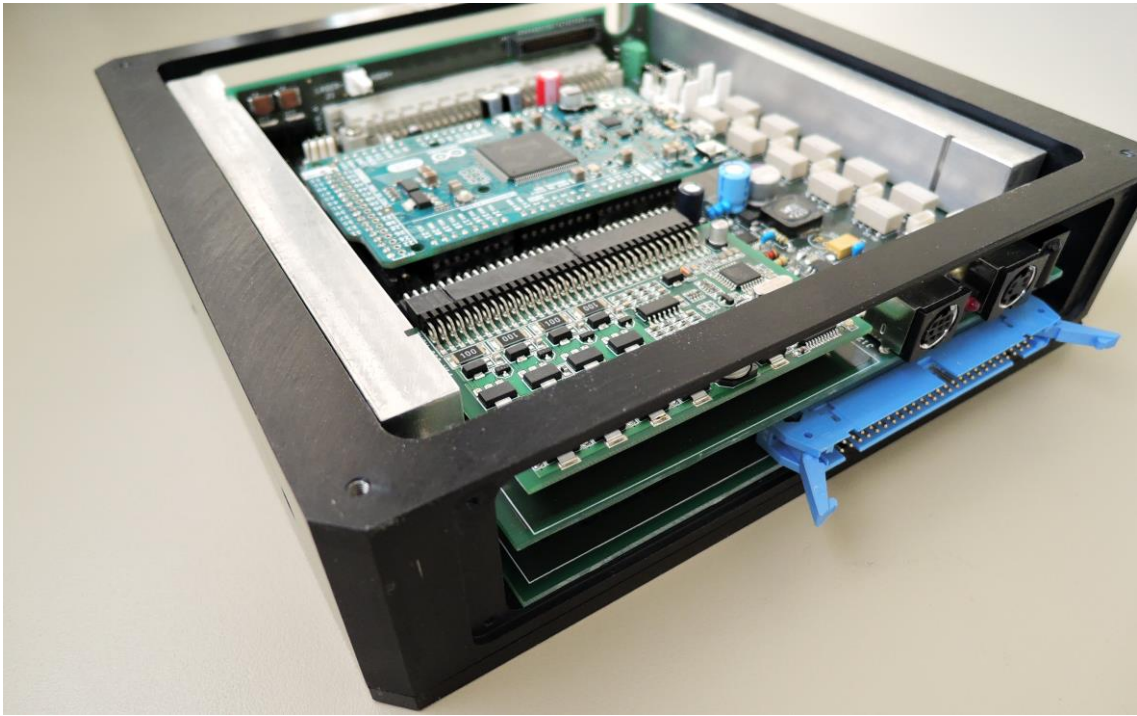


BASE Y CABEZASPM CON MOTORES PIEZOELECTRICOS Y ARDUINO DUE

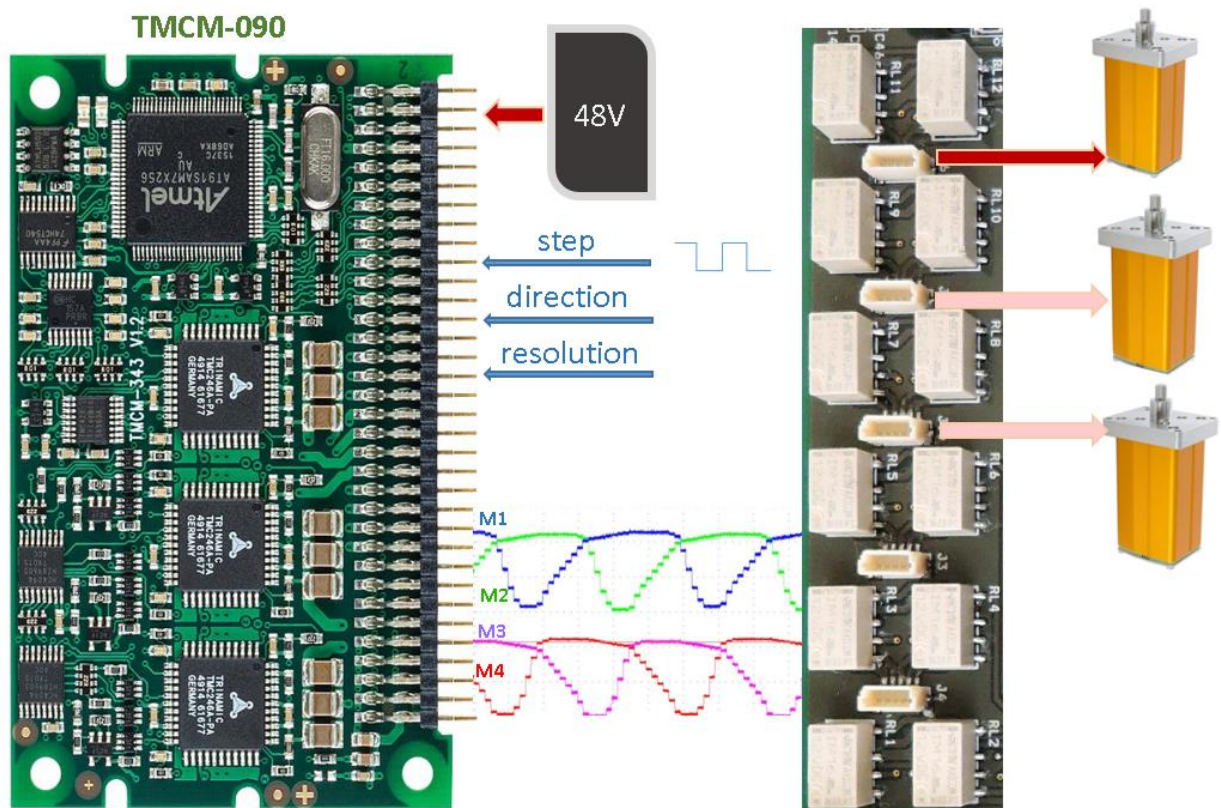


Hardware del sistema

El propósito de la base y cabeza para SPM es el de automatizar el posicionado del fotodiodo y el haz láser, así como la aproximación del sistema de medida con la muestra. El sistema se controla mediante comandos que pueden enviarse desde un PC, tablet o smartphone.

Los movimientos se realizan mediante motores piezoeléctricos de [PiezoMotor](#) y el módulo [TMCM-090](#), un driver de un solo eje (capaz de mover un solo motor de este tipo).

Al módulo TMCM-090 hay que proporcionarle tensión de alimentación de 5V para la lógica y otra tensión de 48V para la potencia de los motores. Además de señales de control para que genere cuatro señales de potencia que mueven el motor.



