CAPÍTULO 3. FASE I: KMS32. MEDIDOR DE ÁNGULO DE ROTACIÓN

3.1 INTRODUCCIÓN

En esta primera fase, como ya hemos dicho, se cambia el micro que existía en el sistema por otro más rápido. El resultado es un nuevo sistema para aplicaciones más exigentes, tanto de velocidad como de precisión. La migración de micro no terminó en una simple adaptación a las nuevas especificaciones del MSP430, sino que se mejoró el código, cambió la rutina matemática y se incorporaron nuevas funciones, como la posibilidad de usar varios registros de memoria para el conversor A/D o sacar datos de forma serie por la UART.

El nuevo sistema se comercializará de manera individual. Además, la base del nuevo programa será el usado en la Fase II del proyecto, cuando se une el medidor de ángulo de rotación al inclinómetro. Para ello, además se ha fabricado una nueva placa mucho más pequeña para facilitar su integración. Esa parte se ve en el punto 4.5.

Se trata pues, del desarrollo del software del microprocesador MSP430F133 de Texas Instrument para controlar y dar salida a los datos obtenidos por el sensor KMT32B.

El sensor KMT32B usa dos puentes de resistencias con los que obtenemos dos señales, una para el seno y otra para el coseno del doble del ángulo. El objetivo es analizar estas dos señales y por medio de una función calcular el ángulo final que nos está midiendo el sensor.

Las funciones a realizar por el micro son:

- controlar el Multiplexador que alterna las señales analógicas procedentes del sensor

- convertir de analógico a digital la señal de entrada proveniente de los A.O. (a su vez del Mux)

- calcular con esos resultados el ángulo final obtenido.

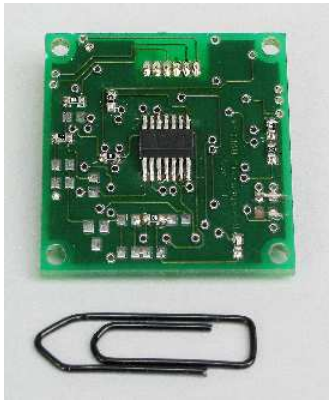
Paralelamente se le ha provisto de comunicación serie por UART para comunicarse con computadora y poder así regular el offset o ver las medidas finales.

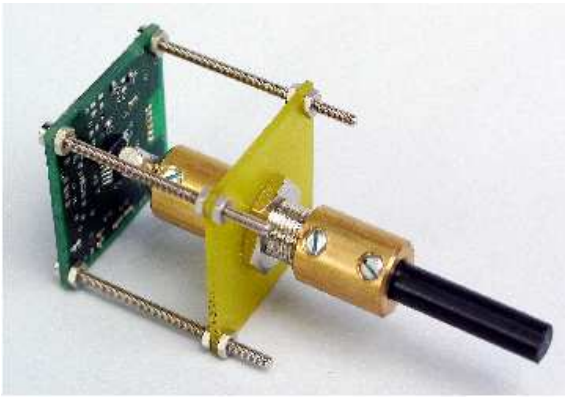
3.2 DESCRIPCIÓN DEL FUNCIONAMIENTO

El micro estará continuamente leyendo una señal por la entrada del ADC que será alternativamente el valor del sen(2α) y cos(2α). Para ello, alterna la salida del multiplexador que entrega las señales. Una vez leída la señal se produce una interrupción por el ADC que, tras tratarla, la almacena para su posterior uso tanto en el cálculo del ángulo como en la posibilidad de sacar en pantalla su valor para la calibración manual. Cabe la posibilidad de elegir el número de registros empleados en la conversión AD con la idea de poder aumentar la precisión de la medida (1-16 registros de memoria).

Una vez que tenemos el valor digital de las dos señales (sen y cos), se calcula el valor del ángulo final con la llamada a la función atan.

Al mismo tiempo, si se pulsa el botón ‘a’ o ‘r’ en el teclado del ordenador conectado al micro por puerto serie, se produce una interrupción de UART. Si hemos pulsado la tecla ‘r’ saca en pantalla el valor de las señales sen y cos (sin offset) para la calibración manual, cálculo del offset. Si hemos pulsado la tecla ‘a’ saca en pantalla el valor del ángulo medido.





