INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

"DISEÑO DE UN SISTEMA PARA LA OBTENCIÓN DE PARÁMETROS DE UN GIROSCOPIO MICROELECTROMECÁNICO"

PROYECTO TERMINAL

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMUNICACIONES Y ELECTRÓNICA

P R E S E N T A N

LUIS SÁNCHEZ MÁRQUEZ

RICARDO FLORES MARTÍNEZ

ASESOR:

M. en C. EDUARDO GABRIEL BALDERAS

MÉXICO D.F. JULIO DE 2015

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA
UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

TEMA DE TESIS

QUE PARA OBTENER EL TITULO DE **INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**
POR LA OPCIÓN DE TITULACIÓN **TESIS COLECTIVA Y EXAMEN ORAL INDIVIDUAL**
DEBERÁ (N) DESARROLLAR **C. LUIS SÁNCHEZ MÁRQUEZ**
C. RICARDO FLORES MARTÍNEZ

"DISEÑO DE UN SISTEMA PARA LA OBTENCIÓN DE PARÁMETROS DE UN GIROSCOPIO MICROELECTROMECÁNICO"

DISEÑAR Y CONSTRUIR UN SISTEMA PARA LA OBTENCIÓN DE LOS PARÁMETROS DE UN GIROSCOPIO MICROELECTROMECÁNICO.

- **INTRODUCCIÓN**
- **ANÁLISIS Y DISEÑO**
- **PRUEBAS Y RESULTADOS**
- **CONCLUSIONES Y RECOMENDACIONES**

CIUDAD DE MÉXICO A 4 DE MAYO DE 2016

A S E S O R E S

M. EN I. EDUARDO GABRIEL BALDERAS

ING. PATRICIA LORENA RAMÍREZ RANGEL
JEFE DEL DEPARTAMENTO DE
INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

Dedico esta tesis a cada una de las personas que conocí en el trayecto de mi carrera, que sin ellas no podría haber llegado a donde estoy.

R.F.M

A mi madre.

L.S.M

Agradecimientos

Le agradezco a mis padres por haberme apoyado y brindado todo lo que necesité en este camino, por ser quienes me dieron fortaleza en los momentos frágiles de mi vida y me aconsejaron día a día en cada problema que se me presentaba.

A mi madre, por haberse esforzado tanto para sacarme adelante los primeros años de mi vida, sobre todo por su paciencia y buenos consejos que siempre me ha brindado.

A mi padre, por depositar su confianza en mí, por enseñarme a decidir sobre los aspectos más importantes de la vida y sobre todo por estar acompañándome durante la carrera.

A mis hermanos, por brindarme momentos de felicidad cuando lo necesite, por apoyarme siempre buscando que viera el lado bueno de las cosas y saber que todo tiene solución.

A mi compañero Luis, por salvar mis estudios en un momento crítico, por volverse mi confidente y por brindarme sus conocimientos en todo momento que los necesite.

A mi primo Tonhatiu, por hacerme compañía en mis horas de estudio y brindarme su ayuda siempre que estuvo en sus manos.

A toda mi familia que en todo momento han sido mi respaldo, mi fortaleza y mi modelo a seguir, sin ellos no podría haber llegado a este momento.

A todos mis amigos, que me brindaron su ayuda, por enseñarme a ser una persona perseverante y seguir superándome, por acompañarme en los momentos malos y buenos, sobre todo por mostrarme lo valioso de la amistad.

Ricardo Flores Martínez

Quiero agradecer principalmente a mis padres Raúl y María Magdalena por el apoyo brindado en el transcurso de mi vida.

A todos mis profesores por haberme compartido un poco de sus conocimientos. A nuestro asesor de tesis, M. en C. Eduardo Gabriel Balderas, por las revisiones y consejos. Al Dr. Alfredo Reyes Barranca por la asignación del tema de tesis.

A todos mis compañeros y amigos de la ESIME, en especial a Ismael Cristhian Mociño Arellano, Gabriel Aguilar Pérez y Luis Armando Hernández Hernández, sin ellos habría sido menos sencillo terminar esta etapa de mi vida. A Ricardo Flores Martínez, compañero de tesis, por permitirme trabajar una vez más a su lado.

Al profesor Luis Martín Flores Nava por brindarme su apoyo, conocimientos y consejos en estos últimos años.

A mi amigo Luis Aldo López Quijano por acompañarme y apoyarme en los seis meses más difíciles que he vivido como estudiante.

A mis hermanos Hugo, Raúl y Alexandro y a mis amigos Adan Rosendo Vera Gutiérrez, José Daniel Guerrero Guadarrama y Luis Fernando Nolasco Ramírez, gracias por acompañarme en la vida.

Luis Sánchez Márquez

CONTENIDO

Índice de figuras

Índice de tablas

Resumen

En este proyecto se muestra el diseño y construcción de un sistema para obtener los parámetros de un giroscopio (1. Sensitividad: indica la cantidad de mili volts entregados por el giroscopio con el cambio de velocidad, 2.Voltaje proporcionado por el giroscopio cuando se encuentra en reposo) con el que se gráfica y visualiza dicha información en una PC.

En el primer capítulo se presenta una introducción de lo que consiste la tecnología de giroscopios MEMS, así como de los FPGAs de Xilinx y el lenguaje VHDL (utilizado para programar al FPGA).

En el segundo capítulo, se muestra el análisis y diseño de todos y cada uno de los bloques que componen el proyecto, para formar así el sistema de obtención de parámetros de un giroscopio microelectromecánico al cual se le asigna el nombre Gyro-Tech.

En el tercer capítulo se presentan todas las pruebas y resultados obtenidos en el diseño y construcción de cada bloque que compone al proyecto terminal, para poder así concluir con algunos comentarios y observaciones para los ingenieros que tengan la oportunidad de seguir trabajando y desarrollando un proyecto similar en un futuro.

CAPÍTULO 1

Introducción

1.1 Giroscopio MEMS

Un giroscopio es un dispositivo que se utiliza para la navegación y la medición de la velocidad angular. Los giroscopios están disponibles para poder medir la velocidad de rotación en 1, 2, o 3 direcciones.

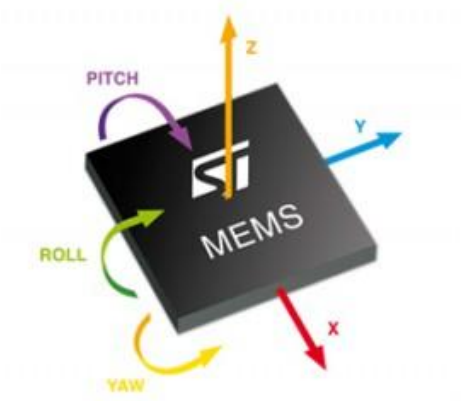


Figura 1.1.1: Giroscopio de estructura vibrante.

Los giroscopios de estructura vibrante son dispositivos MEMS (Micro-Machined Electro-Mechanical Systems o Sistemas Micro Electro Mecánicos),figura 1.1.1, que están disponibles en el mercado, accesibles y de tamaño pequeño. Para la comprensión de la operación de un giroscopio de estructura vibrante es conveniente la comprensión del fenómeno conocido como coriolis. En un sistema de rotación, cada punto gira con la misma velocidad. A medida que se acerca al eje de rotación del sistema, la velocidad de rotación

(velocidad angular) sigue siendo la misma, pero la velocidad en la dirección perpendicular al eje de rotación disminuye (velocidad tangencial). Por lo tanto, con el fin de viajar en línea recta hacia o desde el eje de rotación mientras se está en un sistema en rotación, la velocidad tangencial debe ser aumentada o disminuida con el fin de mantener la misma posición angular relativa (longitud) en el cuerpo. El acto de ralentización o de aumento de la velocidad es aceleración y la que actúa es la fuerza de coriolis en la aceleración de la masa del objeto, cuya longitud se debe mantener. La fuerza de coriolis es proporcional tanto a la velocidad angular del objeto en rotación como a la velocidad del objeto en movimiento hacia o desde el eje de rotación.

Los giroscopios de estructura vibrante contienen una masa micro-mecanizada que está conectada a un alojamiento exterior por un conjunto de resortes. Esta carcasa exterior está conectada a la placa de circuito fijo por un segundo conjunto de resortes ortogonales, figura 1.1.2.

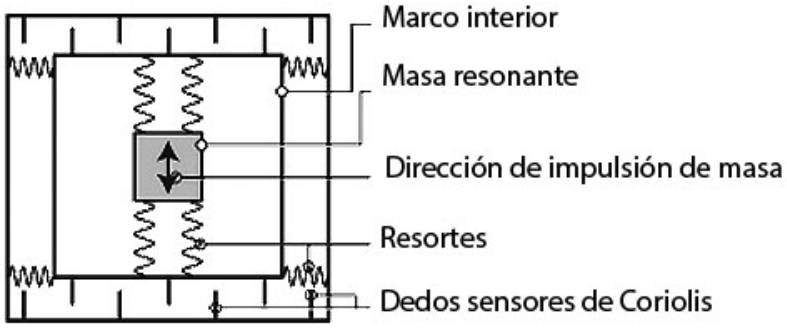


Figura 1.1.2: Partes internas de un Giroscopio de Estructura Vibrante.

La masa es excitada en forma sinusoidal continuamente a lo largo de la primera serie de resortes. Cualquier rotación del sistema inducirá una aceleración de coriolis en la masa, empujándola en la dirección de la segunda serie de resortes. A medida que la masa es impulsada alejándose del eje de rotación, la masa será empujada perpendicularmente en una dirección, y a medida que es impulsada de nuevo hacia el eje de rotación, esta será empujada en la dirección opuesta debido a la fuerza de Coriolis que actúa sobre la misma, figura 1.1.3.

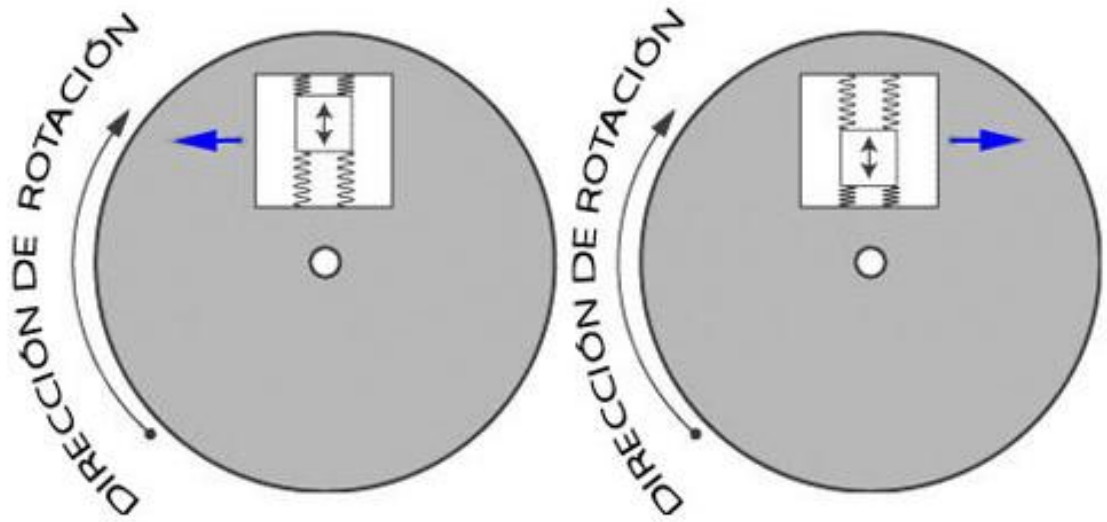


Figura 1.1.3: Toma de muestra de un cambio de rotación.

La fuerza de coriolis es detectada por los dedos sensores capacitivos que se encuentran a lo largo de la carcasa de la masa y la estructura rígida. Como la masa es empujada por la fuerza de Coriolis, un diferencial de capacitancia será detectado a medida que los dedos de detección se aproximan entre sí. Cuando la masa se empuja en la dirección opuesta, los diferentes conjuntos de dedos sensores se aproximan entre sí, por lo que el sensor puede detectar tanto la magnitud y dirección de la velocidad angular del sistema, figura 1.1.4.

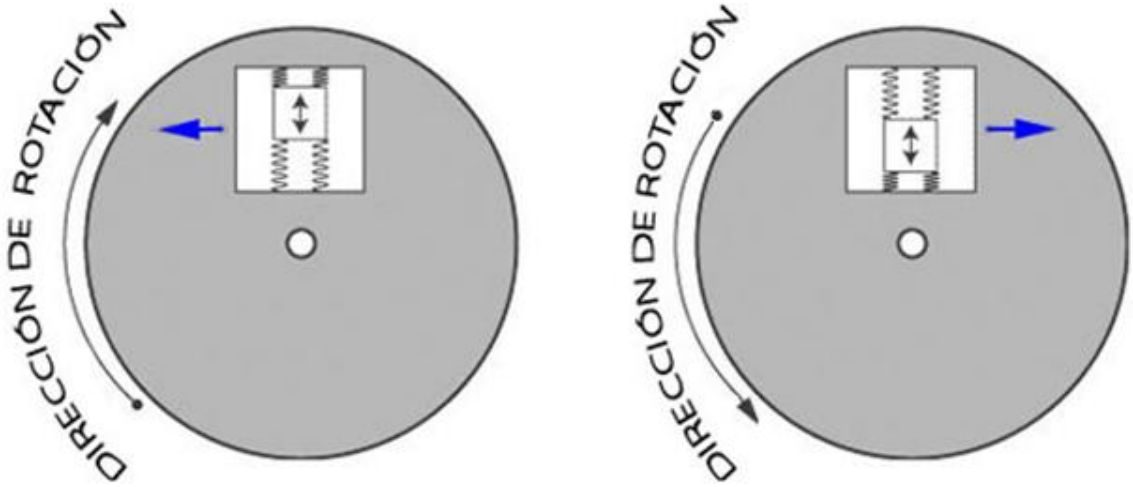


Figura 1.1.4: Magnitud y dirección de velocidad angular.

