Universidad Zaragoza
1542

TRABAJO FIN DE GRADO

CONTROL DE VELOCIDAD CON MICROCONTROLADOR DE UN MOTOR DE CONTINUA

SPEED CONTROL WITH MICROCONTROLER OF A DC MOTOR

Autor/es

Alejandro Villarroya Sanz

Director/es

D. José María Martínez Montiel

D. Luis Ángel Barragán Pérez

Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza

2016/2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

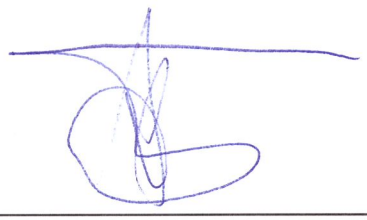
D./Dª. ALEJANDRO VILLARROYA SANZ,

con nº de DNI 73015093-Y en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado de tecnologías industriales, (Título del Trabajo) Control de velocidad con microcontrolador de un motor de continua,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 02 de Febrero de 2017



Fdo: _____

RESUMEN

En el presente trabajo de fin de grado se van a exponer los diferentes pasos que conciernen al control de velocidad de un motor de continua con microcontrolador. Para esto será necesario el montaje de un circuito electrónico que permita el control del motor en primer lugar. Una vez se ha estudiado el funcionamiento y programación del microcontrolador MSP432, dispositivo que realizará tanto el control del sistema como el cálculo de la velocidad del motor, se comienza a analizar los métodos más adecuados a la hora de medir la velocidad del motor. Exponiendo sus problemas y virtudes en lo que a este cálculo se refiere. Se tendrán en consideración tanto los ajustes de resolución que puedan derivarse de un ajuste óptimo del sensor, que en este caso será el encoder acoplado al eje del motor, como de los recursos de que dispone el MSP432 ya sean fuentes de reloj, ajustes de los timer, tiempo de muestreo, etc.

Una vez hecho esto, se procede a la identificación del modelo matemático que rige el comportamiento del motor ante una entrada de tipo escalón. Para esto se analizará visualmente este comportamiento en una GUI que previamente se habrá analizado y que permitirá afinar esta identificación.

A continuación se explicarán características del nuevo MSP432, su funcionamiento, su arquitectura, la unidad de coma flotante y la importancia que cobra para este proyecto con sus características. Más adelante se explican las fuentes de reloj disponibles, sus rangos de trabajo, la elección de las mismas y cuál ha sido el ajuste concreto para el fin que se expone. Asimismo se explicarán los temporizadores disponibles para este microcontrolador, las opciones de ajuste que proporcionan sus registros de captura y comparación, y de nuevo el ajuste de los mismos para este trabajo. Como parte que incluye la arquitectura de los modelos ARM viene incorporado el nuevo modo de procesar y ordenar las interrupciones lo que se explicará en detalle junto con las necesidades que se cubren con el correcto ajuste de las interrupciones y los registros de prioridad para las mismas.

Tras estudiar opciones para hacer uso de un programa que permita crear una GUI se explicará el mismo, las opciones que ofrece y los recursos utilizados. Todo ello, con el objetivo de ofrecer respuesta visual a la respuesta del motor y tener un mayor control sobre el comportamiento del sistema, así como de los datos obtenidos. Además se estudiará la posibilidad de hacer funcionar el programa fuera del entorno de programación.

El diseño de un controlador digital se realizará mediante la herramienta de software libre elegida. A continuación este diseño se simulará y se explicarán las características de este diseño, así como la implementación de alguna mejora para el controlador. Una vez hecha la simulación se implementará el controlador en C y se verificarán experimentalmente los resultados simulados, mostrando finalmente el funcionamiento del controlador en diversas etapas. Como último apartado se procederá a sacar las conclusiones pertinentes y las posibles líneas de trabajo futuro a modo de propuesta.

ÍNDICE

1. Introducción

1.1 Antecedentes

Bajo la definición de sistema "Conjunto de reglas o principios sobre una materia racionalmente enlazados entre sí". Controlar este sistema significa mantenerlo en comprobación e intervención, y finalmente el objetivo del mismo es alcanzar un fin que me permita darle una utilidad. En este proyecto el sistema está compuesto de varias partes que adecuadamente conectadas conforman un fin.

Este proyecto tiene como marcos en los que se apoya, los campos de la ingeniería relacionados con la electrónica digital y de potencia así como la rama referida al control de sistemas automáticos. Ambas ramas han sido vistas durante la carrera y sus conocimientos resultan esenciales para el entendimiento de la tarea a realizar. De asignaturas como sistemas automáticos y sistemas de control se adquieren conocimientos referentes al de análisis, la identificación y el diseño de controladores, esto es, el estudio de los sistemas mediante herramientas de simulación que ayudan a comprender la dinámica de este campo. Por otro lado de materias como electrónica digital y de potencia, se adquieren conocimientos referentes a las distintas etapas de potencia, modulación por ancho de pulsos de señales y programación de microcontroladores entre otros. Todos estos conocimientos conformarán la base sobre la que este trabajo evoluciona. Asimismo este proyecto constituye una oportunidad para evaluar el análisis de determinadas herramientas no utilizadas antes por el alumno y que apoyadas por un aprendizaje previo, demostrarán ser de utilidad para el mismo.

1.2 Objetivos y alcance

Los objetivos de este proyecto son varios, lo cuales se encuentran interconectados y cuya correcta ejecución y logro es consecuencia directa de la adecuada realización de los pasos a seguir. Los objetivos son:

-Estudio tanto de la arquitectura como de la programación del micro MSP432 de forma que pueda actuar con la máxima resolución a la hora de hacer tanto el cálculo de la velocidad como de un controlador para un motor de corriente continua. Se ahondará además en las posibilidades que ofrece en cuanto a sus características y nuevos desarrollos. El alcance de este estudio lo limita el estudio mismo del proyecto pues se refiere a un uso concreto.

-Identificación de la planta del motor DC mediante una aproximación del estudio de la respuesta a una entrada dada en bucle abierto. Para ello será necesario el estudio de una herramienta que permita procesar la información del micro y representarla gráficamente.

En cuanto al alcance de este estudio, se limitará a una identificación sencilla, pues aunque esta se podría hacer en profundidad, no es el objetivo de este proyecto.

-Análisis de las posibilidades del software libre de Octave como herramienta de simulación y cálculo, el cuál se ofrece como recurso suplementario al programa de cálculo de MATLAB . Esta herramienta en combinación con los recursos aprendidos en las asignaturas de ingeniería de control y sistemas automáticos me permitirá evaluar su utilidad para la implementación en sistemas reales.

- Implementación del sistema de control en el MSP432 de esta forma serán verificados los resultados salidos de las herramientas de simulación con los que se compararán.

1.3 Estructura de la memoria

A continuación se expone un breve resumen de lo que explicará en cada uno de los temas de este trabajo.

Capítulo 1: Al final de este tema se verá de forma gráfica las partes de las que se compone el sistema a utilizar para el control del motor. Además se explicará de forma breve y concisa cada una de ellas de forma que se comprenda su función dentro del mismo y como la desempeñan.

Capítulo 2: En este capítulo se valorarán las formas en la que el microcontrolador es capaz de medir la velocidad del motor a partir de la señal digital generada por el encoder, analizando así los posibles errores y formas de solventarlos. En consecuencia a esto, se elegirá aquel método que sea de mayor precisión y nos dé por tanto una mayor resolución. En última instancia y mediante una interfaz gráfica, se estudiará la forma de la respuesta en Bucle abierto a una entrada escalón, y se realizará la aproximación que más se ajuste a la forma de esta respuesta a modo de calcular la función de transferencia de la planta.

Capítulo 3: En este capítulo se explicarán los aspectos básicos del MSP432 que se usarán en el trabajo fin de grado. En primer lugar se hará hincapié en la arquitectura interna del microcontrolador. La CPU, la memoria y la unidad de coma flotante. En el apartado siguiente se explica la programación de los temporizadores, su funcionamiento y la utilización de los mismos dentro del control del sistema. En el tercer apartado se analizarán las diferentes interrupciones que pueden darse en el micro y los vectores de interrupción. Se añade además una explicación de las mismas para la consecución de la medida de la velocidad del motor. Por último el apartado final será para exponer la interfaz gráfica *Graphical User Interface* (GUI), el modo en que se implementa, la utilidad que adquiere y las posibilidades que ofrece.

Capítulo 4: Este capítulo está divido en tres partes. El software libre Octave, su funcionamiento y posibilidades a la hora de utilizarlo en un campo como es el de los sistemas automáticos. En el segundo apartado y tras haber identificado la planta del

motor, se simulará su comportamiento en Bucle cerrado y así aplicando conocimientos de control se diseñará un controlador que me permita conseguir una respuesta adecuada a unas determinadas especificaciones.

Capítulo 5: En la verificación experimental implementaremos el código C correspondiente al controlador, ya simulado en Octave, al programa del micro y comprobaremos que los resultados se asemejan lo mayor posible a la simulación previa.

Capítulo 6: Tras realizar el estudio y comprobación experimental de los objetivos planteados previamente, se alcanzará a valorar los resultados y se sacarán las conclusiones pertinentes. Además se valora posibles líneas futuras de trabajo y mejoras a modo propuesta.

1.4 Descripción del sistema

A continuación se procede a explicar de forma breve el circuito de control así como la función de aquellas partes que desempeñan una mayor tarea dentro del mismo.

El esquema de la figura 1.1 el cual está sacado de la práctica 2 de la asignatura del grado Electrónica digital y Potencia será el encargado de controlar el sistema.

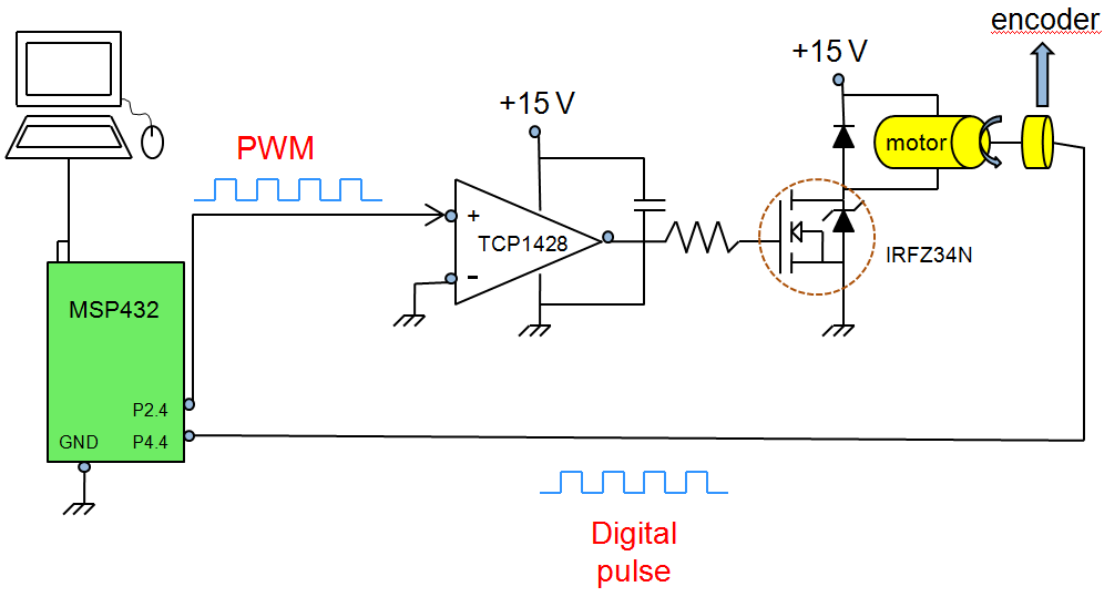


Figura 1-1. Esquema del circuito de control

Disponemos de un motor de CC, este es alimentado con una tensión de 15V y a su vez tiene acoplado a su eje un encoder. Este generará un número determinado de pulsos digitales según la velocidad de giro del motor y el propio ajuste del encoder, el cual está ajustado a 400 pulsos por revolución inicialmente.

En primer lugar el microcontrolador genera una señal PWM por *Hardware* que es captada por un driver (TC1428CPA), este eleva el voltaje de tal forma que permita conmutar un transistor MOSFET (IRFZ34N). Con la conmutación del transistor se logra

regular la tensión aplicada al motor. La tensión produce el giro del motor y a su vez que el encoder genere los pulsos digitales.

Segundo, el MSP432 captará esta señal digital del encoder por uno de los periféricos y realizando un procesado de la información por software, con un tiempo de muestreo de 1ms inicialmente, añadirá la implementación de un sistema de control será capaz de ajustar el duty de la señal PWM para que el motor gire a una velocidad determinada. La GUI proporcionará referencia visual de los cálculos realizados tanto para obtener la velocidad del motor como para posteriormente implementar un sistema de control.

En cuanto a las partes que lo componen:

1. **Motor CC**: Se trata de un motor de corriente continua del fabricante MELLOR ELECTRICS, con una tensión de alimentación máxima de 24 V pudiendo alcanzar las 3500 Rpm sin carga y un consumo de corriente de 0.22 A.

2. **Encoder**: Este se encuentra acoplado al eje del motor y es regulable en la medida en que se pueden ajustar los pulsos según la resolución que se desea obtener en función de las vueltas del motor. Esta resolución va desde los 48 a los 2048 pulsos digitales, de onda cuadrada, por vuelta del motor. Además de forma adyacente con el nº de pulsos generados va asociado un nº de Rpm máximo que puede procesar. El encoder será el responsable por tanto de marcar en gran parte la resolución de medida de las vueltas por minuto del motor.

3. **Microcontrolador MSP432P401r**: Este se ha programado en C mediante el programa *Code Composer Studio 6.1.2* Es la parte que hará tanto el cálculo de la velocidad del motor como del controlador digital. Además cuenta con un núcleo ARM que permite cálculos de gran velocidad en coma flotante.

4. **TC1428CPA**: Este driver está compuesto por un par de amplificadores operacionales, uno de ellos inversor. Se trata de un driver de alta frecuencia de conmutación (hasta pulsos de 25 ns) que es idóneo para recoger señales de tipo PWM de bajo voltaje y elevar la tensión y la corriente que entra por la puerta del MOSFET y lo hace conmutar.

5. **IRFZ34N**: Se trata de un transistor MOSFET de alta frecuencia de conmutación. La tensión umbral Vth se encuentra entre 2 - 4 V con unos límites de tensión en la puerta de +-20 V. Con una tensión en corte máxima de V_{DDS} =55V y una corriente máxima en conducción de I_D =29 A. Dado que el microcontrolador es capaz de aportar una tensión máxima por sus puertos de 3 V se hace uso de un driver y de esta forma, elevar esta tensión por encima de la tensión umbral y que el transistor entre en saturación. Este transistor tiene unos tiempos td(on)=31 ns, tr=40 ns y td(off)=31 ns, tf=40 ns.

2. Análisis y modelado del sistema

2.1 Medición de la velocidad

En este apartado expongo los dos métodos más eficaces que hay a la hora de medir la velocidad del motor. Se expondrá uno frente a otro y se concluirá cual de ambos es el más oportuno utilizar y por qué. Aquí el papel del encoder y su ajuste será esencial para el análisis [1]. De forma que se tenga una referencia para comenzar a evaluar los resultados tanto teórica como experimentalmente, el ajuste del encoder será de 400 pulsos/revolución y el tiempo de muestreo de 1 ms con una frecuencia de reloj de 24 Mhz. Los ajustes usados en cuanto a temporizadores, reloj, etc se explicarán en capítulos posteriores.

2.1.1 **Medición de la velocidad por pulsos**

De las dos maneras en las que realizo la medición esta es la que a la hora de cuantificar la fiabilidad de la medición, mayor dependencia del ajuste del encoder tiene. Los pasos seguidos a la hora de medir y probar son:

1- Mediante un generador de señales se ajusta una frecuencia determinada y con una forma de onda cuadrada. Debido a que el microcontrolador no es capaz de soportar más de 3.3V de tensión de entrada, se ajusta a este valor.

2- Para asegurar que se suministra suficiente tensión de entrada al micro, se alimenta un inversor a la tensión de 3.3 V aproximadamente, y esta tensión será la que finalmente conecte al micro por un puerto que se habrá asignado previamente como puerto de entrada y que será el que capte la señal digital del encoder acoplado al eje del motor y por lo tanto sus conmutaciones.

3- La idea es que, tras cada flanco de subida o bajada generado por el encoder, el cual está asociado al nº de revoluciones por minuto del motor, se produzca una interrupción del micro y en ese instante aumente en una unidad una variable declarada globalmente. Una vez el contador del Timer0 alcance su máximo valor y resetee (suceso que se producirá cada 1ms) se vuelca el valor del contador en una variable que mediante diversas operaciones dará el valor de la velocidad del motor como en la ecuación (1).

$$w(rpm) = \frac{contador\left(nº\frac{flancos}{ms}\right)\cdot 1000\left(\frac{ms}{sg}\right)\cdot 60\left(\frac{sg}{min}\right)}{400\left(\frac{pulsos}{revolución}\right)} \qquad (1)$$

4- Una vez tenemos esta variable tras cada periodo de muestreo se guardará su valor en un vector y lo que se hará será procesarla mediante la GUI y representar este vector, con lo que se obtendrá la respuesta en tiempo a una entrada de tipo escalón.

2.1.2 Medición del periodo

De nuevo teniendo la señal digital generada por el encoder y asociada a la velocidad del motor en revoluciones por minuto. Introduciré esta señal por uno de los puertos I/O en este caso el pin 5.6 del micro. Previamente se habrá programado una función que medirá el intervalo de tiempo que hay entre cada uno de los flancos de subida de la siguiente forma:

1- Se programa el Timer2 del micro de tal forma que cuando se detecte un flanco de subida se produzca una interrupción. Además se activarán los registros que habilitan la captura de dicho instante en el registro TA2CCR1 del microcontrolador.