3.3 PROGRAMA IMPLEMENTADO

Pasemos a continuación a ver el programa implementado, para lo cual lo dividiremos en 3 bloques básicos, el programa principal, las interrupciones y las rutinas secundarias, que serán utilizadas por el programa principal de forma lógica y ordenada para conseguir que el funcionamiento del sistema sea el deseado.

- módulo principal (con las interrupciones para el ADC y UART) - **KMS32MSP.c**
- módulo de inicio del ADC -**ADC_Init.c**
- módulo de control del Mux, ADC y retrasos – **CalcAngle.c**
- módulo para la inicialización y envío de datos a UART – **Serial.c**
- módulo para la transmisión serie de medidas y el ángulo – **uart.c**
- módulo para el cálculo de la atan que nos dará el resultado final – **atan.c**
- módulo para sacar las señales que van al Multiplexador – **Mux.c**
- módulo para generar los retrasos – **delay.c**

3.3.1 Módulo Principal

El programa principal será el encargado de realizar una inicialización correcta del sistema, configurando de forma adecuada tanto el microcontrolador (pines de entrada y de salida, frecuencia de operación, entradas analógicas o digitales, etc.), como de llamar a las rutinas necesarias para el cálculo final del ángulo. Veamos las acciones que hace el programa (haciendo uso de las diferentes subrutinas cuando sea necesario) separándolas en los mismos pasos que realizaría durante el funcionamiento normal del circuito:

- Apagado del WatchDog
- Configuración del cristal. El oscilador interno de 8 MHz. Se elige también como reloj máster
- Fijación de offset y números de registro de memoria empleados. El offset, una vez calculado, se introduce en este campo. También es posible elegir aquí cuántos registros de memoria vamos a usar del ADC. A mayor número de registros

tendremos mayor precisión, pero también una mayor duración de la rutina para calcular el ángulo. Según cada aplicación se debe llegar a una solución de compromiso entre velocidad y precisión.

- Inicialización del ADC. Llamada a la función **ADC_Init** (ADC_Init.c). Depende del número de registros decididos

- Habilitar las interrupciones (bit global de interrupciones)

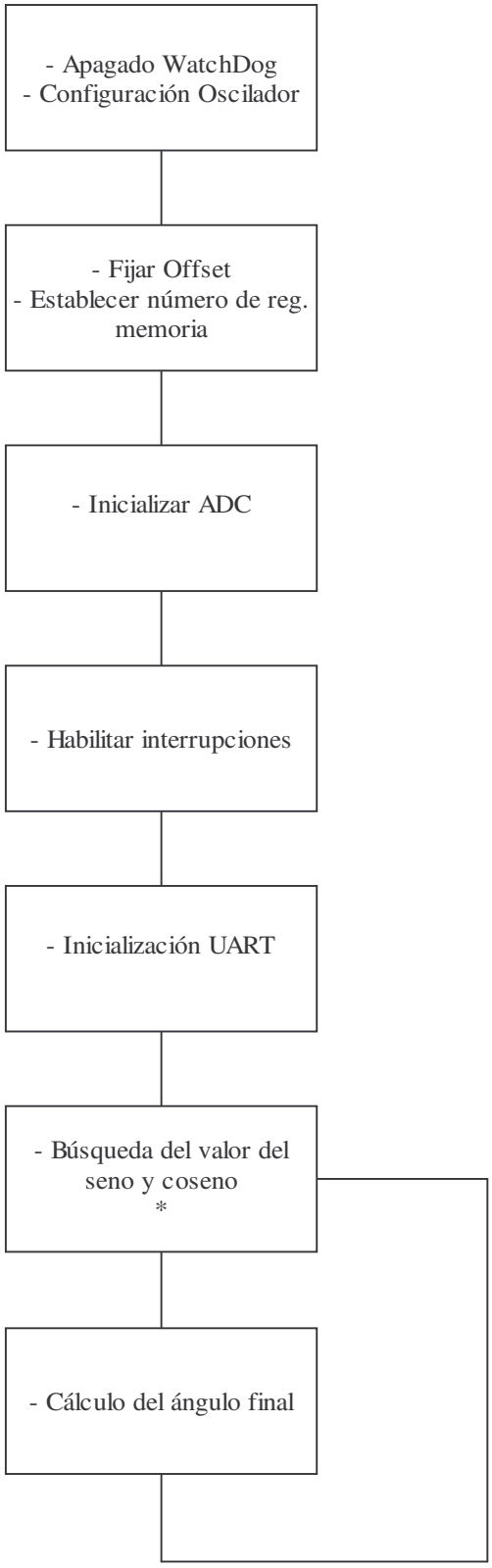
- Inicialización de la UART. Llamada a la función **UART_Init** (Serial.c). Configura la UART para la comunicación con el PC

