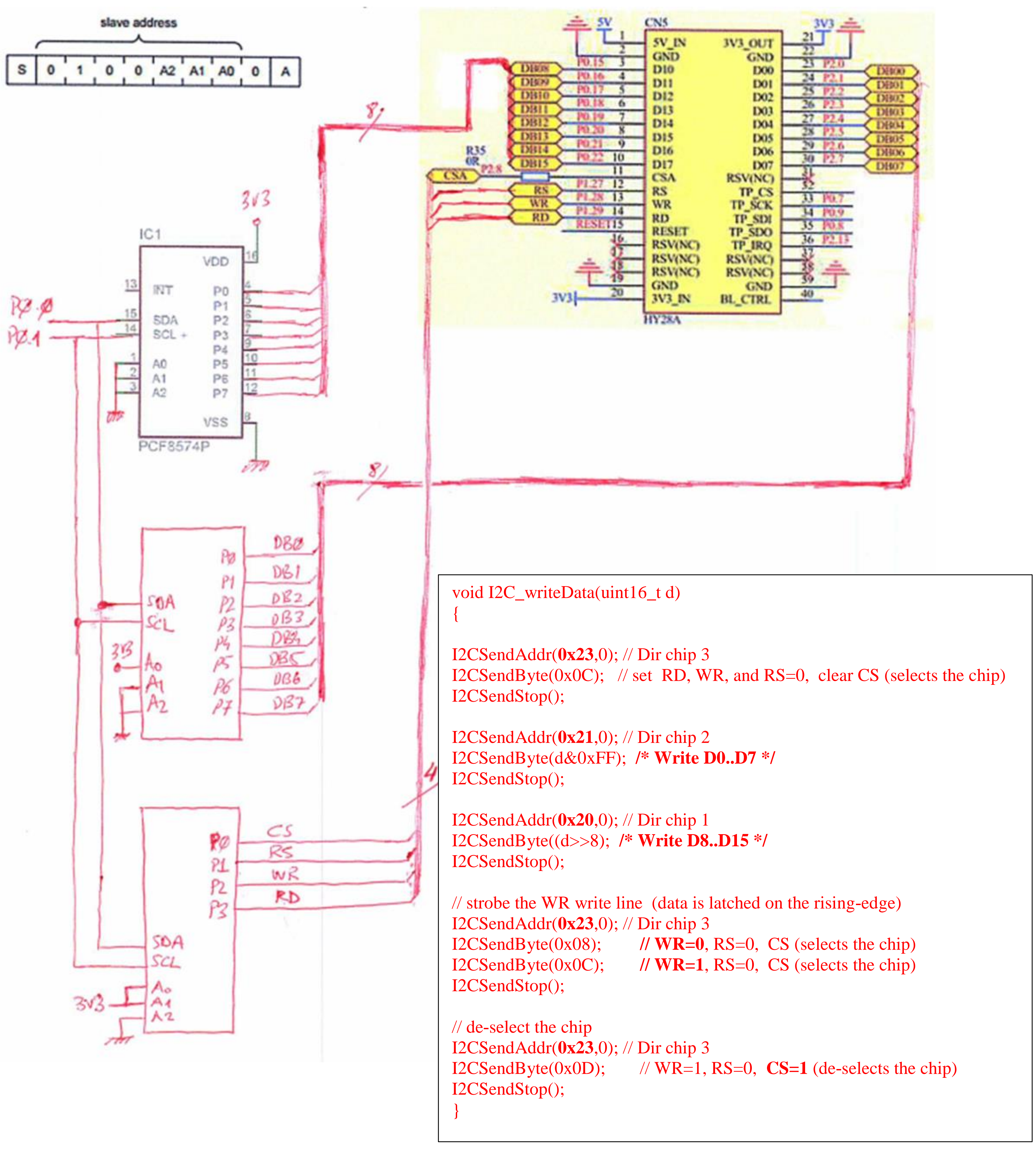
```
void uart3_init(void)
{
  LPC_PINCON->PINSEL9|=3<<26;   // RXD3 (P4.29)
  LPC_PINCON->PINSEL9|=3<<24;   // TXD3 (P4.28)
  LPC_UART3->LCR=0x83;          // N=8bits, sin paridad (DLAB=1)
  LPC_UART3->DLL=14;
  LPC_UART3->DLM=0;
 LPC_UART3->LCR=0x03;               // (DLAB=0)
}
```