2- Tras cada interrupción ocurrida en el Timer2, el valor de la captura que es guardado en TA2CCR1 se volcará en la variable *sig*. Si la resta entre las variables *sig* y *prev* resulta en un valor negativo, es decir, se da una captura en distinto tiempo de muestreo, lo que se hará será sumar a ese valor los 23999 valores que puede tomar el TA2CCR0 y asignarlo a la variable *fin* que guarda el valor de la velocidad. De esta forma la variable *fin* tomará siempre el valor absoluto de la resta entre flancos de subida figura 2.1, con lo que se podrá traducir directamente mediante la ecuación (2) a revoluciones por minuto:

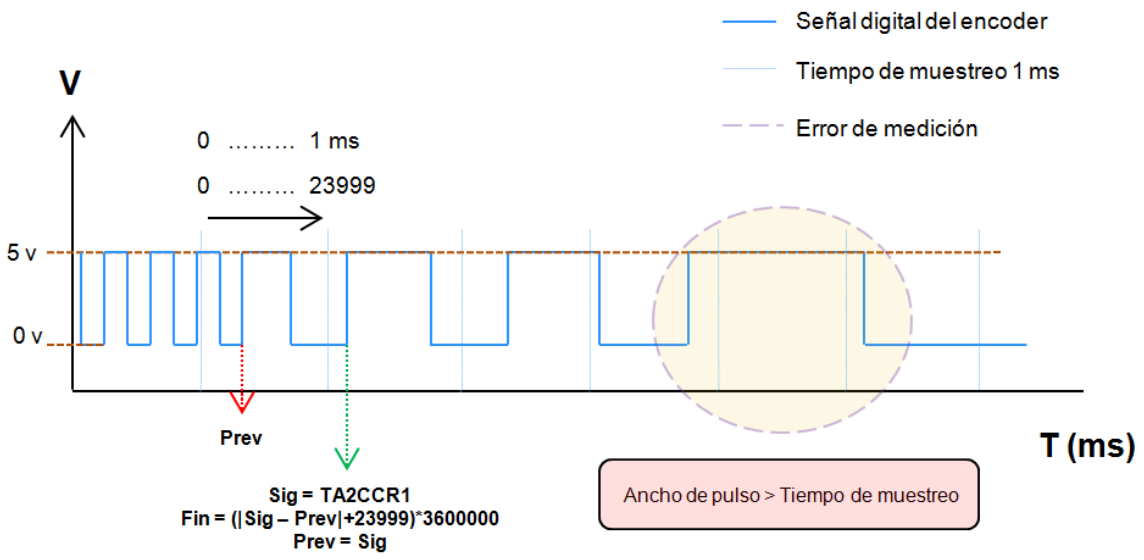


Figura 2.1 Gráfica explicativa del error por diferencia de flancos

$$w(rpm) = \frac{\frac{24000(bits/ms)}{fin(bits)} \cdot 1000\left(\frac{ms}{min}\right) \cdot 60(\frac{sg}{min})}{400(\frac{pulsos}{rev})} \quad (2)$$

3- De nuevo representando en el tiempo esta transformación final de la variable *fin* obtendremos una respuesta a la entrada escalón que se podrá procesar una vez guardada.

```
sig = TA2CCR1;//se almacena el valor de la captura en la variable sig

if ((float)(sig-prev)<=0) {//si la diferencia entre las variables
capturadas en flancos de subida consecutivos es =< 0

      fin = (float)(24000*1000*60/((23999 + sig -prev)*400));//se
      calcula la velocidad a rpm y se asigna a la variable fin
}
else {//si la captura de los flancos está dentro del mismo tiempo de
muestreo

      fin = (float)(24000*1000*60/((sig-prev)*400));//cálculo de la
      velocidad
      }
prev = sig;//asigno variable sig a variable prev
```

Figura 2.2 Código C para el cálculo de velocidad por periodo

Se muestra en figura 2.2 la codificación para el cálculo por medición del periodo.

2.1.3 Problemas de cuantización

A la hora de priorizar y elegir un método final que resulte fiable, habrá que tener en cuenta ciertos parámetros que son modificables y que resultan en mejora de los resultados de medición según el método a utilizar. Asimismo se exponen también aquellos parámetros que limitan la calidad de la estimación.

De los problemas que se dan, comunes a ambos métodos se tiene:

1- Fuente de alimentación con ruido. El ruido en la fuente de alimentación provoca inestabilidad en la tensión que cae en bornes del motor y por lo tanto afecta directamente a la anchura de los pulsos generados por el encoder. En mejora de este fenómeno se utilizarán condensadores que filtren el ruido.

2- La resolución de los pulsos no es estable. Esto incide en la subida y bajada de los flancos de pulso y por tanto en la anchura del pulso. Afecta a ambos métodos pero no de igual forma, pues el conteo de pulsos no está sujeto directamente a este problema que más adelante se explica.

Problemas del conteo por pulsos.

En primera instancia hay que contar que este método se basa en contar el número de pulsos que existen dentro de un periodo de muestreo, y por tanto durante el transitorio en el que la anchura de los pulsos está variando de forma rápida, este tipo de estimación no tiene demasiada validez. Dicho esto, la precisión de este método tiene como limitación el número de pulsos generados por vuelta en el encoder, y el tiempo de muestreo. Esto es así porque, a 400 pulsos por vuelta de configuración inicial, y con una velocidad máxima del motor, procesable por el enconder, de 3500 rpm me encuentro con nº máximo de pulsos por periodo de muestreo igual a (3500 rpm ·400 pulsos/rev) / (60 seg/min ·1000 miliseg/seg) = 23,333 pulsos/ miliseg, con lo que el error máximo de cuantización es igual a 3500 rpm /23,333 pulsos = 150 rpm/ pulso.

Otro de los errores que se dan, es que al bajar la velocidad del motor de tal forma que los pulsos generados sean de mayor anchura que el tiempo de muestreo, lo que corresponde a 150 rpm, se detecta como mucho un flanco por tiempo de muestreo, y por consiguiente será imposible estimar la velocidad en el rango entre 0 y 150 rpm.

Como solución y para mejorar la calidad de estimación:

1. El encoder, que es ajustable, permite regular los pulsos generados por vuelta hasta los 2048 y que el motor gire con un máximo de 7500 rpm. Lo que significa que en un mismo periodo de tiempo se tendrá un mayor número de pulsos. Así que el error máximo de cuantización es igual a (60s/min ·1000 ms/s)/ 2048 pulsos/rev = 29,296 rpm/ pulso.

2. Además, si se aumenta el tiempo de muestreo, en este caso hasta los 2 ms, lo que se logra con esto es que, el número máximo de pulsos que se pueden dar en un periodo de muestreo, sea del doble y por consiguiente el error máximo por cuantización cae a la mitad = 14,648 rpm/ pulso. Aumentar más el tiempo de muestreo podría suponer un problema a la hora de calcular un controlador y de disminuir el tiempo de respuesta, ya que estos van ligados al tiempo de muestreo. Por lo que esta he considerado que es la forma óptima de estimación de la velocidad por conteo de pulsos.

Problemas del conteo por intervalo de tiempo.

Este tipo de estimación va sujeta fuertemente a 2 parámetros, la frecuencia de reloj y su precisión, y la estabilidad de los pulsos digitales que se obtienen del encoder. Como soluciones se tiene:

1. La frecuencia de reloj da la resolución de la captura, pues con una frecuencia de reloj de 24 Mhz y un periodo de muestreo de 1 ms el registro TA2CCR1 puede tomar hasta 24000 valores con un error máximo de cuantización de 1/24000 valores/ ms ·10^6 ms/ns = 41,666 ns/ valor. Lo que se traduce en un error máximo menor que una revolución por minuto, es decir la estimación teórica por

diferencia de flancos con una frecuencia de reloj de 24 Mhz, sin sumar otros errores, es perfecta. Por ello no sería necesario aumentar la frecuencia de reloj, por ejemplo hasta los 48 Mhz. Además para el caso en que la anchura de los pulsos fuera mayor de un tiempo de muestreo (1ms) no se podría calcular la velocidad ya que el timer es de 16 bits (65536 valores) a no ser que se hiciera una división sobre el timer lo que de nuevo no permitiría una mejora de la estimación.

2. Por otro lado al no obtener del encoder unos pulsos de anchura estable, pudiendo ser esto consecuencia de una mala alimentación del motor o bien la longitud de los cables genera ruido, etc, y dado que este método se basa en la diferencia del valor entre flancos de subida, cualquier error se traduce en error de cuantización, y es independiente del tiempo de muestreo ya que únicamente se representa la diferencia entre el último par de flancos de subida antes de que desborde TA2CCR0. Lo que por supuesto es positivo pues se representa la velocidad en el momento exacto de desborde del timer, al contrario que en el conteo por pulsos.

3. Por último podría suponer un problema que las operaciones que tiene que realizar el micro para calcular las Rpm y posteriormente el controlador le tomara un tiempo mayor que el tiempo entre flancos de subida. Esto se podría dar si la tensión de alimentación fuera la máxima que permite el motor, 24V y girara a 3500 Rpm, ya que se generarían 119,4 pulsos/ ms que equivalen a un intervalo de 8,3 μs. El cálculo del tiempo que le lleva al micro realizar la división para el cálculo de las Rpm se hace conmutando un puerto de salida justo antes de operar y justo después. El tiempo es de 3,2 μs aproximadamente así que en principio no existiría problema. Sin embargo dado que se obtienen unos pulsos algo inestables si que podría suponer un problema. En principio este error no se dará ya que, debido a las limitaciones físicas del circuito, se alimentará a 15 V.

Para hacer la comparación final, lo que se hace es lo siguiente. Tras cada periodo de muestreo se guarda el dato de la velocidad actual en una variable de tipo vector y mediante la GUI se representa gráficamente. Además se ajustarán los parámetros del sistema de forma que se tenga máxima resolución en ambos métodos de estimación. Como la resolución del encoder es indiferente para uno y prioritario para el otro de los métodos, finalmente se ajustará este para que genere 2048 pulsos/ rev. En cuanto al muestreo para el conteo por pulsos se aumenta este hasta los 2 ms, mientras que para el otro se mantiene en 1 ms.

El primero de los métodos que se ve en la figura 2.3 es la estimación de velocidad por diferencia entre flancos de subida.

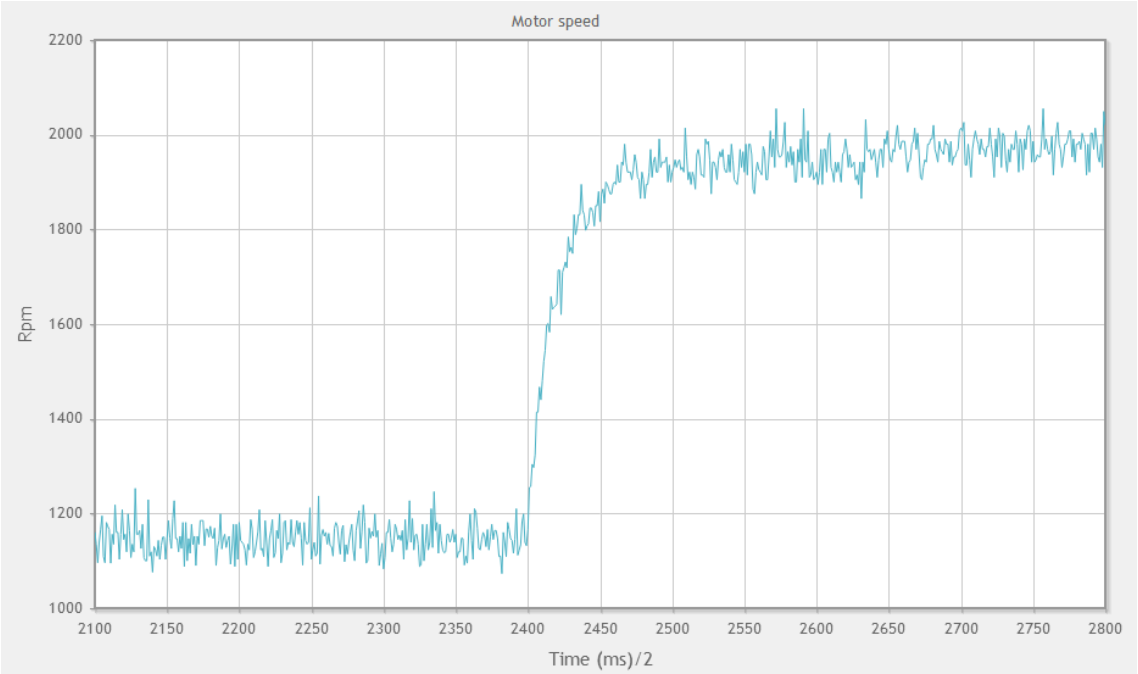


Figura 2.3 Respuesta a escalón (conteo por diferencia entre flancos de subida)

Como se puede observar en la representación de los valores que se van obteniendo a una entrada escalón, la zona estable tiene un gran rango de dispersión en cuanto a los valores finales, con un rango de dispersión aproximado de 100 Rpm.

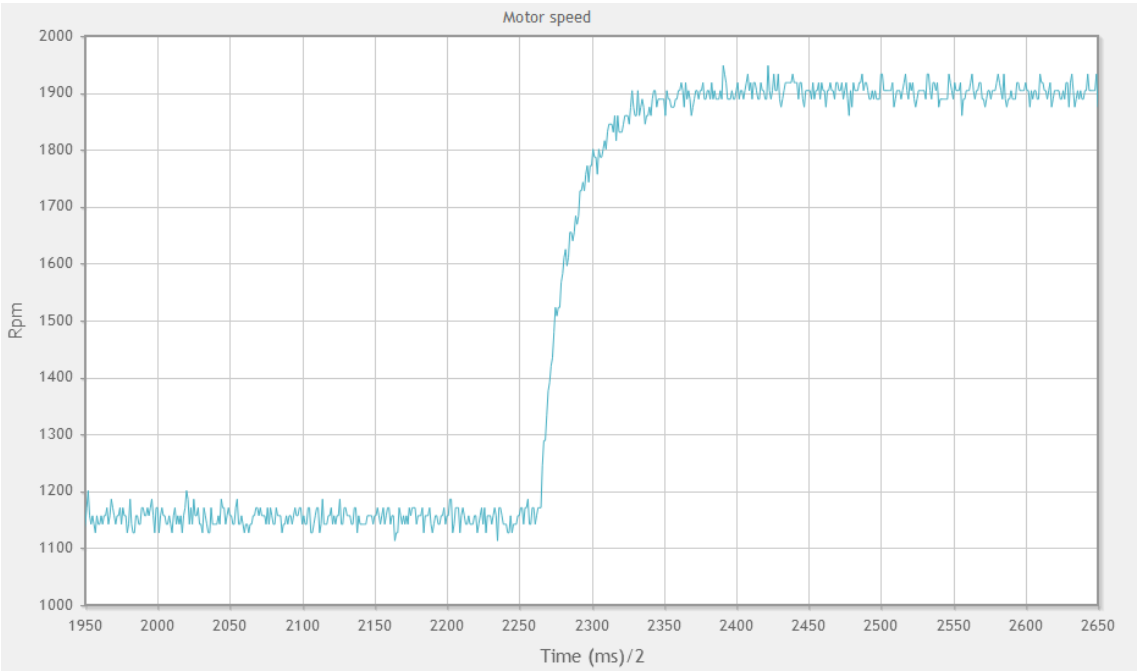


Figura 2.4 Respuesta a escalón (conteo por pulsos)

En cambio si lo comparamos con la figura 2.4 con la misma entrada escalón para el método de conteo de pulsos. Se puede apreciar que el rango de dispersión aproximado es de 50 Rpm para el conteo por periodo, por tanto la calidad de la medida del método por conteo de pulsos da un mejor resultado de estimación. Esto servirá para hacer una mejor

caracterización de la función de transferencia de la planta del motor lo que finalmente se traducirá en un mejor control, más afín a las características reales de la planta. Por último añadir que el periodo elegido para la señal PWM es de 2000 ciclos de reloj o tiene una frecuencia de 12 Khz, lo que permite conseguir 2000 valores de velocidad angular. Sin embargo en la práctica esto no es así ya que este rango de valores se ve limitado por el número máximo de pulsos generados por el encoder que es 256 pulsos/miliseg. De esta forma no se introduce error de cuantización.

2.2 Identificación de la planta

Este apartado y su correcta resolución será clave para que el control del motor sea lo más afinado posible. La identificación de la planta significa, dada una entrada, aproximar la función característica de respuesta de un determinado sistema a un modelo teórico que reproduzca su comportamiento. El objetivo de esto es conocer dada una entrada, en este caso ciclos de duty, que son los ciclos de reloj que se mantiene en alto la señal PWM, saber cuál es la salida que será dada en Rpm. En definitiva conocer la relación entrada-salida del sistema G(s) figura 2.5.