A continuación se muestra una imagen de un Giroscopio MEMS, visto a través de un microscopio, figura 1.1.5.

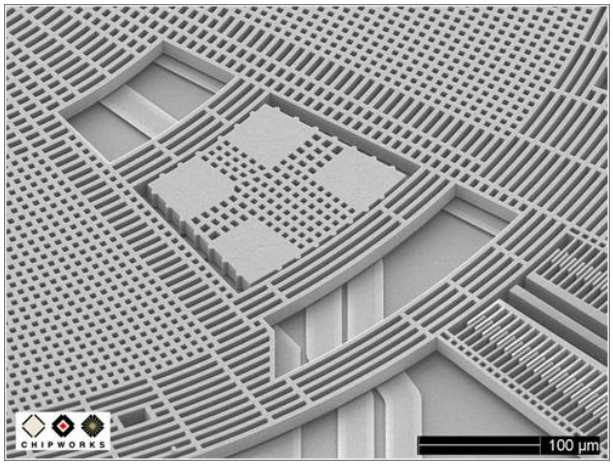


Figura 1.1.5: Un giroscopio MEMS visto a través de un microscopio.

1.2 FPGAs

Los FPGA (Field Programmable Gate Array) son circuitos lógicos programables directamente por el usuario, lo cual requiere de herramientas de costo relativamente bajo, como lo son el software de desarrollo y el dispositivo grabador. La grabación o programación de uno de estos dispositivos se puede llevar a cabo en milisegundos.

Los FPGA son muy utilizados por fabricantes que producen tecnología a baja escala, como por ejemplo diseñadores de equipos de propósito específico, los cuales no pueden justificar la producción de ASICs por los bajos volúmenes de dispositivos que venden.

Arquitectura general de un FPGA

Un FPGA consiste en arreglos de varios bloques programables (bloques lógicos) los cuales están interconectados entre sí y con celdas de entrada/salida mediante canales de conexión verticales y horizontales, tal como muestra la figura 1.2.1.

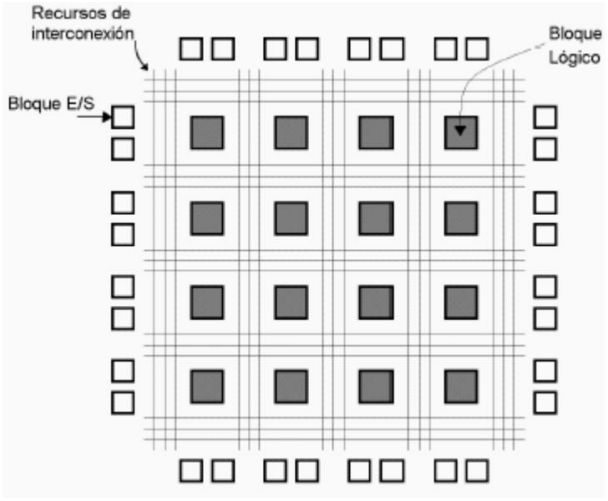


Figura 1.2.1: Arquitectura básica de un FPGA

Bloques Lógicos

El bloque lógico consta de una parte combinacional, que permite implementar funciones lógicas booleanas, más una parte secuencial que permite sincronizar la salida con una señal de reloj externa e implementar registros.

- Bloque lógico basado en LUT (look-up table): Una LUT es un componente de células de memoria SRAM que almacena una tabla de verdad. Las direcciones de las células son las entradas de la función lógica que queremos implementar, y en cada celda de memoria se guardan el resultado para una de las combinaciones de las entradas. En una LUT de n x 1 es posible implementar cualquier función lógica de n entradas.

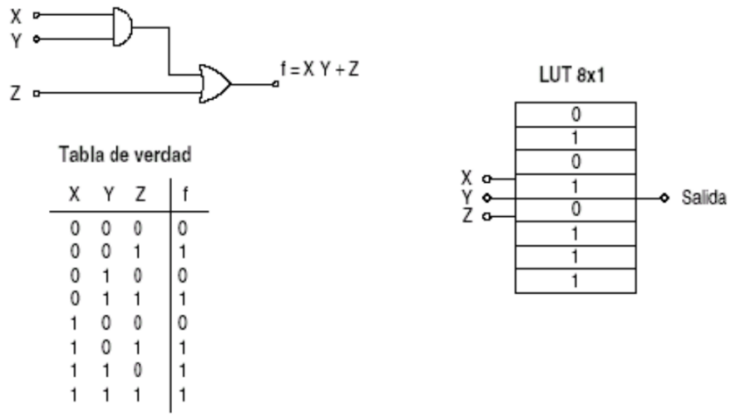


Figura 1.2.2: Ejemplo de implementación de una función lógica de tres entradas en una LUT de 8x1

Interconexión entre bloques programables.

Además de los bloques programables también es importante la tecnología utilizada para crear las conexiones entre los canales (tecnología de programación).

- SRAM (StaticRAM): Estas guardan la configuración del circuito. Esto quiere decir que las SRAM son utilizadas como generadores de funciones y además son usadas para controlar multiplexores (que están incluidos en los FPGAs) y la interconexión entre bloques. En éstas el contenido se almacena mediante un proceso de configuración en el momento de encendido del circuito que contiene al FPGA. Ya que al ser SRAM, el contenido de la memoria se pierde cuando se deja de suministrar energía; la información binaria de las celdas SRAM generalmente se almacena en memorias seriales EEPROM conocidas como memorias de configuración o celdas de configuración. En el momento de encendido del circuito toda la información binaria es transferida a los bloques e interconexiones del FPGA mediante el proceso de configuración el cual es generalmente automático, dado que el propio FPGA contiene un circuito interno que se encarga de hacer toda la programación.

Bloques de entrada/salida (IOB).

La función de un bloque de entrada/salida es permitir el paso de una señal hacia dentro o hacia el exterior del dispositivo. Además debe contar con recursos tales como:

• Salidas configurables como TRI-STATE u open-collector.
• Entradas con posibilidad de pull-up o pull-down programables.
• Registros de salida.
• Registros de entrada.

Es decir, estos bloques conectan el circuito con el exterior, ofrecen salidas de tres estados y sirven para almacenar temporalmente las señales de entrada y salida.

1.3 VHDL

VHDL, proviene del acrónimo VHSIC (Very High Speed Integrated Circuit) Hardware Description Language. VHDL es un lenguaje de descripción y modelado diseñado para describir (en una forma que los humanos y las máquinas puedan leer y entender) la funcionalidad y la organización de sistemas hardware digitales, placas de circuitos, y componentes. VHDL fue desarrollado como un lenguaje para

el modelado y simulación lógica dirigida por eventos de sistemas digitales, y actualmente se lo utiliza también para la síntesis automática de circuitos.

El netlist

El netlist es la primera forma de describir un circuito mediante un lenguaje, y consiste en dar una lista de componentes, sus interconexiones y las entradas y salidas. No es un lenguaje de alto nivel por lo que no describe cómo funciona el circuito sino que simplemente se limita a describir los componentes que posee y las conexiones entre ellos.

VHDL describe estructura y comportamiento

Existen dos formas de describir un circuito. Por un lado se puede describir un circuito indicando los diferentes componentes que lo forman y su interconexión, de esta manera tenemos especificado un circuito y sabemos cómo funciona; esta es la forma habitual en que se han venido describiendo circuitos y las herramientas utilizadas para ello han sido las de captura de esquemas y las descripciones netlist.

La segunda forma consiste en describir un circuito indicando lo que hace o cómo funciona, es decir, describiendo su comportamiento. Naturalmente esta forma de describir un circuito es mucho mejor para un diseñador puesto que lo que realmente lo que interesa es el funcionamiento del circuito más que sus componentes. Por otro lado, al encontrarse lejos de lo que un circuito es realmente puede plantear algunos problemas a la hora de realizar un circuito a partir de la descripción de su comportamiento.

El VHDL va a ser interesante puesto que va a permitir los dos tipos de descripciones:

Estructura: VHDL puede ser usado como un lenguaje de Netlist normal y corriente donde se especifican por un lado los componentes del sistema y por otro sus interconexiones.

Comportamiento: VHDL también se puede utilizar para la descripción comportamental o funcional de un circuito. Esto es lo que lo distingue de un lenguaje de Netlist. Sin necesidad de conocer la estructura interna de un circuito es posible describirlo explicando su funcionalidad. Esto es especialmente útil en la simulación ya que permite simular un sistema sin conocer su estructura interna, pero este tipo de descripción se está volviendo cada día más importante porque las actuales herramientas de síntesis permiten la creación automática de circuitos a partir de una descripción de su funcionamiento.

CAPÍTULO 2

Análisis y diseño

2.1 Descripción del proyecto

Cuando se encuentra estático un giroscopio microelectromecánico (MEMS) proporciona un nivel de tensión constante en una de sus terminales (llamada RATEOUT).

Si se hace girar al dispositivo sobre el eje z en el sentido de las manecillas del reloj y a una velocidad angular ω (figura 2.1.1), entonces el voltaje entregado por el giroscopio se incrementa. Si se hace girar en el sentido opuesto a las manecillas del reloj, entonces dicho voltaje disminuye.

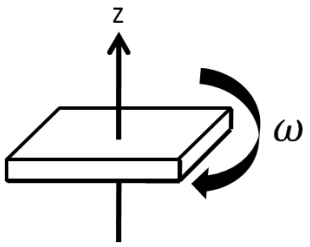


Figura 2.1.1: Giroscopio girando sobre el eje z a una velocidad ω

Lo anterior se puede expresar mediante la siguiente ecuación matemática

$$v(\omega) = m\,\omega + b$$

Donde

$v(\omega)$ = volts proporcionados por el MEMS en la terminal Rateout.

$\omega =$ velocidad angular a la que gira el MEMS (dada en °/s). Puede ser positiva o negativa, dependiendo del sentido de giro.

$m =$ sensitividad. Indica la cantidad de volts que se incrementa $v(\omega)$ con cada cambio de ω (cada °/s).

$b =$ voltaje proporcionado por el MEMS cuando se encuentra en reposo ($\omega = 0$).

El proyecto consta de 3 fases:

1ª fase: Diseño y construcción de un sistema para la obtención de los parámetros m y b del modelo matemático de un giroscopio microelectromecánico.

2ª fase: Comprobación del modelo. Teniendo el sistema diseñado y construido, se utilizan vectores de prueba para excitar al giroscopio con diferentes velocidades y con esto poder calcular el error existente entre el modelo matemático obtenido y los nuevos resultados.

3ª fase: Modelo compensado en temperatura. Se realiza un sistema tal que se pueda obtener el modelo del giroscopio tomando en cuenta las variaciones de temperatura del dispositivo.

Este trabajo sólo se concentra en resolver la primera etapa del proyecto, así el objetivo de este proyecto terminal es diseñar y construir un sistema para obtener los parámetros m y b de un giroscopio microelectromecánico.

Para hacerlo, se hace girar al MEMS a diferentes velocidades, tanto en el sentido de las manecillas del reloj, como en el sentido opuesto, y se almacenan los valores $v(\omega)$ adquiridos en cada velocidad.

Posteriormente se grafica ω vs $v(\omega)$ y se realiza un ajuste por mínimos cuadrados de los datos adquiridos (aproximándolos a una recta) para obtener los parámetros m y b del giroscopio con el mínimo error.

Es conveniente mencionar que los parámetros m y b también dependen de la temperatura a la cual se encuentra trabajando el dispositivo, esta también la proporciona el MEMS en una de sus terminales. Este proyecto no utiliza dicho dato para la obtención de los parámetros, sin embargo sí se toma en consideración para en un futuro poder seguir trabajando sobre él.

Para comprobar el funcionamiento del sistema de obtención de parámetros se utiliza un giroscopio comercial ADXRS300, cuya hoja de especificaciones se encuentra en el apéndice F.

Sus parámetros a 25°C son:

Sensitividad:	mín: 4.6 mV/°/s	típico: 5 mV/°/s	máx: 5.4mV/°/s
Voltaje en reposo:	mín: 2.3V	típico: 2.5V	máx: 2.7V
Rango dinámico:	±300°/s		

A continuación, mediante un diagrama a bloques, se explica el significado de cada bloque principal del cual está compuesto el proyecto.

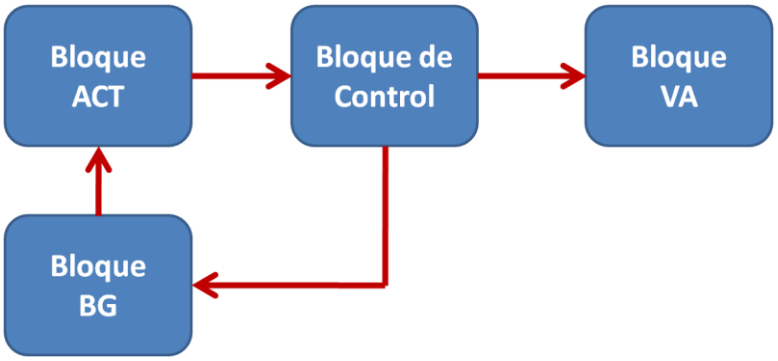


Figura 2.1.2: Diagrama a Bloques.