Vt=25e6/(16*14) = 111607 baudios

T= (1/Vt)* Nº caracteres * 10 bits/caracter

l) Implemente la conexión del panel HY28B (Display LCD) mediante la interfaz I2C con utilizando el PCF8574 y reescriba el código de la función ***void writeData(uint16_t d)*** a partir de la información del **Anexo 3** (No tener en cuenta las dos instrucciones que configuran los pines del bus de datos como salidas)

NOTA: Sólo lo que afecta al LCD (no al Touch Panel).




```
void I2C_writeData(uint16_t d)
{

I2CSendAddr(0x23,0); // Dir chip 3
I2CSendByte(0x0C);   // set  RD, WR, and RS=0,  clear CS (selects the chip)
I2CSendStop();

I2CSendAddr(0x21,0); // Dir chip 2
I2CSendByte(d&0xFF);  /* Write D0..D7 */
I2CSendStop();

I2CSendAddr(0x20,0); // Dir chip 1
I2CSendByte((d>>8);  /* Write D8..D15 */
I2CSendStop();

// strobe the WR write line  (data is latched on the rising-edge)
I2CSendAddr(0x23,0); // Dir chip 3
I2CSendByte(0x08);       // WR=0, RS=0,  CS (selects the chip)
I2CSendByte(0x0C);       // WR=1, RS=0,  CS (selects the chip)
I2CSendStop();

// de-select the chip
I2CSendAddr(0x23,0); // Dir chip 3
I2CSendByte(0x0D);       // WR=1, RS=0,  CS=1 (de-selects the chip)
I2CSendStop();
}
```

CUESTIÓN 2

Se desea añadir a un sistema de control de acceso basado en el LPC1768 la funcionalidad de poder acceder mediante http por una conexión Ethernet cable. En Figura 2 se muestra una página web que se desea implementar y en la Figura 2 el código HTML correspondiente.

Suponiendo que se utiliza la pila de comunicaciones RL de Keil, se pide:

- Escriba el contenido de las funciones del fichero HTTP_CGI.c necesarias para que, tras pulsar el botón "Enviar" en la página web que se obtiene tras acceder al fichero "acceso.cgi", se escriba en la variable global "*char clave[40]*" el texto introducido en el campo clave y en la variable "*unsigned char tipo_usuario*" un **1** si es un usuario normal y un **2** si es un administrador.
- Escriba el contenido del fichero "acceso.cgi" y las funciones del fichero http_CGI.c necesarias para que al acceder al fichero "acceso.cgi", se envíe el código HTML adecuado teniendo en cuenta que el tipo de usuario representado debe ser coherente con el contenido de la variable "tipo_usuario".

Nota 1: Al pulsar "Enviar" se envía la información correspondiente a la información introducida en la página. Algunos ejemplos son: "tipo=administrador&clave=pepito23", "tipo=normal&clave="

Nota 2: Para ahorrar tiempo, escriba el código del fichero "acceso.cgi" completando y modificando sobre el código HTML presentado aquello que sea necesario.

Figura 2 - Captura de pantalla del Navegador

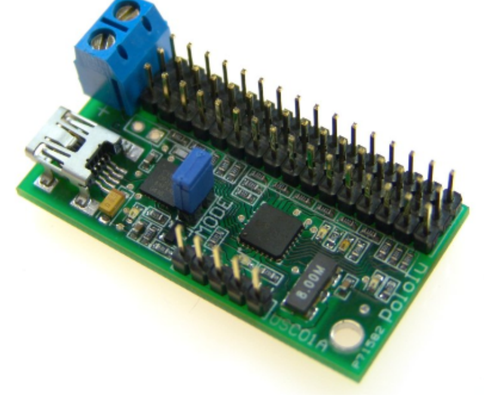
```
t   <!DOCTYPE html>
t     <html>
t       <head>
t         <meta content="text/html; charset=windows-1252" http-equiv="content-type">
t         <title>Control de acceso</title>
t       </head>
t       <body>
t         <h1 style="text-align: center;">Seleccione el tipo de acceso e introduzca clave</h1>
t         <table style="width: 100%" border="0">
t           <tbody>
t             <tr>
t               <td>
t                 <form method="GET" action="acceso.cgi">
t                   <br>Tipo de usuario:<br>
c 1                 <input %s value="normal" name="tipo" type="radio"> normal<br>
c 1                 <input %s "checked" value="administrador" name="tipo"type="radio">Administrador<br>
t                   <br>Clave: <input size="10" value="pepito23" name="clave" type="text"><br>
t                   <br><input value="Enviar" type="submit">
t                 </form>
t               </td>
t               <td><img alt="carnet" src="foto.jpg"><br></td>
t             </tr>
t           </tbody>
t         </table>
t         <p>Imagen modificada de
t            https://c2.staticflickr.com/8/7144/6845733727_6fd78d8df0_b.jpg </p>
t       </body>
t   </html>
```