- Comienzo del bucle infinito. Se repite continuamente lo siguiente:

    - Llamada a la función **CalcAngle** (CalcAngle.c). Esta función lanza la conversión del ADC para obtener los valores tanto del seno como del coseno.

    - Asignación del valor final del ángulo a la variable gfDisplayAngle mediante la llamada a la función **atan_green** (atan.c), se le pasan los valores obtenidos de sen y cos. Una descripción más detallada de esta función se hará más adelante.

A continuación se puede ver la estructura del módulo principal en el diagrama de flujo. El esqueleto del programa es muy sencillo, simplemente inicializa los distintos módulos del micro empleados y llama a las funciones que manejan el multiplexador y calculan el resultado final. En asterisco señalo el bloque que se desarrollará posteriormente en diagrama de flujo también.



Módulo Principal

3.3.2 Rutinas

A continuación se pasa a describir el funcionamiento de cada una de las rutinas llamadas por el módulo principal como aquellas subrutinas llamadas por otras. Para procurar que no sea muy lioso se intentará seguir una jerarquía. Se ordenan según son llamadas por el módulo principal.

ADC_Init :

- inicialización del registro ADC12CTL0 y ADC12CTL1 (tipo de conversión, duración del tiempo de sampling, tensión de referencia, etc.). Se fija la tensión interna de 2,5V como referencia interna para el convertidor.

- inicialización de tantos registros de memoria como hayan sido establecidos previamente en la función main. Con un mínimo de uno y un máximo de dieciséis. Se toma siempre el mismo canal para la conversión, el pin por donde llega la señal del multiplexador (y amplificadores).

- habilitar la conversión. Se convierte el mismo canal tantas veces como registros hayamos configurado.

UART_Init:

- fijación de la velocidad de transmisión, número de bits, activación de interrupción para la recepción, etc. 9600 Baud. 8 bits. Se activa la interrupción de la UART, de manera que cuando escribamos por el puerto serie, se produce una interrupción para tratar lo que llega.

CalcAngle:

- se encarga de manejar el multiplexador para alternar la señal que le llega al micro sen-cos-sen-cos-... Primero conmuta la señal que llega, espera un pequeño tiempo para que la señal que llega se estabilice y por « polling », sabe cuando tiene que conmutar a la otra señal. Espera hasta que un puntero cambie de valor, lo que significa que la conversión ha terminado. En la interrupción del convertidor analógico/digital se almacenan los valores convertidos del seno y del coseno.

- para manejar el multiplexador llama a la función **Mux**.

Mux:

- cambia el valor de salida de dos pines del puerto 5 para así controlar el multiplexador. Pone su valor en alto o bajo. Con 00 tenemos el seno y con 10 el coseno.

- para producir los retrasos y espera a que la señal se estabilice se llama a la función **Delay**.

Delay:

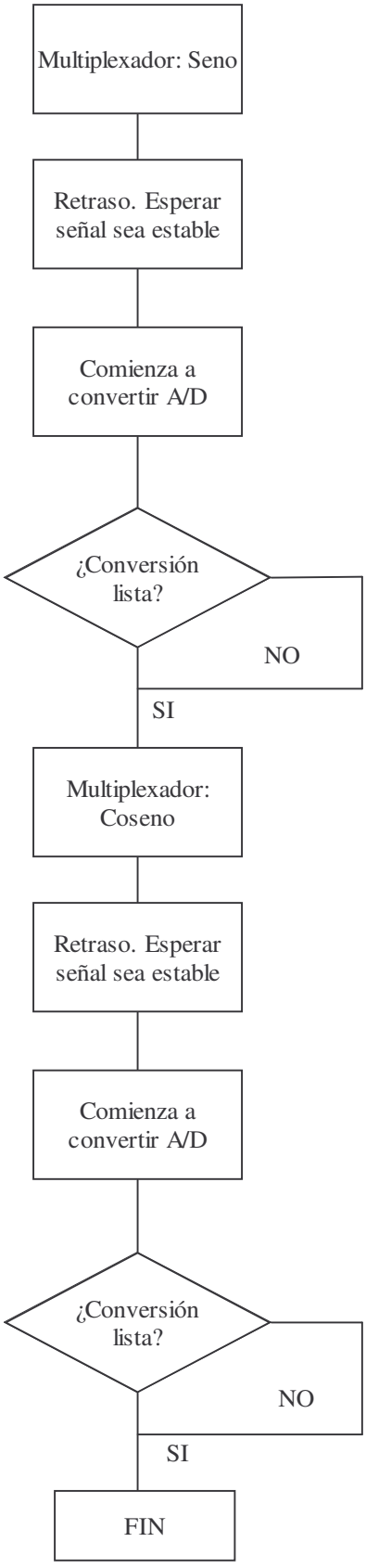
- tiene un bucle que se repite tantas veces como le digamos. Con un cálculo del tiempo de ciclo de instrucción podemos retrasarlo el tiempo deseado. En este caso son 15 µs. El tiempo se calculó para que fuese el óptimo, que la señal fuese estable pero que no provocase retrasos innecesarios.

atan_green:

- una vez que tenemos los valores convertidos del seno y coseno (manejados por CalcAngle y almacenados en la interrupción del ADC) se llama a esta función para calcular el ángulo final.

- esta función usa una aproximación de la función atan. En un principio se usó la de la librería math.h pero se trata de una función muy costosa en el tiempo. Puesto que buscábamos velocidad de cálculo, una de las partes del proyecto ha sido encontrar una función alternativa a la de la librería, que en menos tiempo diese un resultado lo suficientemente preciso. En el siguiente apartado analizaremos la función en detalle.

Se pone a continuación el diagrama de flujo de la función **CalcAngle**, de manera que se vea mejor su funcionamiento:


Multiplexador: Seno
Retraso. Esperar señal sea estable
Comienza a convertir A/D
¿Conversión lista?
NO
SI
Multiplexador: Coseno
Retraso. Esperar señal sea estable
Comienza a convertir A/D
¿Conversión lista?
NO
SI
FIN

CalcAngle

SendRawValue:

- usada por la interrupción de la UART, se encarga de sacar por el puerto serie el valor del seno y coseno sin offset.

- utiliza la función **putchar**, una variación, de la conocida función de la librería stdio.h

- va sacando byte a byte (o carácter a carácter) del valor final.

SendAngleValue:

- similar a la función anterior pero en este caso saca el valor del ángulo medido.

3.3.3 Interrupciones

Paralelamente al programa principal hay dos tipos de interrupciones programadas. La interrupción del convertidor analógico digital, que se activa al terminar de convertir la señal seno o coseno tantas veces como registros de memoria estén programados, y la interrupción de la UART, que se activa al llegar algún dato al buffer de recepción del interfaz de comunicación.

La interrupción del convertidor analógico/digital es necesaria para saber cuándo ha terminado de convertir la señal a digital. En esta misma función interrupción se tratan los datos obtenidos en los distintos registros de memoria de manera que asigna un valor definitivo para el seno y otro al coseno. Estas señales seno y coseno son las que nos da el sensor KMT32 y con las que posteriormente, a través de la función **atan_green**, sacamos el valor del ángulo final medido.

La interrupción de la UART se añadió al programa para que pudiésemos sacar por la pantalla del PC tanto los valores del seno y coseno (y así calibrar el sensor manualmente, como veremos luego) como el valor del ángulo medido. Al introducir un dato por la UART, salta la interrupción, y si es un dato válido devuelve los valores pedidos.

Interrupción ADC (ADC12ISR)

- lo primero que se hace al entrar en la interrupción (recordemos que entra al terminar de convertir el valor de la señal analógica seno o coseno a digital) es sumar el

valor de todos los registros de memoria empleados, para posteriormente calcular el valor medio obtenido.