- Bloque ACT (Adquisición, Conversión y Transmisión)

Este bloque tiene la función de adquirir la señal analógica de velocidad y temperatura del giroscopio y con el ADC del micro-controlador convertirla a digital y por último con una comunicación serial realizar la transmisión RF.

- Bloque BG (Base Giratoria)

Este bloque tiene una base giratoria la cual debe rotar a una velocidad deseada para excitar al sensor, este bloque incluye la etapa de potencia para manejar al motor de DC.

- Bloque de control

Este bloque consta de una tarjeta de desarrollo para FPGAs la cual es la encargada de controlar al módulo receptor de RF, de variar la velocidad del motor con técnica PWM y también el manejo de la comunicación serial con una PC.

- Bloque VA (Visualización y Análisis)

Por medio del software de LabVIEW se visualizan y analizan los datos recibidos.

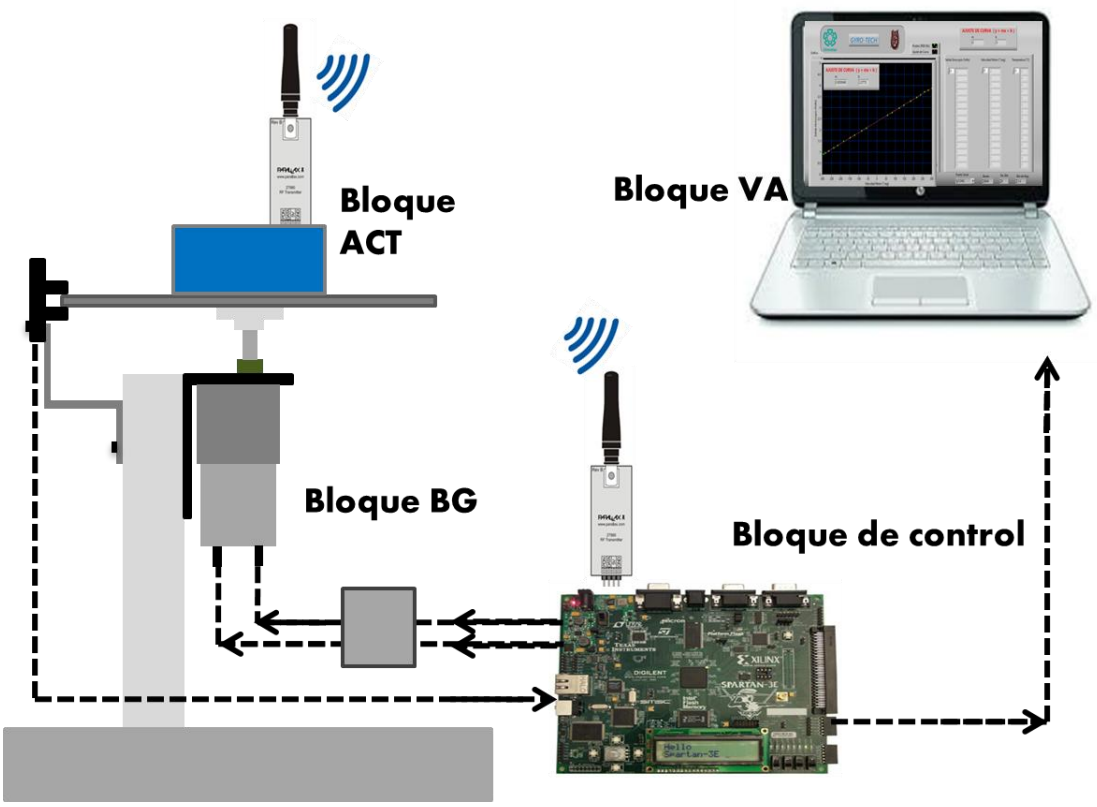


Figura 2.1.3: Sistema de obtención de parámetros de un giroscopio microelectromecánico.

2.2 Bloque ACT (Adquisición, Conversión y Transmisión)

El giroscopio MEMS proporciona dos señales:

1) Un voltaje mediante el cual se permite conocer la velocidad a la cual está girando y 2) Un nivel de tensión que indica la temperatura interna a la cual se encuentra trabajando el dispositivo.

La función del bloque ACT es adquirir estos niveles de tensión y convertirlos a una señal digital para transmitirlos de forma serial al FPGA (vía inalámbrica)

Para realizar la tarea de adquisición y conversión se utiliza el microcontrolador ATmega48 de ATMEL el cual cuenta con un ADC (convertidor analógico a digital) de 10 bits de resolución, es decir, al hacer la conversión con un voltaje de referencia de 5 V, se tiene una resolución en el convertidor de 4.8mV, que cumple con lo mínimo requerido para los cambios de velocidad de giro del MEMS por cada grado por segundo $\left(5\,\frac{mV}{^{\circ}/_{s}}\right)$.

Convertidor Analógico a Digital.

Para realizar este programa es conveniente tener en consideración que se requieren diez bits de resolución, una señal de referencia para el circuito convertidor $V_{ref} = 5\text{V}$ y dos canales analógicos de entrada para la conversión de dos señales: velocidad angular y temperatura del MEMS.

° Reloj del circuito de conversión

Utilizando un reloj externo de 4MHz, es necesario seleccionar el valor del pre escalador para que divida la frecuencia entre 32 y así tener un reloj del ADC de 125kHz, esto se realiza para cumplir con la condición que indica el fabricante de que el circuito de aproximaciones sucesivas, es el método que utiliza el ATmega48 para realizar la conversión, debe tener un reloj con una frecuencia de entre 50kHz y 200kHz.

° Modo de conversión

Debido a que se requiere realizar la conversión de dos señales analógicas, se tiene que estar cambiando continuamente el canal del convertidor. En el modo conversión simple se tiene el control de cuándo inicia una conversión, así que se puede cambiar el canal antes de iniciarla y con esto tener la certeza de que el valor convertido es el de la señal que se desea.

Para reducir el error en la conversión debido al ruido, se realiza un promedio de 16 conversiones de cada señal. Esto es, se crea una variable a la cual se le suman los valores de las conversiones terminadas y, una vez sumadas 16 conversiones, se realiza el promedio.

Transmisión

Como lo que se desea transmitir son datos digitales de forma serial, se debe utilizar el USART (Transmisor y Receptor Síncrono y Asíncrono Universal) con el que cuenta el ATmega48.

La conversión del ADC son diez bits, así que se toma la decisión de separarla en dos palabras de ocho bits de datos cada una, la primera de ellas contiene los dos bits más significativos de la conversión y la otra palabra los ocho bits menos significativos, tal como se muestra en la figura 2.2.1.

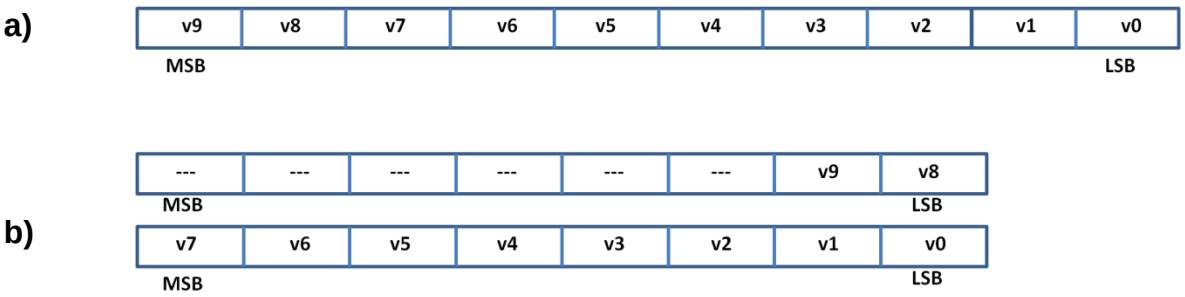


Figura 2.2.1: a) Diez bits de datos de la conversión del ADC de la velocidad. b) Se separan los diez bits quedando dos bytes, uno con los dos bits más significativos (v9 y v8) y el otro con los ocho bits menos significativos (v7 - v0) de la conversión.

Para esta parte del programa, es conveniente tener en consideración que se envían tramas de ocho bits de datos, sin bit de paridad, un solo bit de parada, velocidad de transmisión de 9600 bauds, modo asíncrono, además de que se cuenta con un oscilador externo de 4MHz.

Cada trama transmitida está constituida como se muestra en la figura 2.2.2. Se puede observar que en una sola trama se mandan cuatro palabras, dos de la conversión de la velocidad angular y en seguida las dos de la temperatura.

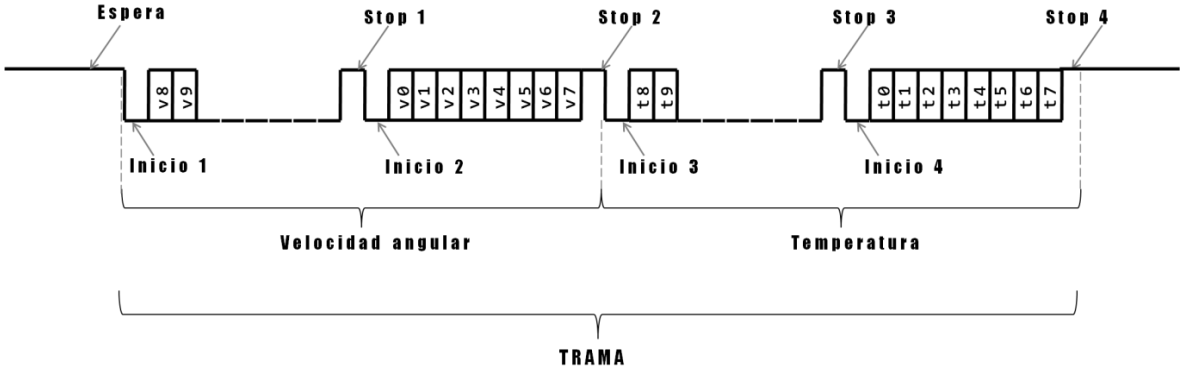


Figura 2.2.2: Se muestra la manera en que está conformada una trama.

Las tramas transmitidas son como se muestra en la figura 2.2.3. Se deja un tiempo de 300 ms entre trama y trama para que se pueda sincronizar la recepción en el FPGA. En la sección 2.4.1 *Módulo Recep del FPGA* se explica cómo se logra esa sincronía.

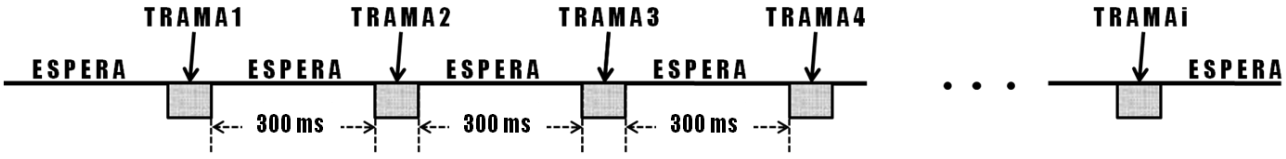


Figura 2.2.3: Las tramas se encuentran con una separación de 300 ms entre ellas.

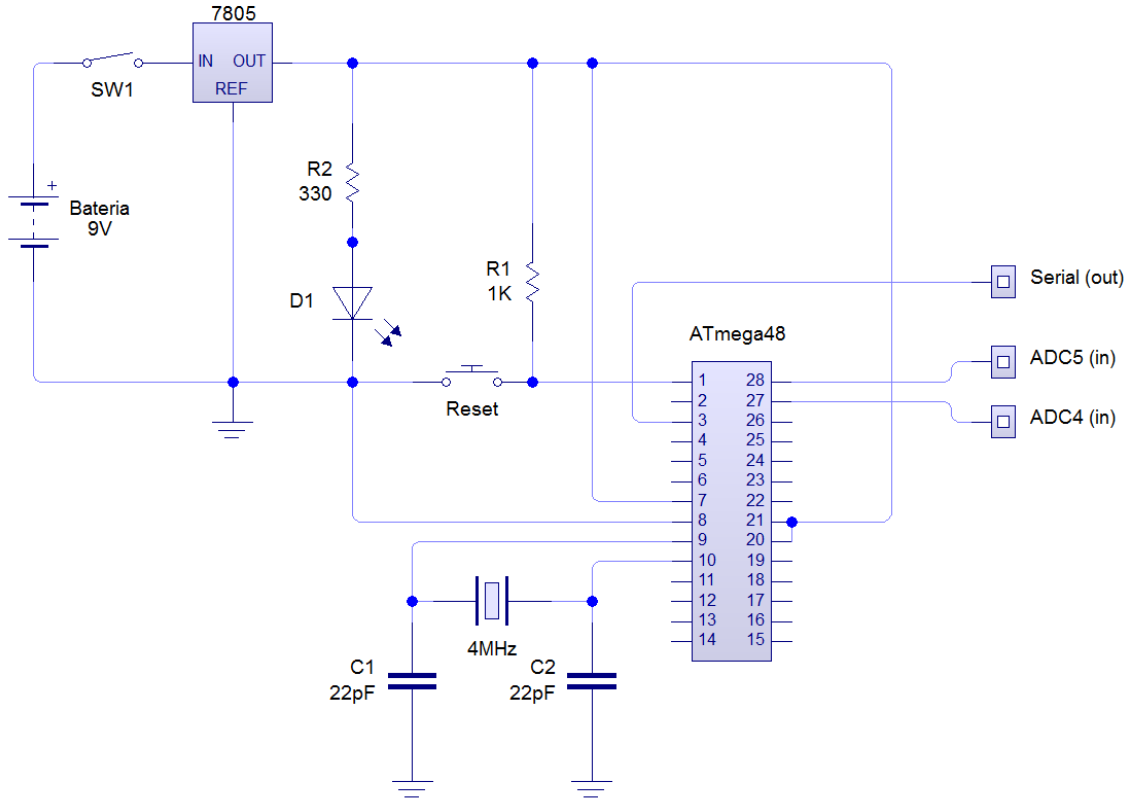


Figura 2.2.4: Circuito utilizado para el bloque ACT. *Serial* es la salida por la cual el USART transmite los datos. En *ADC5* entra la señal de la velocidad proporcionada por el MEMS. En *ADC4* entra la señal de la temperatura proporcionada por el MEMS.