Fig. 2 - Código HTML

En el diagrama se indica que siempre que se espere un dato (con el bit más significativo a cero) y se reciba el bit más significativo a uno (> 127), se pasa al estado de espera. Sería más correcto diferenciar entre recibir algo que sea mayor que 128 saltando a "Esperando 0x80" del caso en el que se reciba un 0x80 que se saltaría a "Esperando ox01". No se incluye esta diferencia por no complicar la máquina de estados pero sería la solución más correcta.

```
void cgi_process_var (U8 *qs) {
  U8 *var;

  var = (U8 *)alloc_mem (40);
  do {
    qs = http_get_var (qs, var, 40);
    if (var[0] != 0) {
      if (str_scomp (var, "clave=") == __TRUE) {
        str_copy(clave, &var[6]);
      }
      else if (str_scomp (var, "tipo=normal") == __TRUE) {
        tipo_usuario = 1;
      }
      else if (str_scomp (var, "tipo=administrador") == __TRUE) {
        tipo_usuario = 2;
      }
    }
  }while (qs);
  free_mem ((OS_FRAME *)var);
}

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
  U32 len = 0;

  switch (env[0]) {
    case '1' :
       len = sprintf((char *)buf,(const char *)&env[2],(tipo_usuario==1)?"":"checked"));
       break;
    case '2' :
       len = sprintf((char *)buf,(const char *)&env[2],(tipo_usuario==2)?"":"checked"));
       break;
  }
  return ((U16)len);
}
```

CUESTION 3



Se desea realizar con un LPC1768 un controlador de servomotores compatible con el "USB 16-Servo Controller" de *Pololu* que permite controlar 16 servomotores a través de una comunicación **serie asíncrona**. Para ello se dispone de una tarjeta con un conversor **USB–Serie** asíncrona y un LPC1768.

Suponiendo implementada la función void SetServo (uint8_t num_servo, uint32_t TH) que modifica la señal PWM correspondiente al servo "num_servo" con un nivel alto de TH microsegundos y suponiendo que el programa principal está vacío "while(1);" , se pide:

a) Realice el diagrama de estados que iría en la interrupción del puerto serie que implemente los comandos 2, 3 y 4 del protocolo de comunicación del "USB 16-Servo Controller" *de Pololu* (Ver anexo).

b) Codifique en código o pseudocódigo la función de interrupción del puerto serie que implemente la máquina de estados correspondiente. Escriba el código **sólo de dos** estados, el inicial y uno de los que activan los servos.

Se suponen definidas las variables num, com y temp como variables locales estáticas dentro de la rutina de atención a la interrupción al puerto serie que no cambian su valor al salir de la interrupción.

.

dato ≠ 2,3,4

dato == 0x80

Espera 0x80

Espera 0x01

dato == 0x01

Espera Comando

dato == 2,3,4 / com = dato

Espera servo num

dato ≠ 0x01

dato > 15

dato>128 / 1*

dato<15 & com==2 / num = dato

dato>127

Espera Dato 1 com 2

dato<15 & com==3 | com==4 / num=dato

Espera dato 1 com 3,4

dato > 127

dato > 127

dato < 128 / temp = dato << 7

dato<127 & com==3 / 2*

Espera dato 2 3,4

dato<127 & com==4 / 3*

1* $SetServo\left(num, \frac{dato \times 1000}{127} + 1000\right)$

2* $SetServo\left(num, \frac{dato \mid temp}{255} \times 1000 + 1000\right)$

3* $SetServo\left(num, dato \mid temp\right)$

```
void UART0_IRQHandler(void) {
   static uint8_t estado = E_ESPERANDO_0x80;
   static uint8_t num, com, temp;

   switch(LPC_UART0->IIR&0x0E) {

   case 0x04:
     dato = LPC_UART0->RBR;
     switch (estado) {
        case E_ESPERANDO_0x80:
           if (dato == 0x80)
              estado = E_ESPERANDO_0x01;
           break;
           .
           .
           .
        case E_ESPERANDO_DATO2_COM3_COM4:
           if ((dato < 127) && (com == 3))
              SetServo(num, (dato | temp) * 1000 / 255 + 1000);
            else
              SetServo(num, (dato | temp));

           }
           estado = E_ESPERANDO_0x80;
           break;
     }
     break;

   default:
      break;

   }
}
```

