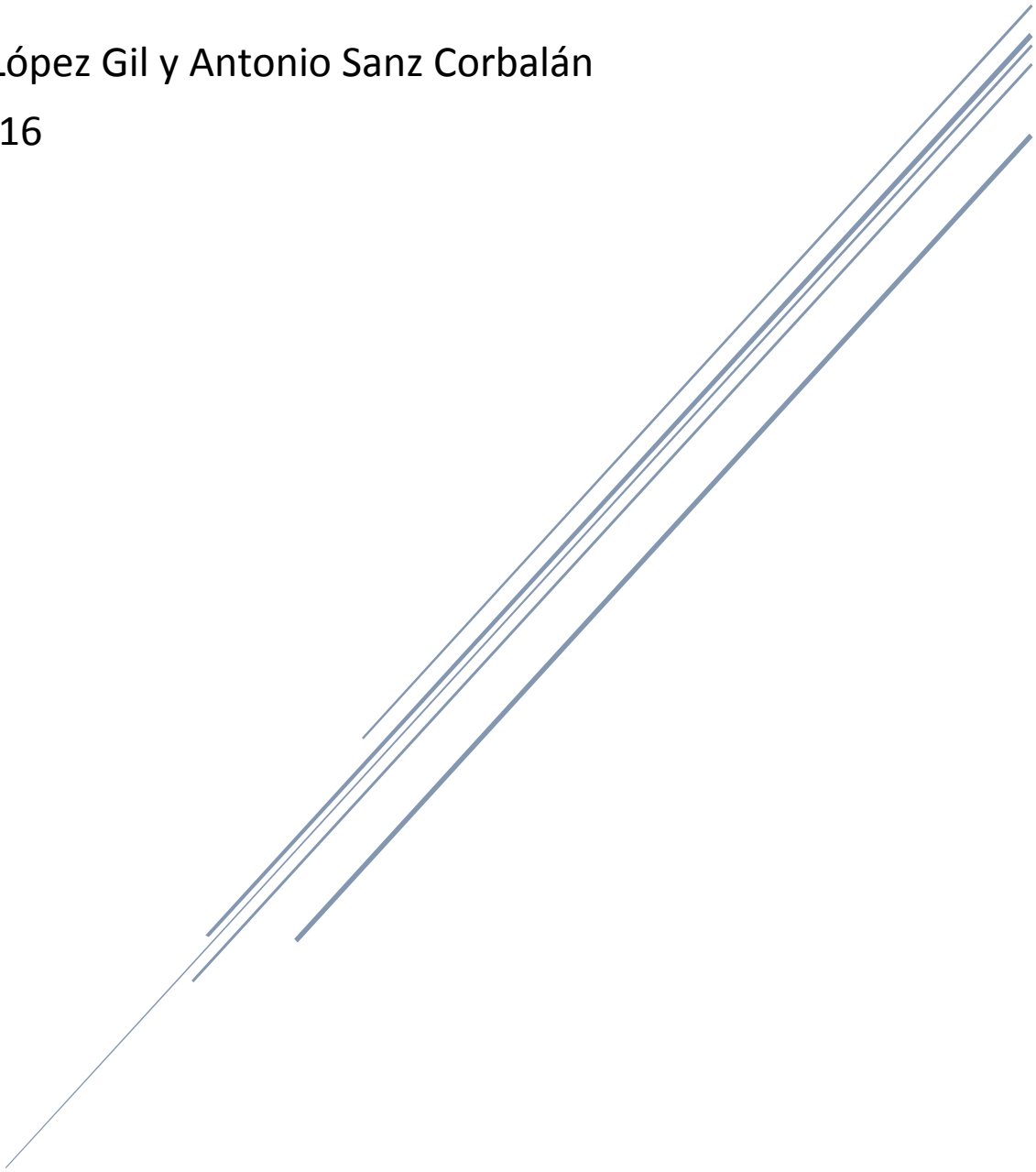
SEDA GITT

Diseño de un robot espía con tracción diferencial

Javier López Gil y Antonio Sanz Corbalán

7/6/2016

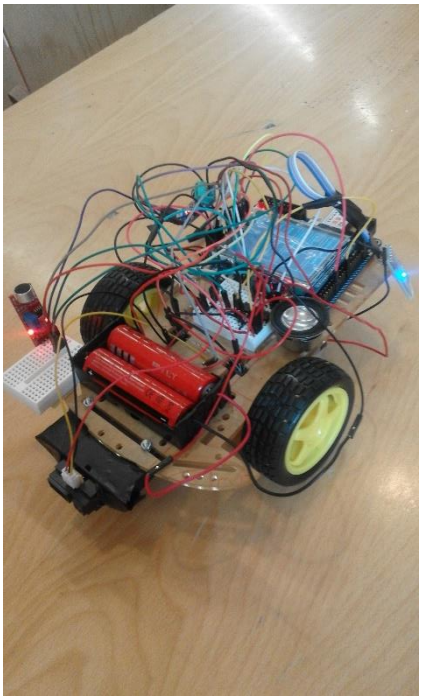


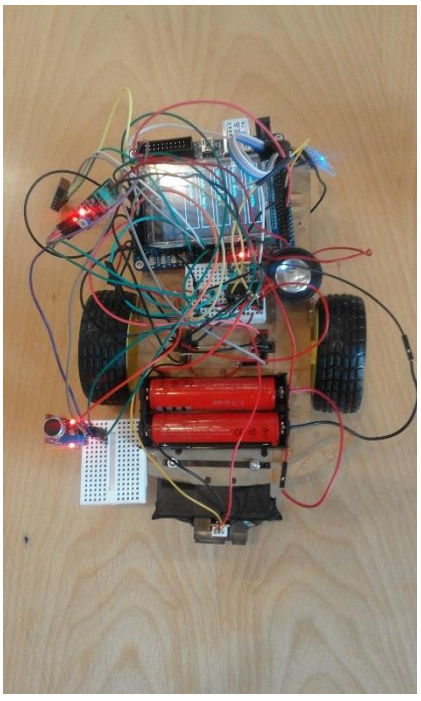
Universidad Alcalá de Henares

Índice

Introducción

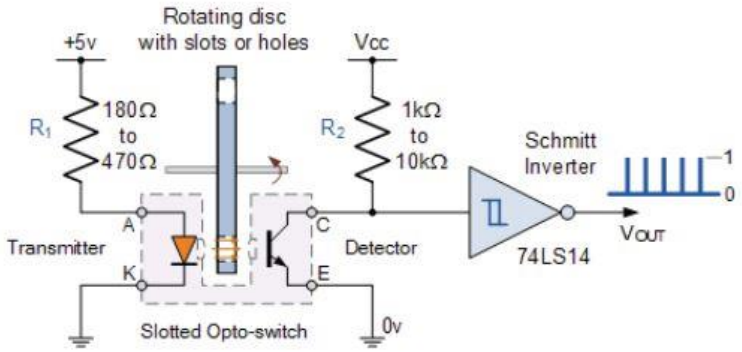
En el siguiente documentado se explica el desarrollo del proyecto final de la asignatura de Sistemas electrónicos digitales avanzados. El proyecto consiste en la realización de un robot espía capaz de realizar diversas funciones que explicaremos más adelante. Primero explicaremos el funcionamiento general del robot, a continuación haremos una descripción de Hardware aportando esquemas de bloques donde mostremos los pines que hayamos utilizado en nuestro diseño. También adjuntaremos el software con todas las funciones explicadas, comentadas y con esquemas de tipo state chart. Por ultimo haremos un análisis de ejecutabilidad de todas las tareas que realice nuestro robot. Se añadirá también un manual de usuario que muestre el uso de todas las aplicaciones con las que cuenta el robot diseñado.





Descripción del sistema diseñado

El proyecto ha sido realizado utilizando una placa LPC-1768 miniDK2 con procesador cortex M3. Además hemos utilizado un chasis de plástico con dos ruedas de 3.5 cm de radio y una rueda loca metálica. Las ruedas de 3,5 cm de radio irán unidas a motores de 6V y 240 rpm. Entre las ruedas y los motores hemos colocado un circuito detector de pulsos que contará el número de pulsos a moverse las ruedas, con la ayuda de un disco con ranuras acoplado a cada rueda. Los encoders están formado físicamente por un led, un fototransistor y una ranura entre medias donde colocaremos el disco con ranuras. De modo que cuando el led transmita un haz de luz puede ocurrir que el fototransistor lo reciba y se produzca una tensión en ese extremo o que el haz sea cortado por el disco y no se detecte nada en el otro extremo del encoder.

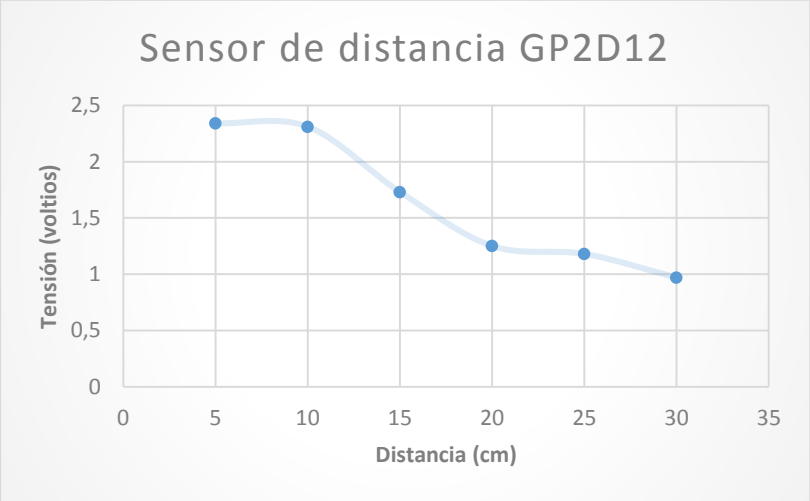


El robot cuenta con dos fuentes de alimentación. Una fuente con dos pilas de 3,7 V (tensión total de 7,4 V) será para los dos motores. La otra fuente estará formado por cuatro pilas 1,5V y alimentara la placa y alguno de los periféricos que utilicemos.
Para controlar la velocidad y el sentido de los motores disponemos de un circuito puente en H L293D. La tabla de la verdad para controlar el sentido de los motores es la siguiente:

S1	S2	S3	S4	Movimiento
0	0	0	0	Detenido
1	0	0	1	Avanza
0	1	0	0	Giro izquierda
0	0	1	0	Giro derecha
0	1	1	0	Marcha atrás

S1 y s2 hacen referencia al motor izquierdo y s3 con s4 referencian al motor derecho. La velocidad de los motores son regulados con dos salidas PWM (pulse width modulation) de la placa mini DK2. Ajustando el ciclo de trabajo de cada PWM logramos variar la velocidad de cada motor de manera independiente.

Nuestro robot también será capaz de detectar objetos por medio de un sensor de distancia. Nosotros hemos utilizado el sensor GP2D12 que tiene un rango de detección equivalente al mostrado en la tabla siguiente:



Distancia	Voltaje
5	2,34
10	2,31
15	1,73
20	1,25
25	1,18
30	0,97

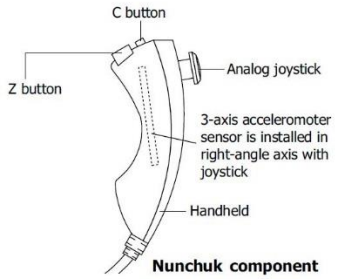
El sensor GP2D12 está formado por un led y un fotosensor. Para calcular la distancia a la que se encuentra el objeto, el led emite un haz de luz y su reflejo se recibe en el fotosensor. Dependiendo del ángulo de incidencia, que varía según la distancia a la que se encuentre el obstáculo, el fotosensor genera una tensión más alta o más baja, dentro de un rango de 0 y 2,8 voltios. El dispositivo GP2D12 ira conectado a una entrada ADC de la placa y por medio de unas ecuaciones calcularemos la distancia a la que se encuentra el objeto según la tensión recibida por el canal ADC.

$$Vout = \frac{2.1 - 0.2}{0.08}\left(\frac{1}{L + 0.42}\right) + 0.2$$

$$L = \frac{23.75}{V - 0.2} - 0.42\ [cm]$$

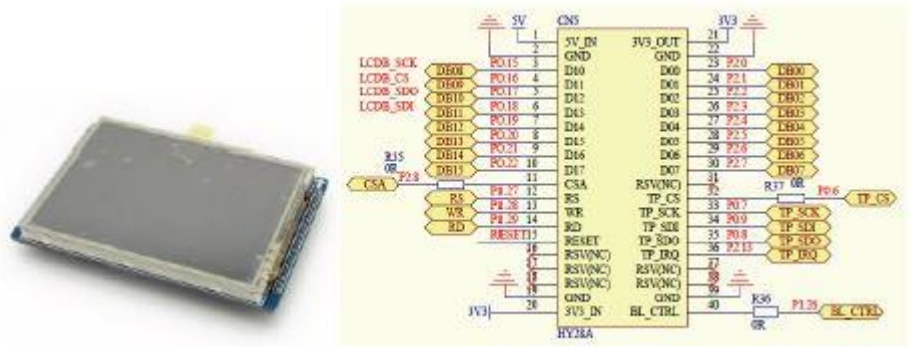
Para poder manejar el robot empleamos distintos modos de comunicación con este. Uno de ellos es mediante comunicación asíncrona por medio de tecnología bluetooth. Para ello tenemos conectado a la placa mini DK2 un módulo bluetooth (HC-06). De esta forma podremos enviar comandos desde el teléfono móvil o desde el ordenador y mantener una comunicación de tipo master-slave. Desde el maestro podremos enviar comandos del tipo A30D40I20R10F y el robot ejecutara dicha secuencia realizando una recta de 30 cm, un giro a derechas de 90 grados, una recta de 40 cm, un giro a izquierdas de 90 grados, una recta de 20 cm y un retroceso de 10 cm.

También es posible controlar el robot mediante comunicación síncrona I2C. Disponemos de un mando de la consola WII que dispone de un acelerómetro y unos registros donde se almacenan las posiciones de los ejes x, y, z. Conectando el mando WII a la placa mini-DK2 gracias a un adaptador que nos evita tener que cortar el cable del mando, podemos leer dichas posiciones de memoria e interpretarlas como movimientos del robot. Para la comunicación I2C empleamos una serie de funciones que explicaremos más adelante en el apartado del I2C.



Data byte receive								Address
Joystick X								0x00
Joystick Y								0x01
Accelerometer X (bit 9 to bit 2 for 10-bit resolution)								0x02
Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)								0x03
Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)								0x04
Accel. Z bit 1	Accel. Z bit 0	Accel. Y bit 1	Accel. Y bit 0	Accel. X bit 1	Accel. X bit 0	C-button	Z-button	0x05

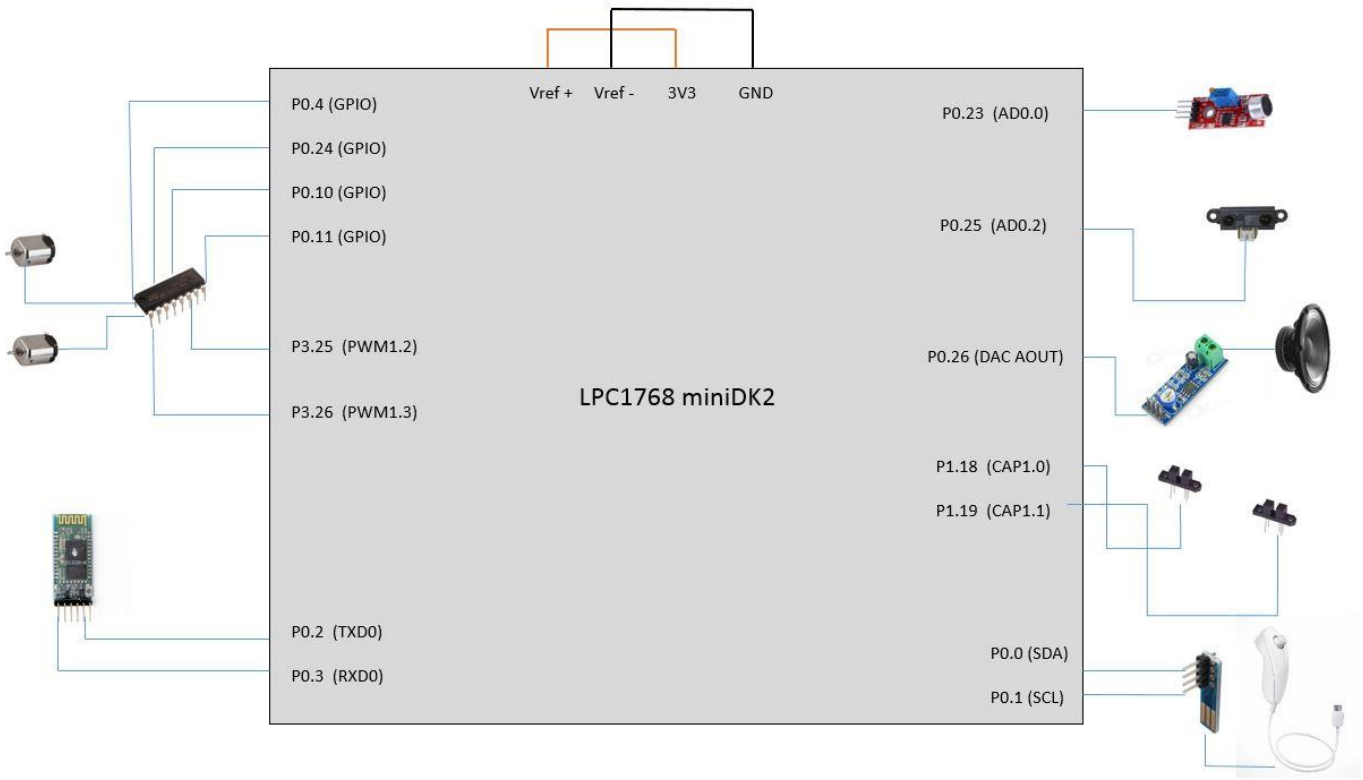
La placa mini DK2 cuenta con una pantalla TFT de 16 bits y 2,8 pulgadas. La pantalla la utilizamos para poder configurar el ciclo de trabajo de cada PWM, o para configurar el umbral de detección del sensor de distancia. A parte también se utiliza la pantalla para poder introducir secuencias de movimientos, mostrar valores como la velocidad o la distancia a la que se encuentra el objeto detectado por el sensor. La pantalla dispondrá de un menú principal y de submenús que faciliten al usuario el control.



El robot dispondrá también de un pequeño sistema de audio formado por un altavoz y un amplificador (LM386) conectado a la salida AOUT del DAC de la placa. Con este sistema de audio emitiremos una sinusoide cada vez que el sensor de distancia detecte un obstáculo dentro del umbral de medición. También reproduciremos el sonido que hayamos grabado con un micrófono conectado a una entrada ADC de la placa mini DK2.

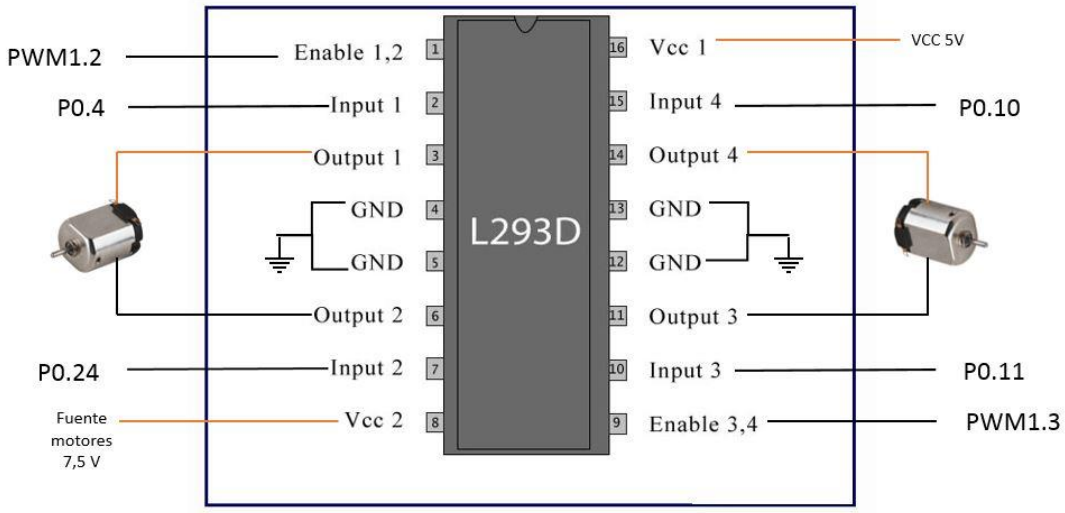
Descripción del Hardware

Esquema completo de todo el circuito

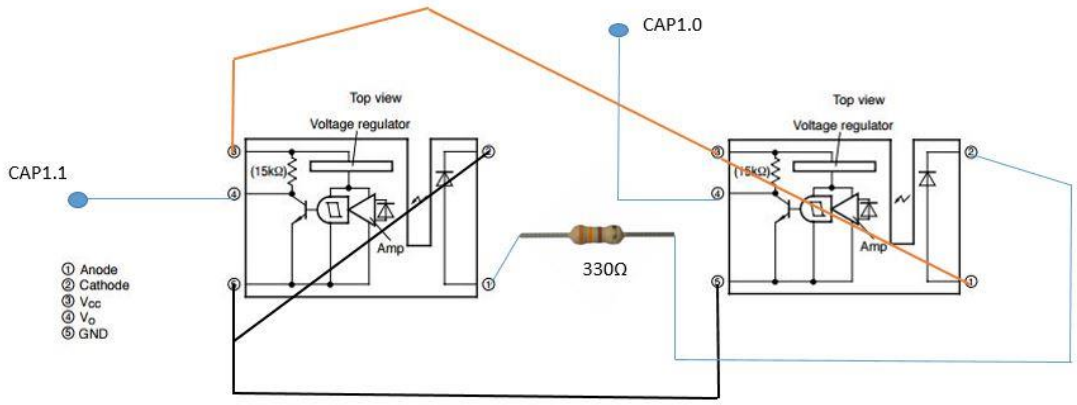


Conexiones entre dispositivos

Puente en H

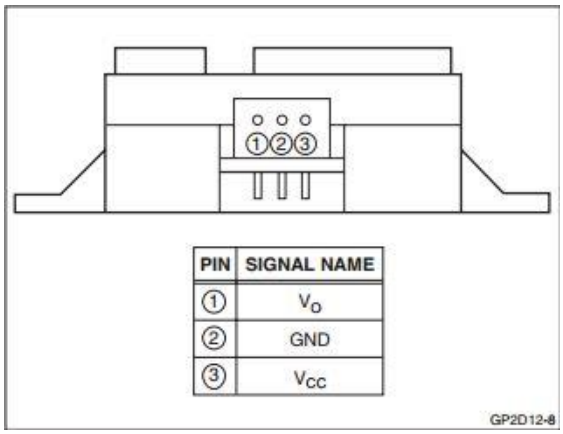


Encoders



Sensor de distancia (GP2D12):

El sensor de distancia GP2D12 tiene 3 entradas. Dos entradas son para tierra y alimentación a 3.3 Voltios. La entrada analógica Vo se conecta al pin 0.25 de la placa, configurada como canal AD0.2 del ADC.



Amplificador de audio (LM386):

El amplificador de audio tiene 4 entradas y dos salidas. Las salidas van conectadas al altavoz. Las entradas VCC y GND se conectan a 5 voltios y a tierra respectivamente. La entrada Vin se conecta al DAC de la placa.

Micrófono:

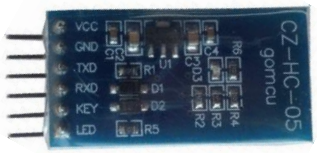
El micrófono tiene cuatro pines de entrada. El pin de entrada digital D0 no se utiliza. El pin A0 se conecta con el pin 0.23 de la placa correspondiente al canal AD0.0 del ADC. Los otros dos pines G y + se conectan a masa y a 5 voltios respectivamente.

Nunchuk Wii junto con el adaptador:

El nunchuk va conectado con el adaptador. El adaptador dispone de 4 pines de salida. Dos de ellos (- y +) son pines de alimentación masa y VCC 5 voltios. El pin C se conecta con el pin 0.1 de la placa y el pin d del adaptador se conecta con el pin 0.0 de la placa.

UART (módulo HC-05):

El modulo bluetooth HC-05 tiene 6 pines de los cuales utilizaremos únicamente 4. Dos pines son para masa y alimentación a 5 voltios. El pin RXD va conectado al pin 0.2 (TXD) de la placa y el pin TXD del módulo va conectado con el pin 0.3 (RXD) de la placa.



Descripción del software

A continuación explicaremos como se han realizado los distintos ficheros.c de nuestro proyecto. Para ello haremos una descripción de las funciones que hayamos realizado en cada fichero.

motores.c

En la función motores.c configuramos las PWM 1.2 y 1.3 con un valor MR0 de 50Khz. Para controlar el ciclo de trabajo de las PWM definimos dos funciones llamadas duty1 y duty2 donde recalculamos el valor de ciclo de trabajo dependiendo del valor actual que tenga la variable ciclo. Además también tenemos 4 funciones (sumarciclo1, sumarciclo2, restarciclo1, restarciclo2) para aumentar o disminuir el valor de la variable ciclo1 o ciclo2.

movimientos.c

En el fichero movimientos.c definimos los movimientos básicos que realizará nuestro robot. Para ello hemos definido cuatro funciones, recta, atrás, derecha e izquierda donde ponemos a cero o a uno las salido GPIO definidas para los motores (P0.4, P0.24, P10 y P0.11) siguiendo la tabla de la verdad para puente H explicado anteriormente. En esta misma función también declaramos el ADC que utilizaremos para el sensor de distancia. Hemos habilitado el canal AD0.2 por interrupción con una frecuencia de 12.5 MHz y una prioridad igual a 2. En la atención a la interrupción del ADC guardamos el valor leído en la variable Vo_sensor y comparamos si el valor de tensión medido por el sensor de distancia es superior a la variable límite definida como 1.45 voltios, en cuyo caso se llamará a la función parar que pone a cero el valor de las GPIO de los motores y a continuación llamaremos a la función Reproducirpitido con dos posibles valores frecuencia en el tono que reproduciremos, distinguiendo según la variable umbralcm. Hemos creado una función Parado en la que comprobamos si el flag llamado pararf está a cero, es decir el robot está parado o a uno cuando el robot no está parado.

En este mismo fichero también configuramos los timers 1 y 3. Utilizamos el timer 1 para medir los pulsos generados por los encoders. Hemos configurado los dos pines en modo captura, el 1.19 como CAP1.0 y el pin 1.19 como CAP1.1. Los dos pines en modo captura, el 1.18 como CAP1.0 y el pin 1.19 como CAP1.1. Configuramos el MR0 para que se produzca la conversión del ADC que nos medirá la tensión procedente del sensor de distancia, se deben producir dos matchs para que se inicie la conversión del ADC. Le asignamos una prioridad de 2, la cual es bastante prioritaria, debido a que el sensor de distancia debe ser prioritario frente al resto de movimientos del coche para poder evitar un impacto.
En la interrupción Timer1_IRQHandler, cada vez que un encoder genera un pulso, aumentamos en una unidad la variable Nflancos correspondiente y borramos el flag de interrupción.
El timer 3, lo utilizamos para evaluar los cambios producidos en el nunchuk de nuestro robot y para poder mostrar la velocidad del robot mientras esta en movimiento. Le hemos asignado una prioridad de 4, debido a que el timer 1 debe ser más prioritario para que el robot se detenga cuando el sensor de distancia detecte una distancia menor a la distancia umbral. Para

el nunchuk hemos iniciado el MR1 con un valor de 25MHz/10. Para mostrar la velocidad cada 0.5 segundos hemos iniciado el MR2 del timer 3.
En la interrupción del timer 3 (Timer3IRQ_Handler), en primer lugar borramos el flag de interrupción y evaluamos los movimientos del nunchuk, realizamos esta evaluación por rangos debido a que el valor del nunchuk cuando no tiene movimiento no tiene un valor fijo. Evaluamos también si el flag de interrupción del MR2 se ha activado, si se ha activado mostramos la velocidad en la pantalla mediante la función MostrarVelocidad, siempre y cuando el robot este en movimiento

Para calcular la distancia hemos definido una función llamada Distancia. En ella calculamos sobre dos variables denominadas distancia1 y distancia2 el valor de la distancia recorrida por cada rueda. La forma de hacerlo es calculando la circunferencia de la rueda, es decir 2π por el radio y por el número de flancos detectados dividido entre el número de ranuras que tiene la ruedecilla situada entre los encoders.
La función velocidad calcula primero el número de flancos mirando el valor del registro del CR0 del timer 1 y restándole el valor que contiene la variable anterior, el resultado lo guardamos en la variable N1. A continuación calculamos sobre las variables velo1 y velo 2 la velocidad de cada rueda. Para ello hemos dividido la distancia recorrida entre el contador CRO que nos mide el tiempo entre flancos de captura.
Por ultimo en este fichero tenemos una serie de pequeñas funciones como MostrarVelocidad que cuando la llamamos muestra en pantalla mediante un buffer nombrado buffer1, el valor de la velocidad en formato float con precisión de 3 decimales. También tenemos una función llamada MostrarSensor donde mostramos el valor de la variable Vo_sensor sobre un buffer nombrado volt. Las funciones SumarUmbral y RestarUmbral se utilizan para sumar o restar un centímetro de distancia en la variable umbralcm, que utilizamos como límite de distancia para que se detenga el coche. Por defecto tenemos definida esta variable con un valor de 18,5 cm y podremos aumentarla o disminuirla.

Audio

En el fichero audio.c definimos las funciones responsables de grabar audio y reproducir audio a través del micrófono y del amplificador respectivamente. Para el control del micrófono hemos configurado con la función init_ADC el canal AD0.0 en el pin 0.23, asociamos la conversión al match 1 del timer 0, el cual tiene una frecuencia de 4 kHz, se deben producir dos Match para que se inicie la conversión, con lo cual se muestrea a 8 Khz.

Cuando damos a grabar audio, ponemos el flag grabacion a 0, iniciamos el ADC canal AD0.0, y ponemos a 0 el TC del timer 1, para prevenir errores procedentes del TC.

El proceso de grabación de audio lo realizamos dentro de la función de atención a la interrupción del ADC. Dentro de esta función comprobamos si el resultado de la conversión del ADC está relacionada con el canal AD0.0 responsable de grabar el audio con el micrófono o con el canal AD0.2 responsable del sensor de distancia a través del registro ADGDR en los bits [24:26] . Si la interrupción ha sido producida por el canal AD0.0 del microprocesador, procedemos a introducir la muestra en el buffer audio1, cuando se han producido 16000, es decir 2 segundos de grabación, ponemos el flag grabación a 0 y el flag índice a 1, mostrando por pantalla el menú principal del robot. Las muestras de audio las guardamos en un buffer llamado audio1 con un tamaño máximo para 16000 muestras.

Al reproducir el audio grabado, actualizamos el valor del MR3 del timer 3 para que muestre a 8 kHz, pones el TC del contador a 0 y el flag de reproducción a 1. Dentro de la interrupción TIMER3_IRQHandler, comprobamos si ha sido el MR3, el que ha producido la interrupción, en caso afirmativo, miramos si el flag de reproducción a 1. Si el flag de reproducción está a 1, introducimos en registro DACR del DAC el dato j del audio desplazado 6 posiciones como se indica en el manual del LPC1768. Una vez se ha recorrido todo el buffer audio1, se pone a 0 el flag de reproducción y la variable j.

I2C.c

Para el control del robot por comunicación I2C hemos utilizado el módulo 0 del I2C. En el fichero 12c.c hemos definido varias funciones para proceder con la comunicación, en este caso entre la placa mini DK2 y el nunchuk.
Hemos definido una función I2C0Init en la que aplicamos un protocolo de inicialización para establecer la comunicación entre el nunchuk y los pines SDA y SCL de la placa. Primero llamamos a la función I2CSendAddr pasando por referencia la dirección del Nunchunk 0x52 y el bit rw igual a 0 (escritura). Dentro de la función configuramos los pines SDA Y SCL como salidas y a continuación realizamos un START mediante un FIOCLEAR primero del pin SDA y luego del SCL, por ultimo enviamos con la función I2CSendByte la dirección 0x52 desplazada 1 bit a la derecha, es decir 0xA4. Segundo llamamos a la función I2CSendByte pasándole por referencia 0xF0. Esta función realiza el envío de bytes mediante FIOSET y FIOCLEAR del pin SDA intercalando dicho envío con pulsos de reloj en el pin SCL. En tercer lugar se vuelve a llamar a la función I2CSendByte con un valor 0x55. En cuarto lugar se llama a la función I2CSendStop que deja el bus de datos en reposo subiendo el pin SCL y bajando el pin SDA. Por último se repite el mismo proceso variando los bytes enviados. Primero se realiza un I2CSendAddr, luego dos veces un I2CSendByte con los bytes 0xFB y 0x00 y para finalizar se deja el bus en reposo llamando a la función I2CSendStop. La función I2CGetByte no la hemos utilizado dentro del proceso de inicialización pero es necesaria para leer un byte del slave y enviar un ACK para cualquier byte que no sea el último o NACK si es el último byte leído. En la función se configura el pin SDA como entrada de datos, y leeremos un byte entre cada pulso de reloj enviado por SCL.

uart.c

En el fichero uart.c definimos las funciones que se van a encargar del correcto funcionamiento del bluetooth, que permitirá entre sus funciones mover, realizar la grabación de audio y la reproducción de audio mediante comandos enviados por nuestro dispositivo maestro.

Hemos definido una función init_uart0(baudios), vamos a utilizar la banda de 9600 baudios para conectarnos a nuestro dispositivo Bluetooth, iniciamos los pines 0.2 y 0,3 como pines de transmisión y recepción respectivamente. Habilitamos la interrupción UART_IRQHandler, con una prioridad de 4, al igual que la prioridad del i2c.

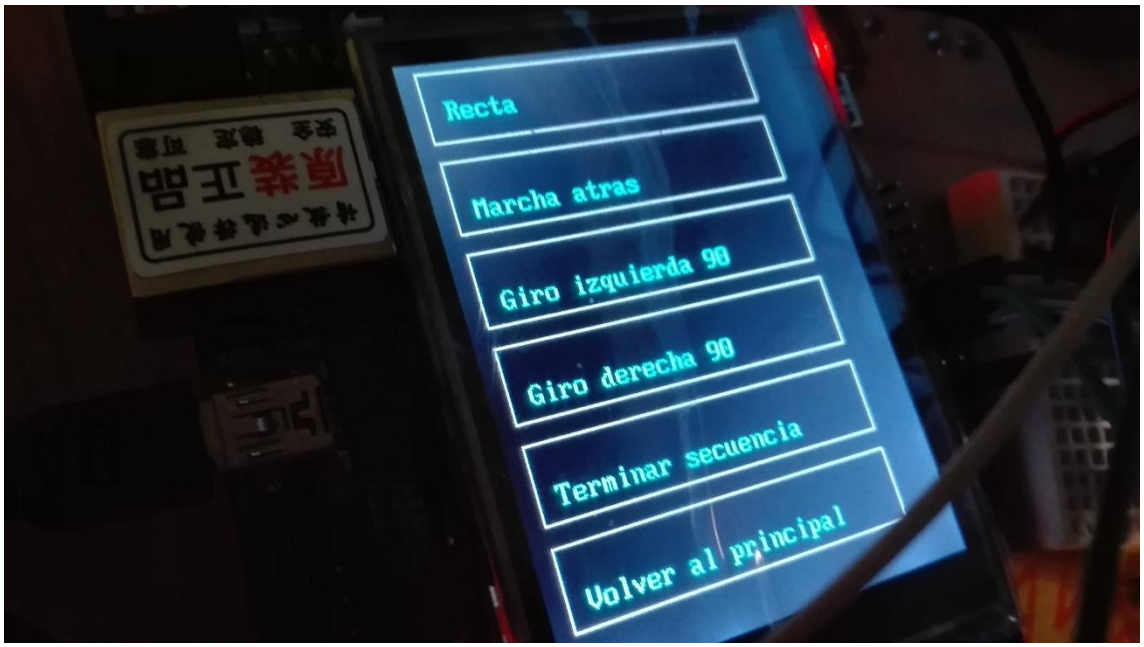
La interrupción de la uart es asíncrona, en esta interrupción vamos a guardar cada instrucción recibida en el bufferuart de 20 caracteres. Hasta que no se recibe un

carácter 0x0D, no se empiece a leer el bufferuart. Cuando se recibe el carácter return, llamamos a la función de LeerComando() definida también en este archivo.

En la función LeerComando(), hemos realizado una máquina de estados. Si se recibe una 'A' convertimos con la función atoi la siguiente instrucción recibida, lo que nos dará la distancia a recorrer, después se llama a la función Recta(distancia), hasta que no se pare no se pasara a leer la siguiente instrucción, se realiza las suma i+2, para poder pasar a la siguiente instrucción. Si se recibe una 'R' convertimos con la función atoi la siguiente instrucción recibida, lo que nos dará la distancia a recorrer, después se llama a la función Atrás(distancia), hasta que no se pare no se pasara a leer la siguiente instrucción, se realiza las suma i+2, para poder pasar a la siguiente instrucción. Si se recibe una 'D', llamamos a la función Derecha(20), para realizar el giro de 90º, los dos siguientes dígitos corresponden a la distancia que se va a avanzar hacia delante. Si se recibe una 'I', llamamos a la función Izquierda(20), para realizar el giro de 90º, los dos siguientes dígitos corresponden a la distancia que se va a avanzar hacia delante. Si se recibe una 'G', llamamos a la función GrabarAudio().Si se recibe una 'S', llamamos a la función ReproducirAudio()

HTTP_demo.c

En este fichero, main.c, se definen los menús que se van a mostrar por la pantalla, y las funciones que normalmente se van a ejecutar en el microprocesador LPC1768, en este archivo hemos definido la interrupción del timer 3(Timer3IRQ_Handler), esta interrupción la hemos explicado anteriormente, en el archivo movimientos.c. También definimos una función Menu2 (), el cual nos muestra por pantalla el siguiente menú

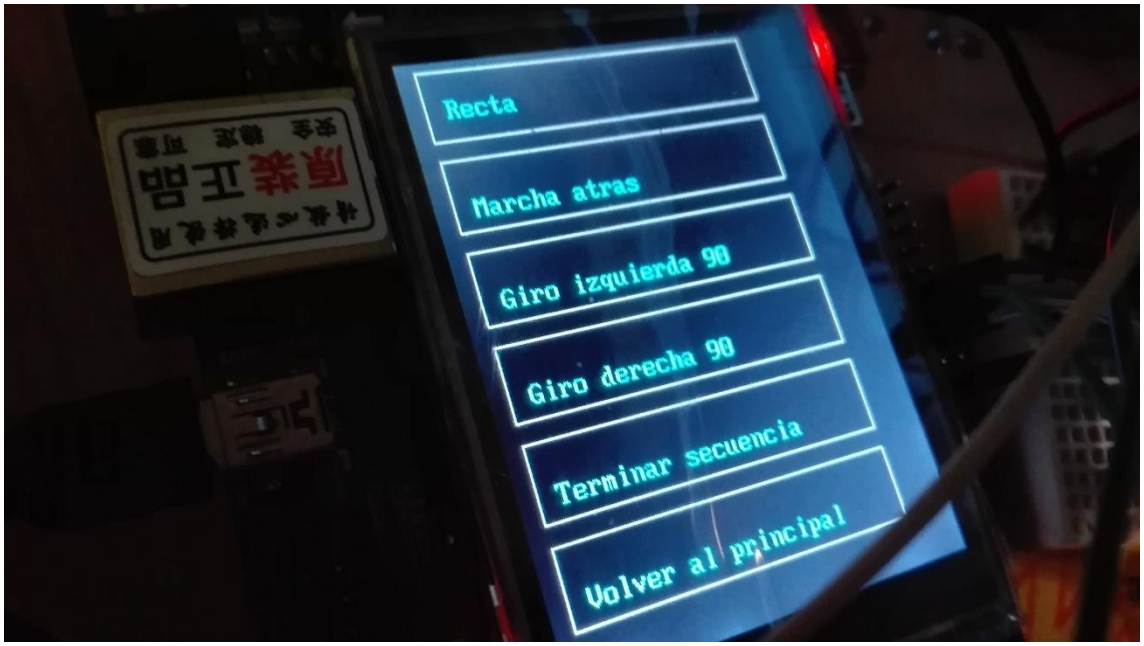


En la función int main(void), lo primero que hacemos es llamar a la distintas inicializaciones de nuestro código como son: config_motor(), config_pwm1(), init_ADC(), iniADC(), TP_init(), lcdInitDisplay(), I2C0Init(), TIMERS_config(), init_Timer0(), lo que nos permite que todos los

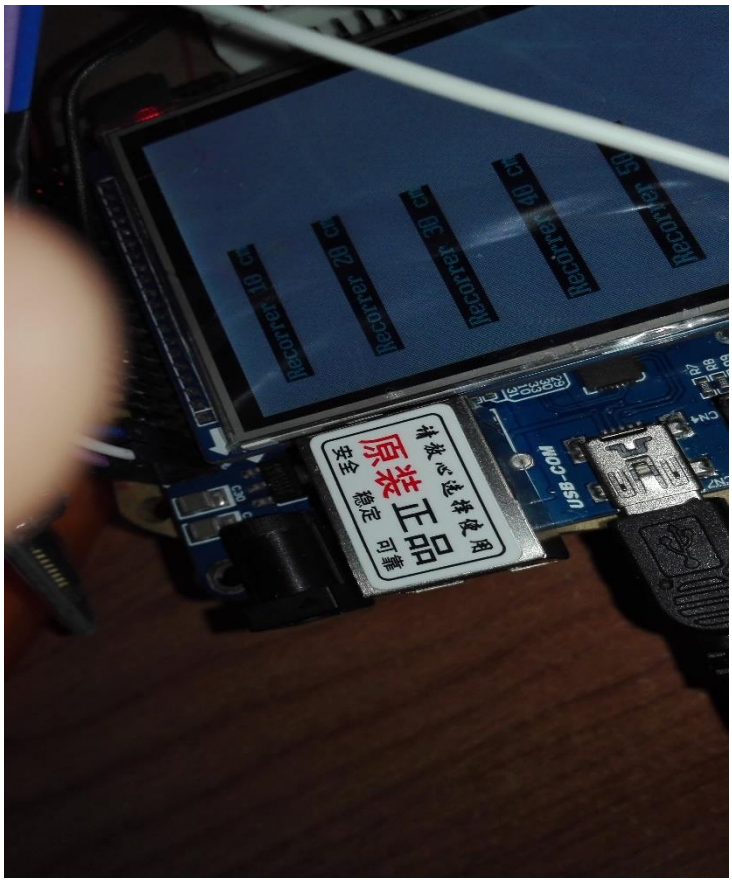
módulos tengan un funcionamiento correcto. También realizamos la calibración de la pantalla con la función TouchPanel_Calibrate().