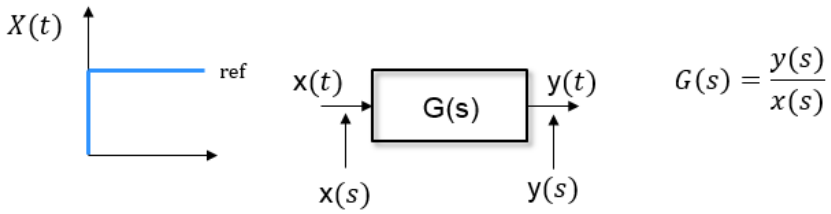


Figura 2.5 Función de transferencia del sistema

Los dos métodos para resolver esto son, o bien se conocen las ecuaciones físicas que rigen el sistema o se realiza un análisis experimental con los datos de la respuesta del sistema a un estímulo externo. Lo correcto sería complementar ambos métodos. Por simplicidad, se resolverá por aproximación experimental [2]. A modo de evitar los posibles errores de estimación que se puedan dar a velocidades bajas, se parte de una tensión de referencia. El procedimiento de análisis es el siguiente:

1. Teniendo una referencia de 1500 ciclos de duty , y representando en el tiempo, se introduce otra referencia hasta los 2000 ciclos de duty, es decir, una entrada escalón. Analizando visualmente la respuesta en la GUI, se puede concluir que esta tiene la forma de un sistema de primer orden sin retardo, cuya función de transferencia es la ecuación (3).

$$G(s) = \frac{K}{\tau s + 1} \tag{3}$$

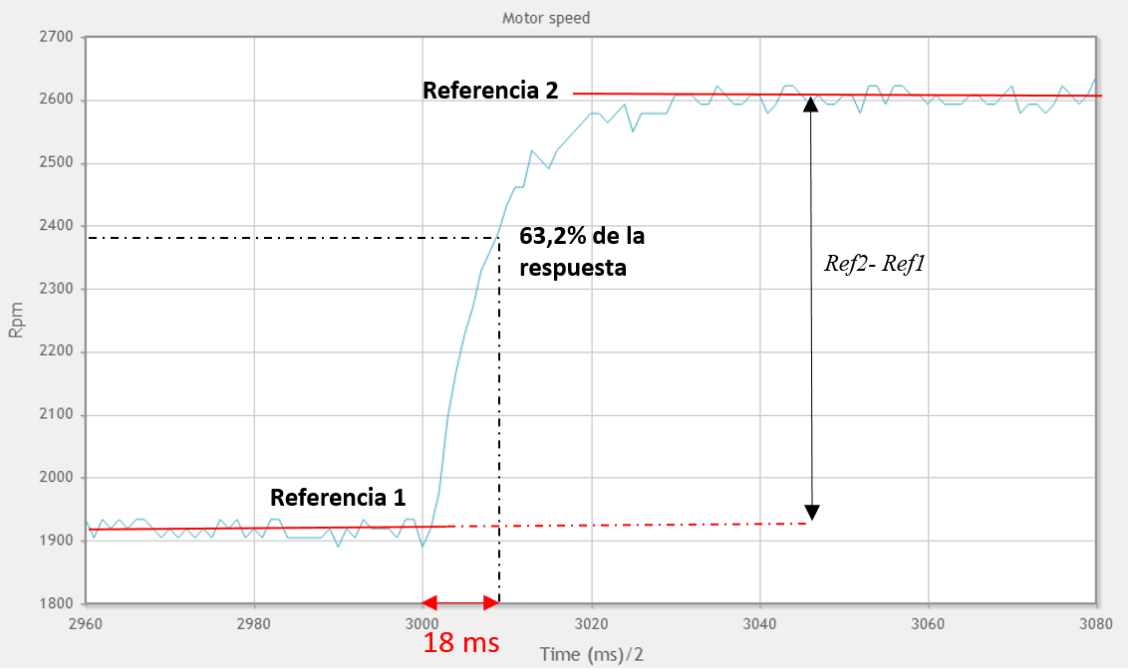


Figura 2.6 Identificación de la planta del sistema

2. Tras la toma de datos del micro, se exportan los mismos a un fichero y se cogen los correspondientes a ambas regiones de establecimiento. Se hace la media de cada una de las regiones y de esta forma se aproximan a un único valor. Así entonces se podrá obtener la diferencia entre ambas referencias como se ve en la figura 2.6. Una vez calculada la amplitud del escalón proporcionado al sistema, se halla el polo de la planta donde la señal de respuesta alcanza el 63,2% de la amplitud de la entrada escalón. Como previamente se ha determinado el instante de tiempo donde la referencia cambia, se halla el tiempo que le ha tomado llegar al mismo desde la primera referencia, y de esta forma se obtiene τ. De esta forma se obtiene el polo de la planta. A modo de obtener una aproximación mayor se han repetido estos pasos 10 veces tras lo cual se hace la media de los resultados y se aproxima a un valor que en este caso es 0.018 seg.

3. Para hallar la ganancia, se aumenta el ciclo de duty al máximo y de la gráfica se saca el valor que alcanza. De esta forma se divide la salida por la entrada y se obtiene la ganancia K=1,275 la cuál tiene en este caso unidades en Rpm/ciclos de duty. De esta forma se puede concluir que el sistema se comporta aproximadamente según la ecuación (4).

$$G(s) = \frac{1.275}{0.018s + 1} \tag{4}$$

3.Microcontrolador MSP432

En primer lugar decir que la placa de desarrollo MSP-EX432P401R de la figura 3.1 que se va a utilizar en este trabajo se encuentra dividida en dos partes, la que refiere al microcontrolador MSP432P401R y la de la sonda XDS-110ET para programación y depuración de código, que hace uso de la nueva tecnología *EnergyTrace+* para mediciones de energía visibles mediante la GUI del *EnergyTrace*. Existen dos modelos de esta familia disponibles, el MSP432P401R y el MSP432P401M, cuya única diferencia radica exclusivamente en que el primero tiene 256 KB de Memoria Flash y 64 KB de RAM frente a los 128 KB de Flash y 32 KB de RAM del segundo. Tanto sus procesadores, rangos de operación, sistemas de reloj, etc, son exactamente iguales [3].

Figura 3.1 Placa de desarrollo MSP-EXP432P401R

3. 1 Arquitectura

En este apartado se explica la arquitectura del MSP432 que es en definitiva, la estructura de su funcionamiento. De las dos arquitecturas más usadas en micros, como son la *Von Neumman* y la *Harvard* el MSP432 hace uso esta segunda, que se diferencia de la primera por tener una memoria de programa y una memoria de datos diferenciadas y que tienen como objetivo trabajar al mismo tiempo. La ventaja de esto es que como los buses son independientes, la CPU puede estar ejecutando una instrucción mientras lee la siguiente. Con esto se consigue una mayor velocidad de ejecución en los programas. Asimismo, el hecho de que ambas memorias estén separadas permite que las instrucciones ocupen una única posición en la memoria del programa de forma que obtengamos una menor longitud de programa [4].

Siguiendo la línea del MSP430, su procesador es de tipo **RISC** (Reduced Instruction Set Computer) lo que permite una programación más sencilla, además la multitarea del procesador aparte del aumento de velocidad en el proceso también consigue que cada

instrucción se procese con la misma velocidad. La CPU del nuevo MSP432 está formada alrededor del concepto de ARM Cortex-M4F de núcleo 32-bit, y se beneficia de las herramientas de desarrollo y de software asociadas a los procesadores ARM. Este está basado en la arquitectura del ARMv7-M con lo que la definición y longitud de los tipos de datos, como se muestra en las tablas 3.1 y 3.2 [5] cambia con respecto al MSP430.

C type	stdint.h type	Bits	Sign	Range
char	uint8_t	8	Unsigned	0 .. 255
signed char	int8_t	8	Signed	-128 .. 127
unsigned short	uint16_t	16	Unsigned	0 .. 65,535
short	int16_t	16	Signed	-32,768 .. 32,767
unsigned int	uint32_t	32	Unsigned	0 .. 4,294,967,295
int	int32_t	32	Signed	-2,147,483,648 .. 2,147,483,647
unsigned long long	uint64_t	64	Unsigned	0 .. 18,446,744,073,709,551,615
long long	int64_t	64	Signed	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

Tabla 3.1 Tipos de datos para la arquitectura ARMv7-M

C type	IEE754 Name	Bits	Range
float	Single Precision	32	-3.4E38 .. 3.4E38
double	Double Precision	64	-1.7E308 .. 1.7E308

Tabla 3.2 Datos de coma flotante, arquitectura ARMv7-M

Debido a que el tamaño natural de los datos para el procesador ARM es de 32-bits es preferible usar int como variable a short. El procesador podría usar más instrucciones en el cálculo de un short que de un int. En caso de que se quisiera portar código de otras plataformas sobretodo de plataformas de 8-bit o 16-bit el tamaño de los tipos de datos sería diferente. Es por ello que sería preferible definir los datos a tipo stdint.h lo que será portable a lo largo de todas las plataformas.

3.1.1 ARM Cortex-M4F

El procesador ARM Cortex-M4F proporciona una plataforma de alto rendimiento que atiende a los requerimientos del sistema como son, implementación mínima de la memoria, reducción del número de pins, así como, un bajo consumo de energía. La figura 3.1 muestra el diagrama de bloques de la CPU [6].

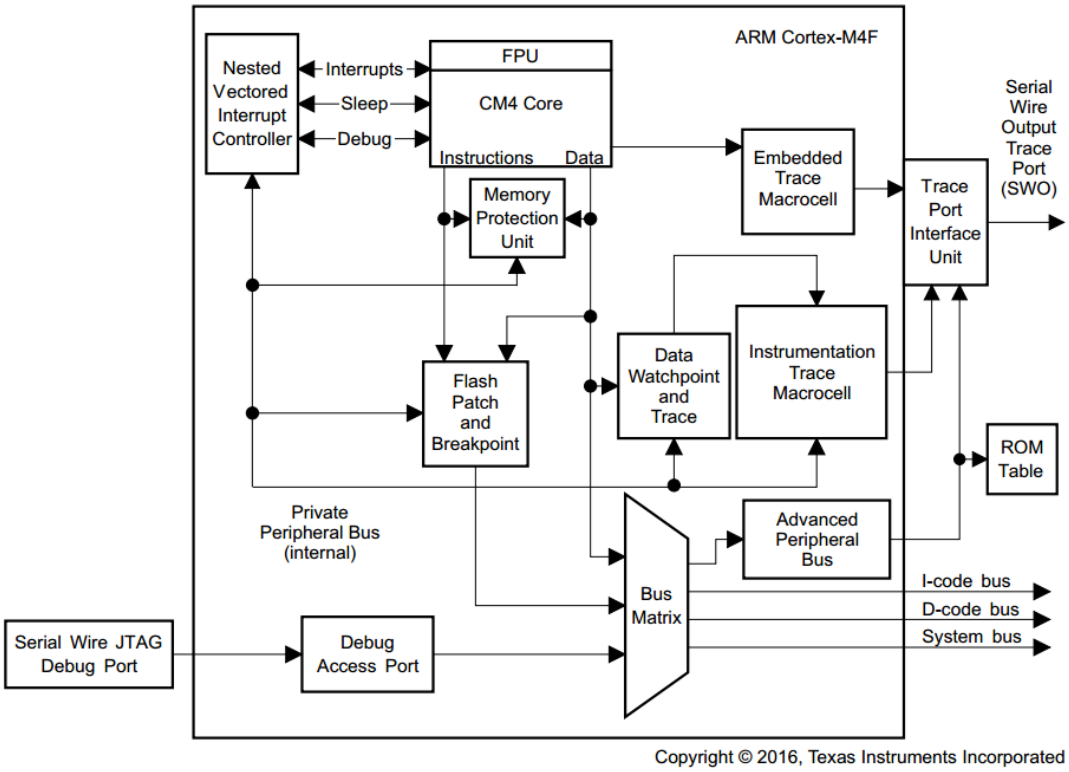


Figura 3-1 diagrama de bloques CPU

El Cortex-M4F tiene dos modos de operación:

- Thread: Usado para ejecutar el software de aplicación. El procesador entra en el modo *Thread* cuando sale de un reset.
- Handler: Usado para controlar las excepciones. Cuando el procesador termina de procesar una excepción vuelve al modo *Thread.*

El procesador utiliza una pila descendente, esto implica que el *Stack pointer* señala el último elemento apilado en memoria. Si el procesador introduce un nuevo elemento en la pila, el *Stack pointer* decrementa y escribe el elemento en una nueva localización de la memoria. El procesador implementa dos pilas: *Main stack* y *Process stack*, con un marcador para ambas contenido en registros independientes. En el modo *Thread* el registro CONTROL usa tanto el *Main stack* como el *Process stack.* Mientras que en el modo *Handler* el procesador utiliza siempre la pila *Main.* La tabla 3-2 muestra el conjunto de registros del Cortex-M4F.

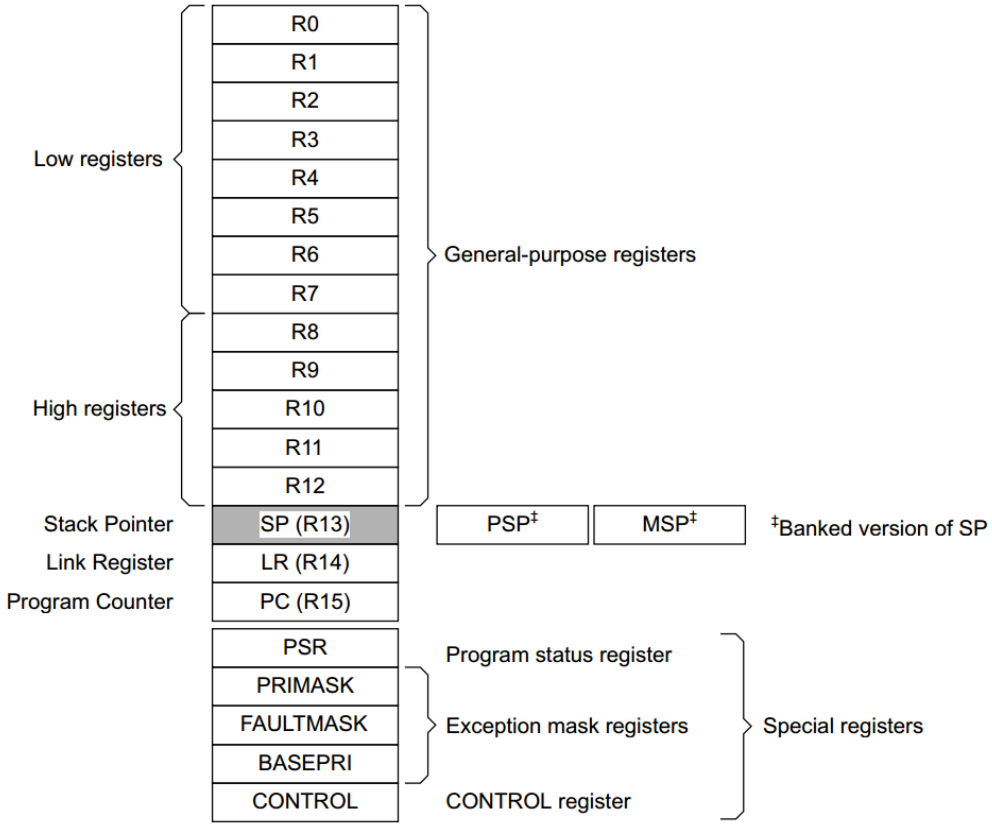


Tabla 3-2 Set de registros del Cortex-M4F

Del R0 al R12 se trata de registros de 32 bits de uso genérico para operaciones con datos. No poseen un uso especial definido por su arquitectura. Los registros del R13, R14 y R15 tienen funciones especiales.

- R13: Este se trata del registro *Stack pointer (SP)*. En el modo *Thread* la función del registro cambia dependiendo del bit ASP en el registro CONTROL. Si el bit ASP está a 0 este registro es el *Main stack pointer (MSP)* por el contrario si está a 1 el registro será el *Process Stack pointer (PSP)*. Si se produce un Reset el bit ASP se pone a 0 y el procesador carga el MSP con el valor de la dirección 0x0000.0000.
- R14: Se trata del *Link register (LR)* y almacena información de retorno para las subrutinas, llamadas de función y excepciones.
- R15: Se trata del registro *Program counter (PC)* y contiene la dirección de la siguiente instrucción a ejecutar.
- PSR: *Program status register* dividido en tres partes y muestra el estado del programa de la aplicación, el estado de la programa de ejecución y el número de excepción de la actual *Interrupt service routine* (ISR).

El sistema de Cortex-M4F está compuesto por los siguientes componentes:

- SysTick: Timer de 24 bits que cuenta hacia abajo y que puede ser usado como un Timer RTOS(*Real-Time Operating System*) o como un simple contador

- NVIC (*Nested Vectored Interrupt Controller*): Controlador embebido de interrupciones que proporciona procesado de las interrupciones de baja latencia. (más tarde explicado en el apartado 3.4)
- SCB (*System Control Block*): La interfaz de programación del procesador. El SCB proporciona información del sistema de implementación y del sistema de control, incluye configuración, control e información sobre las excepciones del sistema.
- MPU (*Memory Protection Unit*): Mejora la fiabilidad del sistema definiendo los atributos de la memoria en diferentes regiones de la misma. La MPU proporciona hasta 8 regiones diferentes y una región opcional predefinida en segundo plano.
- FPU (*Floating-Point Unit*): Unidad de coma flotante.

3.1.2 FPU

Como parte esencial del MSP432 este incorpora la unidad de coma flotante o FPU que será de gran importancia para el cálculo de operaciones en coma flotante a gran velocidad y precisión como son, el cálculo de las revoluciones del motor, así como, para la implementación de las ecuaciones que rigen el comportamiento del controlador en *Bucle Cerrado*. La FPU aporta [7]:

- Instrucciones de 32 bits para operaciones de procesado de datos de tipo float.
- Instrucciones combinadas de multiplicación y acumulación para una mayor precisión
- Apoyo Hardware para conversión, sumas, multiplicación con acumulación, división y raíces cuadradas.
- 32 registros de 32 bits tipo float o 16 registros de 64 bits de tipo double.

El conjunto de instrucciones de coma flotante Cortex-M4F no da soporte a todas las operaciones definidas en el estándar IEEE 754-2008. Las operaciones no incluidas en el estándar IEEE754 cuyo uso está disponible son:

- Redondeo de un número de coma flotante a número entero.
- Conversión de binario a decimal y viceversa.
- Comparación directa entre valores de tipo float y double.

Por contra la FPU no da soporte para el cálculo de operaciones con datos de tipo double (64-bits). En lo que a este trabajo concierne, la FPU cobra importancia en las operaciones como son multiplicaciones, divisiones y sumas de números decimales las cuales redundan en un mejor cálculo de los parámetros del sistema y por ende de un ajuste mejor de la respuesta del sistema. Deja así como únicos errores posibles los resultados que atañen a los componentes físicos del sistema y a una posible identificación errónea de la planta.

Existen 3 modos en los que la FPU puede trabajar:

1. Full-compliance: La FPU procesa todas las operaciones de acuerdo a la normativa IEEE 754. Este modo se encuentra por defecto activado.

2. Flush-to-Zero: Activando el bit FZ del control del registro (FPSCR) se activa este modo. Cuando esta activo, valores próximos al cero se tratan como cero incrementando enormemente la velocidad de ejecución a expensas de una menor precisión.
3. NaN (*Not a Number*): Activando el bit DN del registro FPSCR se habilita este modo NaN por defecto. En este modo el resultado de procesar una operación que involucra un NaN o bien genera un NaN, devuelve el NaN.

3.2 Fuentes de reloj

En este apartado se explican las diferentes fuentes de reloj de las que dispone el MSP432 y los registros utilizados para programar el microcontrolador a modo de optimizar sus recursos y así obtener un resultado favorable en su utilización.

El sistema de reloj puede ser configurado para operar sin componentes externos o con hasta dos cristales externos o con resonadores o bien con una resistencia exterior bajo control del software. El sistema de reloj incluye las siguientes fuentes de reloj [8]:

- LFXTCLK: Oscilador de baja frecuencia (LFXT) que se puede utilizar con cristales de baja frecuencia de reloj de 32768 Hz, estándar, resonadores o fuentes de reloj externas en el rango de 32 kHz o por debajo.
- HFXTCLK: Oscilador de alta frecuencia (HFXT) que se puede utilizar con cristales estándar o resonadores en el rango entre 1 MHz y 48 MHz.
- DCOCLK: Oscilador controlado digitalmente (DCO) de frecuencias programable y con 3 MHz como frecuencia por defecto.
- VLOCLK: Oscilador de baja frecuencia y muy bajo consumo (VLO) con 9.4 KHz de frecuencia típica.
- REFOCLK: Oscilador interno de baja frecuencia y bajo consumo (REFO) con frecuencias seleccionables de 32.768 kHz o 128 kHz.
- MODCLK: Oscilador interno de bajo consumo de 25 MHz de frecuencia típica.
- SYSOSC: Oscilador interno con 5 MHz de frecuencia típica.