ANEXO 1. Código en lenguaje ‘C’ de la implementación de la aplicación Ejercicio 1

```
// LPC1768 con CCLK = 100 MHz
#include "LPC17xx.H"
#include <stdint.h>

enum {ESTADO_RESET, ESPERANDO_START, AVANZANDO, DETENIDO_EN_OBSTACULO, RETROCEDIENDO};

uint16_t Tabla_SinX[20] =
       {512,670,812,926,998,1023,998,926,812,670,512,353,211,97,25,0,25,97,211,353};

uint8_t fin_retroceso= 0, timeout= 0, estado= ESPERANDO_START, indice_SinX= 0;
uint16_t distancia, adcVal;

//-----------------------------------------------------------------------------
void Motor_AVANZA(){
   LPC_GPIO0->FIOCLR = 1<<1;
   LPC_PWM1->MR1 = 2000;
   LPC_PWM1->LER =  1<<1     ;        //  COMPLETAR CODIGO
   LPC_GPIO0->FIOSET = 1<<0;
}

//-----------------------------------------------------------------------------
void Motor_RETROCEDE(){
   LPC_GPIO0->FIOCLR = 1<<0  ;        //  COMPLETAR CODIGO
   LPC_PWM1->MR1 = 2000;
   LPC_PWM1->LER = 1<<1;
   LPC_GPIO0->FIOSET = 1<<1  ;        //  COMPLETAR CODIGO
}

//-----------------------------------------------------------------------------
void Temporizar(uint32_t ms){
   timeout=0;
   LPC_TIM2->TCR = 1<<1;
   LPC_TIM2->MR0 = ms * 1000;
   LPC_TIM2->TCR = 1<<0;
}

//-----------------------------------------------------------------------------
void TIMER0_IRQHandler(void) {
   LPC_TIM0->IR |= 1<<0      ;        //  COMPLETAR CODIGO
   fin_retroceso = 1;
   LPC_TIM0->MCR = 0;
}

//-----------------------------------------------------------------------------
void TIMER1_IRQHandler(void) {
   LPC_TIM1->IR |= 1<<0;
   LPC_DAC->DACR = Tabla_SinX[indice_SinX++]<<  6   ; //  COMPLETAR CODIGO
   if(indice_SinX == 20) indice_SinX = 0;
}

//-----------------------------------------------------------------------------
void TIMER2_IRQHandler(void) {
   LPC_TIM2->IR |= 1<<0;
   timeout = 1;
}
```

```
//------------------------------------------------------------------------------
void EINT0_IRQHandler() {
   LPC_SC->EXTINT = 1<<0;
   LPC_TIM0->TCR = 1<<1;
   LPC_TIM0->TCR = 1<<0;
   Temporizar(1000* 30    );              //  COMPLETAR CODIGO

   Motor_AVANZA();
   estado = AVANZANDO;
}

//------------------------------------------------------------------------------
void SysTick_Handler(void) {

adcVal = (LPC_ADC->ADGDR >>   4  ) &  0xFFF     ; //COMPLETAR CODIGO (leer ADC)
distancia = 200 - (adcVal * 330)/4095;

switch(estado){
   case AVANZANDO:
      if(timeout){
         LPC_TIM2->TCR = 1<<1;
         LPC_GPIO0->FIOCLR = 1<<1 | 1<<0;
         estado = ESPERANDO_START;
      }
      if(distancia <= 10){
         LPC_GPIO0->FIOCLR =  1<<1 | 1<<0 ; //COMPLETAR CODIGO (stop motores)
         Temporizar(5000);
         LPC_TIM1->MR0 = 1250;
         LPC_TIM1->TCR = 1<<0;
         estado = DETENIDO_EN_OBSTACULO;
      }
      break;

   case DETENIDO_EN_OBSTACULO:
      if(timeout){
         LPC_TIM1->TCR = 1<<1;
         LPC_TIM1->MR0 = 2500;
         LPC_TIM1->TCR = 1<<0   ;    //COMPLETAR CODIGO (habilitar tono 500 Hz)
         fin_retroceso=0;
         LPC_TIM0->MR0 = LPC_TIM0->TC/2;
         LPC_TIM0->TC = 0;
         LPC_TIM0->MCR = 1<<0;
         Motor_RETROCEDE();
         estado = RETROCEDIENDO;
         }
      break;

   case RETROCEDIENDO:
      if(fin_retroceso){
         LPC_TIM1->TCR = 1<<1;
         LPC_DAC->DACR = 0;
         LPC_GPIO0->FIOCLR = 1<<1 | 1<<0;
         estado = ESPERANDO_START;
      }
   break;
   case ESPERANDO_START:
   case ESTADO_RESET:
   default:
   break;
   }
}
```