El código completo en C++ se incluye en el apéndice B para su consulta.

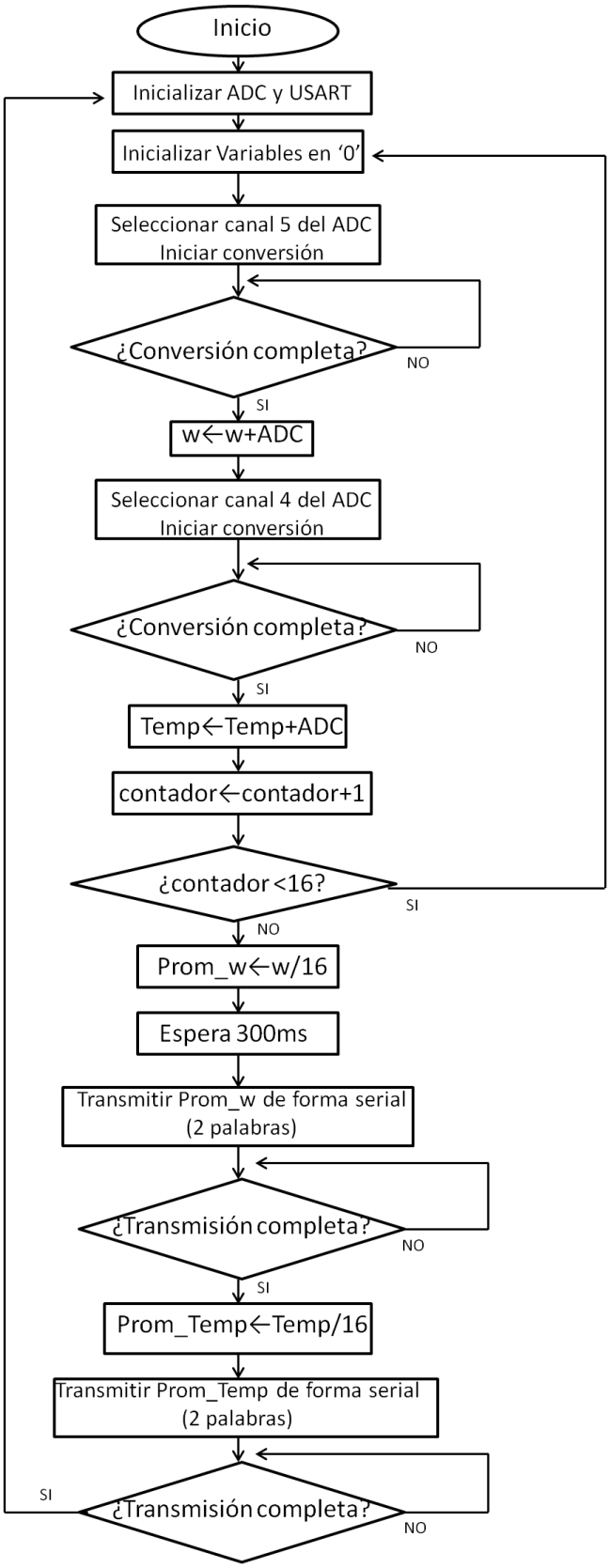


Figura 2.2.5: Diagrama de flujo del código en C++ para la conversión y transmisión.

2.2.1 Módulos de RF

Debido a que la tarjeta en la que está montado el MEMS (bloque ACT) se encuentra sobre una base giratoria, es difícil enviar los datos de forma serial a través de un cable, por tal motivo se toma la decisión de ocupar comunicación inalámbrica.

Para seleccionar los módulos de transmisión y recepción adecuados, se debe tener en consideración las características de la señal que es enviada.

1. Los datos enviados son digitales (0 Volts y 5 Volts)
2. Son datos seriales.
3. La velocidad a la que se transmiten los datos es de 9600 bauds.
4. Los datos son enviados a una distancia de aproximadamente 40 cm.

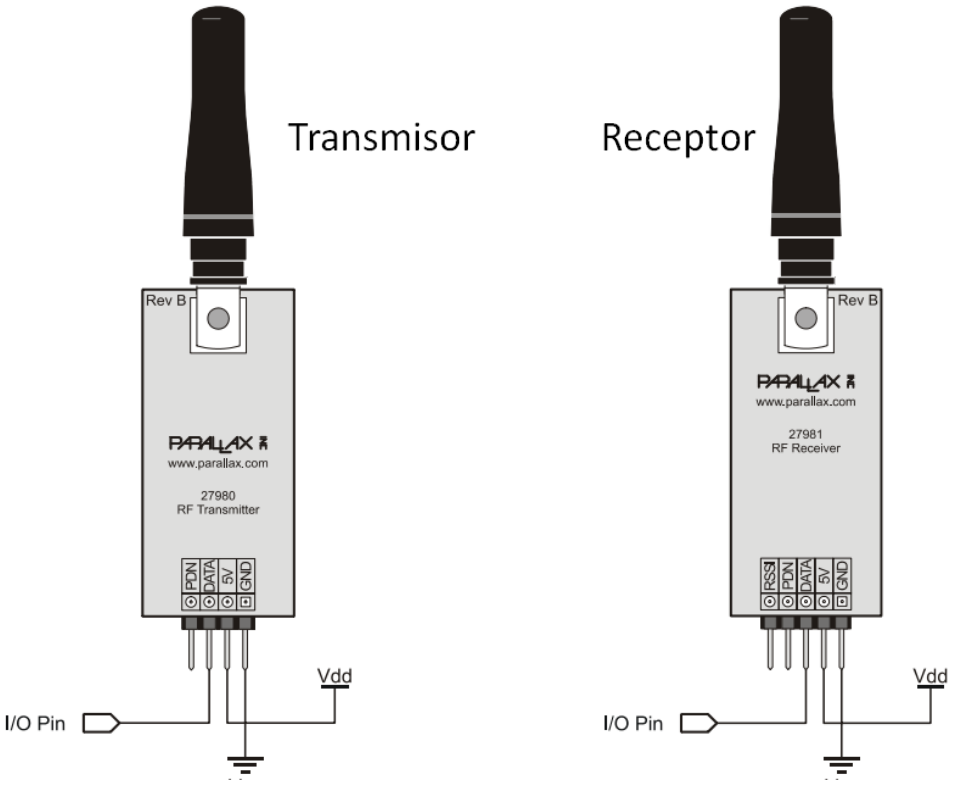


Figura 2.2.1.1: Diagrama de conexión de los módulos transmisor y receptor.

Unos módulos que cumplen con estas características son los Parallax #27980 (transmisor) y #27981 (receptor) mostrados en la figura 2.2.1.1. La velocidad de transferencia de datos de estos módulos es de entre 1200 bauds a aproximadamente 19.2 k bauds, utilizan modulación OOK (del inglés Modulación de Encendido-Apagado) o ASK (del inglés Modulación por Desplazamiento de Amplitud). La distancia máxima a la que puede recibir la información es de 150 metros. Además son sencillos de utilizar, ya que en el caso del transmisor sólo se debe conectar a una alimentación de 5 V y aplicar los datos a la terminal DATA del módulo. En el caso del receptor, además de la alimentación de 5V, los datos recibidos se toman de la terminal DATA.

2.3 Bloque BG (Base Giratoria)

El motor utilizado para el proyecto es un motorreductor metálico con relación de engranaje 100:1 y cuyas especificaciones con 12 V de alimentación son las siguientes:

- Velocidad: 100RPM
- Corriente sin carga: 300mA
- Corriente máxima: 5A

La razón para utilizar un motor con caja de engranes es para poder hacerlo girar a velocidades bajas (1°/s de ser posible).

Figura 2.3.1: Motor con relación de engranaje 100:1 de Pololu.

Para poder cambiar el sentido de giro del motor, además de variar su velocidad, se utiliza un módulo fabricado por Pololu que incluye el puente H VNH5019A-E. Cuyas características son:

- Voltaje de operación: 5.5 – 22V
- Corriente a la salida: 12 A
- Puede trabajar con PWM con frecuencias de hasta poco más de 20kHz

Las terminales ocupadas del módulo se muestran en la figura 2.3.2.

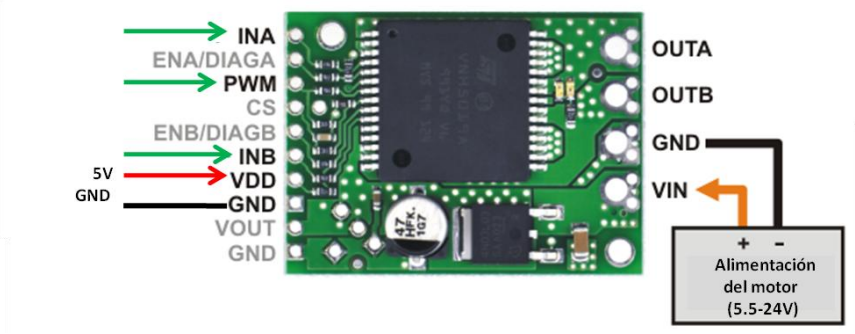


Figura 2.3.2: Terminales ocupadas del módulo con puente H VNH5019A-E.

Donde INA e INB determinan el sentido de giro del motor. Cuando INA=1 e INB=0 el motor gira en el sentido de las manecillas del reloj. Si INA=0 e INB=1 el motor gira en el sentido opuesto a las manecillas del reloj.

OUTA se conecta a una de las terminales del motor y OUTB a la otra terminal.

2.3.1 Encoder (Interruptor óptico H21B1)

Este dispositivo es necesario para que el módulo GPS (Grados Por Segundo) del FPGA adquiera los datos que le permitan realizar el cálculo del tiempo que tarda el motor en dar una vuelta. En la sección 2.4.2 *Módulo GPS del FPGA* se profundiza en este tema.

Se arma el circuito como se muestra en la figura 2.3.1.1. La salida del encoder se toma del colector. Debido a que en estos sensores existe ruido en la señal de salida, se coloca un inversor Schmitt para eliminarlos.

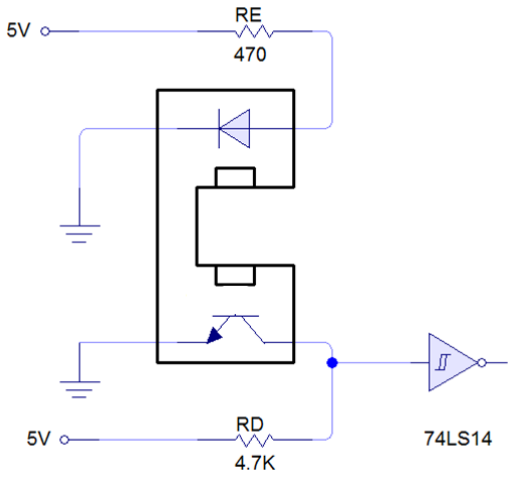


Figura 2.3.1.1: Se muestra la forma de conectar el interruptor óptico.

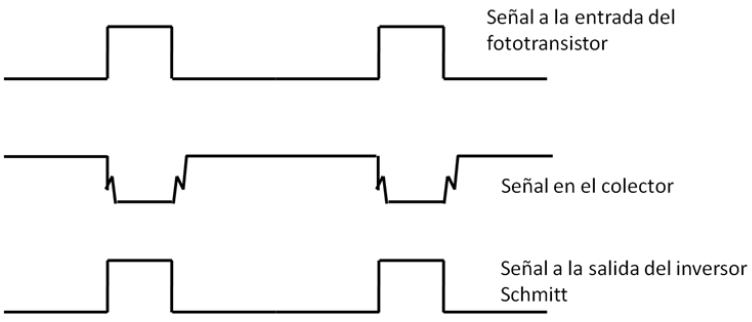


Figura 2.3.1.2: Al conectar un inversor Schmitt se tiene una señal sin ruido.

2.4 Bloque de control

Las funciones del bloque de control, que utiliza una tarjeta de desarrollo para FPGA, son:

- Variar la velocidad del motor utilizando la técnica PWM.
- Almacenar 20 datos de velocidad y temperatura provenientes del bloque ACT.
- Auxiliar en el cálculo del tiempo que tarda en dar una vuelta el eje del motor.
- Crear una interfaz sencilla por medio de un LCD para el usuario.
- Transmitir los datos de forma serial hacia LabVIEW.

Se crean módulos (o componentes) en el FPGA para que cumplan con alguna de las tareas requeridas. Los módulos son: Recep, GPS, PWM, Memoria R/W, Mensajes y Tran.

Teniendo estos módulos funcionando de forma independiente, se acondicionan para poder interconectarlos y se crea un último módulo: Control. Este es el encargado de controlar el orden de ejecución de los demás componentes.