Del módulo de reloj se encuentran disponibles cinco señales de reloj:

- ACLK: *Auxiliary clock*. Cuya frecuencia máxima de operación es 128 kHz.
- MCLK: *Master clock*. El MCLK es usado por la CPU y directamente por algunos módulos de los periféricos.
- HSMCLK *Subsystem master clock*. HSMCLK es seleccionable vía software por módulos periféricos individuales.
- SMCLK: *Low speed subsystem master clock.* Utiliza el HSMCLK como fuente de reloj. Su frecuencia está limitada a la mitad del máximo de frecuencia de HSMCLK.
- BCLK: *Low speed backup domain clock.* El BCLK tiene su frecuencia restringida a un máximo de 32768 kHz.

Cabe destacar que el MSP432 puede operar a una frecuencia máxima de 48 MHz. Para este proyecto utilizo como fuente de reloj el DCO con una frecuencia de 24MHz que me permitirá obtener una resolución adecuada según los datos a obtener. El DCO está dividido en seis rangos de frecuencia calibrados con un centro de frecuencia para cada rango respectivamente. Estos, son seleccionables mediante los bits DCORSEL.

Haciendo uso de los bits DCOTUNE la aplicación puede sintonizar la frecuencia dentro de un rango seleccionado en cantidades nominales del 0.2% para ajustar la frecuencia de DCO, ya sea aumentando o disminuyéndola, desde la frecuencia central calibrada. Los rangos se solapan con sus adyacentes resultando así en un continuo rango de frecuencias disponibles para su utilización. El DCO puede ser utilizado por las fuentes de reloj MCLK, HSMCLK o SMCLK. Para los modos de bajo consumo LPM3, LPM4, LPM3.5 o LPM4.5 no se encuentra disponible y estaría desactivado [9].

El DCO además puede operar tanto con resistencia interna como externa. Lo primero no requiere ningún componente adicional y está configurado por defecto, con una tolerancia de +/-3%. En cambio para el modo de resistencia externa se necesita conectar una resistencia del pin DCOR a tierra con lo que se puede alcanzar un +/-0.6% de tolerancia. Este modo resulta en una mayor tolerancia general comparada con el funcionamiento de resistencia interna [10]. El modo de resistencia externa se selecciona poniendo DCORES = 1. Para cambiar al modo de resistencia externa se ha de hacer siempre con el DCORSEL por defecto (DCORSEL = 1). Los registros de CSDCOERCAL0 y CSDCOERCAL1 se pueden utilizar para calibrar el DCO en el modo de operación de resistencia externa. Debido a que realizo el cálculo mediante el conteo por pulsos, este no necesita de una gran precisión de reloj, es por ello que con el modo resistencia interna y haciendo uso de una de las frecuencias centrales calibradas (24 MHz) será suficiente.

Todos los registros del reloj están protegidos con una contraseña excepto (CSSTAT) y (CSIFG). Esta contraseña está definida en el registro CSKEY de la tabla 3-2 que controla el acceso a los registros del reloj. Para habilitar la escritura en los registros del reloj se escribe la contraseña correcta en el CSKEY. Para inhabilitar acceso a escritura, se escribe cualquier valor en el CSKEY [11].

Bit	Field	Type	Reset	Description
31-16	Reserved	R	0h	Reserved. Always reads as 0.
15- 0	CSKEY	RW	A596h	Write CSKEY = xxxx_695Ah to unlock CS registers. All 16 LSBs need to be written together. Writing CSKEY with any other value causes CS registers to be locked and any writes to these registers are ignored, while reads are still performed. Always reads back A596h.

Tabla 3-2 Modos de funcionamiento del Reloj

A continuación se presenta la tabla de los registros control del reloj DCO.

Bit	Field	Type	Reset	Description
31-25	Reserved	R	0h	Reserved. Always reads as 0.
24	Reserved	RW	0h	Reserved. Must be written as zero.
23	DCOEN	RW	0h	Enables the DCO oscillator regardless if used as a clock resource. 0b = DCO is on if it is used as a source for MCLK, HSMCLK , or SMCLK and clock is requested , otherwise it is disabled. 1b = DCO is on.
22	DCORES	RW	0h	Enables the DCO external resistor mode. 0b = Internal resistor mode 1b = External resistor mode
21-19	Reserved	R	0h	Reserved. Always reads as 0.
18-16	DCORSEL	RW	1h	DCO frequency range select. Selects frequency range settings for the DCO. 000b = Nominal DCO Frequency (MHz): 1.5; Nominal DCO Frequency Range (MHz): 1 to 2 001b = Nominal DCO Frequency (MHz): 3; Nominal DCO Frequency Range (MHz): 2 to 4 010b = Nominal DCO Frequency (MHz): 6; Nominal DCO Frequency Range (MHz): 4 to 8 011b = Nominal DCO Frequency (MHz): 12; Nominal DCO Frequency Range (MHz): 8 to 16 100b = Nominal DCO Frequency (MHz): 24; Nominal DCO Frequency Range (MHz): 16 to 32 101b = Nominal DCO Frequency (MHz): 48; Nominal DCO Frequency Range (MHz): 32 to 64 110b to 111b = Nominal DCO Frequency (MHz): Reserved--defaults to 1.5 when selected; Nominal DCO Frequency Range (MHz): Reserved--defaults to 1 to 2 when selected.
15-13	Reserved	R	0h	Reserved. Always reads as 0.
12-10	Reserved	RW	0h	Reserved. Must be written as zero.
9-0	DCOTUNE	RW	0h	DCO frequency tuning select. 2s complement representation. Value represents an offset from the calibrated center frequency for the range selected by the DCORSEL bits.

Tabla 3-3 Modos de funcionamiento del reloj

Como se observa en la tabla 3-3 del registro de control 0 (CSCTL0) desde este registro se elige la frecuencia central calibrada a utilizar. La fuente de reloj a utilizar por el *MCLK,* y por *SMCLK* y *HSMCLK* se establece desde el CSCTL1. Además en este registro se pueden establecer las divisiones para las señales de reloj. En la figura 3.2 se muestra el ajuste del reloj.

```
void Iniclock (void){
      CSKEY = 0x695A; // habilitación de escritura de los registros CS
      CSCTL0 = 0; // reset DCO parámetros
      CSCTL0 = DCORSEL_4; // selección DCO 4 (24MHz)
      CSCTL1 |= SELS_3| SELM_3; // SMCLK, HSMCLK = DCOCLK  MCLK=DCOCLK
      CSKEY = 0; // deshabilitar escritura de los registros CS
```

Figura 3.2 Configuración del reloj DCO

3.3 Temporizadores

El MSP432 cuenta con 4 timers o contadores de 16 bits configurables por software que tienen hasta siete registros de captura/comparación cada uno. Estos timers pueden admitir comparaciones/capturas, generar señales PWM, asi como temporización de intervalos.

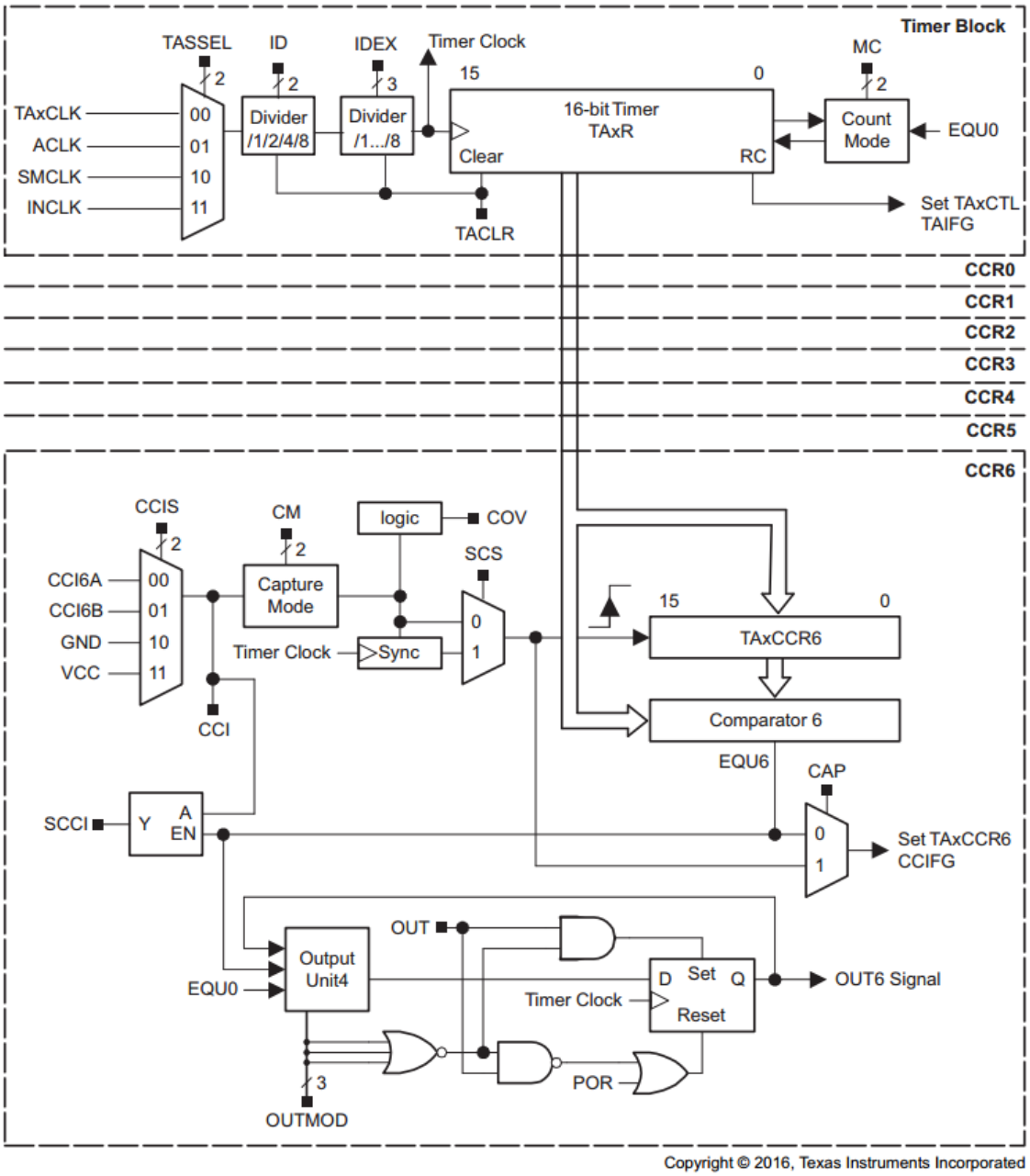


Figura 3.3 Modos de funcionamiento del Timer

Como se observa en la figura 3.3, el registro TAxR (refiriéndome con x a los 4 timers disponibles) incrementa o decrementa con cada flanco de subida de la señal de reloj. Este registro es legible y se puede escribir vía software [12]. Además cuando el contador desborda puede generar una interrupción. El reloj del timer puede tener como fuente las señales de reloj de ACLK, SMCLK o externamente por TAxCLK o INCLK. En caso de tener una fuente externa de reloj los primeros dos pulsos de reloj se pierden y se sincroniza en el tercero. Las fuentes de reloj son seleccionables desde los bits TASSEL. Estas además pueden ser divididas entre los valores de 2, 4 u 8 antes de asignarse al timer usando los bits ID.

Para este trabajo no hará falta puesto que al fijar la velocidad del reloj en 24 MHz y muestrear cada 2 ms utilizo 48000 de los 65536 (2^{16}) valores del timer. Este valor es

cierto para el cálculo de la velocidad por pulsos, en cambio para el conteo por diferencia de captura de flancos el tiempo de muestreo es de 1ms y por lo tanto este valor máximo del TAnCCR0 es 23999. Al mismo tiempo se hace uso de otro timer para generar la señal de PWM con un ajuste en el valor de desborde de 2000 ciclos de reloj. Los 4 modos de operación son:

1. Stop: El timer se encuentra parado.
2. Up: El timer cuenta repetidamente desde cero hasta el valor del TAxCCR0. El flag de interrupción del TAxCCR0 CCIFG se pone a 1 cuando el contador del timer llega al valor TAxCCR0. El flag de TAIFG de interrrupción se pone a 1 cuando cuenta de TAxCCR0 a cero. La figura 3.4 lo muestra.

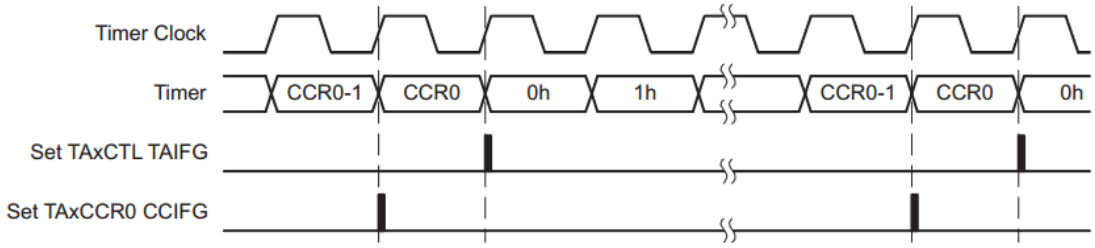


Figura 3.4 Modos de funcionamiento del Timer

3. Continuous: El timer cuenta de cero hasta el valor máximo del timer 0FFFFh. En este caso solo se activaría el flag TAIFG al igual que en el modo UP.
4. Up/down: El timer cuenta repetidamente desde cero hasta el valor de TAxCCR0 y de nuevo hasta cero por lo que el periodo es el doble de TAxCCR0. En este modo los flags de CCIFG y TAIFG se ponen a 1 una sola vez durante el periodo. CCIFG se activa cuando el timer cuenta de TAxCCR0-1 a TAxCCR0, y el TAIFG cuando cuando el timer termina de contar de 0001h a 0000h. La figura 3.5 siguiente lo muestra.

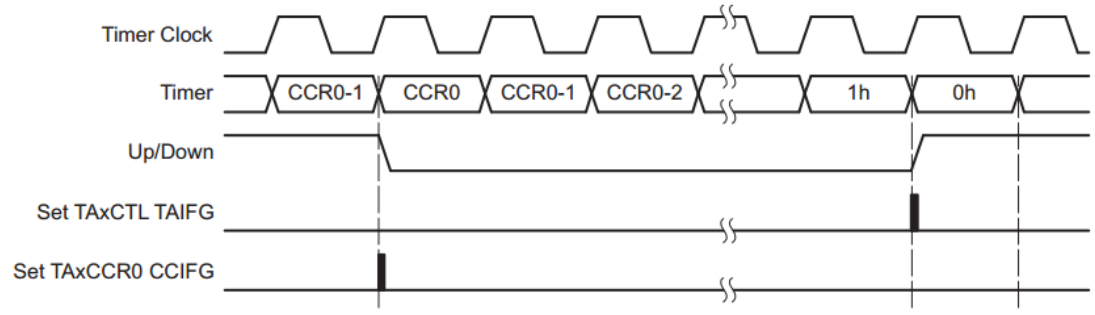


Figura 3.5 Modos de funcionamiento del Timer

De los modos mencionados previamente hago uso del Up, es decir, cuento hasta un valor determinado previamente y acto seguido retorna el contador del timer a cero.

3.3.1 Captura y comparación

Hasta 7 bloques de comparación y captura se encuentran disponibles en el Timer_A. Cualquiera de ellos puede ser utilizado tanto para capturar datos de tiempo como para generar intervalos de tiempo [13].

El modo captura se seleccciona activando el bit CAP del registro TAxCCTLn (n representa el un registro específico de captura y comparación). El bit CM selecciona si la captura se hace o bien en un flanco de subida o en uno de bajada o en ambos. Cuando se produce una captura el valor del timer en ese instante es copiado al registro TAxCCRn y el flag TAxCCRn CCIFG se activa. La señal de captura puede ser asíncrona al reloj del timer así que activando el bit SCS se logra sincronizar esta con la siguiente señal del reloj del timer. Si durante una captura se produce otra, previa a la lectura de la primera, el bit COV del registro TAxCCTLn se activa, dicho bit se tendría que poner a cero por software.

En el modo comparación es necesario que el bit CAP = 0, este modo es el usado para generar la señal de PWM. Cuando el TAxR llega al valor de TAxCCRn, el flag CCIFG se activa y la señal interna EQUn = 1. Cada bloque de salida contiene una unidad de salida y cada unidad tiene 8 modos de generar una señal. Estos son los mismos que en el MSP430 y se seleccionan con el bit OUTMOD_x del registro TAxCCTLn. Los modos se muestran en la tabla 3-4:

OUTMOD_x	Modo	El timer alcanza el valor de TAxCCRn	El timer alcanza TAxCCR0
001	Set	Se pone a 1	-
010	Toogle/Reset	Conmuta	Se pone a 1
011	Set/Reset	Se pone a 1	Se pone a 0
100	Toggle	Conmuta	-
101	Reset	Se pone a 0	-
110	Toggle/Reset	Conmuta	Se pone a 1
111	Reset/Set	Se pone a 0	Se pone a 1

Tabla 3.4 Modos de funcionamiento del Timer

Para este trabajo será el modo OUTMOD_7 el que rija la señal de salida, que corresponde a conmutar a 0 la señal una vez alcanza un valor determinado TAxCCRn y a 1 cuando el valor alcance el TAxCCR0. En la figura 3.6 se observa la configuración del timer para este trabajo:

```
void IniTimer (void){
      TA0CTL |= TASSEL_2+ MC_1 +TAIE; //fuente reloj= SMCLK, Up
      TA0CCR0 = 47999;
      TA2CTL |= TASSEL_2+ MC_1 +TAIE;
      TA2CCTL1 = OUTMOD_7;
      TA2CCR0 = 2000;
      TA2CCR1 = 0;
```

Figura 3.6 Código C de configuración del Timer

3.4 Interrupciones

En el siguiente apartado se especificarán los tipos de interrupciones que pueden darse, sus modos de operación , como se genera la señal de PWM. Asimismo se explican el vector de interrupción *Nested Vector Interrupt Controller* (NVIC) incorporado en la arquitectura

del Cortex-M4 y que dota al micro de un nuevo modo de procesar y ordenar tanto las interrupciones como el resto de excepciones con respecto al MSP430. En primer lugar se enumeran algunos tipos de excepciones que se pueden dar [14]:

1. Reset: Esta excepción es tratada de forma especial frente al resto. Cuando se da, las operaciones que pudiera estar realizando el procesador se paran y al restablecerse del reset la ejecución de instrucciones se reanuda desde la dirección almacenada en la tabla de vectores.
2. NMI: La *Nonmaskable Interrupt (NMI)* puede ser activada por una señal NMI o mediante software utilizando el registro de control de estados (ICSR). Esta tiene la prioridad mayor despues del reset.
3. Hard fault: Esta es aquella que se produce cuando se da algún tipo de error durante el procesado de excepciones. Este tipo de excepciones tiene una prioridad máxima en comparación con cualquier otro tipo de excepción configurable.
4. Memory Management fault: Producido debido a un fallo relacionado con la proteción de memoria.
5. Bus fault: Se da con acceso a vacantes en el mapa de memoria y accediendo a perifericos de 16 bits con accesos de 32.
6. Usage fault: Es una excepción producida debido a un fallo en la ejecución de una instrucción.
7. Interrupt (IRQ): La interrupción o IRQ es una excepción que tiene lugar en un periférico o que bien es generada mediante software y que es tratada por el NVIC. Todas las interrupciones son asíncronas a la ejecución de instrucciones.