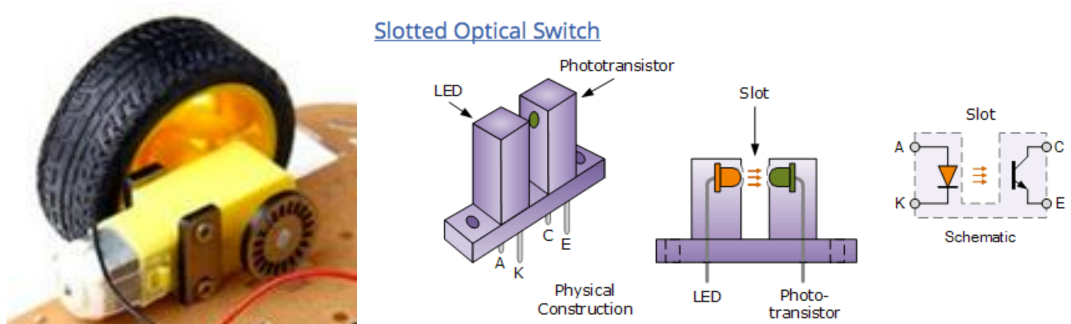
```
//-----------------------------------------------------------------------------
int main (void)
{
   SysTick_Config(SystemCoreClock/1000); // SystemCoreClock es CCLK

   LPC_SC->PCONP |= 1<<12;
   LPC_PINCON->PINSEL1|= 1<<16;
   LPC_ADC->ADCR =  1<<21 | 1<<16 | 24<<8 | 1<<1;

   LPC_PINCON->PINSEL4|=  1<<0;
   LPC_PWM1->PR = 5-1;
   LPC_PWM1->MR0 = 5000;

   LPC_GPIO0->FIOCLR = 1<<1 | 1<<0;
   LPC_GPIO0->FIODIR = 1<<1 | 1<<0;
                                     // EINT0 Rising Edge
   LPC_SC->EXTMODE  = 1<<0;          // El cableado del botón START depende
   LPC_SC->EXTPOLAR = 1<<0;          // de estas líneas de código
   LPC_PINCON->PINSEL4 |= 1<<20;
   NVIC_EnableIRQ(EINT0_IRQn);

   LPC_PINCON->PINSEL3 |= 3<<20;
   LPC_TIM0->PR= 0;
   LPC_TIM0->CTCR = 1;
   NVIC_EnableIRQ(TIMER0_IRQn);

   LPC_TIM1->TCR = 1<<1;
   LPC_TIM1->PR  = 0;
   LPC_TIM1->MCR = 1<<1 | 1<<0;
   NVIC_EnableIRQ(TIMER1_IRQn);

   LPC_SC->PCONP |= 1<<22;
   LPC_TIM2->TCR = 1<<1;
   LPC_TIM2->PR  = 25-1;
   LPC_TIM2->MCR = 1<<2 | 1<<1 | 1<<0;
   NVIC_EnableIRQ(TIMER2_IRQn);

   LPC_PINCON->PINSEL1|= 2<<20   ;     // COMPLETAR CODIGO (habilitar DAC)

   while(1);
}
```

ANEXO 2. Encoder óptico y Puente en H (L293D)

Encoder óptico

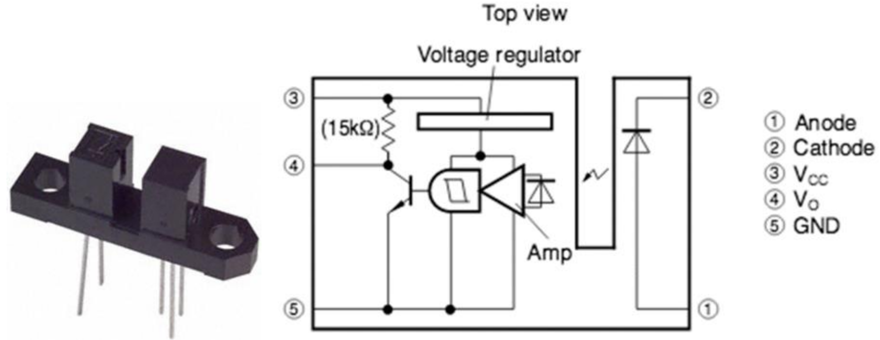
Se trata de un sistema que permitirá medir la velocidad de giro de cada motor a partir del disco de **24 ranuras** que incorpora el kit, y de un dispositivo *optical-switch* a través del cual se obtendrá una señal cuadrada cuya frecuencia dependerá de la velocidad de giro de la rueda. **Además, a partir del número de pulsos obtenidos desde un instante inicial se podrá determinar la distancia recorrida por el robot, en función del diámetro de la rueda**.