Las señales de potencia son dos pares M1, M2 y M3, M4 de 48 voltios pico a pico, periódicas y desfasadas entre sí de forma apropiada. Un periodo de onda completo es un paso. Una fracción de periodo es un micropaso. Un periodo se puede dividir en 256, 512, 1024 o 2048 micropasos, esta es la “resolución” del sistema. “Frecuencia de

onda” es el número de periodos por segundo y la “frecuencia de micropaso” es el número de periodos por segundo multiplicado por la resolución. A partir de ahora cuando aludamos a frecuencia debemos entender “frecuencia de micropaso”. Ya que el movimiento se hace en micropasos y es el parámetro relevante del sistema.

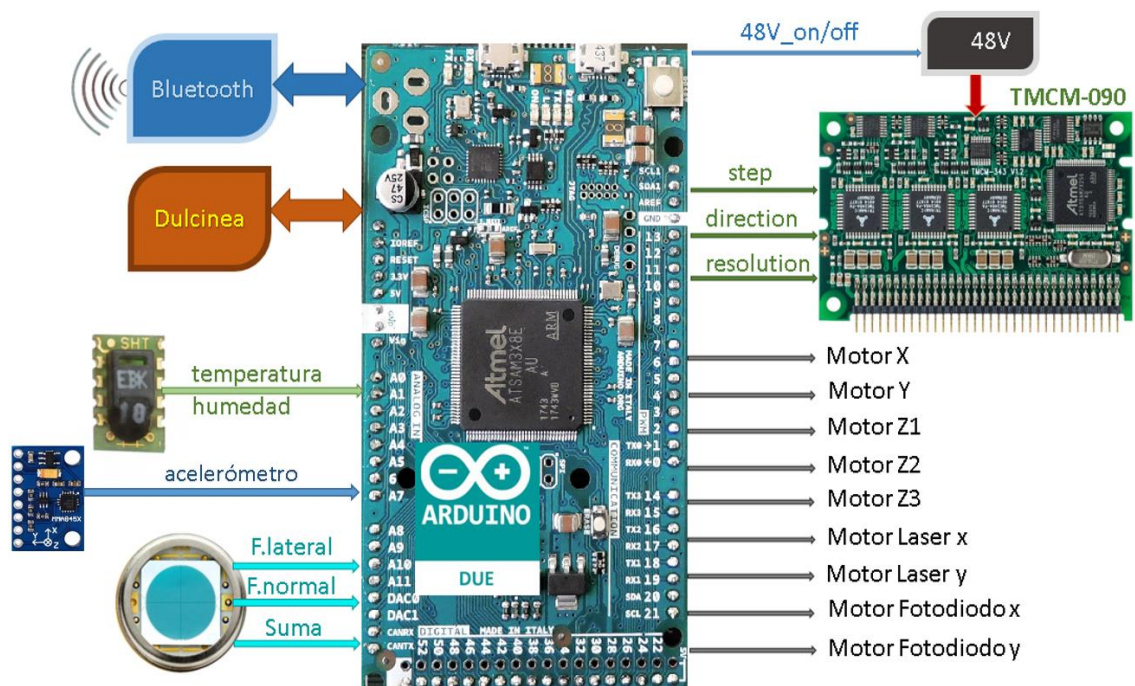
Las señales de control del driver TCM-090 son:

Step: Cada pulso recibido hace que el piezomotor avance un micropaso. Su frecuencia es la “frecuencia de micropaso”.

Direction: Determina la dirección del movimiento. Los motores se mueven en dos sentidos, arriba o abajo, izquierda o derecha, según su función.

Resolution: fracciones en que se divide un periodo completo. A mayor resolución menor es la distancia que recorre el piezomotor con cada micropaso y viceversa.

Wave: El driver puede generar 4 tipos diferentes de onda. Actualmente se utiliza un solo tipo de onda por lo que no entraremos en detalles.



Este módulo es para un solo motor, pero en el sistema hay 3 motores en el eje Z, dos para el fotodiodo y otros dos para el láser más dos auxiliares x e y son en total 7 motores. La solución aplicada es multiplexar las 4 señales de potencia de salida a los

motores mediante relés que conectan al módulo un solo motor o un conjunto de ellos en cada momento.

Para darle la funcionalidad deseada, la base incorpora un módulo Arduino DUE que monta un microcontrolador ATSAM3X8E de 32 bits y 84MHz, que genera las señales de control del módulo TMCM-090, activar su alimentación de 48V y seleccionar el motor apropiado en cada momento.

Arduino DUE además debe de encargarse de la lectura del sensor de humedad-temperatura, adquirir las señales de fuerza normal, fuerza lateral y suma del fotodiodo de cuatro cuadrantes, leer la posición de un acelerómetro MMA8452 de 12 bits y 3 ejes, recibir comandos a través de Dulcinea y a mediante Bluetooth. También lee los bits del DSP que llegan a través del conector de Dulcinea. Estos bits son DSP_48V y DSP_CLK que funcionan como sigue:

DSP_CLK	Si el parámetro frecuencia es 0, un pulso DSP_CLK provoca un micropaso si hay un motor activo.
DSP_48V=0	La alimentación de 48V de potencia del módulo desactivada
DSP_48V=1	La alimentación de 48V de potencia del módulo no desactivada

En la placa del circuito hay 4 leds. Uno rojo está en el frontal y se enciende mientras se ejecuta algún comando. Otros 3 están sobre la PCB.

Led rojo: Está encendido cuando los 48V están activados.

Led ámbar: Está encendido cuando hay algún motor en movimiento.

Led azul: Para test.

Software del Arduino DUE

El Arduino DUE que monta la base está programado con un firmware para darle la funcionalidad deseada mediante un conjunto de funciones, que podemos dividir en 4 grupos:

1. De comunicación; Son las que procesan los comandos que envía el PC a través de Dulcinea y del Bluetooth. Estas a su vez llaman a las funciones básicas cuando sea necesario.
2. Básicas; son las encargadas de actuar físicamente sobre el hardware del sistema, como cambiar los pines step, dirección, resolución del módulo TMCM-

090, activar y desactivar la alimentación de 48V para la potencia de dicho módulo, realizar el multiplexado de motores, leer el sensor de humedad - temperatura y el acelerómetro (no implementado actualmente), muestrear las señales del fotodiodo y procesar las señales del DSP. También cambia las variables de estado del sistema como la frecuencia, la resolución, el número de pasos a dar, el motor activo, el estado de marcha o paro.

