DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR DIGITAL MEDIANTE TÉCNICAS DE PROTOTIPADO RÁPIDO PARA APLICACIONES DE ELECTRÓNICA DE POTENCIA.

ALIRIO ERNESTO ARIZA MENDEZ.

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA
FACULTAD DE INGENIERIA ELECTRONICA
BUCARAMANGA
2013

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR DIGITAL MEDIANTE TÉCNICAS DE PROTOTIPADO RÁPIDO PARA APLICACIONES DE ELECTRÓNICA DE POTENCIA.

INVESTIGADOR:
ALIRIO ERNESTO ARIZA MENDEZ.

DIRECTOR DEL PROYECTO:
PhD. OMAR PINZÓN ARDILA.

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERÍA
FACULTAD DE INGENIERIA ELECTRONICA
BUCARAMANGA
2013

Nota de aceptación.

_____
_____
_____
_____
_____

_____
Presidente del Jurado

_____
Firma del Jurado

_____
Firma del Jurado

Bucaramanga, Marzo 14 de 2013.

AGRADECIMIENTOS

A mi director de tesis, el Doctor Omar Pinzón, por su apoyo, paciencia y dedicación para ayudarme en la realización del proyecto.

A Dios por regalarme cada día la oportunidad de ser una mejor persona, por las personas que ha puesto en mi vida y por permitir que este trabajo sea posible.

A mis familiares y seres queridos por acompañarme y brindarme todo su apoyo en los momentos más difíciles.

A mi madre que me ha formado como persona de bien y siempre lo ha dado todo con el amor más grande para verme triunfar como profesional.

A mis maestros que aportaron todo su conocimiento en mi proceso de formación académica.

CONTENIDO

LISTA DE FIGURAS.

LISTA DE TABLAS

RESUMEN GENERAL DEL TRABAJO

TITULO: **DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR DIGITAL MEDIANTE TÉCNICAS DE PROTOTIPADO RÁPIDO PARA APLICACIONES DE ELECTRÓNICA DE POTENCIA.**

AUTOR: **ALIRIO ERNESTO ARIZA MÉNDEZ.**

DIRECTOR: **OMAR PINZÓN ARDILA.**

RESUMEN

El presente documento muestra el uso de herramientas de prototipado rápido de Simulink para la implementación de un controlador digital sobre un procesador de señales digitales TMS320F28335 de Texas Instruments. Se utilizaron las plataformas *Embedded Coder* y *Simulink coder* para conectar a Simulink con el entorno de desarrollo *Code Composer Studio* y generar de forma automática el código de control. Desde Embedded Coder se realizaron las tareas de compilación, depuración, descarga y ejecución del código sobre el DSP. La verificación y validación del código generado se realizó mediante el intercambio de datos en tiempo real RTDX de Texas Instruments, transfiriendo información sobre el desempeño del sistema desde el DSP hacia un computador en tiempo real y sin interferir con la ejecución de la aplicación. La planta controlada es un motor DC de 12 volts. Se construyó una interfaz de usuario gráfica con la herramienta GUIDE de Matlab para la visualización de las señales del sistema y la manipulación de la señal de referencia.

Palabras clave: Sistemas de prototipado rápido, sistemas embebidos, Simulink Embedded Coder, Micro-controlador TI C2000, Code Composer Studio, Controlador digital.

GENERAL SUMMARY OF WORK OF GRADE

TITLE: **DESIGN AND IMPLEMENTATION OF A DIGITAL CONTROLLER USING RAPID PROTOTIPING TECHNIQUES FOR POWER ELECTRONICS APPLICATIONS.**

AUTHOR: **Alirio Ernesto Ariza Méndez.**

FACULTY: **Electronic Engineering.**

DIRECTOR: **Omar Pinzón Ardila.**

ABSTRACT

This document shows the use of the Simulink rapid prototyping tool to implement a digital controller on a TMS320F28335 Digital Signal Processor of Texas Instruments. Embedded Coder and Simulink Coder Platforms were used to connect with the Integrated Development Environment Code Composer Studio and automatically generate he source code. The tasks of compiling, debugging, downloading and running the code on the Digital Signal Processor were done directly from Embedded Coder. The verification and validation of generated code was made by the exchanging data in real time RTDX tool of Texas Instruments, transferring information about the system performance from the DSP to a computer in real time without interfering with the execution of the application. The controlled plant is a 12 volts DC motor. A graphical user interface was built with the MATLAB tool GUIDE to display the system signals and manipulate the reference signal.

Keywords: Rapid prototyping, embedded systems, EmbeddedCoder, Code Composer Studio, Digital Controller.

INTRODUCCIÓN

El crecimiento acelerado de los procesos tecnológicos, industriales y de fabricación ha sido influenciado por el desarrollo de la ingeniería en campos como el control automático, permitiendo conseguir un comportamiento óptimo de los sistemas dinámicos, mejorar la productividad y simplificar el trabajo de muchas operaciones manuales, repetitivas y rutinarias.[1] En los sistemas modernos de control automático, las estrategias de control son comúnmente implementadas utilizando software, dando lugar al desarrollo de los sistemas embebidos.

La implementación final de un sistema de control embebido requiere de una etapa previa de pruebas para determinar si los diseños cumplen con los requerimientos de control. Las técnicas de prototipado rápido se han convertido en una alternativa viable y eficiente para el desarrollo de sistemas de control debido a que posibilitan evaluar sistemas y algoritmos de procesamiento de señales sobre hardware en tiempo real, permitiendo ajustar los sistemas en desarrollo hasta obtener los resultados esperados.[2] Por otro lado, estas técnicas permiten construir prototipos directamente desde los modelos conceptuales o de diagramas de bloques de sistemas de control, reduciendo de esta forma el tiempo de implementación de los controladores, especialmente si los algoritmos de control son complejos y requieren realizar muchas tareas e iteraciones.

Con el propósito de reafirmar la validez de las técnicas de prototipado rápido como alternativas simples y efectivas al control automático de procesos, en el presente trabajo de grado se hace uso de estas técnicas para el diseño y la implementación de un controlador digital. A partir de modelos de simulación construidos en la plataforma software Simulink, se utilizó la herramienta *Embedded Coder* y el entorno de desarrollo Code Composer Studio de Texas Instruments para generar e implementar código sobre un controlador digital y para comunicar dicho dispositivo con un computador permitiendo evaluar su desempeño.

Los objetivos propuestos de la investigación están orientados a brindar opciones prácticas para el desarrollo de controladores que normalmente requieren de algoritmos bastante complejos y por tanto, implican gastos considerables de tiempo y dinero.

---

[1] OGATA Katsuhiko. Ingeniería de control moderna. Cuarta edición. Pearson Educación S.A. Madrid 2003. Pág 1.

[2] MATHWORKS. Rapid Prototyping. [En linea] http://www.mathworks.com/rapid-prototyping/ [citado en 2012].

1. OBJETIVOS

1.1. Objetivo general.

- Implementar un controlador digital utilizando un DSP de 32 bits de la empresa *Texas Instruments* mediante técnicas y herramientas de prototipado rápido que proporcionan las plataformas *Matlab* y *Simulink* para aplicaciones de potencia.

1.2. Objetivos específicos.

- Identificar la función de trasferencia del modelo de la planta (motor DC) utilizando técnicas de prototipado rápido.
- Diseñar y simular una estrategia de control (PID avanzado) para la planta seleccionada. Generar el código en lenguaje C para el controlador propuesto e implementarlo en un DSP C28x
- Construir un circuito electrónico para acondicionamiento de señales de las etapas de control y potencia.
- Comparar los resultados experimentales y de simulación
- Diseñar e implementar una interfaz gráfica de usuario para monitorear y sintonizar el controlador mediante la herramienta GUIDE de Matlab.

2. MARCO TEÓRICO

2.1. SISTEMAS EMBEBIDOS

La palabra embebido, está directamente ligada a los sistemas de computación. Un sistema embebido se conoce como un circuito electrónico diseñado para cumplir una labor específica en un producto. [3] Se diferencian de un computador convencional porque un sistema embebido soluciona un problema específico y normalmente no necesitan la intervención constante del usuario para su funcionamiento. En un sistema embebido las estrategias de control son comúnmente implementadas sobre software.

Un sistema embebido es entonces, una combinación software y hardware en un equipo electrónico y/o mecánico. Los hornos micro-ondas son un ejemplo donde un computador (procesador) y un software están involucrados en el calentamiento de alimentos. Los sistemas embebidos se encuentran en casi cualquier dispositivo electrónico en uso actualmente. Es frecuente encontrar sistemas embebidos en conjunto con otros sistemas.[4] Por ejemplo los automóviles modernos contienen un conjunto de sistemas embebidos donde cada uno se ocupa de una tarea específica.

Los sistemas de control embebido los componen principalmente la planta, un controlador, actuadores que operan la planta de acuerdo a las señales del controlador, y los sensores. En un sistema embebido las señales de baja potencia del controlador deben interactuar con las de alta potencia de la planta.

En el sistema de control embebido implementado en el presente trabajo, se generó el software de control para un elemento hardware de procesamiento de señales digitales con el fin de manipular la variable de velocidad de un motor eléctrico.

[3] GALEANO Gustavo. Programación de sistemas embebidos en C. Editorial Alfaomega Colombiana S.A. 2009.

[4] BARR Michael, MASSA Anthony. Programming Embedded Systems. Editorial O'Reilly. Segunda edición. EEUU 2006.

2.2. PROTOTIPADO RÁPIDO

El prototipado rápido es una técnica para la elaboración de sistemas de control de prueba que permitan validar los algoritmos diseñados ejecutándolos ya sea de forma directa en la planta, en un modelo de simulación o en un sistema que combina partes de la planta y un modelo de ella.[5] El propósito del prototipado rápido es obtener una idea confiable del desempeño de un controlador diseñado antes de llevarlo a implementación final sobre un sistema físico.

En el diseño de sistemas de control, cuando se obtiene el modelo de un controlador, éste no necesariamente se lleva a una implementación definitiva debido a que podría requerir de modificaciones debido a detalles físicos como las características del dispositivo procesador que ejecutará las acciones de control. Para disminuir costos de implementación, la creación de prototipos de prueba resulta bastante útil.

Los productos de prototipado rápido de MathWorks automatizan el trabajo de desarrollo de modelos y algoritmos de control sobre hardware para ejecutarlos en tiempo real, permitiendo centrar la atención a la evaluación del desempeño de los modelos y la mejora de los mismos en lugar de dedicar esfuerzos y tiempo a escribir y modificar código de programación para los dispositivos microprocesadores.

[5] MOSTERMAN Pieter, PRABHU Sameer, ERKKINEN Tom. An industrial embedded control system design process. Montreal 2004.

2.3. PROCESADOR DIGITAL DE SEÑALES TMS320F28335

Figura 2.1 DSP TMS320F28335.

Fuente: Internet: http://www.ti.com/product/tms320f28335

El avance tecnológico alcanzado en la construcción de circuitos integrados digitales, ha contribuido en el desarrollo de poderosos dispositivos, utilizados en la implementación de sistemas para el procesamiento de señales.[6]

Un sistema de procesamiento digital de señal aplica operaciones matemáticas a señales físicas representadas de forma digital mediante secuencias de muestras. Para realizar el procesado, las señales físicas se capturan mediante transductores y se digitalizan mediante convertidores analógico-digital.[7]

En sistemas donde el procesamiento se hace complejo, se utilizan procesadores digitales de señales (DSP por sus siglas en inglés). Los DSP son microprocesadores diseñados con arquitecturas especiales para acelerar los cálculos matemáticos intensos necesarios en sistemas embebidos de tiempo real.

Un microprocesador cuenta con dos unidades básicas para manipular señales provenientes del mundo físico, leer operaciones de una memoria de datos, escribir datos de nuevo en la memoria y actualizar los módulos de salida. Dichas unidades son la unidad de control y la unidad de procesamiento central o CPU.

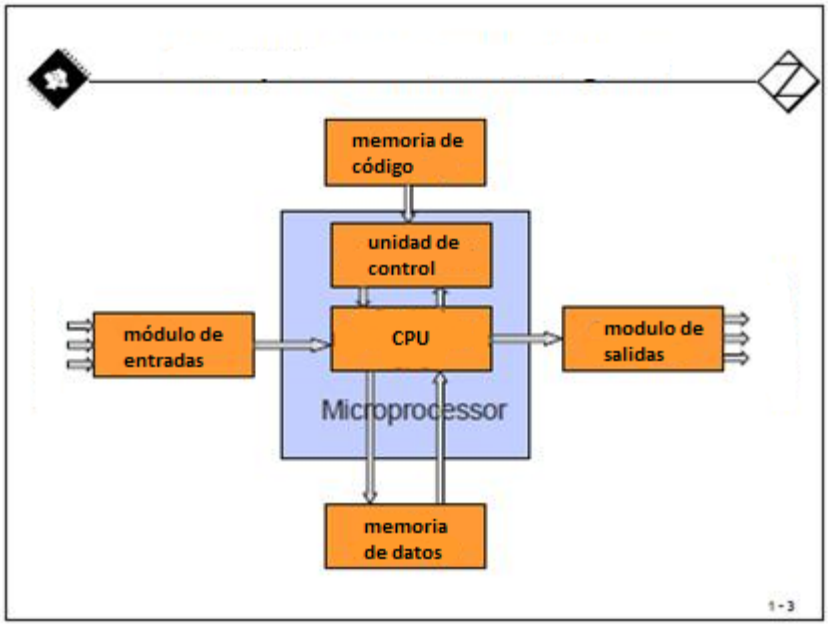
---

[6] DINIZ Paulo, DA SILVA Eduardo, NETTO Sergio. Digital signal processing. System Analysis and Design. 2 ed. Cambridge University. 2010. p. 2.

[7] SALAZAR Jordi. Procesadores Digitales de Señal. Universidad Politécnica de Cataluña. España. [en línea] http://arantxa.ii.uam.es/~taao1/teoria/tema1/pdf/Procesadores_dig.pdf. [citado en 2012].

Las actividades que realiza el microprocesador son cíclicas y se encuentran almacenadas en una memoria de código que se cargan cada vez que se cumple un ciclo de máquina completo.[8] La figura 2.2 muestra las partes principales de un DSP.

Figura 2.2 Partes Funcionales de un procesador.



Fuente: Internet: http://cnx.org/content/m36699/latest/

Uno de los principales fabricantes de DSP es la compañía *Texas Instruments* con su serie TMS320 que abarca las familias C2000, C5000 y C6000. La familia C2000 es la más económica, y se caracteriza por su núcleo optimizado para ejecutar múltiples algoritmos complejos y por sus periféricos de alto rendimiento, propicios para aplicaciones de control en tiempo real. Las familias C5000 y C6000 por su parte, son de más alta jerarquía, con velocidades de trabajo del orden de 1.2 GHz y bajo consumo de energía son ideales para aplicaciones de procesamiento de audio, voz y video, así como en comunicaciones inalámbricas. Para el desarrollo del trabajo expuesto en este texto, se utilizó el DSP TMS320F28335, cuyas características más importante se explican a continuación.

[8] BORMANN Frank, Introduction to TMS320F28335. Texas Instruments Inc. 2011.

2.3.1 Descripción Funcional del DSP TMS320F28335

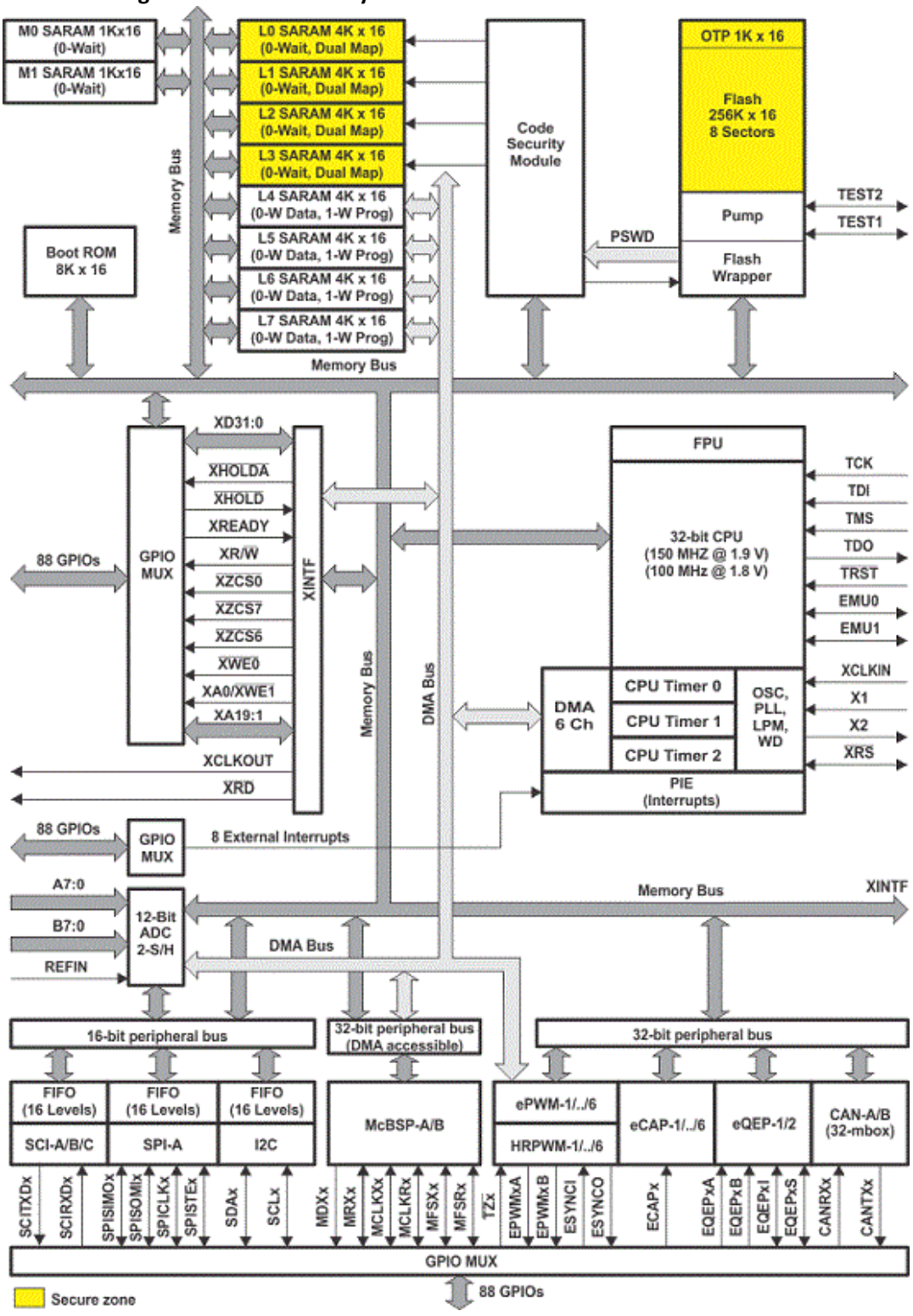
El DSP TMS320F28335 hace parte de la gama de procesadores embebidos de 32 bits C28x / Delfino de la familia C2000 de *Texas Instruments* creados para tener alto rendimiento en aplicaciones de control de tiempo real.

A continuación se enumeran las principales características del dispositivo, entre las cuales se destacan su CPU de 32 bits con arquitectura Harvard y aritmética de coma flotante, la velocidad de trabajo de 150 MHz y sus periféricos de comunicaciones.