- a continuación comprueba si la señal convertida se trata del seno o del coseno.

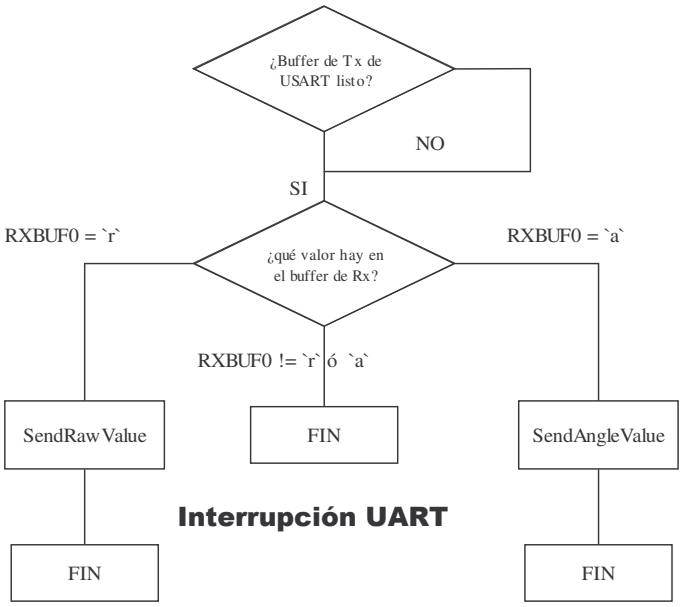
- una vez que sabemos de qué señal se trata, hacemos la media aritmética dividiendo el valor total sumado por el número de registros empleados. El valor obtenido lo almacenamos como valor “crudo”, sin offset. Esto se hace para poder calibrar cuando sea necesario. A continuación le restamos el offset (obtenido por la calibración) y almacenamos el valor definitivo del seno o coseno.

Interrupción UART (usart0_rx)

- en esta interrupción entramos al introducir un byte en el buffer de recepción del interfaz serie, UART. La información que nos es útil es, el ángulo final medido, por supuesto, y el valor del seno y coseno sin offset, para poder hallar el offset y calibrar así el sensor.

- si el byte almacenado es una ‘r’, entonces sacamos por el puerto serie el valor del seno o coseno “crudo”, esto es, sin offset. La forma de sacar el valor es usando la función **SendRawValue**, explicada con antelación.

- si el byte almacenado es una ‘a’, entonces sacamos por el puerto serie el valor del ángulo final medido. La forma de sacar los datos es usando la función **SendAngleValue**, explicada con antelación.



3.4 MEJORAS EN ALGORITMO MATEMÁTICO

En el caso del cálculo del ángulo de giro usando el sensor KMT32 es necesario usar la función arcotangente **atan.** Esta función se puede encontrar de librería en **math.h**. Sin embargo, aunque en un principio se implementó así, una vez medidos los resultados, el tiempo de respuesta del sistema era bastante elevado.