El orden de ejecución es el siguiente:

1) Mostrar mensajes de bienvenida en el LCD
2) Indicar al usuario que presione el botón “enter” para iniciar la prueba.
3) Se asigna la velocidad del motor con el %C.U. del PWM.
4) Se espera 4 segundos para que el motor estabilice su giro.
5) Se almacena un valor de velocidad angular y uno de Temperatura del MEMS recibidos vía inalámbrica.
   Se calcula y almacena el tiempo que tarda en dar una vuelta el eje del motor.
6) Se incrementa la velocidad del motor.
7) Se repiten 9 veces del paso 4 al 6.
8) Se cambia el sentido de giro del motor.
9) Se repiten 10 veces del paso 4 al 6, sólo que en el paso 6 se disminuye la velocidad en lugar de incrementarla.
10) Teniendo 20 valores almacenados de velocidad, temperatura y periodo (tiempo que tarda en dar una vuelta el eje del motor), se indica al usuario que la prueba terminó para que presione el botón “envía” y con ello poder iniciar la transmisión de los datos de forma serial hacia LabVIEW.

Para las siguientes secciones es conveniente auxiliarse de un diagrama a bloques, el cual contiene las conexiones de señales para los módulos diseñados en el

FPGA, figura 2.4.1. Todos ellos forman parte del bloque de control del diagrama a bloques general.

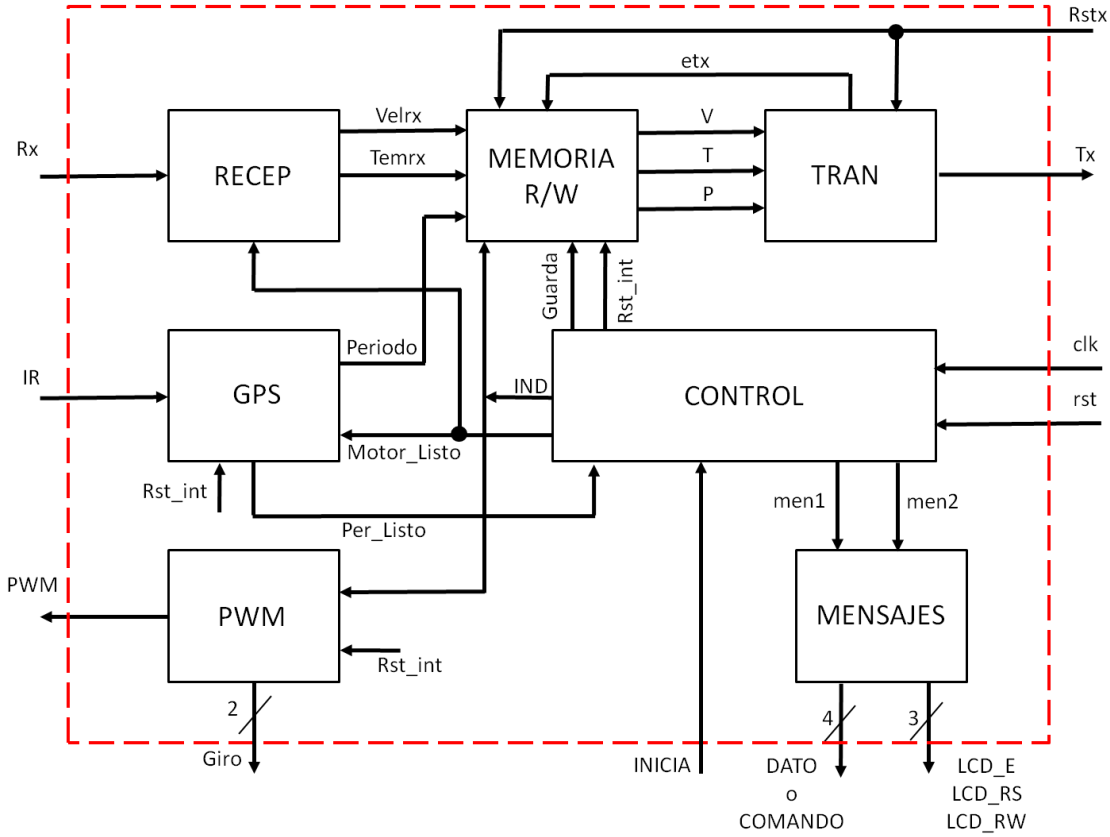


Figura 2.4.1: Diagrama a bloques en el FPGA (bloque de control)

2.4.1 Módulo Recep del FPGA

Este bloque tiene como función capturar y almacenar una de las tramas transmitidas por el bloque ACT.

Se sabe que se está transmitiendo una trama cada segundo, por lo cual lo primero que se realiza es detectar el inicio de la trama y el encargado de realizar esta tarea es el siguiente proceso descrito en VHDL (la recepción permanece deshabilitada hasta que se detecte el bit de sincronía).

```
--Contador para el bit de sincronía
process (rst,clk,rx)
begin
   if rst='1' or rx='0' then
      sincro <=(others=>'0');
   elsif clk'event and clk='1' then
      sincro<=sincro+1;
   end if;
end process;

--Habilita o deshabilita la recepción (Listo, Bit de sincronía y bit de inicio)
process (rst,rx,trx,sincro,Listo)
begin
   if rst='1' then
      erx <= '0';
   elsif Listo='1' and rx='0' and sincro>X"0DD40A0"then
         erx <= '1';           -- Habilita la recepción
   elsif trx="1111010" then --Otra opcion: elsif Listo='0' then
      erx <= '0';           -- Deshabilita la recepción
   end if;
end process;
```

Con un contador, que se incrementa mientras la señal recibida “rx” sea un estado alto, y un comparador se detecta el inicio de la trama, esto es, si dicho contador (llamado “sincro”) llega a la cuenta de “0DD40A0” (equivalente a 290ms), entonces el siguiente flanco de bajada (rx=’0’ del código) es el indicador de que el primer bit de inicio es detectado, es decir, la trama ha comenzado (véase figura 2.4.1.1).

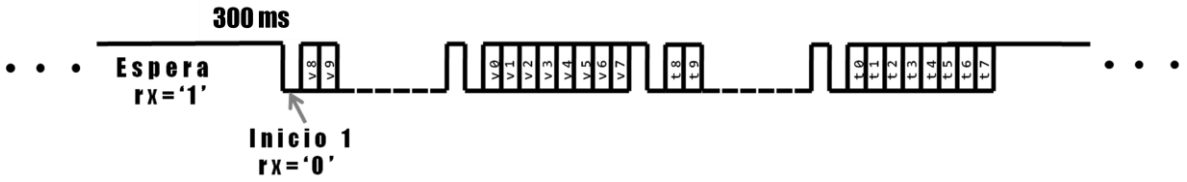


Figura 2.4.1.1: La señal recibida ‘rx’ se encuentra en estado alto durante 300ms, hasta que aparece Inicio1 (rx=’0’) que es el inicio de la trama.

Para almacenar los bits de datos se muestrea la trama tres veces más rápido que la velocidad a la que llega (3 x 9600 bauds). Esto equivale a decir que

cada bit recibido se “divide” en tres y se lee su contenido en la segunda parte del bit (ver figura 2.4.1.2).

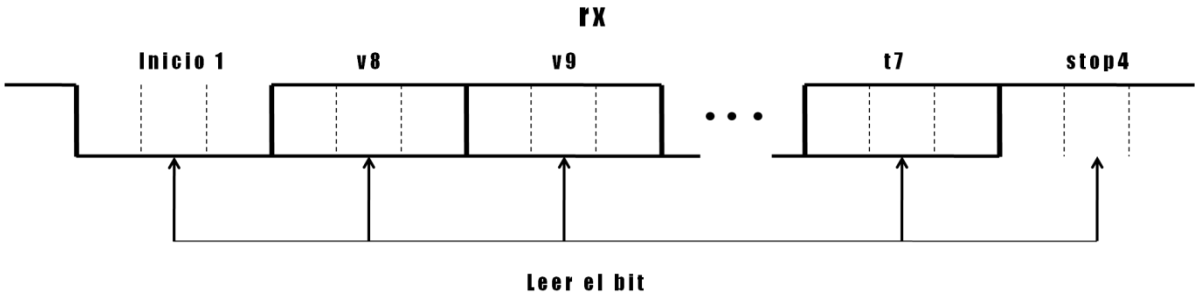


Figura 2.4.1.2: Los bits de la trama son muestreados 3 veces más rápido de la velocidad a la que llegan y son leídos a 2/3 partes de su duración.

Los bits de datos se almacenan en 2 registros, uno para los datos de la velocidad y el otro para los de la temperatura. Se almacenan en orden, de mayor peso a menor peso, como se muestra en la figura 2.4.1.3.

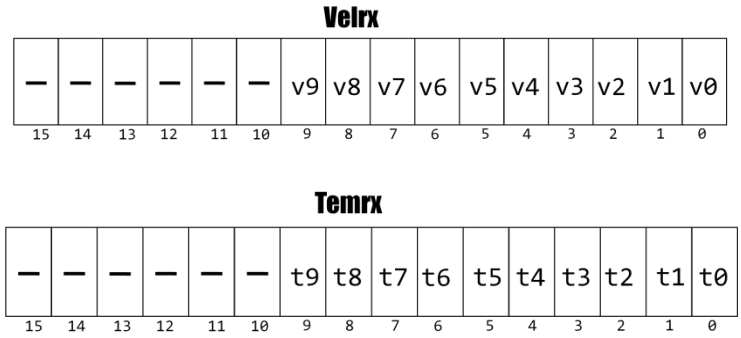


Figura 2.4.1.3: Registros en los que se almacenan los bits de datos recibidos.

Todo este módulo permanece deshabilitado hasta que recibe un pulso de la máquina de estados llamado “Motor_Listo”. Una vez almacenados los registros Velrx y Temrx se deshabilita la recepción hasta que vuelve a llegar el pulso “Motor_Listo”.

2.4.2 Módulo GPS del FPGA

La función de este módulo es ayudar en el cálculo del tiempo que tarda el eje del motor en dar una revolución.

Teniendo la señal del encoder sin rebotes (llamada “SinReb”) se realiza una señal de referencia (“ref”) la cual cambia de nivel lógico con cada flanco de subida de “SinReb” tal como se muestra en la figura 2.4.2.1. El siguiente proceso en VHDL es el encargado de formar a la señal “ref”.

```
-- Señal de Referencia
process (SinReb,rst)
begin
   if rst='1' then
      ref <= '0';
   elsif SinReb'event and SinReb='1' then
      ref <= not ref;
   end if;
end process;
```

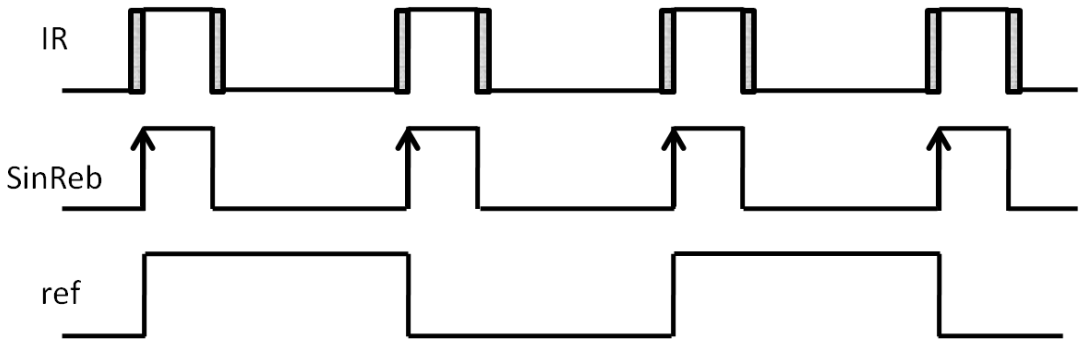


Figura 2.4.2.1: Se muestra la manera en que se forma la señal ref. “IR” es la señal del encoder con rebotes.

Otro proceso en VHDL realiza un contador (“cnt1”) que se incrementa cada 1 ms mientras la señal “ref” se encuentre en estado alto. En un registro llamado “periodo” se almacena el valor del contador cada vez que llega el flanco de bajada de “ref”. Véase figura 2.4.2.2.

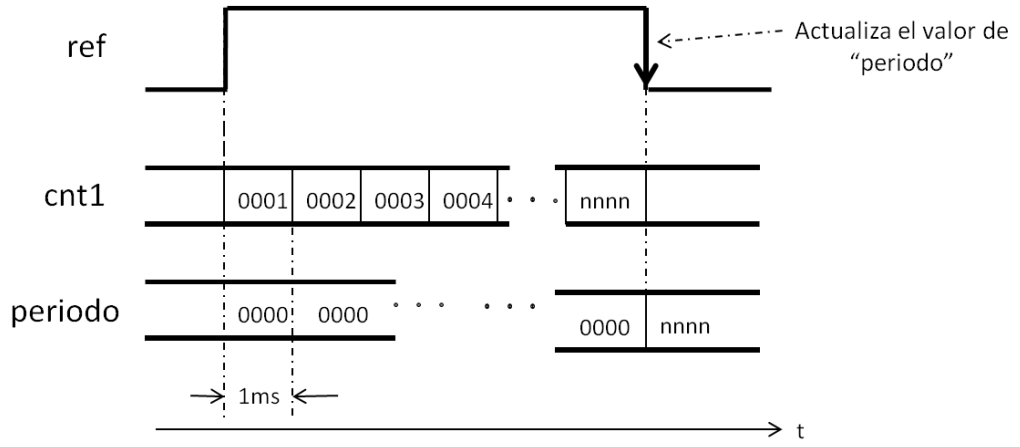


Figura 2.4.2.2: "cnt1" se incrementa cada 1 ms mientras "ref" se encuentra en estado alto. El valor de "periodo" se actualiza con el flanco de bajada de "ref".