- Ciclo de trabajo de 6.67 ns (150 MHz).
- Aritmética de coma flotante.
- 6 canales de acceso a memoria directo DMA.
- 6 módulos de modulación de ancho de pulso PWM con 18 salidas.
- 6 canales de modulación de ancho de pulso de alta resolución HRPWM.
- 6 entradas de modo de captura mejorada eCAP.
- 2 módulos de codificador de pulsos en cuadratura eQEP.
- Módulo de temporizador de perro guardián.
- 16 canales de conversor analógico-digital ADC de 12 bits y frecuencia de muestreo de 80 KHz y un periodo de conversión de 80 ns.
- Interfaz periférica serial SPI.
- 3 módulos de comunicación serial SCI.
- 2 módulos de comunicaciones eCAN (controller area network).
- 1 módulo de comunicaciones I2C (inter-Integrated Circuit).
- 88 pines de entrada/salida de propósito general.
- 8 Interrupciones externas.
- Memoria Flash embebida de 256K x 16.

La figura 2.3 muestra los bloques funcionales del TMS320F28335 donde se puede observar la distribución y conexión de los elementos anteriormente nombrados dentro del DSP.

Figura 2.3 Distribución y conexiones internas del TMS320F28335.



Fuente: Internet: http://www.ti.com/lit/ds/sprs439m/sprs439m.pdf.

CPU y Buses de Memoria.

La CPU C28x del TMS320F28335 contiene una unidad de punto flotante (FPU) de 32 bits IEEE 754, lo cual permite obtener buen desempeño en los cálculos matemáticos realizados dentro de los algoritmos construidos en lenguaje de alto nivel C/C++. El dispositivo se hace tan eficiente para las tareas matemáticas de un DSP como lo es para las tareas de control de sistemas a las que normalmente se destina un micro-controlador.

En cuanto a arquitectura, los dispositivos C28x, cuentan con buses para lectura de datos, buses para escritura de datos y buses para lectura de programa que comunican periféricos y memorias con la CPU[9]. Debido a que estos buses cuentan con líneas de 32 datos, se posibilita la ejecución de operaciones de 32 bits en un mismo ciclo de máquina. Del mismo modo, la arquitectura de múltiples buses, permite recibir una instrucción, leer datos y escribirlos dentro de un ciclo de máquina.

Adicionalmente, la CPU del F28335 cuenta con 3 temporizadores (*timers*) de 32 bits llamados *CPU-Timer 0, CPU-Timer 1 y CPU-Timer 2*, junto con los temporizadores presentes en los módulos PWM. El *Timer 2* está reservado para el DSP mientras que los otros dos pueden ser utilizados por el usuario.

Interfaz JTAG.

El TMS320F28335 cuenta con interfaz JTAG IEEE 1149.1 estándar. Dicha interfaz está definida por una circuitería construida dentro de un circuito integrado para la depuración de código y para realizar pruebas del circuito.[10] Adicionalmente, el dispositivo soporta el modo de operación "real time" con el cual se puede modificar el contenido de la memoria y de locaciones de registros y periféricos mientras el procesador se está ejecutando.

Mapa de Registros.

El TMS320F28335 contiene cuatro espacios de registros de diferentes periféricos. La primera localidad contiene registros de los módulos ADC, registros de temporizadores de la CPU, registros flash, y registros de algunas interrupciones externas. La segunda localidad abarca registros de algunos puertos PWM, CAN, CAP y pines de propósito general. Dentro del siguiente espacio se encuentran

[9] TEXAS INSTRUMENTS. TMS320F28x Digital Signal Controller – Manual de Datos. 2007. [Disponible en linea] http://www.ti.com/lit/ds/sprs439m/sprs439m.pdf.

[10] IEEE STANDARDS ASSOCIATION. 1149.1-2001 IEEE Standard Test Acces Port and Boundary Scan Architecture. [en línea] http://standards.ieee.org/findstds/standard/1149.1-2001.html [citado en 2012].

registros de algunas interrupciones externas y los registros de los módulos de comunicaciones SCI, SPI, I2C, y los de algunos módulos ADC. La cuarta localidad contiene registros de los módulos PWM de alta resolución.

2.3.2 Periféricos

Como se explicó, el TMS320F28335 cuenta con múltiples periféricos útiles para aplicaciones de control de sistemas de tiempo real; a continuación se presenta una explicación detallada de los dos periféricos utilizados para el desarrollo del trabajo final, el control de velocidad de un motor DC.

2.3.2.1. Módulos PWM mejorado.

El periférico de modulador de ancho de pulso mejorado ePWM es un elemento clave en el control de sistemas electrónicos comerciales e industriales, como motores, fuentes conmutadas, fuentes de energía ininterrumpida UPS, entre otros. Un periférico ePWM puede desempeñar el papel de un convertidor Digital-Analógico DAC, donde el ciclo de trabajo es equivalente al valor analógico de la señal.[11] El DSP F28335 cuenta con 6 módulos PWM mejorado (ePWM1, ePWM2, ePWM3, ePWM4, ePWM5, ePWM6) con contadores de base de tiempo de 16 bits. Cada módulo cuenta con dos canales de salida (A y B).

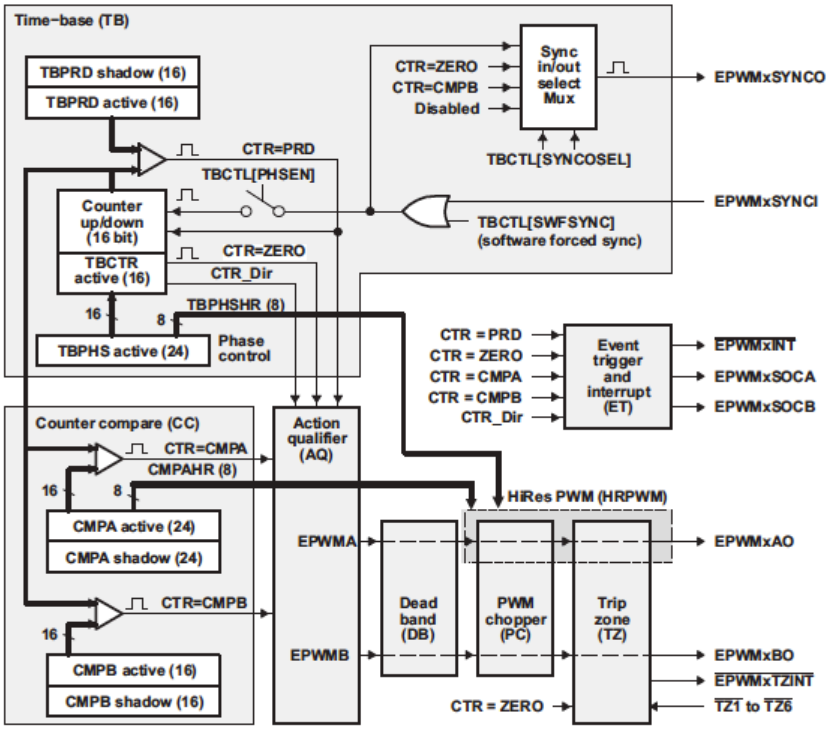
Los módulos de PWM mejorado están conformados por siete bloques principales conectados como lo muestra la figura 2.4. El primero de ellos, el sub-módulo de tiempo base determina la sincronización de todos los eventos propios de cada módulo. Los registros de este sub-módulo, permiten realizar algunas funciones como definir la frecuencia de la señal PWM y sincronizar los contadores de tiempo base de los módulos PWM entre sí.

El segundo sub-módulo se encarga de realizar la comparación continua entre el valor del contador de tiempo base y los registros comparador de conteo A y contador de conteo B (CMPA y CMPB respectivamente) para generar un evento específico. De esta forma se puede controlar el ciclo útil de la señal PWM.

Otro sub-módulo muy importante es el calificador de acción. Este bloque controla la conmutación de la señal de salida de acuerdo a condiciones que se configuran vía software a partir de la comparación de registros. De igual forma, se tienen los sub-módulos generador de zona muerta, *chopper* y *trip zone* encargados de los retardos en las conmutaciones de la señal de salida y la reacción ante fallos.

[11] TEXAS INSTRUMENTS. TMS320x2833x Enhanced Pulse Width Modulator (ePWM) Module - Guía de Referencia. 2008. [Disponible en línea] http://www.ti.com/lit/ug/sprug04a/sprug04a.pdf

Figura 2.4 Diagrama de bloques de un módulo ePWM.



Fuente: Internet: http://www.ti.com/lit/ug/sprug04a/sprug04a.pdf

Asimismo, se tiene la posibilidad de disminuir considerablemente el tamaño de paso (mínimo cambio de la posición del flanco entre dos valores consecutivos) del ciclo útil de la señal de salida, utilizando los módulos de alta resolución HRPWM disponibles en el canal A de cada periférico ePWM.

El módulo HRPWM mejora las capacidades de resolución de tiempo de un modulador de ancho de pulso convencional, por lo cual es utilizado especialmente en aplicaciones en las que la resolución de la señal generada cae por debajo de 9 o 10 bits. Lo anterior ocurre cuando la frecuencia de la señal PWM es demasiado alta (por encima de 200 KHz) con un reloj de CPU de 100 MHz.[12] Para el desarrollo del presente trabajo se utilizaron los módulos de PWM convencional debido a que la frecuencia de las señales generadas son inferiores a 20 KHz, es decir, un tamaño de paso menor a 0,02%.

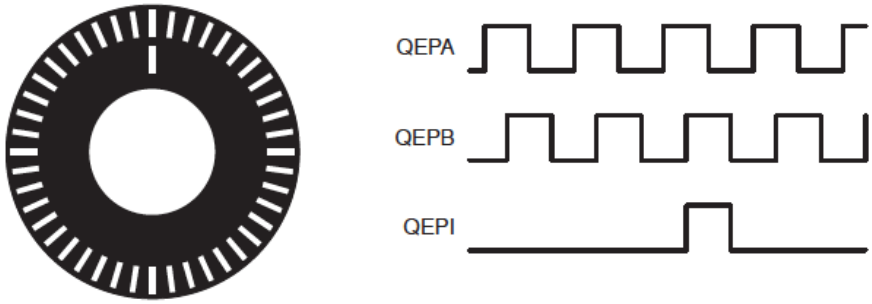
[12] TEXAS INSTRUMENTS. TMS320x2833x High Resolution Pulse Width Modulator (HRPWM). Guía de referencia. 2009. [disponible en línea] http://www.ti.com/lit/ug/sprug02b/sprug02b.pdf

2.3.2.2. Módulos QEP mejorado.

El módulo de codificador de pulsos en cuadratura mejorado eQEP es utilizado para recibir pulsos provenientes de un codificador digital, permitiendo obtener información sobre la velocidad, posición y/o dirección de un sistema mecánico.[13] En aplicaciones de sistemas en movimiento de alto rendimiento y control de posición, los módulos eQEP son de gran utilidad.

Los módulos eQEP del DSP expuesto, cuentan con un hardware externo de cuatro entradas (Canal A, Canal B, índice y entrada estroboscópica). Dichas entradas se conectan al puerto de un codificador convencional, donde dos pulsos (QEPA y QEPB) desfasados 90 grados uno del otro, son generados a partir del movimiento de un disco giratorio con un determinado número de marcas como lo muestra la figura 2.5. La señal QEPB se adelanta o se atrasa a la señal QEPA dependiendo del sentido de giro del disco. Una marca adicional en el disco, hace que se genere un tercer pulso cada revolución (QEPI).

Figura 2.5 Discon de un codificador de pulsos en cuadratura y señales de salida.



Fuente: Internet: http://www.ti.com/lit/ug/sprug05a/sprug05a.pdf.

La entrada estroboscópica se utiliza de forma opcional para inicializar o guardar el conteo de posición, o simplemente para indicar que el motor llegó a una posición determinada.

Cada módulo eQEP cuenta con las siguientes bloques principales que se pueden configurar mediante software dependiendo de la tarea para la cual se va a destinar el módulo.

[13] TEXAS INSTRUMENTS. TMS320x2833x Enhaced Quadrature Encoder Pulse (eQEP) module. Guia de referencia. 2008. [disponible en linea] http://www.ti.com/lit/ug/sprug05a/sprug05a.pdf

- Selección programable de pines para cada entrada. Los canales de ambos módulos eQEP del TMS320F28335 pueden estar disponibles en dos pines diferentes. El usuario puede configurar el pin de cada canal vía software.
- Unidad decodificadora de cuadratura. Esta unidad es la encargada de transferir la señal de pulsos a un contador de posición, y la señal de sentido de giro o dirección.
- Unidad de control y conteo de posición. En esta unidad se configuran los registros y contadores que se utilizan en la medición de posición.
- Unidad de captura de flanco. Cuando se desea hacer una medición de velocidades bajas, es útil configurar esta unidad para medir el tiempo (en unidades pre-escaladas del reloj del sistema) que tarda el cambio de una posición a otra del codificador de cuadratura.
- Unidad de base de tiempo para medición de velocidades medias y altas. El periférico eQEP cuenta con un temporizador de 32 bits sincronizado por el reloj del sistema para generar interrupciones periódicas. En cada interrupción el contador de posición transfiere el valor de la cantidad de pulsos que ocurrieron en dicho periodo y luego se reinicia para realizar el conteo de nuevo.

2.3.2.3. Otros Periféricos.

La aplicación final de la investigación expuesta en el presente texto es el control PID de un motor DC para lo cual se utilizaron los periféricos PWM y QEP del TMS320F28335 explicados anteriormente. Sin embargo, el dispositivo cuenta con otros periféricos como se muestra a continuación.

- Módulos Convertidores Analógico-Digital ADC de 12 bits para medición de voltajes de 0-3 V. Los módulos trabajan a una velocidad de muestreo de 25 MHz (12.5 MSPS).
- 3 temporizadores de 32 bits para propósito general; CPU-Timer 0, CPU-Timer 1 y CPU-Timer 2.
- Módulos de captura mejorada. El dispositivo cuenta con 6 módulos de captura mejorada (eCAP1, eCAP2, eCAP3, eCAP4, eCAP5, and eCAP6) controlados por el reloj del sistema para uso en aplicaciones en las que el cronometraje preciso de eventos externos se hace importante.
- Periféricos para comunicaciones como SPI, SCI, I2C.
- Módulo de red de área de controlador CAN versión 2.0 con una tasa de muestreo de 1 Mbps.

2.3.3 Software de programación.

Las aplicaciones para los dispositivos de cada familia de procesadores y micro-controladores de *Texas Instruments* son desarrolladas y depuradas con la herramienta software *Code Composer Studio (CCS)*. CCS es un entorno de desarrollo integrado (IDE) que dispone de compiladores para cada dispositivo, editor de código fuente, entorno de creación de proyectos, depurador, simuladores, sistema de operación de tiempo real entre otras características. El entorno está basado en el sistema software de código abierto Eclipse (Sistema abierto para la creación de entornos de desarrollo software) y puede ejecutarse en sistemas operativos Windows y Linux.

El presente proyecto se desarrolló con la versión *Platinum* 3.3 de CCS de forma indirecta desde Simulink para compilar y depurar el código generado, así como para realizar la comunicación entre el computador y el procesador utilizado.

2.4. EMBEDDED CODER DE SIMULINK

Simulink es una plataforma para simulación de múltiples dominios y diseño de sistemas dinámicos que cuenta con un entorno gráfico interactivo y un conjunto de librerías configurables. El software permite diseñar, simular, implementar y evaluar sistemas de control, de procesamiento de señales, de comunicaciones, entre otros.

Bajo el entorno de *Simulink* es posible construir prototipos directamente desde los modelos de algoritmos y sistemas de control. Con Simulink se puede generar código automáticamente para hardware desde los modelos diseñados usando herramientas como *Simulink Coder* o *Embedded Coder*, permitiendo centrar el trabajo en evaluar el comportamiento del sistema, no en la programación.[14]

Embedded Coder es una Interfaz de Programación de Aplicaciones (API) desde donde se puede generar código C y C++ compacto, verificarlo y optimizarlo para usarlo sobre procesadores embebidos y/o tarjetas de prototipado rápido mediante enlaces disponibles para entornos de desarrollo como Code *Composer Studio*, *VisualDSP, Eclipse, Green Hills MULTI IDE* y para sistemas operativos como Linux. La ejecución de código generado desde Simulink requiere que dicho código sea ajustado al hardware específico de destino.

[14] MATHWORKS. Rapid Prototyping. [En linea] http://www.mathworks.com/rapid-prototyping/ [citado en 2013].

2.4.1 Link para Code Composer Studio.

Como se mencionó anteriormente, *Code Composer Studio* (CCS) es un entorno de desarrollo integrado de Texas Instruments utilizado en la creación de software para DSP's y micro-controladores de la misma compañía. CCS puede ser utilizado junto con *Embedded Coder* para compilar y cargar el código generado sobre un dispositivo particular (de las familias C2000, C5000, y C6000) una vez dicho código se ajuste al hardware deseado. Después de cargar el código sobre el dispositivo de destino, es posible acceder a información del procesador corriendo a través de un enlace desde *Embedded Coder*. Con la creación de objetos de Matlab, *Embedded Coder* permite transferir información hacia y desde CCS, la memoria y los registros del procesador con el cual se trabaja.[15]

Embedded Coder automatiza completamente el desarrollo de la aplicación software embebida y hace más fácil evaluar las posibles diferencias entre los resultados de un modelo de simulación y los resultados del procesador ejecutando el código generado.

Para conectar a Matlab o Simulink con Code Composer Studio, *Embedded Coder* utiliza objetos (de programación) que pueden se manipulados para configurar el entorno de desarrollo. De igual forma, contiene tres elementos principales que permiten la comunicación con CCS y la generación y verificación de código fuente. Estos elementos son *Automation Interface, Project Generator* (que serán explicados en los siguientes capítulos) y un elemento de Verificación.

2.4.2 IDE Automation Interface.

IDE Automation Interface es una Interfaz de Programación de Aplicaciones (API) que proporciona un conjunto de métodos utilizados por Embedded Coder para comunicarse con CCS.[16] A través de *Automation Interface* se realiza la depuración de código y se hace el intercambio de datos entre Matlab y el procesador.

Los métodos que utiliza *Automation Interface* se configuran a partir de la creación de objetos de programación, donde se encuentran las propiedades de la conexión entre Matlab y CCS. Para la creación de un nuevo objeto se puede utilizar la función *ticcs* de Matlab especificando la tarjeta y el dispositivo de destino.* Una

[15] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Working with Texas InstrumentsTM Code Composer Studio 3.3 IDE.

[16] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Working with Texas InstrumentsTM Code Composer Studio 3.3 IDE.

* Más información sobre la creación de un objeto se puede encontrar en la Guía de usuario de Embedded Coder.

vez hecha la conexión entre Matlab y CCS, es posible leer y modificar características de la misma, y transferir datos entre Matlab y el DSP específico mediante la manipulación de los diferentes métodos del objeto. La implementación de objetos *ticcs* se basa en las capacidades de programación orientada a objetos de Matlab.

2.4.2.1. Intercambio de datos en tiempo real RTDX.

El intercambio de datos entre Matlab y el DSP puede realizarse de forma eficiente a través del enlace RTDX de *Texas Instruments* que proporciona un medio de comunicación bilateral para interactuar con el procesador mientras ejecuta aplicaciones en el mundo real. RTDX permite la transferencia de ordenes desde el computador y de información desde el procesador sobre su estado y el de la aplicación sin interferir en la misma.

El enlace RTDX puede ser utilizado con *Embedded Coder* y CCS para acelerar el desarrollo y despliegue de aplicaciones sobre procesadores C2000 de *Texas Instruments*, gracias a que mediante el intercambio de datos, es posible evaluar y analizar los algoritmos o modelos creados en Matlab o Simulink.

Con RTDX es posible:

- Enviar datos desde la memoria del procesador.
- Cambiar las características de la aplicación.
- Realizar cambios a los algoritmos si es requerido sin detener el programa en ejecución.