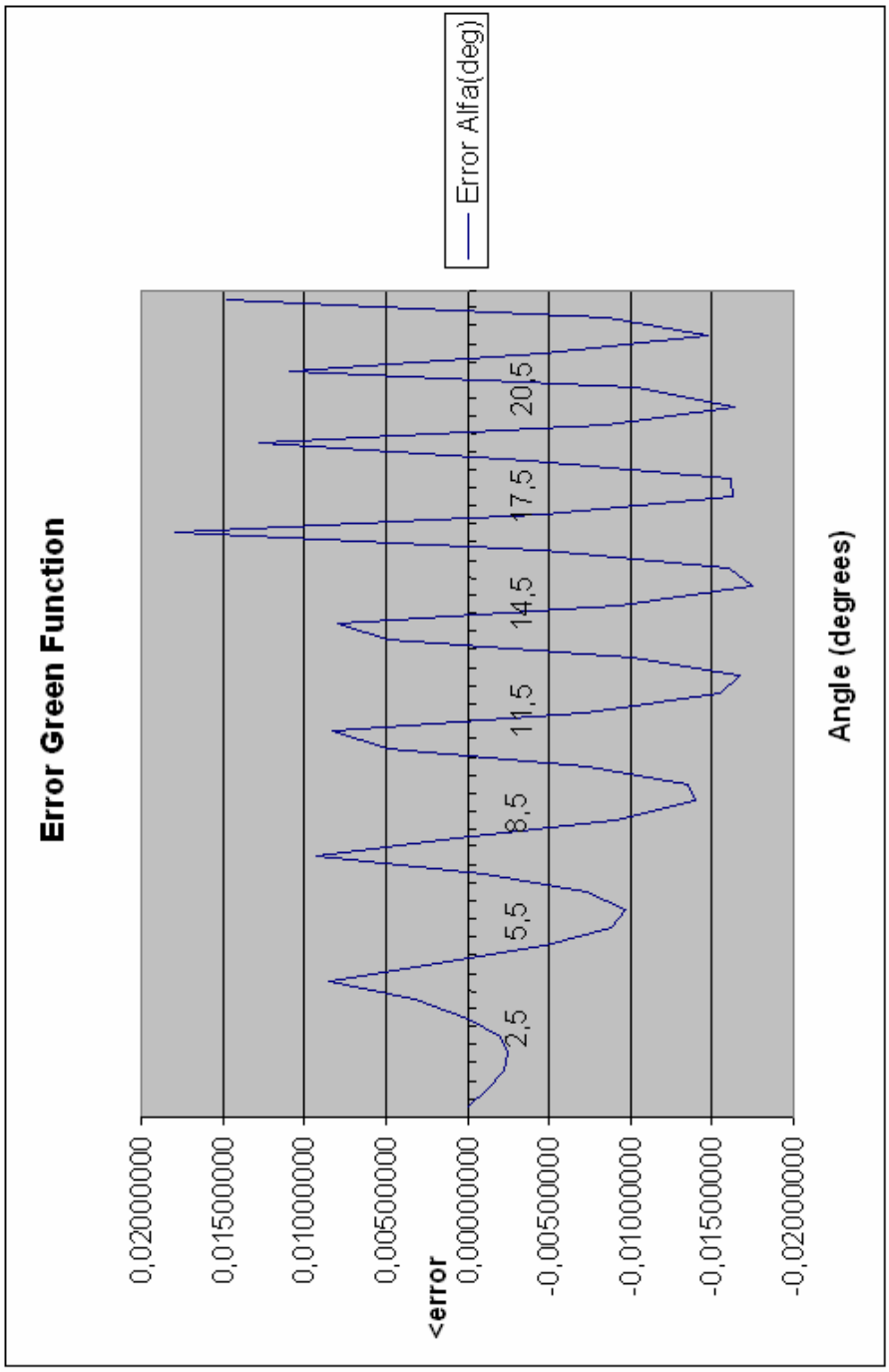
Era necesario encontrar otra función arcotangente, que sin ser muy lenta, consiguiese buenos resultados. Buscando entre varias alternativas, todas ellas con un margen de error bastante elevado, se tomó la decisión de implementar un algoritmo encontrado, escrito por Robert Green, empleado de Sony (para Play Station). Por esta razón el nombre de la función es **atan_green**. Se basa en polinomios de Taylor y Minimax en el rango [0..1]. Dividimos la función en 8 segmentos equiespaciados aplicamos minimax.

La función obtenida fue la siguiente:

$fn[1] = 0.9974133042\ x$
$fn[2] = 0.004072621 + 0.964989344\ x$
$fn[3] = 0.017899968 + 0.910336056\ x$
$fn[4] = 0.044740546 + 0.839015512\ x$
$fn[5] = 0.084473784 + 0.759613648\ x$
$fn[6] = 0.134708924 + 0.679214352\ x$
$fn[7] = 0.192103293 + 0.602631128\ x$
$fn[8] = 0.253371504 + 0.532545304\ x$