El periodo se calcula de la siguiente manera

$$T = \text{periodo} * 1ms$$

Este cálculo no se realiza en el FPGA, ya que al evitar hacer la multiplicación se reducen tiempos en la ejecución de los procesos. Así que el registro "periodo" se envía a LabVIEW y este se encarga de realizar la operación.

El contador cnt1 se diseña de tal forma que la señal "periodo" sea de máximo 16 bits.

De la caracterización del motor se tiene que el tiempo máximo y mínimo que tarda el motor en dar una vuelta es:

$$t_{mín} \approx 1\ seg\ \ ;\ \ t_{máx} \approx 10\ seg$$

Se selecciona la base de tiempo del contador cnt1 de 1ms ya que cuando el motor tarda 10 segundos en dar una vuelta se tiene una cuenta máxima de cnt=10,000 (en hexadecimal cnt=2710), es decir, se puede representar "periodo" con 16 bits, y cuando el motor tarda 1 segundo en dar una vuelta se tiene una cuenta de cnt=1,000, la cual es una buena resolución para el cálculo de "periodo".

La razón por la que "periodo" es de 16 bits es para que se almacene en la memoria (otro módulo del FPGA en el cual se almacenan datos de 16 bits) y posteriormente se pueda transmitir hacia LabVIEW de forma serial en sólo 2 palabras de 8 bits. En la sección 2.4.5 *Módulo Tran del FPGA* se habla más a detalle sobre dicha transmisión.

Este módulo permanece deshabilitado hasta que un pulso llamado “Motor_Listo” (proveniente de la máquina de estados) es recibido. Una vez que se realiza el cálculo de “periodo” se levanta una bandera llamada “Per_Listo” y se deshabilita el módulo GPS. Permanece deshabilitado hasta que llega nuevamente el pulso “Motor_Listo”.

Proceso en VHDL encargado de realizar tanto la asignación de cnt1 a “periodo” como de levantar la bandera “Per_Listo” que indica que el módulo ha terminado de realizar su tarea.

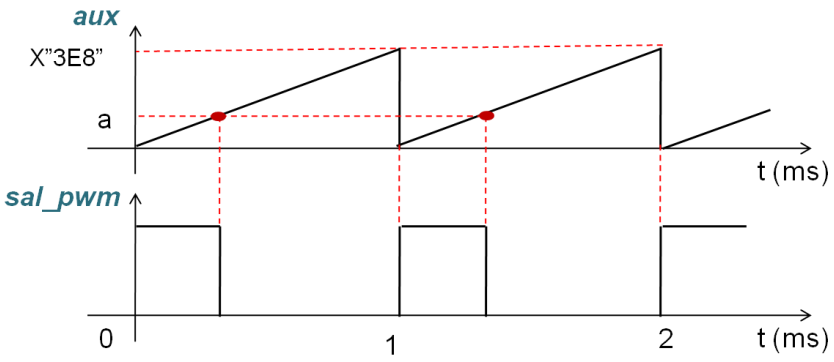
```
-- Calcula el Periodo
process (ref,rst,Motor_Listo)
begin
   if rst='1' or Motor_Listo='1' then
      Periodo <= (others => '0');
      Per_Listo <='0';
   elsif ref'event and ref='0' then
      Periodo <= cnt1; -- hay que hacer en labview la multiplicacion: Periodo x 1ms
      Per_Listo <='1';
   end if;
end process;
```

2.4.3 Módulo PWM del FPGA

La función de este módulo es proporcionar una señal cuadrada de periodo constante cuyo porcentaje de ciclo útil cambia según es requerido.

La forma de realizarlo en lenguaje VHDL es:

Se definen el periodo de la señal T=1ms y el porcentaje de ciclo útil de la misma (‘a’). Se realiza un contador, llamado ‘aux’, y se compara este último con el valor de ‘a’, si son iguales se manda la señal de salida ‘sal_pwm’ a estado bajo. La señal se vuelve a poner en estado alto una vez que ha transcurrido el 1ms. Todo esto se resume en la figura 2.4.3.1.




```
process (clk1M,rst,aux,a)
begin
   if rst='1' or aux=X"3E8" then  -- Tiempo (1000us)
      aux <= (others => '0');
      sal_pwm <= '1';
   elsif clk1M'event and clk1M='1' then
      aux <= aux + 1 ;
      if aux = a then
         sal_pwm <= '0';
      end if;
   end if;
end process;
```

Figura 2.4.3.1: Se muestra el proceso en VHDL encargado de realizar las comparaciones. También se muestra la señal resultante de las comparaciones.

Para el proyecto se requiere de 20 valores distintos de %C.U., así que se definen estos valores y se conectan a las entradas de un multiplexor. Con las llaves de selección del multiplexor se va asignando el valor de “a” para poder formar el pwm con el %C.U. deseado (véase figura 2.4.3.2).

```
case(sel) is
   --Aumentando el PWM
   when "00000" => a <= x"012";--pwm
   when "00001" => a <= x"045";--i0
   when "00010" => a <= x"059";--i1
   when "00011" => a <= x"06D";--i2
   when "00100" => a <= x"08B";--i3
   when "00101" => a <= x"09F";--i4
   when "00110" => a <= x"0BD";--i5
   when "00111" => a <= x"0DB";--i6
   when "01000" => a <= x"0EF";--i7
   when "01001" => a <= x"103";--i8
   when "01010" => a <= x"108";--i9
   --Disminuyendo el PWM
   when "01011" => a <= x"108";--i10
   when "01100" => a <= x"103";--i11
   when "01101" => a <= x"0EF";--i12
   when "01110" => a <= x"0DB";--i13
   when "01111" => a <= x"0BD";--i14
   when "10000" => a <= x"09F";--i15
   when "10001" => a <= x"08B";--i16
   when "10010" => a <= x"06D";--i17
   when "10011" => a <= x"059";--i18
   when "10100" => a <= x"045";--i19
   when others => null;
end case;
```

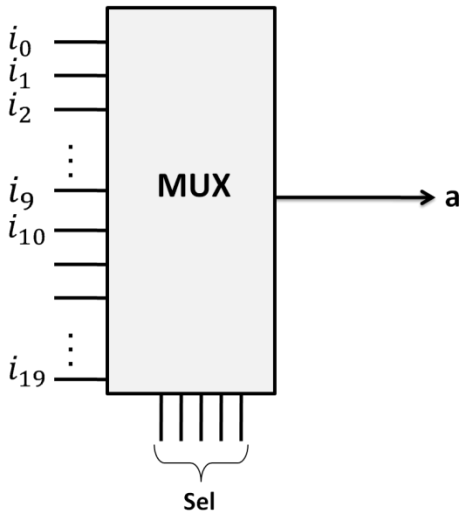


Figura 2.4.3.2: Se muestra el código en VHDL para realizar el multiplexor de 20 entradas y 1 salida, cuyas llaves de selección están conformadas por "sel".

Es conveniente mencionar que las entradas i_0 a i_9 (desde sel=00001 a sel=01010) son utilizadas para el giro en el sentido de las manecillas del reloj y las entradas i_{10} a i_{19} son para el sentido opuesto a las manecillas del reloj (en este sentido de giro el %CU va decrementando).

"sel" es un contador de 5 bits el cuál se incrementa cada vez que recibe un flanco de un pulso llamado "IND" (proveniente de la máquina de estados).

Este módulo también se encarga de proporcionar las señales que determinan el sentido de giro del motor (llamadas Giro(0) y Giro(1)), estas se aplican a las entradas INA e INB del puente H mencionado en el subcapítulo 2.3 *Bloque BG*.

2.4.4 Módulo Memoria R/W del FPGA

Este módulo es el encargado de escribir o leer los datos de temperatura interna del MEMS, velocidad angular del MEMS (registros Temrx y Velrx provenientes del módulo Recep) y “periodo” (obtenida del módulo GPS) en un arreglo de 20 localidades para cada uno (Tem, Vel, Per).

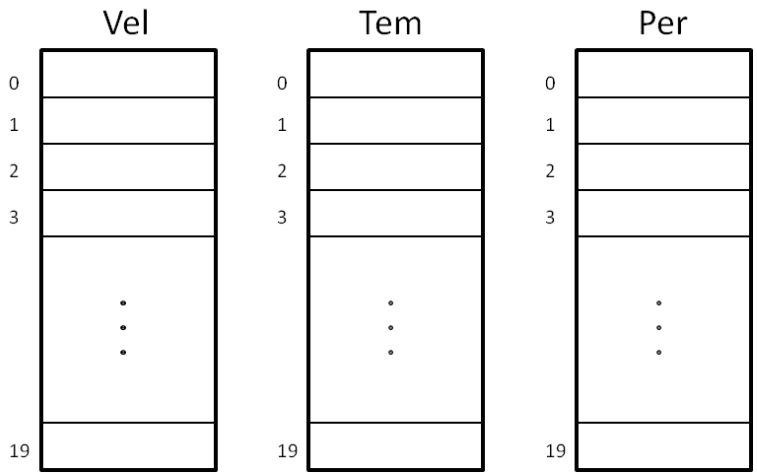


Figura 2.4.4.1: Se muestran los tres arreglos en los que se almacenan la velocidad angular, la temperatura y el periodo.

Con una señal (llamada “we”) se selecciona si se desea leer o escribir en la memoria. Se escribe si we=1, se lee si we=0.

Otra señal (“address”) selecciona la localidad de la memoria a leer o escribir. Una vez seleccionada la localidad de la memoria se lee o se escribe en los tres arreglos.

Proceso de lectura y escritura

```
process(Senal_1,WE)
begin

   if Senal_1'event and Senal_1='1' then
      if(WE='1') then      -- Almacena los datos en la memoria
         Tem(address) <= Temrx;
         Vel(address) <= Velrx;
         Per(address) <= Periodo;
      else                 -- Lee los datos de la memoria
         T <= Tem(address);
         V <= Vel(address);
         P <= Per(address);
      end if;
   end if;

end process;
```

Proceso de cambio de dirección

```
process(rst, rst_int, Senal_2, address,Tem)
begin

   if rst='1' or rst_int='1' or rstx='1' then
      address <= 0;
   elsif Senal_2'event and Senal_2='1' then --Cambio de dirección
      if address < Tem'high then
         address <= address + 1;
      else
         address <= 0;
      end if;
   end if;
```

Para escribir se requiere de 2 pulsos. Uno (llamado “IND”) se encarga de cambiar el valor de “address” (la localidad seleccionada) y la función del otro pulso (“Guarda”) es indicar a la memoria que almacene los valores que recibe de velocidad, temperatura y periodo en la localidad seleccionada.

Para leer es similar, sólo que se utiliza una sola señal (“etx”) tanto para cambiar la localidad seleccionada como para leer el valor de dicha localidad.

Para evitar realizar 2 procesos diferentes para el cambio de localidad y guardar o leer en la memoria, se crean dos señales “senal_1” y “senal_2 “ a las cuales se les asigna “IND”, “Guarda” o “etx” según se lee o se escribe en la memoria.

Proceso de asignación de senal_1 y senal_2

```
-- Senal_1 y Senal_2 se asignan de acuerdo a si se lee o se escribe en la memoria
with WE select
   senal_1<= Guarda when '1',
             etx    when '0',
             '0'    when others;
with we select
   senal_2<= IND    when '1',
             etx    when '0',
             '0'    when others;
```

2.4.5 Módulo Tran del FPGA (Transmisión de datos mediante el puerto Serial)

Para implementar la transmisión del puerto serial, se crea un programa que contiene un conjunto de módulos, los cuales son los siguientes:

- Entrada: contiene los arreglos de valores que se transmiten.
- Serial: en él se programa la velocidad de transmisión, el proceso para cargar los datos, asignación de carácter y el Protocolo de Transmisión.
- Generador: genera la señal de habilitación de transmisión.

Con estos tres módulos y un cuarto, el de control, se puede configurar, controlar y enviar los datos a la interfaz LabVIEW.

Para cada uno de los módulos es necesario explicar más a fondo su programación, de esta manera queda más comprensible cada parte de la transmisión de datos.

Entrada

Este módulo consta de 3 arreglos para cada una de las variables (VEL, TEM y PER). El tamaño del arreglo depende del número de datos que se deseen guardar. Es importante recordar que la longitud de cada localidad de los arreglos es de 16 bits, para poder enviar los datos en sólo 2 palabras de 8 bits.

También dentro de este módulo se encuentra una rutina que es la encargada de leer las localidades de cada uno de los arreglos, la cual requiere de una señal de sincronía.

```
process(clk, rst) is

 begin
      if rst = '1' then
         address<= 0;
      elsif rising_edge (clk) then
         if address < VEL'high then
            address <= address + 1;
         else
            address <= 0;
         end if;
      end if;
end process ;
```

Generador

En este módulo se crea la señal "etx" para habilitar al serial como entrada de sincronía al módulo de entradas. También es importante mencionar que se genera una señal de parado llamado "STOP", la cual es la encargada de indicar el momento en el que se ha transmitido todo el arreglo.

Serial

Primeramente en este módulo de serial es donde se crea la señal ttx (reloj de transmisión), quien es el encargado de llevar la sincronía para la transmisión de los datos. También ttx tiene la tarea de cambiar las palabras de transmisión (V1V2, T1T2 y P1P2).