El desarrollo de algoritmos para el control digital requiere de una evaluación detallada de dichos algoritmos para determinar si cumplen con el objetivo de control. Habilitar la interacción de tiempo real entre el DSP y el computador, permite inspeccionar de forma fácil un proceso en acción, sus resultados y su desempeño.[17] El envío de datos hacia el procesador por ejemplo, tiene un efecto sobre la acciones del dispositivo y la representación gráfica de los datos provenientes del mismo ayudan a analizar el comportamiento del proceso casi en tiempo real.

---

[17] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Working with Texas InstrumentsTM Code Composer Studio 3.3 IDE. Getting Started with RTDX.

* Toda la información sobre la configuración de la conexión, la creación de canales y la transmisión y recepción de datos vía RTDX se puede encontrar en la Guía de usuario de Embedded Coder.

Para realizar un intercambio de datos vía RTDX, solo es necesario configurar la comunicación de tiempo real desde el objeto de Matlab que se creó para conectar con CCS y crear y habilitar los diferentes canales de entrada y salida por los cuales se enviarán los datos.* Se pueden crear tantos canales como se desee.

2.4.3 IDE Project Generator.

Una vez creado el enlace entre Matlab y Simulink con CCS mediante la interfaz de *Automation Interface*, *Embedded Coder* está listo para generar código desde modelos, compilarlo, cargarlo sobre un dispositivo objetivo e intercambiar información con el mismo. De esto se encarga el componente *IDE Project Generator*.

Project Generator es una interfaz que abarca un conjunto de métodos para la creación de proyectos en CCS y generar código con *Embedded Coder*.[18] Con la interfaz se realizan las siguientes actividades:

- Creación y construcción automática de proyectos para código generado con *Embedded Coder*.
- Generación automática de código óptimo y específico para un procesador determinado.
- Descarga, compilación y depuración automática del código en CCS desde Matlab.
- Personalizar la generación de código desde el menú de configuración de parámetros y los bloque de preferencia del procesador en Simulink.
- Descargar el código generado y ejecutarlo sobre el hardware directamente desde Matlab o Simulink.

Para la generación e implementación de código sobre un DSP a partir de un modelo de simulación en Simulink, se debe agregar y configurar un bloque llamado *Target Preferences*, el cual se utiliza para especificar información como el dispositivo y la tarjeta; la configuración del hardware; las condiciones de la memoria del procesador y características de la generación.[19] La información de este bloque será utilizada por Embedded Coder para la generación de código del modelo.

[18] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Working with Texas InstrumentsTM Code Composer Studio 3.3 IDE.

[19] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Project and Build Configurations for Embedded Targets.

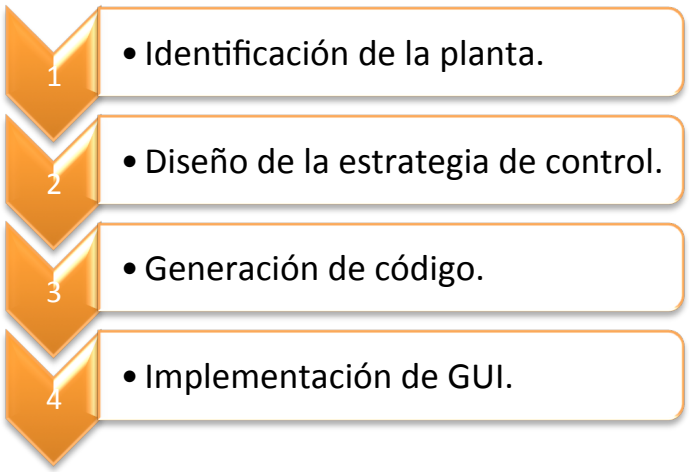
Posterior al paso de agregar el bloque *Target Preferences,* se debe realizar la configuración de los parámetros para configurar la generación del código del modelo creado y la forma como el código se ejecutará sobre el DSP.

3. METODOLOGÍA Y PROCEDIMIENTO

El uso de las herramientas de prototipado rápido que integran las plataformas Simulink y CCS ha sido muy poco explorado en la academia e industria regional. Por este motivo, para la ejecución del proyecto se realizó un estudio exploratorio y descriptivo que permitió realizar un acercamiento a la herramienta trabajada. Se hizo una documentación de diferentes fuentes de información con el propósito de obtener y presentar, con el máximo rigor o exactitud posible, la información sobre las herramientas y materiales que se utilizaron, así como de los procedimientos y técnicas más apropiados para alcanzar los objetivos propuestos. Esta documentación se basó principalmente en los manuales de usuario y hojas de datos de las plataformas, herramientas y dispositivos.

Con el objetivo de utilizar el conocimiento adquirido luego de la documentación para proponer soluciones prácticas al diseño de sistemas de control, se diseñó un procedimiento que permitiera obtener resultados objetivos sobre la confiabilidad de la herramienta trabajada. La figura 3.1 muestra el procedimiento realizado para la implementación del controlador digital.

Figura 3.1 Procedimiento realizado.



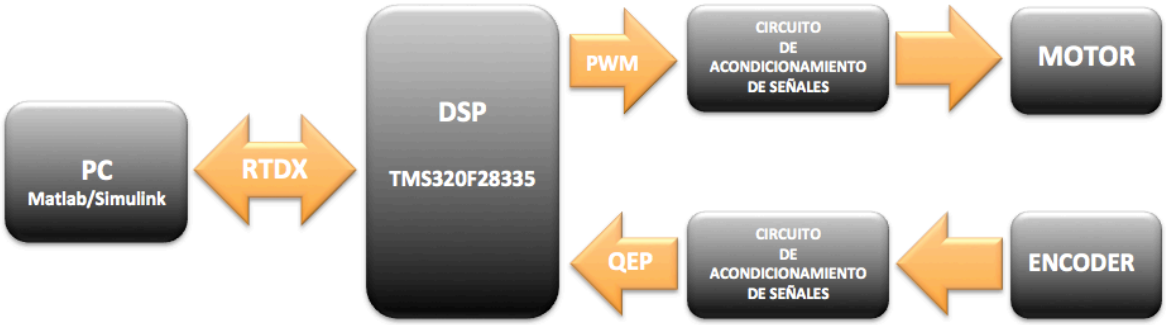
Fuente: El autor.

El proceso de desarrollo involucra etapas comúnmente utilizadas en el prototipado rápido de sistemas de control como lo son el diseño del sistema desde Simulink, la generación y el despliegue de código en el procesador, el monitoreo y la verificación de resultados desde una interfaz gráfica de usuario. Sin embargo se debió realizar una identificación previa de la planta que se controló para probar el prototipo construido, en este caso un motor DC.

3.1. HARDWARE UTILIZADO.

La herramienta de prototipado rápido de *Embedded Coder* se aplicó en el presente trabajo para el diseño y la implementación de un controlador digital de velocidad en el cual se integran los elementos hardware mostrados en la figura 3.2.

Figura 3.2 Diagrama de conexiones del hardware utilizado.



Fuente: El autor.

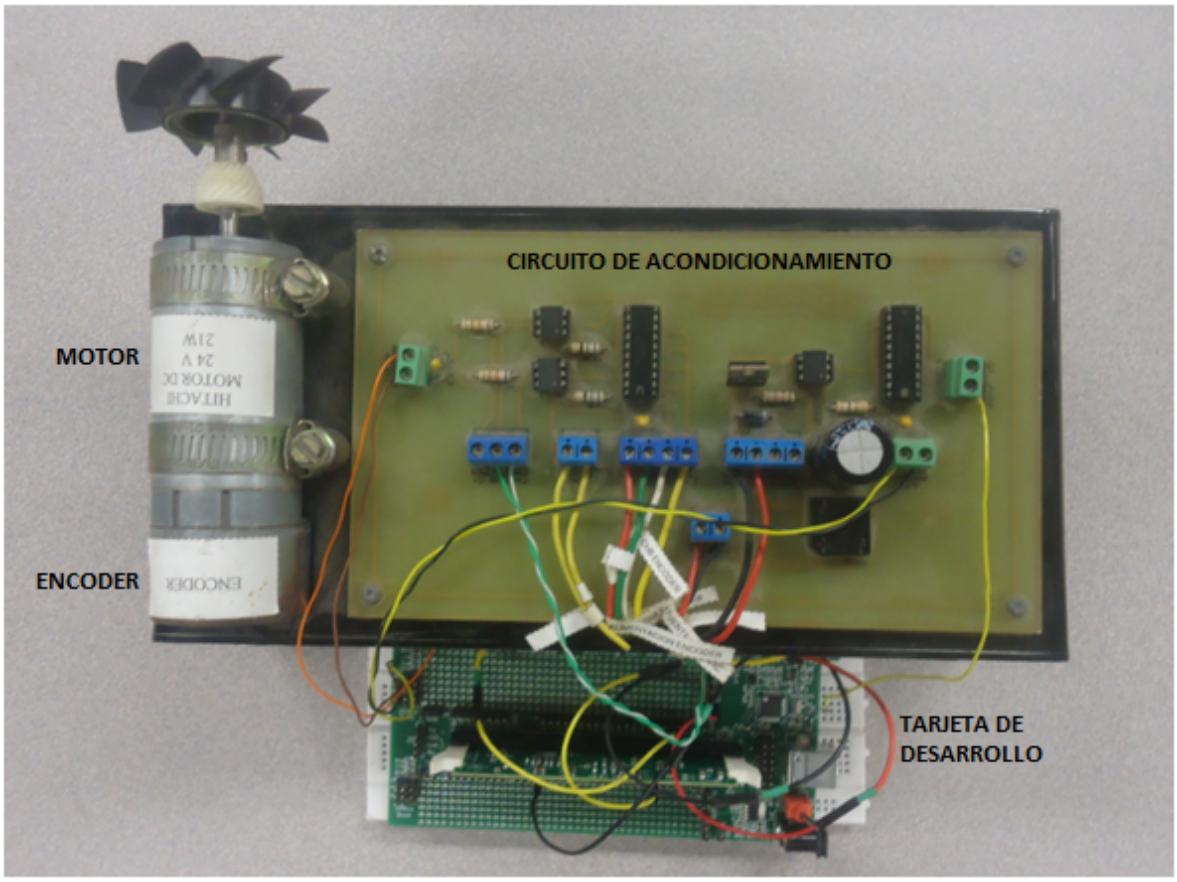
La tarjeta de evaluación TMS320C2000 con el DSP F28335 realiza las tareas de control de acuerdo a la señal de error producida por la diferencia entre el valor de referencia indicado por un usuario y la lectura de la velocidad de la planta.

La señal de referencia se transmite desde un computador a través de una interfaz gráfica vía intercambio de datos en tiempo real RTDX. El computador también recibe información sobre las variables del sistema como la velocidad de salida del motor y la energía aplicada al mismo en porcentaje de ciclo útil de la señal PWM generada.

Para aislar las señales de control y las señales de potencia, se utilizó un circuito electrónico que acondiciona la señales PWM provenientes del DSP y las señales de pulsos en cuadratura del sensor utilizado.

Finalmente se tiene la planta que consiste en un motor de 12 Voltios con un codificador de pulsos en cuadratura acoplado para medir la posición instantánea del motor.

Figura 3.3 Sistema físico de la planta y el controlador.



Fuente: El autor.

La siguiente 3.3 muestra el sistema planta y controlador con los respectivos circuitos de acondicionamiento de señal.

3.1.1 Planta a controlar.

La figura 3.4 muestra el motor que se utilizó como planta a controlar. Es un motor Hitachi DC de 24 Volts y 21 Watts, su velocidad a tensión nominal es 1200 rpm y a su eje se encuentra acoplando un codificador de pulsos de cuadratura.

El codificador de pulsos en cuadratura se alimenta a 5 Volts y cuenta con dos canales de salida (A y B) cuyas señales están desfasadas 90º una de la otra. El dispositivo genera 100 pulsos digitales cada revolución del eje del motor.

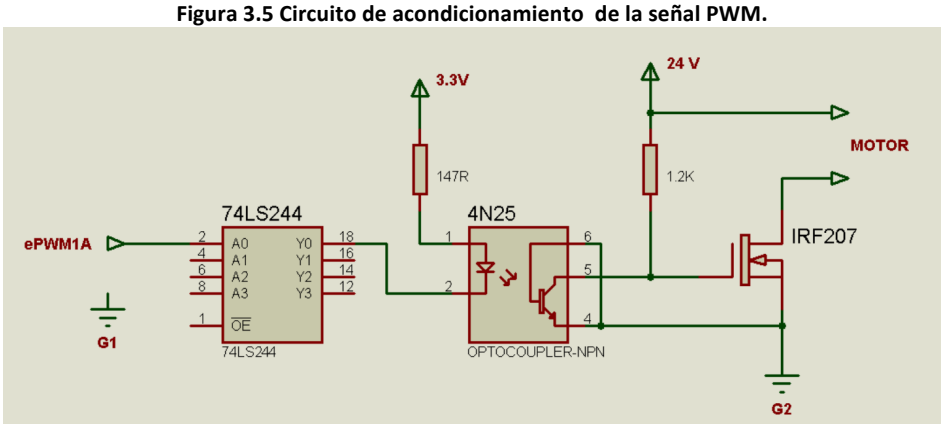
Figura 3.4 Motor DC de 12 Volts.

Fuente: El autor.

3.1.2 Circuito de acondicionamiento de señales.

La figura 3.5 muestra el circuito utilizado para amplificar la señal PWM y alimentar el motor DC. G1 corresponde a la tierra del DSP mientras que G2 corresponde a la tierra de potencia. Se utilizó un buffer y un opto-acoplador para aislar las señales de control y potencia, mientras que MOSFET actúa como interruptor de potencia para la señal PWM aplicada al motor.

Figura 3.5 Circuito de acondicionamiento de la señal PWM.

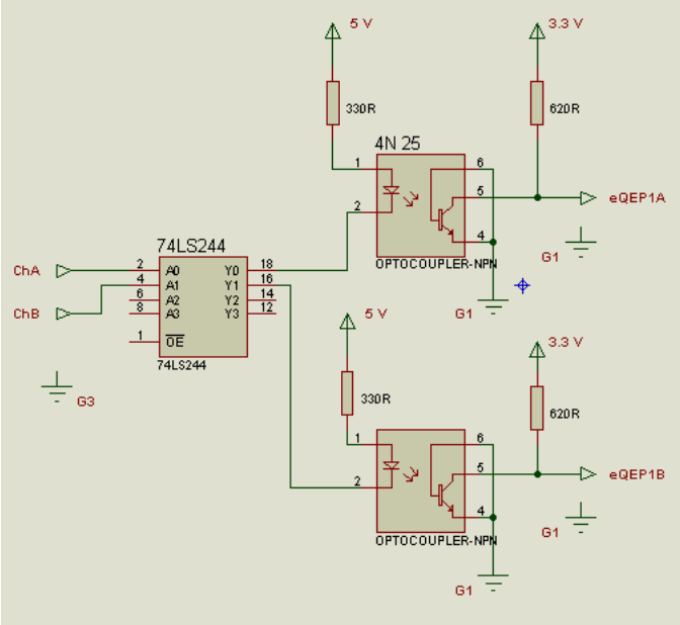


Fuente: El autor.

Para acondicionar las señales del codificador digital se utilizó el circuito mostrado en la figura 3.6. El codificador de pulsos en cuadratura se alimenta a 5 Volts, por lo tanto, se utilizó un opto-acoplador para aislar la señal y reducirla de una tensión de

5 Volts a 3.3 Volts permitidos a la entrada del periférico eQEP1 del DSP. G3 corresponde a la referencia de la alimentación del codificador.

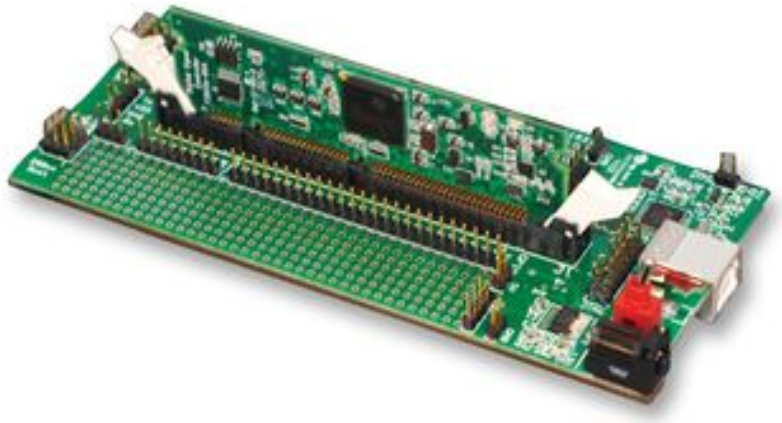
Figura 3.6 Circuito de acondicionamiento de señales QEP



Fuente: El autor.

3.1.3 Tarjeta de evaluación y desarrollo.

Figura 3.7 Tarjeta de evaluación TMS320C2000.



Fuente: Internet: http://www.ti.com/tool/tmdsdock28335

Como se mencionó anteriormente, el corazón del sistema de control embebido de la aplicación desarrollada en el presente trabajo corresponde a un controlador de señales digitales (DSC) TMS320F28335 de *Texas Instruments*, donde se ejecuta el software con los algoritmos de control. Las conexiones del dispositivo con otros elementos del sistema como el motor DC y el sensor codificador de cuadratura, se realizó a través de la tarjeta de experimentación TMS320C2000 mostrada en la figura 3.7.

La tarjeta de experimentación C2000 se alimenta a 5 Volts y brinda una alternativa fácil, rápida y de bajo costo para evaluar los dispositivos TMS320F28x. Consiste en una tarjeta madre de conexión con un zócalo para tarjetas de control DIMM de 100 pines. La tarjeta madre proporciona acceso a las señales digitales de propósito general y señales de los módulos ADC del Controlador de Señales digitales DSC.[20] También cuenta con:

- Un área de prototipado a cada lado del conector DIMM100 para conexiones adicionales.
- Posibilidad de utilizar el conector USB como emulador o un conector externo para emulación JTAG.
- Un conector para comunicación seria UART.
- Pines de alimentación de 5 Volts y 3.3 Volts para las áreas de prototipado.

Al sócalo DIMM100 se conectó una tarjeta de control F28335 Delfino como la que se muestra en la figura 3.8, la cual cuenta con la circuitería necesaria para la operación del DSC como reloj, regulador de alimentación, entre otros.

Figura 3.8 Tarjeta de control F28335 Delfino

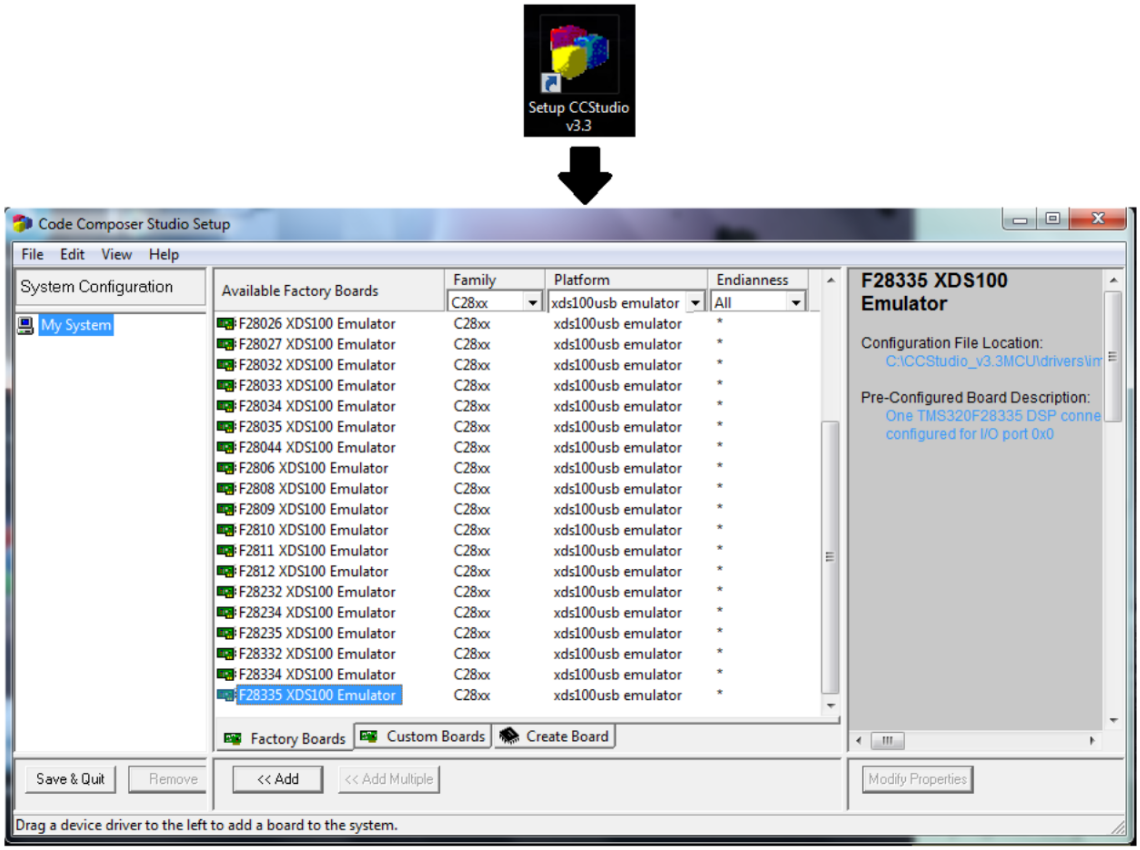
Fuente: El autor.