Lo siguiente que se nos muestra es el siguiente menú:

Si pulsamos sobre el recuadro de “Control del coche”, pasaremos a llamar a la función Menu2() en la cual se nos muestra la siguiente imagen



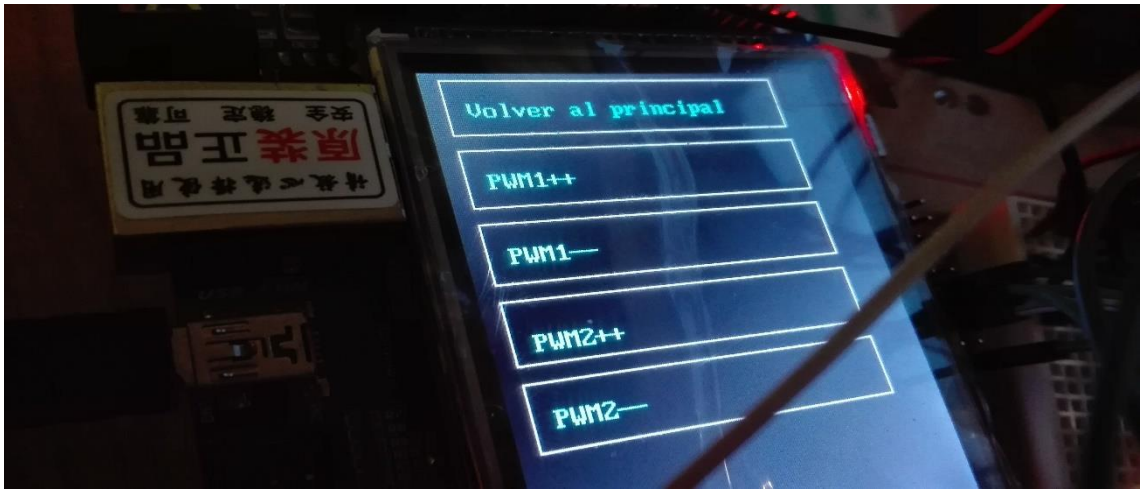
En este menú vamos a poder determinar una secuencia de movimientos que realizara nuestro robot, si damos sobre Recta o marcha atrás nos mostrara el siguiente menú para poder escoger la distancia que deseamos que realice el robot.



Si ya hemos seleccionado todos los movimientos damos a Terminar Secuencia y se llamaran a las funciones en la misma secuencia que lo hemos seleccionado.
Cuando termina de realizar todas las secuencias se realiza un fillScreen y nos mostrara en Menu 2 de nuevo para poder seleccionar de nuevo otra secuencia.
Si seleccionamos Volver al menú principal, volvemos al menú 1

En el menú 1 si seleccionamos “Grabar Audio”, nos lleva a otro estado en el cual después de dos segundos se grabara lo que digamos por el micrófono durante un cuarto de segundo, por mediación de la función GrabarAudio(), cuando se active el flag de DMA_completa volveremos a tener el menú 1 de nuevo.

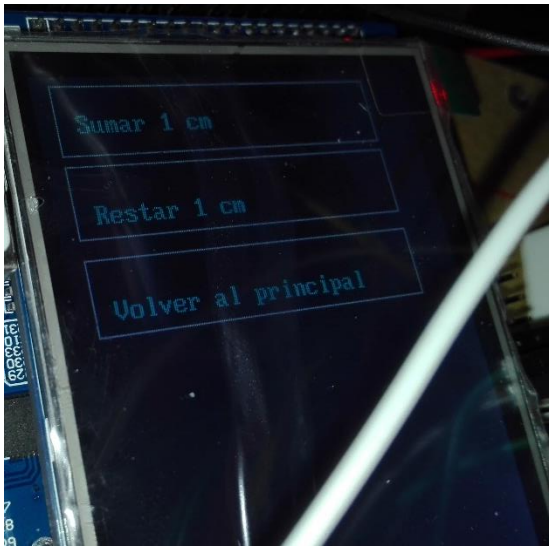
Si seleccionamos en el menú principal “Control motores”, nos aparecerá el siguiente menú



El cual nos permite seleccionar la potencia de los motores, inicialmente la potencia de ambos motores está en un 55%, si seleccionamos PWM1++, aumentaremos la potencia del motor derecho en un 10%, a través de la función duty1(ciclo1), si seleccionamos PWM1--, disminuiremos la potencia del motor derecho en un 10%, a través de la función duty1(ciclo1) . Si seleccionamos PWM2++, aumentaremos la potencia del motor izquierdo en un 10%, si seleccionamos PWM2--, a través de la función duty2(ciclo2), disminuiremos la potencia del motor izquierdo en un 10%, a través de la función duty2(ciclo2)
Al seleccionar Volver al principal, volvemos al menú anterior.

En el menú principal al seleccionar “Reproducir Audio”, llamamos a la función ReproducirAudio(), que nos reproducirá los datos grabados por el micrófono durante 5000 milisegundos, y nos devuelve al menú principal.

La última opción que tenemos en el menú principal es “Cambiar umbral”, que no lleva al siguiente menú, para poder cambiar el umbral de detección en centímetros

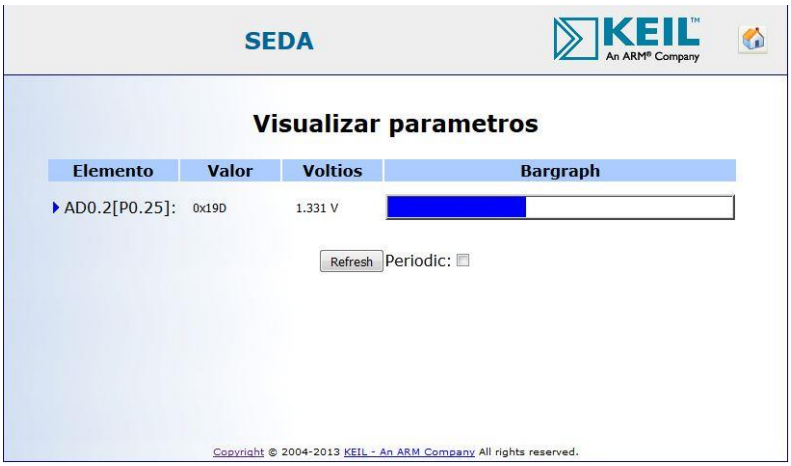


Si seleccionamos “Sumar 1 cm” llamamos a la función SumarUmbral()
Si seleccionamos “Restar 1 cm” llamamos a la función RestarUmbral()

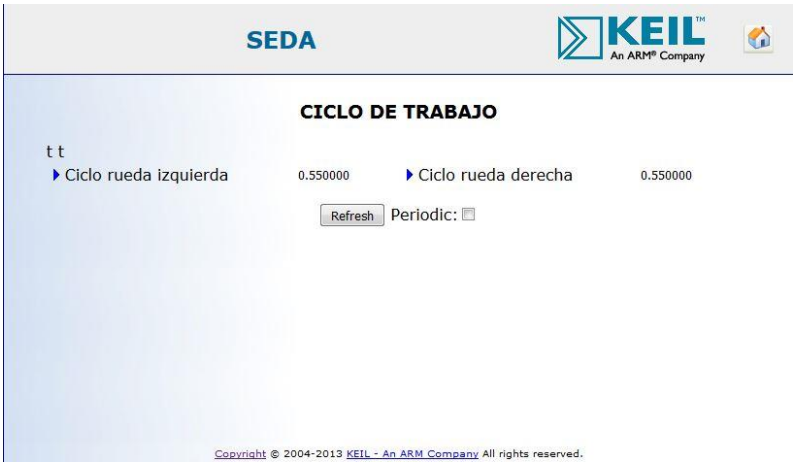
HTTP

El robot desarrollado debe ser capaz de comunicarse con el usuario a través del protocolo Ethernet. Conectando el puerto Ethernet de la placa Mini-DK2 al ordenador es posible acceder mediante el navegador a la dirección IP de nuestra placa (192.168.5.8) e introduciendo una contraseña (admin) y usuario (admin) poder introducir secuencias de ruta y visualizar variables como la velocidad, el ciclo de trabajo de cada rueda o el valor del sensor en voltios. A continuación vamos a describir los diferentes archivos realizados para poder implementar los distintos servicios.
La primera pantalla que visualizamos al entrar en la página es index.htm, este menú de inicio muestra todas las procedimientos que podemos realizar.

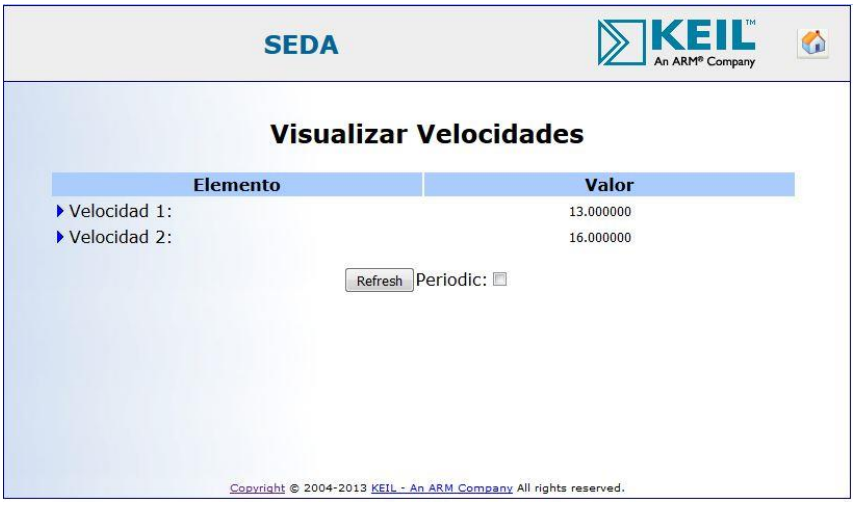
El fichero ad.cgi permite visualizar a través del navegador el valor del sensor de distancia en voltios, además muestra un gráfico que representa la proximidad a la que se encuentra el objeto del robot.



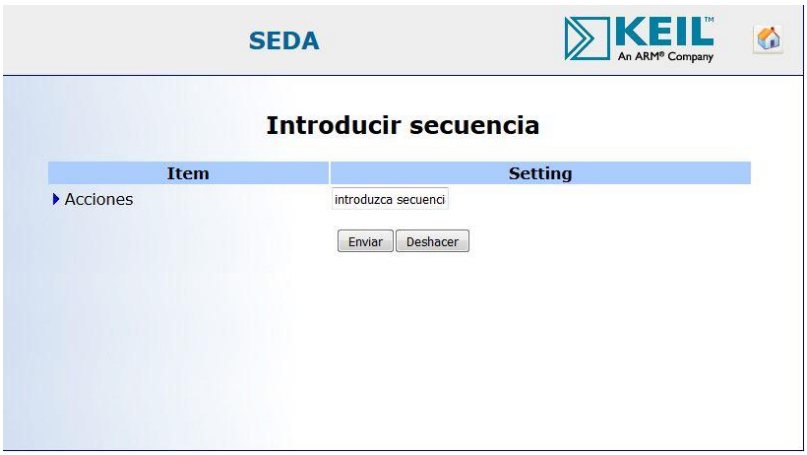
El fichero buttons.cgi muestra el ciclo de trabajo de cada PWM.



El fichero language.cgi muestra por pantalla la velocidad de cada rueda.



El fichero lcd.cgi lo utilizamos para introducir una secuencia siguiendo el mismo protocolo de órdenes que para la comunicación bluetooth. Es decir para mandar una secuencia que ordene recorrer una recta de 50 cm enviaríamos A50F.



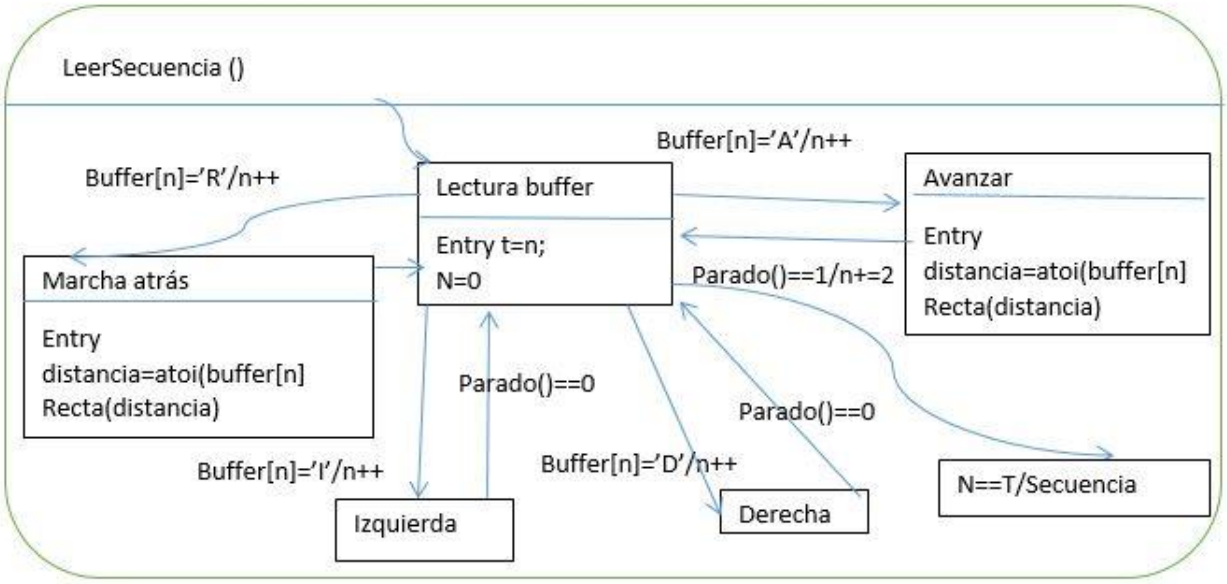
Cada fichero .cgi cuenta con su fichero .cgx homologo que se encarga de actualizar la información contenida en la página web siguiendo un formato XML.
En el fichero HTTP_CGI.c definimos que valor toma cada variable que queremos mostrar en la página web. En el caso de sensor de distancia realizamos un case con tres casos. En el caso 1 mostramos el valor en hexadecimal, en el segundo caso los mostramos los voltios en decimal y en el tercer caso hacemos un porcentaje para dibujar el gráfico.

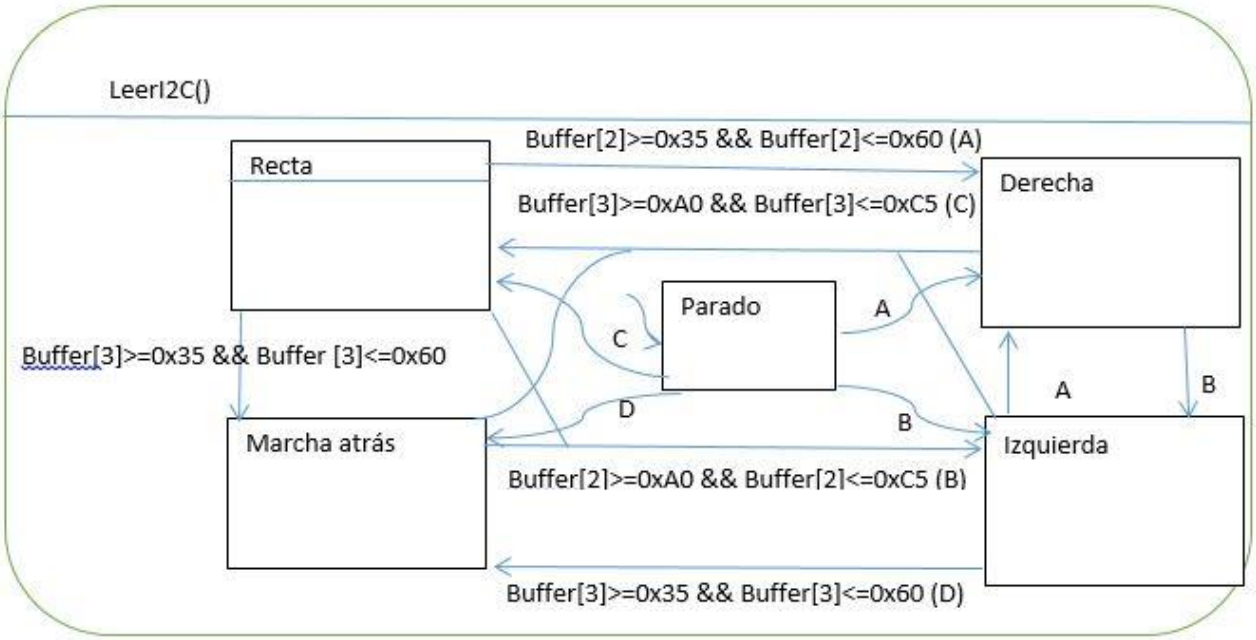
```
switch (env[2]) {
  case '1':
    adv = AD_in (2);
    len = sprintf((char *)buf,(const char *)&env[4],adv);
    break;
  case '2':
    len = sprintf((char *)buf,(const char *)&env[4],(float)adv*3.3/1024);
    break;
  case '3':
    adv = (adv * 100) / 1024;
    len = sprintf((char *)buf,(const char *)&env[4],adv);
    break;
```

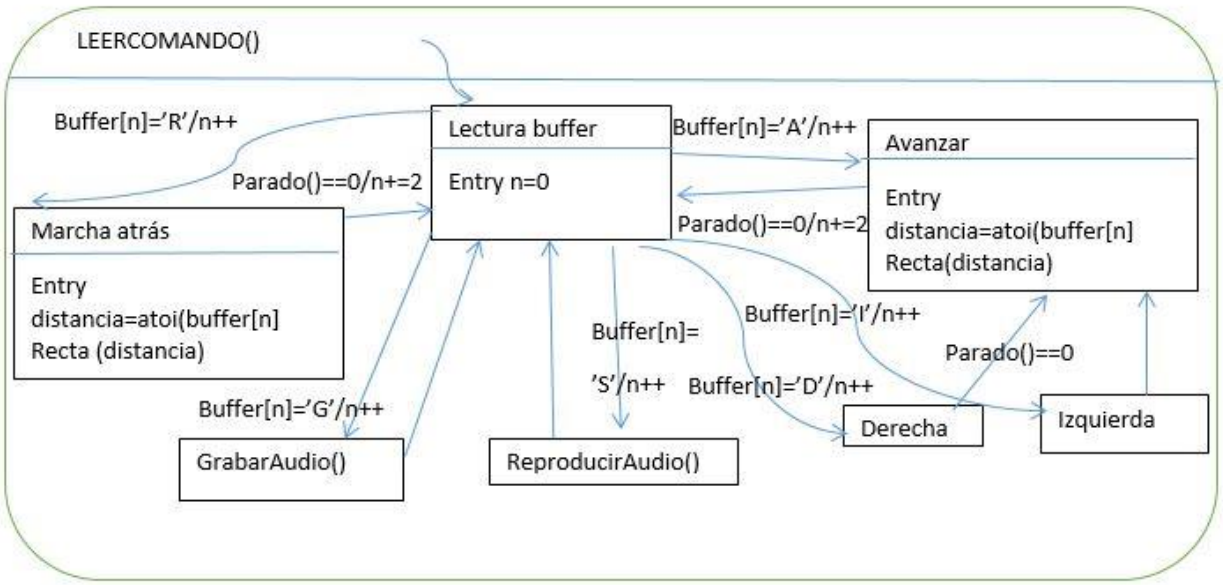
Watchdog

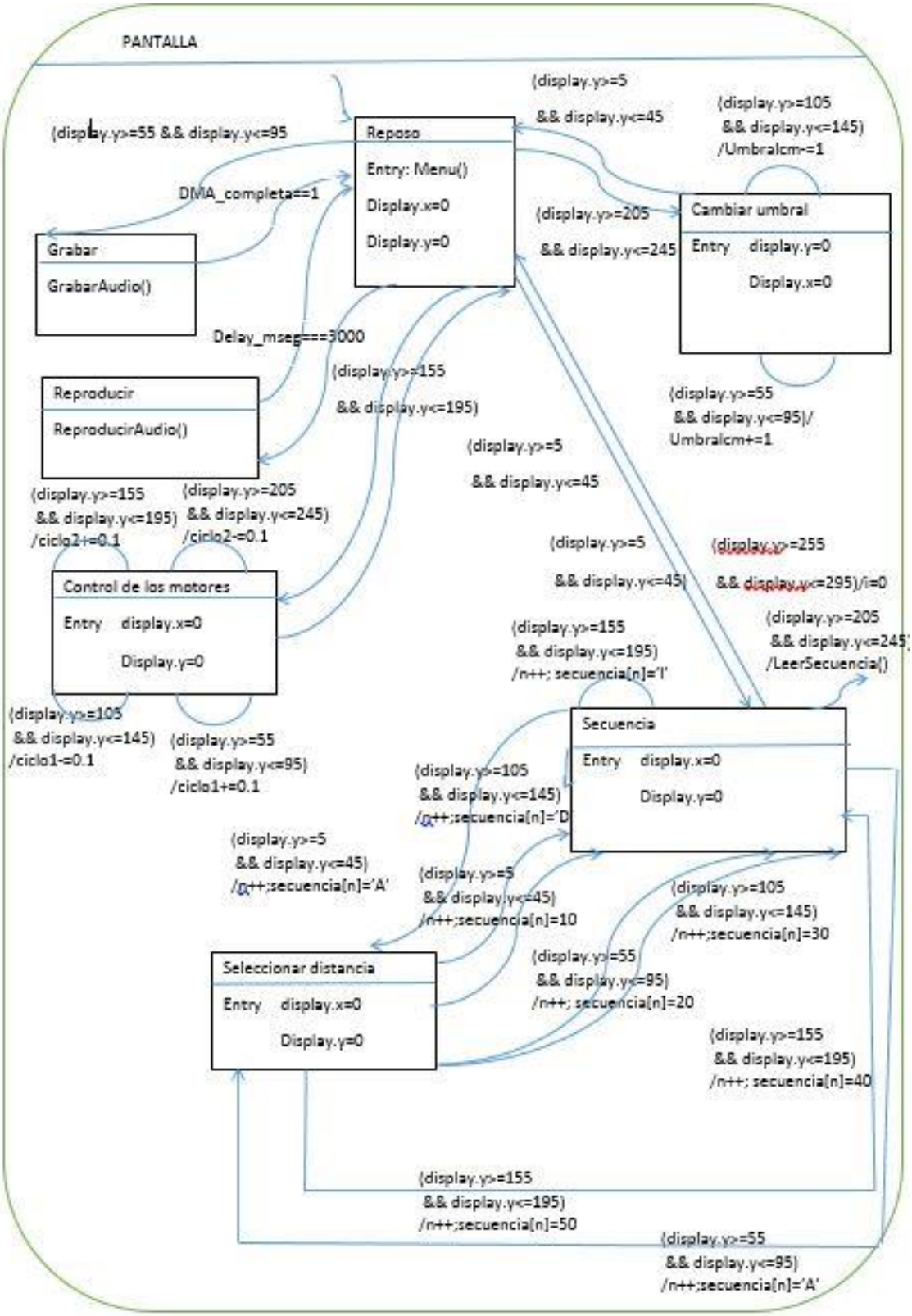
Es importante mencionar la utilidad del módulo Watchdog que permite hacer un reset automático del chip interno en caso de esperas activas que produzcan que el WDTC llegue a 0. Cargando en el registro WDFEED del Watchdog los valores 0xAA y 0x55 indicamos que realice un reset cada vez que el valor cargado en WDTC llegue a 0, en nuestro caso cada 4 segundos. Cada vez que llamamos a la función WD_FEED() restauramos el valor de WDTC a 4 segundos

State Chart del sistema






LEERCOMANDO()
Buffer[n]='R'/n++
Lectura buffer
Entry n=0
Buffer[n]='A'/n++
Avanzar
Entry
distancia=atoi(buffer[n]
Recta(distancia)
Parado()==0/n+=2
Parado()==0/n+=2
Marcha atrás
Entry
distancia=atoi(buffer[n]
Recta (distancia)
Buffer[n]=
'S'/n++
Buffer[n]='I'/n++
Parado()==0
Buffer[n]='D'/n++
Buffer[n]='G'/n++
Derecha
Izquierda
GrabarAudio()
ReproducirAudio()


PANTALLA
Reposo
Entry: Menu()
Display.x=0
Display.y=0
Grabar
GrabarAudio()
Reproducir
ReproducirAudio()
Cambiar umbral
Entry display.y=0
Display.x=0
Control de los motores
Entry display.x=0
Display.y=0
Secuencia
Entry display.x=0
Display.y=0
Seleccionar distancia
Entry display.x=0
Display.y=0
(display.y>=55 && display.y<=95
DMA_completa==1
Delay_mseg===3000
(display.y>=5
&& display.y<=45
(display.y>=105
&& display.y<=145)
/Umbralcm-=1
(display.y>=205
&& display.y<=245
(display.y>=55
&& display.y<=95)/
Umbralcm+=1
(display.y>=155
&& display.y<=195)
(display.y>=155
&& display.y<=195)
/ciclo2+=0.1
(display.y>=205
&& display.y<=245)
/ciclo2-=0.1
(display.y>=255
&& display.y<=295)/i=0
(display.y>=205
&& display.y<=245)
/LeerSecuencia()
(display.y>=155
&& display.y<=195)
/n++; secuencia[n]='I'
(display.y>=105
&& display.y<=145)
/ciclo1-=0.1
(display.y>=55
&& display.y<=95)
/ciclo1+=0.1
(display.y>=105
&& display.y<=145)
/n++;secuencia[n]='D
(display.y>=5
&& display.y<=45)
/n++;secuencia[n]='A'
(display.y>=5
&& display.y<=45)
/n++;secuencia[n]=10
(display.y>=105
&& display.y<=145)
/n++;secuencia[n]=30
(display.y>=55
&& display.y<=95)
/n++; secuencia[n]=20
(display.y>=155
&& display.y<=195)
/n++; secuencia[n]=40
(display.y>=155
&& display.y<=195)
/n++;secuencia[n]=50
(display.y>=55
&& display.y<=95)
/n++;secuencia[n]='A'

Análisis de ejecutabilidad

En nuestro proyecto tenemos cuatro tareas basadas en interrupciones:

Tarea 1: El bluetooth envía datos por el puerto serie a 9600 baudios con 8 bits por dato un bit de stop y un bit de start.

$$T = D = \frac{10}{9600} = 1{,}041\ ms$$

Tarea 2: El ADC utilizado para el sensor de distancia interrumpe periódicamente cada 0,2 segundos.

Tarea 3: El Timer 1 interrumpe cada 0,5 segundos.

Tarea 4: El timer 3 interrumpe cada 0,1 segundos.

Tarea 5: ADC utilizado para grabar el audio captado por el micrófono interrumpe solamente cuando pulsamos en la pantalla la acción de grabar. Cuando está grabando lo hace cada 125ms.

	Prioridad	**C**	**T**	**D**
Tarea 1	4	367 us	1,041	1,041
Tarea 2	2	49,6 us	0,2	0,2
Tarea 3	1	162,2 us	0,5	0,5
Tarea 4	4	9,26 ms	0,1	0,1
Tarea 5	2	12.6 us	125	125

Nuestro programa no tiene regiones críticas en el programa principal. La tarea 2 y 3 pueden interrumpir al resto de tareas. Hemos supuesto que la tarea 4 al corresponder con el manejo de robot mediante i2c con el nunchuk, no se va utilizar al mismo tiempo que lo manejemos mediante bluetooth. Es por esto que hemos supuesto que la tarea 1 y la tarea 4 no se interponen una con otra.

Tarea 2:

$$R2 = C2 + B2 + I2 = C2 + C3 \cdot \left|\frac{R2}{T3}\right| = 49{,}6 + 162{,}2 \cdot \left|\frac{R2}{500000}\right|$$

$$W0 = 211{,}8\ \mu s \qquad W1 = 211{,}8\ \mu s$$

$$R2 = 211{,}8\ \mu s < D2 = 0{,}2\ s$$

Tarea 3:

$$R3 = C3 + B3 + I3 = C3 + C2 \cdot \left|\frac{R3}{T2}\right| = 162{,}2 + 49{,}6 \cdot \left|\frac{R3}{200000}\right|$$

$$W0 = 211{,}8\ \mu s \qquad W1 = 211{,}8\ \mu s$$

$$R3 = 211{,}8\ \mu s < D3 = 0{,}5\ s$$

Tarea 1:

$$R1 = C1 + B1 + I1 = C1 + C2 \cdot \left|\frac{R1}{T2}\right| + C3 \cdot \left|\frac{R1 - C1}{T3}\right|$$

$$= 367 + 49{,}6 \cdot \left|\frac{R1}{200000}\right| + 162{,}2 \cdot \left|\frac{R1 - 367}{500000}\right|$$

$$W0 = 578{,}8\ \mu s \qquad W1 = 578{,}8\ \mu s$$

$$R1 = 578{,}8\ \mu s < D1 = 1{,}041\ ms$$

Tarea 4:

$$R4 = C4 + B4 + I4 = C4 + C2 \cdot \left|\frac{R4}{T2}\right| + C3 \cdot \left|\frac{R4 - C4}{T3}\right|$$

$$= 9260 + 49{,}6 \cdot \left|\frac{R4}{200000}\right| + 162{,}2 \cdot \left|\frac{R4 - 9260}{500000}\right|$$

$$W0 = 9471\ \mu s \qquad W1 = 9471\ \mu s$$

$$R4 = 9471\ \mu s < D4 = 0{,}1\ s$$

Tarea 5:

$$R5 = C5 + B5 + I5 = C5 + C2 \cdot \left|\frac{R5}{T2}\right| + C3 \cdot \left|\frac{R5 - C5}{T3}\right| + C4 \cdot \left|\frac{R5 - C5}{T4}\right|$$

$$= 12{,}6 + 49{,}6 \cdot \left|\frac{R5}{200000}\right| + 162{,}2 \cdot \left|\frac{R5-12{,}6}{500000}\right| + 9260 \cdot \left|\frac{R5-C5}{100000}\right|$$

$$W0 = 9484\ \mu s \qquad W1 = 9484\ \mu s$$

$$R5 = 9484\ \mu s < D5 = 125 ms$$

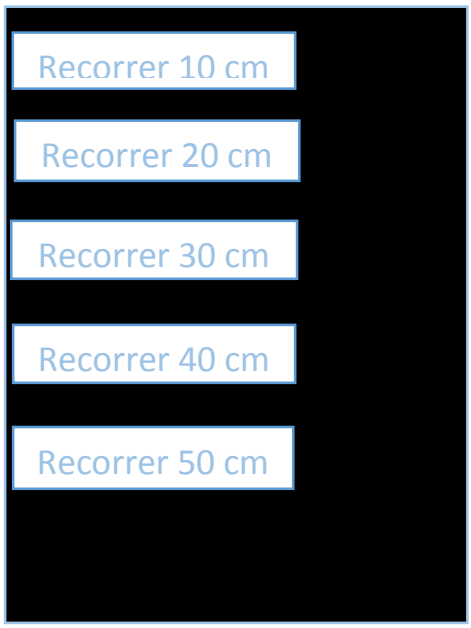
Hemos comprobado que el sistema es ejecutable.

Manual de usuario

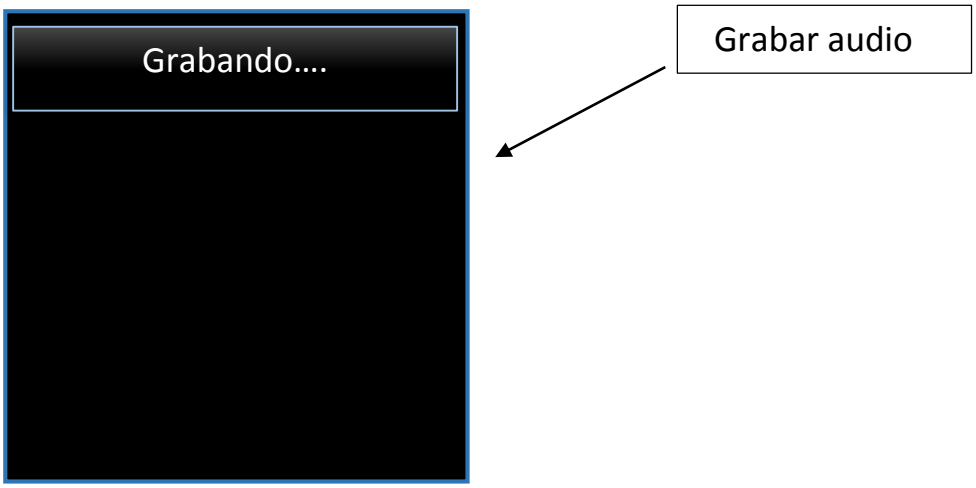
El control del robot es simple. Contamos con una pantalla táctil desde la que podemos controlar la mayoría de funciones de las que dispone nuestro robot. La pantalla despliega distintos menús y el usuario podrá pulsar sobre la opción deseada para ejecutar dicha función. El primer menú que aparece tras realizar un reset es el siguiente:

Para introducir una secuencia por pantalla debemos pulsar control coche y de esa forma accedemos al siguiente submenú.

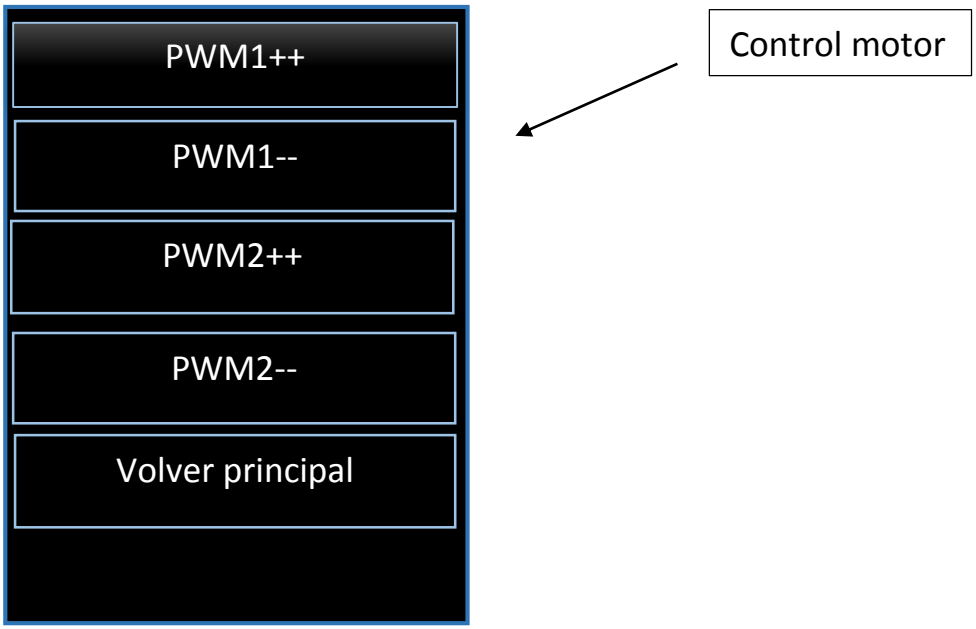
En este submenú podemos encadenar movimientos hasta pulsar en terminar secuencia y ejecutar todos los movimientos introducidos. En caso de pulsar en recta o en marcha atrás accederemos a otro menú para indicar la distancia que queramos recorrer sobre dicho movimiento.



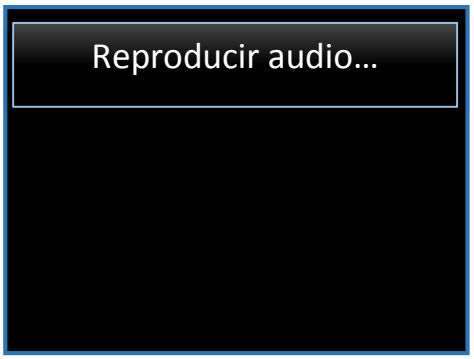
Cuando pulsamos una distancia volvemos al menú anterior para poder introducir otro movimiento o terminar secuencia y ejecutar la ruta.
Si pulsamos grabar audio en el menú principal aparece durante 5 segundos la pantalla siguiente:



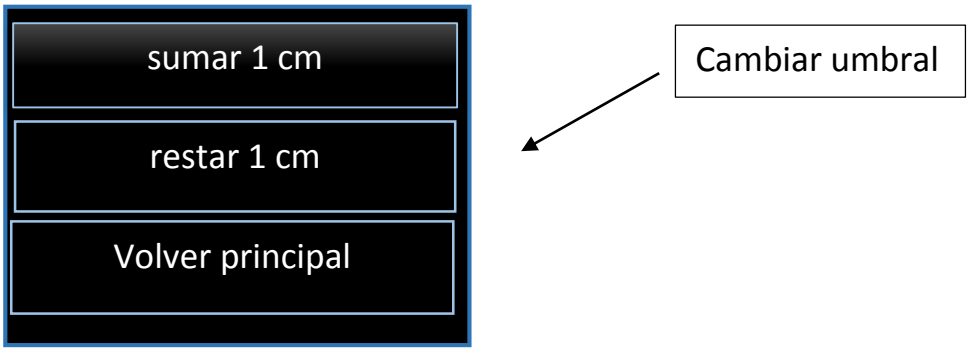
Otra de las opciones del menú es control motor, en ella podemos variar las PWM aumentando o disminuyendo un 10% su ciclo de trabajo.



La opción Reproducir audio reproduce durante 5 segundos el audio grabado en memoria y durante ese tiempo se muestra por pantalla:

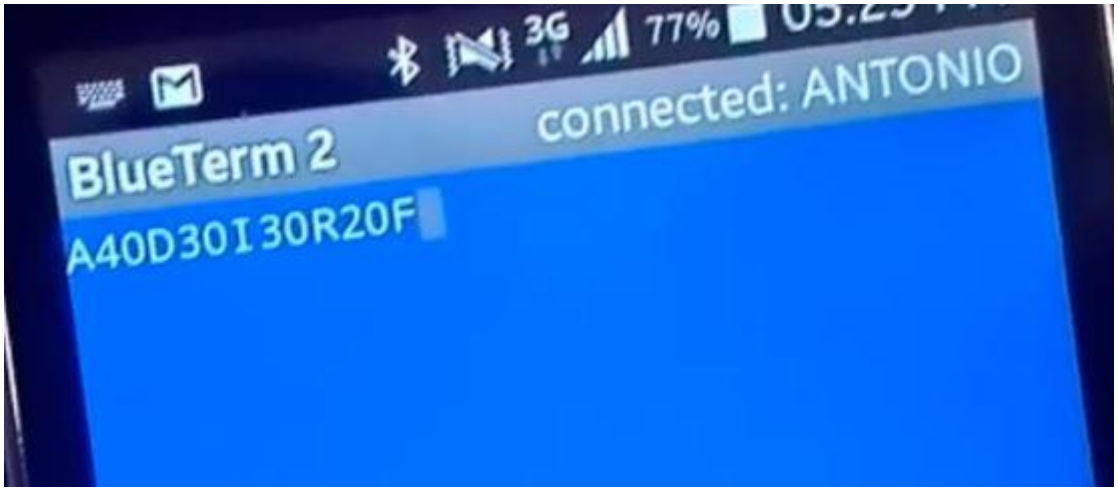


Por último tenemos la opción cambiar umbral en el menú principal que nos permite aumentar o disminuir el umbral de detección del sensor de distancia en unidades de distancia (centímetros). Cuando pulsamos sobre esta opción aparece el siguiente submenú:



Para poder controlar el robot mediante comunicación bluetooth debemos abrir una aplicación desde el móvil que nos permita enviar y recibir comandos con nuestro módulo HC-05, en nuestro caso utilizamos la aplicación blueterm2. La guía de uso desde el móvil es la siguiente:

Para avanzar introducimos la letra A
Para retroceder introducimos la letra R
Para girar 90º a la derecha introducimos la letra D
Para girar 90º a la izquierda introducimos la letra I
Entre cada carácter introducido debemos poner un distancia de dos dígitos máximo.
Al finalizar la secuencia introducida debemos poner la letra F indicando fin de secuencia.



En la captura anterior a modo de ejemplo estamos introduciendo una recta de 40 cm de distancia, un giro de 90º a la derecha, una recta de 30 cm, un giro de 90º a la izquierda, una recta de 30 cm y un retroceso de 20 cm de distancia.

También tenemos la posibilidad de controlar el robot mediante un mando nunchuk por comunicación I2c. El mando nunchuk dispone de un acelerómetro que marca unos valores para el eje x, y, z. Dependiendo de la posición del mando, los valores registrados están entre unos rangos u otros. La siguiente tabla muestra los rangos de valores para cada posición del mando nunchuk.