Otra cuestión a resaltar en el módulo de Serial, este es un pequeño código que se puede llamar precarga, esto se debe a que en cada flanco de bajada de "etx" se carga la información del arreglo a las variables de 8 bits de longitud.

Es importante tener presente que para el modulo de Tran es necesario utilizar un convertidor USB-Serial.

A continuación se muestra la interfaz de Pololu, con el que se decidió trabajar.

Figura 2.4.5.1: Interfaz USB-Serial.

Cuyas características son:

- Voltaje de operación: 3.3 V
- Corriente de alimentación: 25 mA
- Conector: USB mini-B

2.4.6 Módulo Mensajes del FPGA

Para esta tarea es necesario utilizar el LCD (**L**iquid **C**rystal **D**isplay). La tarjeta del Spartan-3E cuenta con uno de 2 líneas por 16 caracteres. Este FPGA controla al LCD por una interfaz de datos de 4 bits que se muestra en la figura 2.4.6.1. A pesar de que este LCD soporta una interfaz de datos de 8 bits, en la tarjeta emplea la interfaz de 4 bits para hacerla compatible con otras tarjetas de Xilinx y para minimizar el número de terminales empleadas.

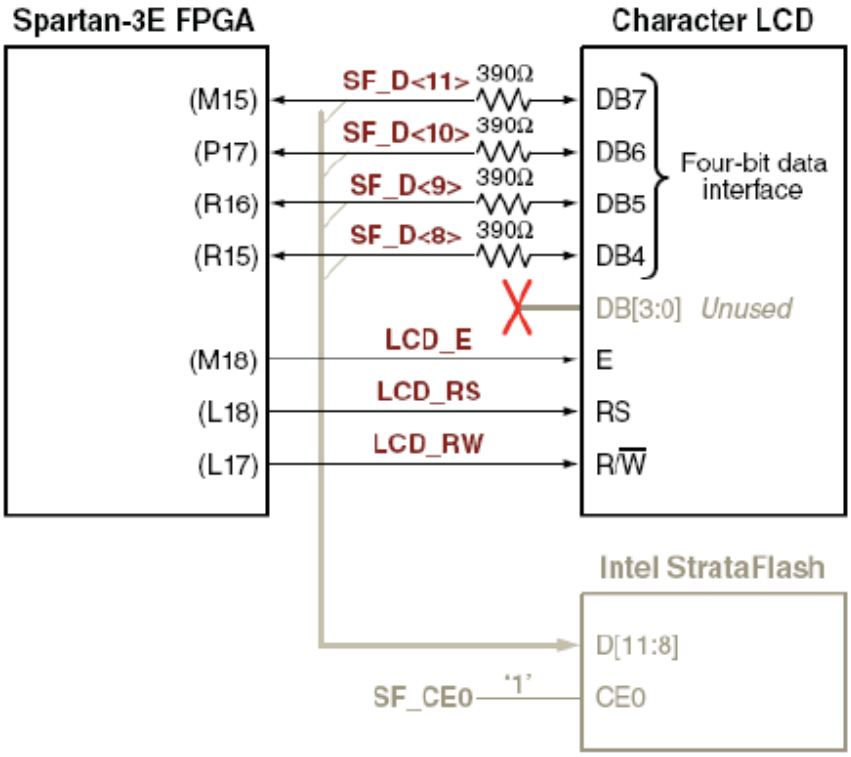


Figura 2.4.6.1: Diagrama de interconexión entre el FPGA y el LCD

Como se muestra en la figura 2.4.6.1, las cuatro señales de datos del LCD también se comparten con las líneas de datos de la memoria StrataFlash SF_D<11:8>. Cuando la memoria StrataFlash es desactivada (SF_CE0 = alto), el FPGA tiene completo acceso lectura/escritura al LCD.

Operación (Interfaz de datos de cuatro bits)

Este tipo de interfaz es la que emplea la tarjeta. La figura 2.4.6.2, ilustra una operación de escritura al LCD, mostrando el tiempo mínimo permitido para la pre colocación, mantenimiento y habilitación, de acuerdo a la duración relativa del pulso de reloj de 50MHz (20ns de periodo) provisto en la tarjeta.

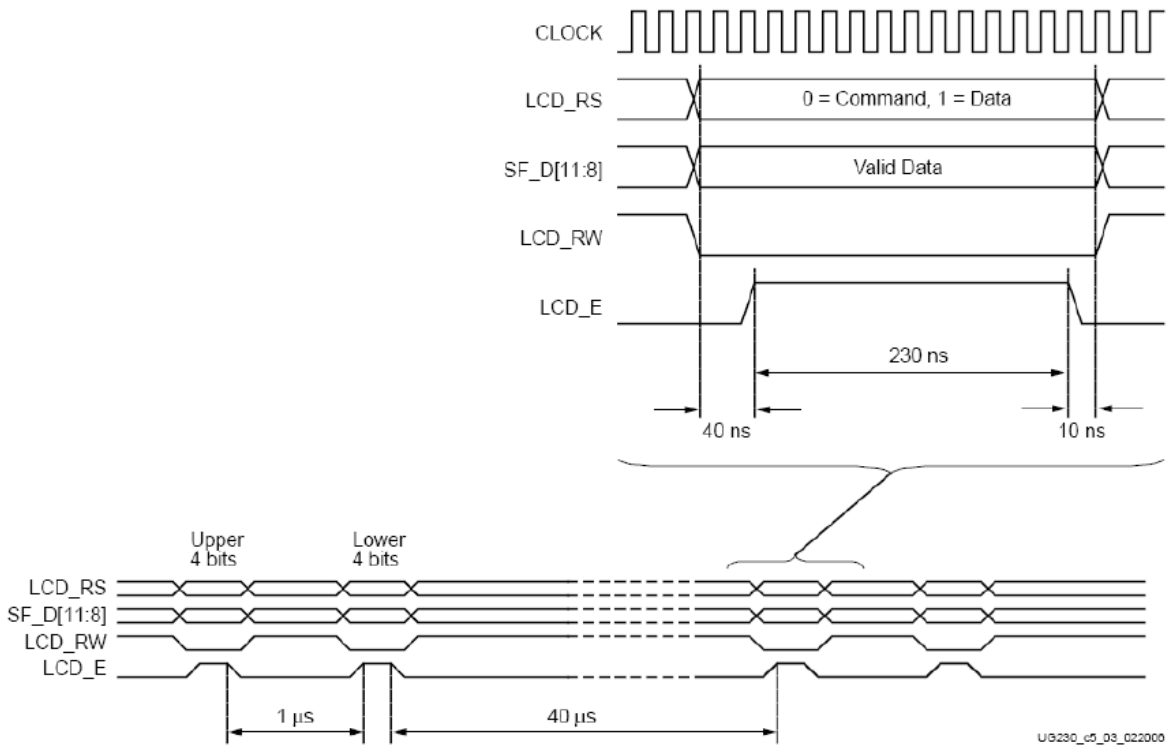


Figura 2.4.6.2: Operación de escritura la LCD

Los valores de los datos sobre SF_D<11:8>, el selector del registro (LCD_RS) y las señales de control de lectura/escritura (LCD_RW) deben ser colocados al menos 40ns antes de que el habilitador LCD_E se ponga en *alto*. La señal de habilitación debe mantenerse en *alto* al menos 230ns o más, el equivalente a 12 o más ciclos de reloj de 50MHz.

En varias aplicaciones, la señal LCD_RW puede conectarse en *bajo*, porque el FPGA generalmente no tiene razón para leer información del LCD.

Todos los comandos y datos de 8 bits se transfieren al LCD empleando dos operaciones secuenciales de 4 bits, espaciados al menos por 1µs, como se muestra en la figura 2.4.6.2. Los cuatro bits más significativos se transfieren primero, seguidos de los cuatro menos significativos. Una operación de escritura de 8 bits debe ser espaciada al menos por 40µs antes de la siguiente comunicación.

A continuación se explica los 3 pasos a seguir para la utilización del LCD:

Inicialización del LCD

La secuencia de inicialización establece primero que la aplicación del FPGA desea emplear la interfaz de datos de cuatro bits al LCD como sigue:

- Esperar 15ms o más, a pesar de que el display está generalmente listo cuando la configuración del FPGA termina. El intervalo de 15ms es equivalente a 750,000 ciclos de reloj a 50MHz.
- Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- Esperar 4.1ms o más que es equivalente a 205,000 ciclos de reloj a 50MHz.
- Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- Esperar 100µs o más que es equivalente a 5,000 ciclos de reloj a 50MHz.
- Escribir SF_D<11:8>=0x3, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- Esperar 40µs o más que es equivalente a 2,000 ciclos de reloj a 50MHz.
- Escribir SF_D<11:8>=0x2, mantener el pulso LCD_E en Alto por 12 ciclos de reloj.
- Esperar 40µs o más que es equivalente a 2,000 ciclos de reloj a 50MHz.

Configuración del LCD

Después de que la inicialización se completa, la interfaz de cuatro bits se establece. La siguiente parte de la secuencia es configurar al LCD:

- Configurar el display para operar sobre la tarjeta Spartan-3E con el comando Function_Set, 0x28.
- Preparar el display para incrementar automáticamente el indicador de la dirección con el comando Entry_Mode_Set, 0x06.
- Encender el display y deshabilitar el cursor y el parpadeo con el comando Display_On/Off, 0x0C.
- Finalmente, permitir al menos 1.64ms (82,000 ciclos de reloj) después de emitir el comando Clear_Display.

Escribiendo datos en el LCD

Para escribir datos en el display se especifica la dirección de inicio, seguida de uno o más valores de datos. Antes de escribir cualquier dato, se introduce el

comando Set_DD_RAM_Address para especificar la dirección inicial de 7 bits en la DD_RAM.

Escribir datos en el display empleando el comando Write_Data_to_CG_RAM o DD_RAM. El valor de datos de 8 bits representa la tabla de búsqueda de la dirección dentro de la CG_ROM o la CG_RAM. El bitmap almacenado en la CG_ROM o la CG_RAM maneja la matriz de 5x8 puntos para representar el carácter asociado.

Si el contador de la dirección es configurado para auto-incrementarse, la aplicación puede escribir secuencialmente múltiples códigos de caracteres y cada carácter es almacenado automáticamente y mostrado en la siguiente localización disponible.

Sin embargo, continuando con la escritura de caracteres, eventualmente decae de la última a la primera línea del display. Los caracteres adicionales no aparecen automáticamente en la segunda línea porque el mapa DD_RAM no es consecutivo de la primera a la segunda línea.

Nota: si la aplicación no utiliza al LCD de caracteres, el pin LCD_E se pone en *bajo* para deshabilitarlo. También el pin LCD_RW se pone en *bajo* para prevenir a la pantalla LCD de datos presentes.

Código en VHDL

Para el código a utilizar en VHDL, es necesario dejar en claro algunos puntos importantes de él. Se empieza por el tamaño de cada instrucción el cual es de 32 bits, en este caso se utiliza el código hexadecimal para un manejo más sencillo. En la figura 2.4.6.3, se muestra la información que contiene cada uno de los bits.

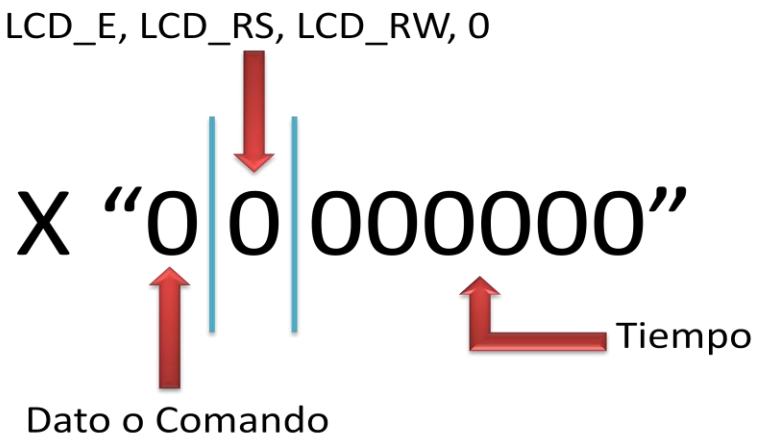


Figura 2.4.6.3: Información contenida en la Instrucción

Otro punto a considerar es el hecho de repetir la instrucción dos veces, esto se debe a que se requiere un cierto tiempo en alto (230ns) para la señal LCD_E.

X"880007D0", -- 40us

X"8000000C", -- 230ns

Como ya se mencionó, se utiliza una interfaz de 4 bits por lo cual se requiere mandar primeramente los cuatro bits más significativos y posteriormente los menos significativos del dato o comando.

X"880007D0", -- 40us

X"8000000C", -- 230ns

X"28000032", -- 1us

X"2000000C", -- 230ns

Cumpliendo con los tiempos indicados en la figura 2.4.6.2, se logra leer el dato o comando.

En cada uno de los módulos se tiene un proceso para la lectura de las instrucciones, el cual se muestra a continuación.

```
process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
   listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
         listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
```

La señal CNTR es un contador que se incrementará cada ciclo de reloj de 50 MHz, siempre y cuando la bandera (listo_in) se encuentre activada. Para la segunda condición se espera que concluya el tiempo definido en la instrucción correspondiente para así poder asignar los valores de dato o comando y las señales de control. En la tercera condición se pregunta si ya se llegó a la última instrucción del arreglo, si no es así se incrementa la señal "index", pasando a la siguiente instrucción, de lo contrario, a través de una bandera (listo_out), se indica que el proceso concluyó.

Para tener un mejor panorama del programa primeramente se presenta un diagrama de flujo, figura 2.4.6.4, que contiene la técnica empleada de banderas para pasar de un mensaje a otro.