[20] TEXAS INSTRUMENTS. TMS320C2000 Experimenter Kit Overview. Guía de inicio rápido. 2009.

Configuración de la tarjeta en CCS.

Una vez instalado el software de desarrollo Code Composer Studio v3.3, se configuró para la tarjeta y el dispositivo trabajados. Abriendo la aplicación "*Setup CCStudio v3.3*" (creada normalmente en el escritorio del computador luego de la instalación de CCS) se despliega una ventana como la mostrada en la figura 3.9.

Figura 3.9 Ventana de configuración de dispositivo de CCS v3.3



Fuente: El autor.

Se agregó al sistema trabajado el emulador USB JTAG "*F28335 XDS100*" que se encuentra dentro de la familia C28x y la plataforma "*xds100 usb emulator*". De esta forma el DSC queda listo para ser programado desde Simulink y/o Matlab.

3.2. GENERACIÓN DE CÓDIGO.

Desde Simulink es posible crear modelos para dispositivos hardware de la misma forma como se crea cualquier modelo de simulación, utilizando los bloques de la librería “Embedded targets”, los bloques de los productos “System Toolboxes” y bloques convencionales con algunas excepciones.

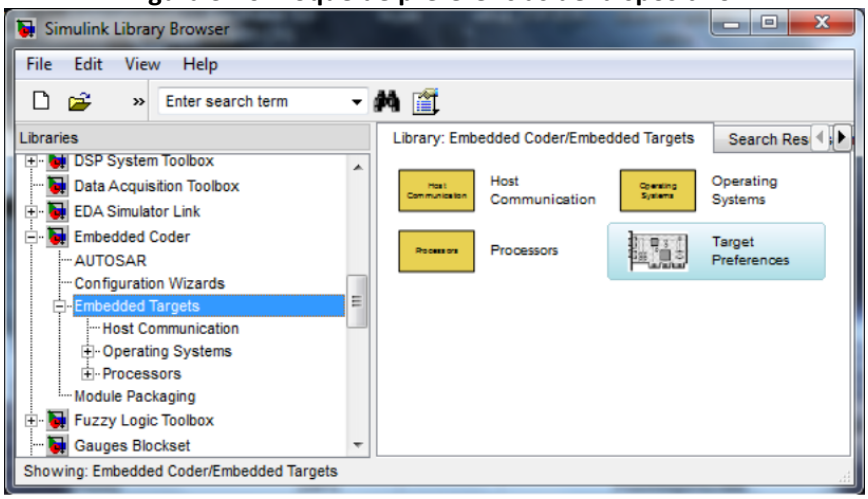
El trabajo expuesto en el presente documento tiene como fundamento el uso de la plataforma *Embedded Coder* de Simulink para generar, compilar y cargar código sobre el DSC F28335. Independiente de la aplicación a la cual se destine la plataforma, hay un conjunto de parámetros inherentes a la tarjeta y procesador utilizado, al entorno de desarrollo con el cual se conecta Embedded Coder (en este caso es CCS) y al código generado que se deben configurar para evitar errores en la generación y compilación.

A continuación se explica la configuración realizada a estos parámetros y las herramientas proporcionadas por Embedded Coder, Simulink y Matlab para la creación de los modelos, la comunicación con el procesador y la generación y compilación del código.

3.2.1 Bloque de preferencias del dispositivo.

El bloque de preferencias del dispositivo o “target preferences”, guarda la información sobre el procesador y tarjeta trabajados, las condiciones del hardware, configuración de la memoria y características de la generación de código. La figura 3.10 muestra el aspecto del bloque.

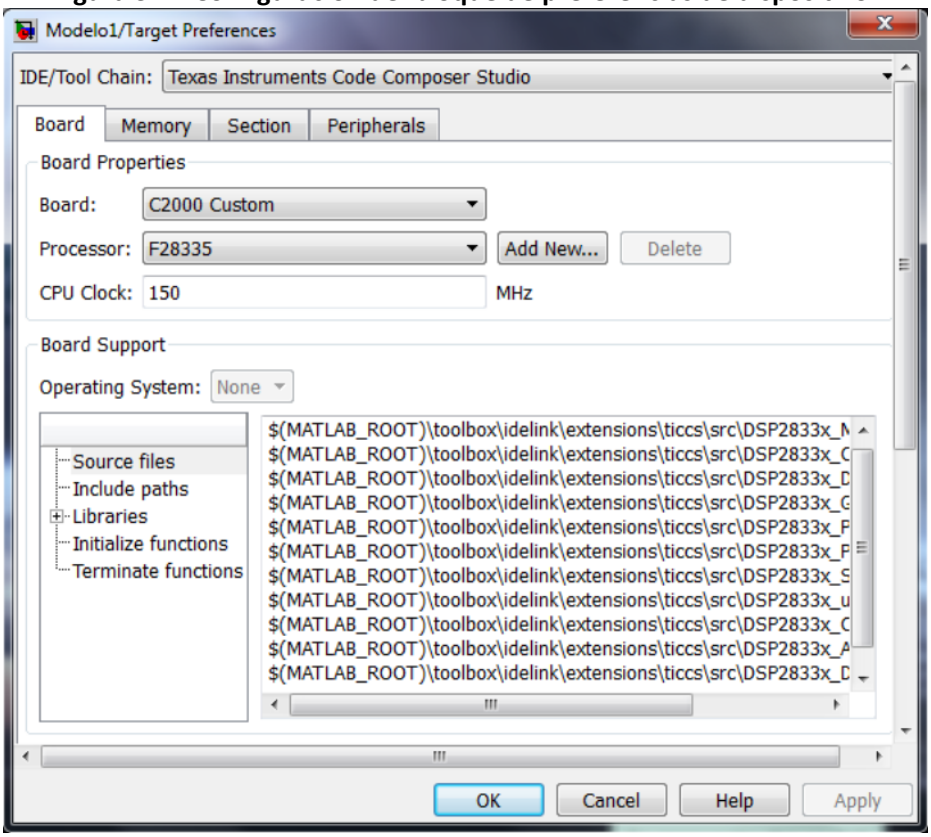
Figura 3.10 Bloque de preferencias del dispositivo.



Fuente: El autor.

Este bloque se encuentra dentro de la librería “Embedded Targets” y debe agregarse al modelo antes de la generación del código. La figura 3.11 muestra las configuraciones realizadas al bloque para el desarrollo del proyecto.

Figura 3.11 Configuración del bloque de preferencias de dispositivo.



Fuente: El autor.

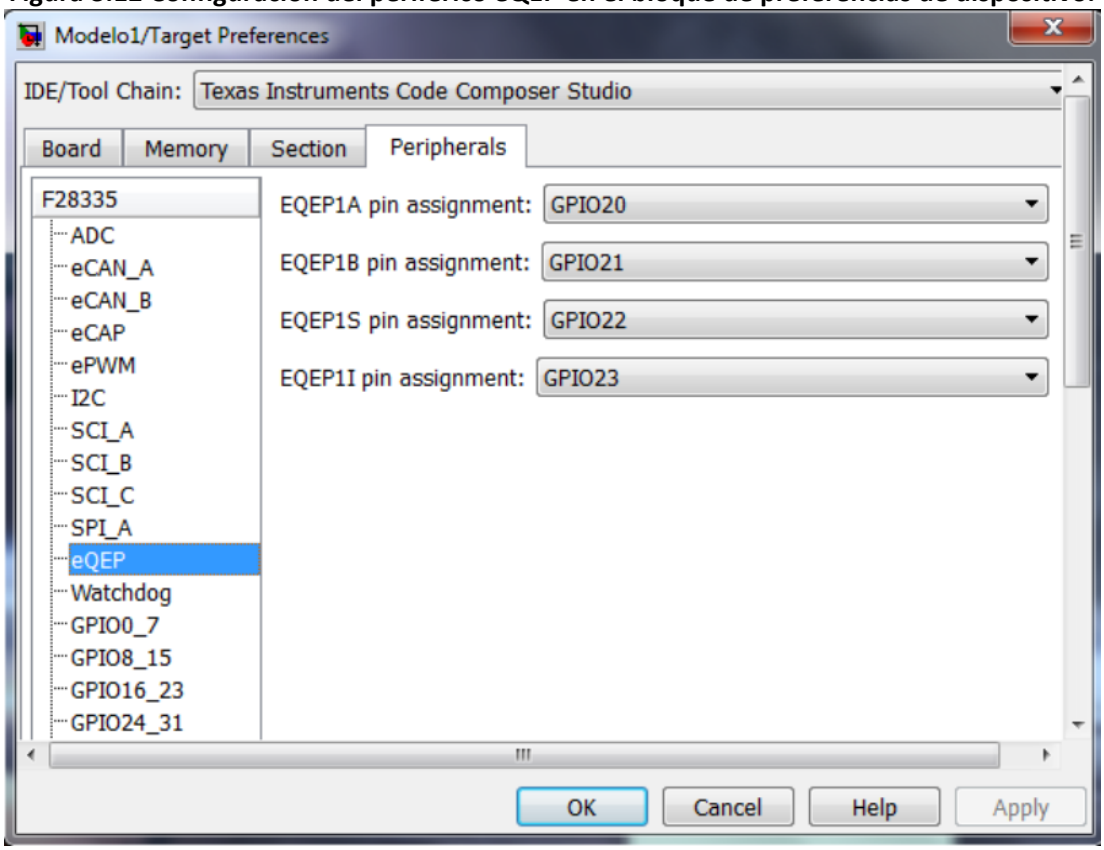
En la opción de selección “*IDE/Tool Chain*” se asigna el entorno de desarrollo con el cual Embedded Coder se conectará para compilar y cargar el código generado; en este caso se seleccionó la opción “Texas Instruments Code Composer Studio”. Dentro del recuadro “*Board Properties*” se indican la tarjeta utilizada (C2000 custom), el procesador (F28335) y la velocidad del reloj del sistema (150 MHz).

En el recuadro “*Board Suport*” se establecen algunas condiciones para la generación de código como ubicaciones del código y librerías utilizadas. Para los códigos generados en el proyecto no se realizaron modificaciones a este último recuadro ni a las pestañas de memoria y secciones.

En la figura 3.12 se observa la pestaña de periféricos del dispositivo objetivo. En esta sección se configuran algunas características de los periféricos del DSP como módulos ADC, PWM, comunicaciones, codificador de cuadratura, entradas y

salidas de propósito general, entre otros. La única configuración realizada en esta pestaña para el desarrollo del proyecto se hizo al módulo eQEP donde se asignaron los pines GPIO20, GPIO21, GPIO22 y GPIO23 como canales eQEP1A, eQEP1B, eQEP1S y eQEP1I respectivamente.

Figura 3.12 Configuración del periférico eQEP en el bloque de preferencias de dispositivo.



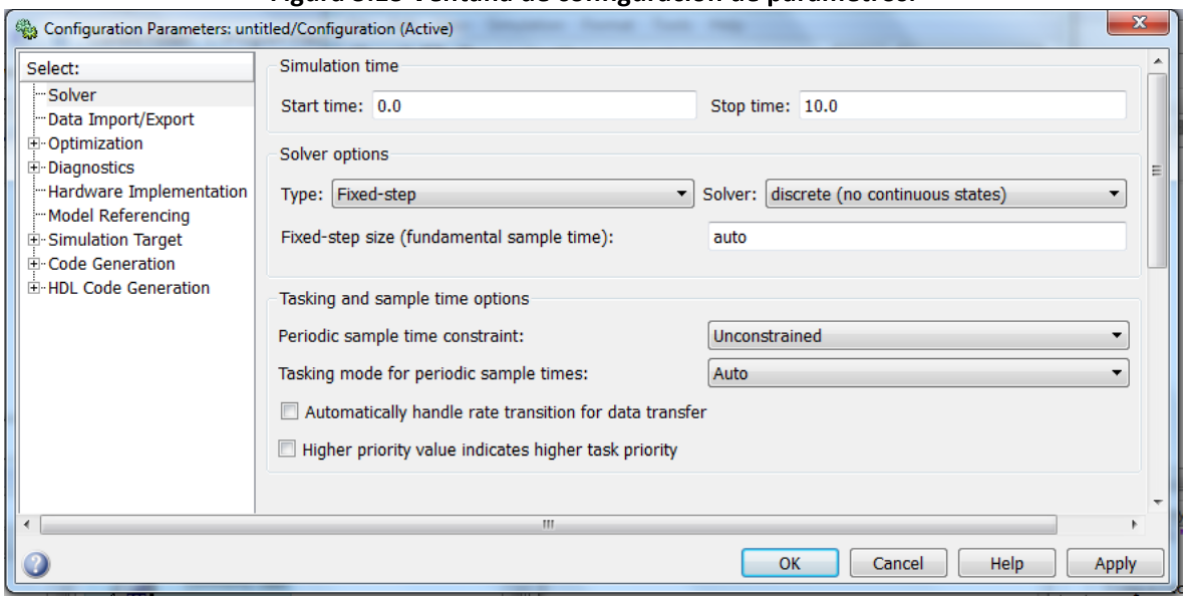
Fuente: El autor.

3.2.2 Configuración de parámetros.

Antes de generar código desde un modelo de Simulink, es necesario configurar los parámetros de creación y ejecución del código en la ventana de diálogo de configuración de parámetros de Simulink. La figura 3.13 muestra el aspecto de la ventana de configuración de parámetros que se puede desplegar en el menú de simulación del modelo o digitando el comando Cntrl+E.

La ventana cuenta con diferentes categorías propias del modelo, simulaciones y código generado, que se pueden configurar de acuerdo al uso que se le va a dar al modelo construido. En este documento se hace referencia a tres de dichas categorías que se configuraron para generar e implementar código sobre el DSP F28335.

Figura 3.13 Ventana de configuración de parámetros.



Fuente: El autor.

3.2.2.1. Solver.

El solver o solucionador se encarga de calcular los estados del modelo en pasos sucesivos de tiempo. En esta categoría se configuran los tiempo de inicio y detención de una simulación y el tipo de algoritmo de integración numérica. Para la generación de código, el tiempo de parada es infinito (Inf), es decir, el modelo se ejecutará sin detenerse. La categoría se configuró de la siguiente forma:

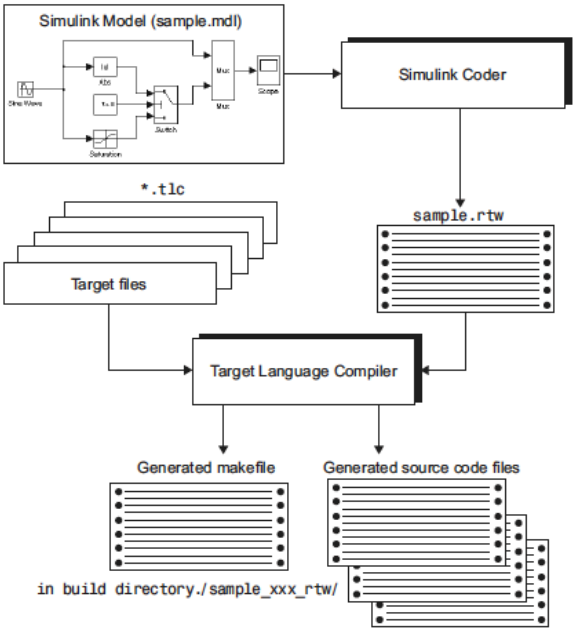
- *Solver type: Fixed-step.*
- *Solver: Discrete (no continuous states).*
- *Fixed-step size: auto.*
- *Tasking mode for periodic sample times: Single tasking*.

3.2.2.2. Code Generation.

La categoría de generación de código permite configurar el tipo de archivos de procesador (de extensión .tlc) que Simulink va a utilizar para convertir el archivo del modelo .mdl creado en código de lenguaje C o C++ como lo muestra la figura 3.14.

Cuando se genera código a partir de un modelo de Simulink con extensión .mdl, el primer paso en el proceso automático es la generación de un archivo con extensión .rtw que contiene toda la información necesaria del modelo. Una herramienta llamada Compilador de lenguaje de dispositivo (TLC) utiliza el archivo .rtw en combinación con los archivos de dispositivo .tlc para generar el código.

Figura 3.14 Funcionamiento del compilador de lenguaje de dispositivo TLC.



Fuente: MATHWORKS. Simulink Coder - Target Language Compiler. 2012

La configuración de los parámetros de generación de código se realizó de la siguiente forma:

- *System target file*: *Idelink_grt.tlc.* Esta opción permite establecer el uso de archivos de tiempo real genérico configurados para generar código C propio de un entorno de desarrollo, el cual se especifica en la categoría IDE Link de la ventana de configuración de parámetros.

3.2.2.3. IDE Link.

Cuando se selecciona el archivo *Idelink_grt.tlc* en el paso anterior, aparece una nueva pestaña en la ventana de configuración de parámetros llamada *IDE Link* la cual se configuró de la siguiente forma:

- *Build format: Project.*
- *Build Action: Build and execute.*
- *Configuration: custom.*
- Los espacios *Compiler options string* y *Linker options string* se dejaron vacíos.
- En el recuadro *Link Automation*, se seleccionó la opción *Export IDE link handle to base workspace* para crear el objeto de programación propio de la conexión entre Matlab y CCS.

Algunos elementos de la ventana de configuración de parámetros no se modificaron debido a que sus valores por defecto son los apropiados para la generación de los códigos implementados durante el desarrollo del proyecto.