3. De Interrupción; Cuando el pin conectado a la señal DSP_CLK recibe un pulso, se ejecuta una función de interrupción. Los pulsos de step para avanzar un micropaso se generan mediante un timer que se programa con un tiempo determinado y cuando se cumple, el timer genera una interrupción y se ejecuta una función de interrupción. Lo mismo sucede con otro timer cuya función es muestrear las señales analógicas del fotodiodo.
4. Funciones de test; En general activan o desactivan un solo pin del Arduino DUE. Se utilizan para testear el sistema.

Funciones de comunicación

Son las más importantes. Responden a comandos enviados por el PC a través de Dulcinea o del mando y a su vez pueden ejecutar las funciones básicas. Los comandos se describen en detalle más adelante.

void pc_marcha_motor(void);

Esta función se ejecuta cuando el PC solicita a la base que un determinado motor se ponga en marcha a una frecuencia, resolución y sentido programados en el mismo comando.

void pc_marcha_paro(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar el estado de marcha a paro y viceversa.

void pc_sentido(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar el sentido de movimiento del motor activo.

void pc_frecuencia(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar el valor de la frecuencia.

void pc_resolucion(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar la resolución

void pc_anda_numero_de_pasos(void);

Actualiza o devuelve el valor de los pasos que quedan por dar. Si se le da un valor de 0 (valor por defecto) tras una instrucción de marcha el motor no para hasta que no reciba una instrucción de paro. Pero si se actualiza a un valor distinto de cero, en cada paso se descontará en uno y cuando alcance el valor de cero el motor en marcha se parará.

void pc_motor_activo(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar el motor activo.

void pc_sensor_temperatura_humedad(void);

Esta función se ejecuta cuando el PC solicita a la base el valor de las señales del fotodiodo. La base responde con una cadena que contiene números reales con sus valores.

void pc_fotodiodo(void);

Esta función se ejecuta cuando el PC solicita a la base el valor de las señales del fotodiodo. La base responde con una cadena que contiene números reales con sus valores.

void pc_onda(void);

Esta función se ejecuta cuando el PC solicita a la base cambiar el motor activo. Pero está desactivada porque solo se utiliza un tipo de onda.

void pc_contador(void);

Actualiza o devuelve el valor de un contador. El contador cuenta los pasos del motor en marcha. Se pone a cero cada vez que se pone un motor en marcha.

void pc_reset(void);

Pone la base en el estado inicial.

void pc_variables(void);

La base envía al PC los valores de las variables de estado del sistema: motor activo, resolución, frecuencia, sentido, pasos, estado de marcha o paro y onda.

void scpi_error(void);

El PC solicita a la base el último error que se ha producido. En el proceso de comunicación entre la base y el PC o en el funcionamiento usual del sistema se pueden producir errores; como por ejemplo intentar dar un valor no permitido a una variable. Cuando esto sucede se anota un error en una pila FIFO de errores que el PC puede

consultar. Cuando un error es leído por el PC se retira de la pila. Si no se leen los errores y la pila se sobrepasa, los errores anotados se sobrescriben. La profundidad de la pila es de 16.

Void scpi_idn(void);

Envía al PC la identificación del sistema "Base SPM".

Void scpi_version(void);

Envía al PC la versión del software "Base SPM VX.X".

Void scpi_opc(void);

Envía al PC el carácter uno '1'; Se utiliza para buscar la base en los puertos del PC.

void scpi_cls(void);

Borra la pila de errores.

void bluetooth_marcha_motor(void);

Función que está definida para comunicar con el mando a través de Bluetooth.

Programa un motor, una frecuencia, sentido y número de pasos (si es cero da infinitos pasos) y pone el motor en marcha. La resolución se programa a 256.

void bluetooth_para_motor(void);

Igual que la anterior está definida para el mando. Detiene el motor que está en marcha y envía el carácter '1' al mando para que este verifique que se ha recibido el comando.

void bluetooth_estado(void);

Como las funciones anteriores está pensada para el mando. Devuelve al PC una cadena de caracteres que contiene 3 números reales con el valor en voltios de la "Fuerza Normal", "Fuerza Lateral" y "Suma", una variable entera que puede valer 10 (indica que hay un motor en marcha) o 5 (indica que no hay ningún motor en marcha), y finalmente otro entero con el valor del número de pasos que quedan por dar.