Las excepciones pueden estar en uno de los estados siguientes:

1. Inactiva: La excepción ni está activada ni pendiente
2. Pendiente: La excepción está en espera ser atendida por el procesador. Una petición de interrupción de un periférico por ejemplo puede cambiar el estado de la interrupción a pendiente.
3. Activa: Se da cuando una excepción es atendida por el procesador pero aún no se ha completado. El gestor de excepciones puede interrumpir una interrupción para servir a otra de mayor prioridad, en este caso ambas quedarían en estado activo.
4. Activa y pendiente: Si mientras que se atiende la interrupción se da otra de la misma fuente, que devuelve el estado de la interrupción a pendiente.

3.4.1 NVIC

Lo que en principio se debe saber del NVIC es que se trata de un vector de interrupciones anidadas que ordena y prioriza entre ellas una vez se dan sucesión. Este vector da soporte a [15]:

- 64 Interrupciones
- Prioridad programable en 7 niveles para cada interrupción, donde un mayor nivel corresponde a una menor prioridad y viceversa.
- Baja latencia de procesado de excepciones e interrupciones
- Detección de pulsos y de niveles en el procesado de interrupciones
- Priorización dinámica de las interrupciones
- Interrupciones encadenadas
- Agrupación de valores de prioridad en campos de prioridad y subprioritarios.
- Interrupciones no enmascarables (NMI), cuyas interrupciones no se pueden deshabilitar.

El procesador admite tanto interrupciones de nivel como por pulso. Estas últimas también descritas como interrupciones de disparo por flanco. Una interrupción de nivel se mantiene hasta que es borrada por la *Interrupt service routine* (ISR) que accede al periférico y borra la peticion de interrupción. La interrupción de pulso es aquella que es muestreada de forma sincrónica en el flanco de subida del reloj del procesador. A modo de garantizar que el NVIC pueda detectar esta interrupción, el periférico debe confirmar esta señal durante mínimo un ciclo de reloj, el NVIC detectará el pulso y bloqueará el periférico.

Si el procesador entra en la ISR, borra automáticamente el estado de pendiente de la interrupción. En las interrupciones de nivel si la señal no está anulada antes de que el procesador regrese de la ISR, la interrupción pasa a pendiente y el procesador retorna a ejecutar la ISR. La señal pasa a pendiente produciendose uno de los siguientes acontecimientos:

- El NVIC detecta que una señal de interrupción esta en alto y la interrupción no esta activada.
- El NVIC detecta un flanco de subida en la señal de interrupción
- Por software se escribe en el bit correspondiente de interrupción pendiente.

Como se ha comentado previamente, el NVIC permite habilitar o inhabilitar las 64 interrupciones para ello consta de dos registros de 32 bits respectivamente en los cuales cada bit corresponde a una interrupción determinada. Los registros ICER0 e ICER1 y ISER0 e ISER1 permiten inhabilitar y habilitar respectivamente las excepciones disponibles. ICER0 e ISER0 modifican los bits del 0 al 31 correspondientes a las 32 primeras expcepciones y ICER1 e ISER1 a los bits correspondientes a las 32 siguientes excepciones .

3.4.2 **Tabla vector del NVIC**

Esta tabla contiene el valor de reset del *Stack Pointer (SP)* y los vectores de excepción o las direcciones de cada una de las excepciones. Esta tabla es modificable en el archivo *msp432_startup_ccs.c* generado cada vez que se crea un proyecto en el depurador *Code Composer Studio (CCS)* y accesible desde el *Proyect Explorer* de CCS.

En ella están listadas cada una de las excepciones y la posición que adquieren dentro de los dos registros que controlan la habilitación y borrado de las interrupciones. Para que cada vez que se produzca una interrupción se llame a la función que ha de procesarse con esa interrupción se ha de escribir el nombre de la función que se quiere llamar en el listado y en la posición correspondiente a la interrupción, como en la tabla 3-5, de esta forma se identifica la función como ISR.

```
void (* const interruptVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_END),
                                        /* The initial stack pointer */
    resetISR,                           /* The reset handler         */
    nmiISR,                             /* The NMI handler           */
    faultISR,                           /* The hard fault handler    */
    defaultISR,                         /* The MPU fault handler     */
    defaultISR,                         /* The bus fault handler     */
    defaultISR,                         /* The usage fault handler   */
    0,                                  /* Reserved                  */
    0,                                  /* Reserved                  */
    0,                                  /* Reserved                  */
    0,                                  /* Reserved                  */
    defaultISR,                         /* SVCall handler            */
    defaultISR,                         /* Debug monitor handler     */
    0,                                  /* Reserved                  */
    defaultISR,                         /* The PendSV handler        */
    defaultISR,                         /* The SysTick handler       */
    defaultISR,                         /* PSS ISR                   */
    defaultISR,                         /* CS ISR                    */
    defaultISR,                         /* PCM ISR                   */
    defaultISR,                         /* WDT ISR                   */
    defaultISR,                         /* FPU ISR                   */
    defaultISR,                         /* FLCTL ISR                 */
    defaultISR,                         /* COMP0 ISR                 */
    defaultISR,                         /* COMP1 ISR                 */
    defaultISR,                         /* TA0_0 ISR                 */
      velocity,                         /* TA0_N ISR                 */
    defaultISR,                         /* TA1_0 ISR                 */
    defaultISR,                         /* TA1_N ISR                 */
    defaultISR,                         /* TA2_0 ISR                 */
    defaultISR,                         /* TA2_N ISR                 */
    defaultISR,                         /* TA3_0 ISR                 */
    defaultISR,                         /* TA3_N ISR                 */
    defaultISR,                         /* EUSCIA0 ISR               */
    defaultISR,                         /* EUSCIA1 ISR               */
    defaultISR,                         /* EUSCIA2 ISR               */
    defaultISR,                         /* EUSCIA3 ISR               */
    defaultISR,                         /* EUSCIB0 ISR               */
    defaultISR,                         /* EUSCIB1 ISR               */
    defaultISR,                         /* EUSCIB2 ISR               */
    defaultISR,                         /* EUSCIB3 ISR               */
    defaultISR,                         /* ADC14 ISR                 */
    defaultISR,                         /* T32_INT1 ISR              */
    defaultISR,                         /* T32_INT2 ISR              */
    defaultISR,                         /* T32_INTC ISR              */
    defaultISR,                         /* AES ISR                   */
    defaultISR,                         /* RTC ISR                   */
    defaultISR,                         /* DMA_ERR ISR               */
    defaultISR,                         /* DMA_INT3 ISR              */
    defaultISR,                         /* DMA_INT2 ISR              */
    defaultISR,                         /* DMA_INT1 ISR              */
    defaultISR,                         /* DMA_INT0 ISR              */
    defaultISR,                         /* PORT1 ISR                 */
    defaultISR,                         /* PORT2 ISR                 */
    defaultISR,                         /* PORT3 ISR                 */
      port4_ISR,                    /* PORT4 ISR
```

Tabla 3-5 Parte de la tabla del NVIC

Su posición dentro de este listado indica la prioridad que tiene cada una de las excepciones. Las funciones ISR serán declaradas previamente como externas, tal y como se observa en la figura 3.7.

```
46 /* External declaration for the reset handler that is to be called when the */
47 /* processor is started                                                      */
48 extern void _c_int00(void);
49 extern void velocity(void);
50 extern void port4_ISR( void );
```

Figura 3.7 Declaración de funciones externas

Si bien la prioridad de las excepciones es modificable, las tres primeras, correspondientes a las excepciones de RESET, NMI y HARD FAULT no son modificables y por lo tanto siempre tendrán la mayor prioridad.

Las IRQ o interrupciones tienen su prioridad modificable vía software la cual se divide en 8 rangos de importancia que van desde el 0 al 7. En principio parten todas las IRQ con el máximo de prioridad que es 0, y es por esto que si se diera el caso de que varias interrupciones se encuentran en estado pendiente al mismo tiempo, sería la posición de las mismas en el vector del NVIC la que determinase su preferencia. Asimismo, si se da el caso de que se está ejecutando una excepción y se da otra con mayor rango de preferencia, esta primera se detiene y se ejecuta esta última. Finalmente, en el caso de que tuviera el mismo rango de prioridad, se ejecutaría aquella que estuviera activada en ese momento y dejaría en estado de pediente la última IRQ independientemente de su orden en el vector de excepciones.

Para modificar este orden de preferencia se hace uso de los registros de IPRn (con n como valor indicativo del grupo de instrucciones al que hace referencia). El NVIC cuenta con 16 registros de IPRn que dividen las 64 excepciones plausibles. Cada uno de los IPRn es un registro de 32 bits dividido en 4 que posibilita establecer el orden de prioridad de las excepciones pertenecientes a ese registro. Como se observa en la tabla 3-6, que hace referencia al IPR6 se asignan las prioridades para las excepciones 24, 25, 26 y 27 que en este caso corresponden a las interrupciones de ADC14, TIMER32_INT1, TIMER32_INT2y TIMER32_INTC referentes al timer de 32bits incorporado en el MSP432 [16].

31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
PRI_27	PRI_26	PRI_25	PRI_24
R/W-0h	R/W-0h	R/W-0h	R/W-0h

Tabla 3-6 Registro de preferencia de interrupciones

El mecanismo de prioridad del hardware solo tiene en cuenta los 3 primeros bits de cada subgrupo es decir, los de mayor valor. Para establecer como ejemplo la IRQ[8] que corresponde a la interrupción 8 con un valor de prioridad de 1, NVIC_IPR3 = 0x20; En este proyecto cobra especial importancia el orden de preferencia de las interrupciones ya que este afecta directamente a la medición de los pulsos generados por el encoder y por ende a la resolución final de la respuesta. A continuación se describen brevemente las funciones programadas que se dan al producirse las diferentes interrupciones y que servirán para explicar el grado de importancia en lo referente a establecer un orden de prioridad de las mismas:

1. **Conteo de pulsos del enconder (port4_ISR)**: Cada vez que se detecta un flanco de subida en el puerto establecido para ello, en este caso el puerto 4 pin 4, se aumenta en 1 el valor de un contador y se desactiva por software el flag de interrupción.
2. **Cálculo de velocidad y control (velocity)**: Esta función se producirá cada vez que el contador TAR del Timer TA0 desborde y tiene por tarea calcular la

velocidad del motor utilizando el valor acumulado en el conteo de pulsos y al mismo tiempo calcular la acción del sistema a controlar que permita regular el duty de la señal PWM para conseguir una respuesta adecuada a los requerimientos preestablecidos.

Las funciones previamente descritas están escritas siguiendo el rango de prioridad que se establece. Siendo conocido el modo de operación del NVIC en lo referente al establecimiento de prioridades, se ha de tener en cuenta los tiempos de procesado de cada función. Puesto que es necesario máxima precisión a la hora de contar el nº de pulsos, será esta función la que adquiera la mayor prioridad y posteriormente la de cálculo de velocidad y control que es la que más ciclos de reloj consume y por lo tanto si estuviera en igualdad de preferencia con la de conteo de pulsos, y esta estuviera activada, se estaría perdiendo información referente al nº de pulsos. A continuación en la figura 3.7 el código referente a esta configuración junto con la activación de las correspondientes interrupciones.

```
NVIC_ISER0 = 1 << (9); //Habilitación de las interrupciones (TA0_N)
NVIC_ISER1 = 1 << (6); //Habilitación interrupciones Puerto 4
NVIC_IPR2 = 0x4000; //Prioridad 2 para la interrupción 9
__enable_interrupt(); //Habilitación de las interrupciones
```

Figura 3.7 Configuración de las interrupciones

Como se ha explicado previamente los registros de escritura (que no de borrado, que son ICER0 e ICER1) ISER0 e ISER1 sirven para activar el bit que habilita cada interrupción que en estos casos corresponden a TA0_N referente a los registros de captura/comparación TA0CCRx y al desborde del timer0. y a la interrupción 38 de la tabla NVIC cuyo bit de activación dentro del registro ISER1 es 6 corriespondiente al puerto 4.

3.4.3 Configuración de los puertos

En este subapartardo se explica brevemente la configuración de los puertos mediante la cual se habilitan también los puertos para las interrupciones [17]. En primer lugar decir que los puertos de *In/Out* (I/O) digitales son programables individualmente, con interrupciones configurables disponibles para algunos puertos con registros de entrada/salida independientes. Este dispositivo tiene 10 puertos de I/O cada uno de ellos con hasta 8 lineas I/O. Cada una de las lineas es configurable como resistencias de *pullup* o *pulldown*. Algunos de los puertos pueden generar interrupciones las cuales pueden ser activadas tanto con un flanco de subida como de bajada. Como en el MSP430 algunos de los registros de configuración son:

- Entrada (PxIN): De lectura únicamente. Indica el valor de la señal en el pin de I/O

- Salida (PxOUT): Si el pin es configurado de entrada y las resistencias de *pullup* o *pulldown* están activas este pin selecciona *pullup* o *pulldown.* En caso de que se configure de salida el bit marca si la salida está en alto o bajo.
- Dirección (PxDIR): Registro para seleccionar un pin de entrada o salida.
- Pullup/Pulldown (PxREN): Que activa o desactiva las resistencias *pullup/pulldown.*
- Función (PxSEL0, PxSEL1): Como los pines de los puertos están multiplexados con otras funciones de los módulos periféricos. Mediante estos registros se selecciona la función del pin.

Las interrupciones de los puertos, activadas mediante el bit PxIE, al igual que en los timers son priorizadas y aquella con la prioridad más elevada genera un número en el registro PxIV que determina la fuente de interrupción. Para activar las interrupciones es necesario activar el bit PxIE correspondiente del puerto y pin. Una vez se produzca una interrupción se activará el flag PxIFG que indica el interrupción pendiente o no. Solo las transiciones de nivel producen interrupciones. La configuración del bit PxIES para el puerto indica si se activa el flag PxIFG en una transición de baja a alta o de alta a baja. Para este trabajo la función de conteo de pulsos de la figura 3.9 se trata de un contador que aumenta su valor en uno cada vez que se produce una interrupción en el puerto 4 pin 4, producida por una transición en la señal digital recibida del encoder. El flag de interrupción se desactiva por software.

```
void port4_ISR( void ){

    x= x +1;
    P4IFG &= ~BIT4;
  }
```

Figura 3.9 Función conteo de pulsos

3.5 Interfaz gráfica GUI

En este apartado se explicará el programa final que se ha utilizado para la tarea de uso de una interfaz gráfica, la ayuda que supone para este proyecto, y la forma en que sus diversos *gadgets* se ponen en práctica.

Tras evaluar la herramienta de *Processing* como posible software libre en el que programar una interfaz gráfica y compararla con la que nos ofrece *Texas Instruments* en el *Code Composer Studio (CCS)* se hará uso de esta última por su complementación con el programa y su facilidad tanto visual como práctica a la hora de crear un entorno de fácil comprensión y de uso sencillo. La *GUI Composer* permite crear aplicaciones con diversos gadgets que proporcionan visibilidad a lo que sucede en la aplicación así como la capacidad de controlar las variables del programa. Asimismo, se da la posibilidad de crear aplicaciones *Standalone* para que estas sean visibles y controlables fuera del entorno de CCS [18].

Dentro de la GUI existen dos modelos de aplicación:

- Modelo programa: El compilador CCS es usado para traducir los símbolos en direcciones. Se usan conexiones vía JTAG o serie, además las escrituras en la tarjeta se hacen o bien vía JTAG o mediante comandos a un monitor que se ejecuta en el dispositivo.
- Modelo Streaming: Se usan conexiones serie o Ethernet y los datos son enviados desde la tarjeta hasta el host.

Es posible ajustar el valor de los datos que se muestran. Esto es, puede que los datos sean almacenados en una unidad de medida en el dispositivo, pero es posible que se quiera utilizar con otra unidad de medida en la GUI. Para esto están las funciones de pre-procesado y post-procesado que permiten respectivamente coger el valor y ajustarlo para su uso en el display, y coger el valor que entra a la aplicación y ajustarlo previamente a la escritura en el dispositivo.

Cuando se crea un nuevo proyecto de GUI se genera un archivo .html. Al arrastrar en la página los diferentes gadgets a utilizar, se escribe el mismo. La GUI ofrece un gran número de gadgets que se pueden usar y personalizar.

Para este trabajo y como se observa en la figura 3.8, se han utilizado los siguientes:

- **Diagrama**: De los disponibles se hace uso del que representa valor almacenado en un vector frente a su posición en el mismo. De esta forma y ajustando que cada posición del vector varíe tras cada periodo de muestreo se obtendrá el desarrollo del valor que se almacena en el tiempo. Este gráfico permite representar hasta 8 vectores de datos diferentes y personalizar cada uno indistintamente. Para este trabajo cobra especial importancia la representación de la velocidad del motor calculada y de la acción correspondiente en cada instante.
- **Dial**: Que marcará el valor de la referencia y permitirá un control de la misma más rápido y directo a la par que impreciso.
- **2 contadores**: Con el primero (*referencia*), marco de forma precisa el valor de la referencia que quiero, el segundo (C*ontroler start at*), marca la posición de los vectores de salida y acción a partir del cual el sistema pasa a estar en Bucle cerrado, y por lo tanto a ser sistema controlado.
- **Casillas**: 2 casillas una (*Start/Stop*) para dar comienzo a la representación de datos y otra (*Controlador On*) para activar o desactivar el controlador.