Movimiento	**Bufferi2c[3]**	**Bufferi2c[2]**
Parado	0x70 / 0x95	0x70 /0x95
Recta	0xA0 / 0xC5	-
Atrás	0x30 / 0x65	-
Derecha	-	0x35 / 0x60
Izquierda	-	0xA0 / 0xC5

Simplemente para manejarlo debemos inclinar la parte delantera del mando hacia delante en caso de querer trazar una recta o inclinarlo con la parte delantera hacia arriba para mover el robot hacia atrás. Para los giros a derechas o a izquierdas, giramos el mando hacia un lado o a otro dependiendo de cuál queramos realizar.

Código fuente

```
****************************** HTTP_demo.c *********************************

#include <stdio.h>
#include <RTL.h>
#include <Net_Config.h>
#include <LPC17xx.h>                /* LPC17xx definitions            */
#include "GLCD_B.h"
#include <LPC17xx.H>
#include "lcddriver.h"
#include "TouchPanel.h"
#include "movimentos.h"
#include "motor.h"
#include "uart.h"
#include "audio.h"
#include "i2c.h"


#include <stdlib.h>
#include <string.h>

#include <stdio.h>


BOOL LEDrun;
BOOL LCDupdate;
BOOL tick;
U32  dhcp_tout;
U8   lcd_text[2][16+1] = {" ",              /* Buffer for LCD text        */
"Waiting for DHCP"};

extern LOCALM localm[];                     /* Local Machine Settings     */
#define MY_IP localm[NETIF_ETH].IpAdr
#define DHCP_TOUT   50                      /* DHCP timeout 5 seconds     */

static void init_io (void);
#define F_cpu 100e6                                             // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4                          // Defecto despues del reset

#define Fvelocidad          2 //Frecuencia para mostrar la velicidad

#define F_muestreo 100 // Fs=100Hz (Cada 10ms se toma una muestra del canal 0)

#define F_out 1000                              // Frecuencia de salida del DAC
#define N_muestras 32
#define V_refp 3.3
```

```
#define PCRTC 9                                                  // Power RTC
#define CLKEN 0                                                  // Hab. time counters
#define RTCIF 0                                                  // Flag interrupción
#define IMSEC 0                                                  // Interrup. incremento segundos
#define CCALEN 4                                                 // Calibr. enable
//#include "GLCD.h"

typedef unsigned char u8;



#define Fpwm 500          // Frecuencia de la señal PWM (500Hz)
#define MR0_fin           Fpclk/Fpwm-1 // Valor de MR0 5BBD7
#define pi 3.1415
#define radio 3.5

int state=2, estado2=1, flagi2c=1;
int j=0;
char buffer2[9], buffer3[9];


void WDT_Feed(void)
{
LPC_WDT->WDFEED=0xAA;
LPC_WDT->WDFEED=0x55;
}
void init_WDT(void)
{
LPC_WDT->WDTC=2*F_pclk;//Timeout=4segundosç
LPC_WDT->WDCLKSEL=0x01;
LPC_WDT->WDMOD=0x03;
LPC_WDT->WDFEED=0xAA;
LPC_WDT->WDFEED=0x55;

}


void Menu2(){
drawRect(10, 5, 200, 40, WHITE);
drawString(20, 20, "Recta", CYAN, BLACK, MEDIUM);

drawRect(10, 55, 200, 40, WHITE);
drawString(20, 75, "Marcha atras", CYAN, BLACK, MEDIUM);

drawRect(10, 105, 200, 40, WHITE);
drawString(20, 125, "Giro izquierda 90", CYAN, BLACK, MEDIUM);

drawRect(10, 155, 200, 40, WHITE);
drawString(20, 175, "Giro derecha 90", CYAN, BLACK, MEDIUM);

drawRect(10, 205, 200, 40, WHITE);
drawString(20, 225, "Terminar secuencia", CYAN, BLACK, MEDIUM);
```

```
drawRect(10, 255, 200, 40, WHITE);
drawString(20, 275, "Volver al principal", CYAN, BLACK, MEDIUM);


}
void TIMER3_IRQHandler(void) {
int i=0;
static unsigned char bufferi2c[]={0,0,0,0,0,0,0,0};

if((LPC_TIM3->IR & (1<<1))){

I2CSendAddr(0x52, 0);
I2CSendByte(0x00);
I2CSendStop();
I2CSendAddr(0x52, 1);
for( i=0; i<5; i++)
bufferi2c[i]=I2CGetByte(0);
I2CGetByte(1);
I2CSendStop();

if(flagadc==1)
{
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
LPC_GPIO0->FIOCLR |= (1<<4); //ponemos a 0 salida P0.4
LPC_GPIO0->FIOCLR |= (1<<24); //ponemos a 0 salida P0.24
}
else{
if(((bufferi2c[3]>=0x70) &&(bufferi2c[3]<=0x95))&& ( bufferi2c[2]>=0x70) &&
(bufferi2c[2]<=0x95)){//Parado
NVIC_DisableIRQ(ADC_IRQn);
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
LPC_GPIO0->FIOCLR |= (1<<4); //ponemos a 0 salida P0.4
LPC_GPIO0->FIOCLR |= (1<<24); //ponemos a 0 salida P0.24

}
else if((bufferi2c[3]>=0xA0) && (bufferi2c[3]<=0xC5)){//Adelante
NVIC_EnableIRQ(ADC_IRQn);
LPC_GPIO0->FIOCLR |=(1<<24); //ponemos a 0 salida P0.4
LPC_GPIO0->FIOSET |= (1<<4); //ponemos a 1 salida P0.24*/
LPC_GPIO0->FIOSET |= (1<<11); //ponemos a 1 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.11
if(bufferi2c[3]>=0xBB){
ciclo1=0.7;
ciclo2=0.7;
}
else {
ciclo1=0.4;
ciclo2=0.4;
}

}
```

```
else if((bufferi2c[3]>=0x30) && (bufferi2c[3]<=0x65)){//Atras
NVIC_DisableIRQ(ADC_IRQn);
LPC_GPIO0->FIOCLR |=(1<<4); //ponemos a 0 salida P0.24
LPC_GPIO0->FIOSET |= (1<<24); //ponemos a 1 salida P0.4
LPC_GPIO0->FIOSET |= (1<<10); //ponemos a 1 salida P0.11
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.10
if(bufferi2c[3]>=0x50){
ciclo1=0.7;
duty1(ciclo1);
duty2(ciclo1);
}
else {
ciclo1=0.4;
ciclo2=0.4;
duty1(ciclo1);
duty2(ciclo2);
}
}
else if((bufferi2c[2]>=0x35) && (bufferi2c[2]<=0x60)){//Derecha

LPC_GPIO0->FIOSET |=(1<<11); //ponemos a 1 salida P0.10
LPC_GPIO0->FIOCLR|= (1<<10); //ponemos a 0 salida P0.11*/
LPC_GPIO0->FIOCLR |= (1<<4); //ponemos a 0 salida P0.0
LPC_GPIO0->FIOCLR |= (1<<24); //ponemos a 0 salida P0.1
}
else if((bufferi2c[2]>=0xA0) && (bufferi2c[2]<=0xC5)){//Izquierda

LPC_GPIO0->FIOSET |=(1<<4); //ponemos a 0 salida P0.0
LPC_GPIO0->FIOCLR|= (1<<24); //ponemos a 1 salida P0.1*/
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
}}
LPC_TIM3->IR |= (1<<1);

}
else flagi2c=1;

if((LPC_TIM3->IR & (1<<2))){
if(Parado()==1){
MostrarVelocidad();
MostrarSensor();
}
LPC_TIM3->IR |=(1<<3);
}
if((LPC_TIM3->IR & (1<<3))){
LPC_TIM3->IR|= (1<<3);
if(flag_reproducion==1){
//
LPC_DAC->DACR= audio1[j++] << 6; // ?
if(j==8000)
{
flag_reproducion=0;
j=0;
```

```
}
}
}
}



#ifdef RTX_KERNEL
U64 tcp_stack[800/8];                   /* A bigger stack for tcp_task */

/* Forward references */
__task void init      (void);
__task void blink_led  (void);
__task void timer_task (void);
__task void tcp_task   (void);
#endif

/*---------------------------- init ------------------------------------------*/

#ifndef RTX_KERNEL

static void init () {
/* Add System initialisation code here */

//init_io ();
init_TcpNet ();

/* Setup and enable the SysTick timer for 100ms. */
SysTick->LOAD = (SystemCoreClock / 10) - 1;
SysTick->CTRL = 0x05;
}

#else

__task void init (void) {
/* Add System initialisation code here */

//init_io ();
init_TcpNet ();

/* Initialize Tasks */
os_tsk_prio_self (100);
os_tsk_create (blink_led, 20);
os_tsk_create (timer_task, 30);
os_tsk_create_user (tcp_task, 0, &tcp_stack, sizeof(tcp_stack));
os_tsk_delete_self();
}

#endif

/*---------------------------- timer_poll ------------------------------------*/
```

```
#ifndef RTX_KERNEL

static void timer_poll () {
/* System tick timer running in poll mode */

if (SysTick->CTRL & 0x10000) {
/* Timer tick every 100 ms */
timer_tick ();
tick = __TRUE;
}
}

#else

__task void timer_task (void) {
/* System tick timer task */
os_itv_set (10);
while (1) {
timer_tick ();
tick = __TRUE;
os_itv_wait ();
}
}

#endif

/*---------------------------- init_io -----------------------------------------*/

//static void init_io ()
//{

/* Configure the GPIO for Push Buttons */
//LPC_PINCON->PINSEL4 &= 0xFFCFFFFF;
// LPC_GPIO2->FIODIR   &= 0xFFFFFBFF;


/* Configure the GPIO for LED1 and LED2. */
//LPC_GPIO3->FIODIR   |= (1<<25)|(1<<26);
//}
/*---------------------------- fputc -------------------------------------------*/

//int fputc(int ch, FILE *f)
//          {
/* Debug output to serial port. */

// if (ch == '\n')  {
//    while (!(LPC_UART0->LSR & 0x20));
//  LPC_UART0->THR = 0x0D;
//  }
// while (!(LPC_UART0->LSR & 0x20));
//LPC_UART0->THR = (ch & 0xFF);
//return (ch);
//}
```

```
/*---------------------------- LED_out ---------------------------------------*/

void LED_out (U32 val) {
const U8 led_pos[2] = { 25, 26 }; // LED1 and LED2 (P3.25 y P3.26)
U32 i,mask;

for (i = 0; i < 2; i++) {
mask = 1 << led_pos[i];
if (val & (1<<i)) {
//  LPC_GPIO3->FIOCLR = mask;  // LED ON
}
else {
// LPC_GPIO3->FIOSET = mask;  // LED OFF
}
}

}


/*---------------------------- AD_in -----------------------------------------*/

U16 AD_in (U32 ch) {
/* Read ARM Analog Input */
U32 val = 0;
U32 adcr;

if (ch < 8) {
adcr = 0x01000000 | (1 << ch);
LPC_ADC->ADCR = adcr | 0x00200100;       /* Setup A/D: 10-bit @ 9MHz  */

do {
val = LPC_ADC->ADGDR;                /* Read A/D Data Register    */
} while ((val & 0x80000000) == 0);       /* Wait for end of A/D Conv. */
LPC_ADC->ADCR &= ~adcr;                /* Stop A/D Conversion       */
val = (val >> 6) & 0x03FF;             /* Extract AINx Value (10 bits) */
}

return (val);
}

U16 AD_in1 (U32 ch) {
/* Read ARM Analog Input */
U32 val = 0;
Velocidad();

if (ch==1) {
val=velo1;
}
if (ch==2){
val=velo2;
}
```

```
return (val);
}

/*---------------------------- get_button -----------------------------------*/

U8 get_button (void) {
/* Read Mini-DK2 Digital Input */
U32 val = 0;
/* ISP button ?*/
if ((LPC_GPIO2->FIOPIN & (1 << 10)) == 0) val |= 0x01; // P2.10 ?

/* KEY1 button ?*/
if ((LPC_GPIO2->FIOPIN & (1 << 11)) == 0) val |= 0x02; // P2.11 ?

/* KEY2 button ? */
if ((LPC_GPIO2->FIOPIN & (1 << 12)) == 0) val |= 0x04; // P2.12 ?

return (val);
}


/*---------------------------- upd_display ----------------------------------*/

static void upd_display () {
/* Update GLCD Module display text. */

fillScreen(BLACK);
drawString(60,270,"IP:", WHITE,BLACK, MEDIUM);
drawString(52,300,(char *)lcd_text[1], WHITE,BLACK, MEDIUM);
//GUI_Text(60,144,lcd_text[0],White,Red);
//GUI_Text(52,160,lcd_text[1],White,Red);

LCDupdate =__FALSE;
}

/*---------------------------- dhcp_check -----------------------------------*/

static void dhcp_check () {
/* Monitor DHCP IP address assignment. */

if (tick == __FALSE || dhcp_tout == 0) {
return;
}
#ifdef RTX_KERNEL
tick = __FALSE;
#endif
if (mem_test (&MY_IP, 0, IP_ADRLEN) == __FALSE && !(dhcp_tout & 0x80000000)) {
/* Success, DHCP has already got the IP address. */
dhcp_tout = 0;
sprintf((char *)lcd_text[0]," IP address:");
sprintf((char *)lcd_text[1]," %d.%d.%d.%d", MY_IP[0], MY_IP[1],
MY_IP[2], MY_IP[3]);
LCDupdate = __TRUE;
```

```
return;
}
if (--dhcp_tout == 0) {
/* A timeout, disable DHCP and use static IP address. */
dhcp_disable ();
sprintf((char *)lcd_text[1]," DHCP failed    " );
LCDupdate = __TRUE;
dhcp_tout = 30 | 0x80000000;
return;
}
if (dhcp_tout == 0x80000000) {
dhcp_tout = 0;
sprintf((char *)lcd_text[0]," IP address:");
sprintf((char *)lcd_text[1]," %d.%d.%d.%d", MY_IP[0], MY_IP[1],
MY_IP[2], MY_IP[3]);
LCDupdate = __TRUE;
}
}


/*---------------------------- blink_led -----------------------------------*/

#ifndef RTX_KERNEL

static void blink_led () {
/* Blink the LED1 and LED2 on Mini-DK2 board */
const U8 led_val[2] = {0x01,0x02};
static U32 cnt;

if (tick == __TRUE) {
/* Every 100 ms */
tick = __FALSE;
if (LEDrun == __TRUE) {
LED_out (led_val[cnt]);
if (++cnt >= sizeof(led_val)) {
cnt = 0;
}
}
if (LCDupdate == __TRUE) {
upd_display ();
}
}
}

#else

__task void blink_led () {
/* Blink the LED1 and LED2 on Mini-DK2 board */
const U8 led_val[2] = {0x01, 0x02 };
U32 cnt = 0;

LEDrun = __TRUE;
while(1) {
```

```
/* Every 100 ms */
if (LEDrun == __TRUE) {
LED_out (led_val[cnt]);
if (++cnt >= sizeof(led_val)) {
cnt = 0;
}
}
if (LCDupdate == __TRUE) {
upd_display ();
}
os_dly_wait(10);
}
}

#endif


/*----------------------------------------------------------------------------*/

#ifndef RTX_KERNEL
int main (void) {
/* Main Thread of the TcpNet */

char secuencia[7];
int n=0, aux=0, t=0;
int estado=1, estado3=1, terminado=0, f_uart=0, u=0, distancia_uart=0;


init();

LEDrun = __TRUE;
dhcp_tout = DHCP_TOUT;


ptr_rx=bufferuart;                    // inicializa el puntero de recepción al comienzo del buffer
uart0_init(9600);                                                    // configura la UART0 a 9600
baudios, 8 bits, 1 bit stop


config_motor();
config_pwm1();
init_ADC();
iniADC();
//init_WDT();

TP_Init();

lcdInitDisplay();
TouchPanel_Calibrate();
I2C0Init();
TIMERS_config();
init_TIMER0();
duty1(0.55);
```

```
duty2(0.55);
flaggrabacion=0;


while(1)
{

timer_poll ();
main_TcpNet ();
dhcp_check ();
blink_led ();
//comprobar_punto();
//WDT_Feed();
switch(flagadc){
case 1:
parar();
Reproducirpitido(1);
break;
}


switch (estado)
{


case 1:

drawRect(10, 5, 200, 40, WHITE);
drawString(20, 20, "Control del coche", CYAN, BLACK, MEDIUM);

drawRect(10, 55, 200, 40, WHITE);
drawString(20, 75, "Grabar Audio", CYAN, BLACK, MEDIUM);

drawRect(10, 105, 200, 40, WHITE);
drawString(20, 125, "Control motores", CYAN, BLACK, MEDIUM);

drawRect(10, 155, 200, 40, WHITE);
drawString(20, 175, "Reproducir Audio", CYAN, BLACK, MEDIUM);

drawRect(10, 205, 200, 40, WHITE);
drawString(20, 215, "Cambiar umbral", CYAN, BLACK, MEDIUM);

display.y=0;
display.x=0;
estado=1;

getDisplayPoint(&display, Read_Ads7846(), &matrix ) ;
drawCircle(display.x, display.y, 3, RED);

if((display.y >= 5) && (display.y <= 45)){
estado=2;
estado2=1;
fillScreen(BLACK);
```

```
display.y=0;
display.x=0;
}

if((display.y >= 55) && (display.y <= 95)){
estado=3;
flaggrabacion=1;
init_ADC();
LPC_TIM0->TC=0x00;
fillScreen(BLACK);
display.y=0;
}



if((display.y >= 105) && (display.y <= 145)){
estado=4;
fillScreen(BLACK);
display.y=0;
}

if((display.y >= 155) && (display.y <= 195)){
estado=5;
fillScreen(BLACK);
LPC_TIM3->MR3=(F_pclk/8000)-1;
flag_reproducion=1;

display.y=0;
LPC_TIM0->TC=0x00;
}


if((display.y >= 205) && (display.y <= 245)){
estado=6;
fillScreen(BLACK);
display.y=0;
}
break;


case 2:
switch(estado2){
case 1:

estado2=1;
Menu2();

getDisplayPoint(&display, Read_Ads7846(), &matrix ) ;
drawCircle(display.x, display.y, 3, RED);


if((display.y >= 5) && (display.y <= 45)){
estado2=2;
```

```
display.y=0;
Inidist();
secuencia[n]='A';
n++;
fillScreen(WHITE);}

if((display.y >= 55) && (display.y <= 95)){
estado2=2;
fillScreen(WHITE);
display.y=0;
secuencia[n]='R';
n++;


}


if((display.y >= 105) && (display.y <= 145)){

secuencia[n]='I';
n++;
fillScreen(BLACK);
estado2=1;
display.y=0;


}

if((display.y >= 155) && (display.y <= 195)){

secuencia[n]='D';
n++;
fillScreen(BLACK);
estado2=1;
display.y=0;
}




if((display.y >= 205) && (display.y <= 245)){
t=n;
n=0;
terminado=0;
while(terminado==0){
if(flagadc==1){
parar();
Reproducirpitido(1);
}
else{
switch(estado3){
case 1:
if(n==t){
```

```
terminado=1;
estado2=1;
display.y=0;
fillScreen(BLACK);

}
else{
if(secuencia[n]=='A'){

estado3=2;}

if(secuencia[n]=='R'){
estado3=3;

}
if(secuencia[n]=='I'){
estado3=1;
Izquierda(20);
if(Parado()==0){
Inidist();
n++;}}

if(secuencia[n]=='D'){
estado3=1;
Derecha(20);
if(Parado()==0){
n++;
Inidist();}}
}
break;

case 2:
aux=(int)(secuencia[n+1]);


Recta(aux);

if(Parado()==0){
n+=2;
estado3=1;
Inidist();
}
break;

case 3:
aux=(int) secuencia[n+1];
Atras(aux);
if(Parado()==0){
n+=2;
estado3=1;
Inidist();
}
break;
```

```
}

}}
n=0;

}



if((display.y >= 255) && (display.y <= 295)){
estado=1;
fillScreen(BLACK);
display.y=0;
}
break;

case 2:

drawRect(10, 5, 200, 40, WHITE);
drawString(20, 20, "Recorrer 10 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 50, 200, 40, WHITE);
drawString(20, 70, "Recorrer 20 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 100, 200, 40, WHITE);
drawString(20, 120, "Recorrer 30 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 150, 200, 40, WHITE);
drawString(20, 170, "Recorrer 40 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 200, 200, 40, WHITE);
drawString(20, 220, "Recorrer 50 cm", CYAN, BLACK, MEDIUM);

estado2=2;

getDisplayPoint(&display, Read_Ads7846(), &matrix ) ;
drawCircle(display.x, display.y, 3, RED);


if((display.y >= 5) && (display.y <= 40)){
secuencia[n]=10;
n++;
estado2=1;
display.y=0;
fillScreen(BLACK);}
if((display.y >= 50) && (display.y <= 90)){
secuencia[n]=20;
n++;
estado2=1;
display.y=0;
fillScreen(BLACK);
}
```

```
if((display.y >= 100) && (display.y <= 140)){
secuencia[n]=30;
n++;
estado2=1;
display.y=0;
fillScreen(BLACK);}

if((display.y >= 150) && (display.y <= 190)){
secuencia[n]=40;
n++;
estado2=1;
display.y=0;
fillScreen(BLACK);}

if((display.y >= 200) && (display.y <= 240)){
secuencia[n]=50;
n++;
estado2=1;
display.y=0;
fillScreen(BLACK);}




break;




}
break;
case 3:
drawString(10, 10, "GRABANDO...", CYAN, BLACK, LARGE);



if(indice==1){
estado=1;
flaggrabacion=0;
iniADC();
fillScreen(BLACK);
display.y=0;
}

break;


case 4:
```

```
drawRect(10, 5, 200, 40, WHITE);
drawString(20, 20, "Volver al principal", CYAN, BLACK, MEDIUM);

drawRect(10, 55, 200, 40, WHITE);
drawString(20, 70, "PWM1++", CYAN, BLACK, MEDIUM);

drawRect(10, 105, 200, 40, WHITE);
drawString(20, 120, "PWM1--", CYAN, BLACK, MEDIUM);

drawRect(10, 155, 200, 40, WHITE);
drawString(20, 175, "PWM2++", CYAN, BLACK, MEDIUM);

drawRect(10, 205, 200, 40, WHITE);
drawString(20, 220, "PWM2--", CYAN, BLACK, MEDIUM);
sprintf(buffer2, "%.3f", ciclo1);
drawString(10,255,buffer2,WHITE, BLACK, MEDIUM);
sprintf(buffer3, "%.3f", ciclo2);
drawString(120,255,buffer3,WHITE, BLACK, MEDIUM);


getDisplayPoint(&display, Read_Ads7846(), &matrix ) ;
drawCircle(display.x, display.y, 3, RED);

if((display.y >= 5) && (display.y <= 45)){
estado=1;
fillScreen(BLACK);
display.y=0;
}

if(((display.y >= 55) && (display.y <= 95))){
Sumarciclo1();

display.y=0;
}
if(((display.y >= 105) && (display.y <= 145))){
Restarciclo1();
display.y=0;
}

if(((display.y >= 155) && (display.y <= 195))){
Sumarciclo2();
display.y=0;
}
if(((display.y >= 205) && (display.y <=245))){
Restarciclo2();
display.y=0;
}
break;

case 5:
drawString(90, 5, "REPRODUCIENDO...", CYAN, BLACK, MEDIUM);
drawRect(10, 5, 200, 40, WHITE);
init_DAC();
```

```
NVIC_EnableIRQ(TIMER0_IRQn);

if(flag_reproducion==0){
LPC_PINCON->PINSEL1&=~ (2<<20);
fillScreen(BLACK);
estado=1;
LPC_TIM3->MR3=F_pclk;
}
break;

case 6:
estado=6;
drawRect(10, 5, 200, 40, WHITE);
drawString(20, 20, "Sumar 1 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 50, 200, 40, WHITE);
drawString(20, 70, "Restar 1 cm", CYAN, BLACK, MEDIUM);

drawRect(10, 100, 200, 40, WHITE);
drawString(20, 120, "Volver al principal", CYAN, BLACK, MEDIUM);

sprintf(buffer2, "%.3f", umbralcm);
drawString(10,200,buffer2,WHITE, BLACK, MEDIUM);

getDisplayPoint(&display, Read_Ads7846(), &matrix ) ;
drawCircle(display.x, display.y, 3, RED);
if((display.y >= 5) && (display.y <= 45)){
SumarUmbral();
display.y=0;
}
if((display.y >= 50) && (display.y <= 90)){
RestarUmbral();
display.y=0;
}
if((display.y >= 100) && (display.y <= 140)){
estado=1;
display.y=0;
fillScreen(BLACK);

}


break;
}

}

}



#else
```

```
__task void tcp_task (void) {
/* Main Thread of the TcpNet. This task should have */
/* the lowest priority because it is always READY. */
dhcp_tout = DHCP_TOUT;
while (1) {
main_TcpNet();
dhcp_check ();
os_tsk_pass();
}
}

/*
int main (void) {
/* Start with 'init' task.
os_sys_init(init);
while(1);
}
*/

#endif


**************************** uart.c ****************************************

#include <LPC17xx.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "uart.h"
#include "audio.h"
#include "movimentos.h"

#define F_pclk 25e6

uint8_t flag_segundo=0;
char bufferuart[20];                // Buffer de recepción de 30 caracteres
char *ptr_rx;                       // puntero de recepción
char rx_completa;                   // Flag de recepción de cadena que se activa a "1" al recibir la tecla
return CR(ASCII=13)
char *ptr_tx;                       // puntero de transmisión
char tx_completa;                   // Flag de transmisión de cadena que se activa al transmitir el caracter
null (fin de cadena)
char fin=0;
char finhttp=0;


void LeerComando(void)
{

        int distancia, estado=1, n=0, av=0, t=0;
        char *aux=NULL, *aux1=NULL;
```

```
//while(rx_completa==0); // Espera introducir una cadena de caracteres terminada con CR (0x0D)
	//rx_completa=0;                                        // Borrar flag

tx_cadena_UART0("\nIntroduce un comando:(Recta, Marcha atras, Girar a la derecha, Girar a la izquierda
o  fin)\n\n\n\r");
do{
	if(rx_completa){                                                    // Comprabamos la llegada de una
cadena por RXD

		if(flagadc==1)
			Reproducirpitido(1);
		else{

		switch(estado){
			case 1:
				if (bufferuart[n]=='A') {
					estado=2;
					av=1;



				}
				else if (bufferuart[n]=='R') {
					estado=2;
					av=0;

				}
				else if (bufferuart[n]=='D') {
					Derecha(20);
					if(Parado()==0){
						av=1;
						estado=2;

					}


				}
				else if (bufferuart[n]=='I') {
					Izquierda(20);
					if(Parado()==0){
						av=1;
						estado=2;

					}}

				else if (bufferuart[n]=='F') {
					fin=1;
						rx_completa=0;                                  // Borrar
flag para otra recepción
				}
				else if (bufferuart[n]=='G') {
					if(t==0){
						flaggrabacion=1;
```

```
                                        t=1;
                                        init_ADC();
                                        LPC_TIM0->TC=0x00;
                                }
                                if(t==1){
                                                if(indice==1){
                                                                estado=1;
                                                                t=0;
                                                                flaggrabacion=0;
                                                        iniADC();

        NVIC_DisableIRQ(ADC_IRQn);
                                                                n++;
                                                        iniADC();
                                                }


                                }}
                                else if (bufferuart[n]=='S') {
                                if(t==0){
                                        LPC_TIM3->MR3=F_pclk/8000-1;
                                        flag_reproducion=1;
                                                init_DAC();
                                 NVIC_DisableIRQ(ADC_IRQn);
                                         NVIC_SetPriority(TIMER3_IRQn,3);


                                        t=1;
                                //      LPC_TIM3->TC=0x00;
                                }
                                if(t==1){
                                                                init_DAC();

        //NVIC_EnableIRQ(TIMER0_IRQn);




        if(flag_reproducion==0){

NVIC_SetPriority(TIMER3_IRQn,4);

        NVIC_EnableIRQ(ADC_IRQn);

        LPC_PINCON->PINSEL1&=~ (2<<20);

                                                                        t=0;

        LPC_TIM3->MR3=F_pclk;
                                                                        n++;
                                                                }
```

```
                                        }}


                                    else {tx_cadena_UART0("\nComando erroneo\n\r"); fin=1;}
                                    break;


                            case 2:
                                        aux=&bufferuart[n+1];
                                        Distancia();
                                        distancia=atoi(aux);
                                        if( av==1)
                                                Recta(distancia);
                                        else
                                                Atras(distancia);

                                        if(Parado()==0){
                                                estado=1;
                                                n+=3;
                                                Inidist();;
                                        }
                                        break;



                                    }
                        }

                }}while(fin==0);
tx_cadena_UART0("FIN del programa\n\r");
}
// Función para enviar una cadena de texto
// El argumento de entrada es la dirección de la cadena, o
// directamente la cadena de texto entre comillas1111
/*
 * UART0 interrupt handler
 */
void UART0_IRQHandler(void) {

   switch(LPC_UART0->IIR&0x0E) {

            case 0x04:                                                                  /* RBR, Receiver
Buffer Ready */
            *ptr_rx=LPC_UART0->RBR;                                         /* lee el dato recibido y lo
almacena */
                if (*ptr_rx++ ==13)                                     // Caracter return --> Cadena completa
                                                                {
                                                                        *ptr_rx=0;                      /* Añadimos el
caracter null para tratar los datos recibidos como una cadena*/
                                                                        rx_completa = 1;/* rx completa */
                                                                        ptr_rx=bufferuart;          /* puntero al
inicio del buffer para nueva recepción */
                                                                 fin=0;
```

```
                                                    LeerComando();
                                                }
                    break;


  case 0x02:                                                    /* THRE, Transmit Holding
Register empty */
                    if (*ptr_tx!=0) LPC_UART0->THR=*ptr_tx++;          /* carga un nuevo dato para ser
transmitido */
                    else tx_completa=1;
                    break;

   }
}



void tx_cadena_UART0(char *cadena)
{
ptr_tx=cadena;
tx_completa=0;
LPC_UART0->THR=*ptr_tx++;        // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
}                                                   // activar flag interrupción por registro
transmisor vacio


static int uart0_set_baudrate(unsigned int baudrate) {
   int errorStatus = -1; //< Failure

   // UART clock (FCCO / PCLK_UART0)
  // unsigned int uClk = SystemCoreClock / 4;
   unsigned int uClk =SystemCoreClock/4;
   unsigned int calcBaudrate = 0;
   unsigned int temp = 0;

   unsigned int mulFracDiv, dividerAddFracDiv;
   unsigned int divider = 0;
   unsigned int mulFracDivOptimal = 1;
   unsigned int dividerAddOptimal = 0;
   unsigned int dividerOptimal = 0;

   unsigned int relativeError = 0;
   unsigned int relativeOptimalError = 100000;

   uClk = uClk >> 4; /* div by 16 */

   /*
    *  The formula is :
    * BaudRate= uClk * (mulFracDiv/(mulFracDiv+dividerAddFracDiv) / (16 * DLL)
    *
    * The value of mulFracDiv and dividerAddFracDiv should comply to the following expressions:
    * 0 < mulFracDiv <= 15, 0 <= dividerAddFracDiv <= 15
    */
```

```
   for (mulFracDiv = 1; mulFracDiv <= 15; mulFracDiv++) {
      for (dividerAddFracDiv = 0; dividerAddFracDiv <= 15; dividerAddFracDiv++) {
         temp = (mulFracDiv * uClk) / (mulFracDiv + dividerAddFracDiv);

         divider = temp / baudrate;
         if ((temp % baudrate) > (baudrate / 2))
            divider++;

         if (divider > 2 && divider < 65536) {
            calcBaudrate = temp / divider;

            if (calcBaudrate <= baudrate) {
               relativeError = baudrate - calcBaudrate;
            } else {
               relativeError = calcBaudrate - baudrate;
            }

            if (relativeError < relativeOptimalError) {
               mulFracDivOptimal = mulFracDiv;
               dividerAddOptimal = dividerAddFracDiv;
               dividerOptimal = divider;
               relativeOptimalError = relativeError;
               if (relativeError == 0)
                  break;
            }
         }
      }

      if (relativeError == 0)
         break;
   }

   if (relativeOptimalError < ((baudrate * UART_ACCEPTED_BAUDRATE_ERROR) / 100)) {

      LPC_UART0->LCR |= DLAB_ENABLE;      // importante poner a 1
      LPC_UART0->DLM = (unsigned char) ((dividerOptimal >> 8) & 0xFF);
      LPC_UART0->DLL = (unsigned char) dividerOptimal;
      LPC_UART0->LCR &= ~DLAB_ENABLE;   // importante poner a 0

      LPC_UART0->FDR = ((mulFracDivOptimal << 4) & 0xF0) | (dividerAddOptimal & 0x0F);

      errorStatus = 0; //< Success
   }

   return errorStatus;
}

void uart0_init(int baudrate) {

   LPC_PINCON->PINSEL0|=(1<<4)|(1<<6);// Change P0.2 and P0.3 mode to TXD0 and RXD0

   LPC_UART0->LCR &= ~STOP_1_BIT & ~PARITY_NONE; // Set 8N1 mode (8 bits/dato, sin pariad, y 1 bit
de stop)
```

```
   LPC_UART0->LCR |= CHAR_8_BIT;

   uart0_set_baudrate(baudrate);// Set the baud rate


   LPC_UART0->IER = THRE_IRQ_ENABLE|RBR_IRQ_ENABLE;// Enable UART TX and RX interrupt (for
LPC17xx UART)
   NVIC_EnableIRQ(UART0_IRQn);// Enable the UART interrupt (for Cortex-CM3 NVIC)

                    NVIC_SetPriority(UART0_IRQn,5);


}


void LeerSe(char* bufferhttp)
{


          int distancia, estado=1, n=0, av=0, t=0;
          char *aux=NULL, *aux1=NULL;


do{


                    if(flagadc==1)
                              Reproducirpitido(1);
                    else{

                    switch(estado){
                              case 1:
                                        if (bufferhttp[n]=='A') {
                                                  estado=2;
                                                  av=1;



                                        }
                                        else if (bufferhttp[n]=='R') {
                                                  estado=2;
                                                  av=0;

                                        }
                                        else if (bufferhttp[n]=='D') {
                                                  Derecha(20);
                                                  if(Parado()==0){
                                                            av=1;
                                                            estado=2;

                                                  }
```

```
                                        }
                                        else if (bufferhttp[n]=='I') {
                                                Izquierda(20);
                                                if(Parado()==0){
                                                        av=1;
                                                        estado=2;

                                                }}

                                        else if (bufferhttp[n]=='F') {
                                                finhttp=1;

                                        }
                                        else if (bufferhttp[n]=='G') {
                                                if(t==0){
                                                        flaggrabacion=1;
                                                        t=1;
                                                        init_ADC();
                                                        LPC_TIM0->TC=0x00;
                                                }
                                                if(t==1){
                                                                        if(indice==1){
                                                                                        estado=1;
                                                                                        t=0;
                                                                                        flaggrabacion=0;
                                                                                iniADC();

NVIC_DisableIRQ(ADC_IRQn);
                                                                                        n++;
                                                                                iniADC();
                                                                        }


                                                }}
                                                else if (bufferhttp[n]=='S') {
                                                if(t==0){
                                                        LPC_TIM3->MR3=F_pclk/8000-1;
                                                        flag_reproducion=1;
                                                                        init_DAC();
                                                 NVIC_DisableIRQ(ADC_IRQn);
                                                         NVIC_SetPriority(TIMER3_IRQn,3);


                                                        t=1;
                                                //      LPC_TIM3->TC=0x00;
                                                }
                                                if(t==1){
                                                                                        init_DAC();

//NVIC_EnableIRQ(TIMER0_IRQn);
```

```
        if(flag_reproducion==0){

NVIC_SetPriority(TIMER3_IRQn,4);

        NVIC_EnableIRQ(ADC_IRQn);

        LPC_PINCON->PINSEL1&=~ (2<<20);

                                                                                t=0;

        LPC_TIM3->MR3=F_pclk;
                                                                                n++;
                                                                        }


                                        }}


                                else { finhttp=1;}
                                break;


                        case 2:
                                        aux=&bufferhttp[n+1];
                                        Distancia();
                                        distancia=atoi(aux);
                                        if( av==1)
                                                Recta(distancia);
                                        else
                                                Atras(distancia);

                                        if(Parado()==0){
                                                estado=1;
                                                n+=3;
                                                Inidist();;
                                        }
                                        break;



                                }
                }

        }while(finhttp==0);
        finhttp=0;
}
```

```
********** i2c.c **********

/**********************************/
/*Funciones de control del bus I2C*/
/* SDA=P0.0 y SCL=P0.1          */
/**********************************/
#include <LPC17xx.h>
#include "i2c.h"
#include "movimentos.h"

#define SDA 0 //pin 0
#define SCL 1 //pin 1
#define dir_nunchunk 0x52 //direccion del nunchunk




void I2C0Init( void )
{

        I2CSendAddr(dir_nunchunk, 0);
        I2CSendByte(0xF0);
        I2CSendByte(0x55);
        I2CSendStop();
        I2CSendAddr(dir_nunchunk, 0);
        I2CSendByte(0xFB);
        I2CSendByte(0x00);
        I2CSendStop();


}

//Calculates and returns the xAxis acceleration in Gs
void I2Cdelay(void)//retardo minimo de 4.7 us
{
 unsigned char i;
 for(i=0;i<100;i++); //Modificar límite para garantizar los tiempos (Bus standar -->F_max=100kHz)
}

//Genera un pulso de reloj (1 ciclo)
void pulso_SCL(void)
{
        LPC_GPIO0->FIOSET=(1<<SCL);     // Genera pulso de reloj (nivel alto)
        I2Cdelay();
        LPC_GPIO0->FIOCLR=(1<<SCL);     // Nivel bajo
        I2Cdelay();
}

void I2CSendByte(unsigned char byte)
{
 unsigned char i;
```

```
 for(i=0;i<8;i++){

            if (byte &0x80)  LPC_GPIO0->FIOSET=(1<<SDA); // envia cada bit, comenzando por el MSB
            else  LPC_GPIO0->FIOCLR=(1<<SDA);
            byte = byte <<1;              // siguiente bit
            pulso_SCL();
           }

//Leer ACK que envía el Slave (el Master ha de enviar un pulso de reloj)
// CONFIGURAR PIN SDA COMO ENTRADA;       //espera ACK(config. pin como entrada)
 LPC_GPIO0->FIODIR&=~(1<<SDA);
 pulso_SCL();

// CONFIGURA PIN SDA COMO SALIDA;
 LPC_GPIO0->FIODIR|=(1<<SDA);                        // Dejamos SDA de nuevo como salida
}


//Función que envía START + Byte de direcci el Slave (con bit LSB inicando R/W)
void I2CSendAddr(unsigned char addr, unsigned char rw)
{
 //CONFIGURAR PINs SDA, SCL COMO SALIDAS;  // Por si se nos olvidada en la conf. general.
 LPC_GPIO0->FIODIR|=(1<<SDA)|(1<<SCL);

 LPC_GPIO0->FIOSET|=(1<<SDA)|(1<<SCL);    // SDA y SCL a nivel alto para garantizar el
                                                                                                  //     nivel     de
reposo del bus + tiempo.
 I2Cdelay();
 LPC_GPIO0->FIOCLR=(1<<SDA);
//condicion de START: Bajar SDA y luego SCL
 I2Cdelay();
 LPC_GPIO0->FIOCLR=(1<<SCL);
 I2Cdelay();

 I2CSendByte((addr=addr<<1) + rw);  //envia byte de direccion
                                                                                  //addr, direccion (7bits)
                                                                  //rw=1, lectura
                                                                                  //rw=0, escritura

}


// Función para leer un Byte del Slave. El Master envía al final de la lectura
// el bit ACK o NACK (si es último byte leído) que se pasa como argumento de la función.
unsigned char I2CGetByte(unsigned char ACK)
{
// ACK = 0, para cualquier byte que no sea el ultimo.
// ACK = 1 (NACK), despues de leer el ultimo byte
 unsigned char i, byte;
 //CONFIGURAR PIN SDA COMO ENTRADA;  //configura pin SDA como entrada
 LPC_GPIO0->FIODIR&=~(1<<SDA);
 for(i=0;i<8;i++){                                    //lee un bit comenzando por el MSB
```

```
                    LPC_GPIO0->FIOSET=(1<<SCL);//mientras SCL=1
                    I2Cdelay();
                    byte=byte<<1;
                    if(LPC_GPIO0->FIOPIN&(1<<SDA)) byte++;                    //Si leemos "1" sumamos
para introducir el "1"
                    LPC_GPIO0->FIOCLR=(1<<SCL);                                        //Si  leemos  "0"
solo desplazamos (se introduce un "0")
                    I2Cdelay();
                   }

 //CONFIGURAR PIN SDA COMO SALIDA;   // Master envía un ACK por cada byte leído.
 LPC_GPIO0->FIODIR|=(1<<SDA);

 if(ACK)LPC_GPIO0->FIOSET=(1<<SDA);         // ACK o (NACK) es funcion del último byte leído
        else LPC_GPIO0->FIOCLR=(1<<SDA);

 pulso_SCL();                                    // Pulso de reloj para su envío
 return (byte);
}


void I2CSendStop(void)
{
 LPC_GPIO0->FIOCLR=(1<<SDA);
 I2Cdelay();
 LPC_GPIO0->FIOSET=(1<<SCL);                // Subir SCL, y después SDA!! para dejar el bus en reposo
 I2Cdelay();
 LPC_GPIO0->FIOSET=(1<<SDA);
 I2Cdelay();
}

void MostrarI2C(unsigned char* buffer)
{
        char b1[8];

        sprintf(b1,"%d",buffer[2]);
        drawString(10, 100, b1, RED, BLACK, LARGE);
        sprintf(b1,"%d",buffer[3]);
        drawString(10, 150, b1, RED, BLACK, LARGE);
        sprintf(b1,"%d",buffer[4]);
        drawString(10, 200, b1, RED, BLACK, LARGE);
}
```

```
************************** movimientos.c ***************************

#include <LPC17xx.H>
#include <Math.h>
#include "movimentos.h"
#include "audio.h"
#define radio 3.5
#include "i2c.h"

#define N_samples_wave      128                                          // Nº de muestras por
ciclo


#define pi 3.1415
#define F_cpu 100e6                                          // Defecto Keil (xtal=12Mhz)
#define F_pclk F_cpu/4                        // Defecto despues del reset

#define Fvelocidad          2 //Frecuencia para mostrar la velicidad

#define F_muestreo 8000


float limite=1.45, umbralcm=18.6; // valor maximo por el cual paramos los motores.
float Vo_sensor;
uint32_t N1, N2, anterior, anterior2, d1, d2=0;
int Nflancos1, Nflancos2;

float distancia1=0.0, distancia2=0.0, velo1=0.0, velo2=0.0;
char buffer1[5],volt[5];
int audio1[8000];
int i=0;
int pararf=0,flag=0, flagadc=0, flaggrabacion, indice;



void iniADC(void)
{
LPC_SC->PCONP|= (1<<12);                                          // POwer ON
LPC_PINCON->PINSEL1|= (1<<18);            // ADC input= P0.25 (AD0.2)
LPC_PINCON->PINMODE1|= (2<<18);           // Deshabilita pullup/pulldown
LPC_SC->PCLKSEL0|= (0x00<<8);             // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
LPC_ADC->ADCR= (0x01<<2)|                           // Canal 2
(0x01<<8)|                              // CLKDIV=1   (Fclk_ADC=25Mhz /(1+1)= 12.5Mhz)
(0x01<<21)|                                 // PDN=1
(7<<24);                                 // Inicio de conversión con el Match 0 del Timer 1

LPC_ADC->ADINTEN= (1<<0)|(1<<2);          // Hab. interrupción fin de conversión canal 0 (DONE canal 0
+ DONE global)
NVIC_SetPriority(ADC_IRQn,2);                           //
NVIC_SetPriorityGrouping(3);
NVIC_EnableIRQ(ADC_IRQn);
```

```
}


void ADC_IRQHandler(void)
{

if(((LPC_ADC->ADGDR>>24)&0x7)==0x0)//audio
{
if(flaggrabacion==1)
{


audio1[i++]=(int)((LPC_ADC->ADDR0 >>4)&0xFF);

if(i>=8000)
{
indice=1;
i=0;
//flaggrabacion=0;
//NVIC_DisableIRQ(ADC_IRQn);
}
}
}
else if(((LPC_ADC->ADGDR>>24)&0x7)==0x2){
Vo_sensor= (float)((LPC_ADC->ADDR2 >>4)&0xFFF)*3.3/4095;          // se borra automat. el flag DONE al
leer ADCGDR
indice=0;
if(Vo_sensor >= limite)
{

flagadc=1;

}
else
flagadc=0;
}
}

int Flag1(void)
{
if(flag==0)
return 0;
if(flag==1)
return 1;
return 0;
}

int Parado(void)
{
if(pararf==0)
return 0;
if(pararf==1)
return 1;
```

```
return 0;
}


void TIMERS_config(void)
{
LPC_SC->PCONP|= (1<<2);                                    // Power ON:  timer1

LPC_PINCON->PINSEL3|= (3<<4);  //Pin1.18 como Capture1.0
LPC_PINCON->PINSEL3|= (3<<6);  //Pin1.19 como Capture1.1
LPC_TIM1->PR = 0x0000;   //?
LPC_TIM1->MCR = 0x0003;  //Se realiza una interrupcion cuando se alcanza MR1  se resetean
LPC_TIM1->CCR = 0x0036;  // flanco bajada CAP1.0 e interrupcion
LPC_TIM1->EMR = 0x00C1;  //?
LPC_TIM1->TCR = 0x01;    //?
LPC_TIM1->MR0 = F_pclk/5 -1; // Se han de producir DOS Match para iniciar la conversión!!!!


NVIC_SetPriority(TIMER1_IRQn,1);   //?
NVIC_EnableIRQ(TIMER1_IRQn);      //?



//####### TIMER 3 #########/
LPC_SC->PCONP |= (1<<23);
LPC_TIM3->PR = 0x0000;
LPC_TIM3->MCR = 0x006C9; //MAtch 1
LPC_TIM3->EMR |= (3<<4) | (1<<0);
LPC_TIM3->TCR = 0x01;
LPC_TIM3->MR0=2*F_pclk -1 ;
LPC_TIM3->MR1= F_pclk/7-1;
LPC_TIM3->MR2= F_pclk/2-1;
LPC_TIM3->MR3=F_pclk;



NVIC_SetPriority(TIMER3_IRQn,4);   //?
NVIC_EnableIRQ(TIMER3_IRQn);      //?

}



void Distancia(void) //calculo de la distancia reccorrida a partir del numero de flancos
{
distancia1= 2*pi*radio*Nflancos1/20;        //medidad en centimetros
distancia2= 2*pi*radio*Nflancos2/20;        //medidad en centimetros

}


void Velocidad(void) //Calcular velocidad
```

```
{


Distancia();
N1=100*LPC_TIM1->CR0-anterior;
N2=100*LPC_TIM1->CR1-anterior2;
velo1=fabs((distancia1-d1)/(N1/F_pclk));
velo2=distancia2-d1/(N2/F_pclk);
anterior=LPC_TIM1->CR0;
d1=distancia2;

anterior=LPC_TIM1->CR0;
anterior2=LPC_TIM1->CR1;
d1=distancia1;
d2=distancia2;

}


void Atras(int dist)
{
pararf=1;



LPC_GPIO0->FIOCLR |=(1<<4); //ponemos a 0 salida P0.4
LPC_GPIO0->FIOSET |= (1<<24); //ponemos a 1 salida P0.24*/
LPC_GPIO0->FIOSET |= (1<<10); //ponemos a 1 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
Distancia();

if((dist<=distancia2)||flagadc==1)
{
pararf=0;
parar();
flag=0;
}
}

void Recta(int dist)
{
pararf=1;
LPC_GPIO0->FIOSET |=(1<<4); //ponemos a 1 salida P0.0
LPC_GPIO0->FIOCLR|= (1<<24); //ponemos a 0 salida P0.1*/
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOSET |= (1<<11); //ponemos a 1 salida P0.11
Distancia();
if((dist<=distancia2)||flagadc==1)
{
pararf=0;
parar();
flag=0;
}
```

```
}

void Derecha(int dist)
{
pararf=1;
LPC_GPIO0->FIOSET |=(1<<4); //ponemos a 0 salida P0.0
LPC_GPIO0->FIOCLR|= (1<<24); //ponemos a 1 salida P0.1*/
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
Distancia();
if((dist<=distancia1)||(dist<=distancia2)||flagadc==1)
{
parar();
flag=0;
}
}
void Izquierda(int dist)
{
pararf=1;
LPC_GPIO0->FIOSET |=(1<<11); //ponemos a 1 salida P0.11
LPC_GPIO0->FIOCLR|= (1<<10); //ponemos a 0 salida P0.10*/
LPC_GPIO0->FIOCLR |= (1<<4); //ponemos a 0 salida P0.0
LPC_GPIO0->FIOCLR |= (1<<24); //ponemos a 0 salida P0.1
Distancia();
if((dist<=distancia1)||(dist<=distancia2)||flagadc==1)
{
pararf=0;
parar();
flag=0;
}
}


void parar(void)
{
anterior=0;
anterior2=0;
pararf=0;
d1=0;
d2=0;
LPC_GPIO0->FIOCLR |=(1<<4); //ponemos a 0 salida P0.0
LPC_GPIO0->FIOCLR         |= (1<<24); //ponemos a 0 salida P0.1*/
LPC_GPIO0->FIOCLR |= (1<<10); //ponemos a 0 salida P0.10
LPC_GPIO0->FIOCLR |= (1<<11); //ponemos a 0 salida P0.11
//          Inidist();

}



void TIMER1_IRQHandler(void) {
```

```
if ((LPC_TIM1->IR)&(1<<4)){                                    //Comprobar si se activa interrupcion de
CAP1.0
LPC_TIM1->IR|= (1<<4);              // borra el flag
Nflancos1++;


}
if ((LPC_TIM1->IR)&(1<<5)){                              //Comprobar si se activa interrupcion de CAP1.1
LPC_TIM1->IR|= (1<<5);              // borra el flag
Nflancos2++;
N2=LPC_TIM1->CR1;//-temp2;

}
if((LPC_TIM1->IR & (1<<0)))
LPC_TIM1->IR |= (1<<0);



if(((LPC_TIM1->IR & (1<<1)))){

if(pararf==1){


flag=1;

}
LPC_TIM1->IR |= (1<<1);

}
}
```

```
void Inidist(void){
anterior=0;
d1=0;
distancia1=0.0;

distancia2=0.0;
Nflancos1=0;
Nflancos2=0;
velo1=0.0;
velo2=0.0;
}

void MostrarVelocidad(void){
Velocidad();
sprintf(buffer1, "%.3f", velo1);
drawString(10, 200, buffer1, RED, BLACK, LARGE);
```

```
flag=0;
}

void MostrarSensor(void)
{
sprintf(volt, "%.3f", Vo_sensor);
drawString(70,140,volt,WHITE, BLACK, MEDIUM);
}


void SumarUmbral(void)
{
umbralcm+=1;
limite=((1.9/0.08)*(1/(umbralcm+0.42))) +0.2;
}


void RestarUmbral(void)
{
umbralcm-=1;
if(umbralcm<=0)
umbralcm=0;
limite=((1.9/0.08)*(1/(umbralcm+0.42))) +0.2;

}
```