3.2.3 Generación, compilación y descarga del código.

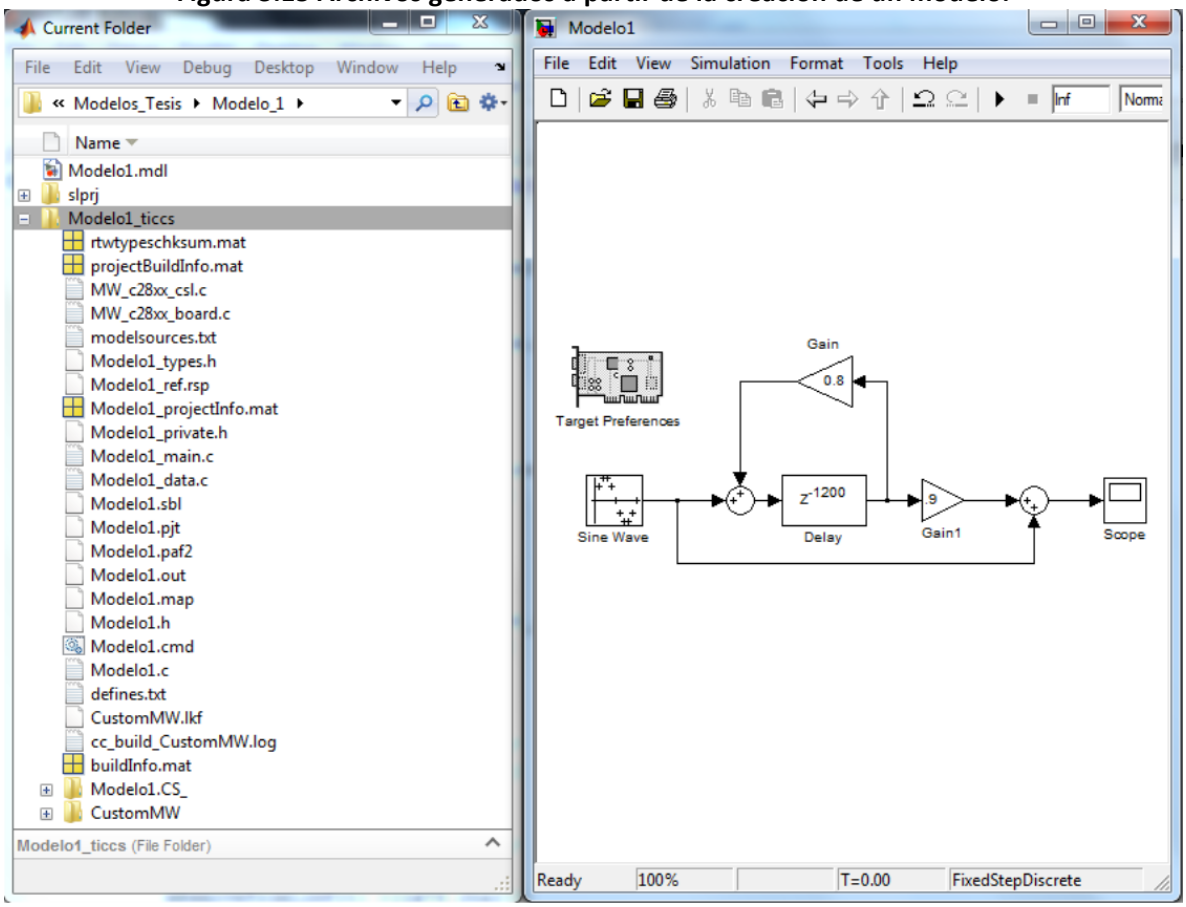
Una vez agregado el bloque de preferencias del dispositivo y configurados los parámetros, se procede a generar el código del modelo y descargarlo en el procesador para su posterior ejecución. En la barra de herramientas del modelo se selecciona la opción *tools* > *Code generation* > *Build Model* para realizar la generación del código y la compilación del mismo a través del enlace con CCS. Debido a que se configuró la opción en la ventana de configuración de parámetros, también se realiza la descarga y ejecución del código sobre el procesador.

Después de realizar la orden de construcción del modelo, Simulink y Embedded Coder realizan el siguiente procedimiento observable en la ventana de Matlab:

- Conectarse con Code Composer Studio.
- Crear una carpeta en el directorio de Matlab con el nombre del modelo seguido de las letras “ticcs” donde se generan los archivos con extensiones .rtw, .h y .c.
- Conectar el procesador con CCS y Matlab.
- Construir el proyecto en CCS con el nombre del modelo y la extensión .pjt.
- Descargar el archivo ejecutable con el nombre del modelo y la extensión .out sobre el DSP.
- Ejecutar el programa descargado en el DSP.

La figura 3.15 muestra el directorio de Matlab luego de generar y descargar código desde un modelo de simulación llamado Modelo1. En este caso se pueden observar entre otros, el archivo del modelo de simulación (Modelo1.mdl) y la carpeta generada con los archivos Modelo1.rtw (que contiene información codificada del modelo), Modelo1.c, modelo1_data.c y modelo_main.c (generados por el compilador de lenguaje de dispositivo TLC utilizando el archivo *idelink_grt.tlc*). También se puede observar el archivo del proyecto generado a partir de la compilación del código y su ejecutable correspondiente descargado sobre el procesador. El archivo *projectBuildInfo.mat* contiene información sobre el proyecto de CCS generado.

Figura 3.15 Archivos generados a partir de la creación de un modelo.



Fuente: El autor.

NOTA: Antes de realizar la construcción del modelo, se recomienda verificar que se cuenta con el soporte para la conexión entre Matlab y CCS, y que la tarjeta y el procesador trabajados se encuentran configurados. La verificación puede realizarse con el comando *ccsboardinfo* como lo muestra la figura 3.16.

Figura 3.16 Espacio de trabajo de Matlab luego de verificar soporte con CCS.

```
Board Board                          Proc Processor  Processor
 Num  Name                           Num  Name       Type
 ---  -------------------------------  ---  -------------
1  C6xxx Simulator (Texas Instrum .0   6701     TMS320C6701
0  C6x13 DSK (Texas Instruments)   0   CPU      TMS320C6x1x
```

Fuente: THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario.

Mientras el programa se ejecuta dentro del DSP, es posible acceder a información sobre registros y direcciones de memoria y enviar información al dispositivo a través del objeto creado para la conexión entre Matlab y CCS y haciendo uso de los comandos *regwrite* y *regread* como lo muestra la tabla 3.1. Para detener la ejecución desde la ventana de comandos de Matlab, reiniciar el procesador o borrar el enlace, se pueden utilizar los comandos *halt, restart* y *clear* respectivamente.

Tabla 3.1 Funciones de enlace entre Matlab y CCS.

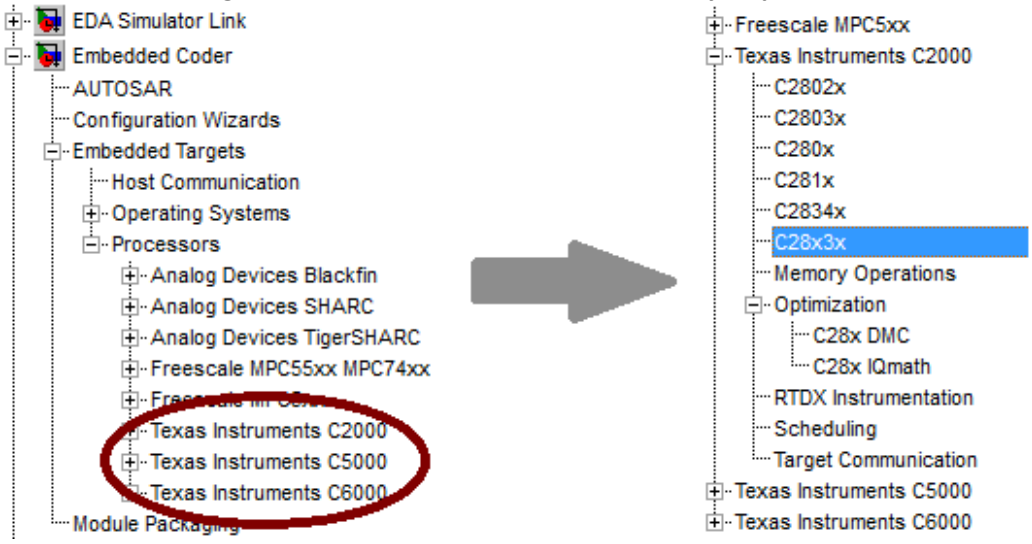
Comando	Función
visible(IDE_Obj,1)	1:pone visible CCS en el escritorio. 0: Oculta el entorno. IDE_Obj hace referencia al objeto creado.
display(IDE_Obj)	Informacion del estado del target
info(IDE_Obj)	Condiciones del target
isrunning(IDE_Obj)	informacion de si el target referenciado en el objeto IDE_Obj esta corriendo
load(IDE_Obj,'archivo.out',20)	Carga el ejecutable en el procesador.
run(IDE_Obj,'runtohalt',30)	Se ejecuta el programa en el dispositivo referenciado en el objeto IDE_Obj
halt(IDE_Obj)	Se detiene la ejecución del programa en el dispositivo referenciado en el objeto IDE_Obj
restart(IDE_Obj)	Se reinicia el dispositivo referenciado en el objeto IDE_Obj
clear IDE_Obj	Borra el enlace entre Matlab, CCS y el dispositivo.
valor_reg= IDE_Obj.regread('Reg','binary',20)	Guarda en la variable valor_reg el valor del registro Reg.
DE_Obj.regwrite('Reg',Num,'binary',20)	Escribe el número Num en el registro Reg.

3.2.4 Librería de bloques para dispositivos C28x3x

Simulink contiene un conjunto de librerías asociadas a los dispositivos *Texas Instruments,* las cuales pueden ser agregadas a un modelo para trabajarlos como si se tratara de un bloque convencional. Este conjunto de bloques se encuentran dentro de la librería de *Embedded Coder* como lo muestra la Figura 3.17. Dentro de la librería *Texas Instruments C2000* se puede acceder a los periféricos de cada dispositivo y a otros bloques que pueden ser utilizados para la transmisión de

datos en tiempo real, operaciones aritméticas complejas y tareas de interrupciones para el procesador utilizado. Para el desarrollo del proyecto, se trabajó con las librerías *C28x3x, RTDX Instrumentation* y *Optimization*.

Figura 3.17 Librerías de Embedded Coder y dispositivos C28x3x.

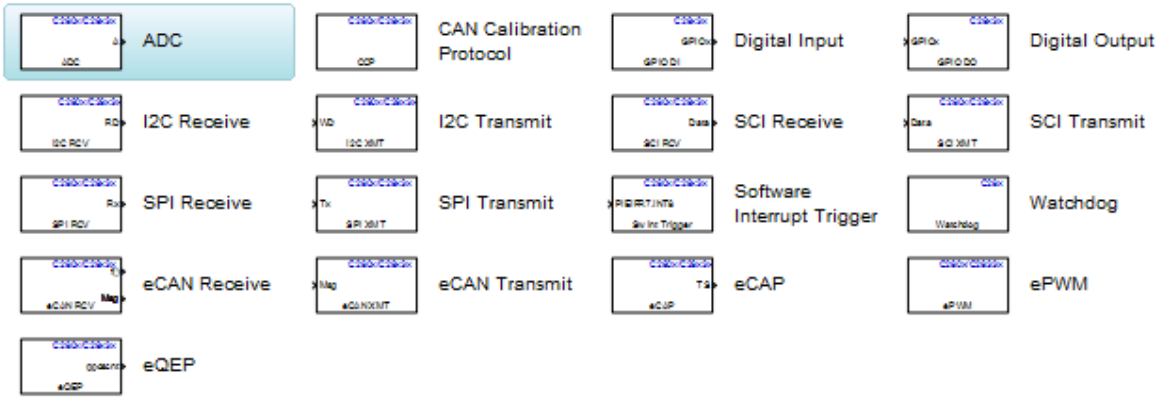


Fuente: El autor.

Cuando se utilizan estos bloques, el software Simulink Coder genera código de forma automática e inserta los controladores de hardware de cada bloque agregado al modelo. Estos controladores se incluyen también en el código C generado.

La figura 3.18 muestra los bloques proporcionados para dispositivos C28x3x. Cada bloque se puede configurar para personalizar el funcionamiento de su periférico asociado en el DSP cuando se ejecute el código en el mundo real.

Figura 3.18 Bloques de periféricos para dispositivos C28x3x.



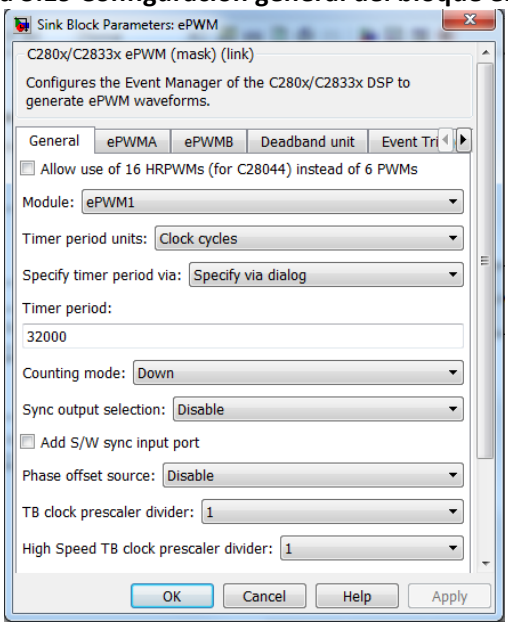
Fuente: El autor.

A continuación se explican los bloques utilizados para la implementación del controlador digital de velocidad sobre el procesador TMS320F28335.

3.2.4.1. Bloque ePWM.

El bloque ePWM configura y controla la generación de las señales de salida en los periféricos de modulación de ancho de pulso del procesador. Cada bloque tiene la opción de configurar un módulo PWM del DSP, es decir, dos canales de salida (A y B).

Figura 3.19 Configuración general del bloque ePWM



Fuente: El autor.

La figura 3.19 muestra los valores que se fijaron en la ventana de parámetros generales para el bloque PWM en los modelos creados para la generación de código. Los principales parámetros configurados tienen las siguientes funciones:

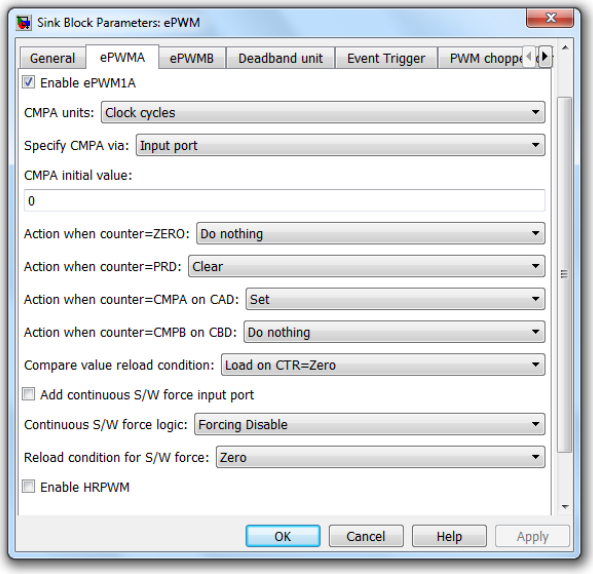
- *Module*: Especifica el módulo que se va a trabajar. Existen 6 módulos PWM disponibles en el DSP.
- *Timer period units*: Define las unidades de tiempo del periodo de la señal generada. Para evitar cálculos de conversión innecesarios se asignaron unidades de ciclo de reloj.
- *Specify timer via*: El periodo de la señal puede ser variado de forma externa seleccionando la opción “Input port”, sin embargo, para las aplicaciones desarrolladas el periodo de la señal permanece constante; por consiguiente su valor fue fijado en la ventana de diálogo.
- *Timer period*: Hace referencia al periodo de la señal PWM que se va a generar en ciclos de reloj. Este periodo se asignó a 32000 unidades, es decir, una frecuencia de 4.68 KHz (El reloj del sistema es de 150 MHz).
- *Counting mode:* Este parámetro hace referencia a la forma en la cual el temporizador de periodo realizará el conteo de tiempo. En este caso se fijó un conteo descendente, es decir, desde 32000 hasta cero.
- *TB clock prescaler divider:* Hace referencia a la base de tiempo del contador, es decir, la cantidad de ciclos del reloj principal que harán que el contador incremente. Con el pre-escalador fijado en 1, el contador se incrementa cada ciclo del reloj principal (150 MHz).

En la figura 3.20 se muestra la configuración realizada a la pestaña ePWMA para establecer características de conmutación de la señal de salida PWMA.

El canal PWMA se configuró para que el valor del registro CMPA con el cual se va a comparar el contador en todo momento (para determinar el ciclo útil de la señal de salida) sea establecido de forma externa como una entrada al bloque.

El valor del registro CMPA corresponde al ciclo útil de la señal de salida y puede darse en unidades porcentuales del periodo total del contador o en ciclos de reloj como fue establecido en este caso, por lo tanto debe estar dentro del rango de cero a 32000. Se fijó el valor inicial igual a cero, es decir, un ciclo útil inicial de 0% hasta que su valor sea modificado en el puerto de entrada al bloque.

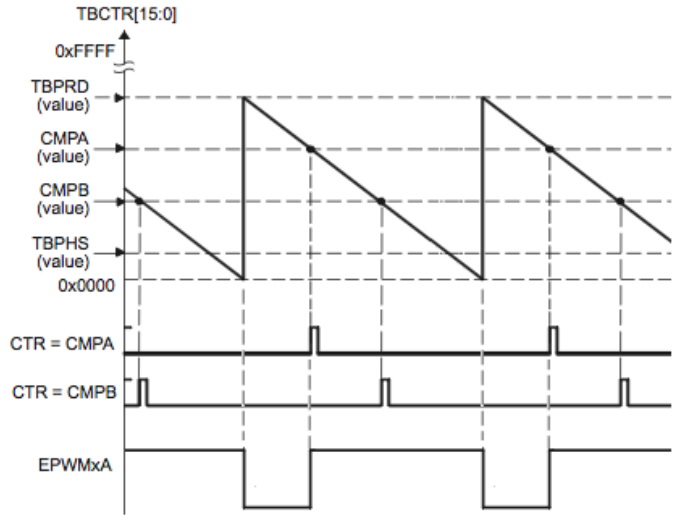
Figura 3.20 Configuración del canal ePWMA



Fuente: El autor.

La configuración fijada ocasiona que la señal de salida conmute a un nivel lógico alto cada vez que el contador se hace igual al valor del registro CMPA y a un nivel lógico bajo cuando se hace igual al periodo establecido (32000). La figura 3.21 permite comprender el funcionamiento del módulo configurado. Cuando el contador se hace igual a otros valores como cero o el valor del registro CMPB (registro de comparación del canal B) no se genera ningún cambio en la señal de salida PWMA.

Figura 3.21 Señales del módulo ePWM1A con la configuración hecha.



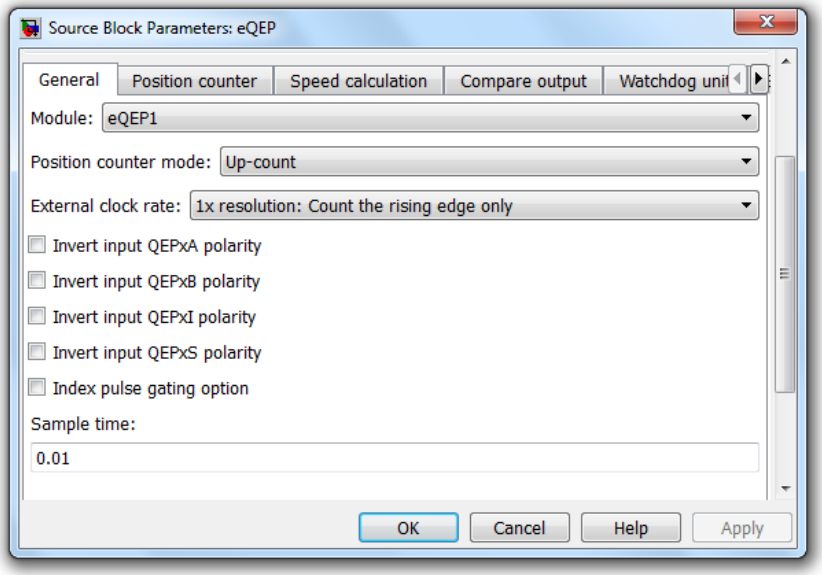
Fuente: El autor.

Las otras pestañas de la ventana de configuración del bloque PWM utilizado se dejan en sus valores por defecto.