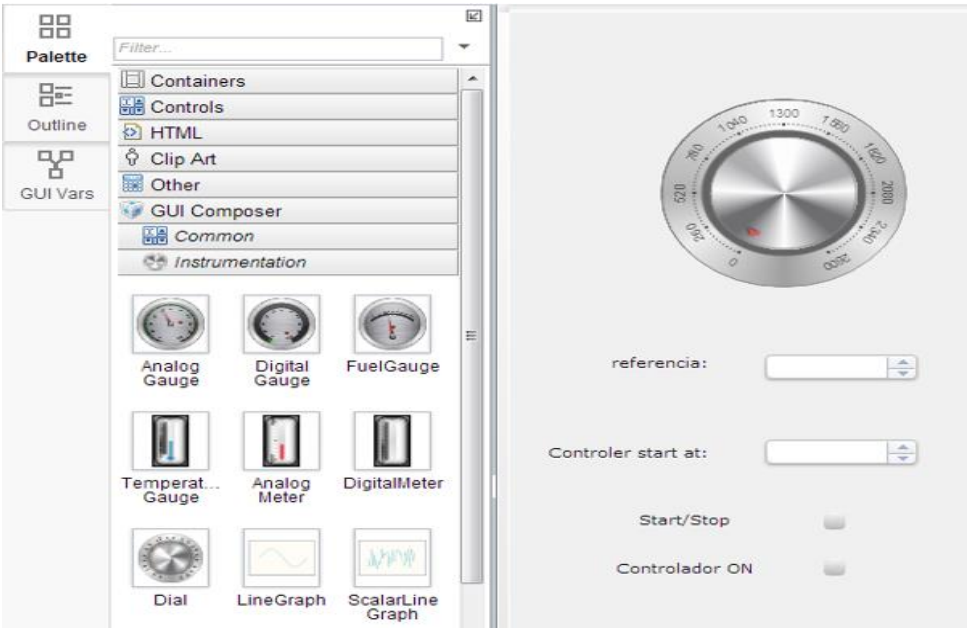


Figura 3.8 Panel de configuración de la GUI

Cada uno de los gadgets se puede unir a una o más variables dependiendo del tipo de gadget. Estos se unen a variables declaradas de tipo *volatile* dentro del programa. De esta forma quedan vinculados y son modificables dentro de la GUI en tiempo real.

Además siguiendo la wiki de Texas Instruments [19] encuentro disponibles algunos ejemplos de utilidad como el del que hago uso. Modificando el *app.js* que se genera tras crear un proyecto se pueden incluir funciones que procesen la información tanto de entrada y salida a la GUI de nuestra aplicación. En concreto he creado unos gadgets, figura 3.9 que junto con el código de *app.js* permiten guardar los valores almacenados en el vector de velocidad en un archivo, durante el tiempo que la casilla *logging enabled* está activa, en una locación a elección.

Figura 3.9 Gadgets de almacenamiento de datos

Para concluir una vez se carga en la memoria flash del micro el programa, eligiendo previamente el tipo de microcontrolador a utilizar, el proyecto CCS y el tipo de conexión, se puede exportar el proyecto de la app a una carpeta específica y lanzar la aplicación. La figura 3.10 muestra la visualización del panel de control de la app en funcionamiento *Standalone*. En concreto muestra la evolución de la acción (rojo) y de la respuesta (azul) ante diversos cambios de referencia.

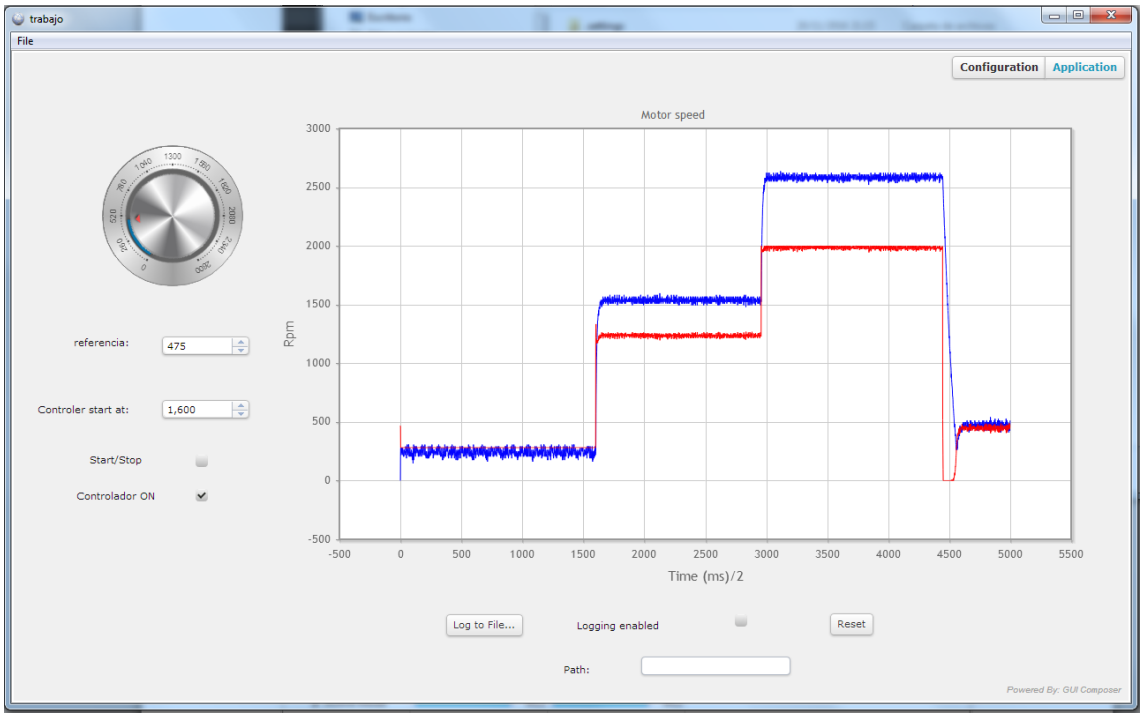


Figura 3.10 Aplicación funcionando en Standalone

Nota: El programa de GUI Composer Runtime es necesario para que la aplicación funcione en el modo Standalone sin embargo no estaba actualizado y era erróneo para la versión de CCS 6.1.2, tras comunicar con los técnicos de Texas Instruments y ver donde estaba el fallo, me enviaron la versión 6.2.0.1 con la cuál, ya es posible trabajar. De aquí en adelante tanto para esta versión de CCS como las posteriores será necesaria esta versión para el correcto funcionamiento de la aplicación en Standalone.

3.Diseño del controlador digital

4.1 Software libre Octave

Como parte de este proyecto se analiza la posibilidad y el uso de herramientas de software libre y su calidad a la hora de utilizarlas en el ámbito que me ocupa. Es por ello que he elegido trabajar con Octave por su similitud de sintaxis en los comandos con el programa de MATLAB y que me brinda unas herramientas que son de gran utilidad.

Antes de nada hay que saber que según la WIKIPEDIA "Octave fue concebido como un programa para el análisis numérico y más concretamente para el diseño de reactores químicos". Hoy en día se puede afirmar que Octave es una herramienta en continuo desarrollo y que gracias a los paquetes de instalación gratuita uno puede hacer uso del mismo para usos en diversos ámbitos ya sean estadísticos, de elementos finitos, procesamiento de imágenes, etc. El uso que voy a hacer de OCTAVE será tanto para la simulación del sistema de control así como para el cálculo de un controlador que me dé una respuesta adecuada a mi sistema.

Es por ello que los paquetes de instalación del que haré uso son el de control y el de algebra lineal. La versión sobre la que voy a trabajar es la 4.0.1 la cual dispone de una GUI y un editor de textos, es decir, no hace falta trabajar sobre la pantalla de comandos.

GNU Octave por el momento carece de un entorno gráfico de bloques como tiene Matlab en simulink con bibliotecas de bloques que facilitan enormemente la edición del sistema de control. Es decir, en sistemas con realimentación, varios bloques, diversas entradas etc, sería necesario introducir un conjunto de comandos para unir en un único modelo las distintas características del sistema de control [20].

Como opción también disponible, se puede utilizar la versión online de OCTAVE de recursos ligeramente limitados respecto a la de descarga que ofrece, si uno se registra, la opción de cargar ficheros .m y guardarlos asociándolos a una cuenta de correo. Esto se considera de gran utilidad puesto que permite trabajar en cualquier parte sin necesidad de cargar con una memoria y el programa instalado en el ordenador.

Es por ello que conociéndose las carencias que tiene Octave a nivel de aplicación en comparación con MATLAB, su utilización solo puede entenderse por el hecho de que mientras Octave se encuentra disponible de forma gratuita, la licencia de MATLAB representa un coste considerable.

4.2 Diseño y simulación del controlador

En este siguiente apartado se va a explicar un método para hallar el controlador del sistema. Se utilizarán las funciones de Octave que serán de ayuda a la hora de minimizar el tiempo de cálculo y que además darán soporte visual a los mismos.

En primer lugar dado que tras la identificación experimental del sistema se concluye que se trata de un sistema de primer orden, la idea es realizar un controlador que permita cumplir determinadas especificaciones, como son: estabilidad, tiempo de respuesta adecuado, eliminación de las perturbaciones y nulo error de posición. En mi caso las especificaciones van a ser en el dominio de la frecuencia:

- Frecuencia de corte (wc=100rad/s)
- Margen de fase (Mf=70)

Con estas especificaciones hallo el controlador PI de la ecuación (5).

$$C(s) = \left(Kp + \frac{Ki}{s}\right) \tag{5}$$

$$C(s) = (1.0583 + \frac{121.9874}{s})$$

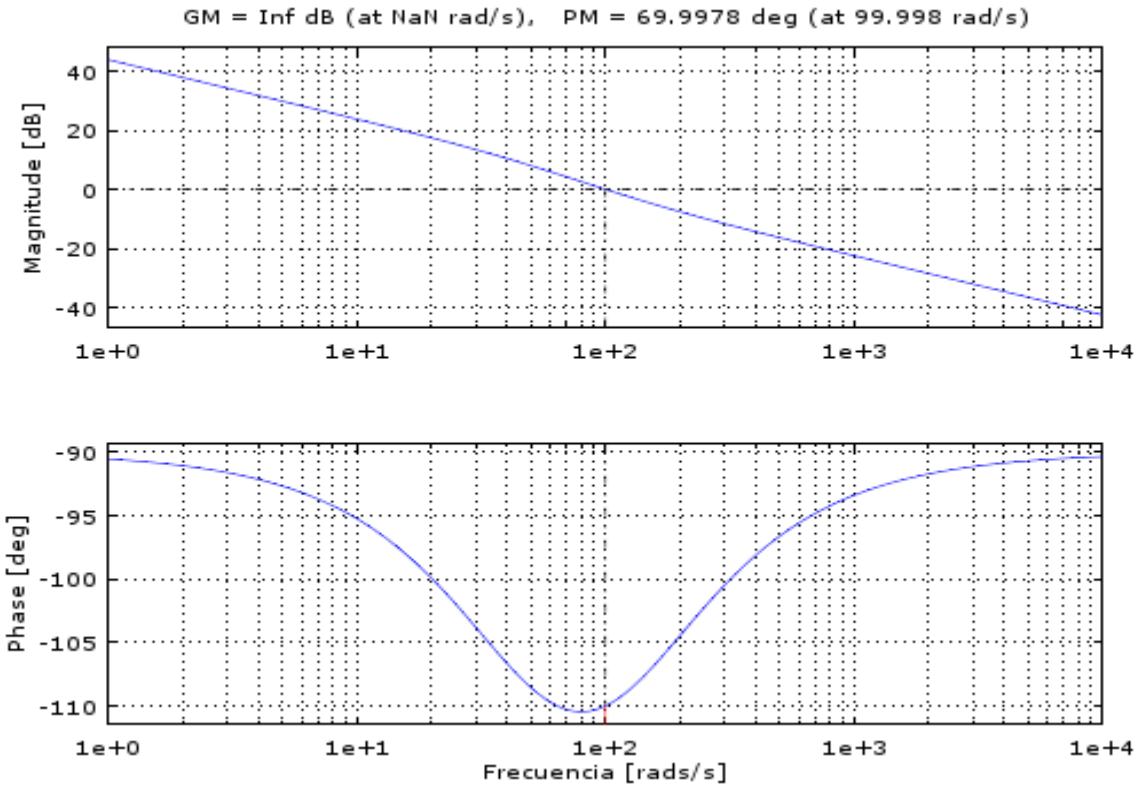


Figura 4.1 Bode del sistema en Bucle abierto

Como se observa en la figura 4.2 correspondiente al sistema de controlador y planta en bucle abierto cuyo bode se representa en la figura 4.1, se muestra que la implementación del controlador PI resulta un filtro de paso bajo, el cuál atenúa aquellas frecuencias superiores a la frecuencia de corte, además eliminará el error en la región de estabilidad del sistema. Con un margen de ganancia infinito el sistema es estable e independiente de la ganancia. La ecuación (6) muestra el controlador discretizado mediante la aproximación de tustin.

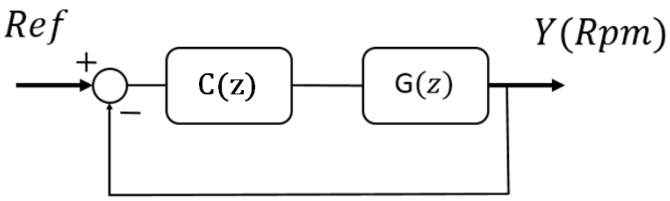


Figura 4.2 Sistema de control discretizado y realimentado

$$\frac{u(z)}{e(z)} = \frac{1.18z - 0.9363}{z - 1} \tag{6}$$

En la figura 4.3 se representa la respuesta a una entrada escalón de 1800 rpm y la evolución de la acción en el tiempo.

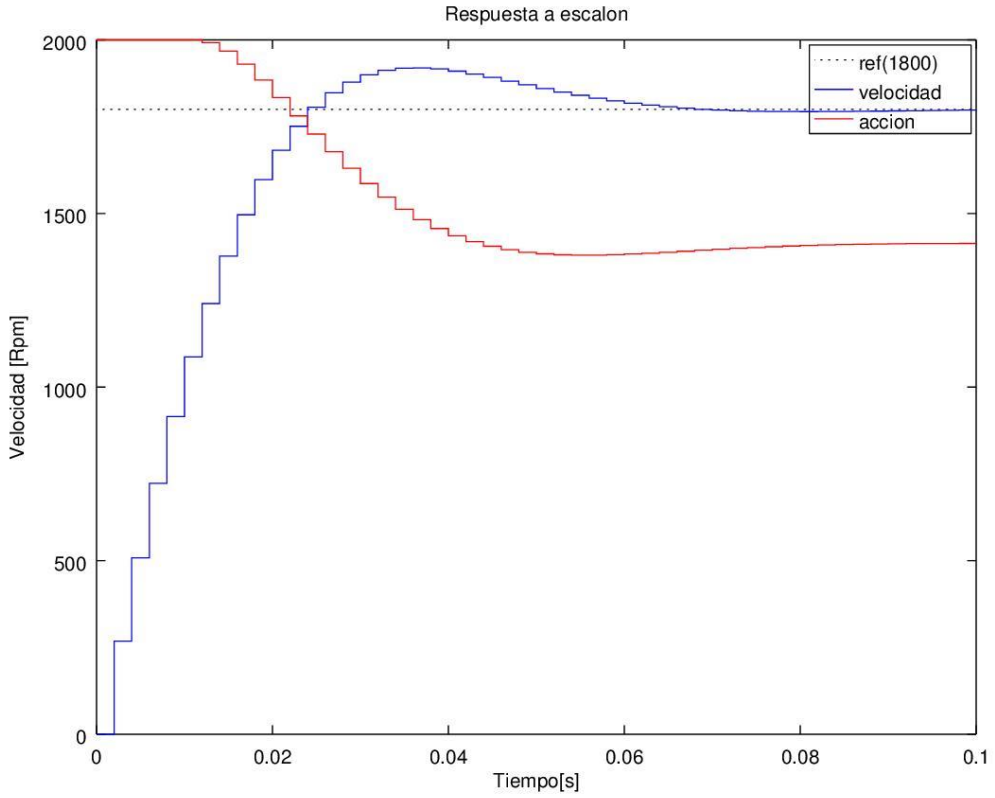


Figura 4.3 Respuesta del sistema controlado a escalón

Como se puede observar la acción satura rápidamente ya que la integral del error se hace cada vez mayor, es por ello que a modo de evitar el efecto de windup, como uno de los métodos adecuados se puede implementar un prefiltrado para la referencia que permita al sistema alcanzar la respuesta sin necesidad de saturar el sistema.

Para esto se ha elegido realizar un filtrado digital de respuesta impulsional finita (FIR), figura 4.4 en cuatro pasos, y cuyos valores se actualizan cada 3 periodos de muestreo dando como resultado:

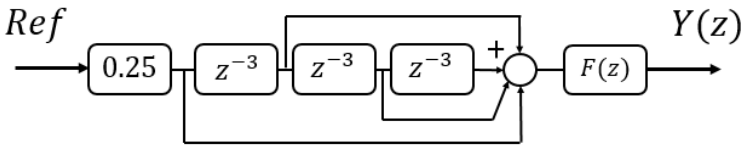


Figura 4.4 Filtro de respuesta impulsional finita (FIR)

Un perfecto ajuste entre nº de pasos y periodos de retardo supondría una correcta mejora en el filtrado de la señal referencia y por ende en la calidad de la respuesta, sin embargo redundaría negativamente en el coste computacional para el micro controlador.