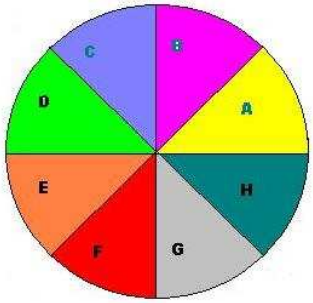
Comparándola con otras opciones de desarrollos en serie y demás fue la solución que menos error daba. En este caso, como tenemos que calcular primero el doble del ángulo y luego dividirlo entre dos, el margen de actuación de la función era de 0º a 45º en el doble ángulo y de 0º a 22,5º en el ángulo real medido. En cualquier caso mediante desplazamientos geométricos es posible calcular todo el rango de valores de 0º a 180º con esta función. En esto se basa la función **atan_green**, en primeramente calcular el valor del ángulo como si se tratase del sector de 0º a 45º y luego extrapolar el valor al resto de cuadrantes (o sectores).

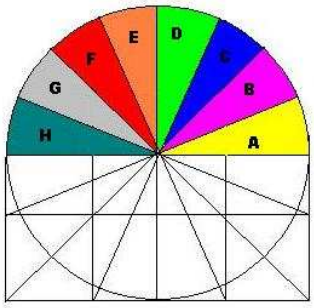
El margen de error máximo es de **0.01798260**. A continuación se muestra la gráfica que representa el error por ángulo (ángulo real medido, no el doble ángulo).



La forma de extrapolar el resultado del primer sector, de 45º, al resto de valores, es viendo los valores del seno y del coseno. Qué valor es mayor que el otro y si se tratan de valores negativos o positivos. De esta forma podemos conocer en qué rango de ángulo se encuentra y trasladar el resultado a esa zona. De eso se encarga el final de la función **atan_green**. En la tabla siguiente se pueden ver las

correspondencias de los ángulos que dan como resultado la función **atan_green** (doble ángulo) y el ángulo final.



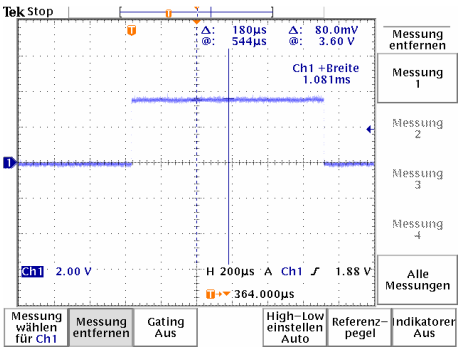


A	Sen > 0	Cos > 0	Cos>Sen
B	Sen > 0	Cos > 0	Sen>Cos
C	Sen > 0	Cos < 0	Sen>Cos
D	Sen > 0	Cos < 0	Cos>Sen
E	Sen < 0	Cos < 0	Cos>Sen
F	Sen < 0	Cos < 0	Sen>Cos
G	Sen < 0	Cos > 0	Sen>Cos
H	Sen < 0	Cos > 0	Cos>Sen

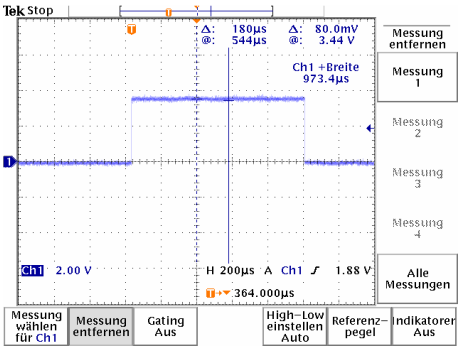
Para medir el tiempo que tardaba la rutina que se encarga de calcular la arcotangente, se uso el osciloscopio. Provocamos que un pin de salida del micro tomase el valor alto al inicio de la rutina, y al final de la rutina lo devolvíamos a su valor bajo. De esta manera se podía ver un pulso en el osciloscopio de tamaño la duración de la rutina. Además de la duración propia del cálculo de la arcotangente también es más lento el programa a medida que usemos más registros de memoria.

Para comprobar si se cumplían las especificaciones exigidas para el producto se midió la duración del programa inicial (tiempo de cálculo del ángulo) y también de la función **atan** concretamente. Se hicieron 3 medidas distintas con 3, 8 y 16 registros de memoria respectivamente. Los resultados fueron los siguientes:

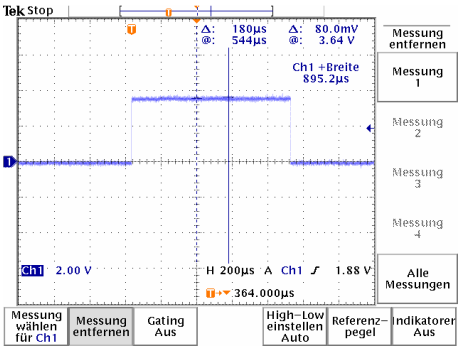
- **16** registros del **ADC**: 1081us. aprox.