3.2.4.2. Bloque eQEP.

El módulo de codificador de pulsos en cuadratura mejorado eQEP, se utiliza para interactuar con un codificador rotacional o lineal y obtener información sobre su posición, dirección de desplazamiento y velocidad.

Figura 3.22 Configuración general del bloque eQEP

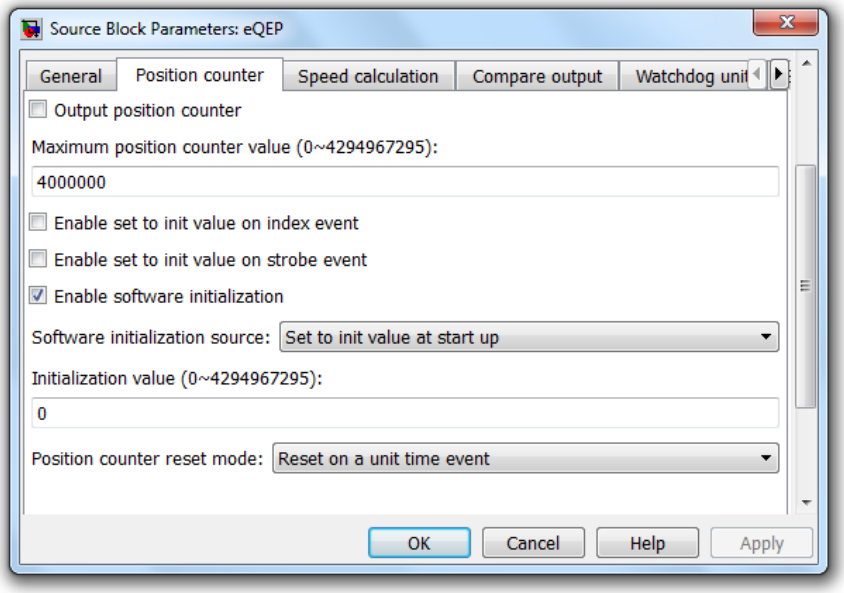


Fuente: El autor.

Las figuras 3.22, 3.23 y 3.24, muestran la configuración del bloque eQEP para obtener en su salida el valor de la velocidad del motor DC utilizado en el proyecto.

En la pestaña de configuración general se seleccionan parámetros como el tipo de conteo y el periodo de muestras en la salida del bloque. Para la lectura de la velocidad del codificador utilizado en el proyecto, se utilizó el módulo eQEP1 configurado para realizar un conteo ascendente de flancos positivos (debido a que no se tiene inversión de giro) y actualizando la señal de salida cada 10 ms.

Figura 3.23 Configuración del contador de posición de bloque eQEP



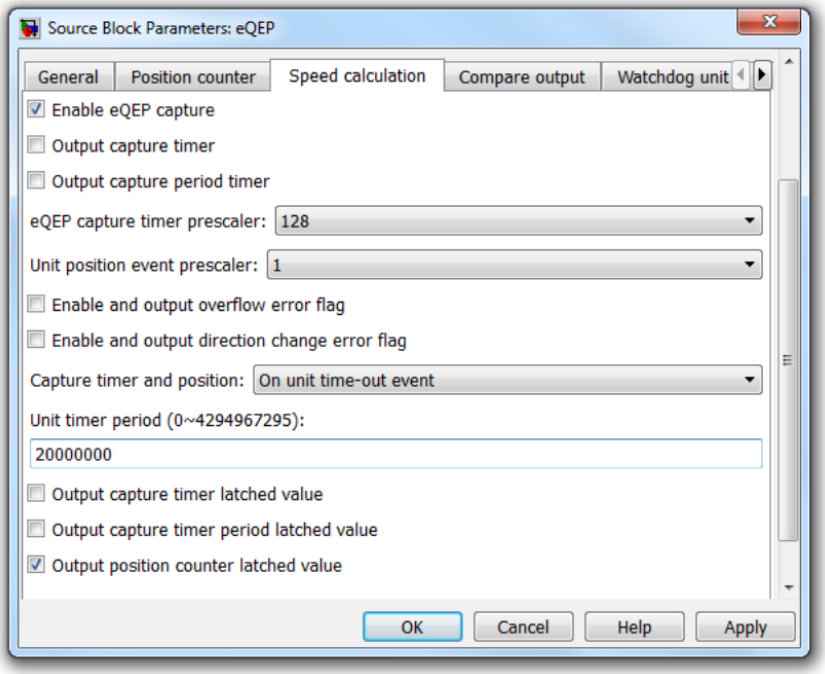
Fuente: El autor.

La pestaña para configurar los parámetros del contador de posición proporciona la posibilidad de fijar el valor del contador como una salida del bloque, permitiendo tener una lectura de la posición del codificador digital; esta opción se deshabilitó debido a que solo se tiene la necesidad de tener el valor de la velocidad.

Dentro de la ventana de configuración del contador de posición también se debe establecer el valor máximo del contador. Para el bloque utilizado se fijó un valor superior a la cantidad de pulsos generados por el codificador con el motor girando a la máxima velocidad en un rango de tiempo igual al del periodo del temporizador de unidades. Este temporizador de unidades genera interrupciones periódicas en las cuales se transfiere y reinicia el valor del contador de posición.

Por otro lado, también se estableció la inicialización en cero del contador cuando el programa inicia su ejecución sobre el DSP habilitando la casilla *Enable software initialization* y seleccionando la opción *set to init value at start up* con valor igual a cero.

Figura 3.24 Configuración del cálculo de velocidad en el bloque eQEP



Fuente: El autor.

En la pestaña de cálculo de velocidad se configuran los parámetros referentes al cálculo y lectura de la velocidad del codificador de cuadratura.

Para obtener la velocidad se seleccionó la casilla *Enable eQEP capture*. Esta opción habilita la unidad de base de tiempo. De esta forma, el temporizador de unidad (QUTMR) de 32 bits sincronizado por el reloj del sistema genera interrupciones periódicas cada vez que iguala el valor establecido en el registro de periodo de unidad (QUPRD). En cada interrupción el contador de posición transfiere el valor de la cantidad de pulsos que ocurrieron en dicho periodo y luego se reinicia para realizar el conteo de nuevo. El periodo del temporizador se estableció en 20.000.000 unidades.

La velocidad en (pulsos de codificador / segundos) puede deducirse de la siguiente ecuación.

$$V = \frac{X}{(QCAPCTL\_UPPS) * (QUPRD) * SYSCLKOUT} = \frac{X}{20000000 * 150000000^{-1}}$$

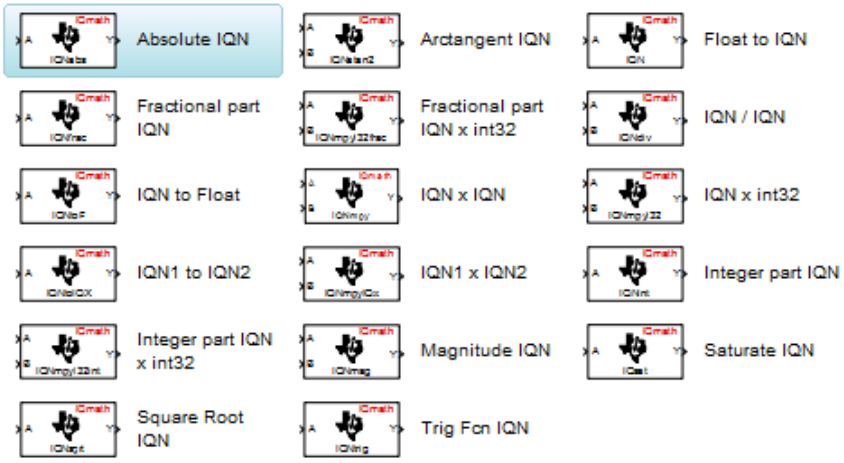
Donde X hace referencia a la cantidad de flancos del codificador detectados, QCAPCTL_UPPS al pre-escalador de los flancos del codificador y SYSCLK al periodo del reloj del sistema en segundos.

La información sobre el valor del contador de posición en el momento de cada interrupción queda disponible como una señal de salida del bloque eQEP mediante la selección de la casilla *Output position counter latched value*.

3.2.5 Librerías IQmath y DMC.

Texas Instruments proporciona una libraría que puede ser utilizada en el código fuente, para optimizar el desarrollo de operaciones matemáticas de punto flotante en la familia de procesadores C28x. *Embedded Coder* cuenta con el conjunto de bloques IQmath mostrados en la figura 3.25, que corresponden a las funciones presentes en la librería IQmath de Texas Instruments.

Figura 3.25 Librería de bloques IQMATH



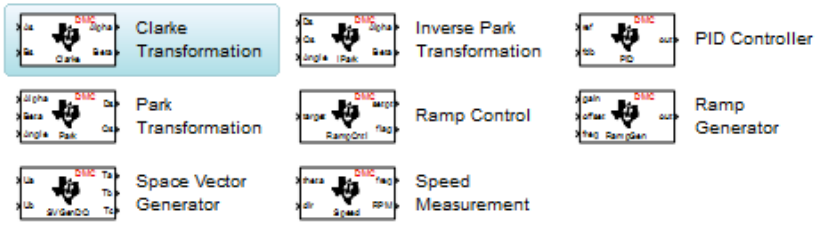
Fuente: El autor.

Las entradas y salidas de los bloques IQmath son generalmente datos de punto fijo y con formato Q. La posición de la coma dentro de cada número binario es expresada en la notación de formato Q, representación de los números de punto fijo de Texas Instruments. Esta notación toma la forma Qm.n, donde *Q* designa que el número se encuentra en notación de formato Q, *m* es el número de bits usados para designar la parte entera del dato y *n* es el número de bits usados para designar la parte fraccionaria del dato.

En formato Q, el bit más significativo es designado como bit de signo. Por lo tanto, para representar un dato de punto fijo con signo en formato Q requiere m+n+1 bits.

Las funciones IQmath son utilizadas para el control digital de motores dentro de la librería DMC de Texas Instruments para dispositivos C28x. La librería DMC contiene un conjunto de módulos que realizan operaciones de transformación, control y generación de señales; algunos de estos módulos pueden ser utilizados desde un modelo de Simulink para la generación automática de código. La figura 3.26 muestra los módulos de la librería disponibles en bloques dentro de Embedded Coder. El desarrollo del proyecto incluye el uso del bloque *PID Controller* para implementar el controlador de velocidad de la planta.

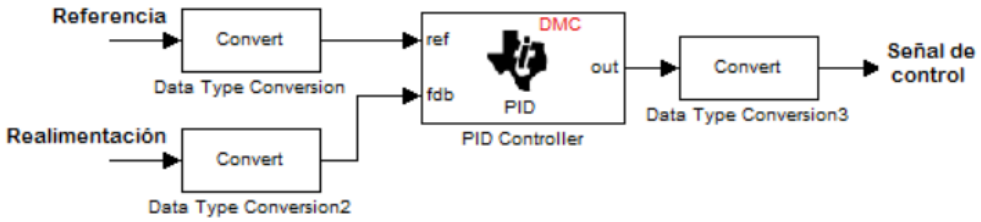
Figura 3.26 Librería de bloques DMC.



Fuente: El autor.

El bloque *PID Controller* tiene como entradas la señal de referencia (ref) y la señal de realimentación (fdb) y en la salida, la señal de control que se aplica a la planta. Las señales de entrada y salida tienen formato Q0.15 por lo que debe agregarse un bloque de conversión de tipo de dato a cada una para evitar errores por compatibilidad de formato como lo muestra la figura 3.27.

Figura 3.27 Conversión de tipo de datos conectados a los puertos del bloque PID.



Fuente: El autor.

La señal de control generada tiene la siguiente forma:

$$U(s) = \left[K_P + \frac{K_I}{s} + K_D s\right] E(s)$$

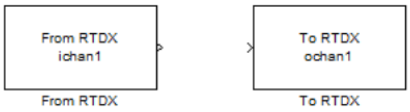
Donde U(s) es la señal de control generada, E(s) la señal de error, Kp, Ki y Kd son las ganancias proporcional, integral y derivativa respectivamente del controlador. El valor de las ganancias son configuradas en las propiedades del bloque.

3.2.6 Módulo de comunicaciones RTDX.

El intercambio de datos en tiempo real RTDX es una herramienta de Texas Instruments que permite conocer de forma continua el desempeño de un procesador en una aplicación del mundo real a través de la transferencia de datos entre el procesador y un computador sin interferir con la ejecución de la aplicación del dispositivo. RTDX ayuda a evaluar y analizar desde Matlab los algoritmos creados, y cambiar las características de operación acelerando el proceso de desarrollo y despliegue de sistemas de control embebido.[21]

La figura 3.28 muestra los bloques de transmisión (*To RTDX*) y recepción (*From RTDX*) de la herramienta RTDX disponibles en Simulink.

Figura 3.28 Bloques RTDX

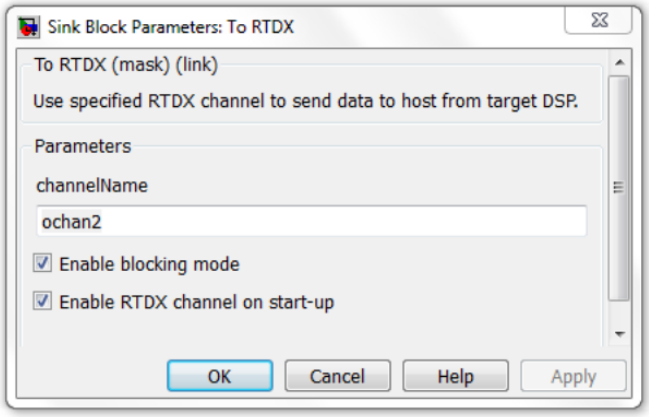


Fuente: El autor.

Cada bloque corresponde a un canal de comunicación que debe configurarse como los muestran las figuras 3.29 y 3.30. El bloque *To RTDX* realiza la transmisión de datos desde el DSP hacia el computador. Para la aplicación creada se utilizaron dos bloques (ochan1 y ochan2), uno para enviar la información de la velocidad del motor y otro para enviar el valor de la señal de control. Se habilitó la transmisión desde el inicio de la ejecución del programa en el DSP.

[21] THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012. Working with Texas InstrumentsTM Code Composer Studio 3.3 IDE.

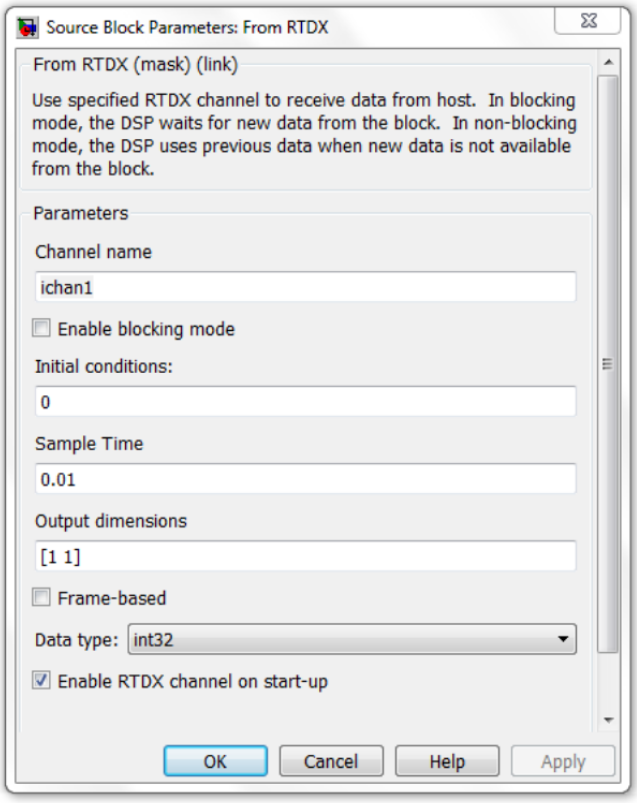
Figura 3.29 Configuración del bloque RTDX de transmisión.



Fuente: El autor.

Para la recepción del dato de la señal de referencia desde Matlab, se utilizó el canal ichan1 configurado en un bloque *From RTDX*. Se utilizó una velocidad de muestreo del canal de 0.01 segundos.

Figura 3.30 Bloque RTDX de recepción.



Fuente: El autor.

Los bloques configuran los canales RTDX en el DSP, pero para realizar la comunicación, en Matlab se debe realizar un procedimiento que incluye configurar y habilitar los canales, realizar la transmisión y finalmente cerrar el enlace y deshabilitar los canales.

3.2.6.1. Configuración y habilitación de canales RTDX en Matlab.

Para la configuración y habilitación de los canales primero debe crearse el enlace entre Matlab y CCS mediante el objeto de programación.

Se utilizaron las funciones de Matlab mostrados en la tabla 3.2:

Tabla 3.2 Funciones configurar y habilitar canales RTDX.

Función	Descripción	Ejemplo
Configure	Establece el número de buffers que van a almacenar los datos y el tamaño en bytes de cada buffer	IDE_Obj.rtdx.configure(1024,4); % define 4 buffers de 1024 bytes cada uno. IDE_Obj hace referencia al objeto.
Open	Abre un canal RTDX específico	IDE_Obj.rtdx.open('ochan1'); %Abre el canal ochan1.
enable	Habilita los canales abiertos dando inicio a la comunicación RTDX	IDE_Obj.rtdx.enable;

3.2.6.2. Transmisión de datos.

El envío de un dato desde el computador hacia el DSP puede realizarse en cualquier momento, sin embargo, la recepción solo se hace si hay un nuevo mensaje en los buffers. La tabla 3.3 muestra las funciones utilizadas para la transmisión.

Tabla 3.3 Funciones para la transmisión de datos.

Función	Descripción	Ejemplo
Msgcount	Devuelve el número de datos en los buffers que se encuentran en espera	IDE_Obj.rtdx.msgcount(‘ochan1’) %Devuelve el número de datos en espera del canal ‘ochan1’
Flush	Remueve el último dato de un canal específico para mantenerlo actualizado.	IDE_Obj.rtdx.flush(‘ochan1’,3); %Remueve los últimos 3 datos en el buffer del canal ochan1.
Readmsg	Lee uno o más mensajes de un canal.	IDE_Obj.rtdx.readmsg (‘ochan1’,’int32’); %Lee el dato recibido en el canal ochan1.
Writemsg	Escribe un dato en un canal.	IDE_Obj.rtdx.writemsg (‘ichan1’,16000); %Escribe el número 16000 en el canal ichan1.

3.2.6.3. Cierre de enlace e Inhabilitación de canales RTDX.

Para finalizar la ejecución, se inhabilitan los canales creados con la función *disable* de la misma forma como se utilizó la función *enable*.

3.3. IDENTIFICACIÓN DE LA PLANTA.

Cuando se desea controlar un sistema, antes se debe tener conocimiento de qué es lo que se va a controlar. Se debe contar con un modelo que describa el comportamiento de la planta para facilitar su análisis y el diseño de estrategias de control. [22] El modelo puede expresarse mediante las ecuaciones integro-diferenciales deducidas a través de las leyes físicas y químicas que rigen la naturaleza de la planta.

También se puede llegar a un modelo de la planta determinando su función de transferencia mediante una serie de mediciones rigurosas de las variables del sistema cuando éste es excitado con una o varias señales de entrada. Los datos de las mediciones permiten inferir un modelo aproximado haciendo uso de técnicas de identificación.

La obtención del modelo aproximado del motor se realizó utilizando la herramienta *ident* de Matlab a partir de los resultados obtenidos luego de aplicar una señal de excitación PWM del 50% de ciclo útil al motor. La planta con la que se trabajó en el proyecto se aproximó a un sistema de primer orden de la forma:

$$H(s) = \frac{K}{1 + T_p * s}$$

Donde: H(s) corresponde a la función de transferencia en el dominio de la frecuencia que describe el comportamiento del motor; K corresponde a la ganancia del sistema y Tp a la constante de tiempo.