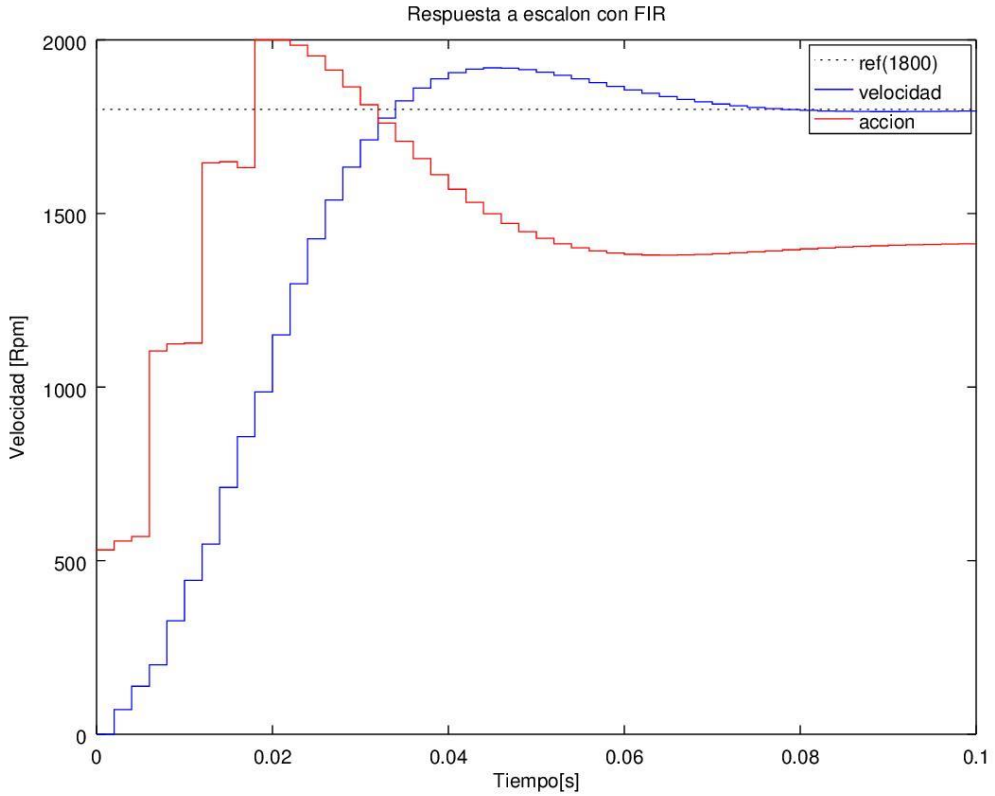


Figura 4.5 Respuesta del sistema controlado con prefiltrado de referencia

De esta forma, como se puede observar en la figura 4.5, apenas satura la acción, aunque si que se puede comprobar que la dinámica de nuestro sistema es ligeramente más lenta. Aumentando el número de periodos de muestreo en los que se mantiene la referencia se conseguiría una acción menor por parte del motor pero a su vez el tiempo de respuesta aumentaría, de esta forma considero adecuado este ajuste.

A modo de entender mejor el controlador calculado y de analizar sus prestaciones se exponen las funciones de sensibilidad [21]. La figura 4.6 muestra la función de sensibilidad de control de ecuación (7), que representa la evolución de la acción para una entrada escalón.

$$Su(s) = \frac{C(s)}{(1 + G(s)C(s))} \tag{7}$$

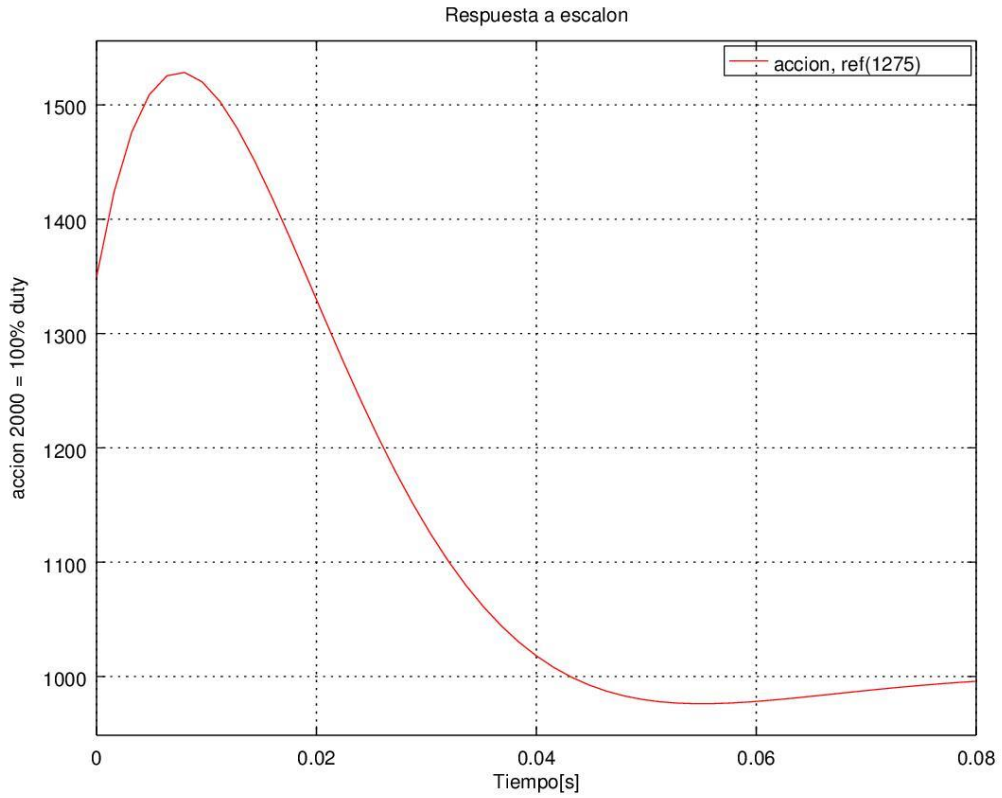


Figura 4.6 Gráfica de esfuerzo del controlador a entrada escalón

Como se observa en la figura 4.6 habiendo introducido una entrada escalón de 1275 Rpm que corresponde a la mitad de la velocidad máxima posible a alcanzar por el motor, aumentará la acción de control hasta 1.53 sobre la acción en la región de estabilidad, la cual se estabiliza a partir de los 40 ms (aprox). Siendo así, se puede calcular que la acción saturaría con una entrada escalón superior a las 1660 Rpm aproximadamente.

La figura 4.7 muestra la función de sensibilidad dada por la ecuación (8), que representa el rechazo del sistema realimentado a las perturbaciones de la figura 4.8 mediante su respuesta en frecuencia.

$$S(s) = \frac{1}{(1 + G(s)C(s))} \tag{8}$$

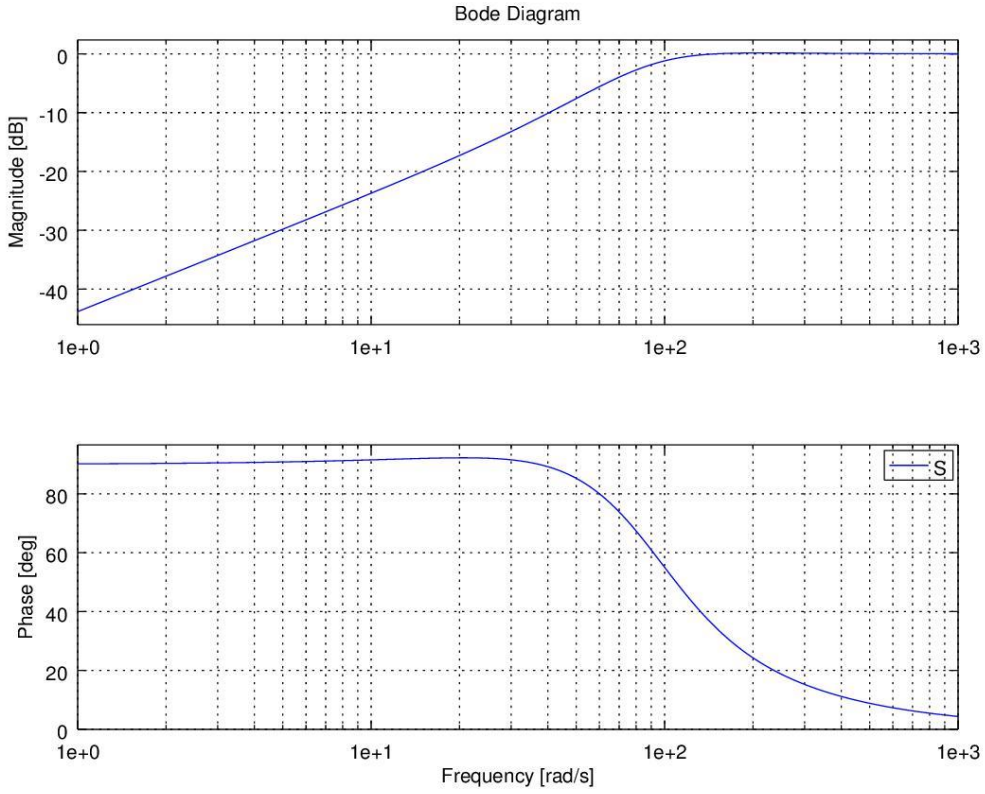


Figura 4.7 Diagrama de bode de la función de sensibilidad

Este sistema responde al esquema de la figura 4.8. Para frecuencias bajas hasta los 100 rad/s se atenúan las perturbaciones llegándose a anular estas en régimen permanente (w= 0). Las perturbaciones sin embargo no serán atenuadas a partir de wc=100 rad/s lo que convierte al sistema en un filtro paso alto frente a perturbaciones de alta frecuencia.

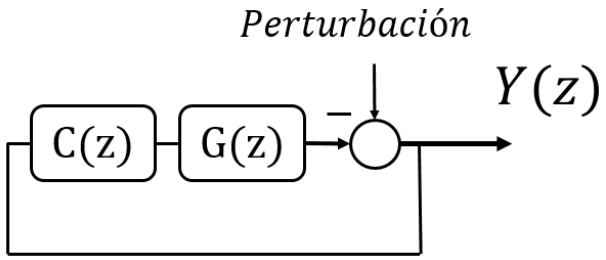


Figura 4.8 Esquema de rechazo de perturbaciones en Bucle cerrado

5.Verificación experimental

En el siguiente capítulo se exponen y explican los resultados obtenidos al implementar el controlador calculado sobre la planta real como se observa en la figura 5.1 el sistema de control calculado en el capítulo previo, así como algunos de los ajustes y ensayos realizados para mejorar las prestaciones de este cálculo.

Figura 5.1 Puesto de trabajo

En primer lugar se explica la diferencia que supone realizar el cálculo para el microcontrolador tanto en coma fija como en coma flotante. Si bien la unidad en coma flotante aportada por la plataforma de Cortex-M4F dota al microcontrolador de una velocidad superior para el cálculo de operaciones en coma flotante, la memoria que ocupa este tipo de datos puede verse reducida si transformo estos mismos a operaciones en coma fija. Redundando así para el controlador calculado previamente en una disminución del memoria flash ocupada de 1000 bytes aproximadamente.

El tiempo de cálculo de la función *velocity*, que abarca tanto el cálculo en coma flotante tanto de la velocidad como del controlador con el prefiltro, se puede ver en la figura 5.2

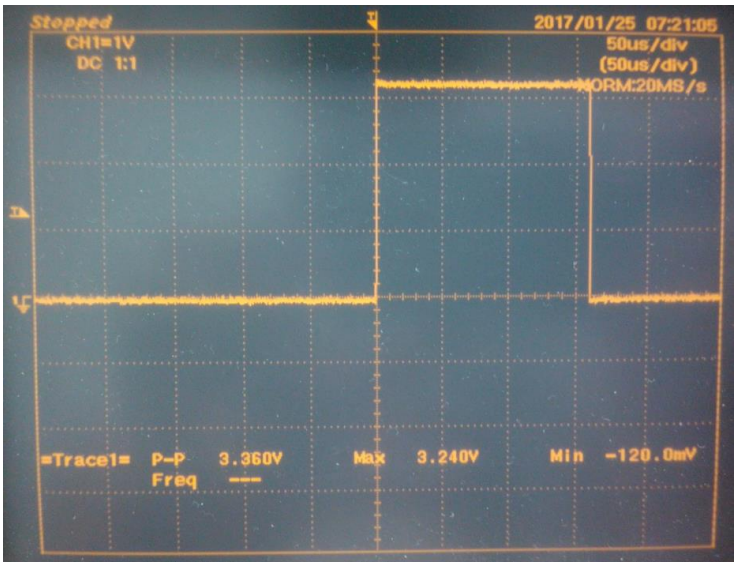


Figura 5.2 Medición del tiempo de la función *velocity* en coma flotante

El tiempo aproximado de cálculo es de 160 µs que corresponde al 8% del tiempo de muestreo que es 2 ms. En cambio haciendo los cálculos en coma fija el tiempo de cálculo se reduce considerablemente hasta los 50 µs como se observa en la figura 5.3

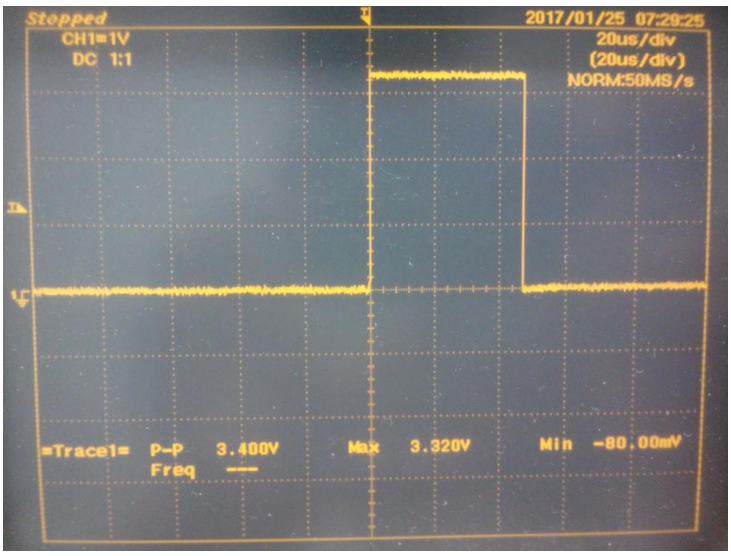


Figura 5.3 Medición del tiempo de la función *velocity* en coma fija

A su vez, se puede comprobar la importancia de la activación de los registros de prioridad para las interrupciones. Pues tal y como está configurado el vector de interrupciones, el tiempo de cálculo de la función de cálculo de velocidad y control si no se ordena la prioridad entre interrupciones, se pierde calidad en el conteo de los pulsos y por tanto en la resolución de la respuesta. En la figura 5.4 se observa como el tiempo requerido para calcular la velocidad y el controlador imposibilita el conteo de algunos pulsos por tanto, al desactivar el controlador, la respuesta es otra ya que de nuevo las interrupciones dadas por los flancos de subida recogidos del encoder son cuantizados. Representándose así la respuesta real del motor.

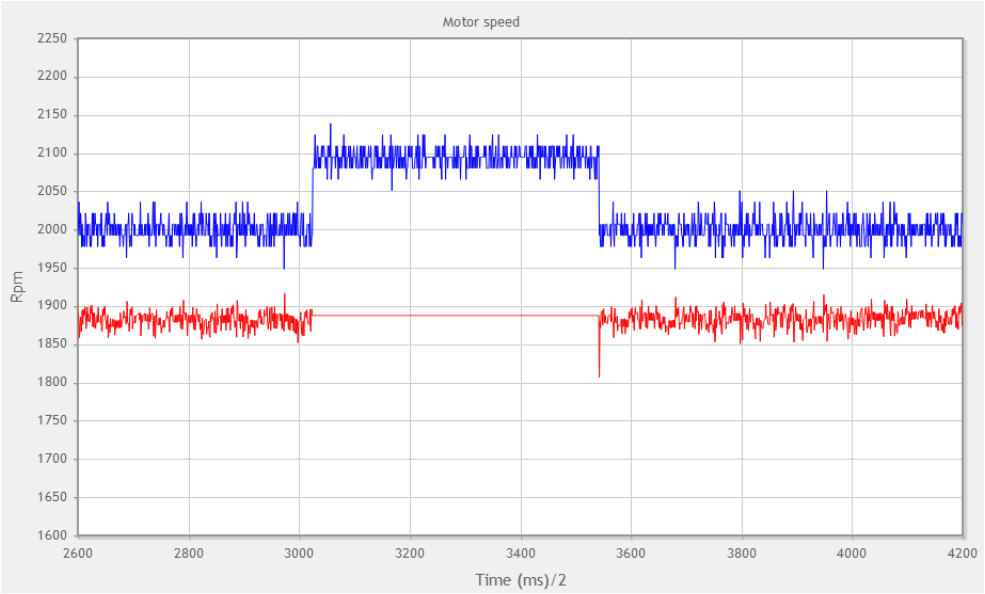


Figura 5.4 Diagrama de respuesta del sistema sin configurar prioridad de interrupciones

Una vez se establece un orden de prioridad entre las interrupciones, si se desactiva el controlador durante la toma de datos en la región de estabilidad, no se observa cambio debido al conteo de pulsos. Además, la acción necesaria para alcanzar la respuesta final es menor puesto que al contar todos los pulsos no hay pérdida de información y por tanto la respuesta final calculada en la figura 5.5 se aproxima más a la respuesta real del motor.

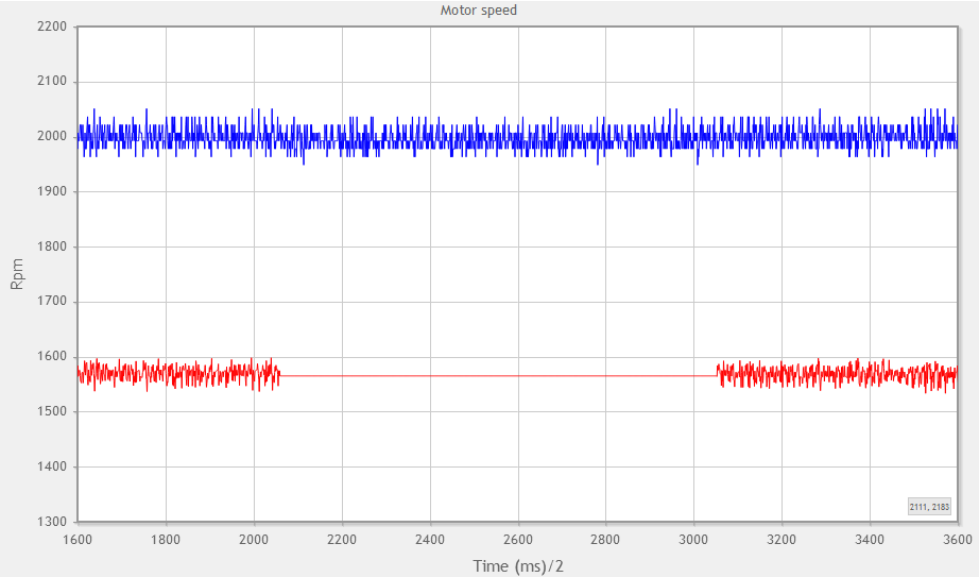


Figura 5.5 Diagrama de respuesta del sistema con prioridad de interrupciones activadas

En la figura 5.6 se observa como sería la respuesta a una entrada escalón de 1800 Rpm sin implementar el prefiltrado de la entrada.