Funciones básicas implementadas en el Arduino DUE

Estas funciones son llamadas por las funciones de comunicación.

int cambia_onda(unsigned int);

Esta función esta por posibles cambios futuros, ya que actualmente el modo de onda es siempre el mismo.

int cambia_frecuencia_resolucion(unsigned int ,unsigned int);

Cambia la frecuencia y la resolución del módulo TMCM-090 (ver el apartado de descripción de comandos para más detalles). Ambos parámetros se cambian en una sola función porque primero hay comprobar si son compatibles entre sí. Si la frecuencia es 0, los pulsos de micropasos serán los que lleguen por la señal DSP_STEP.

int cambia_motor(unsigned int);

Cambia los relés para que las señales de potencia del módulo TMCM-090 actúen sobre el motor que queramos mover.

int cambia_sentido(unsigned int);

Cambia el pin de sentido del módulo TMCM-090.

int marcha_paro_motor(unsigned int);

Hace que la señal de step del módulo TMCM-090 reciba pulsos para que avance al motor seleccionado.

void activa_48V(void);

Activa la alimentación de 48V de potencia del módulo TMCM-090.

void desactiva_48V(void);

Desactiva la alimentación de 48V de potencia del módulo TMCM-090.

void programa_pasos(int);

Programa en un contador el número de micropasos que se quieren dar. Si se escribe 0 da micropasos hasta que se reciba un comando de parada.

Funciones de interrupción

void timer_CLK(void);

En función de la frecuencia programada, Arduino DUE programa un timer interno que genera una interrupción periódicamente que ejecuta esta función. En ella se genera un pulso en el pin "step" de Arduino DUE.

void clk_externo(void);

Esta función se ejecuta cada vez que Arduino DUE recibe un pulso (en el flanco de bajada) por el pin DSP_CLK. En ella se genera un pulso en el pin "Step" de Arduino DUE.

void timer_ADC(void);

Arduino DUE adquiere las señales del fotodiodo de cuatro cuadrantes, “Fuerza normal”, “Fuerza lateral”, y “Suma” con un periodo de muestreo constante. Para eso programa un timer interno que genera una interrupción cada 500 microsegundos. Para cada señal tiene una pila de 64 muestras y cuando el PC le solicita el valor con el comando correspondiente, hace una media de las 64 muestras, las convierte en un valor real equivalente a voltaje y las envía en una cadena de caracteres.

Funciones de test

Son de menor interés, sin entrar en detalles de cada función, pone en estado alto 1 o bajo 0 cada pin funcional del sistema. Se utilizan en fase de desarrollo.

```
void desactiva_rele_z1(void);
void activa_rele_z1(void);
void desactiva_rele_z2(void);
void activa_rele_z2(void);
void desactiva_rele_z3(void);
void activa_rele_z3(void);
void desactiva_rele_y(void);
void activa_rele_y(void);
void desactiva_rele_x(void);
void activa_rele_x(void);
void desactiva_rele_hd(void);
void activa_rele_hd(void);
void desactiva_led_3(void);
void activa_led_3(void);
void desactiva_led_2(void);
void activa_led_2(void);
void desactiva_led_1(void);
void activa_led_1(void);
void desactiva_led_0(void);
void activa_led_0(void);
void desactiva_res_1(void);
void activa_res_1(void);
void desactiva_res_0(void);
void activa_res_0(void);
void activa_mode_0(void);
void desactiva_mode_0(void);
void activa_mode_1(void);
void desactiva_mode_1(void);
void activa_dir(void);
void desactiva_dir(void);
```

```

void activa_clk(void);
void desactiva_clk(void);
void desactiva_motor_head_1(void);
void activa_motor_head_1(void);
void desactiva_motor_head_0(void);
void activa_motor_head_0(void);
void desactiva_i22(void);
void activa_i22(void);
void desactiva_i21(void);
void activa_i21(void);
void desactiva_i12(void);
void activa_i12(void);
void desactiva_i11(void);
void activa_i11(void);
void mueve_motor(void);
void para_motor(void);
void modo_depuracion_no(void);
void modo_depuracion_si(void);

```

Las dos últimas funciones ponen un flag interno a 0 (modo funcionamiento normal) o 1 (modo depuración) y se utiliza en fase de desarrollo, para enviar cadenas por el puerto serie con información relevante sobre el estado del sistema.

Comunicación con el PC y el mando

La base comunica con un PC a través de Dulcinea y mediante Bluetooth, con una aplicación para Android que actúa como mando. Para eso Arduino DUE utiliza sus puertos [serie](#).

Arduino DUE inicializa el puerto “Serial1” para comunicar con la PC a través de Dulcinea con la siguiente configuración.