3.3.1 Excitación de la planta.

La señal escalón utilizada para excitar el sistema fue construida desde el DSP mediante la implementación de código. Este código fue generado desde simulnik por a partir del modelo mostrado en la figura 3.4. En el modelo se agregaron bloques para leer la posición del codificador digital en el módulo eQEP1, realizar la conversión a velocidad y transmitirla al computador mediante el módulo de comunicación RTDX.

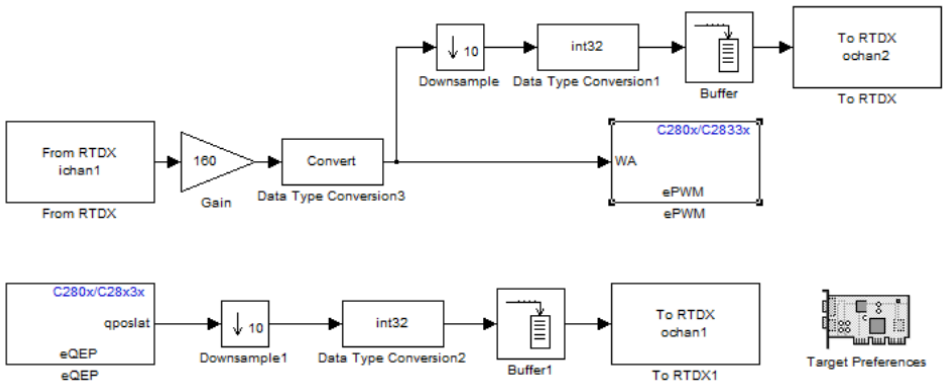
El usuario envía la orden de inicio a través del canal de entrada RTDX ichan1. El dato transmitido es un valor entre cero y cien correspondiente al porcentaje de

[22] MOSTERMAN Pieter, PRABHU Sameer, ERKKINEN Tom. An industrial embedded control system design process. Montreal 2004.

ciclo útil deseado en la señal PWM. Este dato es amplificado por el bloque de ganancia para llevarlo al puerto de entrada del registro CMPA del módulo PWM1A como un número entre 0 y 32000.

Los bloques *downsample y Buffer* se utilizaron para tomar muestras cada 0.1 segundos del ciclo útil y la velocidad, y transmitirlas en vectores de 50 datos vía RTDX.

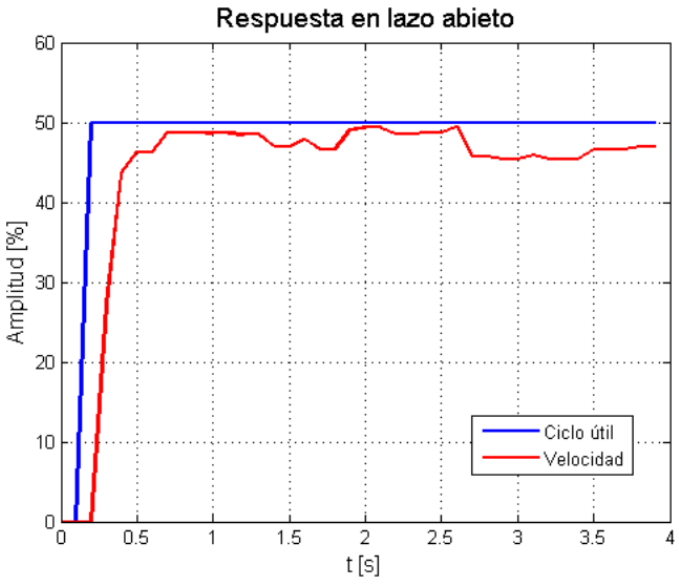
Tabla 3.4 Modelo contruído para generación de señal escalón y lectura de velocidad.



Fuente: El autor.

Para el ensayo, se prueba con una señal PWM del 50% de ciclo de trabajo obteniendo la siguiente respuesta de la velocidad del motor:

Figura 3.31 Señal PWM aplicada y velocidad obtenida.



Fuente: El autor.

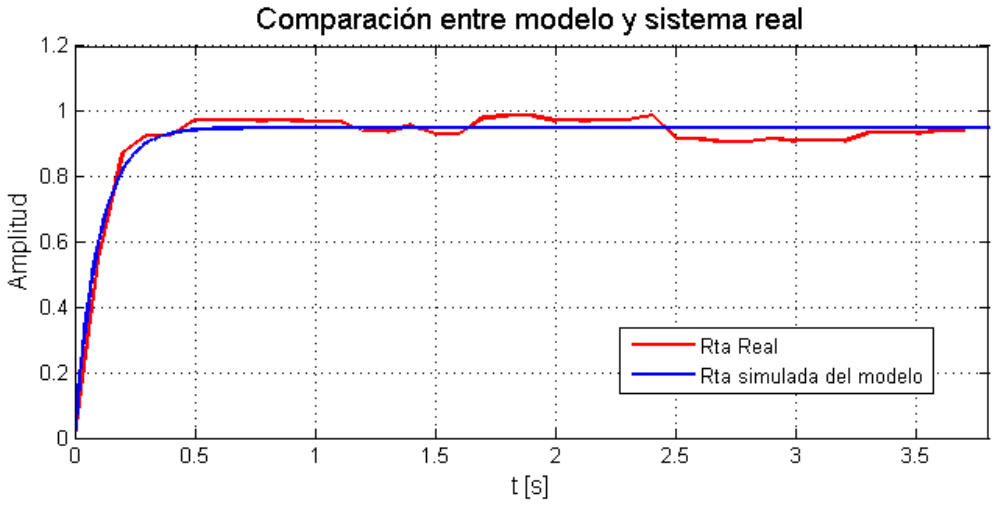
3.3.2 Obtención de función de transferencia.

Los datos obtenidos luego de aplicar la señal PWM del 50% de ciclo útil al motor, fueron llevados a la herramienta ident de Matlab para identificar la función de transferencia del motor aplicando el método de la curva de reacción. Se obtuvo el siguiente modelo aproximado con un valor de aproximación del 88.9%:

$$H(s) = \frac{0.95}{0.1\,s + 1} \qquad FIT = 88.9$$

El método utilizado es suficiente dada la complejidad de la planta y su comportamiento propio de un sistema LTI. Para validar el modelo obtenido, se comparó la respuesta en lazo abierto ante entrada escalón del sistema real con la respuesta que tendría el modelo aproximado como lo muestra la figura 3.32.

Figura 3.32 Comparación entre la respuesta del sistema real y el modelo obtenido.

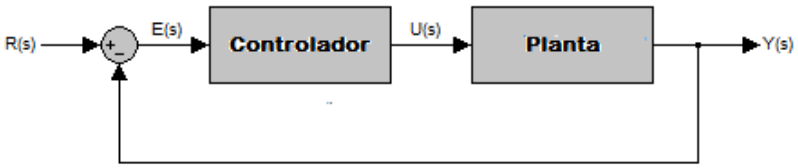


Fuente: El autor.

3.4. DISEÑO DE LA ESTRATEGIA DE CONTROL.

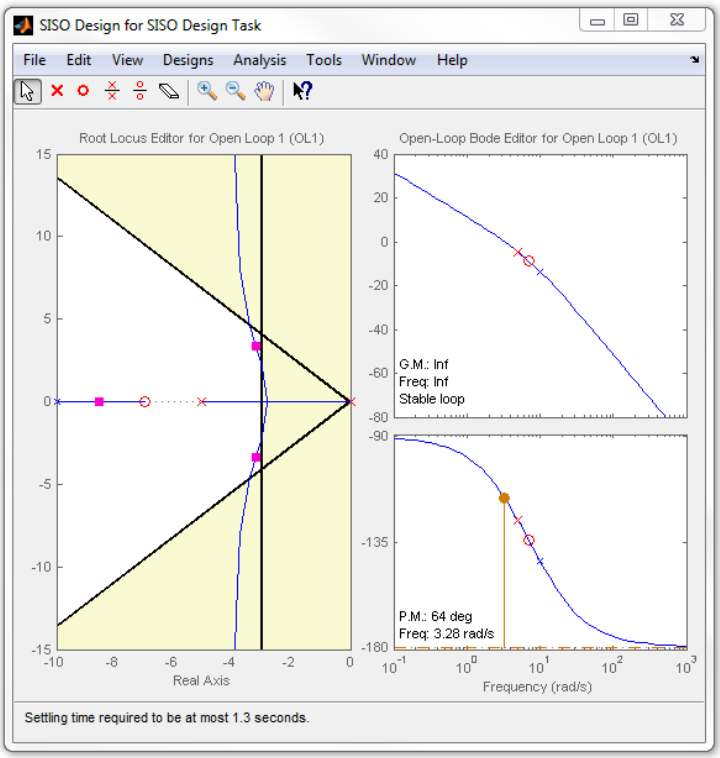
Luego de contar con un modelo aproximado de la planta, se utilizó la herramienta *sisotool* de Matlab para diseñar un controlador PID de lazo en la configuración mostrada por la figura 3.33.

Figura 3.33 Lazo de control implementado



Fuente: El autor.

Figura 3.34 Lugar de las raices y región deseada de los polos de lazo cerrado



Fuente: El autor.

Para disminuir el error de posición en la señal de salida se agregó una componente integral al controlador con un polo en el origen. También se

establecieron condiciones de estado transitorio de la respuesta del sistema como un sobre-impulso menor al 5% y un tiempo de establecimiento de 1.2 segundos. Este tiempo de establecimiento se fijó tres veces mayor al obtenido en lazo abierto con el fin de disminuir las corrientes en el inicio y generar un arranque más suave de la planta. La figura 3.34 muestra el lugar de las raíces y la ubicación de los polos de lazo cerrado para cumplir con los parámetros establecidos. El controlador obtenido es el siguiente:

$$C(s) = 3.92\ \frac{1 + 0.14s}{s\ (\ 1 + 0.2\ s\ )}$$

Las ganancias del controlador obtenido son:

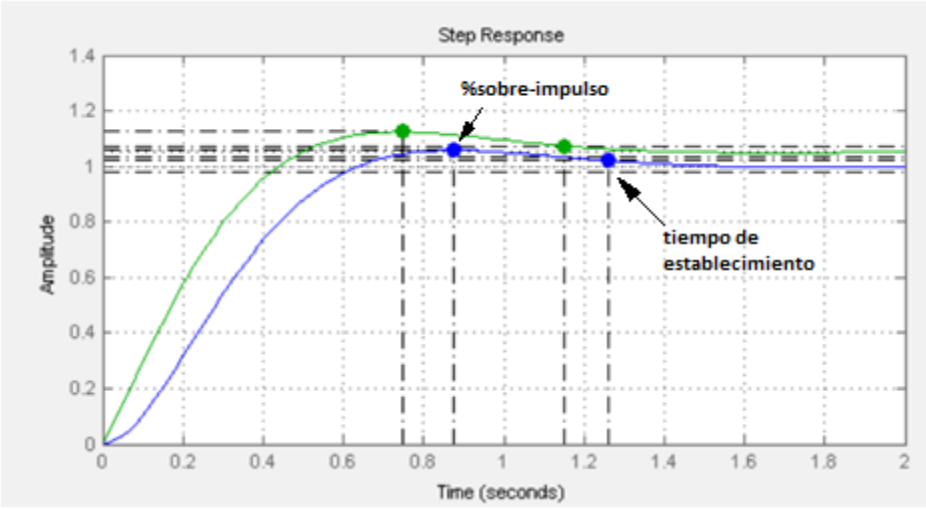
$$Kp = 0.2\ ;\ \ Ki = 3.9\ ;\ \ Kd = 0.47$$

3.4.1 Simulación.

En la figura 3.35 se muestra la simulación de la respuesta del sistema ante una entrada escalón unitario con el controlador PID en lazo cerrado. La línea en color azul representa el comportamiento de la señal de salida, mientras que la línea verde hace referencia a la señal de control (ciclo útil de la señal PWM).

Se espera que en el estado estacionario el error se elimine y que en el estado transitorio la señal se estabilice en un tiempo entre 1.2 y 1.3 segundos.

Figura 3.35 Simulación de la respuesta del sistema en lazo cerrado.



Fuente: El autor.

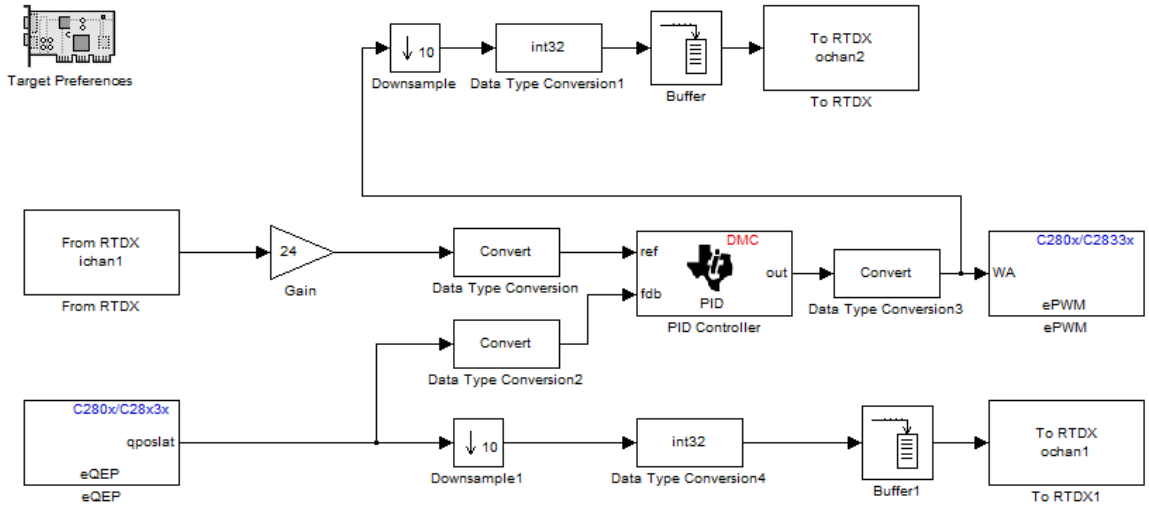
3.5. IMPLEMENTACIÓN Y VERIFICACIÓN.

Con el controlador diseñado, se construyó un modelo en Simulink para generar el código del controlador y la comunicación RTDX. En paralelo se diseñó una interfaz de usuario gráfica para la manipulación de la variable de referencia y la verificación del comportamiento del sistema embebido en tiempo real.

3.5.1 Construcción del modelo de Simulink.

El modelo de la figura 3.36 contiene la estrategia de control diseñada, así como los módulos y periféricos utilizados para la interacción del controlador con el usuario y la planta. El funcionamiento y configuración de los bloques principales se explicó anteriormente. Cabe aclarar que el nombre de los canales RTDX creados en el modelo, debe coincidir con el nombre de los mismos cuando se habilitan desde Matlab.

Figura 3.36 Modelo del controlador en Simulink.



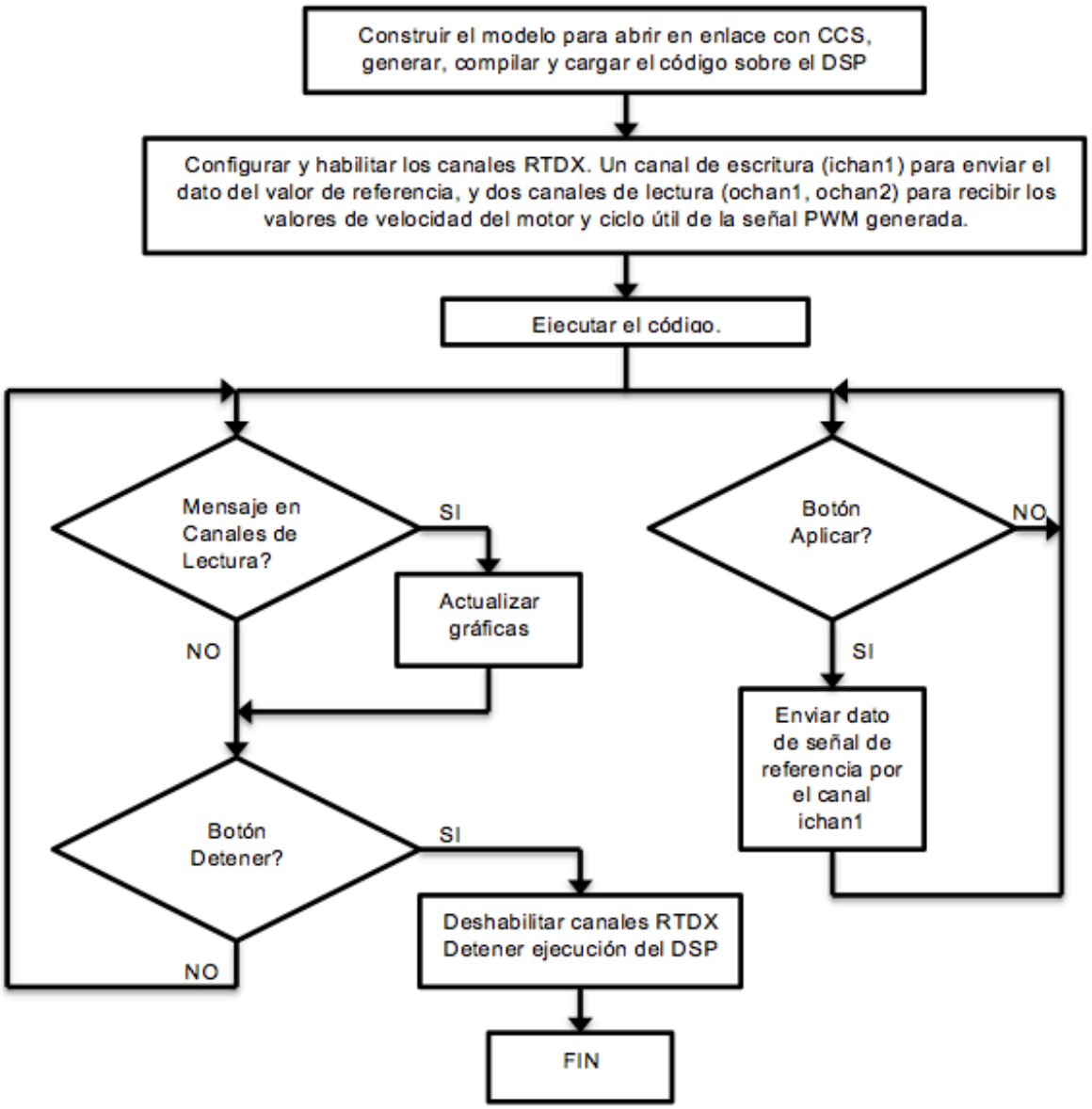
Fuente: El autor.

Con el modelo listo, se desarrolló una interfaz de usuario gráfica desde donde el modelo fue construido para generar el código e implementarlo finalmente sobre el TMS320F28335.

3.5.2 Implementación del código sobre la planta y creación de GUI.

La figura 3.37 muestra el diagrama de flujo de la interfaz gráfica de usuario construida con la herramienta GUIDE de Matlab para intercambiar datos entre Matlab y el DSP sin interferir con la ejecución del programa.