********** motores.c **********

```
#include <LPC17xx.H>
#include "lcddriver.h"
#include "motor.h"
```

```
#include "lpc17xx_gpio.h"

#define Fpclk 25e6          // Fcpu/4 (defecto después del reset)
#define Fpwm 500            // Frecuencia de la señal PWM (500Hz)
#define MR0_fin             Fpclk/Fpwm-1 // Valor de MR0 5BBD7


float ciclo1=0.55,ciclo2=0.55;

void config_motor(void) //configuracion pines motores
{
LPC_PINCON->PINSEL0&=~(3<<8); // P0.0 como GPIO
LPC_PINCON->PINSEL1&=~(3<<16); // P0.1 como GPIO
LPC_GPIO0->FIODIR|=(1<<4); //puerto habilitado como output
LPC_GPIO0->FIODIR|=(1<<24); // puerto habilitado como output

LPC_PINCON->PINSEL0&=~(3<<20); // P0.10 como GPIO
LPC_PINCON->PINSEL0&=~(3<<22); // P0.11 como GPIO
LPC_GPIO0->FIODIR|=(1<<10); //puerto habilitado como output
LPC_GPIO0->FIODIR|=(1<<11); // puerto habilitado como output



};

void config_pwm1(void)// Configuracion de PWM1
{
LPC_PINCON->PINSEL7|=(3<<18); // P3.25 salida PWM (PWM1.2)
LPC_PINCON->PINSEL7|=(3<<20); // P3.26 salida PWM (PWM1.3)
LPC_SC->PCONP|=(1<<6); // Habiltar alimentacion
LPC_PWM1->MR0=Fpclk/Fpwm-1;
LPC_PWM1->PCR|=(1<<10); //Habilitar salida del PWM1.2
LPC_PWM1->PCR|=(1<<11); //Habilitar salida del PWM1.3
LPC_PWM1->PCR&=~(1<<2);//Deshabilito flancon doble 1.2
LPC_PWM1->PCR&=~(1<<3);//Deshabilito flancon doble 1.3
LPC_PWM1->MCR|=(1<<1); // Se resetea el contador cuando se llega a MR0
LPC_PWM1->TCR|=(1<<0)|(1<<3);//Habilitar timer y habilitar modo PWM
};

void duty2(float ciclo)//configuracion de MR3
{
LPC_PWM1->MR3=ciclo*MR0_fin;// tiempo en alta
LPC_PWM1->LER|=(1<<3)|(1<<0);//SE reinicia el contador MR0 y MR3
};
void duty1(float ciclo)//configuracion de MR2
{
LPC_PWM1->MR2=ciclo*MR0_fin;
LPC_PWM1->LER|=(1<<2)|(1<<0);
};
```

```
void Sumarciclo1(void){
ciclo1+=0.1;
if(ciclo1>=1)
ciclo1=1;
duty1(ciclo1);

};

void Sumarciclo2(void){
ciclo2+=0.1;
if(ciclo2>=1)
ciclo2=1;
duty2(ciclo2);
};

void Restarciclo2(void){
ciclo2-=0.1;
if(ciclo2<=0)
ciclo2=0;
duty2(ciclo2);
};

void Restarciclo1(void){
ciclo1-=0.1;
if(ciclo1<=0)
ciclo1=0;
duty1(ciclo1);
}
;
```

********** audio.c **********

```
#include <LPC17xx.H>
#include <math.h>
#include "movimentos.h"
#define F_cpu 100e6                                    // Defecto Keil (xtal=12Mhz)
```

```
#define F_pclk F_cpu/4                          // Defecto despues del reset
#define F_muestreo 8000             // Frecuencia de muestreo del ADC

//Configuracion del DMA canal 1 (ADC)
#define WIDTH_VALUE 0                                   // Size Mode Transfer = 8 bits
#define OFFSET_ADC 0x11                         // Para acceder a ADDR0[8..15]
#define SBURST 0x00
#define DBURST 0x00
#define TAM_BLOCK_DMA 1000
          // Tamaño del bloque (max. 4095)
unsigned char buffer[TAM_BLOCK_DMA];                                    // destino de las muestras
volatile  int x,escala, DMA_completa;
int flag_reproducion=0;
// Generador senoidal con el DAC
#define N_samples_wave      1000                                                        //      Nº      de
muestras por ciclo
#define PI          (float)(3.141592654)
uint32_t F_out=100;
                                // Frecuencia salida senoidal
uint32_t sinusoide[N_samples_wave];


// Linked struct DMA canal 0 (DAC)
struct {
uint32_t  source;                       // Start of source area
uint32_t  destination;// Start of destination area
uint32_t  next;      // Address of next strLLI in chain
uint32_t  control;   // DMACCxControl register
} LLI0;

void delay_mseg(uint16_t ms)
{
uint16_t i,j;
for( i = 0; i < ms; i++ )       for( j = 0; j < 1141; j++ );
}


/*Timer 0 en modo Output Compare (reset T0TC on Match 1)
Counter clk: 25 MHz         Start Conversion ADC --> MAT0.1 (toggle on Match)
Cada 2 Match se provoca el INICIO DE CONVERSIÓN DEL ADC*/
void init_TIMER0(void)
{
LPC_SC->PCONP|=(1<<1);                                                  //
LPC_TIM0->PR = 0x00;                                                            //
LPC_TIM0->MCR = 0x10;                                                   //
LPC_TIM0->MR1 = (F_pclk/F_muestreo/2)-1; // DOS Match para iniciar la conversión!!!!   Muestreo a 8
kHz
LPC_TIM0->EMR = 0x00C2;

LPC_TIM0->TCR = 0x01;                                           //


}
```

```
void init_ADC(void)
{
LPC_SC->PCONP|= (1<<12);                                        // POwer ON
LPC_PINCON->PINSEL1|= (1<<14);          // ADC input= P0.23 (AD0.0)
LPC_PINCON->PINMODE1|= (2<<14);         // Deshabilita pullup/pulldown
LPC_SC->PCLKSEL0&=~(3<<24);                     // CLK ADC = CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
//LPC_SC->PCLKSEL0|=(1<<24);                    // CLK ADC = CCLK = 100 MHz (Para muestrear a 500kHz)
LPC_ADC->ADCR= 0;
LPC_ADC->ADCR= (0x01<<0)|                        // Canal 0
//                                              (0x02<<8)|                      //      CLKDIV=2 (Fclk_ADC= 100Mhz /(2+1)= 33.3 Mhz) para muestrear a 500Khz!!!
(0x01<<8)|                      // CLKDIV=1   (Fclk_ADC= 25Mhz /(1+1)= 12.5 Mhz)
(4<<24)|                                        // Inicio de conversión con el Match 1 del Timer 0
(0x01<<21);                             // PDN=1
LPC_ADC->ADINTEN= (1<<0);                                       //      Hab.    interrupción    fin     de conversión canal 0, pero no en el NVIC!!!! (necesario para el DMA)
NVIC_SetPriority(ADC_IRQn,1);                   //

NVIC_EnableIRQ(ADC_IRQn);

}


void init_DMA_canal1(void)
{

//POWER-ON
LPC_SC->PCONP |= 1 << 29;
LPC_GPDMA->DMACConfig = (1<<0); // Hab. DMA

//Origen y destino.
LPC_GPDMACH1->DMACCDestAddr = (uint32_t)buffer;
LPC_GPDMACH1->DMACCSrcAddr = LPC_ADC_BASE+OFFSET_ADC; // ADR0 (0x40003420)

//Periferico de origen: ADC y Transfer type - Desde periferico a memoria (P2M)

//| S.BURST=1  | D.Burst=1  |  S.Width       |    D.Width    |  Incr. Dest | TC interrupt enable.
LPC_GPDMACH1->DMACCControl|=                                                                      (SBURST<<12)|(DBURST<<15)|(WIDTH_VALUE<<18)|(WIDTH_VALUE<<21)| ( 1 << 27 ) | (1U << 31);
LPC_GPDMACH1->DMACCControl|= TAM_BLOCK_DMA;//Tamaño maximo.

//ORIGEN: ADC|   Dest=mem. |    P2M          |    ERROR/TC MASK          | Inicio
LPC_GPDMACH1->DMACCConfig =  ( 4 << 1 ) | (0x00 << 6) | (2 << 11) | (0 << 14) | (1 << 15)| 1;

}

void init_DMA_canal0(void)
{
```

```
LPC_SC->PCONP  |= (1<<29);
                    // Power DMA
LPC_GPDMA->DMACConfig = 1;                          // enable the GPDMA controller
LPC_GPDMA->DMACSync   = (1<<6);                                                  // enable synchro
logic for all reqs

// Linked CH0
LLI0.source      = (uint32_t) &sinusoide[0];
LLI0.destination = (uint32_t) &(LPC_DAC->DACR);
LLI0.next        = (uint32_t) &LLI0;
LLI0.control       = 1<<26 | 2<<21 | 2<<18 | N_samples_wave; //Transfersize=   WAVE_SAMPLE_NUM,
SWidth=32bits, DWidth=32bits, Source Increment

LPC_GPDMACH0->DMACCSrcAddr  = (uint32_t) &sinusoide[0];
LPC_GPDMACH0->DMACCDestAddr = (uint32_t) &(LPC_DAC->DACR);
LPC_GPDMACH0->DMACCLLI      = (uint32_t) &LLI0; // linked lists for ch0
LPC_GPDMACH0->DMACCControl  = N_samples_wave   // transfer size (0 - 11) = N muestras /ciclo
| (0 << 12)          // source burst size (12 - 14) = 1
| (0 << 15)          // destination burst size (15 - 17) = 1
| (2 << 18)          // source width (18 - 20) = 32 bit
| (2 << 21)          // destination width (21 - 23) = 32 bit
| (0 << 24)          // source AHB select (24) = AHB 0
| (0 << 25)          // destination AHB select (25) = AHB 0
| (1 << 26)          // source increment (26) = increment
| (0 << 27)          // destination increment (27) = no increment
| (0 << 28)          // mode select (28) = access in user mode
| (0 << 29)          // (29) = access not bufferable
| (0 << 30)          // (30) = access not cacheable
| (0 << 31);         // terminal count interrupt disabled

LPC_GPDMACH0->DMACCConfig   =  1                                                          //
channel enabled (0)
| (0 << 1)                                                          // source peripheral (1 - 5) = none
| (7 << 6)                                                          // destination peripheral (6 - 10) =
DAC
| (1 << 11)                                                         // flow control (11 - 13) = MEM to
PERF
| (0 << 14)                                                         // (14) = mask out error interrupt
| (0 << 15)                                                         // (15) = mask out terminal count
interrupt
| (0 << 16)                                                         // (16) = no locked transfers
| (0 << 18);                                                  // (27) = no HALT
//F_out (salida del DAC)
LPC_DAC->DACCNTVAL = (F_pclk/N_samples_wave/F_out) -1;  // (Ts DAC = F_out/N_samples  < Tsetup
DAC = 1useg. !!!!)

/* DMA, timer running, dbuff */
LPC_DAC->DACCTRL   = (1<<3) | (1<<2 )| (1<<1);

}

void DMA_IRQHandler(void)
{
```

```
LPC_GPDMA->DMACIntTCClear|= (1<<1); //Borramos interrupcion canal 1;
DMA_completa=1;
}



void ReproducirAudio(void)
{
int i=0;
for(i=0; i<1000; i++)
sinusoide[i]=buffer[i];
LPC_PINCON->PINSEL1|= (2<<20);                                          // enable AOUT
(P0.26) pin
init_DMA_canal0();

delay_mseg(5000);
LPC_PINCON->PINSEL1&= ~(2<<20);                                                 //
disable AOUT (P0.26) pin*/
}
void init_DAC(void)
{
LPC_PINCON->PINSEL1|= (2<<20);          // DAC output = P0.26 (AOUT)
LPC_PINCON->PINMODE1|= (2<<20);         // Deshabilita pullup/pulldown
LPC_SC->PCLKSEL0|= (0x00<<22);          // CCLK/4 (Fpclk después del reset) (100 Mhz/4 = 25Mhz)
LPC_DAC->DACCTRL=0;                                                     //
}
void Reproducirpitido(int a)
{
int i=0;
if(a==1){
for(i=0; i < N_samples_wave; i++)
sinusoide[i] = (uint32_t)(511 + 511*sin(2*PI*i/N_samples_wave))<< 6; // DACR bits 6-15 VALUE (valor ya
desplazado!!!)
}
if(a==2){
for(i=0; i < N_samples_wave; i++)
sinusoide[i] = (uint32_t)(511 + 511*sin(2*PI*i/8*N_samples_wave))<< 6; // DACR bits 6-15 VALUE (valor
ya desplazado!!!)
}
LPC_PINCON->PINSEL1|= (2<<20);                                          // enable AOUT
(P0.26) pin


delay_mseg(5700);
LPC_PINCON->PINSEL1&=~ (2<<20);                                                 //
disable AOUT (P0.26) pin

}



********** HTTP_CGI.c **********
```

```
/*----------------------------------------------------------------------------
*      RL-ARM - TCPnet
*-----------------------------------------------------------------------------
*      Name:    HTTP_CGI.C
*      Purpose: HTTP Server CGI Module
*      Rev.:    V4.22
*-----------------------------------------------------------------------------
*      This code is part of the RealView Run-Time Library.
*      Copyright (c) 2004-2011 KEIL - An ARM Company. All rights reserved.
*----------------------------------------------------------------------------*/

#include <Net_Config.h>
#include <stdio.h>
#include "uart.h"
#include <LPC17xx.h>
#include <stdlib.h>
#include <string.h>
#include "audio.h"
#include "movimentos.h"
#include "motor.h"

/* ---------------------------------------------------------------------------
* The HTTP server provides a small scripting language.
*
* The script language is simple and works as follows. Each script line starts
* with a command character, either "i", "t", "c", "#" or ".".
*   "i" - command tells the script interpreter to "include" a file from the
*         virtual file system and output it to the web browser.
*   "t" - command should be followed by a line of text that is to be output
*         to the browser.
*   "c" - command is used to call one of the C functions from the this file.
*         It may be followed by the line of text. This text is passed to
*         'cgi_func()' as a pointer to environment variable.
*   "#' - command is a comment line and is ignored (the "#" denotes a comment)
*   "." - denotes the last script line.
*
* ---------------------------------------------------------------------------*/

/* http_demo.c */
extern U16 AD_in (U32 ch);
extern U8  get_button (void);

/* at_System.c */
extern  LOCALM localm[];
#define LocM   localm[NETIF_ETH]

/* Net_Config.c */
extern struct tcp_cfg   tcp_config;
extern struct http_cfg  http_config;
#define tcp_NumSocks    tcp_config.NumSocks
#define tcp_socket      tcp_config.Scb
#define http_EnAuth     http_config.EnAuth
#define http_auth_passw http_config.Passw
```

```
extern BOOL LEDrun;
extern void LED_out (U32 val);
extern BOOL LCDupdate;
extern U8   lcd_text[2][16+1];
char bufferhttp2[20];
/* Local variables. */
static U8 P2;
static char const state[][9] = {
"FREE",
"CLOSED",
"LISTEN",
"SYN_REC",
"SYN_SENT",
"FINW1",
"FINW2",
"CLOSING",
"LAST_ACK",
"TWAIT",
"CONNECT"};

/* My structure of CGI status U32 variable. This variable is private for */
/* each HTTP Session and is not altered by HTTP Server. It is only set to  */
/* zero when the cgi_func() is called for the first time.                  */
typedef struct {
U16 xcnt;
U16 unused;
} MY_BUF;
#define MYBUF(p)        ((MY_BUF *)p)

/*---------------------------------------------------------------------------
* HTTP Server Common Gateway Interface Functions
*---------------------------------------------------------------------------*/

/*---------------------------- cgi_process_var -----------------------------*/

void cgi_process_var (U8 *qs) {
/* This function is called by HTTP server to process the Querry_String   */
/* for the CGI Form GET method. It is called on SUBMIT from the browser. */
/*.The Querry_String.is SPACE terminated.                                 */
U8 *var;
int s[4];

var = (U8 *)alloc_mem (40);
do {
/* Loop through all the parameters. */
qs = http_get_var (qs, var, 40);
/* Check the returned string, 'qs' now points to the next. */
if (var[0] != 0) {
/* Returned string is non 0-length. */
if (str_scomp (var, "ip=") == __TRUE) {
/* My IP address parameter. */
sscanf ((const char *)&var[3], "%d.%d.%d.%d",&s[0],&s[1],&s[2],&s[3]);
```

```
LocM.IpAdr[0]   = s[0];
LocM.IpAdr[1]   = s[1];
LocM.IpAdr[2]   = s[2];
LocM.IpAdr[3]   = s[3];
}
else if (str_scomp (var, "msk=") == __TRUE) {
/* Net mask parameter. */
sscanf ((const char *)&var[4], "%d.%d.%d.%d",&s[0],&s[1],&s[2],&s[3]);
LocM.NetMask[0] = s[0];
LocM.NetMask[1] = s[1];
LocM.NetMask[2] = s[2];
LocM.NetMask[3] = s[3];
}
else if (str_scomp (var, "gw=") == __TRUE) {
/* Default gateway parameter. */
sscanf ((const char *)&var[3], "%d.%d.%d.%d",&s[0],&s[1],&s[2],&s[3]);
LocM.DefGW[0]   = s[0];
LocM.DefGW[1]   = s[1];
LocM.DefGW[2]   = s[2];
LocM.DefGW[3]   = s[3];
}
else if (str_scomp (var, "pdns=") == __TRUE) {
/* Default gateway parameter. */
sscanf ((const char *)&var[5], "%d.%d.%d.%d",&s[0],&s[1],&s[2],&s[3]);
LocM.PriDNS[0]  = s[0];
LocM.PriDNS[1]  = s[1];
LocM.PriDNS[2]  = s[2];
LocM.PriDNS[3]  = s[3];
}
else if (str_scomp (var, "sdns=") == __TRUE) {
/* Default gateway parameter. */
sscanf ((const char *)&var[5], "%d.%d.%d.%d",&s[0],&s[1],&s[2],&s[3]);
LocM.SecDNS[0]  = s[0];
LocM.SecDNS[1]  = s[1];
LocM.SecDNS[2]  = s[2];
LocM.SecDNS[3]  = s[3];
}
}
}while (qs);
free_mem ((OS_FRAME *)var);
}


/*--------------------------- cgi_process_data ------------------------------*/

void cgi_process_data (U8 code, U8 *dat, U16 len) {
/* This function is called by HTTP server to process the returned Data    */
/* for the CGI Form POST method. It is called on SUBMIT from the browser. */
/* Parameters:                                                 */
/*   code  - callback context code                             */
/*           0 = www-url-encoded form data                     */
/*           1 = filename for file upload (0-terminated string)*/
/*           2 = file upload raw data                          */
```

```
/*         3 = end of file upload (file close requested)             */
/*         4 = any xml encoded POST data (single or last stream)       */
/*         5 = the same as 4, but with more xml data to follow         */
/*            Use http_get_content_type() to check the content type    */
/*   dat   - pointer to POST received data                             */
/*   len   - received data length                                      */
/*************************************************************************/

U8 passw[12],retyped[12];
U8 *var,stpassw;

switch (code) {
case 0:
/* Url encoded form data received. */
break;

default:
/* Ignore all other codes. */
return;
}

P2 = 0;
LEDrun = __TRUE;
if (len == 0) {
/* No data or all items (radio, checkbox) are off. */
LED_out (P2);
return;
}
stpassw = 0;
var = (U8 *)alloc_mem (40);
do {
/* Parse all returned parameters. */
dat = http_get_var (dat, var, 40);
if (var[0] != 0) {
/* Parameter found, returned string is non 0-length. */
if (str_scomp (var, "led0=on") == __TRUE) {
P2 |= 0x01;
}
else if (str_scomp (var, "led1=on") == __TRUE) {
P2 |= 0x02;
}

else if (str_scomp (var, "ctrl=Browser") == __TRUE) {
LEDrun = __FALSE;
}
else if (str_scomp (var, "pw=") == __TRUE) {
/* Change password. */
str_copy (passw, var+3);
stpassw |= 1;
}
else if (str_scomp (var, "pw2=") == __TRUE) {
/* Retyped password. */
str_copy (retyped, var+4);
```

```
stpassw |= 2;
}
else if (str_scomp (var, "lcd1=") == __TRUE) {
/* LCD Module Line 1 text. */
str_copy (lcd_text[0], var+5);
str_copy (bufferhttp2, lcd_text[0]);
LeerSe(bufferhttp2);
LCDupdate = __TRUE;
}
else if (str_scomp (var, "lcd2=") == __TRUE) {
/* LCD Module Line 2 text. */
str_copy (lcd_text[1], var+5);
LCDupdate = __TRUE;
}
}
}while (dat);
free_mem ((OS_FRAME *)var);
LED_out (P2);

if (stpassw == 0x03) {
len = strlen ((const char *)passw);
if (mem_comp (passw, retyped, len) == __TRUE) {
/* OK, both entered passwords the same, change it. */
str_copy (http_auth_passw, passw);
}
}
}


/*--------------------------- cgi_func ---------------------------------------*/

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
/* This function is called by HTTP server script interpreter to make a    */
/* formated output for 'stdout'. It returns the number of bytes written   */
/* to the output buffer. Hi-bit of return value (len is or-ed with 0x8000)*/
/* is a repeat flag for the system script interpreter. If this bit is set */
/* to 1, the system will call the 'cgi_func()' again for the same script  */
/* line with parameter 'pcgi' pointing to a 4-byte buffer. This buffer    */
/* can be used for storing different status variables for this function.  */
/* It is set to 0 by HTTP Server on first call and is not altered by      */
/* HTTP server for repeated calls. This function should NEVER write more  */
/* than 'buflen' bytes to the buffer.                                     */
/* Parameters:                                                            */
/*   env    - environment variable string                                 */
/*   buf    - HTTP transmit buffer                                        */
/*   buflen - length of this buffer (500-1400 bytes - depends on MSS)     */
/*   pcgi   - pointer to session local buffer used for repeated loops     */
/*            This is a U32 variable - size is 4 bytes. Value is:         */
/*            - on 1st call = 0                                           */
/*            - 2nd call    = as set by this function on first call       */
TCP_INFO *tsoc;
U32 len = 0;
U8 id, *lang;
```

```
static U32 adv, velo, vel2;

switch (env[0]) {
/* Analyze the environment string. It is the script 'c' line starting */
/* at position 2. What you write to the script file is returned here. */
case 'a' :
/* Network parameters - file 'network.cgi' */
switch (env[2]) {
case 'i':
/* Write the local IP address. The format string is included */
/* in environment string of the script line.                 */
len = sprintf((char *)buf,(const char *)&env[4],LocM.IpAdr[0],
LocM.IpAdr[1],LocM.IpAdr[2],LocM.IpAdr[3]);
break;
case 'm':
/* Write local Net mask. */
len = sprintf((char *)buf,(const char *)&env[4],LocM.NetMask[0],
LocM.NetMask[1],LocM.NetMask[2],LocM.NetMask[3]);
break;
case 'g':
/* Write default gateway address. */
len = sprintf((char *)buf,(const char *)&env[4],LocM.DefGW[0],
LocM.DefGW[1],LocM.DefGW[2],LocM.DefGW[3]);
break;
case 'p':
/* Write primary DNS server address. */
len = sprintf((char *)buf,(const char *)&env[4],LocM.PriDNS[0],
LocM.PriDNS[1],LocM.PriDNS[2],LocM.PriDNS[3]);
break;
case 's':
/* Write secondary DNS server address. */
len = sprintf((char *)buf,(const char *)&env[4],LocM.SecDNS[0],
LocM.SecDNS[1],LocM.SecDNS[2],LocM.SecDNS[3]);
break;
}
break;

case 'b':
/* LED control - file 'led.cgi' */
if (env[2] == 'c') {
/* Select Control */
len = sprintf((char *)buf,(const char *)&env[4],LEDrun ? "" : "selected",
LEDrun ? "selected" : "");
break;
}
/* LED CheckBoxes */
id = env[2] - '0';
if (id > 7) {
id = 0;
}
id = 1 << id;
len = sprintf((char *)buf,(const char *)&env[4],(P2 & id) ? "checked" : "");
break;
```

```
case 'c':
/* TCP status - file 'tcp.cgi' */
while ((len + 150) < buflen) {
tsoc = &tcp_socket[MYBUF(pcgi)->xcnt];
MYBUF(pcgi)->xcnt++;
/* 'sprintf' format string is defined here. */
len += sprintf((char *)(buf+len),"<tr align=\"center\">");
if (tsoc->State <= TCP_STATE_CLOSED) {
len += sprintf ((char *)(buf+len),
"<td>%d</td><td>%s</td><td>-</td><td>-</td>"
"<td>-</td><td>-</td></tr>\r\n",
MYBUF(pcgi)->xcnt,state[tsoc->State]);
}
else if (tsoc->State == TCP_STATE_LISTEN) {
len += sprintf ((char *)(buf+len),
"<td>%d</td><td>%s</td><td>-</td><td>-</td>"
"<td>%d</td><td>-</td></tr>\r\n",
MYBUF(pcgi)->xcnt,state[tsoc->State],tsoc->LocPort);
}
else {
len += sprintf ((char *)(buf+len),
"<td>%d</td><td>%s</td><td>%d.%d.%d.%d</td>"
"<td>%d</td><td>%d</td><td>%d</td></tr>\r\n",
MYBUF(pcgi)->xcnt,state[tsoc->State],
tsoc->RemIpAdr[0],tsoc->RemIpAdr[1],
tsoc->RemIpAdr[2],tsoc->RemIpAdr[3],
tsoc->RemPort,tsoc->LocPort,tsoc->AliveTimer);
}
/* Repeat for all TCP Sockets. */
if (MYBUF(pcgi)->xcnt == tcp_NumSocks) {
break;
}
}
if (MYBUF(pcgi)->xcnt < tcp_NumSocks) {
/* Hi bit is a repeat flag. */
len |= 0x8000;
}
break;

case 'd':
/* System password - file 'system.cgi' */
switch (env[2]) {
case '1':
len = sprintf((char *)buf,(const char *)&env[4],
http_EnAuth ? "Enabled" : "Disabled");
break;
case '2':
len = sprintf((char *)buf,(const char *)&env[4],http_auth_passw);
break;
}
break;
```

```
case 'e':
/* Browser Language - file 'language.cgi' */
lang = http_get_lang();
if (strcmp ((const char *)lang, "en") == 0) {
lang = "English";
}
else if (strcmp ((const char *)lang, "en-us") == 0) {
lang = "English USA";
}
else if (strcmp ((const char *)lang, "en-gb") == 0) {
lang = "English GB";
}
else if (strcmp ((const char *)lang, "de") == 0) {
lang = "German";
}
else if (strcmp ((const char *)lang, "de-ch") == 0) {
lang = "German CH";
}
else if (strcmp ((const char *)lang, "de-at") == 0) {
lang = "German AT";
}
else if (strcmp ((const char *)lang, "fr") == 0) {
lang = "French";
}
else if (strcmp ((const char *)lang, "sl") == 0) {
lang = "Slovene";
}
else {
lang = "Unknown";
}
len = sprintf((char *)buf,(const char *)&env[2],lang,http_get_lang());
break;

case 'f':
/* LCD Module control - file 'lcd.cgi' */
switch (env[2]) {
case '1':
len = sprintf((char *)buf,(const char *)&env[4],lcd_text[0]);
break;
case '2':
len = sprintf((char *)buf,(const char *)&env[4],lcd_text[1]);
break;
}
break;

case 'g':
/* AD Input - file 'ad.cgi' */
switch (env[2]) {
case '1':
adv = AD_in (2);
len = sprintf((char *)buf,(const char *)&env[4],adv);
break;
case '2':
```

```
len = sprintf((char *)buf,(const char *)&env[4],(float)adv*3.3/1024);
break;
case '3':
adv = (adv * 100) / 1024;
len = sprintf((char *)buf,(const char *)&env[4],adv);
break;
}
break;
case 'h':
/* AD Input - file 'ad.cgi' */
switch (env[2]) {
case '1':
velo = AD_in1 (1);
len = sprintf((char *)buf,(const char *)&env[4],(float)velo);
break;

case '4':
vel2 = AD_in1 (2);
len = sprintf((char *)buf,(const char *)&env[4],(float)vel2);
break;

}
break;

case 'x':
/* AD Input - xml file 'ad.cgx' */
adv = AD_in (2);
len = sprintf((char *)buf,(const char *)&env[1],adv);
break;
case 'z':
/* AD Input - xml file 'ad.cgx' */

velo= AD_in1 (1);
len = sprintf((char *)buf,(const char *)&env[1],(float)velo);
break;
case 'i':
/* AD Input - xml file 'ad.cgx' */

vel2= AD_in1 (2);
len = sprintf((char *)buf,(const char *)&env[1],(float)vel2);
break;



case 'v':
/* Button state - xml file 'button.cgx' */

switch (env[2])
{
case '1':
len = sprintf((char *)buf,(const char *)&env[4],velo1);
break;
case '2':
len = sprintf((char *)buf,(const char *)&env[4],velo2);
```

```
break;
case '3':
len = sprintf((char *)buf,(const char *)&env[4],ciclo1);
break;
case '4':
len = sprintf((char *)buf,(const char *)&env[4],ciclo2);
break;
case '5':
len = sprintf((char *)buf,(const char *)&env[4],distancia2);
break;
}
break;
case 'y':
/* Button state - xml file 'button.cgx' */
switch (env[2])
{
case '1':
len = sprintf((char *)buf,(const char *)&env[4],velo1);
break;
case '2':
len = sprintf((char *)buf,(const char *)&env[4],velo2);
break;
case '3':
len = sprintf((char *)buf,(const char *)&env[4],ciclo1);
break;
case '4':
len = sprintf((char *)buf,(const char *)&env[4],ciclo2);
break;
case '5':
len = sprintf((char *)buf,(const char *)&env[4],distancia2);
break;
}
break;
return ((U16)len);
}

}
```

********** index.htm **********

```
<head>
<title>Robot Espía</title>
</head>
<body bgColor=#ffffff leftMargin=0 topMargin=10 marginwidth="0" marginheight="0">
<div align=center style="width: 833; height: 470">
<table style="border: 1px solid #000080" height=384 cellSpacing=0 cellPadding=0 width="815">
 <tbody>
 <tr bgColor=#EEEEEE>
  <td style="border-bottom: 1px solid #000080" vAlign=bottom noWrap height=70 margin=50 width="567">
   <h2 align="center"><font face="verdana" color="#006699">SEDA</font></h2>
  </td>
  <td style="border-bottom: 1px solid #000080" vAlign=center noWrap height=73 width="240">
   <a href="http://www.keil.com"><img border="0" src="keil.gif" align="left" width="151" height="47"></a>
 </td>
 </tr>
 <tr>
  <td colSpan=5 height=400 width="805" background="llblue.jpg" style="background-repeat: repeat-y;">
   <div align=center>
   <center>
    <table width="90%" border=0>
     <tbody>
      <tr>
       <td width="95%">
        <h2 align="center"><br>
        <b><font face="verdana" color=#003366>Robot espía</font></b></h2>
        <p align="center">
         <a target="_blank" href="http://www.keil.com/dd/chip/4868.htm">
         <img border="0" src="nxp_logo.gif" width="209" height="62"></a></p>
         <center>
         <table width="797" height="94" border="0" cellpadding="0" cellspacing="0"><tr><td align="center">
         <font face="Verdana"><b> [
          <a href="/network.cgi">Network</a>   |
          <a href="/system.cgi">System</a>     |
          <a href="/lcd.cgi">Secuencia</a>          |
          <a href="/ad.cgi">Sensor</a>           |
          <a href="/buttons.cgi">Velocidades</a>    |
          <a href="/language.cgi">Variables</a> |

  <a href="/tcp.cgi">Statistics</a> ]</b></font>
       </td></tr></table>
      <p><font face="Verdana" size="2">This Web pages are served by the Web server which is part of
       <a href="http://www.keil.com/support/man/docs/rlarm/rlarm_tn_tcpip_prot.htm" target="_blank">
       <b>TCPnet</b></a> in the Real-Time Library.<br>
       Click on the links above to see some status information about the web server
       and the TCP/IP stack.</font></p>
      <p align=center><font face="Verdana" size="2">This example is developed using the
       <b><a href="http://www.keil.com/arm/rvmdkkit.asp" target="_blank"> RealView<sup>®</sup>
       Microcontroller Development Kit</a></b> and the <b><a href="http://www.keil.com/rl-arm/"
```

```
      target="_blank"> Real-Time Library</a></b>.<br>
      For additional information about Keil products, please visit:</font></p>
     <p align=center><a href="http://www.keil.com/" target="_blank"><font face="Verdana" size="4">
      <b>www.keil.com</font></a></b></p>
     <p> </p>
    </center>
   </td>
  </tr></tbody></table></center>
 </div></td></tr>
<tr>
 <td colSpan=5 height=20 width="805" background="llblue.jpg" style="background-repeat: repeat-y;">
  <p align=center><font face="Verdana" size="1"><a
href="http://www.keil.com/company/copyright.htm">
   Copyright</a> © 2004-2013 <a href="http://www.keil.com/company/">KEIL - An ARM Company</a>
   All rights reserved.</font></p>
 </td>
</tr></tbody></table>
</div></body>
</html>
```