Este dispositivo está formado por un LED de infrarrojos y un fototransistor (**TCST2103** de *Vishay*, **H21A1** de *Fairchild*).



Aspecto del disco *encorder* y dispositivo *optical-switch*

Es necesario para polarizar el diodo y conformar los impulsos obtenidos del fototransistor. Un dispositivo que incorpora internamente la etapa conformadora de pulsos es el **GP1A50HRJ00F** de *Sharp*, que simplificará el montaje del *encoder*.



Aspecto y diagrama interno del opto-switch GP1A50HRJ00F de *Sharp*

Driver motores o puente en H (L293D)

Se trata de un dispositivo que incorpora un **doble puente** en H (con transistores bipolares) que conectado a cada motor permite controlar la velocidad a partir de una señal PWM, y el sentido de giro mediante una señal digital. Su diagrama interno y su conexión externa necesaria para controlar la velocidad y el sentido de giro se muestran en la figura.

Necesita dos entradas de alimentación: Vss para la electrónica digital interna (5V) y otra para el puente Vs (5-36V) que depende de la tensión de alimentación del motor utilizado. Admite una corriente máxima de 600mA por puente.

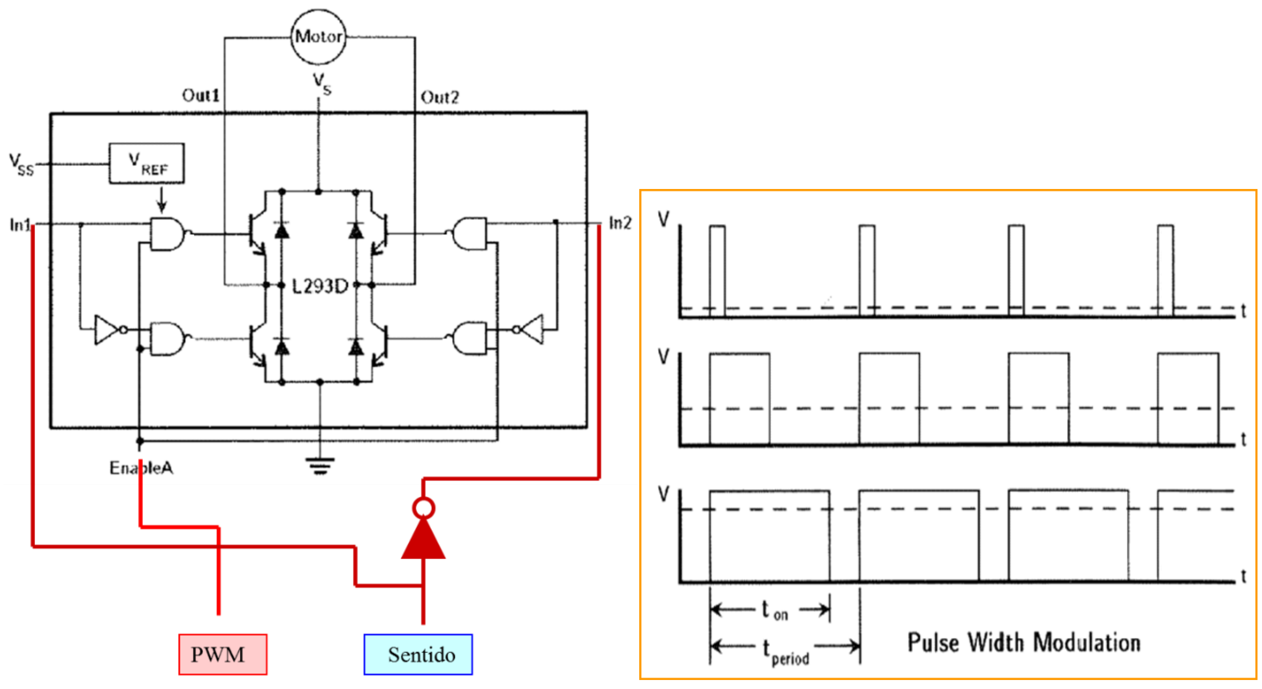
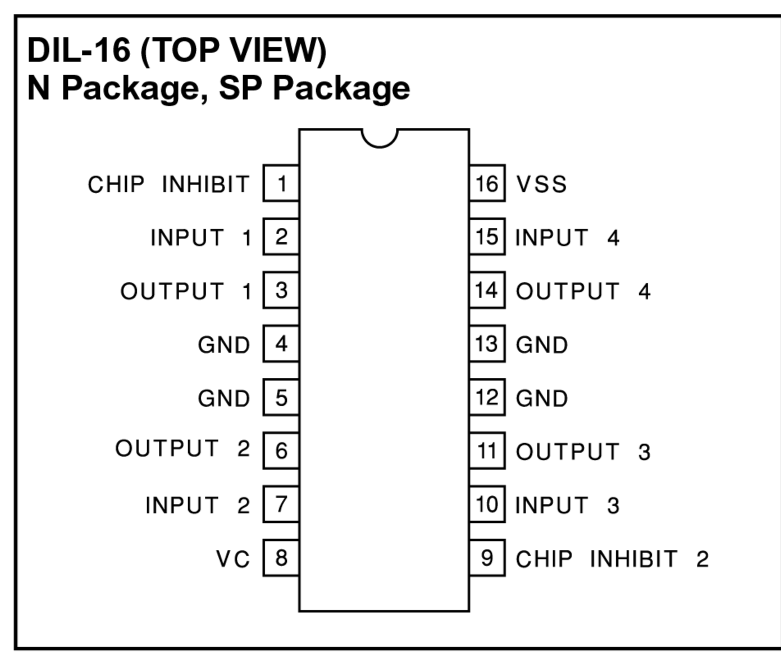