PARAMETRO	VALOR
Bits de datos	8
Paridad	No
Bits de stop	1
Baudrate	57600 bps

Arduino DUE inicializa el puerto “Serial2” para comunicar a través del módulo Bluetooth HC-05 (o compatible) con la siguiente configuración.

PARAMETRO	VALOR
Bits de datos	8
Paridad	No
Bits de stop	1
Baudrate	9600 bps

Aunque en el caso del Bluetooth el baudrate no es relevante, ya que es el baudrate es local, entre Arduino DUE y el dispositivo HC-05 de la base que convierte RS232 a señales de radio bajo protocolo Bluetooth.

Arduino DUE devuelve valores desde la base mediante la función de librería de la plataforma Arduino:

[Serial.println](#)(Variable);

Para procesar la recepción de la variable correctamente, conviene saber que esta función imprime datos en el puerto serie como texto ASCII seguido de un carácter de retorno de carro (ASCII 13 o '\r') y un carácter de nueva línea (ASCII 10 o '\n').

Los comandos que el PC debe enviar a Arduino para controlar la base, son cadenas de caracteres terminadas con retorno de carro '\r'. Pueden tener de ninguno a varios parámetros. Los parámetros van separados del comando y entre sí por el carácter "espacio". Algunos comandos tienen como parámetro el carácter '?' sin espacio o con espacio, su función es la de pedir a la base el o los parámetros actuales relativos a ese comando. A continuación, se ve todo esto en detalle.

Comando para poner en marcha un motor con una resolución, velocidad y sentido determinados

Este comando está pensado para programar las variables, seleccionar un motor y ponerlo en marcha con un solo comando. Otros comandos programan solo una variable individualmente.

MOT:MM <Motor> <Resolución> <Frecuencia> <sentido>

Motor es un entero que puede valer:

Motor	Motor seleccionado
0	Ningún motor
1	Z1

2	Z2
3	Z3
4	Z1 y Z2
5	Z1 y Z3
6	Z2 y Z3
7	Z1, Z2 y Z3
8	X
9	Y
10	Fotodiodo X
11	Laser X
12	Laser Y
13	Fotodiodo Y

El sistema solo puede tener un motor activo (salvo los motores de 4 a 7 que en realidad activa un conjunto de ellos) o ninguno.

Velocidad: es un entero que puede valer entre 0 y 100 (de 0 a 100 mil pasos por segundo pps). La velocidad máxima es de 100 mil pps (se ha bajado respecto a la versión anterior de la base, donde era de 200 mil pps). Cuando la frecuencia se programa a 0, es el DSP el que debe proporcionar los pulsos , para mover el motor seleccionado, a través de Dulcinea.

Resolución: es un entero, debe ser compatible con la velocidad.

Resolución	velocidad
256	$0 < \text{velocidad} < 61$
512	$0 < \text{velocidad} < 100$
1024	$0 < \text{velocidad} < 100$
2048	$0 < \text{velocidad} < 100$

Sentido: Entero que puede valer 0 ó 1.

Ejemplo:

MOT:MM 12 256 30 1

Para poner en marcha el LaserY con resolución 256 a 30 pps. Sentido subida.

Este comando se puede utilizar par a leer las variables:

MOT:MM?

El PC devuelve en una cadena los valores de los parámetros actuales de la base; motor, resolución, frecuencia y sentido.

“MM Motor Resolución Frecuencia Sentido ”

MM son dos caracteres que sirven para identificar el comando a que responde la cadena.

La velocidad real de desplazamiento del piezomotor la determina la “frecuencia de onda”

$$Frecuencia\ de\ onda = \frac{Frecuencia}{Resolución}$$

Aunque hay motores que soportan frecuencias de onda de 3KHz, por seguridad y durabilidad de los motores, en este sistema no puede ser superior a 234Hz.

$$0 \leq Frecuencia\ de\ onda \leq 234Hz$$

Si el comando solicita una velocidad mayor de 60 mil pps y una resolución de 256, reduce la velocidad a 60 mil pps.

Comando para leer el valor de las variables programadas actualmente en la base

MOT:VAR?

La base devuelve una cadena de caracteres con los caracteres VAR para identificar la cadena y a continuación 7 enteros con las variables del sistema: motor activo, resolución, frecuencia, sentido, pasos por dar, estado de marcha o paro y Onda.

Ejemplo de respuesta del sistema: “VAR 7 2048 10 1 3000 1 3”

Comando para poner en marcha o parar el motor seleccionado

MOT:MP <marcha_paro>

marcha_paro: Entero que puede valer 0 ó 1. 0 paro, 1 marcha.