********** ad.cgi **********

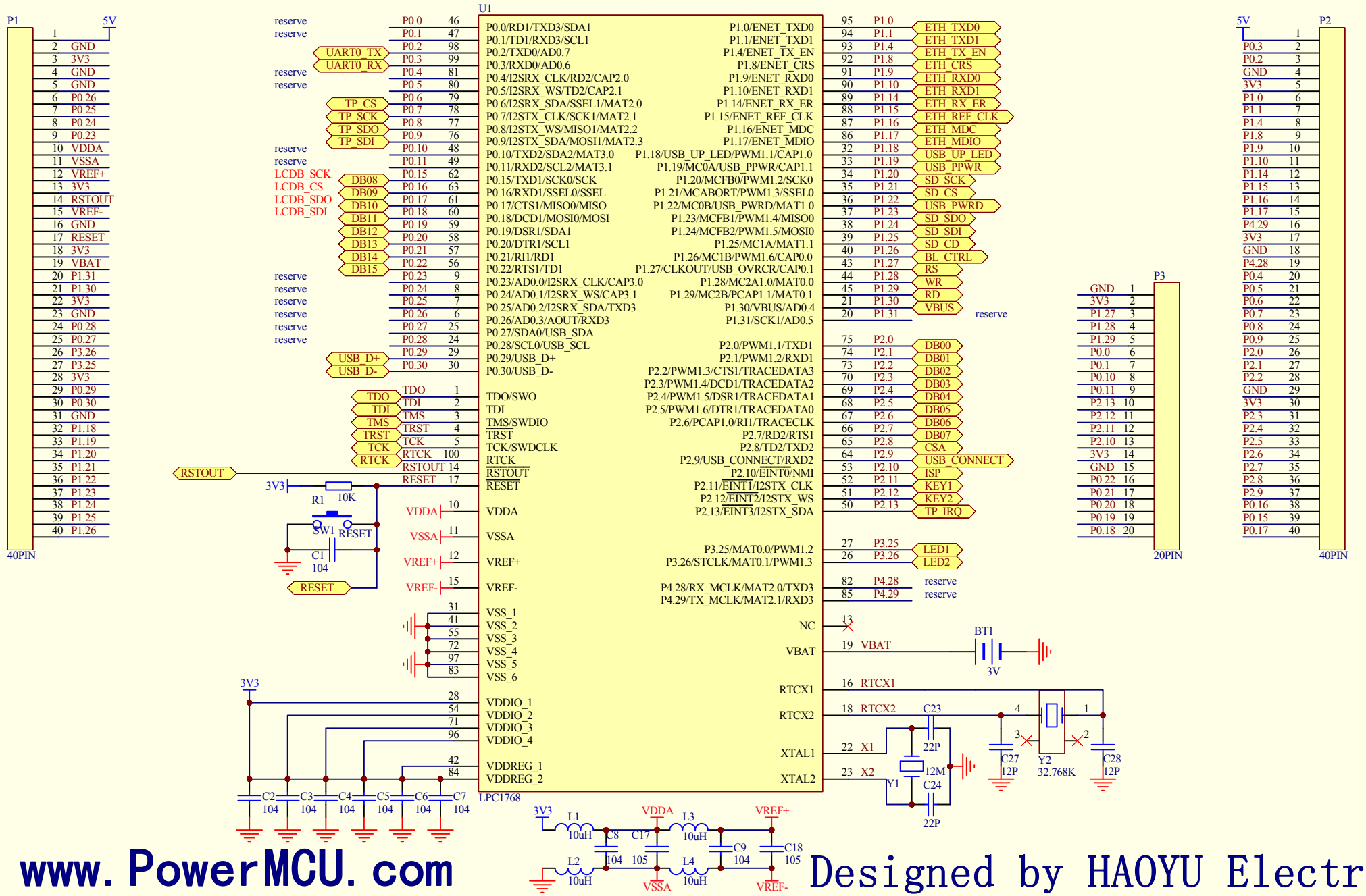
```
t <html><head><title>Visualizar parametros</title>
t <script language=JavaScript type="text/javascript" src="xml_http.js"></script>
t <script language=JavaScript type="text/javascript">
# Define URL and refresh timeout
t var formUpdate = new periodicObj("ad.cgx", 200);
t function plotADGraph() {
t  adVal = document.getElementById("ad_value").value;
t  numVal = parseInt(adVal, 16);
t  voltsVal = (3.3*numVal)/1024;
t  tableSize = (numVal*100/1024);
t  document.getElementById("ad_table").style.width = (tableSize + '%');
t  document.getElementById("ad_volts").value = (voltsVal.toFixed(3) + ' V');
t }
t function periodicUpdateAd() {
t  if(document.getElementById("adChkBox").checked == true) {
t   updateMultiple(formUpdate,plotADGraph);
t   ad_elTime = setTimeout(periodicUpdateAd, formUpdate.period);
t  }
t  else
t   clearTimeout(ad_elTime);
t }
t </script></head>
i pg_header.inc
t <h2 align="center"><br>Visualizar parametros</h2>
t <form action="ad.cgi" method="post" name="ad">
t <input type="hidden" value="ad" name="pg">
t <table border=0 width=99%><font size="3">
t <tr style="background-color: #aaccff">
t  <th width=15%>Elemento</th>
```

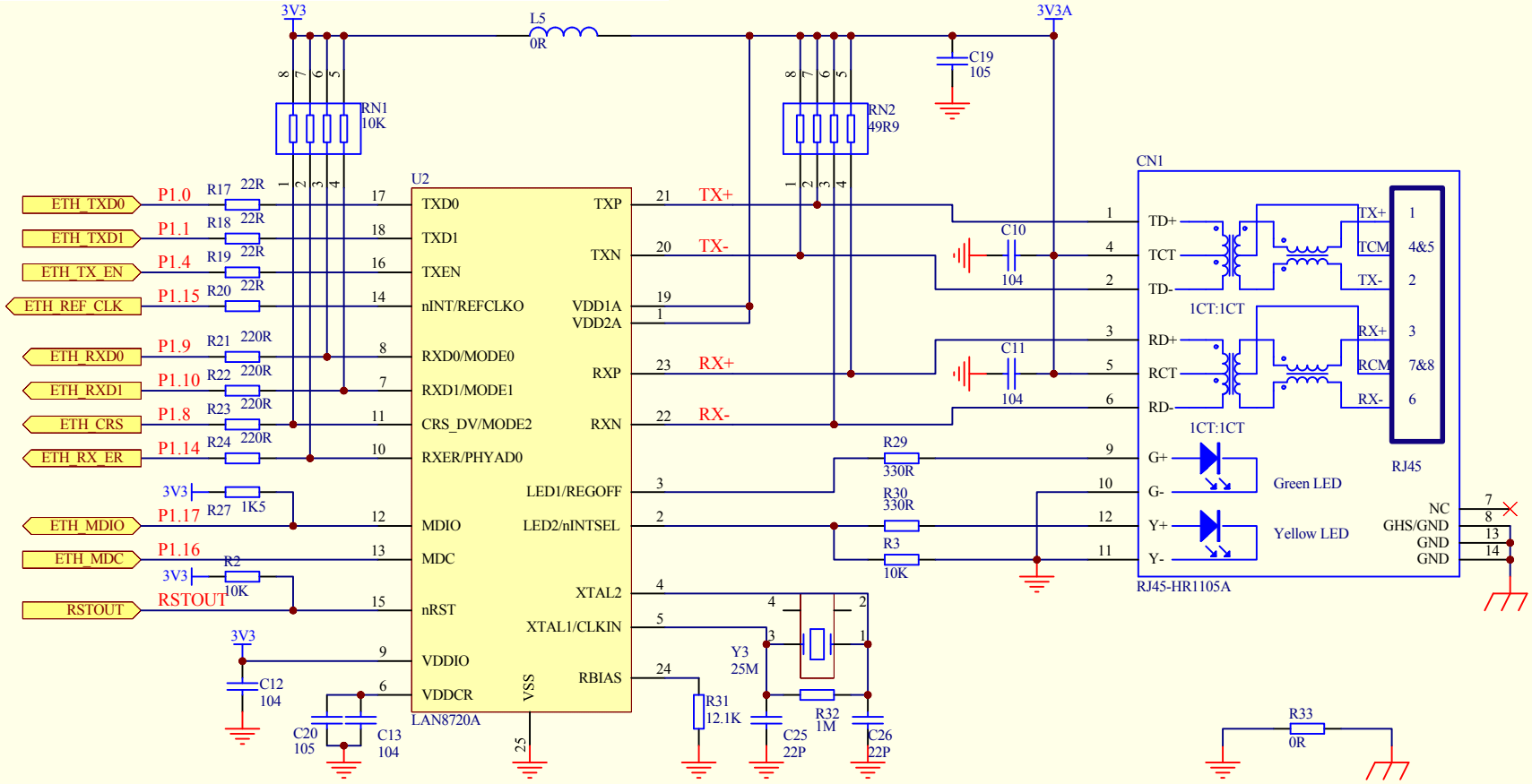
```
t  <th width=15%>Valor</th>
t  <th width=15%>Voltios</th>
t  <th width=55%>Bargraph</th></tr>
t <tr><td><img src="pabb.gif">AD0.2[P0.25]:</td>
t   <td align="center">
t <input type="text" readonly style="background-color: transparent; border: 0px"
c g 1  size="10" id="ad_value" value="0x%03X"></td>
t <td align="center"><input type="text" readonly style="background-color: transparent; border: 0px"
c g 2  size="10" id="ad_volts" value="%5.3f V"></td>
t   <td height=50><table bgcolor="#FFFFFF" border="2" cellpadding="0" cellspacing="0"
width="100%"><tr>
c g 3 <td><table id="ad_table" style="width: %d%%" border="0" cellpadding="0" cellspacing="0">
t <tr><td bgcolor="#0000FF"> </td></tr></table></td></tr></table></td></tr>
t </font></table>
t <p align=center>
t <input type=button value="Refresh" onclick="updateMultiple(formUpdate,plotADGraph)">
t Periodic:<input type="checkbox" id="adChkBox" onclick="periodicUpdateAd()">
t </p></form>
i pg_footer.inc
. End of script must be closed with period

********** ad.cgx ******************************

t <form>
t <text>
t <id>ad_value</id>
c x<value>0x%03X</value>
t </text>
t </form>
.

********** buttons.cgi ******************************

t <html><head><title>Visualizar velocidades</title>
t <script language=JavaScript type="text/javascript" src="xml_http.js"></script>
t <script language=JavaScript type="text/javascript">
# Define URL and refresh timeout
t var formUpdate = new periodicObj("buttons.cgx", 50);
t function plotADGraph() {
t  adVal = document.getElementById("ad_value").value;
t  numVal = parseInt(adVal, 16);
t  voltsVal = (3.3*numVal)/1024;
t  tableSize = (numVal*100/1024);
t  document.getElementById("ad_table").style.width = (tableSize + '%');
t  document.getElementById("ad_volts").value = (voltsVal.toFixed(3) + ' V');
t }
t function periodicUpdateAd() {
t  if(document.getElementById("adChkBox").checked == true) {
t   updateMultiple(formUpdate,plotADGraph);
t   ad_elTime = setTimeout(periodicUpdateAd, formUpdate.period);
t  }
t  else
t   clearTimeout(ad_elTime);
```

```
t }
t </script></head>
i pg_header.inc
t <h2 align="center"><br>Visualizar Velocidades</h2>
t <form action="buttons.cgi" method="post" name="ad">
t <input type="hidden" value="ad" name="pg">
t <table border=0 width=99%><font size="3">
t <tr style="background-color: #aaccff">
t  <th width=50%>Elemento</th>
t  <th width=50%>Valor</th>
t <tr><td><img src="pabb.gif">Velocidad 1:</td>
t   <td align="center">
t <input type="text" readonly style="background-color: transparent; border: 0px"
c h 1  size="10" id="ad_value" value="%f"></td>
t </td></tr>
t <tr><td><img src="pabb.gif">Velocidad 2:</td>
t   <td align="center">
t <input type="text" readonly style="background-color: transparent; border: 0px"
c h 4  size="10" id="ad_value2" value="%f"></td>
t </td></tr>
t </font></table>
t <p align=center>
t <input type=button value="Refresh" onclick="updateMultiple(formUpdate,plotADGraph)">
t Periodic:<input type="checkbox" id="adChkBox" onclick="periodicUpdateAd()">
t </p></form>
i pg_footer.inc
.

**************************** buttons.cgx ********************************

t <form>
t <text>
t <id>ad_value</id>
c z<value>%f</value>
t </text>
t <text>
t <id>ad_value2</id>
c i<value>%f</value>
t </text>
t </form>
.
```

```
************************* language.cgi *************************

t <html><head><title>Ciclo de trabajo</title>
t <script language=JavaScript type="text/javascript" src="xml_http.js"></script>
t <script language=JavaScript type="text/javascript">
# Define URL and refresh timeout
t var formUpdate = new periodicObj("buttons.cgx", 300);
t function periodicUpdate() {
t  if(document.getElementById("refreshChkBox").checked == true) {
t   updateMultiple(formUpdate);
t   periodicFormTime = setTimeout("periodicUpdate()", formUpdate.period);
t  }
t  else
t   clearTimeout(periodicFormTime);
t }
t </script></head>
i pg_header.inc
t <h3 align="center"><br>CICLO DE TRABAJO</h3>
t <form action="buttons.cgi" method="post" id="form1" name="form1">
t <table border="0" width=99%><font size="3">
t <tr>
t  <td><img src="pabb.gif">Ciclo rueda izquierda</td>
t  <td align="center">
t   <input type="text" readonly style="background-color: transparent; border: 1px"
c v 3 size="12" id="ciclo1" value="%f"></td>t  </td>
t  <td><img src="pabb.gif">Ciclo rueda derecha</td>
t  <td align="center">
t   <input type="text" readonly style="background-color: transparent; border: 1px"
c v 4 size="12" id="ciclo2" value="%f"></td>t  </td>
t </tr>
t </font></table>
t <p align="center">
t  <input type="button" id="refreshBtn" value="Refresh" onclick="updateMultiple(formUpdate)">
t  Periodic:<input type="checkbox" id="refreshChkBox" onclick="periodicUpdate()">
t </p></form>
i pg_footer.inc
.

************************ leds.cgi ****************************
t <html><head><title>Control de motores</title>
t <script language=JavaScript>
t function AllSW(st) {
t  for(i=0;i<document.form1.length;i++) {
t   if(document.form1.elements[i].type=="checkbox"){
t    document.form1.elements[i].checked=st;
t   }
t  }
t  document.form1.submit();
t }
t </script></head>
i pg_header.inc
t <h2 align=center><br>Control de motores</h2>
t <p><font size="2">This page shows you how to use the following http form <b>input</b> objects:
```

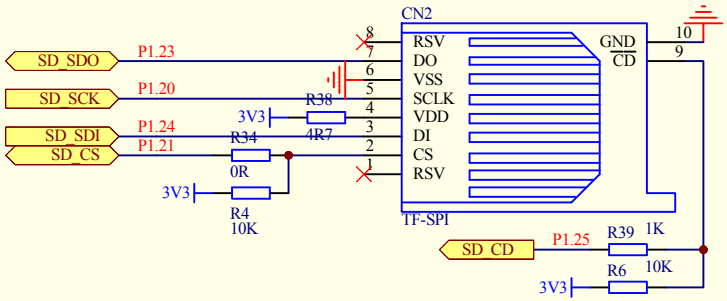
```
t  <b>checkbox</b>, <b>select</b> and <b>button</b>. It uses also a simple <b>Java Script</b>
t  function to check/uncheck all checkboxes and submit the data.<br><br>
t  This Form uses a <b>POST</b> method to send data to a Web server.</font></p>
t <form action=leds.cgi method=post name=form1>
t <input type=hidden value="led" name=pg>
t <table border=0 width=99%><font size="3">
t <tr bgcolor=#aaccff>
t  <th width=40%>Item</th>
t  <th width=60%>Setting</th></tr>
t <tr><td><img src=pabb.gif>LED diode port3 [26-25]:</td>
t <td><table><tr valign="middle">
# Here begin the 'checkbox' definitions
t <td width="5%"></td>
c b 1 <td><input type=checkbox name=led1 OnClick="submit();" %s>PWM2</td>
c b 0 <td><input type=checkbox name=led0 OnClick="submit();" %s>PWM1</td>
t </font></table></td></tr>
t <tr><td><img src=pabb.gif>Aumentar o disminuir PWM1</td>
t <td><input type=button value="  +1   " onclick="AllSW(true)">
t <input type=button value="  -1  " onclick="AllSW(false)"></td></tr>
t <tr><td><img src=pabb.gif>Aumentar o disminuir PWM2</td>
t <td><input type=button value="  +1   " onclick="AllSW(true)">
t <input type=button value="  -1  " onclick="AllSW(false)"></td></tr>
t </table></form>
i pg_footer.inc
.
```


P1
40PIN
P2
40PIN
P3
20PIN
U1
LPC1768
P0.0/RD1/TXD3/SDA1
P0.1/TD1/RXD3/SCL1
P0.2/TXD0/AD0.7
P0.3/RXD0/AD0.6
P0.4/I2SRX_CLK/RD2/CAP2.0
P0.5/I2SRX_WS/TD2/CAP2.1
P0.6/I2SRX_SDA/SSEL1/MAT2.0
P0.7/I2STX_CLK/SCK1/MAT2.1
P0.8/I2STX_WS/MISO1/MAT2.2
P0.9/I2STX_SDA/MOSI1/MAT2.3
P0.10/TXD2/SDA2/MAT3.0
P0.11/RXD2/SCL2/MAT3.1
P0.15/TXD1/SCK0/SCK
P0.16/RXD1/SSEL0/SSEL
P0.17/CTS1/MISO0/MISO
P0.18/DCD1/MOSI0/MOSI
P0.19/DSR1/SDA1
P0.20/DTR1/SCL1
P0.21/RI1/RD1
P0.22/RTS1/TD1
P0.23/AD0.0/I2SRX_CLK/CAP3.0
P0.24/AD0.1/I2SRX_WS/CAP3.1
P0.25/AD0.2/I2SRX_SDA/TXD3
P0.26/AD0.3/AOUT/RXD3
P0.27/SDA0/USB_SDA
P0.28/SCL0/USB_SCL
P0.29/USB_D+
P0.30/USB_D-
TDO/SWO
TDI
TMS/SWDIO
TRST
TCK/SWDCLK
RTCK
RSTOUT
RESET
VDDA
VSSA
VREF+
VREF-
VSS_1
VSS_2
VSS_3
VSS_4
VSS_5
VSS_6
VDDIO_1
VDDIO_2
VDDIO_3
VDDIO_4
VDDREG_1
VDDREG_2
P1.0/ENET_TXD0
P1.1/ENET_TXD1
P1.4/ENET_TX_EN
P1.8/ENET_CRS
P1.9/ENET_RXD0
P1.10/ENET_RXD1
P1.14/ENET_RX_ER
P1.15/ENET_REF_CLK
P1.16/ENET_MDC
P1.17/ENET_MDIO
P1.18/USB_UP_LED/PWM1.1/CAP1.0
P1.19/MC0A/USB_PPWR/CAP1.1
P1.20/MCFB0/PWM1.2/SCK0
P1.21/MCABORT/PWM1.3/SSEL0
P1.22/MC0B/USB_PWRD/MAT1.0
P1.23/MCFB1/PWM1.4/MISO0
P1.24/MCFB2/PWM1.5/MOSI0
P1.25/MC1A/MAT1.1
P1.26/MC1B/PWM1.6/CAP0.0
P1.27/CLKOUT/USB_OVRCR/CAP0.1
P1.28/MC2A1.0/MAT0.0
P1.29/MC2B/PCAP1.1/MAT0.1
P1.30/VBUS/AD0.4
P1.31/SCK1/AD0.5
P2.0/PWM1.1/TXD1
P2.1/PWM1.2/RXD1
P2.2/PWM1.3/CTS1/TRACEDATA3
P2.3/PWM1.4/DCD1/TRACEDATA2
P2.4/PWM1.5/DSR1/TRACEDATA1
P2.5/PWM1.6/DTR1/TRACEDATA0
P2.6/PCAP1.0/RI1/TRACECLK
P2.7/RD2/RTS1
P2.8/TD2/TXD2
P2.9/USB_CONNECT/RXD2
P2.10/EINT0/NMI
P2.11/EINT1/I2STX_CLK
P2.12/EINT2/I2STX_WS
P2.13/EINT3/I2STX_SDA
P3.25/MAT0.0/PWM1.2
P3.26/STCLK/MAT0.1/PWM1.3
P4.28/RX_MCLK/MAT2.0/TXD3
P4.29/TX_MCLK/MAT2.1/RXD3
NC
VBAT
RTCX1
RTCX2
XTAL1
XTAL2
UART0_TX
UART0_RX
TP_CS
TP_SCK
TP_SDO
TP_SDI
LCDB_SCK
LCDB_CS
LCDB_SDO
LCDB_SDI
DB08
DB09
DB10
DB11
DB12
DB13
DB14
DB15
ETH_TXD0
ETH_TXD1
ETH_TX_EN
ETH_CRS
ETH_RXD0
ETH_RXD1
ETH_RX_ER
ETH_REF_CLK
ETH_MDC
ETH_MDIO
USB_UP_LED
USB_PPWR
SD_SCK
SD_CS
USB_PWRD
SD_SDO
SD_SDI
SD_CD
BL_CTRL
RS
WR
RD
VBUS
DB00
DB01
DB02
DB03
DB04
DB05
DB06
DB07
CSA
USB_CONNECT
ISP
KEY1
KEY2
TP_IRQ
LED1
LED2
reserve
RSTOUT
R1
10K
SW1
RESET
C1
104
BT1
3V
C23
22P
C24
22P
12M
Y1
Y2
32.768K
C27
12P
C28
12P
C2
C3
C4
C5
C6
C7
104
L1
L2
L3
L4
10uH
C8
C17
C9
C18
104
105
3V3
VDDA
VSSA
VREF+
VREF-
www.PowerMCU.com
Designed by HAOYU Electroni

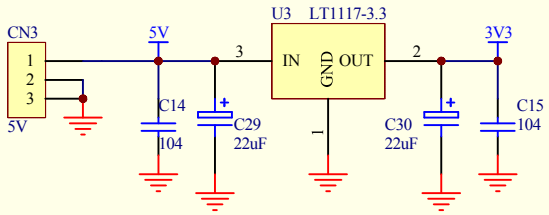


Designed by HAOYU Electronics

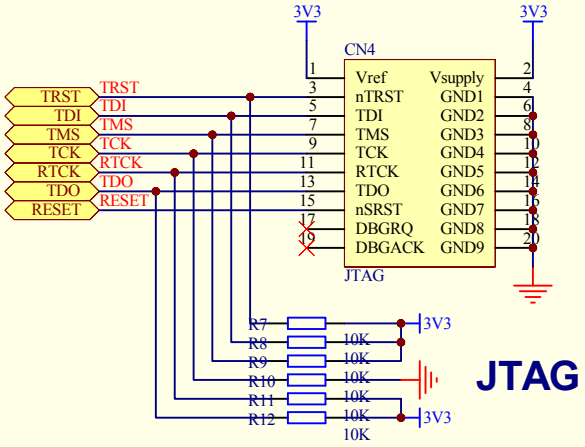
www.PowerMCU.com



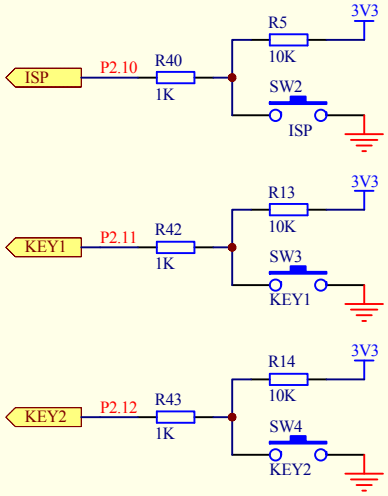
SD_CARD



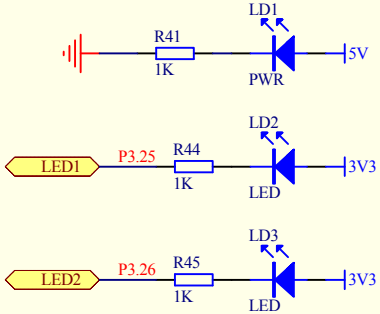
Power



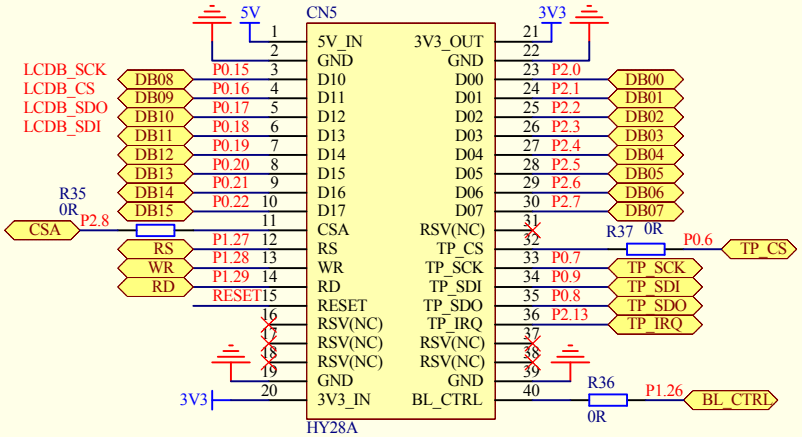
JTAG



USER_KEY



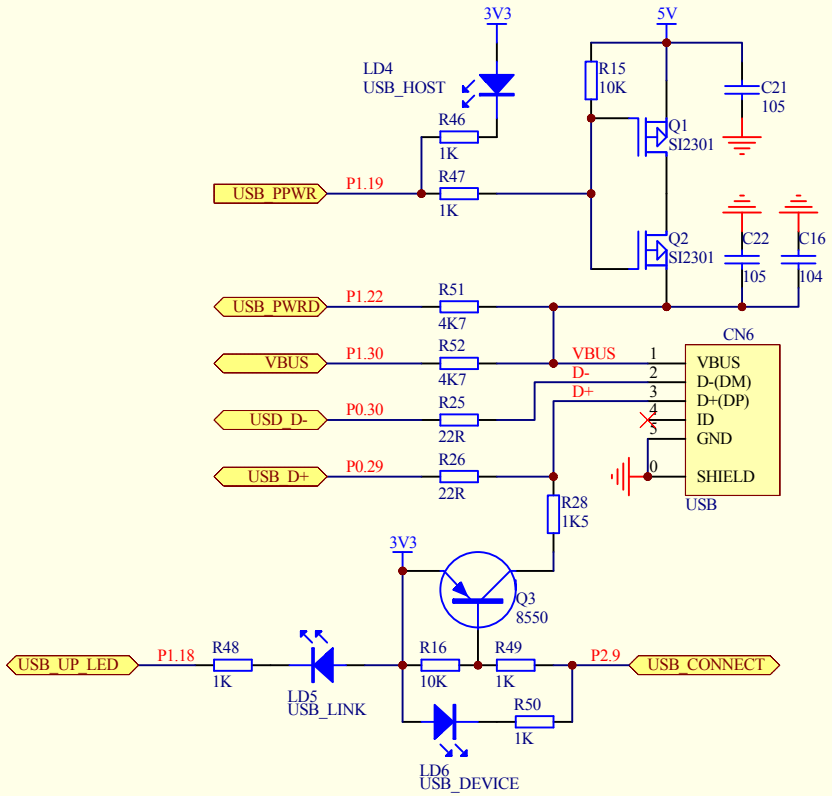
LEDs



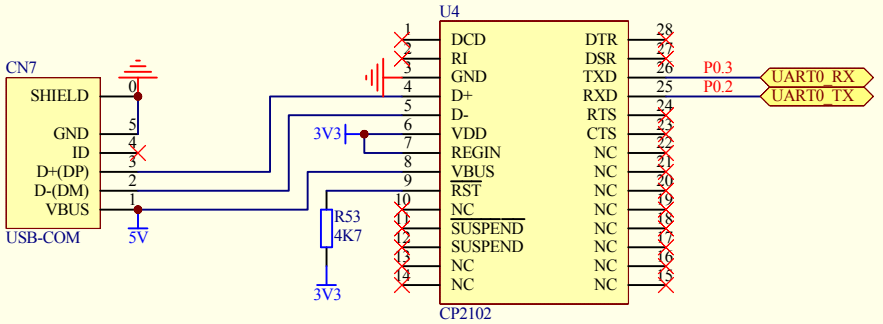
LCD

Designed by HAOYU Electronics

www.PowerMCU.com



USB Host/Device



USB to UART

Designed by HAOYU Electronics

www.PowerMCU.com

ZX-NUNCHUK (#8000339)

Wii-Nunchuk interface board

1. Features

- Interface with Wii Nunchuk remote control directly. Modification the remote control is not required.
- Two-wire Interface. Easy to interface with any modern microcontroller
- Connect to INEX microcontroller board via 3-pin JST connector.
- +5Vdc supply voltage. On-board +3.3V regulator.
- 2x4 cm. size.

2. Board contents :

- ZX-NUNCHUK board x 1
- JST3AA-8 cable x 2
- Documentation x 1

The Wii-Nunchuk remote control is optional. This is also sold separated. Controller is provided in the **Nunchuk Interface kit (#8000347).**

3. General information

There is 3-axis accellerometer sensor *LIS3L02AL* of ST Microelectronics inside the Wii-Nunchuk. It includes a 10-bit A/D converter, an analog joystick and 2-button switches. The figure 1 shows the components of Wii-Nunchuk and connectorís pin assignment.

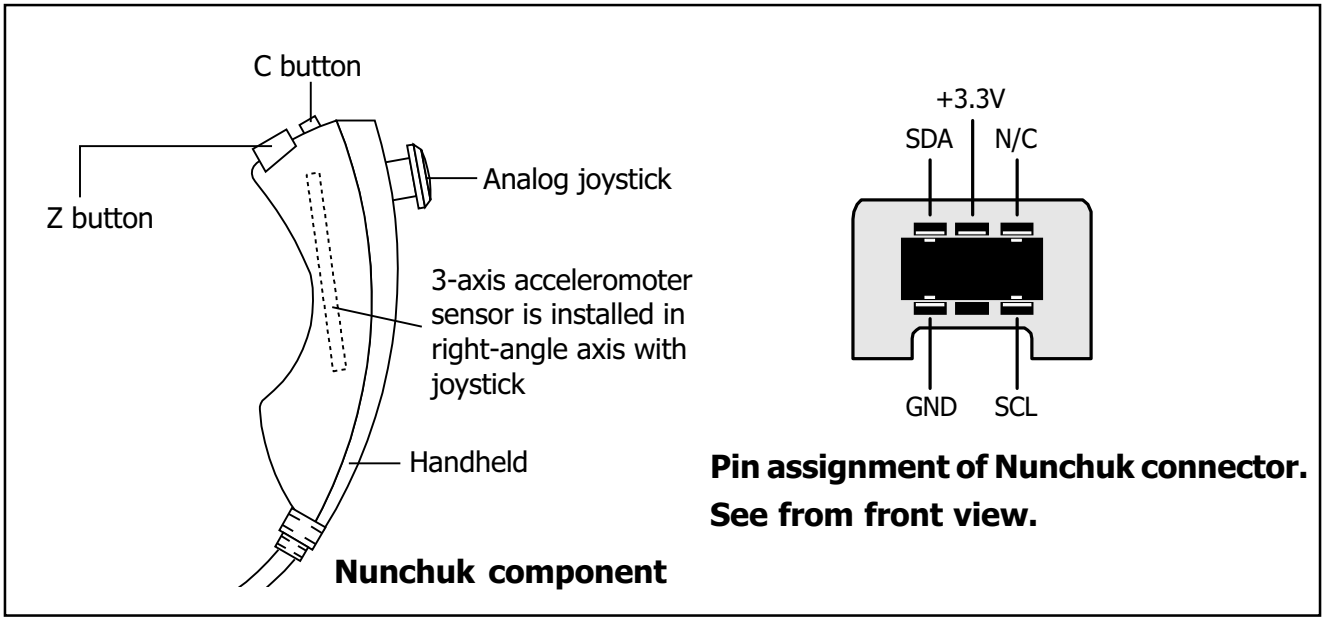


Figure 1 : Wii-Nunchuk remote control information

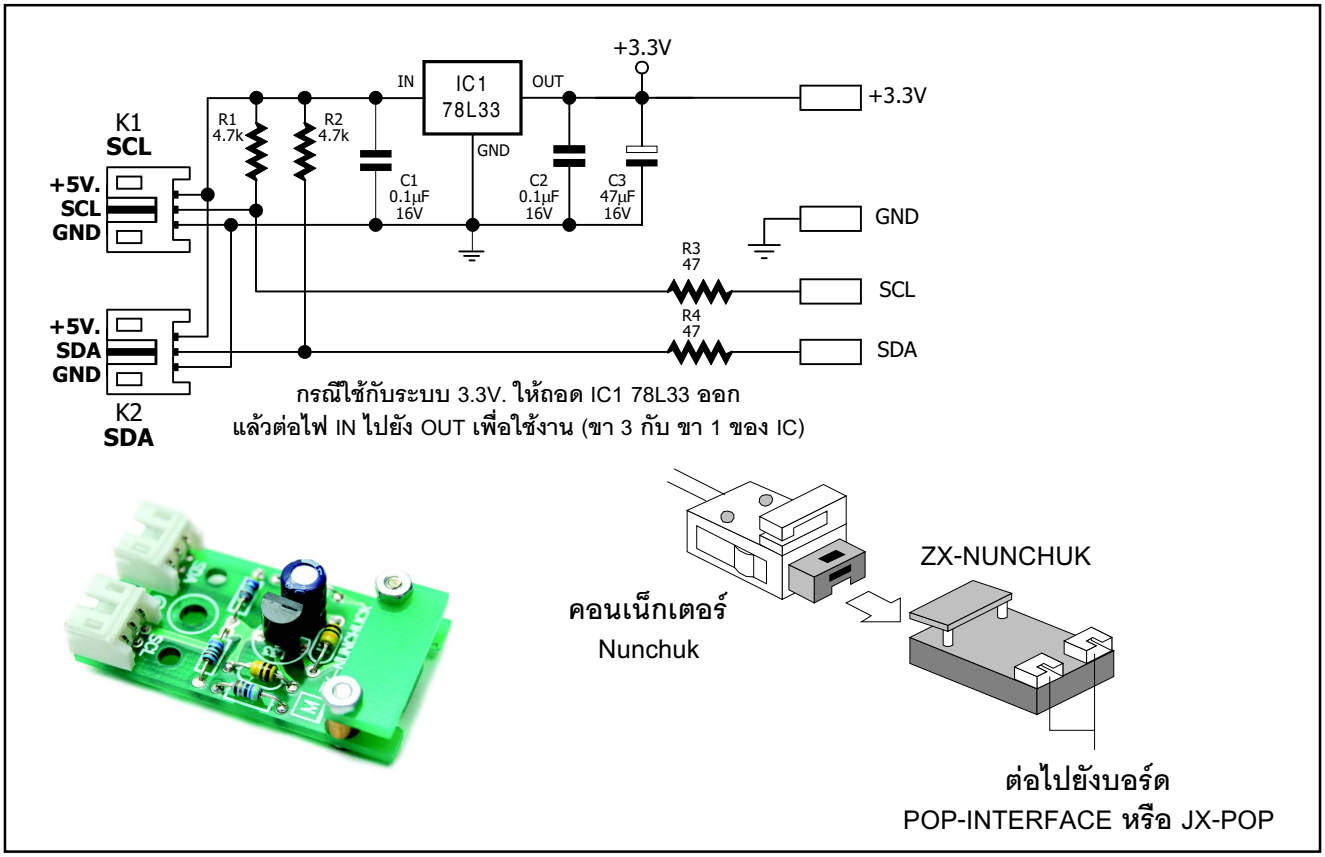


Figure 2 : ZX-NUNCHUK scematic diagram and shows the connecting with Wii-Nunchuk

The wiring signal of Wii-Nunchuck has 4 lines. It includes +3.3V supply voltage (red line), Ground (white line), Serial clock line - SCL (yellow line) and Serail data line - SDA (green line). They are connected to special female connector and assigned pin location are shown in the figure 1.

The interfacing protocol is 2-wire interface or I^2C bus compatible. The logic level is +3.3V. The ZX-NUNCHUK is designed for bridging between any modern microcontroller and Wii-Nunchuk remote control. The 3-axis accellerometer sensor can detect accelleration ± 2g maximum. The schematic diagram of ZX-NUNCHUK and how to interface with the Wii-Nunchuk remote control are shown in figure 2.

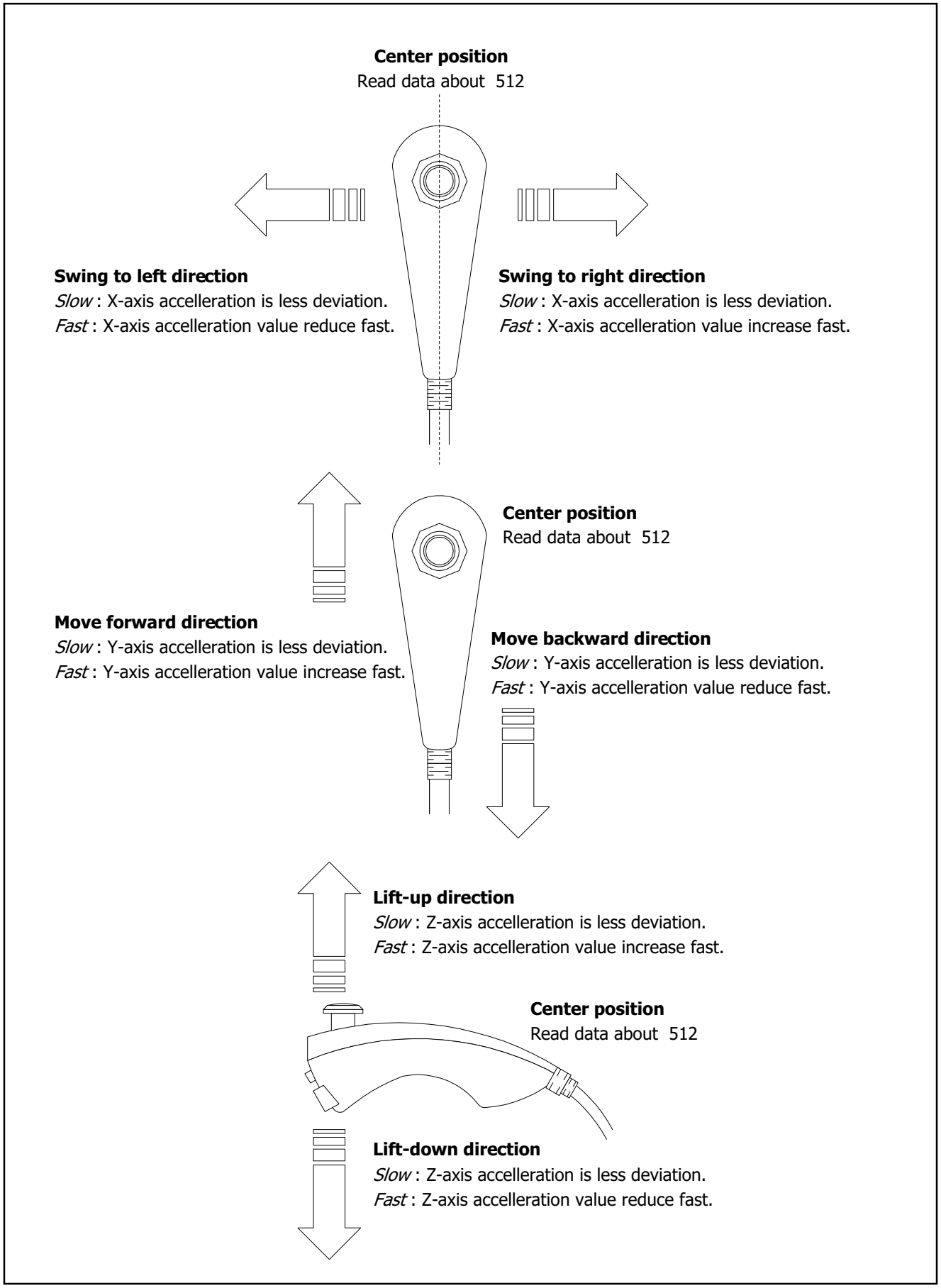


Figure 3 : Wii-Nunchuk physical operation

4. Protocol

Wii-Nunchuck is a slave I²C bus device. It has 2 slave ID for writing (0xA4) and reading (0xA5) data which is as follows :

1	0	1	0	0	1	0	R/$\overline{W}$

The reading data of the Wii-Nunchuk consists of 6 bytes data. Before using these data, programmer must decode these data following :

***Exact data* = (Redaing data *XOR* 0x17) + 0x17**

The summary of exact data after decoding can show as follows :

Data byte receive								**Address**
Joystick X								**0x00**
Joystick Y								**0x01**
Accelerometer X (bit 9 to bit 2 for 10-bit resolution)								**0x02**
Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)								**0x03**
Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)								**0x04**
Accel. Z bit 1	**Accel. Z bit 0**	**Accel. Y bit 1**	**Accel. Y bit 0**	**Accel. X bit 1**	**Accel. X bit 0**	**C-button**	**Z-button**	**0x05**

Byte 0x00 : X-axis data of the joystick

Byte 0x01 : Y-axis data of the joystick

Byte 0x02 : X-axis data of the accellerometer sensor

Byte 0x03 : Y-axis data of the accellerometer sensor

Byte 0x04 : Z-axis data of the accellerometer sensor

Byte 0x05 : bit 0 as Z button status - 0 = pressed and 1 = release

bit 1 as C button status - 0 = pressed and 1 = release

bit 2 and 3 as 2 lower bit of X-axis data of the accellerometer sensor

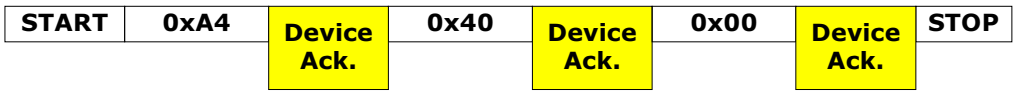
bit 4 and 5 as 2 lower bit of Y-axis data of the accellerometer sensor

bit 6 and 7 as 2 lower bit of Z-axis data of the accellerometer sensor

5. Programming

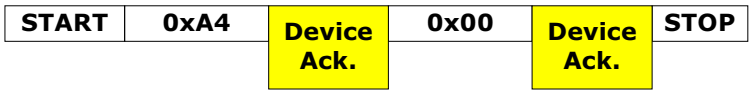
5.1 Initialize start Nunchuk command

Set the Nunchuk as ready after power-on. Write the command 0x40 and 0x00 follows the slave ID byte 0xA4. Normally this command is written at once.