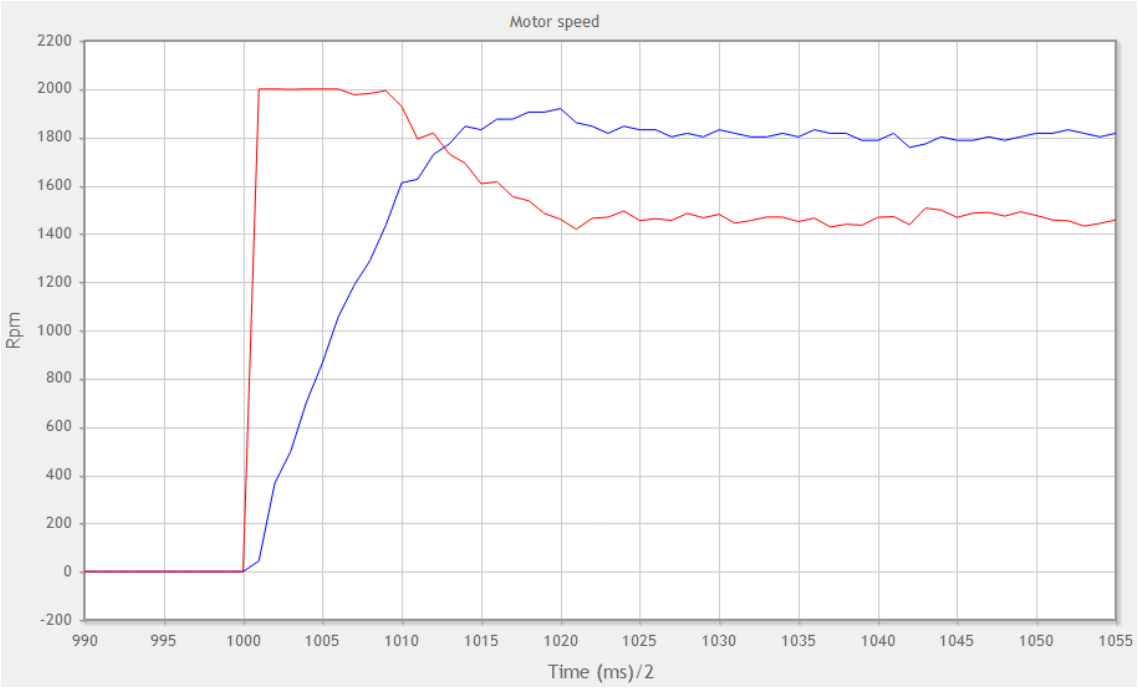


Figura 5.6 Emulación del sistema sin prefiltrado

La acción del controlador se estabilizará aproximadamente tras 0.04 s saturando desde un principio y generando una acción enorme. En la figura 5.7 se observa la respuesta a la misma entrada escalón previa pero con la implementación del filtro FIR que permitirá un ajuste adecuado de la respuesta del sistema evitando prácticamente que sature la acción.

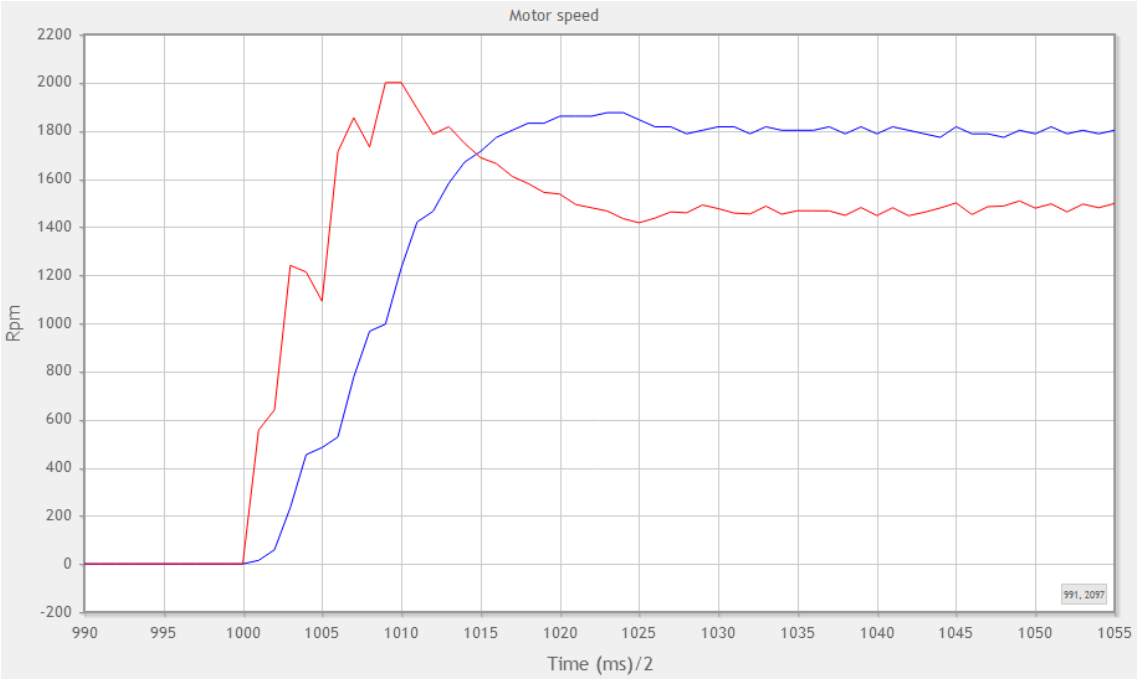


Figura 5.7 Emulación del sistema con prefiltrado

De esta forma se evita el efecto de wind-up puesto que se da tiempo al sistema a seguir la referencia, previniendo este efecto así para grandes cambios de referencia. Sin embargo la sobreoscilación de la respuesta se reduce pero de forma muy leve. Dado que este ajuste no es perfecto, el tiempo de respuesta aumenta ligeramente con respecto al caso anterior.

A continuación en la figura 5.8 se expone el código C referente al controlador calculado. Tras cada tres periodos de muestreo (m) se actualizan las variables que implementan el prefiltro FIR, acto seguido se hallan error y acción del controlador, este último valor se le pasa al registro TA2CCR1 que genera la señal PWM por comparación con el registro TA2CCR0. Por último se actualizan las variables.

```
if ((p)&&(o>(s-1))){

            m=m+1;
            if (m>2){
                        ref4=ref3;
                        ref3=ref2;
                        ref2=ref1;
                        m=0;}
            else{
                        ref4=ref4;
                        ref3=ref3;
                        ref2=ref2;
                        ref1=ref1;
                        m=m;}

            ref=0.25*(ref1+ref2+ref3+ref4);
            ek= ref-y;
            uk = uk_1 +1.18*ek -0.9363*ek_1;

            if (uk>2000)
                  uk= 2000;
            else if (uk<0)
                  uk=0;
            else
                  uk= uk;}
            TA2CCR1 = uk;
            ek_1=ek;
            uk_1=uk;
```

Figura 5.8 Código C del controlador implementado (coma flotante)

La figura 5.9 muestra el mismo controlador en coma fija

```
if ((p)&&(o>(s-1))){
            m=m+1;
            if (m>2){
                                    ref4=ref3;
                                    ref3=ref2;
                                    ref2=ref1;
                                    m=0;}
            else{
                                    ref4=ref4;
                                    ref3=ref3;
                                    ref2=ref2;
                                    ref1=ref1;
                                    m=m;}
            ref1_1=ref1>>2;
            ref2_1=ref2>>2;
            ref3_1=ref3>>2;
            ref4_1=ref4>>2;

            ref=(ref1_1+ref2_1+ref3_1+ref4_1);
            ek= ref-y;

            ek1_1= (int)(15340*ek_1);
            ek1= (int)(19333*ek);
            uk = (ek1-ek1_1)>>14;
            uk = uk +uk_1;

            if (uk>2000)
                  uk= 2000;
            else if (uk<0)
                  uk= 0;
            else
                  uk= uk;}
            TA2CCR1 = uk;
            ek_1=ek;
            uk_1=uk;
```

Figura 5.9 Código C del controlador implementado (coma fija)

En la figura 5.9 se observa el comportamiento del sistema controlado frente a distintos estímulos.

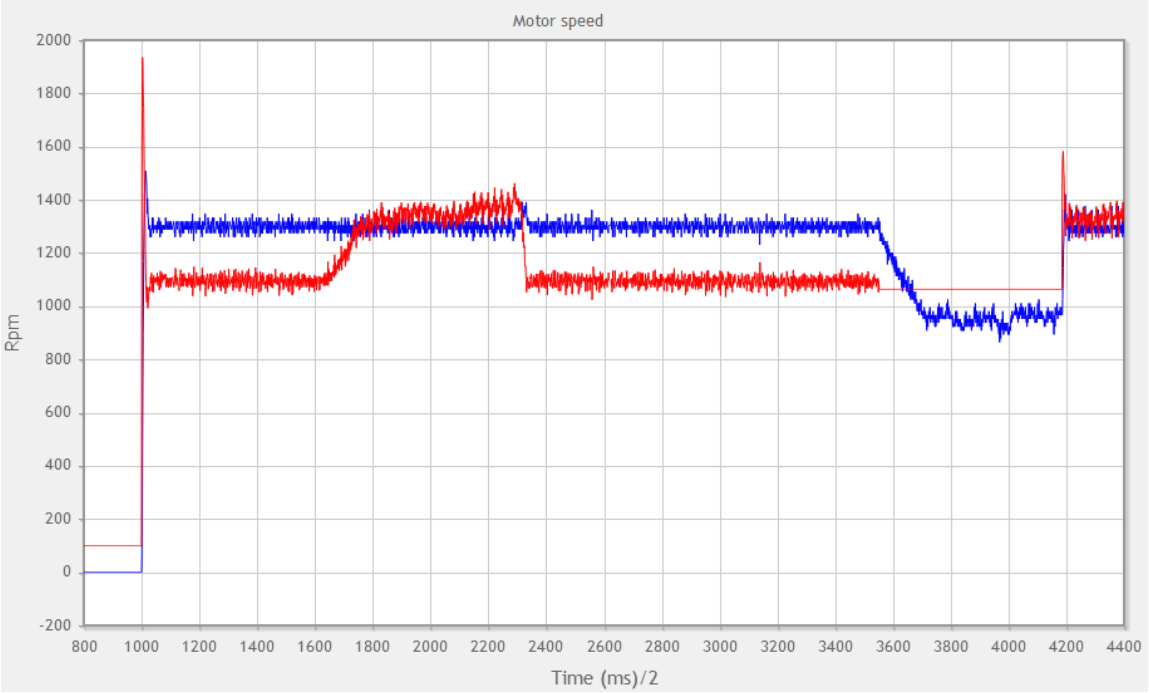


Figura 5.9 Respuesta del sistema a varios estímulos

Se activa el controlador a partir del dato 1000 para una referencia de 1300 Rpm. Después del dato 1600 se aplica una resistencia al giro del motor y la acción aumenta para compensar esta perturbación, no siendo prácticamente apreciable la disminución de velocidad. Tras un tiempo se deja de aplicar resistencia y se observa como brevemente aumenta la velocidad mientras la acción disminuye rápidamente. Esto es así porque la dinámica del controlador no es lo suficientemente rápida para eliminar esta perturbación de forma inmediata. A partir del dato 3500 se desactiva el controlador y se ejerce de nuevo una resistencia en contra del giro del motor, tal y como se observa el motor se ralentiza y cuando llega a la zona de equilibrio no llega a estabilizarse del todo puesto que la fuerza de resistencia que ejerzo (con la mano) no es igual en el tiempo. Por último se activa el controlador de nuevo sin dejar de aplicar resistencia y el motor alcanza la referencia contrarrestando la fuerza de resistencia.

6. Conclusiones y trabajo futuro

6.1 Conclusiones

Como se ha visto a lo largo de este trabajo tras hacer uso del dispositivo MSP432 se puede concluir que la variedad de configuraciones disponibles así como sus prestaciones de cálculo aumentadas gracias a la tecnología del ARM dotan al MSP432 de unas características excelentes para diversos campos de aplicación. Debido a que guarda grandes similitudes con el MSP430 la portabilidad del código se hace especialmente fácil de una plataforma a otra.

Como cambios de relativa importancia, mencionar la nueva estructura del NVIC para procesado y ordenación de las interrupciones, la nueva configuración del reloj DCO con una máximo de frecuencia de 48 Mhz y la posibilidad de ajustar la frecuencia de reloj dentro de los rangos de frecuencias calibradas. El procesador ARM con la FPU que dan al microcontrolador libertad para trabajar a mayor velocidad. En definitiva se trata de una herramienta excelente que cubre los requerimientos que a este trabajo obligan en cuanto a resolución y velocidad de cálculo que a las ecuaciones de velocidad y control atañen.

La GUI del CCS que permite libre modificación del código, y de esta forma libertad a la hora de programar cualquier tipo de aplicación, nos da un sinfín de aplicaciones prácticas con las que aprovechar sus posibilidades. Una de ellas es la identificación del sistema, que resulta de fácil reconocimiento y ajuste gracias a que la GUI proporciona las herramientas para procesar y visualizar los datos tomados por el microcontrolador.

En lo referente al software libre de Octave, este aporta herramientas que son acordes a las necesidades en cuanto al cálculo y simulación de los sistemas de control de este trabajo, resultando sin embargo, por falta de *toolboxes*, un poco más tedioso que su homólogo de pago a la hora de modificar parámetros de simulación pero con los recursos suficientes para simular el sistema de control de este sistema. Su gran atractivo es que es gratuito.

6.2 Líneas futuras

Como línea de trabajo futuro se podría considerar la sustitución del entrenador, utilizado en este trabajo, por un circuito impreso (PCB) con implementación del Hardware, que atenuaría considerablemente los ruidos producidos por los componentes electrónicos. Esto se traduciría en un afine de los resultados experimentales que permitiría aprovechar los recursos de que dispone el MSP432 y así optimizar el sistema de control. Otro de los posibles avances sería un mayor ajuste y estudio del modelo matemático referente a la identificación de la planta. Además se podría implementar un puente en H que permitiese habilitar el giro del motor en ambas direcciones, con lo que esto conllevaría en cuanto a la programación de interrupciones. Por otro lado, se podría perfeccionar el prefiltrado de tal forma que se ajustase y pudiera generar trayectorias que mejoraran la respuesta del sistema de control.

Bibliografía

[1] R. Petrella, M. Tursini, L. Peretti, and M. Zigliotto, "*Speed Measurement Algorithms for Low-Resolution Incremental Encoder Equipped Drives: a Comparative Analysis",* in IEEE Aegean Conference on Electrical Machines and Power Electronics (ACEMP), 2007, pp.780-787

[2] K. Ogata. (2010). *Analisis de la respuesta transitoria y estacionaria*. En Ingeniería de control moderna (7ª edición). Madrid pp. 161-162. Prentice Hall.

[3] Ti.com. (2017). *MSP432P401M Low-Power + High Performance 32-Bit ARM Cortex M4F With 128KB Flash and 32KB RAM | TI.com*. [online] Available at: http://www.ti.com/product/MSP432P401M.

[4] Microcontroladores. (2016). *Introducción y Arquitectura de microcontroladores*. [online] Available at: https://microcontroladoressesv.wordpress.com/arquitectura-de-los-microcontroladores.

[5] Developer.mbed.org. (2016).*C Data Types - Handbook | mbed*. [online] Available at: https://developer.mbed.org/handbook/C-Data-Types.

[6] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 44-48.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[7] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 78-79.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[8] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 304.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[9] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 310-311.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[10] E2e.ti.com. (marzo 2015). *MSP432 FAQ - MSP Low-Power Microcontroller Forum - MSP Low-Power Microcontrollers - TI E2E Community*. [online] Available at: https://e2e.ti.com/support/microcontrollers/msp430/f/166/t/411030

[11] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 318-322.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[12] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 605-609.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[13] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 610-612.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[14] Infocenter.arm.com. (diciembre 2010). *ARM Information Center- Cortex™-M4 Devices Generic User Guide*. *pp. 2.3.1-2.3.3*. [online] Available at: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/index.html.

[15] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 72-73.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[16] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 111-118.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[17] Texas Instruments. (marzo 2015 - diciembre 2016). *Technical Reference Manual SLAU356E. pp. 498-500.* http://www.ti.com/product/MSP432P401R/technicaldocuments

[18] Processors.wiki.ti.com. (septiembre 2017). *Category:GUI Composer - Texas Instruments Wiki*. http://processors.wiki.ti.com/index.php/Category:GUI_Composer.

[19] Processors.wiki.ti.com. (Abril 2014). *GUI Composer/Logging Graph Data-Texas InstrumentsWiki*.
http://processors.wiki.ti.com/index.php/GUI_Composer/Logging_Graph_Data

[20] M. Lohöfener. *"Model-Based Design of Mechatronic Systems with Open Source Software"*. Hochschule Merseburg, University of Applied Sciences. Merseburg, Germany.

[21] I. Díaz Blanco. (2003). *Funciones de sensibilidad.* Departamento de Ingeniería Eléctrica, Electrónica, de Computadores y Sistemas. Universidad de Oviedo.

[22] D. Dang. (marzo 2015 – diciembre 2016). *MSP432™ Platform Porting Guide*. SLAA656B Texas Instruments. *pp 2-12.*
http://www.ti.com/product/MSP432P401R/technicaldocuments

[23] D. Dang. A. Lele. (marzo 2015). *Designing an Ultra-Low-Power (ULP) Application With MSP432™ Microcontrollers*. SLAA668 Texas Instruments. *pp 2-5.*
http://www.ti.com/product/MSP432P401R/technicaldocuments

[24] Texas Instruments. (marzo 2015 - diciembre 2016). *Code Composer Studio™ 6.1+ for MSP432 SLAU575E. pp. 16-21.*
http://www.ti.com/product/MSP432P401R/technicaldocuments

[25] A. Barrado, R. Vázquez, A. Lázaro, J. Pleite, E. Olías. "*Fast Transient Response with Combined Linear-Non-Linear Control Applied to Buck Converters"*. Universidad Carlos III de Madrid. *pp 1587-1592.*