Ejemplo:

MOT:MP 1

Para poner en marcha el motor actual.

MOT:MP?

El PC devuelve el valor actual del parámetro.

La base también responde al comando:

MOT:MP ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con los caracteres MP para identificarse, un espacio el parámetro:

“MP 0” o “MP 1”.

Comando que programa un número de pasos

MOT:AN <Pasos>

Pasos: entero que puede valer entre 0 y 400000.

Ejemplos:

MOT:AN 100000

Programa cien mil pasos.

MOT:AN?

La base devuelve un entero con el número de pasos que queda por dar en una cadena de caracteres.

La base también responde al comando:

MOT:AN ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con el identificador AN, un espacio y el estado del parámetro “pasos”:

“AN Pasos” por ejemplo “AN 3500”.

La cuenta de pasos es descendente. El contador de pasos se programa con el valor deseado y resta uno con cada paso que da, cuando llega a cero detiene el motor. Para mover un motor sin límite, el número de pasos que hay que programar es 0. El valor por defecto es cero, luego, desde el estado inicial, si queremos mover el motor sin límite, no hay que programar este parámetro a 0 porque ya lo está.

Comando para programar el motor activo

MOT:MA <Motor>

Motor: (ver detalles en el comando MOT:MM).

Ejemplos:

MOT:MA 1

Programa el motor Z1 como activo.

MOT:MA?

El sistema devuelve el motor activo.

También se puede utilizar el comando:

MOT:MA ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con MA, un espacio y un entero con el número del motor activo.

Por ejemplo "MA 7".

Determina una frecuencia de pasos por segundo

MOT:FR <Frecuencia>

Frecuencia: (ver detalles en el comando MOT:MM).

Ejemplo:

MOT:FR 10

Programa una frecuencia de 10 mil pps. Por tanto la "frecuencia de onda" será 10000/Resolución.

MOT:FR?

El sistema devuelve la frecuencia actual.

También se puede utilizar el comando:

MOT:FR ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con FR, un espacio y un entero con la frecuencia.

Por ejemplo "FR 10".

Cuando se programa una frecuencia de 0, es el DSP el que proporciona los pulsos, a través de Dulcinea, para mover el motor seleccionado.

Comando que determina una resolución

MOT:RE <Resolución>

Resolución: (ver detalles en el comando MOT:MM).

Ejemplo:

MOT:RE 1024

Programa una resolución de 1024. Por tanto la “frecuencia de onda” será Frecuencia/1024.

MOT:RE?

El sistema devuelve la resolución actual en una cadena de caracteres.

También se puede utilizar el comando:

MOT:RE ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con RE, un espacio y un entero con la resolución.

Por ejemplo “RE 2048”.

Comando para establecer el sentido del movimiento

MOT:SE <Sentido>

Sentido: Entero que puede valer 0 ó 1. 0 bajar, 1 subir.

Ejemplo:

MOT:SE 1

Establece el sentido de subida.

MOT:SE?

El PC devuelve el sentido.

También se puede utilizar el comando:

MOT:SE ? (con un espacio ante la interrogación).

Devuelve una cadena de caracteres con SE, un espacio y un entero 0 o 1.

Por ejemplo “SE 1”.

Comando para pedir a la base los valores de temperatura y humedad del sensor SHT-11/DHT-22

MOT:TH?

La base devuelve una cadena con el siguiente formato:

"Ttemperatura Hhumedad"

El carácter T seguida de un float con el valor de la temperatura, espacio, el carácter H seguido de un float con el valor de la humedad.

Comando para pedir a la base los valores de las señales FN, FL y SUM

MOT:FOT?

La base devuelve una cadena con el siguiente formato:

"FOT fuerza_normal fuerza_lateral suma"

Tres caracteres "FOT" espacio, un float con el valor de la señal fuerza normal, espacio, un float con el valor de la señal fuerza lateral, espacio y float con el valor de la señal suma.

Comando para pedir a la base los valores del acelerómetro de 3 ejes.

MOT:AC?

La base devuelve una cadena con el siguiente formato:

"AC gx gy gz"

Dos caracteres "AC" espacio, y 3 floats (reales) con los valores de la gravedad media en cada eje. Los rangos de g son entre -1 y 1 g (9,81m/s/s).

Comando para pedir la versión del software

MOT: VER?

La base devuelve "Base SPM VX.Y" Donde X.Y es la versión del software de Arduino DUE.