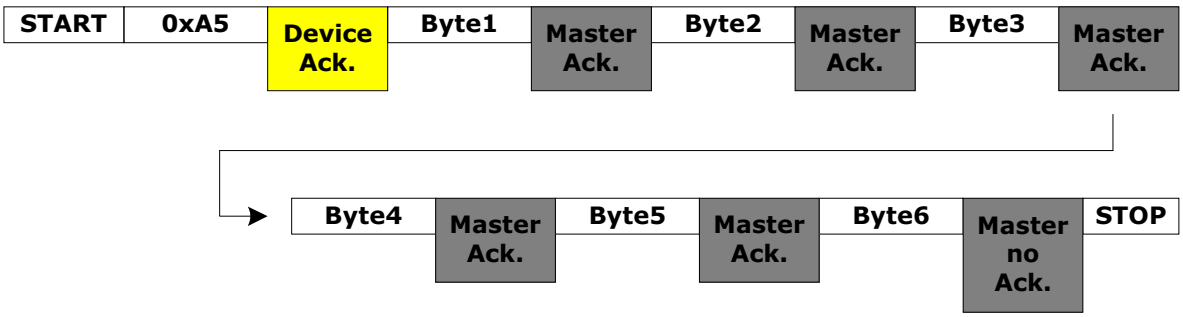
5.2 Conversion command (0x00)

Send this command to get all sensor data and store into the 6-byte register within Nunchuk controller. This must be execute before reading data from the Nunchuk.



5.3 Data read command

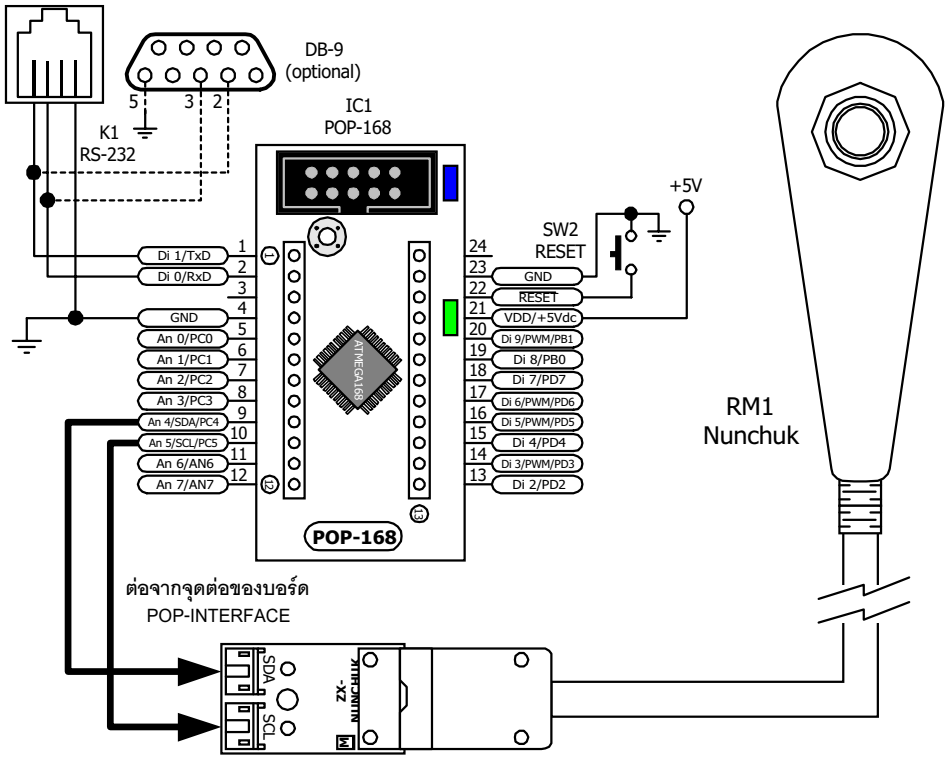
Send the slave ID for reading (0xA5) and wait for the stream data 6-byte from the Nunchuk.



Next, **decode the data to exact data by XOR with 0x17 and plus with 0x17. This step is very important. Do not skip this step !**

6. Example : Interface Nunchuk with Arduino POP-168

This example show the data reading from Wii-Nunchuk by Arduino POP-168 microcontroller module. The POP-168 is fit on the POP-Interface baord or Project board (JX-POP168). Connect with Wii-Nunchuk via ZX-NUNCHUK interface board. The reading data will be shown on the Arduino IDEís Serial monitor. The construction circuit is shown below. The example C code shows in the Listing 1.




```
/*****************************************************************************
 * Code for read data from Wii nunchuck., base on http://www.windmeadow.com/node/42
 * File : ArduinoNunchuk.pde
 * by   : K.Worapoht
 * Hardware: Arduino, POP-168 , AVR ATMega168 ,
 * connect SDA  to PC4 (An4) and SCL  PC5 (An5)
 *****************************************************************************/
#include <Wire.h>

#define nunchuk_ID 0xA4 >> 1

unsigned char buffer[6];// array to store arduino output
int cnt = 0;

void setup ()
{
  Serial.begin (9600);
  Wire.begin ();    // join i2c bus with address 0x52
  nunchuck_init (); // send the initilization handshake
  delay (100);
}
```

Listing 1 : Arduino C code for interfaing Wii-Nunchuk with Arduino POP-168 (continue)

```
void nunchuck_init ()
{
   Wire.beginTransmission (nunchuk_ID);// transmit to device 0x52
   Wire.send (0x40);    // sends memory address
   Wire.send (0x00);    // sends sent a zero.
   Wire.endTransmission ();   // stop transmitting
}

void send_zero ()
{
   Wire.beginTransmission (nunchuk_ID);// transmit to device 0x52
   Wire.send (0x00);    // sends one byte
   Wire.endTransmission ();   // stop transmitting
}

void loop ()
{
   Wire.requestFrom (nunchuk_ID, 6);// request data from nunchuck
   while (Wire.available ())
   {
        buffer[cnt] = nunchuk_decode_byte (Wire.receive ()); // receive byte as an
integer
       cnt++;
   }
   // If we recieved the 6 bytes, then go print them
   if (cnt >= 5)
   {
       print ();
   }

   cnt = 0;
   send_zero (); // send the request for next bytes
   delay (100);
}

// Print the input data we have recieved
// accel data is 10 bits long
// so we read 8 bits, then we have to add
// on the last 2 bits.
void print ()
{
   unsigned char joy_x_axis;
   unsigned char joy_y_axis;
   int accel_x_axis;
   int accel_y_axis;
   int accel_z_axis;
   unsigned char z_button;
   unsigned char c_button;

   joy_x_axis = buffer[0];
   joy_y_axis = buffer[1];
   accel_x_axis = (buffer[2]) << 2;
   accel_y_axis = (buffer[3]) << 2;
   accel_z_axis = (buffer[4]) << 2;

  // byte outbuf[5] contains bits for z and c buttons
  // it also contains the least significant bits for the accelerometer data
  // so we have to check each bit of byte outbuf[5]
   if ((buffer[5] & 0x01)!=0)
   {   z_button = 1;  }
   else
   {   z_button = 0;  }

   if ((buffer[5] & 0x02)!=0)
```

Listing 1 : Arduino C code for interfaing Wii-Nunchuk with Arduino POP-168 (continue)

```
  {   c_button = 1;  }
  else
  {   c_button = 0;  }
  accel_x_axis += ((buffer[5]) >> 2) & 0x03;
  accel_y_axis += ((buffer[5]) >> 4) & 0x03;
  accel_z_axis += ((buffer[5]) >> 6) & 0x03;

  Serial.print (joy_x_axis, DEC);
  Serial.print ("\t");

  Serial.print (joy_y_axis, DEC);
  Serial.print ("\t");

  Serial.print (accel_x_axis, DEC);
  Serial.print ("\t");

  Serial.print (accel_y_axis, DEC);
  Serial.print ("\t");

  Serial.print (accel_z_axis, DEC);
  Serial.print ("\t");

  Serial.print (z_button, DEC);
  Serial.print ("\t");

  Serial.print (c_button, DEC);

  Serial.print ("\r\n");
}

// Encode data to format that most wiimote drivers except
// only needed if you use one of the regular wiimote drivers
char nunchuk_decode_byte (char x)
{
  x = (x ^ 0x17) + 0x17;
  return x;
}
```

Listing 1 : Arduino C code for interfaing Wii-Nunchuk with Arduino POP-168 (final)

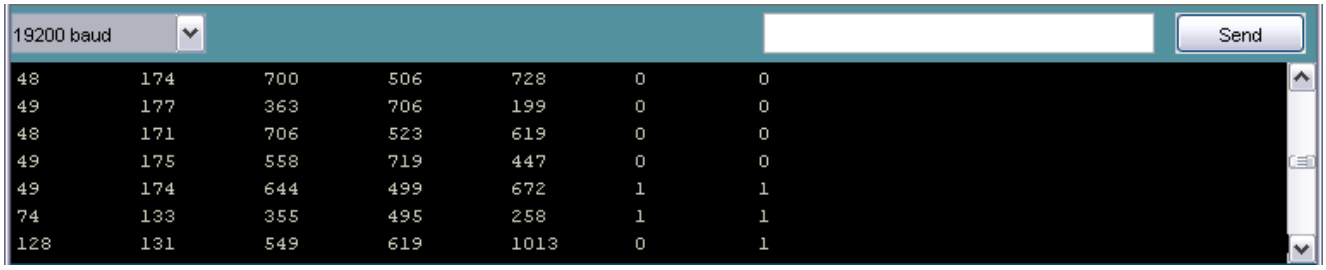


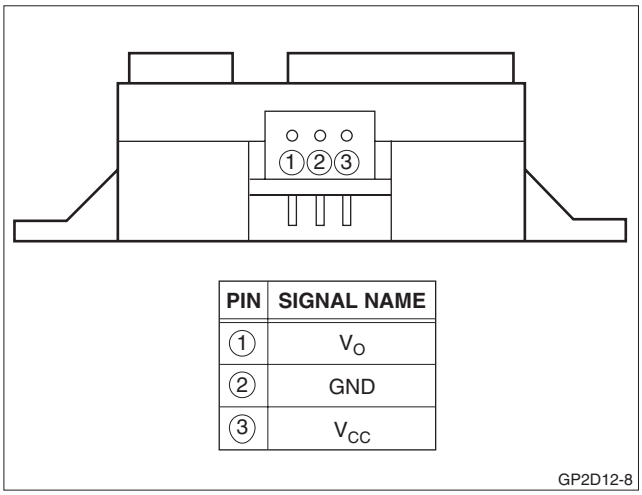
Figure 4 : Shows the reading data from Wii-Ninchuk remote control on the Arduino's serial monitor. First 2 bytes are X and Y axis data of Joystick. Next 3 bytes are X, Y and Z axis of accellerometer sensor and last 2 bytes are C and Z button status. If any button is pressed, the data will equal "0".

FEATURES

- Analog output
- Effective Range: 10 to 80 cm
- LED pulse cycle duration: 32 ms
- Typical response time: 39 ms
- Typical start up delay: 44 ms
- Average current consumption: 33 mA
- Detection area diameter @ 80 cm: 6 cm

DESCRIPTION

The GP2D12 is a distance measuring sensor with integrated signal processing and analog voltage output.



PIN	SIGNAL NAME
①	V_O
②	GND
③	V_{CC}

Figure 1. Pinout

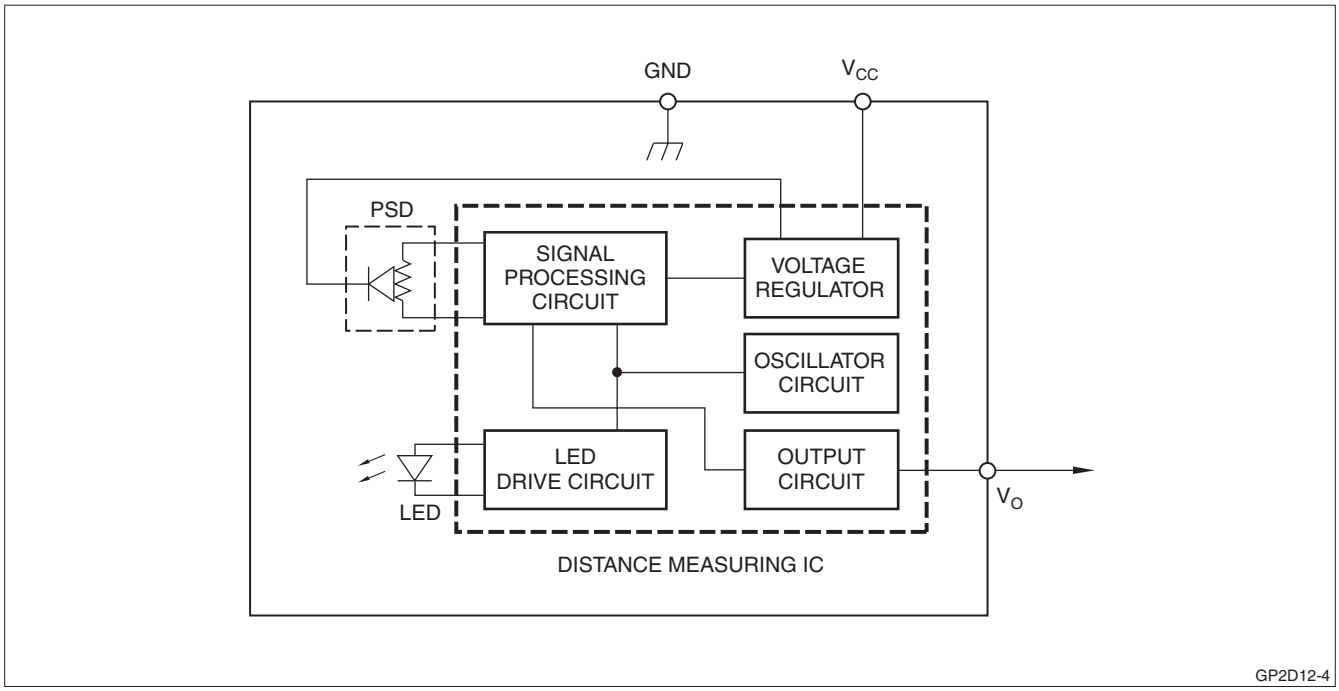


Figure 2. Block Diagram

ELECTRICAL SPECIFICATIONS

Absolute Maximum Ratings

Ta = 25°C, V_{CC} = 5 VDC

PARAMETER	SYMBOL	RATING	UNIT
Supply Voltage	V_{CC}	-0.3 to +7.0	V
Output Terminal Voltage	V_O	-0.3 to (V_{CC} + 0.3)	V
Operating Temperature	Topr	-10 to +60	°C
Storage Temperature	Tstg	-40 to +70	°C

Operating Supply Voltage

PARAMETER	SYMBOL	RATING	UNIT
Operating Supply Voltage	V_{CC}	4.5 to 5.5	V

Electro-optical Characteristics

Ta = 25°C, V_{CC} = 5 VDC

PARAMETER	SYMBOL	CONDITIONS	MIN.	TYP.	MAX.	UNIT	NOTES
Measuring Distance Range	ΔL		10	-	80	cm	1, 2
Output Voltage	V_O	L = 80 cm	0.25	0.4	0.55	V	1, 2
Output Voltage Difference	ΔV_O	Output change at L change (80 cm - 10 cm)	1.75	2.0	2.25	V	1, 2
Average Supply Current	I_{CC}	L = 80 cm	-	33	50	mA	1, 2

NOTES:

1. Measurements made with Kodak R-27 Gray Card, using the white side, (90% reflectivity).
2. L = Distance to reflective object.

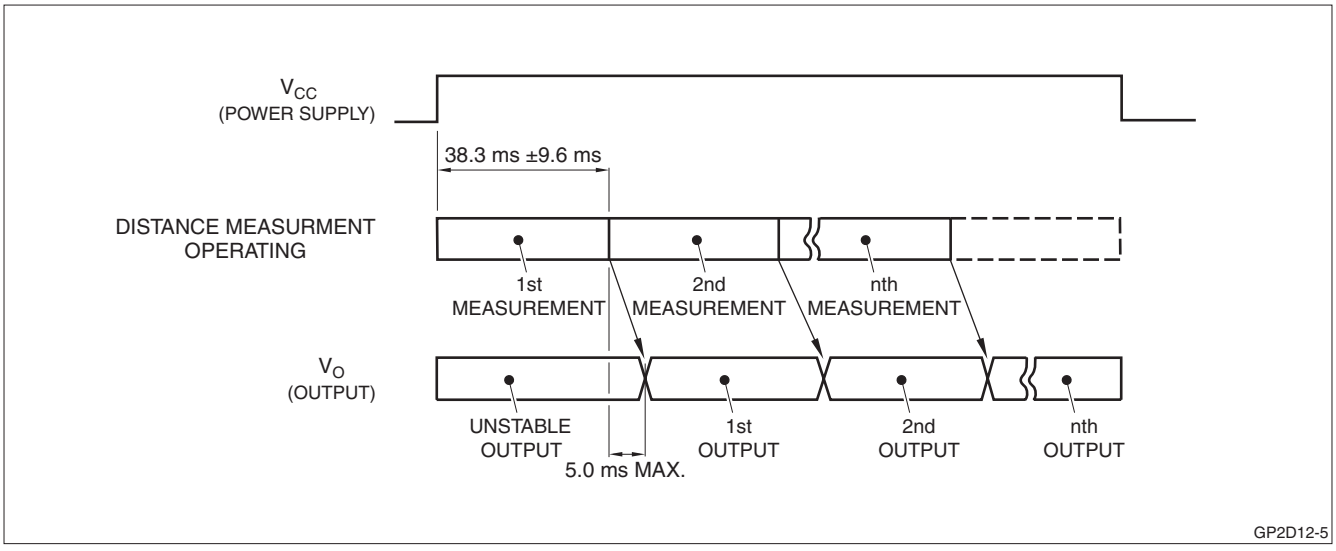


Figure 3. Timing Diagram

RELIABILITY

The reliability of requirements of this device are listed in Table 1.

Table 1. Reliability

TEST ITEMS	TEST CONDITIONS	FAILURE JUDGEMENT CRITERIA	SAMPLES (n), DEFECTIVE (C)
Temperature Cycling	One cycle -40°C (30 min.) to +70°C in 30 minutes, repeated 25 times	Initial × 0.8 > V_O V_O > Initial × 1.2	n = 11, C = 0
High Temperature and High Humidity Storage	+40°C, 90% RH, 500h		n = 11, C = 0
High Temperature Storage	+70°C, 500h		n = 11, C = 0
Low Temperature Storage	-40°C, 500h		n = 11, C = 0
Operational Life (High Temperature)	+60°C, V_{CC} = 5 V, 500h		n = 11, C = 0
Mechanical Shock	100 m/s², 6.0 ms 3 times/±X, ±Y, ±Z direction		n = 6, C = 0
Variable Frequency Vibration	10-to-55-to-10 Hz in 1 minute Amplitude: 1.5 mm 2h in each X, Y, Z direction		n = 6, C = 0

NOTES:
1. Test conditions are according to Electro-optical Characteristics, shown on page 2.
2. At completion of the test, allow device to remain at nominal room temperature and humidity (non-condensing) for two hours.
3. Confidence level: 90%, Lot Tolerance Percent Defect (LTPD): 20%/40%.

MANUFACTURER'S INSPECTION

Inspection Lot

Inspection shall be carried out per each delivery lot.

Inspection Method

A single sampling plan, normal inspection level II based on ISO 2859 shall be adopted.

Table 2. Quality Level

DEFECT	INSPECTION ITEM and TEST METHOD	AQL (%)
Major Defect	Electro-optical characteristics defect	0.4
Minor Defect	Defect to appearance or dimensions (crack, split, chip, scratch, stain)*	1.0

NOTE: *Any one of these that affects the Electro-optical Characteristics shall be considered a defect.

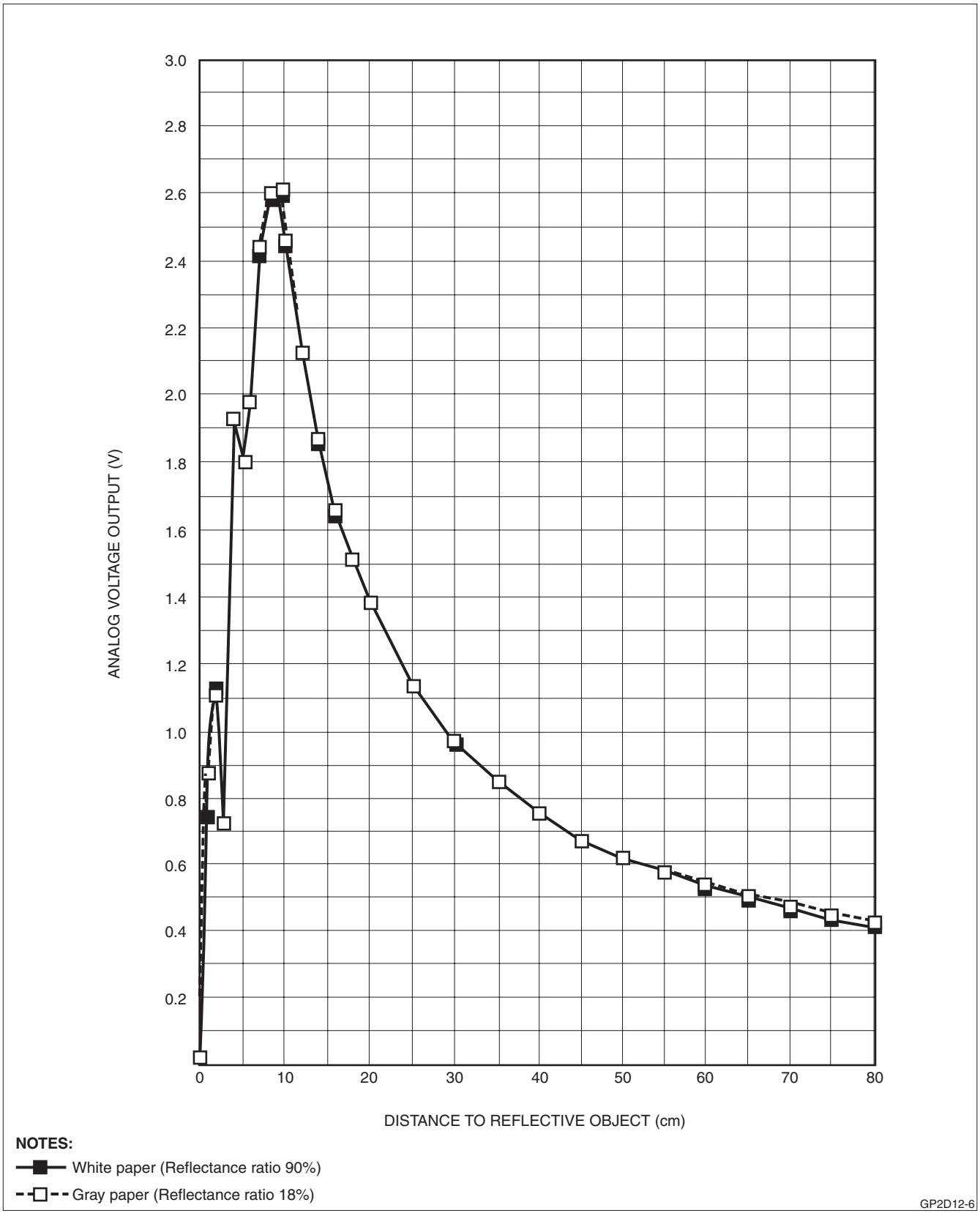


Figure 4. GP2D12 Example of Output/Distance Characteristics

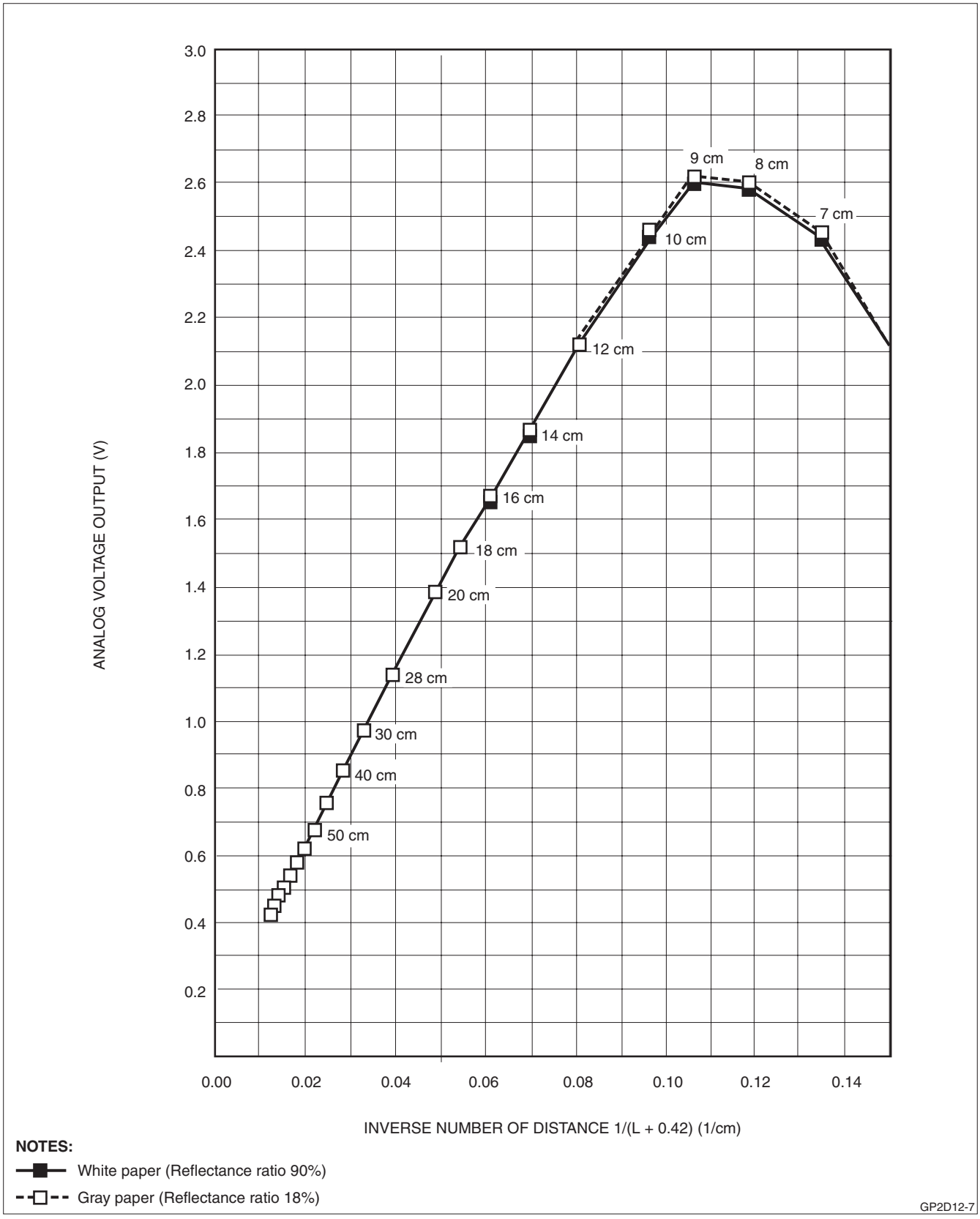
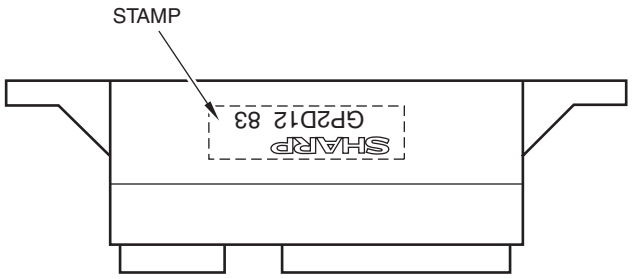


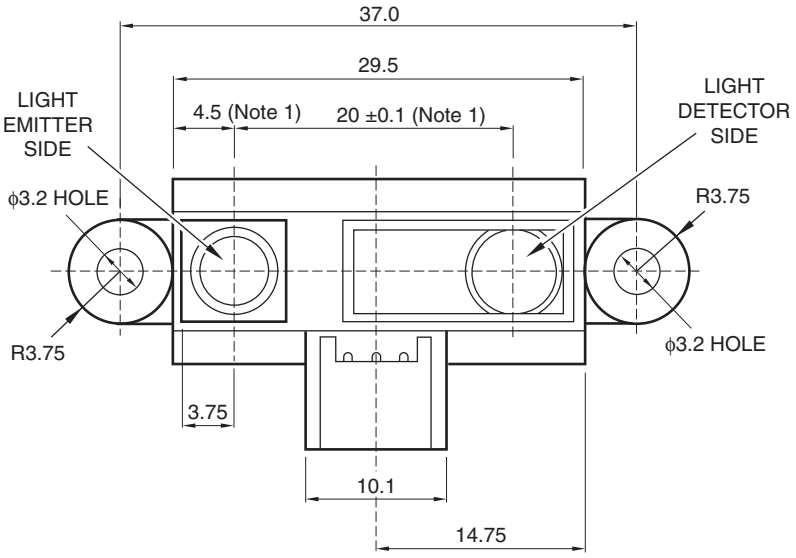
Figure 5. GP2D12 Example of Output Characteristics with Inverse Number of Distance

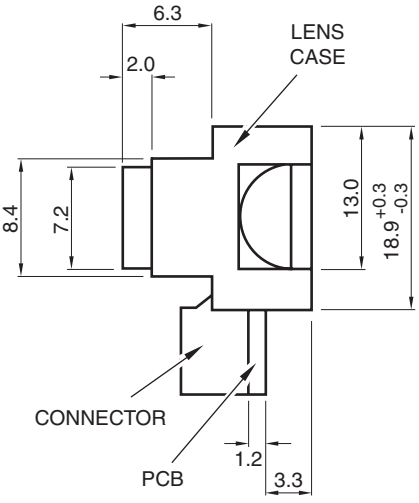
PACKAGE SPECIFICATIONS

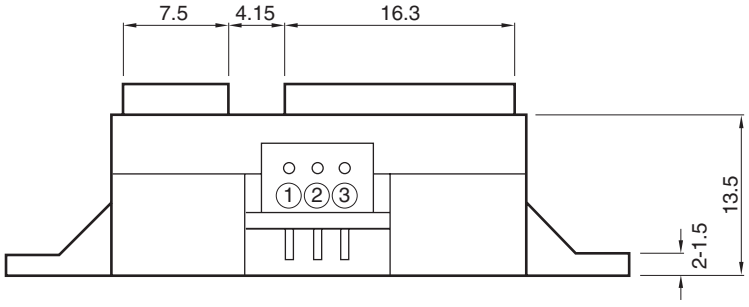


STAMP EXAMPLE

GP2D12 5 3
- Month (1 to 9, X, Y, Z)
- Year (2005:5)
- Model Name







CONNECTOR SIGNAL

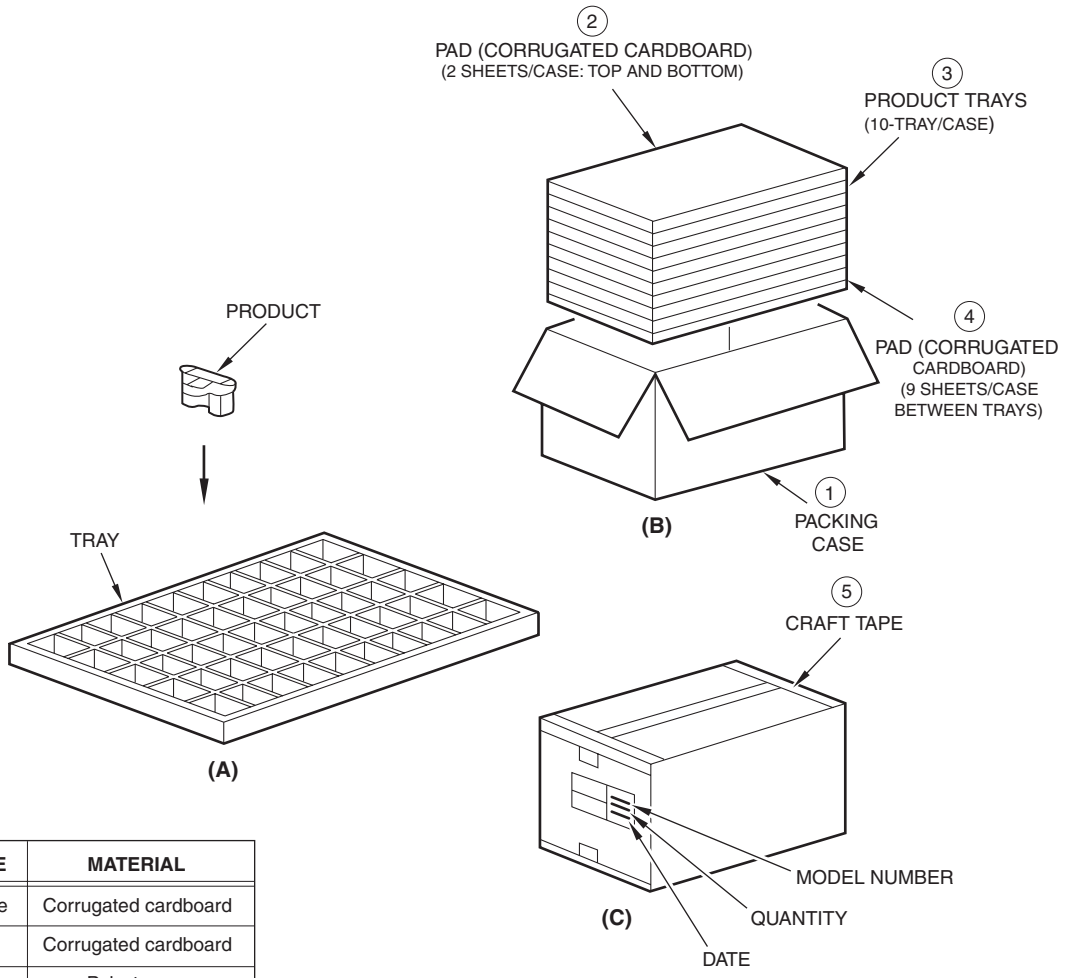
PIN	SIGNAL NAME
①	V_O
②	GND
③	V_{CC}

Connector: J.S.T. Trading Company, LTD
S3B-PH

NOTES:
1. Dimensions shall reference lens center.
2. Unspecified tolerance shall be ±0.3 mm.
3. Scale: 2/1, dimensions are in mm.

GP2D12-3

PACKING SPECIFICATION



PART NAME	MATERIAL
Packing case	Corrugated cardboard
Pad	Corrugated cardboard
Tray	Polystyrene

PACKING METHOD

1. Each tray holds 50 pieces. Packing methods are shown in (A).
2. Each box holds 10 trays. Pads are added to top and bottom, and between layers, as in (B).
   top and bottom. Put pads between each tray (9 pads total) see above drawing (B).
3. The box is sealed with craft tape. (C) shows the location of the Model number, Quantity, and Inspection date.
4. Package weight: Approximately 4 kg.

GP2Y0A02YK-8

NOTES

- Keep the sensor lens clean. Dust, water, oil, and other contaminants can deteriorate the characteristics of this device. Applications should be designed to eliminate sources of lens contamination.
- When using a protective cover over the emitter and detector, ensure the cover efficiently transmits light throughout the wavelength range of the LED (λ = 850 nm ± 70 nm). Both sides of the protective cover should be highly polished. Use of a protective cover may decrease the effective distance over which the sensor operates. Ensure that any cover does not negatively affect the operation over the intended application range.
- Objects in proximity to the sensor may cause reflections that can affect the operation of the sensor.
- Sources of high ambient light (the sun or strong artificial light) may affect measurement. For best results, the application should be designed to prevent interference from direct sunlight or artificial light.
- Using the sensor with a mirror can induce measurement errors. Often, changing the incident angle on the mirror can correct this problem.
- If a prominent boundary line exists in the surface being measured, it should be aligned vertically to avoid measurement error. See Figure 6 for further details.
- When measuring the distance to objects in motion, align the sensor so that the motion is in the horizontal direction instead of vertical. Figure 7 illustrates the preferred alignment.
- A 10 µF (or larger) bypass capacitor between V_{CC} and GND near the sensor is recommended.
- To clean the sensor, use a dry cloth. Use of any liquid to clean the device may result in decreased sensitivity or complete failure.
- Excessive mechanical stress can damage the internal sensor or lens.

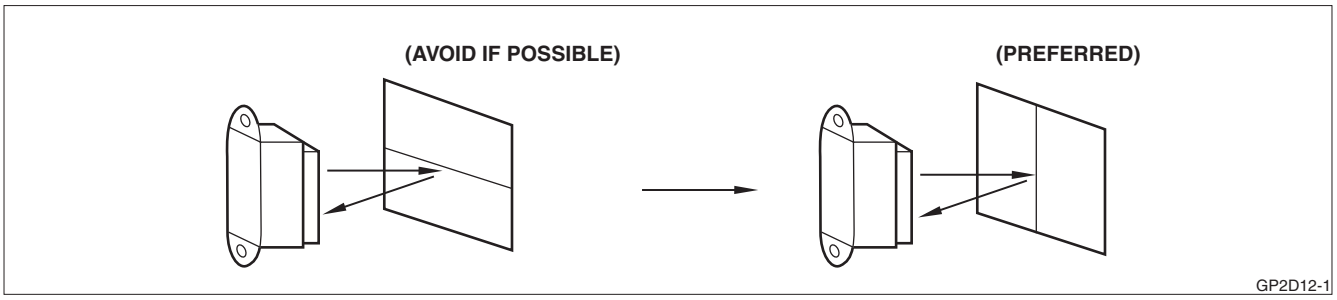


Figure 6. Proper Alignment to Surface Being Measured

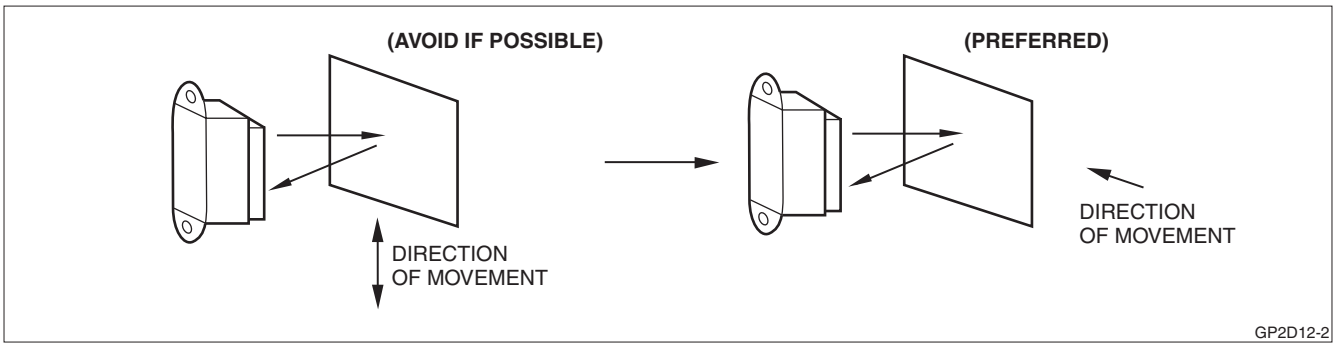


Figure 7. Proper Alignment to Moving Surfaces

NOTICE

The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property right. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP devices.

SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structures and other contents described herein at any time without notice in order to improve design or reliability.

Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. Manufacturing locations are also subject to change without notice.

In the absence of confirmation by device specification sheets, SHARP takes no responsibility for any defects that occur in equipment using any SHARP devices shown in catalogs, data books, etc.

The devices listed in this publication are designed for standard applications for use in general electronic equipment. SHARP's devices shall not be used for or in connection with equipment that requires an extremely high level of reliability, such as military and aerospace applications, telecommunication equipment (trunk lines), nuclear power control equipment and medical or other life support equipment (e.g. Scuba). SHARP takes no responsibility for damage caused by improper use of device, which does not meet the conditions for use specified in the relevant specification sheet.

If the SHARP devices listed in the publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Law of Japan, it is necessary to obtain approval to export such SHARP devices.

Contact and consult with a SHARP representative if there are any questions about the contents of this publication.

SHARP

SHARP CORPORATION
SALES & MARKETING GROUP
ELECTRONIC COMPONENTS & DEVICES
22-22 NAGAIKE-CHO, ABENO-KU, OSAKA 545-8522, JAPAN
PHONE: (81) 6-6621-1221
FAX: (81) 6117-725300, 6117-725301, 6117-725302
http://sharp-world.com/products/device

Specifications are subject to change without notice.

Countries and Areas

U.S.A.

SHARP MICROELECTRONICS
OF THE AMERICAS

North American Head Office
5700 North West Pacific Rim Boulevard
Camas,Washington 98607 USA
PHONE: (1)360-834-2500
FAX: (1)360-834-8903
http://www.sharpsma.com

Western Area

1980 Zanker Road, San Jose, CA 95112
PHONE: (1)408-436-4900
FAX: (1)408-436-0924

5901 Bolsa Ave.
Huntington Beach, CA 92647-2053
PHONE: (1)714-903-4600
FAX: (1)714-903-0295

6390 Greenwich Drive, S uite 175
San Diego, CA 92122
PHONE: (1)858-597-0982
FAX: (1)858-597-8701

Central Area

85 W. Algonquin Road, Suite 280
Arlington Heights, IL 60005
PHONE: (1)847-258-2750
FAX: (1)847-439-2479

6303 Commerce Drive, Suite 175 Irving, TX 75063
PHONE: (1)972-582-1710
FAX: (1)972-580-7537

8911 Capitol of Texas Hwy. Suite 3130
Austin, TX 78759
PHONE: (1)512-349-7262
FAX: (1)512-349-7002

20333 State Hwy. 249, Suite 200 Houston, TX 77070
PHONE: (1)281-378-1520
FAX: (1)281-378-1521

W129 S 9647 Tony Lema Lane Muskego, WI 53150
PHONE: (1)414-529-9568
FAX: (1)414-529-9569

3001 West Big Beaver Road, Suite 722
Troy, ML 48084
PHONE: (1)248-458-1527
FAX: (1)248-458-6255

Eastern Area

1070 N. Kimbles Road, Yardley, PA 19067
PHONE: (1)215-321-5530
FAX: (1)215-321-5534

200 Wheeler Rd., Burlington, MA 01803
PHONE: (1)781-270-7979; (1)781-229-5100
FAX: (1)781-229-9117

8000 Regency Parkway, Suite 280 Cary, NC 27511
PHONE: (1)919-460-0695
FAX: (1)919-460-0795

2321 Sidney St. Pittsburgh, PA 15203
PHONE: (1)412-381-1191
FAX: (1)412-381-1192

4875 North Federal Highway, Third Floor
Ft. Lauderlade, FL 33318
PHONE: (1)954-267-8883
FAX: (1)954-267-0254

EUROPE

SHARP MICROELECTRONICS EUROPE
A division of Sharp Electronics (Europe) GmbH

Head Office
Sonninstrasse 3, 20097, Hamburg, Germany
PHONE: (49)180-5073507
FAX: (49)40-2376-2232
http://www.sharpsme.com/

Germany

SME München Office
Fuerstenriederstrasse 5, 80687 München, Germany
PHONE: (49)89-5468420
FAX: (49)89-54 684250

France

SME Paris Office
1 Rue Raoul Follereau Bussy Saint Georges
77608 Marne la Vallee Cedex 3
PHONE: (33)1 6476 22 22
FAX: (33)1 6476 22 23

Italy

SME Milano Office
Centro Direzionale Colleoni
Palazzo Taurus Ingresso 2
20041 Agrate Brianza, Milano, Italy
PHONE: (390)39-68 99 946
FAX: (390)39-68 99 948

U.K .

SME London Office
Centennial Court, Easthampstead Road,
Bracknell, Berkshire R G12 1YQ, United Kingdom
PHONE: (44)1344-86 99 22
FAX: (44)1344-36 09 03

Ireland

SME Dublin Office
First Floor, Block 1, St. Johns Court, Santry,
Dublin 9, Ireland
PHONE: (353)1-842 87 05
FAX: (353)1-842 84 55

ASIA

SHARP ELECTRONICS (SHANGHAI) CO., LTD.
Microelectronics Sales & Marketing Division
16F, King Tower, 28 Xin Jin Qiao Road,
Pudong DIST, Shanghai 201206 P.R . China
PHONE: (86)21-5854-7710/21-5834-6056
FAX: (86)21-5854-4340/21-5834-6057

Registered Address
No. 11, De Bao Road, Xin Development BLDG
46 Wai Gao Qiao Free Trade Zone, Shanghai
200131, P.R . China

Beijing Office
Room 1062, Beijing Jing An Center No. 8 East
Bei San Huan Road, Chao Yang DIST, Beijing
100028 P.R . China
PHONE: (86) 10-6466-7543/10-6466-6561
FAX: (86) 10-6468-8920
http://sharp-world.com/products/devicechina/
index.html

SHARP-ROXY (HONG KONG) LTD.
Device Sales Division, 17/F, Admiralty Centre,
Tower 1, 18 Harcourt Road, Hong Kong
PHONE: (852)28229311
FAX: (852)28660779
http://www.sharp.com.hk

Shenzhen Representative Office
Room 13B1, Tower C, Electronics Science &
Technology Building, Shen Nan Zhong Road,
Shenzhen, P.R . China
PHONE: (86)755-83273731
FAX: (86)755-83273735

SHARP ELECTRONIC COMPONENTS
(TAIWAN) CORPORATION
8F-A, No. 16, Sec. 4, Nanking E. R d., Taipei, Taiwan
PHONE: (886)2-2577-7341
FAX: (886)2-2577-7326/2-2577-7328

SHARP ELECTRONICS (SINGAPORE) PTE ., LTD.
396 Alexandra Road #07-00
BP Tower Singapore 119954
PHONE: (65) 62713566
FAX: (65) 62713855
http://www.sesl-sharp.com

SHARP MICROELECTRONICS
TECHNOLOGY (M) SDN BHD.
Suite E 408, 4th Floor, East Tower,
Wisma Consplant 1, No. 2 J ln. SS 16/4,
Subng Jaya, 47500, Selangor Darul Ehsan, Malaysia
PHONE: (60) 3-5637-8964
FAX: (60) 3-5638-4029

SHARP ELECTRONIC COMPONENTS
(KOREA) CORPORATION
RM 501 iLsin B/D. 541, Dohwa-dong,
Mapo-ku, Seoul, Korea, 121-701
PHONE: (82)2-711-5813 ~ 8
FAX: (82)2-711-5819

Distributed By

Reference Code SMA05006

GP1A50HRJ00F

Gap : 3mm, Slit : 0.5mm
*OPIC Output
Case package Transmissive Photointerrupter

■ Description

GP1A50HRJ00F is a standard, OPIC output, transmissive photointerrupter with opposing emitter and detector in a case, providing non-contact sensing. For this family of devices, the emitter and detector are inserted in a case, resulting in a through-hole design.

The case is unique because it uses additional screw fixing holes, on both sides 3.2mm diameter, and the shape prevents miss-orientation.

■ Features

1. Transmissive with OPIC output
2. Highlights :
   - Verical Slit for alternate motion detection
   - Output Low Level at intercepting optical path
   - Includes additional screw fixing holes
3. Key Parameters :
   - Gap Width : 3mm
   - Slit Width (detector side) : 0.5mm
   - Package : 25×10×6 mm
4. Lead free and RoHS directive compliant

■ Agency approvals/Compliance

1. Compliant with RoHS directive

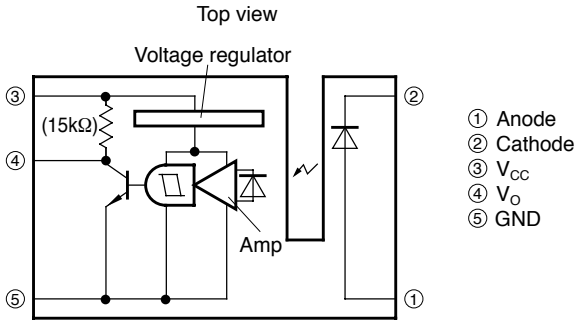
■ Applications

1. General purpose detection of object presence or motion.
2. Example : Printer, FAX, Optical storage unit

* "OPIC"(Optical IC) is a trademark of the SHARP Corporation. An OPIC consists of a light-detecting element and a signal-processing

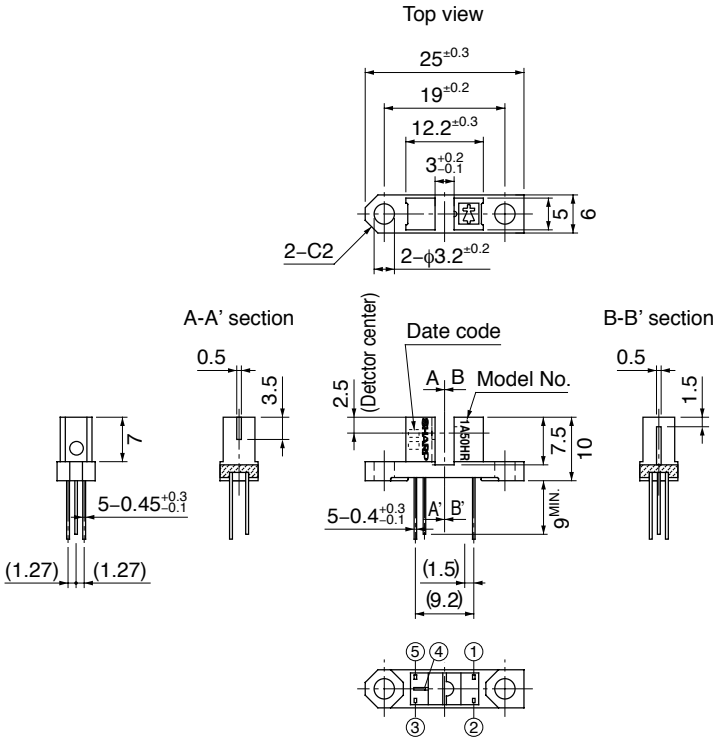
Sheet No.: D3-A03501EN
Date Oct. 3. 2005

■ Internal Connection Diagram



■ Outline Dimensions

(Unit : mm)



• Unspecified tolerance shall be as follows ;

Dimensions(d)	Tolerance
d≤6	±0.1
6<d≤18	±0.2
18<d≤25	±0.025

• ( ) : Reference dimensions

Product mass : approx. 0.5g

Dip soldering material : Sn–3Ag–0.5Cu

Date code (2 digit)

1st digit		2nd digit	
Year of production		Month of production	
A.D.	Mark	Month	Mark
2000	0	1	1
2001	1	2	2
2002	2	3	3
2003	3	4	4
2004	4	5	5
2005	5	6	6
2006	6	7	7
2007	7	8	8
2008	8	9	9
2009	9	10	X
2010	0	11	Y
:	:	12	Z

repeats in a 10 year cycle

Country of origin

Japan, Indonesia or Philippines
(Indicated on the packing case)

■ Absolute Maximum Ratings

(T_a=25°C)

Parameter		Symbol	Rating	Unit
Input	*1 Forward current	I_F	50	mA
	*1, 2 Peak forward current	I_{FM}	1	A
	Reverse voltage	V_R	6	V
	Power dissipation	P	75	mW
Output	Supply voltage	V_{CC}	–0.5 to +17	V
	Output current	I_O	50	mA
	Power dissipation	P_O	250	mW
Operating temperature		T_{opr}	–25 to +85	°C
Storage temperature		T_{stg}	–40 to +100	°C
*3 Soldering temperature		T_{sol}	260	°C

*1 Refer to Fig. 1, 2, 3
*2 Pulse width ≤ 100μs, Duty ratio=0.01
*3 For 5s or less

■ Electro-optical Characteristics

(T_a=25°C)

Parameter			Symbol	Condition	MIN.	TYP.	MAX.	Unit
Input	Forward voltage		V_F	I_F=5mA	–	1.1	1.4	V
	Reverse current		I_R	V_R=3V	–	–	10	μA
Output	Operating supply voltage		V_{CC}	–	4.5	–	17	V
	Low level output voltage		V_{OL}	V_{CC}=5V, I_{OL}=16mA, I_F=0	–	0.15	0.4	V
	High level output voltage		V_{OH}	V_{CC}=5V, I_F=5mA	4.9	–	–	V
	Low level supply current		I_{CCL}	V_{CC}=5V, I_F=0	–	1.7	3.8	mA
	High level supply current		I_{CCH}	V_{CC}=5V, I_F=5mA	–	0.7	2.2	mA
Transfer characteristics	*4 "Low→High" threshold input current		I_{FLH}	V_{CC}=5V	–	1	5	mA
	*5 Hysteresis		I_{FHL}/I_{FLH}	V_{CC}=5V	0.55	0.75	0.95	–
	*6 Responce time	"Low→High" Propagation delay time	t_{PLH}	V_{CC}=5V, I_F=5mA, R_L=280Ω	–	3	9	μs
		"High→Low" Propagation delay time	t_{PHL}		–	5	15	
		Rise time	t_r		–	0.1	0.5	
		Fall time	t_f		–	0.05	0.5	

*4 I_{FLH} represents forward current when output goes from "Low" to "High".
*5 I_{FHL} represents forward current when output goes from "High" to "Low".
*6 Test circuit for response time is shown in Fig.12.

Fig.1 Forward Current vs. Ambient Temperature

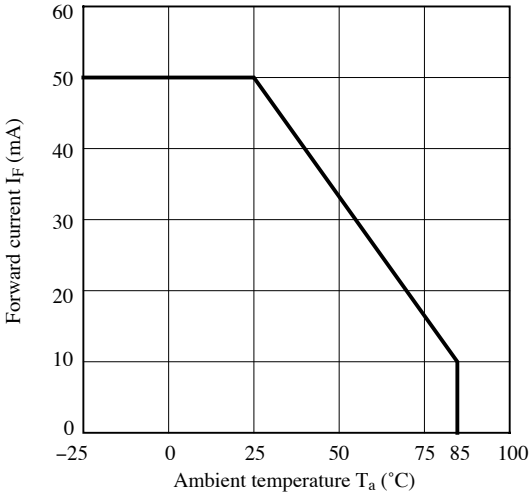


Fig.2 Output Power Dissipation vs. Ambient Temperature

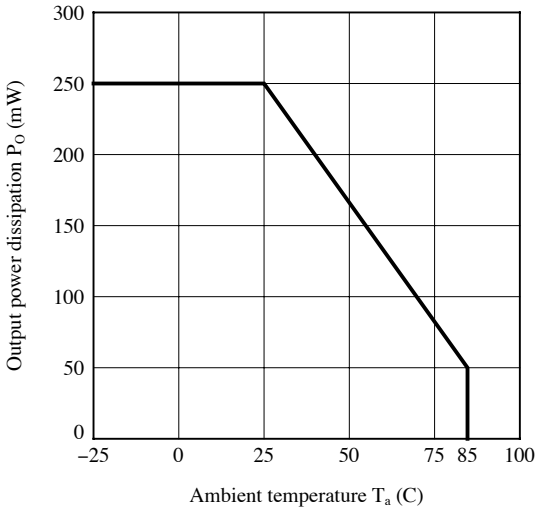


Fig.3 Low Level Output Current vs. Ambient Temperature

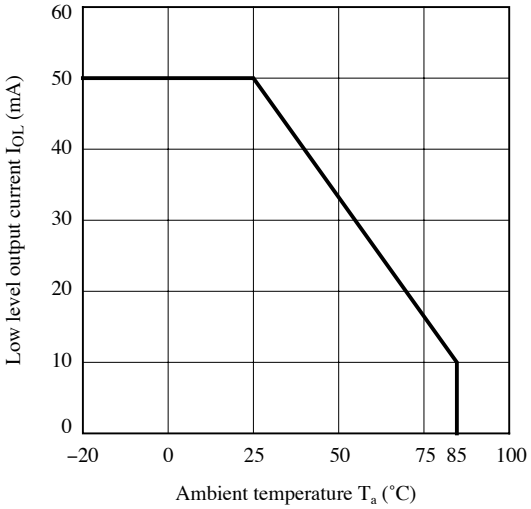


Fig.4 Forward Current vs. Forward Voltage

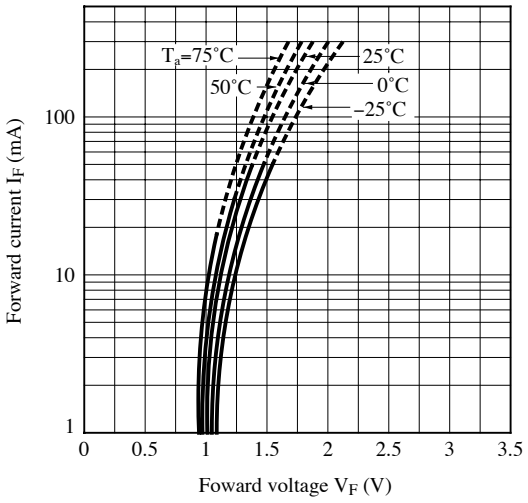


Fig.5 Relative Threshold Input Current vs. Supply Voltage

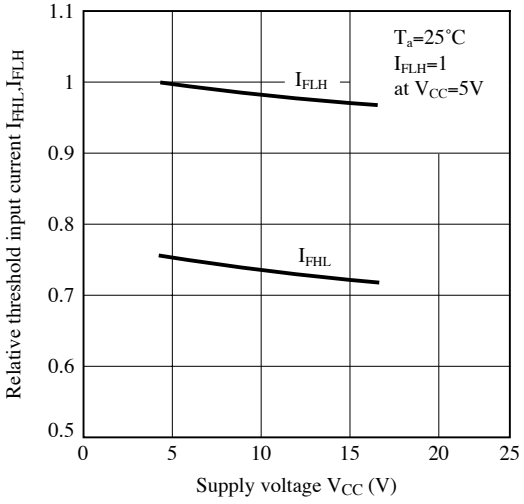


Fig.6 Relative Threshold Input Current vs. Ambient Temperature

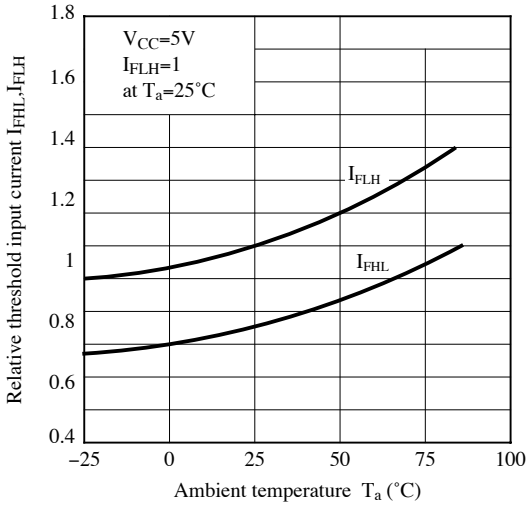


Fig.7 Low Level Output Voltage vs. Low Level Output Current

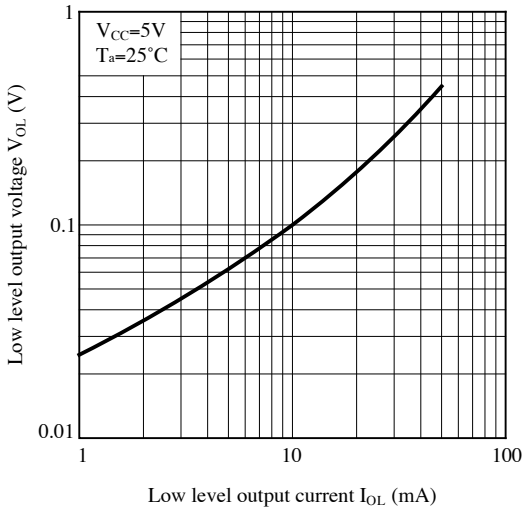


Fig.8 Low Level Output Voltage vs. Ambient Temperature

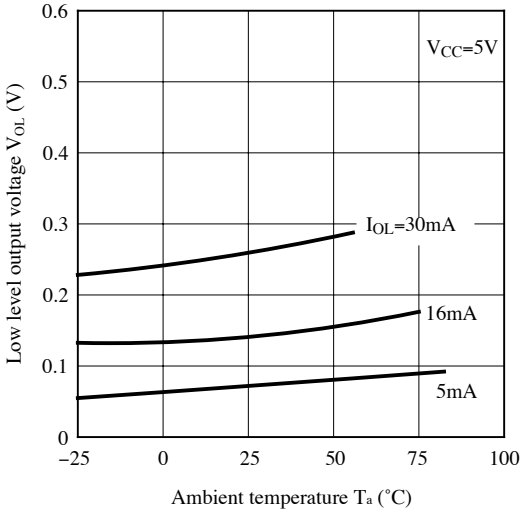


Fig.9 Supply Current vs. Ambient Temperature (H Gain Mode)

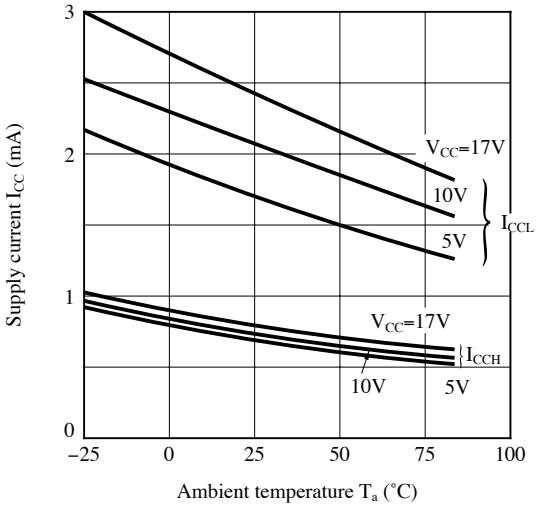


Fig.10 Propagation Delay Time vs. Forward Current

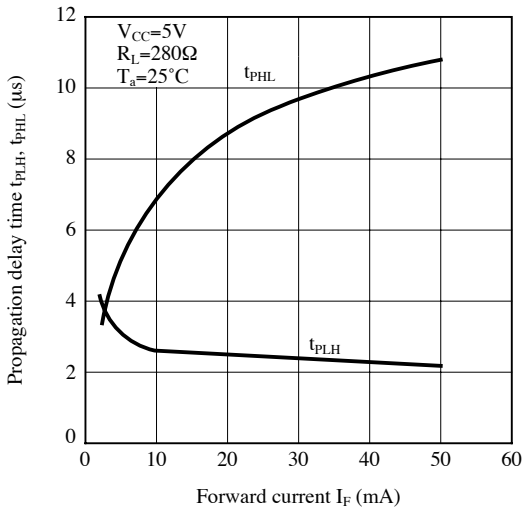


Fig.11 Rise Time,Fall Time vs. Load Resistance

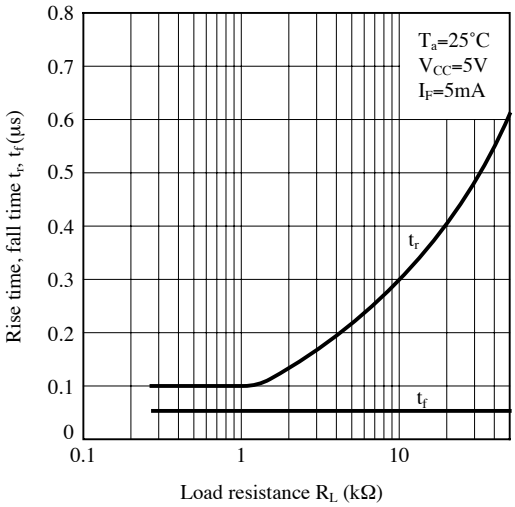
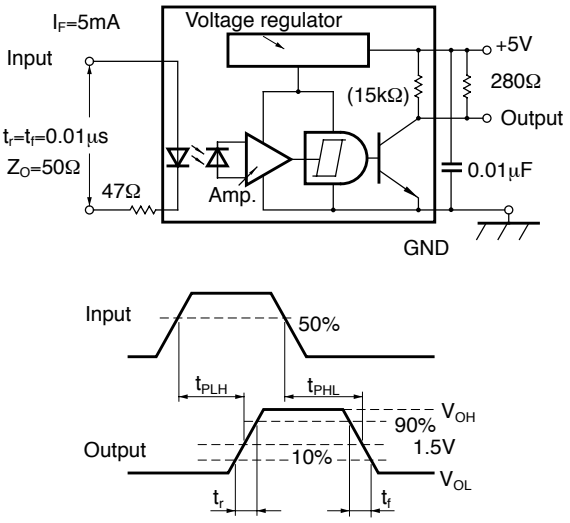


Fig.12 Test Circuit for Response Time



Remarks : Please be aware that all data in the graph are just for reference and not for guarantee.

■ Design Considerations

● Recommended operating conditions

Parameter	Symbol	MIN.	TYP.	MAX.	Unit
Output current	I_O	–	–	16	mA
Forward current	I_F	10	–	20	mA
Operating terperature	T_{opr}	0	–	70	˚C

● Notes about static electricity

Transisiter of detector side in bipolar configuration may be damaged by static electricity due to its minute design.

When handing these devices, general countermeasure against static electricity should be taken to avoid breakdown of devices or degradation of characteristics.

● Design guide

1) Prevention of detection error

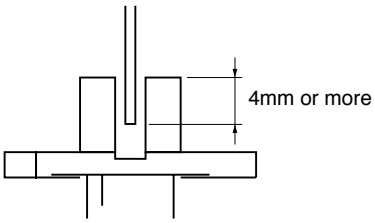
To prevent photointerrupter from faulty operation caused by external light, do not set the detecting face to the external light.

2) In order to stabilize power supply line, connect a by-pass capacitor of more than 0.01μF between V_{CC} and GND near the device.

3) Position of opaque board

Opaque board shall be installed at place 4mm or more from the top of elements.

(Example)



This product is not designed against irradiation and incorporates non-coherent IRED.

● Degradation

In general, the emission of the IRED used in photocouplers will degrade over time.

In the case of long term operation, please take the general IRED degradation (50% degradation over 5 years) into the design consideration.

● Parts

This product is assembled using the below parts.

• Photodetector (qty. : 1) [Using a silicon photodiode as light detecting portion, and a bipolar IC as signal processing circuit]

Category	Maximum Sensitivity wavelength (nm)	Sensitivity wavelength (nm)	Response time (μs)
Photodiode	900	400 to 1 200	3

• Photo emitter (qty. : 1)

Category	Material	Maximum light emitting wavelength (nm)	I/O Frequency (MHz)
Infrared emitting diode (non-coherent)	Gallium arsenide (GaAs)	950	0.3

• Material

Case	Connector terminal flame finish
Black NORYL resin	Solder dip. (Sn–3Ag–0.5Cu)

• Others

Laser generator is not used.

■ Manufacturing Guidelines

● Soldering Method

Flow Soldering:

Soldering should be completed below 260˚C and within 5 s.
Please take care not to let any external force exert on lead pins.
Please don't do soldering with preheating, and please don't do soldering by reflow.

Hand soldering

Hand soldering should be completed within 3 s when the point of solder iron is below 350˚C.
Please solder within one time.
Please don't touch the terminals directly by soldering iron.
Soldered product shall treat at normal temperature.

Other notice

Please test the soldering method in actual condition and make sure the soldering works fine, since the impact on the junction between the device and PCB varies depending on the cooling and soldering conditions.

Flux

Some flux, which is used in soldering, may crack the package due to synergistic effect of alcohol in flux and the rise in temperature by heat in soldering. Therefore, in using flux, please make sure that it does not have any influence on appearance and reliability of the photointerrupter.

● Cleaning instructions

Solvent cleaning :

Solvent temperature should be 45˚C or below. Immersion time should be 3 minutes or less.

Ultrasonic cleaning :

The effect to device by ultrasonic cleaning differs by cleaning bath size, ultrasonic power output, cleaning time, PCB size or device mounting condition etc.

Please test it in actual using condition and confirm that doesn't occur any defect before starting the ultrasonic cleaning.

Recommended solvent materials :

Ethyl alcohol, Methyl alcohol and Isopropyl alcohol.

● Presence of ODC

This product shall not contain the following materials.

And they are not used in the production process for this product.

Regulation substances : CFCs, Halon, Carbon tetrachloride, 1.1.1-Trichloroethane (Methylchloroform)

Specific brominated flame retardants such as the PBBOs and PBBs are not used in this product at all.

This product shall not contain the following materials banned in the RoHS Directive (2002/95/EC).

- Lead, Mercury, Cadmium, Hexavalent chromium, Polybrominated biphenyls (PBB), Polybrominated diphenyl ethers (PBDE).

■ Package specification

● Case package

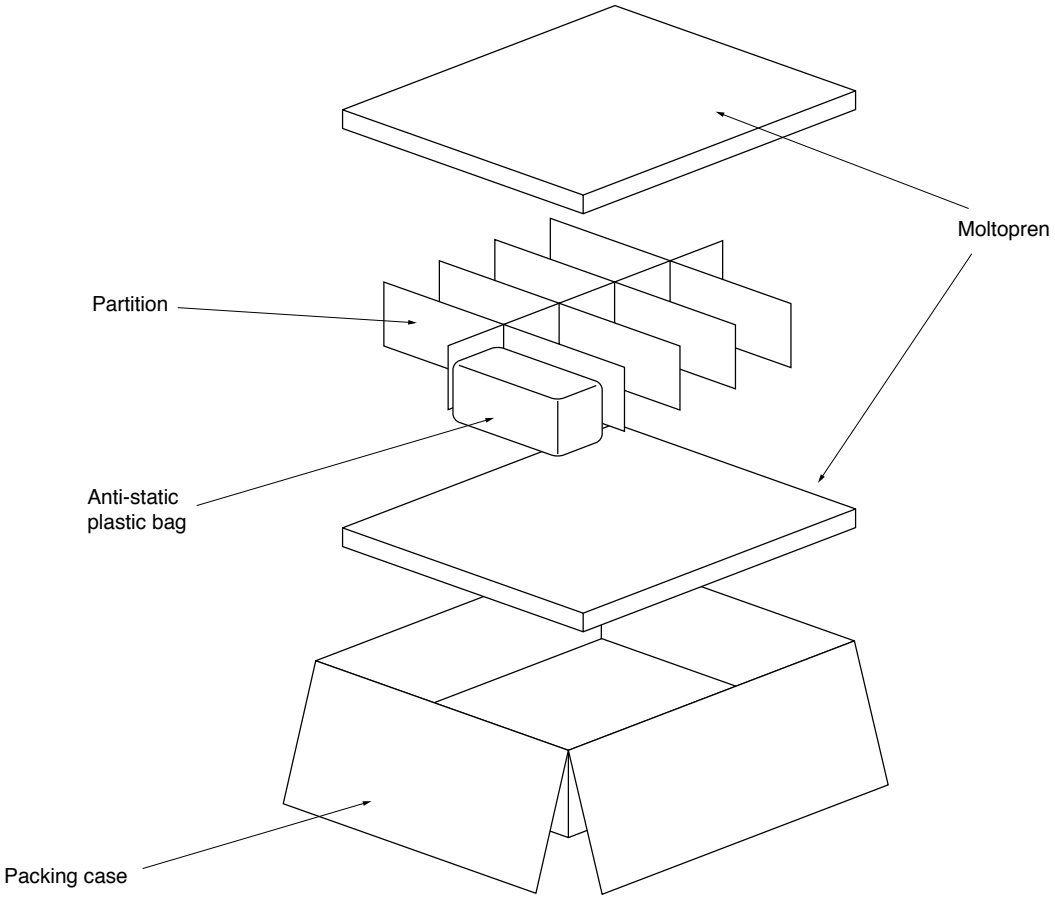
Package materials

Anti-static plastic bag : Polyethtylene
Moltopren : Urethane
Partition : Corrugated fiberboard
Packing case : Corrugated fiberboard

Package method

100 pcs of products shall be packaged in a plastic bag, Ends shall be fixed by stoppers. The bottom ot the packing case is covered with moltopren, and the partition is set in the packing case. Each partition should have 1 plastic bag.
The 10 plastic bags containing a product are put in the packing case.
Moltopren should be located after all product are settled (1 packing contains 1 000 pcs).

Packing composition



■ Important Notices

· The circuit application examples in this publication are provided to explain representative applications of SHARP devices and are not intended to guarantee any circuit design or license any intellectual property rights. SHARP takes no responsibility for any problems related to any intellectual property right of a third party resulting from the use of SHARP's devices.

· Contact SHARP in order to obtain the latest device specification sheets before using any SHARP device. SHARP reserves the right to make changes in the specifications, characteristics, data, materials, structure, and other contents described herein at any time without notice in order to improve design or reliability. Manufacturing locations are also subject to change without notice.

· Observe the following points when using any devices in this publication. SHARP takes no responsibility for damage caused by improper use of the devices which does not meet the conditions and absolute maximum ratings to be used specified in the relevant specification sheet nor meet the following conditions:

(i) The devices in this publication are designed for use in general electronic equipment designs such as:

--- Personal computers
--- Office automation equipment
--- Telecommunication equipment [terminal]
--- Test and measurement equipment
--- Industrial control
--- Audio visual equipment
--- Consumer electronics

(ii) Measures such as fail-safe function and redundant design should be taken to ensure reliability and safety when SHARP devices are used for or in connection with equipment that requires higher reliability such as:

--- Transportation control and safety equipment (i.e., aircraft, trains, automobiles, etc.)
--- Traffic signals
--- Gas leakage sensor breakers
--- Alarm equipment
--- Various safety devices, etc.

(iii) SHARP devices shall not be used for or in connection with equipment that requires an extremely high level of reliability and safety such as:

--- Space applications
--- Telecommunication equipment [trunk lines]
--- Nuclear power control equipment
--- Medical and other life support equipment (e.g., scuba).

· If the SHARP devices listed in this publication fall within the scope of strategic products described in the Foreign Exchange and Foreign Trade Law of Japan, it is necessary to obtain approval to export such SHARP devices.

· This publication is the proprietary product of SHARP and is copyrighted, with all rights reserved. Under the copyright laws, no part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, in whole or in part, without the express written permission of SHARP. Express written permission is also required before any use of this publication may be made by a third party.

· Contact and consult with a SHARP representative if there are any questions about the contents of this publication.

L293x Quadruple Half-H Drivers

1 Features

- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- High-Noise-Immunity Inputs
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

2 Applications

- Stepper Motor Drivers
- DC Motor Drivers
- Latching Relay Drivers

3 Description

The L293 and L293D devices are quadruple high-current half-H drivers. The L293 is designed to provide bidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed to provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V. Both devices are designed to drive inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN.

The L293 and L293D are characterized for operation from 0°C to 70°C.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
L293NE	PDIP (16)	19.80 mm × 6.35 mm
L293DNE	PDIP (16)	19.80 mm × 6.35 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Logic Diagram

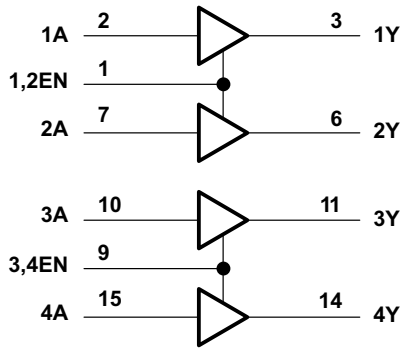


Table of Contents

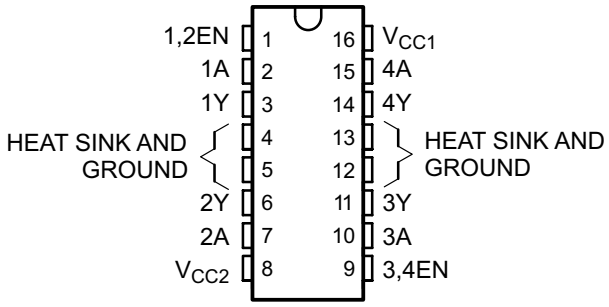
4 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision C (November 2004) to Revision D	Page
• Removed *Ordering Information* table	1
• Added *ESD Ratings* and *Thermal Information* tables, *Feature Description* section, *Device Functional Modes*, *Application and Implementation* section, *Power Supply Recommendations* section, *Layout* section, *Device and Documentation Support* section, and *Mechanical, Packaging, and Orderable Information* section.	1

5 Pin Configuration and Functions

NE Package
16-Pin PDIP
Top View



Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
1,2EN	1	I	Enable driver channels 1 and 2 (active high input)
<1:4>A	2, 7, 10, 15	I	Driver inputs, noninverting
<1:4>Y	3, 6, 11, 14	O	Driver outputs
3,4EN	9	I	Enable driver channels 3 and 4 (active high input)
GROUND	4, 5, 12, 13	—	Device ground and heat sink pin. Connect to printed-circuit-board ground plane with multiple solid vias
V_{CC1}	16	—	5-V supply for internal logic translation
V_{CC2}	8	—	Power VCC for drivers 4.5 V to 36 V

6 Specifications

6.1 Absolute Maximum Ratings

over operating free-air temperature range (unless otherwise noted)(1)

	MIN	MAX	UNIT
Supply voltage, V_{CC1}(2)		36	V
Output supply voltage, V_{CC2}		36	V
Input voltage, V_I		7	V
Output voltage, V_O	–3	V_{CC2} + 3	V
Peak output current, I_O (nonrepetitive, t ≤ 5 ms): L293	–2	2	A
Peak output current, I_O (nonrepetitive, t ≤ 100 µs): L293D	–1.2	1.2	A
Continuous output current, I_O: L293	–1	1	A
Continuous output current, I_O: L293D	–600	600	mA
Maximum junction temperature, T_J		150	°C
Storage temperature, T_{stg}	–65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) All voltage values are with respect to the network ground terminal.

6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001(1)	±2000	V
		Charged-device model (CDM), per JEDEC specification JESD22-C101(2)	±1000	

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.
(2) JEDEC document JEP157 states that 250-V CDM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

			MIN	NOM	MAX	UNIT
	Supply voltage	V_{CC1}	4.5		7	V
		V_{CC2}	V_{CC1}		36	
V_{IH}	High-level input voltage	V_{CC1} ≤ 7 V	2.3		V_{CC1}	V
		V_{CC1} ≥ 7 V	2.3		7	V
V_{IL}	Low-level output voltage		–0.3(1)		1.5	V
T_A	Operating free-air temperature		0		70	°C

(1) The algebraic convention, in which the least positive (most negative) designated minimum, is used in this data sheet for logic voltage levels.

6.4 Thermal Information

THERMAL METRIC(1)		L293, L293D NE (PDIP) 16 PINS	UNIT
$R_{\theta JA}$	Junction-to-ambient thermal resistance (2)	36.4	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	22.5	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	16.5	°C/W
ψ_{JT}	Junction-to-top characterization parameter	7.1	°C/W
ψ_{JB}	Junction-to-board characterization parameter	16.3	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report, SPRA953.
(2) The package thermal impedance is calculated in accordance with JESD 51-7.