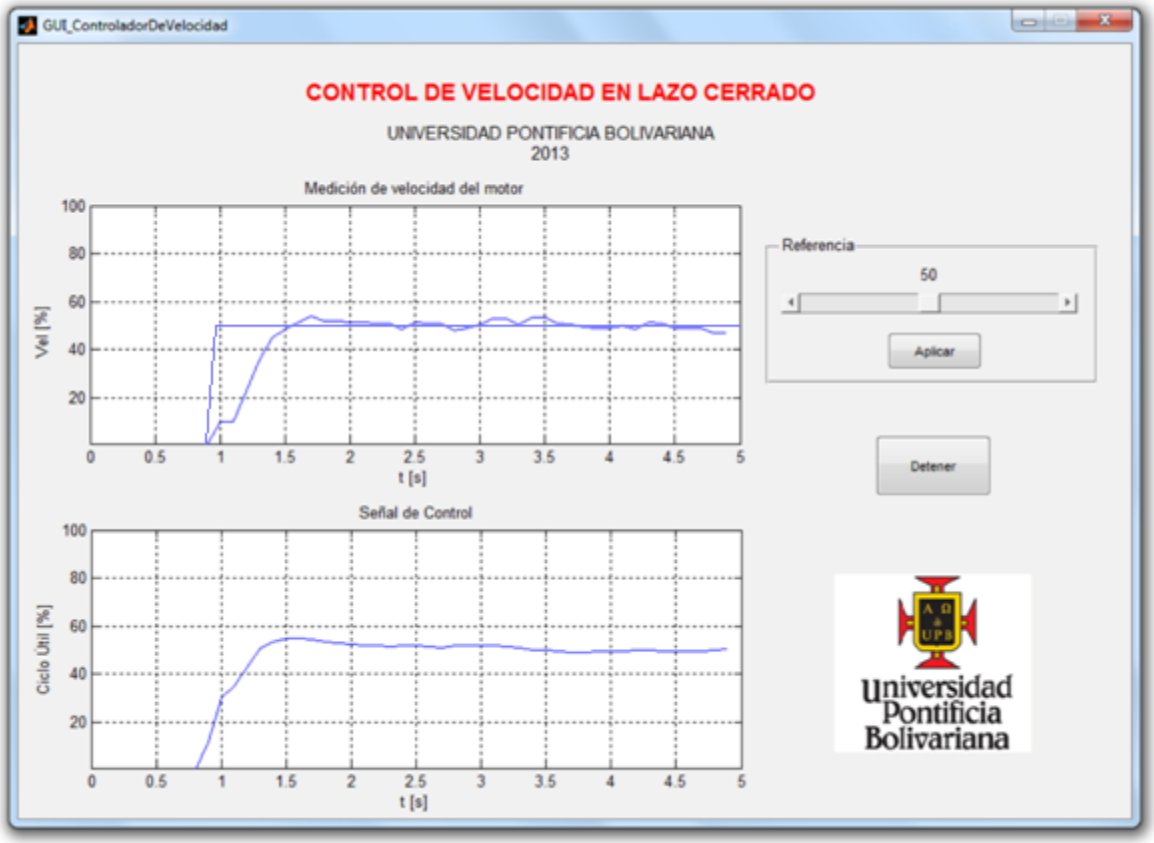
Figura 3.37 Algoritmo de la interfaz gráfica en GUIDE.



Fuente: El autor.

La interfaz construida contiene dos gráficas que muestran la información proveniente del DSP en los canales ochan1 y ochan2, es decir la velocidad del motor y el ciclo de trabajo de la señal de control. La interfaz también contiene un panel para actualizar el valor de referencia o valor deseado y un botón para detener la ejecución del programa como se muestra en la figura 3.38.

Figura 3.38 Interfaz gráfica de usuario.

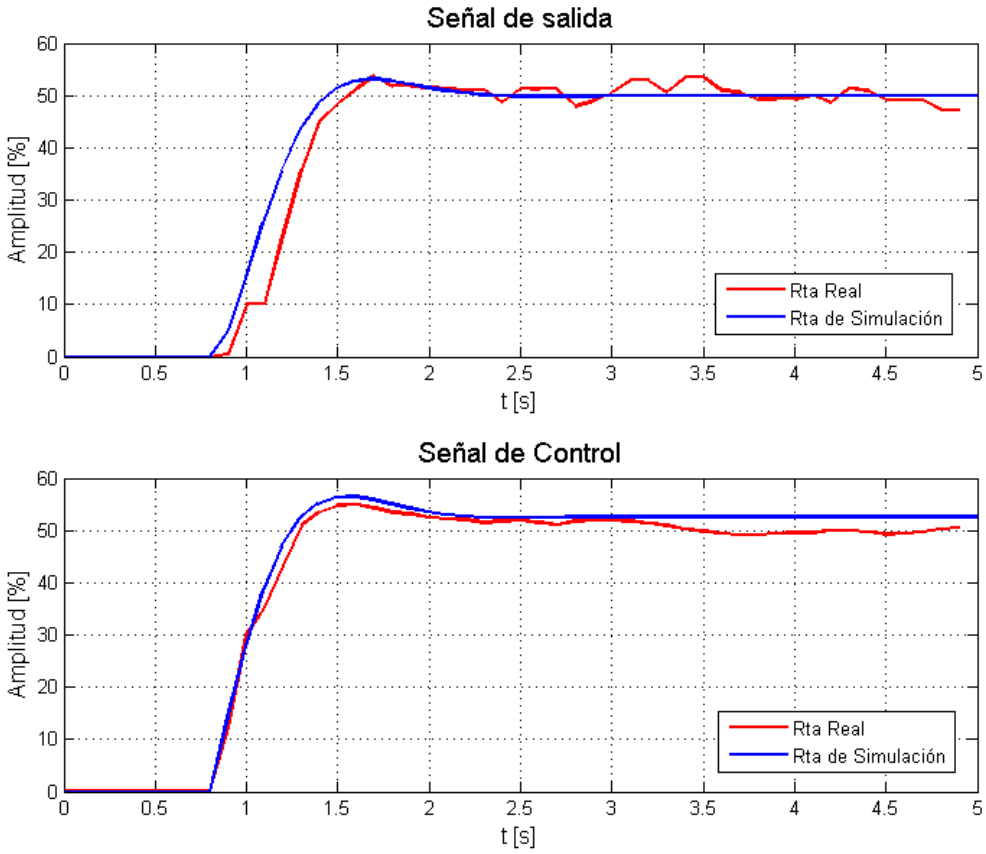


Fuente: El autor.

4. RESULTADOS

Después de generar código fuente e implementarlo sobre el DSP F28335, se realizó la conexión a la planta y se realizó una adquisición de datos con el fin de comparar los resultados obtenidos experimentalmente con los obtenidos en simulación.

Figura 4.1 Resultados rxperimentales y comparación con simulaciones.



Fuente: El autor.

Para poder realizar la comparación, se aplicó una señal de entrada escalón al sistema en reposo excitándolo; la entrada aplicada tiene un valor del 50%. La figura 4.1 muestra el contraste entre la respuesta real y la respuesta esperada en la simulación tanto para la variable de salida como para la señal de control.

Como se muestra en la figura 4.1a, la respuesta real tiene un comportamiento similar al esperado a partir de la simulación. El sobre-impulso es menor al 10% y el tiempo de estabilización de la señal de salida es de aproximadamente 1.3 segundos. Sin embargo, se observa un retardo en la respuesta real durante el ascenso de la señal de velocidad; esto puede deberse al retardo propio del motor que no se tuvo en cuenta para realizar la identificación de la función de transferencia del mismo.

Se observó que la respuesta de estado transitorio de la señal de control es muy parecido al esperado por la simulación, con desviaciones máximas del 8%. El valor final de dicha señal (Estado estable) pudo ser menor al esperado debido a fluctuaciones en la tensión de alimentación de la planta que no siempre es igual a 24 Volts exactos. Sin embargo, ante estas perturbaciones se encontró que el controlador mantiene la señal de velocidad en un valor cercano al valor de referencia (8% en estado permanente), reafirmando el efecto de la componente integradora del controlador en lazo cerrado al eliminar el error de posición del sistema.

5. RECOMENDACIONES

Como trabajo futuro, se plantea la posibilidad de utilizar el modo de simulación *Processor in the loop* de Simulink para simular el desempeño de los algoritmos de control mientras el DSP ejecuta el código en tiempo real pero sin conectarse a la planta de forma física; en este caso la planta es simulada en Simulink. De esta forma se podría reducir esfuerzos a la hora de probar, evaluar y validar los algoritmos. De igual forma se propone utilizar el modo *Hardware in the loop* para visualizar, verificar y validar un modelo con los elementos hardware conectados al DSP.

El trabajo realizado puede servir como punto de partida para la implementación del controlador de un sistema de conversión de potencia con el IGBT de Semikron propiedad de la Universidad Pontificia Bolivariana.

6. CONCLUSIONES

Haciendo uso de herramientas de prototipado rápido proporcionadas por Embedded Coder de Simulink, se logró desarrollar un instrumento software confiable, económico, rápido y poco complejo, que permite verificar el desempeño de un sistema de control diseñado antes de llevarlo a su implementación final.

Mediante el uso de técnicas de prototipado rápido, intercambio de datos en tiempo real RTDX y la herramienta *ident* de Matlab, fue posible realizar la identificación del modelo de un sistema de primer orden que describe de forma aproximada el comportamiento de la planta utilizada.

Se diseñó e implementó un controlador PID digital de velocidad para un motor DC que permite mejorar las respuestas de estado estacionario y transitorio, dando robustez al sistema ante perturbaciones, reduciendo el error de posición y permitiendo un arranque suave del motor.

Mediante el intercambio de datos en tiempo real RTDX de Texas Instruments y la herramienta GUIDE de Matlab, se logró construir una interfaz de usuario gráfica que permitió realizar el análisis y la verificación del controlador implementado para comparar los resultados experimentales con los esperados a partir de simulaciones y ajustar los parámetros del controlador hasta obtener un comportamiento deseado.

Gracias a las herramientas de generación automática de código utilizadas, se presentó una alternativa rápida al diseño e implementación de sistemas de control, que permite al ingeniero centrar la atención en el planteamiento de las estrategias de control y no en la escritura de código para el microprocesador donde se ejecutarán los algoritmos de control.

Con el presente trabajo de investigación se logró reafirmar la validez de las herramientas de prototipado rápido existentes como alternativas simples y efectivas dentro del campo de la implementación del control automático de procesos.

7. BIBLIOGRAFÍA.

- CHEN M., GONG L., ZHOU X. An Economical Rapid Control Prototyping System Design with Matlab/Simulink and TMS320F2812 DSP. China 2009.
- DUMA R., DOBRA P. Rapid prototyping of control systems using Embedded Target for TI C2000 DSP. Romania 2007.
- GALEANO Gustavo. Programación de sistemas embebidos en C. Editorial Alfaomega Colombiana S.A. 2009
- HERCOG Darko, JEZERNIK Karel. Rapid Control Prototyping using MATLAB/Simulink and a DSP-based motor controller. Universidad deMaribor. Eslovenia 2005.
- HERCOG Darko, ROJKO Andreja, ČURKOVIČ Milan, JEZERNIK Karel. Embedded platform for rapid implementation of local and remote motion control experiments. University of Maribor. Eslovenia.
- HRISTU-VARSAKELIS Dimitrios, LEVINE William S. Handbook of Networked and Embedded Control Systems. Birkäuser Boston.
- MATHWORKS. Rapid Prototyping. [En linea] http://www.mathworks.com/rapid-prototyping/ [citado en 2012].
- MATLAB AND SIMULINK. Deploy Embedded Code onto TI C2000. USA 2007.
- OGATA Katsuhiko. Ingeniería de control moderna. Cuarta edición. Pearson Educación S.A. Madrid 2003.
- Rebeschieß S., "MIRCOS - microcontroller-based real time control system toolbox for use with Matlab/Simulink," in *Proc. IEEE International Symposium on Computer-Aided Control System Design*, 1999, 267-272.
- TEXAS INSTRUMENTS. Code Composer Studio Development Tools v3.3. Getting Started Guide. 2007.
- NORIEGA Jairo. Diseño e implementación de un controlador PID digital en un micro-controlador utilizando técnicas de prototipado rápido. Trabajo de grado. Universidad Pontificia Bolivariana. Bucaramanga. 2011.
- BARR Michael, MASSA Anthony. Programming Embedded Systems. Editorial O'Reilly. Segunda edición. EEUU 2006.
- MOSTERMAN Pieter, PRABHU Sameer, ERKKINEN Tom. An industrial embedded control system design process. Montreal 2004.
- DINIZ Paulo, DA SILVA Eduardo, NETTO Sergio. Digital signal processing. System Analysis and Design. 2 ed. Cambridge University. 2010. p. 2.

- SALAZAR Jordi. Procesadores Digitales de Señal. Universidad Politécnica de Cataluña. España. [en línea] http://arantxa.ii.uam.es/~taao1/teoria/tema1/pdf/Procesadores_dig.pdf. [citado en 2012].
- BORMANN Frank, Introduction to TMS320F28335. Texas Instruments Inc. 2011.
- TEXAS INSTRUMENTS. TMS320F28x Digital Signal Controller – Manual de Datos. 2007. [Disponible en linea] http://www.ti.com/lit/ds/sprs439m/sprs439m.pdf.
- IEEE STANDARDS ASSOCIATION. 1149.1-2001 IEEE Standard Test Acces Port and Boundary Scan Architecture. [en línea] http://standards.ieee.org/findstds/standard/1149.1-2001.html [citado en 2012].
- TEXAS INSTRUMENTS. TMS320x2833x Enhanced Pulse Width Modulator (ePWM) Module - Guía de Referencia. 2008. [Disponible en línea] http://www.ti.com/lit/ug/sprug04a/sprug04a.pdf
- TEXAS INSTRUMENTS. TMS320x2833x High Resolution Pulse Width Modulator (HRPWM). Guía de referencia. 2009. [disponible en línea] http://www.ti.com/lit/ug/sprug02b/sprug02b.pdf
- TEXAS INSTRUMENTS. TMS320x2833x Enhaced Quadrature Encoder Pulse (eQEP) module. Guia de referencia. 2008. [disponible en linea] http://www.ti.com/lit/ug/sprug05a/sprug05a.pdf
- THE MATHWORKS INC. Embedded Coder – Matlab & Simulink R2012b. Guía de Usuario. 2012.
- TEXAS INSTRUMENTS. TMS320C2000 Experimenter Kit Overview. Guía de inicio rápido. 2009.
- MATHWORKS. Simulink Coder – Target Language Compiler. Guía de Usuario. 2012.

ANEXO 1. CÓDIGO IMPLEMENTADO PARA LA GUI.

```
function varargout = ControlVelLazoCerrado(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ControlVelLazoCerrado_OpeningFcn,
...
                   'gui_OutputFcn',  @ControlVelLazoCerrado_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

function ControlVelLazoCerrado_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
guidata(hObject, handles);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global CCS_Obj;
try
    Modelo = gcs; %Carga el nombre del modelo abierto
    disp('### Connecting to Code Composer Studio ...');
    CCS_Obj = connectToBoard(Modelo); %Se conecta a la tarjeta y el DSP
    CCS_Obj.visible(1);
    rtwGenSettings = ccslink_getRtwGenSettings;
    UbicacionModelo = fullfile('.', [Modelo
rtwGenSettings.BuildDirSuffix], Modelo); % full pathname

    if (~need2rebuild(Modelo))
        saveState = warning;
        warning off
        projectName = [UbicacionModelo '.pjt'];
        warning(saveState);
    else
        load_system(Modelo);
        make_rtw;
        saveModelState(Modelo);
    end

    % Cambiar al directorio del archivo del modelo para guardar archivos
    % generados
    CCS_Obj.cd(fullfile(pwd,[Modelo rtwGenSettings.BuildDirSuffix]));
```

```
    outFile = [Modelo '.out'];

    if exist(UbicacionModelo '.out'])
        fprintf('### Loading COFF file to target DSP...\n');
        try
            CCS_Obj.reset;
            CCS_Obj.load(outFile,100);
        catch
            clear CCS_Obj;
            msg = lf_message('demos:runDemoLoadError');
            errordlg(msg, 'Error');
            return
        end
        try
            % configuracion de canales RTDX
            r = CCS_Obj.rtdx;
            r.disable;
            r.configure(64000,4,'continuous');
            r.open('ochan1', 'r');
            r.open('ochan2', 'r');
            r.open('ichan1','w');
            r.enable;

            CCS_Obj.run;
        catch ME
            errordlg(ME.message, 'Error');
            return
        end
    else
        msg = lf_message('demos:runDemoReBuildError');
        errordlg(msg, 'Error' , 'modal');
        return
    end
catch ME
    errordlg(ME.message);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
set(handles.figure1,'visible','on');
axes(handles.axes1);
title('Medición de velocidad del motor');
xlabel('t [s]');
ylabel('Vel [%]');
grid on;
axis([0 5 1 100]);
tiempo=[0:0.1:4.9]; %El tiempo de muestreo del eQEP se configuró a 0.01
con un downsample de factor 10
axes(handles.axes2);
title('Señal de Control');
xlabel('t [s]');
ylabel('Ciclo Útil [%]');
grid on;
axis([0 5 1 100]);
tiempo=[0:0.1:4.9];
```

```
while (isrunning(CCS_Obj))
    %Lectura del canal ochan1  RTDX
    numMsgsOchan1 = r.msgcount('ochan1');
    if (numMsgsOchan1 > 1),
        % flush frames as necessary to maintain real-time display
        r.flush('ochan1', numMsgsOchan1-1);
    end
    if (numMsgsOchan1)
        Velocidad = r.readmsg('ochan1', 'int32');
    end
    %Lectura del canal ochan2 RTDX
    numMsgsOchan2 = r.msgcount('ochan2');
        if (numMsgsOchan2 > 1),
            % flush frames as necessary to maintain real-time display
            r.flush('ochan2', numMsgsOchan2-1);
        end
        if (numMsgsOchan2)
            pwm = r.readmsg('ochan2', 'int32');
        end
    %Graficas de velocidad y PWM si hay mensajes a la entrada
    if ((numMsgsOchan1 ~=0) && (numMsgsOchan2 ~= 0))
        axes(handles.axes1);
        PorcentajeVelocidad = (double(Velocidad)./120);
        plot(handles.axes1,tiempo,PorcentajeVelocidad)
        title('Medición de velocidad del motor');
        xlabel('t [s]');
        ylabel('Vel [%]');
        grid on;
        axis([0 5 1 100]);

        axes(handles.axes2);
        ciclo = (double(pwm)./320);
        plot(handles.axes2,tiempo, ciclo);
        title('Señal de Control');
        xlabel('t [s]');
        ylabel('Ciclo Útil [%]');
        grid on;
        axis([0 5 1 100]);
    end

    if(get(handles.Detener,'userdata')==1)
        save('RtaEscalon.mat','PorcentajeVelocidad','cycle','tiempo')
        break
    end
    pause (0.2);
end

disp('Fin del proceso')
%Inhabilitar canales RTDX
try
    r.disable('ichan1');
    r.disable('ochan1');
    r.disable('ochan2');
    r.disable;
```

```
catch
    % Si los canales no están abiertos, no se ejecuta nada
end
%Pausar y detener enlace con CCS
halt(CCS_Obj);
CCS_Obj.reset;
if ishandle(handles.figure1)
    %close(handles.figure1)
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function varargout = ControlVelLazoCerrado_OutputFcn(hObject, eventdata,
handles)
if isfield(handles, 'output')
    varargout{1} = handles.output;
end

function Detener_Callback(hObject, eventdata, handles)
set(handles.Detener,'userdata',1);

function slider1_Callback(hObject, eventdata, handles)
Set_Point=ceil(get(handles.slider1,'value')*100); %El 64000 debe
cambiarse por la velocidad máxima en RPM del motor.
texto1=num2str(Set_Point);
set(handles.text1, 'string', texto1);


function slider1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


function Aplicar_Callback(hObject, eventdata, handles) %Boton apicar,
enviar dato.
global CCS_Obj;
r = CCS_Obj.rtdx;
Set_Point=ceil(get(handles.slider1,'value')*100); %El 64000 debe
cambiarse por la velocidad máxima en RPM del motor.
r.writemsg('ichan1',int32(Set_Point));
```