Comandos incluidos por compatibilidad pero que no hacen nada

MOT:HF y MOT:FE

Comando para poner en marcha un motor a una velocidad determinada, con un sentido y un número de pasos

BTH:MARCHA <Motor> <Velocidad> <sentido> <pasos>

Este comando está pensado para la aplicación Android que funciona como mando.

Motor: Es un entero que puede valer:

Motor	Motor seleccionado
0	Ningún motor
1	Z1
2	Z2
3	Z3
4	Z1 y Z2
5	Z1 y Z3
6	Z2 y Z3
7	Z1, Z2 y Z3
8	X
9	Y
10	Fotodiodo X
11	Laser X
12	Laser Y
13	Fotodiodo Y

Velocidad: es un entero que puede valer 1 , 5, 15, 30, 60.

Velocidad	Frecuencia de pasos
1	1000 pps
5	5000 pps
15	15000 pps
30	30000 pps
60	60000 pps

Sentido: Entero que puede valer 0 ó 1. 0 bajar, 1 subir.

Pasos: Entero que puede valer entre 0 y 300000. Si vale 0 el motor no se parará hasta que la base no reciba un comando de parada.

Un ejemplo de comando seria:

BTH:MARCHA 12 30 1 100000

Motor "Laser Y" a 30000 pps sentido subida y 100 mil pasos.

Comando para parar el motor que esté en movimiento

BTH:PARO

Este comando está pensado para la aplicación Android que funciona como mando.

Al recibir este comando la base parará cualquier motor que esté en marcha y enviará al mando un carácter '1', para indicar que ha recibido el comando y ha detenido el motor.

Comando para pedir el estado de las señales FL, FN y SUM a la base

Este comando está pensado para la aplicación Android que funciona como mando.

BTH:EST

Al recibir este comando la base devuelve una cadena con el siguiente contenido:

"EST FN FL SUM EMP pasos"

EST	Tres caracteres (tipo char) para identificar la cadena de estado
FN	Tipo float con el valor en voltios de la señal "fuerza normal"
FL	Tipo float con el valor en voltios de la señal "fuerza lateral"
SUM	Tipo float con el valor en voltios de la señal "suma"
EMP	Tipo int: 10 =" motor en marcha" 5="motor parado"
pasos	Pasos que quedan por dar (si se programó esta variable)

Si se programó la variable "pasos" con el comando BTH:MARCHA a un valor distinto de 0 ,cuando un motor da un paso esta variable se decrementa y cuando llega a 0 detiene el motor. Pero si se programó a 0 el motor no se detiene hasta recibir un comando de paro.

Comando ERR

Devuelve una cadena que indica si ha habido un error en la comunicación o en la recepción errónea de un comando o parámetro fuera de rango etc. Devuelve el último error introducido en la pila de errores. Estas cadenas son:

0	No hay errores
1	Carácter no valido
2	Comando desconocido
3	Cadena demasiado larga
4	Parámetro inexistente
5	Formato de parámetro no valido
6	Parámetro fuera de rango
7	Frecuencia de onda excesiva
8	Resolución incorrecta
9	Motor incorrecto
10	Modo de onda no permitido
11	Frecuencia y resolución incompatibles
12	Frecuencia incorrecta
13	Sentido incorrecto
14	Resolución ajustada
15	Número de pasos incorrectos
16	Frecuencia ajustada
17	Resolución incorrecta
18	Potencia desactivada por el DSP
19	Modo de onda incorrecto
20	No se permite cambiar de onda
21	Error lectura sensor humedad-temperatura
22	No hay motor seleccionado

Comando CLS!

Los errores vistos en el comando anterior se van acumulando en una pila LIFO, de forma cuando se envía el comando ERR El sistema devuelve el último error que entró en la pila. El comando CLS! borra de la pila los errores acumulados.

Comando *IDN

Devuelve una cadena con la identificación del sistema. En este caso..

“Base SPM”

Nota final sobre comandos

Todos los comandos vistos pueden ser utilizados indistintamente a través de puerto serie o Bluetooth tanto por Tablet, Smartphone o PC.

Repositorios:

El firmware de Arduino DUE se puede bajar desde:

<https://github.com/PatricioCoronado/Base-AFM-Arduino-DUE>

Y Un software de PC para probar la base está en:

https://github.com/PatricioCoronado/BaseSPM_CVI

Esta aplicación se puede utilizar para conectar con la base a través del puerto USB o Bluetooth de la base y testear su funcionalidad.