6.5 Electrical Characteristics

over operating free-air temperature range (unless otherwise noted)

PARAMETER			TEST CONDITIONS		MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage		L293: I_{OH} = −1 A		V_{CC2} – 1.8	V_{CC2} – 1.4		V
			L293D: I_{OH} = − 0.6 A					
V_{OL}	Low-level output voltage		L293: I_{OL} = 1 A			1.2	1.8	V
			L293D: I_{OL} = 0.6 A					
V_{OKH}	High-level output clamp voltage		L293D: I_{OK} = –0.6 A			V_{CC2} + 1.3		V
V_{OKL}	Low-level output clamp voltage		L293D: I_{OK} = 0.6 A			1.3		V
I_{IH}	High-level input current	A	V_I = 7 V			0.2	100	µA
		EN				0.2	10	
I_{IL}	Low-level input current	A	V_I = 0			–3	–10	µA
		EN				–2	–100	
I_{CC1}	Logic supply current		I_O = 0	All outputs at high level		13	22	mA
				All outputs at low level		35	60	
				All outputs at high impedance		8	24	
I_{CC2}	Output supply current		I_O = 0	All outputs at high level		14	24	mA
				All outputs at low level		2	6	
				All outputs at high impedance		2	4	

6.6 Switching Characteristics

over operating free-air temperature range (unless otherwise noted) V_{CC1} = 5 V, V_{CC2} = 24 V, T_A = 25°C

PARAMETER			TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output from A input	L293NE, L293DNE	C_L = 30 pF, See Figure 2		800		ns
		L293DWP, L293N L293DN			750		
t_{PHL}	Propagation delay time, high-to-low-level output from A input	L293NE, L293DNE			400		ns
		L293DWP, L293N L293DN			200		
t_{TLH}	Transition time, low-to-high-level output	L293NE, L293DNE			300		ns
		L293DWP, L293N L293DN			100		
t_{THL}	Transition time, high-to-low-level output	L293NE, L293DNE			300		ns
		L293DWP, L293N L293DN			350		

6.7 Typical Characteristics

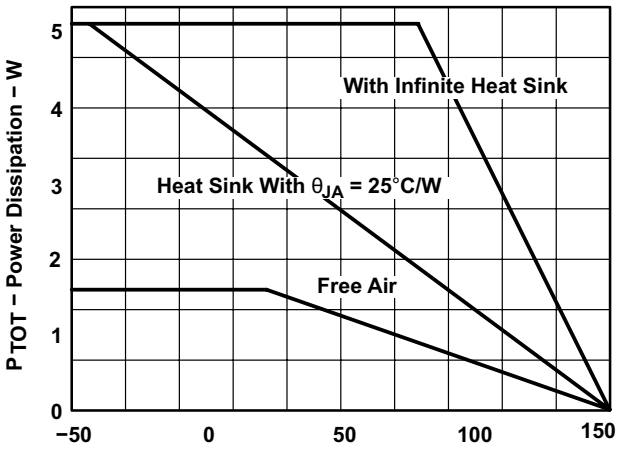
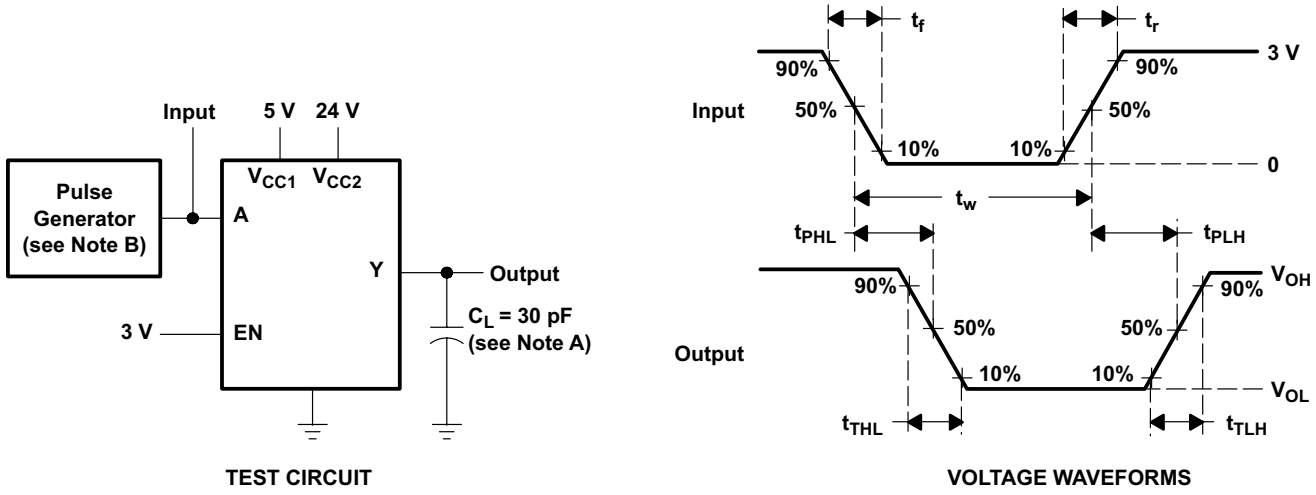


Figure 1. Maximum Power Dissipation vs Ambient Temperature

7 Parameter Measurement Information



TEST CIRCUIT **VOLTAGE WAVEFORMS**

NOTES: A. C_L includes probe and jig capacitance.
B. The pulse generator has the following characteristics: $t_r \leq 10$ ns, $t_f \leq 10$ ns, $t_w = 10$ µs, PRR = 5 kHz, $Z_O = 50$ Ω.

Figure 2. Test Circuit and Voltage Waveforms

8 Detailed Description

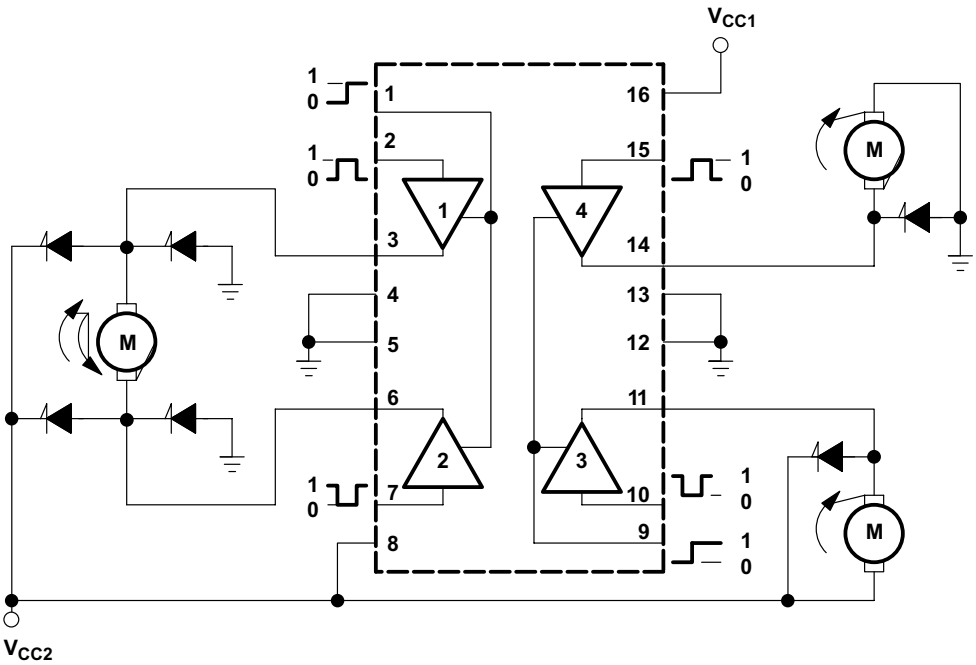
8.1 Overview

The L293 and L293D are quadruple high-current half-H drivers. These devices are designed to drive a wide array of inductive loads such as relays, solenoids, DC and bipolar stepping motors, as well as other high-current and high-voltage loads. All inputs are TTL compatible and tolerant up to 7 V.

Each output is a complete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are active and in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs are off and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

On the L293, external high-speed output clamp diodes should be used for inductive transient suppression. On the L293D, these diodes are integrated to reduce system complexity and overall system size. A V_{CC1} terminal, separate from V_{CC2}, is provided for the logic inputs to minimize device power dissipation. The L293 and L293D are characterized for operation from 0°C to 70°C.

8.2 Functional Block Diagram



Output diodes are internal in L293D.

8.3 Feature Description

The L293x has TTL-compatible inputs and high voltage outputs for inductive load driving. Current outputs can get up to 2 A using the L293.

8.4 Device Functional Modes

Table 1 lists the fuctional modes of the L293x.

Table 1. Function Table (Each Driver)(1)

INPUTS (2)		OUTPUT (Y)
A	EN	
H	H	H
L	H	L
X	L	Z

(1) H = high level, L = low level, X = irrelevant, Z = high impedance (off)
(2) In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

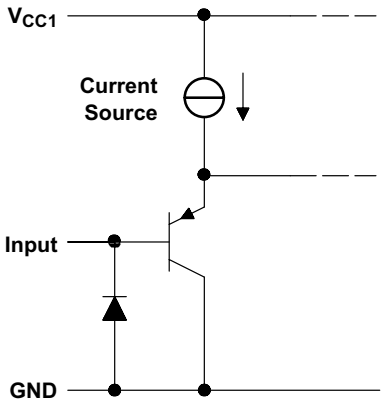


Figure 3. Schematic of Inputs for the L293x

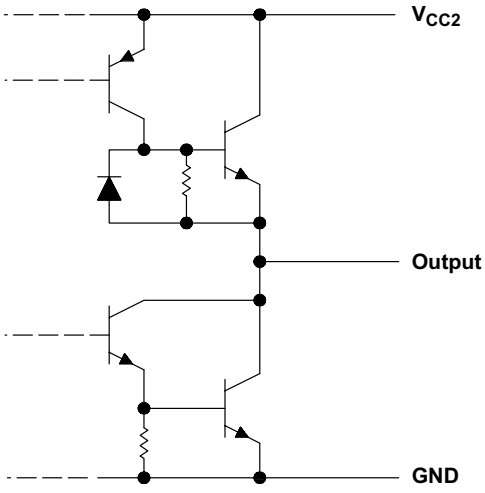


Figure 4. Schematic of Outputs for the L293

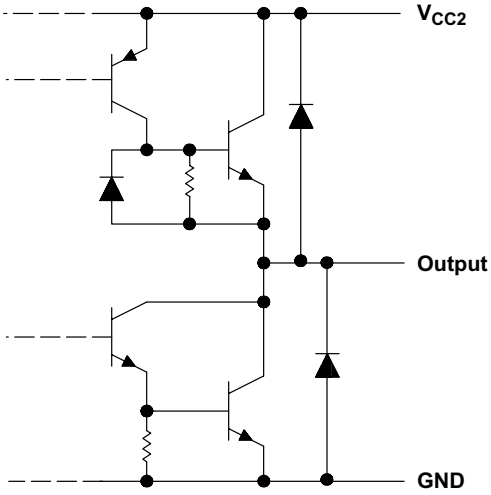


Figure 5. Schematic of Outputs for the L293D

9 Application and Implementation

NOTE

Information in the following applications sections is not part of the TI component specification, and TI does not warrant its accuracy or completeness. TI's customers are responsible for determining suitability of components for their purposes. Customers should validate and test their design implementation to confirm system functionality.

9.1 Application Information

A typical application for the L293 device is driving a two-phase motor. Below is an example schematic displaying how to properly connect a two-phase motor to the L293 device.

Provide a 5-V supply to V_{CC1} and valid logic input levels to data and enable inputs. V_{CC2} must be connected to a power supply capable of supplying the needed current and voltage demand for the loads connected to the outputs.

9.2 Typical Application

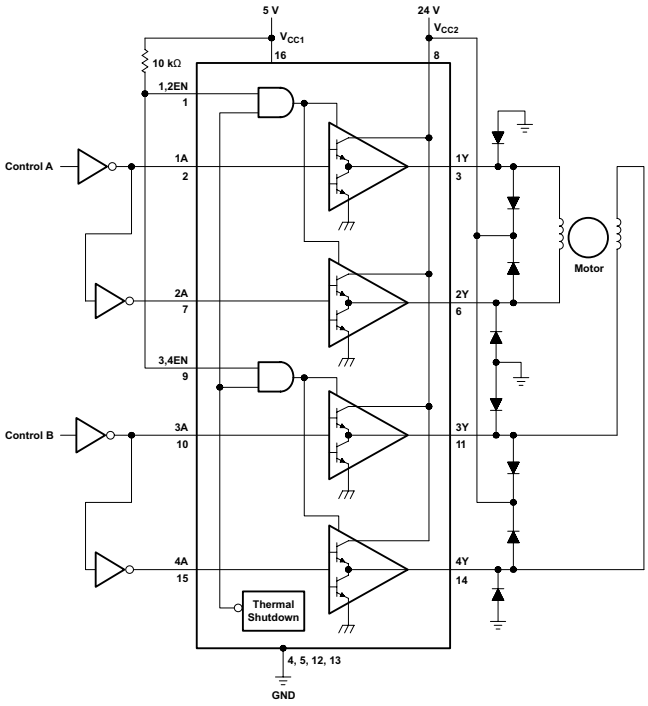


Figure 6. Two-Phase Motor Driver (L293)

9.2.1 Design Requirements

The design techniques in the application above as well as the applications below should fall within the following design requirements.

1. V_{CC1} should fall within the limits described in the *Recommended Operating Conditions*.
2. V_{CC2} should fall within the limits described in the *Recommended Operating Conditions*.
3. The current per channel should not exceed 1 A for the L293 (600mA for the L293D).

9.2.2 Detailed Design Procedure

When designing with the L293 or L293D, careful consideration should be made to ensure the device does not exceed the operating temperature of the device. Proper heatsinking will allow for operation over a larger range of current per channel. Refer to the *Power Supply Recommendations* as well as the *Layout Example*.

Typical Application (continued)

9.2.3 Application Curve

Refer to *Power Supply Recommendations* for additional information with regards to appropriate power dissipation. Figure 7 describes thermal dissipation based on Figure 14.

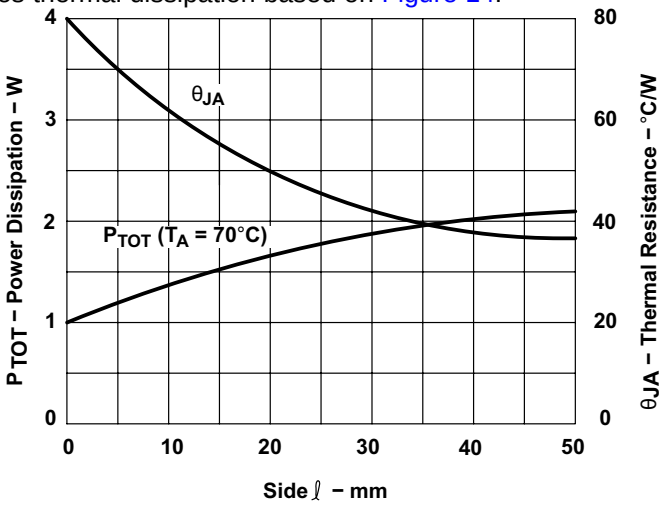


Figure 7. Maximum Power and Junction vs Thermal Resistance

9.3 System Examples

9.3.1 L293D as a Two-Phase Motor Driver

Figure 8 below depicts a typical setup for using the L293D as a two-phase motor driver. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.

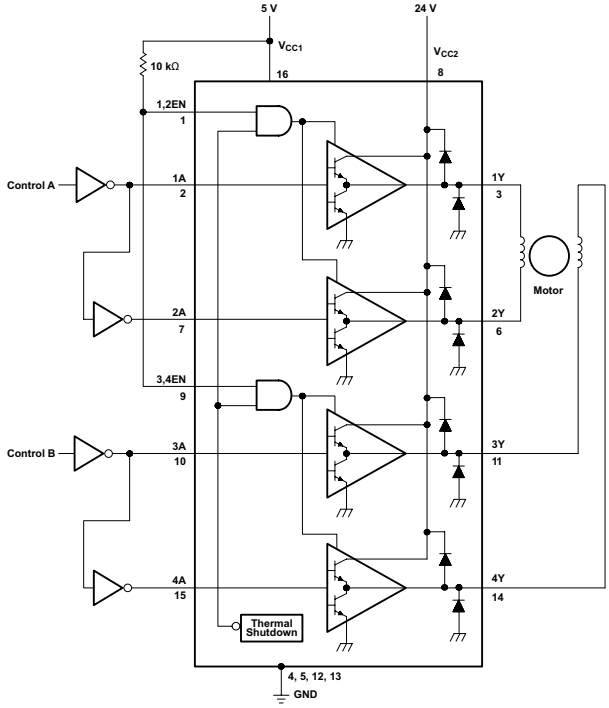
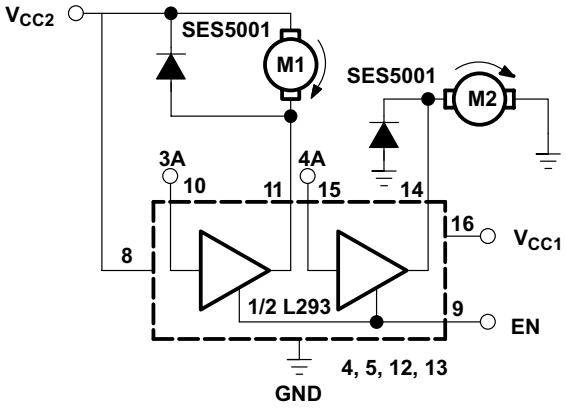


Figure 8. Two-Phase Motor Driver (L293D)

System Examples (continued)

9.3.2 DC Motor Controls

Figure 9 and Figure 10 below depict a typical setup for using the L293 device as a controller for DC motors. Note that the L293 device can be used as a simple driver for a motor to turn on and off in one direction, and can also be used to drive a motor in both directions. Refer to the function tables below to understand unidirectional vs bidirectional motor control. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.



Connections to ground and to supply voltage

Figure 9. DC Motor Controls

Table 2. Unidirectional DC Motor Control

EN	3A	M1 (1)	4A	M2
H	H	Fast motor stop	H	Run
H	L	run	L	Fast motor stop
L	X	Free-running motor stop	X	Free-running motor stop

(1) L = low, H = high, X = don't care

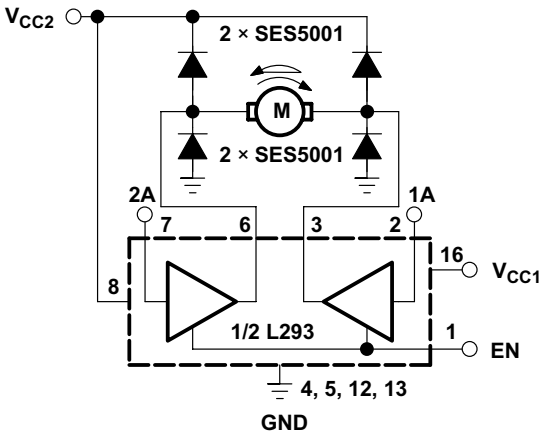


Figure 10. Bidirectional DC Motor Control

Table 3. Bidirectional DC Motor Control

EN	1A	2A	FUNCTION (1)
H	L	H	Turn right
H	H	L	Turn left

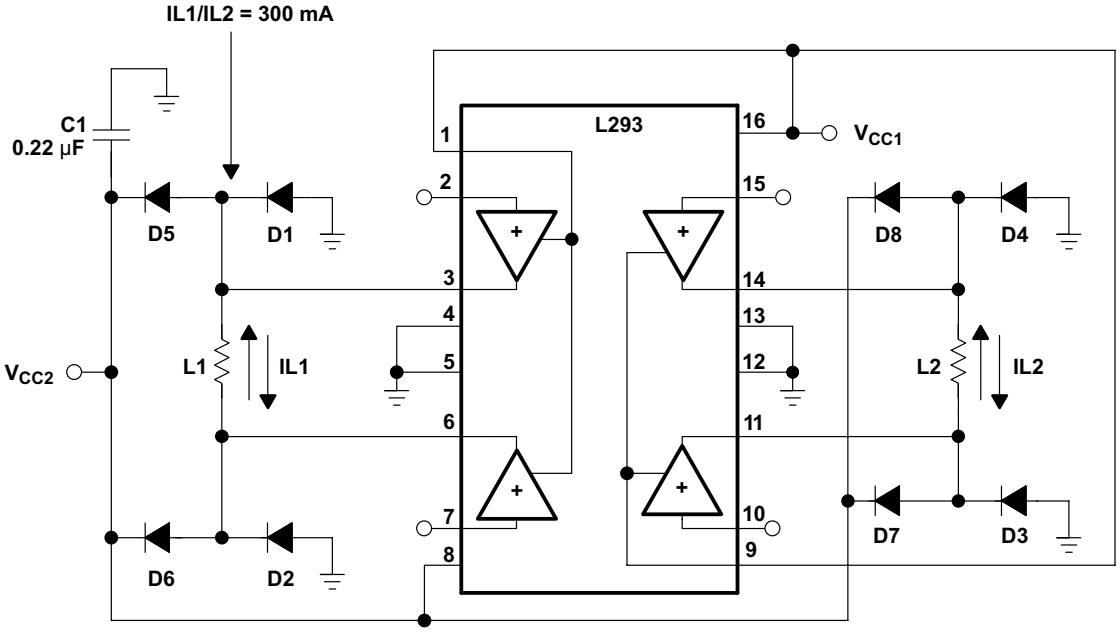
(1) L = low, H = high, X = don't care

Table 3. Bidirectional DC Motor Control (continued)

EN	1A	2A	FUNCTION(1)
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Free-running motor stop

9.3.3 Bipolar Stepping-Motor Control

Figure 11 below depicts a typical setup for using the L293D as a two-phase motor driver. Refer to the *Recommended Operating Conditions* when considering the appropriate input high and input low voltage levels to enable each channel of the device.



D1−D8 = SES5001

Figure 11. Bipolar Stepping-Motor Control

10 Power Supply Recommendations

V_{CC1} is 5 V ± 0.5 V and V_{CC2} can be same supply as V_{CC1} or a higher voltage supply with peak voltage up to 36 V. Bypass capacitors of 0.1 uF or greater should be used at V_{CC1} and V_{CC2} pins. There are no power up or power down supply sequence order requirements.

Properly heatsinking the L293 when driving high-current is critical to design. The Rthj-amp of the L293 can be reduced by soldering the GND pins to a suitable copper area of the printed circuit board or to an external heat sink.

Figure 14 shows the maximum package power PTOT and the θJA as a function of the side of two equal square copper areas having a thickness of 35 µm (see Figure 14). In addition, an external heat sink can be used (see Figure 12).

During soldering, the pin temperature must not exceed 260°C, and the soldering time must not exceed 12 seconds.

The external heatsink or printed circuit copper area must be connected to electrical ground.

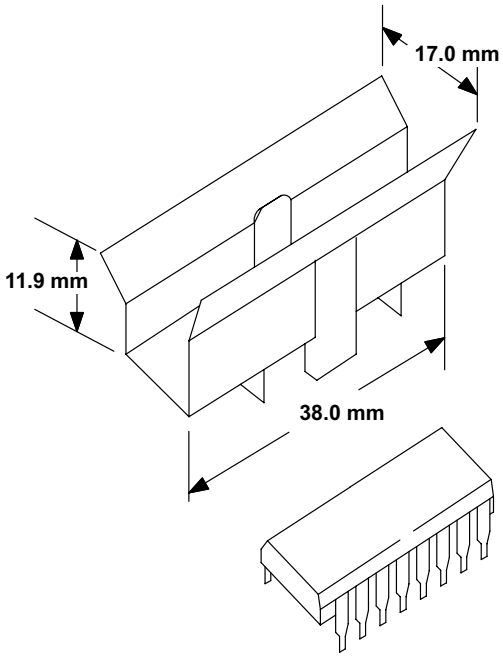


Figure 12. External Heat Sink Mounting Example (θ_{JA} = 25°C/W)

11 Layout

11.1 Layout Guidelines

Place the device near the load to keep output traces short to reduce EMI. Use solid vias to transfer heat from ground pins to ground plane of the printed-circuit-board.

11.2 Layout Example

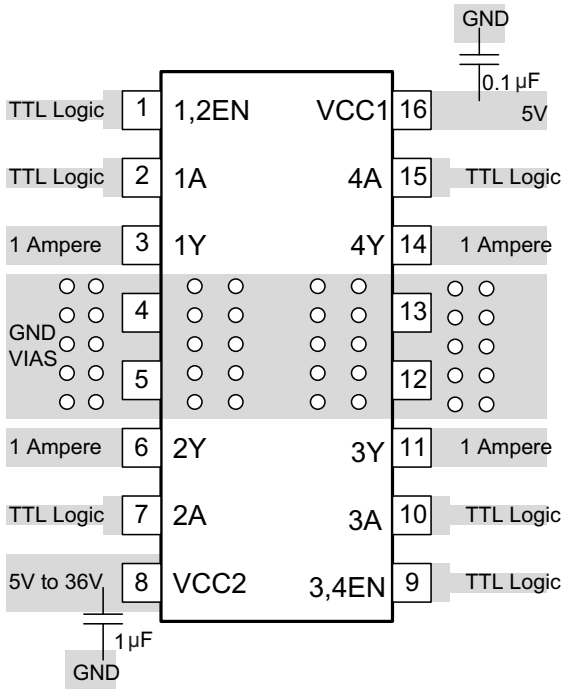


Figure 13. Layout Diagram

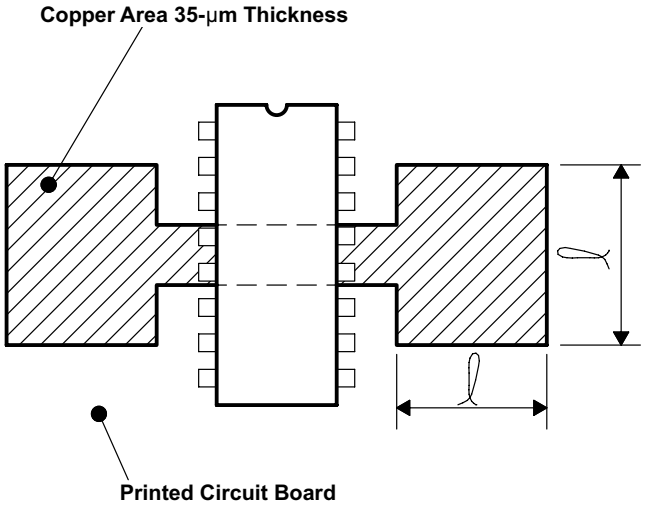


Figure 14. Example of Printed-Circuit-Board Copper Area (Used as Heat Sink)

12 Device and Documentation Support

12.1 Related Links

The table below lists quick access links. Categories include technical documents, support and community resources, tools and software, and quick access to sample or buy.

Table 4. Related Links

PARTS	PRODUCT FOLDER	SAMPLE & BUY	TECHNICAL DOCUMENTS	TOOLS & SOFTWARE	SUPPORT & COMMUNITY
L293	Click here	Click here	Click here	Click here	Click here
L293D	Click here	Click here	Click here	Click here	Click here

12.2 Community Resources

The following links connect to TI community resources. Linked contents are provided "AS IS" by the respective contributors. They do not constitute TI specifications and do not necessarily reflect TI's views; see TI's Terms of Use.

TI E2E™ Online Community ***TI's Engineer-to-Engineer (E2E) Community.*** Created to foster collaboration among engineers. At e2e.ti.com, you can ask questions, share knowledge, explore ideas and help solve problems with fellow engineers.

Design Support ***TI's Design Support*** Quickly find helpful E2E forums along with design support tools and contact information for technical support.

12.3 Trademarks

E2E is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

12.4 Electrostatic Discharge Caution

These devices have limited built-in ESD protection. The leads should be shorted together or the device placed in conductive foam during storage or handling to prevent electrostatic damage to the MOS gates.

12.5 Glossary

SLYZ022 — *TI Glossary*.

This glossary lists and explains terms, acronyms, and definitions.

13 Mechanical, Packaging, and Orderable Information

The following pages include mechanical, packaging, and orderable information. This information is the most current data available for the designated devices. This data is subject to change without notice and revision of this document. For browser-based versions of this data sheet, refer to the left-hand navigation.

PACKAGING INFORMATION

Orderable Device	Status (1)	Package Type	Package Drawing	Pins	Package Qty	Eco Plan (2)	Lead/Ball Finish (6)	MSL Peak Temp (3)	Op Temp (°C)	Device Marking (4/5)	Samples
L293DNE	ACTIVE	PDIP	NE	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type	0 to 70	L293DNE	Samples
L293DNEE4	ACTIVE	PDIP	NE	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type	0 to 70	L293DNE	Samples
L293DWP	OBSOLETE	SOIC	DW	28		TBD	Call TI	Call TI	0 to 70	L293DWP	
L293DWPG4	OBSOLETE	SOIC	DW	28		TBD	Call TI	Call TI	0 to 70		
L293DWPTR	OBSOLETE	SO PowerPAD	DWP	28		TBD	Call TI	Call TI	0 to 70		
L293N	OBSOLETE	PDIP	N	16		TBD	Call TI	Call TI	0 to 70	L293N	
L293NE	ACTIVE	PDIP	NE	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type	0 to 70	L293NE	Samples
L293NEE4	ACTIVE	PDIP	NE	16	25	Pb-Free (RoHS)	CU NIPDAU	N / A for Pkg Type	0 to 70	L293NE	Samples
L293NG4	OBSOLETE	PDIP	N	16		TBD	Call TI	Call TI	0 to 70		

(1) The marketing status values are defined as follows:
ACTIVE: Product device recommended for new designs.
LIFEBUY: TI has announced that the device will be discontinued, and a lifetime-buy period is in effect.
NRND: Not recommended for new designs. Device is in production to support existing customers, but TI does not recommend using this part in a new design.
PREVIEW: Device has been announced but is not in production. Samples may or may not be available.
OBSOLETE: TI has discontinued the production of the device.

(2) Eco Plan - The planned eco-friendly classification: Pb-Free (RoHS), Pb-Free (RoHS Exempt), or Green (RoHS & no Sb/Br) - please check http://www.ti.com/productcontent for the latest availability information and additional product content details.
TBD: The Pb-Free/Green conversion plan has not been defined.
Pb-Free (RoHS): TI's terms "Lead-Free" or "Pb-Free" mean semiconductor products that are compatible with the current RoHS requirements for all 6 substances, including the requirement that lead not exceed 0.1% by weight in homogeneous materials. Where designed to be soldered at high temperatures, TI Pb-Free products are suitable for use in specified lead-free processes.
Pb-Free (RoHS Exempt): This component has a RoHS exemption for either 1) lead-based flip-chip solder bumps used between the die and package, or 2) lead-based die adhesive used between the die and leadframe. The component is otherwise considered Pb-Free (RoHS compatible) as defined above.
Green (RoHS & no Sb/Br): TI defines "Green" to mean Pb-Free (RoHS compatible), and free of Bromine (Br) and Antimony (Sb) based flame retardants (Br or Sb do not exceed 0.1% by weight in homogeneous material)

(3) MSL, Peak Temp. - The Moisture Sensitivity Level rating according to the JEDEC industry standard classifications, and peak solder temperature.

(4) There may be additional marking, which relates to the logo, the lot trace code information, or the environmental category on the device.

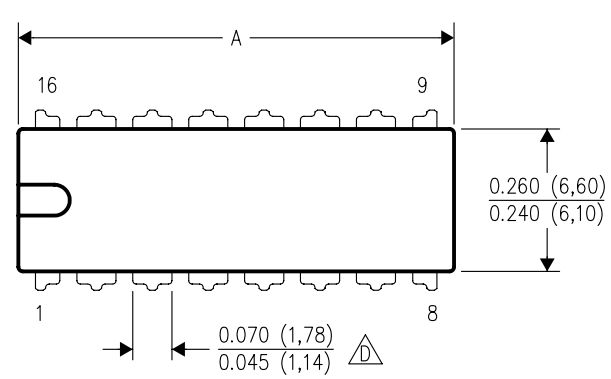
(5) Multiple Device Markings will be inside parentheses. Only one Device Marking contained in parentheses and separated by a "~" will appear on a device. If a line is indented then it is a continuation of the previous line and the two combined represent the entire Device Marking for that device.

(6) Lead/Ball Finish - Orderable Devices may have multiple material finish options. Finish options are separated by a vertical ruled line. Lead/Ball Finish values may wrap to two lines if the finish value exceeds the maximum column width.

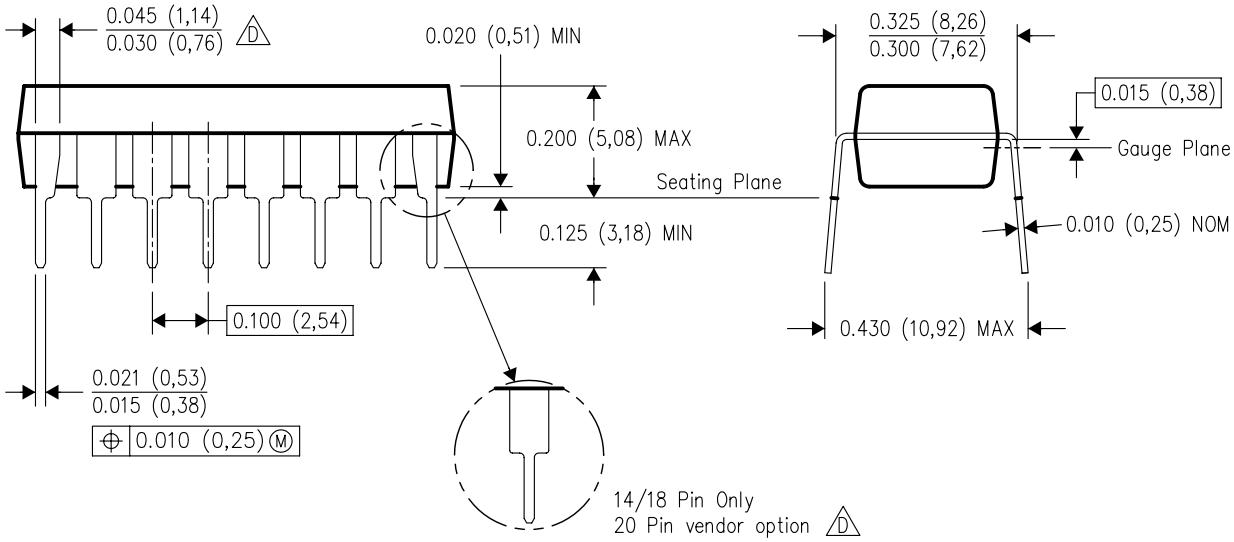
N (R–PDIP–T**)

PLASTIC DUAL–IN–LINE PACKAGE

16 PINS SHOWN



DIM \ PINS **	14	16	18	20
A MAX	0.775 (19,69)	0.775 (19,69)	0.920 (23,37)	1.060 (26,92)
A MIN	0.745 (18,92)	0.745 (18,92)	0.850 (21,59)	0.940 (23,88)
MS–001 VARIATION (C)	AA	BB	AC	AD

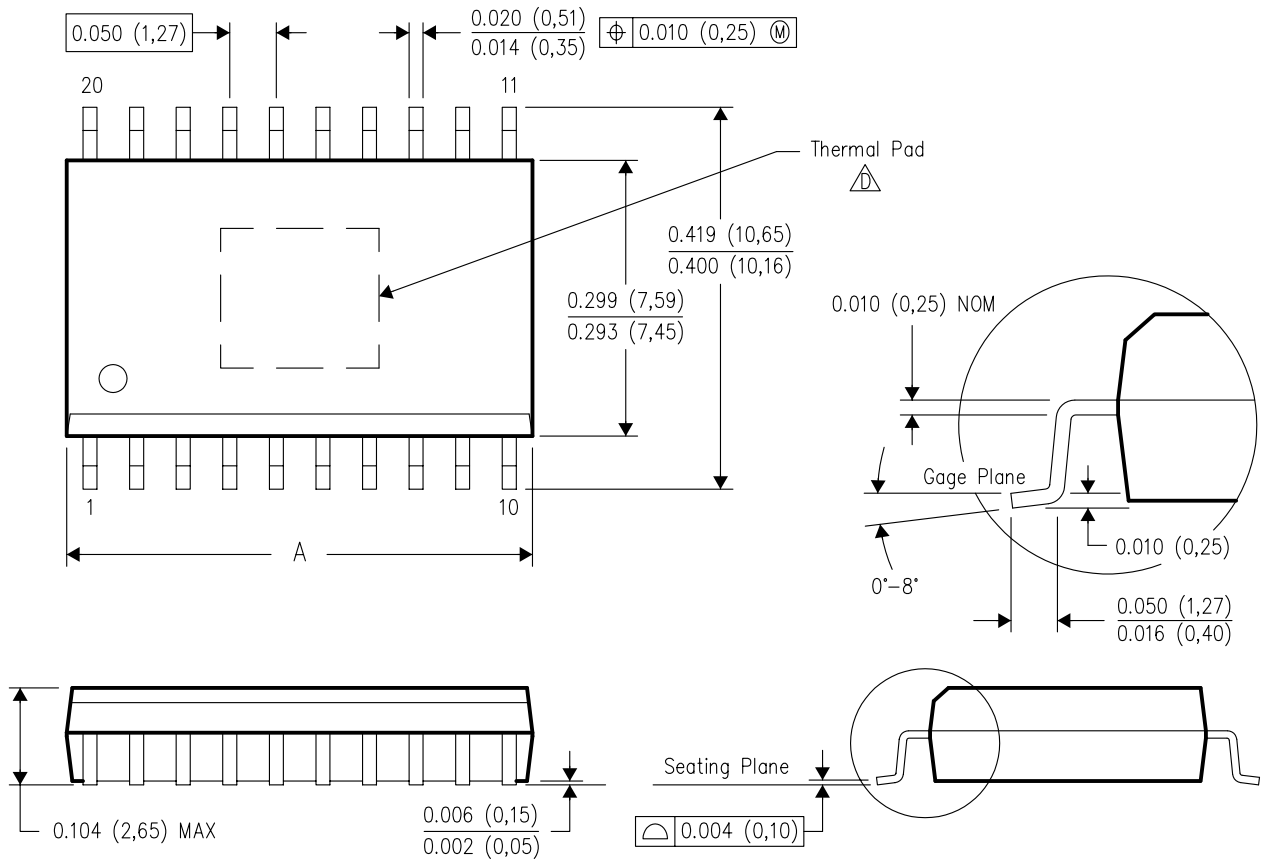


4040049/E 12/2002

NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Falls within JEDEC MS–001, except 18 and 20 pin minimum body length (Dim A).
D. The 20 pin end lead shoulder width is a vendor option, either half or full width.

DWP (R–PDSO–G**) PowerPAD™ PLASTIC SMALL–OUTLINE PACKAGE

20 PINS SHOWN

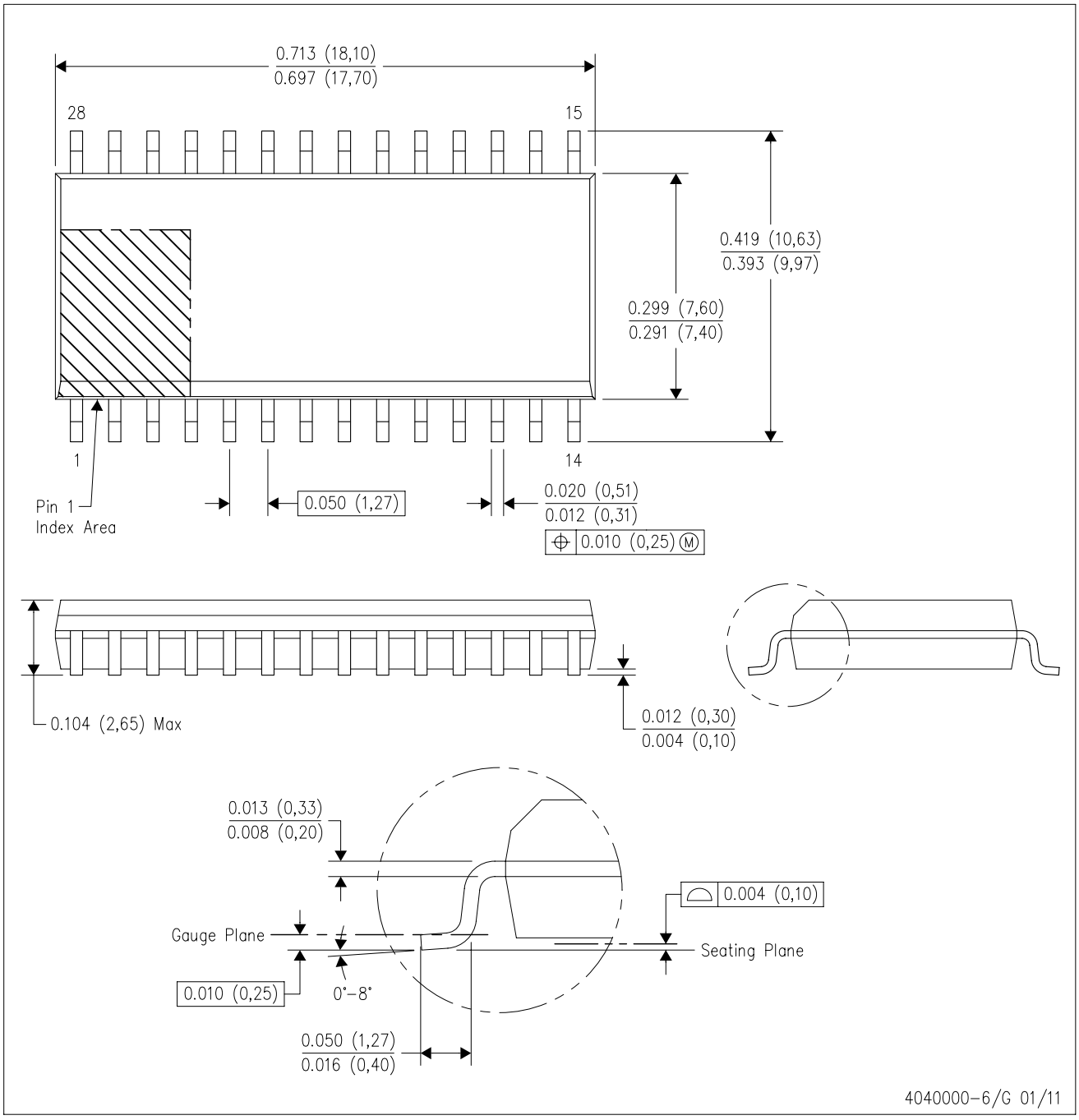


DIM \ PINS **	16	20	24	28
A MAX	0.410 (10,41)	0.510 (12,95)	0.610 (15,49)	0.710 (18,03)
A MIN	0.400 (10,16)	0.500 (12,70)	0.600 (15,24)	0.700 (17,78)

4147575/C 02/05

NOTES:
A. All linear dimensions are in inches (millimeters).
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. This package is designed to be soldered to a thermal pad on the board. Refer to Technical Brief, PowerPad Thermally Enhanced Package, Texas Instruments Literature No. SLMA002 for information regarding recommended board layout. This document is available at www.ti.com <http://www.ti.com>. See the product data sheet for details regarding the exposed thermal pad dimensions.

DW (R–PDSO–G28) PLASTIC SMALL OUTLINE



NOTES:
A. All linear dimensions are in inches (millimeters). Dimensioning and tolerancing per ASME Y14.5M–1994.
B. This drawing is subject to change without notice.
C. Body dimensions do not include mold flash or protrusion not to exceed 0.006 (0,15).
D. Falls within JEDEC MS–013 variation AE.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265