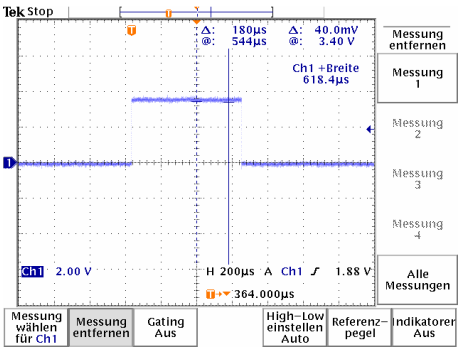
- **8** registros del **ADC**: 973us. aprox.



- **3** registros del **ADC**: 895us. aprox.

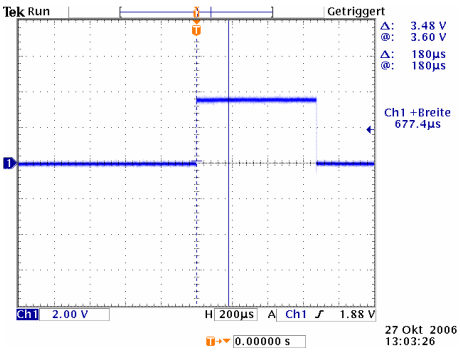


- función '**atan**': 618us. aprox.

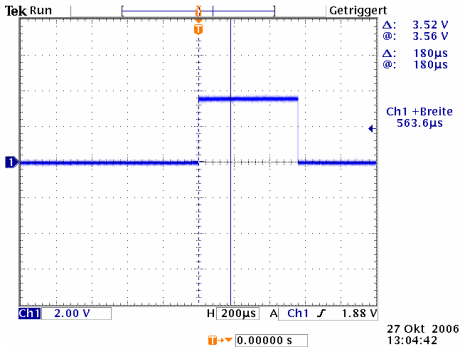


Únicamente la función **atan** tardaba 618 us, lo que la convertía en una función bastante lenta. Si utilizábamos los 16 registros de memoria posible, el tiempo de cálculo total del ángulo se demoraba hasta 1ms. tiempo bastante elevado. Esta fue la razón de la búsqueda de la nueva función para calcular el arcotangente. Los resultados de la nueva función fueron los siguientes:

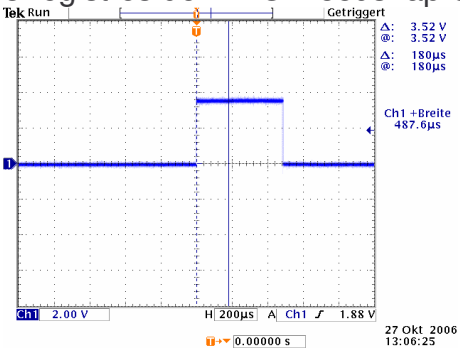
- **16** registros del **ADC**: 675us. aprox.



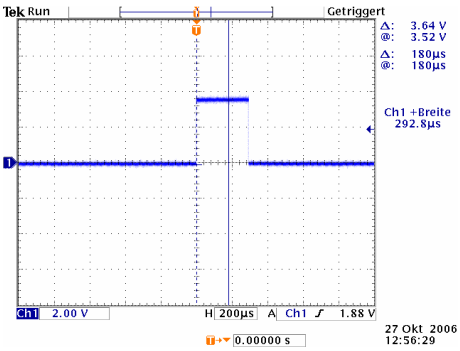
- **8** registros del **ADC**: 563us. aprox.



- **3** registros del **ADC**: 488us. aprox.



- función '**atan_green**': 290us. aprox.



Ahora la nueva función tiene una duración de menos de la mitad de la función de la librería **math.h**, 290 us. reduciéndose considerablemente el tiempo de cálculo del ángulo.