Diagrama interno y conexionado del L293D. Señal de control PWM



ANEXO 3. Librería "lcddriver.c" función *writeData(uint16_t d)*

```
//
//
//     Write Data Timing Example
//
//     RS  -------------------------------------------------- register select = 0 (data)
//
//     CS  ------------------|                |-------------- chip select
//                           |                |
//                           |________________|
//
//     WR ---------------------|          |-----------------write strobe (latches on rising edge)
//                             |          |
//                             |__________|
//                                        :
//                                        :
//     D0..7 ---------------------|-------:---|----------------- write data (all eight data bus)
//                                |       V   |
//                                |___________|
//                                        note: Data should be stable for 10nsec before WR rises
//                                              Data should be stable for 15nsec after WR rises
//                                       data
//                                      captured
//                                       here
//

void writeData(uint16_t d)
{
  uint32_t temp;
   temp = d;
   // set the RD, WR, and RS to the Idle state
   // clear CS (selects the chip)
   FIO_SetValue(LCD_PORT1, _BIT(LCD_RD));
   FIO_SetValue(LCD_PORT1, _BIT(LCD_WR));
   FIO_ClearValue(LCD_PORT2, _BIT(LCD_CS));
   FIO_ClearValue(LCD_PORT1, _BIT(LCD_RS));

   LPC_GPIO2->FIODIR |= 0x000000FF;                /* P2.0...P2.7   Output DB[0..7] */
   LPC_GPIO0->FIODIR |= 0x007F8000;               /* P0.15...P0.22 Output DB[8..15]*/


   LPC_GPIO2->FIOPIN =  temp & 0x000000ff;        /* Write D0..D7 */
   LPC_GPIO0->FIOPIN =  (temp << 7) & 0x007F8000; /* Write D8..D15 */

   // strobe the WR write line  (data is latched on the rising-edge)
   FIO_ClearValue(LCD_PORT1, _BIT(LCD_WR));
   FIO_SetValue(LCD_PORT1, _BIT(LCD_WR));

   // de-select the chip
   FIO_SetValue(LCD_PORT2, _BIT(LCD_CS));

}
```

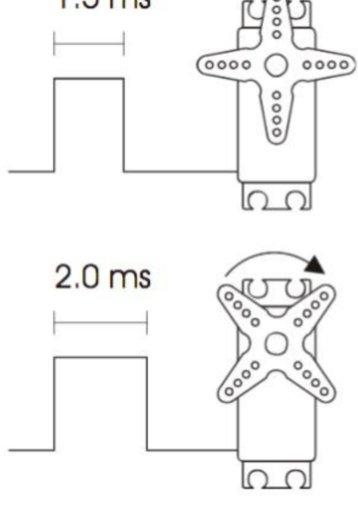
ANEXO 4. Características del servomoto USB 16-Servo Controller

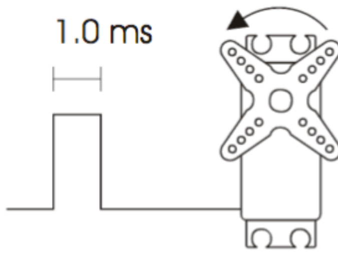
(Extract from "USB 16-Servo Controller" at https://www.pololu.com/file/0J35/usc01a_guide.pdf)

How Servos and the Servo Controller Work

Radio Control (RC) hobby servos are small actuators designed for remotely operating model vehicles such as cars, airplanes, and boats. A servo might typically move the control surface of an airplane or the steering mechanism in a car. A servo contains a small motor and gearbox to do the work, a potentiometer to measure the position of the output gear, and an electronic circuit that controls the motor to make the output gear move to the desired position. Because all of these components are packaged into a compact, low-cost unit, servos are great actuators for robots.

An RC servo has three leads: two for power and ground, and a third for a control signal input. The control signal is a continuous stream of pulses that are 1 to 2 milliseconds long, repeated approximately fifty times per second, as shown below. The width of the pulses determines the position to which the servo moves. The servo moves to its neutral, or middle, position when the signal pulse width is 1.5 ms. As the pulse gets wider, the servo turns one way; if the pulse gets shorter, the servo moves the other way. Typically, a servo will move approximately 90 degrees for a 1 ms change in pulse width, but the exact correspondence between pulse width and servo varies from one servo manufacturer to another.





The Pololu servo controller performs the processor-intensive task of simultaneously generating 16 independent servo control signals. The servo controller can generate pulses from 0.25 ms to 2.75 ms, which is greater than the range of most servos, and which allows for a servo operating range of over 180 degrees.

Protocol. To communicate with the servo controller, send sequences of five or six bytes. The first byte is a synchronization value that must *always* be 0x80 (128). Byte 2 is the Pololu device type number, which is 0x01 for the 16-servo controller. Byte 3 is one of six values for different commands to the controller; the commands are discussed below. Byte 4 is the servo to which the command should apply. Bytes 5 and possibly 6 are the data values for the given command. **In every byte except the start byte, bit seven must be clear. Thus, the range of values for bytes 2-6 is 0-0x7F (0-127).**

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2

Command 2: Set Position, 7-bit (1 data byte)
When this command is sent, the data value is multiplied by the range setting for the corresponding servo and adjusted for the neutral setting. This command can be useful in speeding up communications since only 5 total bytes are sent to set a position. Setting a servo position will automatically turn it on.

Command 3: Set Position, 8-bit (2 data bytes)
This command is just like the 7-bit version, except that two data bytes must be sent to transfer 8 bits. Bit 0 of data 1 contains the most significant bit (bit 7 of your position byte), and the lower bits of data 2 contain the lower seven bits of your position byte. (Bit 7 in data bytes sent over the serial line must always be 0.)

Command 4: Set Position, Absolute (2 data bytes)
This command allows direct control of the internal servo position variable. Neutral, range, and direction settings do not affect this command. Data 2 contains the lower 7 bits, and Data 1 contains the upper bits. The range of valid values is 500 through 5500. Setting a servo position will automatically turn it on.

Resumen en castellano

- El sistema recibe por el puerto serie información sobre la posición de que debe mantener cada uno de los servos.
- La información la recibe en una trama de datos con la siguiente estructura:

start byte = 0x80	device ID = 0x01	command	servo num	data 1	data 2

  - Dos bytes de cabecesa 0x80 0x01
  - Un byte que indica el comando a ejecutar
  - El número de servo al que aplicar el comando
  - El dato que puede ser de uno o dos bytes.
- **El bit de mayor peso de los bytes que forman la traba sólo es uno en el primer byte de la trama. En el resto siempre es cero.**
- Comando 2: seguido de solo un byte de datos que puede tomar valores de 0 a 127. 0 correspondería con 1ms de nivel alto y 127 con 2 ms.
- Comando 3: seguido de dos bytes de datos. El bit de menor peso del primer dato corresponde con el bit más significativo de la posición y los 7 bits de menor peso del segundo dato correspondería con los bits de menor peso de la posición. Con esto se obtienen unos valores entre 0 y 255 que deberá corresponder a 1ms y 2ms respectivamente.
- Comando 4: seguido de dos bytes de datos, ambos con los bits de mayor peso a cero. Este comando permite configurar un servo directamente con el número de microsegundos del nivel alto. El primer dato aporta los 7 bits de mayor peso y el segundo datos los 7 de menor peso.