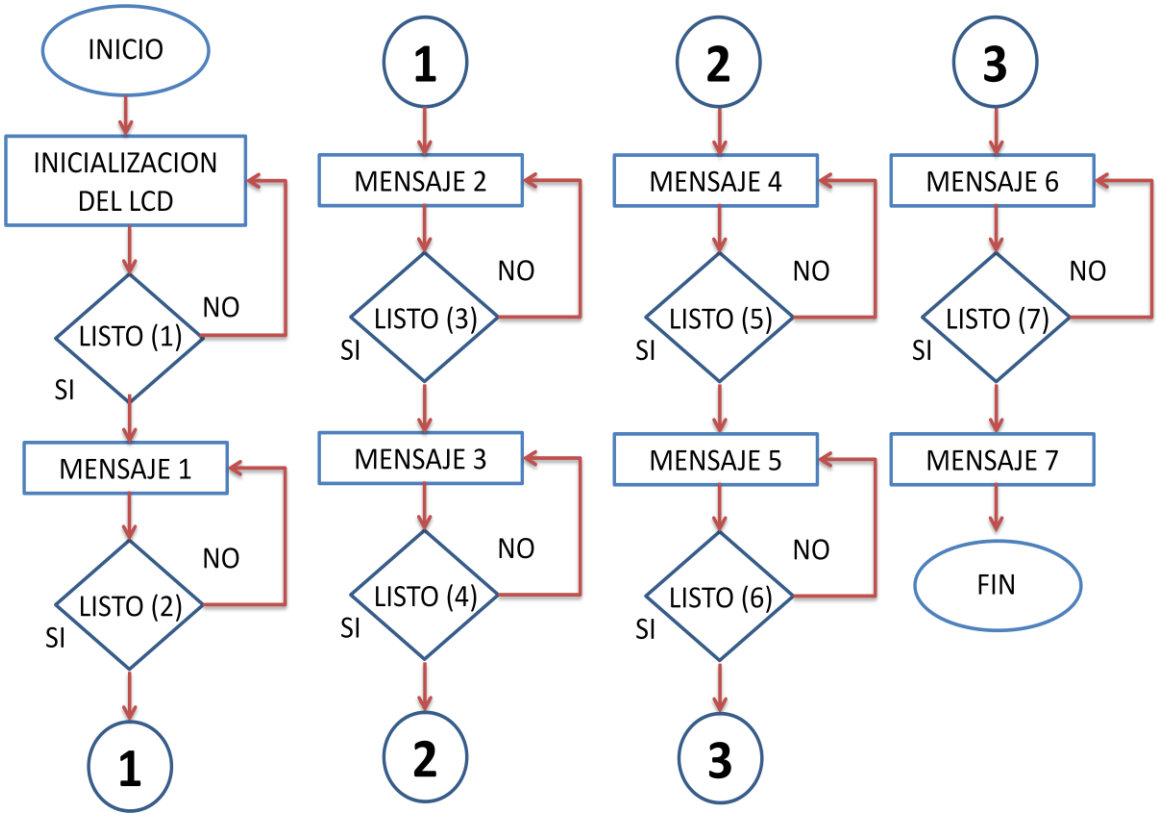


Figura 2.4.6.4: Diagrama a bloques

Por último se presenta el código en VHDL de nivel jerárquico superior, el cual es el encargado de mandar llamar a la rutina de inicialización y cada uno de los mensajes, así mismo a través de multiplexores se selecciona qué dato o comando y cuáles señales de control se enviaran al LCD en cada momento.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.componentes_lcd.all;

entity Mensajes is
    Port(
        CLK:   in      STD_LOGIC;
        RST:   in      STD_LOGIC;
        DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ):=X"0";
        LCD_E:  inout   STD_LOGIC:='0';
        LCD_RS: inout   STD_LOGIC:='0';
        LCD_RW: inout STD_LOGIC:='0';
        SF_CE0: out STD_LOGIC );
end Mensajes;

architecture Behavioral of Mensajes is

signal listo: std_logic_vector (7 downto 0);
signal LCD_E_1, LCD_E_2, LCD_E_3,LCD_E_4,LCD_E_5,LCD_E_6,LCD_E_7,LCD_E_8 : std_logic;
signal LCD_RS_1, LCD_RS_2, LCD_RS_3,LCD_RS_4,LCD_RS_5,LCD_RS_6,LCD_RS_7,LCD_RS_8 : std_logic;
signal LCD_RW_1, LCD_RW_2, LCD_RW_3,LCD_RW_4,LCD_RW_5,LCD_RW_6,LCD_RW_7,LCD_RW_8 : std_logic;
signal DATA_1, DATA_2, DATA_3, DATA_4, DATA_5,DATA_6,DATA_7,DATA_8 : std_logic_vector( 0 to 3 );


begin

SF_CE0 <= '1';  -- Inhabilitar acceso a StrataFlash
```

```
with listo select
LCD_E <= LCD_E_1 when "00000000",
         LCD_E_2 when "00000001",
         LCD_E_3 when "00000011",
         LCD_E_4 when "00000111",
         LCD_E_5 when "00001111",
         LCD_E_6 when "00011111",
         LCD_E_7 when "00111111",
         LCD_E_8 when "01111111",
         '0' when others;

with listo select
LCD_RS <= LCD_RS_1 when "00000000",
          LCD_RS_2 when "00000001",
          LCD_RS_3 when "00000011",
          LCD_RS_4 when "00000111",
          LCD_RS_5 when "00001111",
          LCD_RS_6 when "00011111",
          LCD_RS_7 when "00111111",
          LCD_RS_8 when "01111111",
          '0' when others;

with listo select
LCD_RW <= LCD_RW_1 when "00000000",
          LCD_RW_2 when "00000001",
          LCD_RW_3 when "00000011",
          LCD_RW_4 when "00000111",
          LCD_RW_5 when "00001111",
          LCD_RW_6 when "00011111",
          LCD_RW_7 when "00111111",
          LCD_RW_8 when "01111111",
          '0' when others;


with listo select
DATA <=  DATA_1 when "00000000",
         DATA_2 when "00000001",
         DATA_3 when "00000011",
         DATA_4 when "00000111",
         DATA_5 when "00001111",
         DATA_6 when "00011111",
         DATA_7 when "00111111",
         DATA_8 when "01111111",
         "0000" when others;

modulo1: LCD_init port map ( clk, rst,          LCD_E_1, LCD_RS_1, LCD_RW_1,listo(0),DATA_1);
modulo2: mensaje1 port map ( clk, rst,listo(0), LCD_E_2, LCD_RS_2, LCD_RW_2,listo(1),DATA_2);
modulo3: mensaje2 port map ( clk, rst,listo(1), LCD_E_3, LCD_RS_3, LCD_RW_3,listo(2),DATA_3);
modulo4: mensaje3 port map ( clk, rst,listo(2), LCD_E_4, LCD_RS_4, LCD_RW_4,listo(3),DATA_4);
modulo5: mensaje4 port map ( clk, rst,listo(3), LCD_E_5, LCD_RS_5, LCD_RW_5,listo(4),DATA_5);
modulo6: mensaje5 port map ( clk, rst,listo(4), LCD_E_6, LCD_RS_6, LCD_RW_6,listo(5),DATA_6);
modulo7: mensaje6 port map ( clk, rst,listo(5), LCD_E_7, LCD_RS_7, LCD_RW_7,listo(6),DATA_7);
modulo8: mensaje7 port map ( clk, rst,listo(6), LCD_E_8, LCD_RS_8, LCD_RW_8,listo(7),DATA_8);

end Behavioral;
```

Los códigos completos de los mensajes se incluyen en el apéndice A, para su consulta.

2.4.7 Módulo Control del FPGA (Máquina de Estados)

La función de la máquina es generar las señales necesarias para que los módulos del FPGA (GPS, Recep, PWM, Memoria R/W, Tran y Mensajes) se ejecuten en el orden requerido.

Las señales de salida de la máquina de estados que interactúan con cada módulo se resumen en la tabla 2.4.7.1.

Módulo	Señales requeridas	Tipo de salida de la señal requerida	Función de la señal en el módulo
Recep	Motor_Listo	Mealy (Pulso)	Habilita la recepción
GPS	Motor_Listo	Mealy (Pulso)	Habilita al módulo
	RST_INT	Mealy (Pulso)	Deshabilita al módulo
PWM	IND	Mealy (Pulso)	Cambia el %CU del PWM
	RST_INT	Mealy (Pulso)	Pone el %CU pequeño para que el motor no gire
Mensajes	men1	Moore(Nivel constante)	Si men1=1 se visualizan los mensajes 4 y 5 en el LCD
	men2	Moore(Nivel constante)	Si men2=1 se visualizan los mensajes 6 y 7 en el LCD
Memoria R/W	IND	Mealy (Pulso)	Cambia a la siguiente localidad de la memoria
	Guarda	Mealy (Pulso)	Almacena en la localidad actual
	RST_INT	Mealy (Pulso)	Selecciona la localidad 0 de la memoria
Tran	RST_INT	Mealy (Pulso)	Realiza la transmisión hacia LabVIEW

Tabla 2.4.7.1. Señales de salida de la máquina que interactúan con los diferentes módulos del FPGA.

En el diagrama de transiciones de estados (figura 2.4.7.1) se observa que se tienen salidas tanto de tipo Mealy (pulsos en el cambio de estado) como de tipo Moore (nivel constante mientras la máquina permanece en el estado actual).

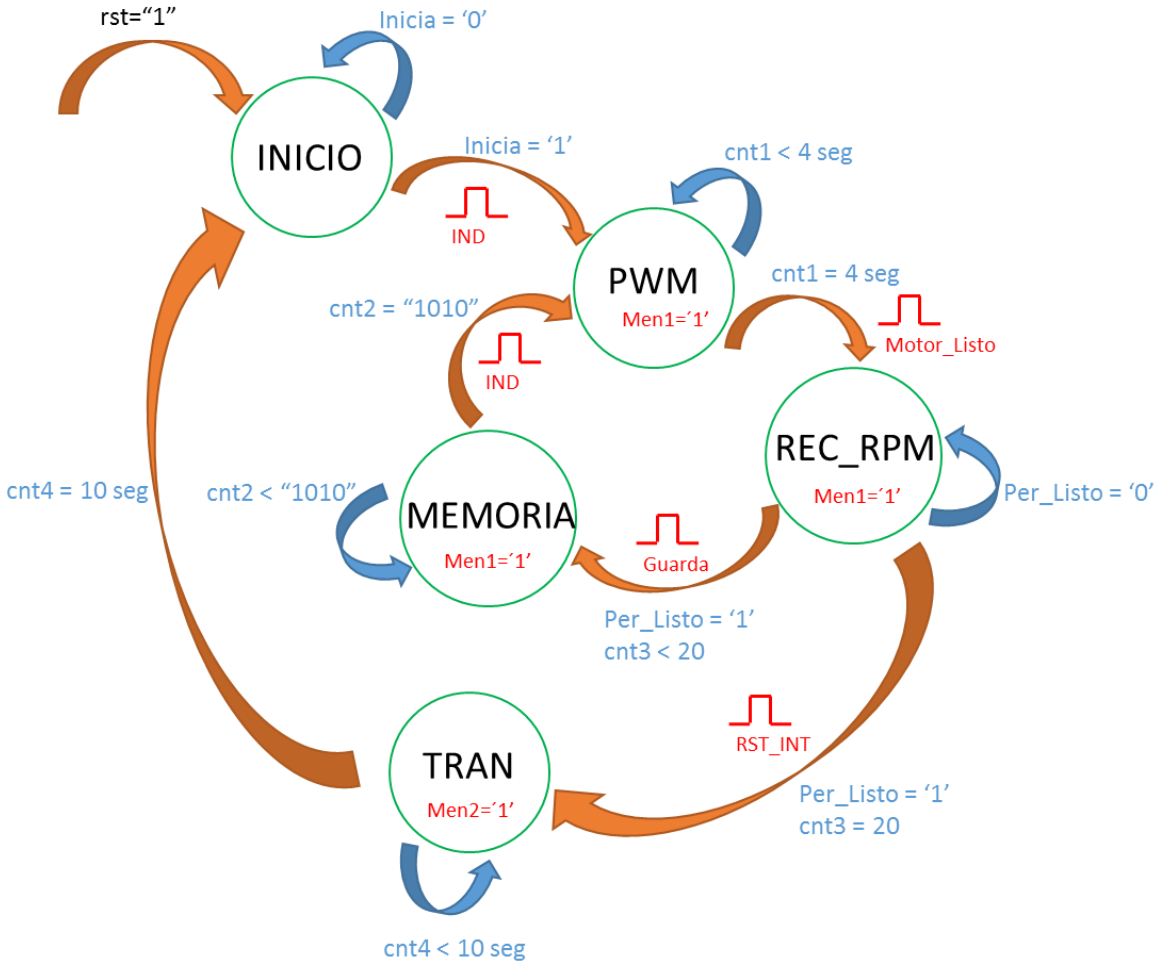


Figura 2.4.7.1: Diagrama de transiciones de estados

Breve explicación del diagrama de transiciones de estados

La máquina de estados permanece en el estado "INICIO" (en el cual sólo se ejecutan los mensajes de bienvenida del módulo "Mensajes") hasta que recibe el pulso "Inicia" (aplicado por el usuario con un push button). Cuando llega ese pulso, la máquina pasa al estado "PWM" generando el pulso "IND" en la transición de un estado al otro.

Inicia el Loop "PWM / REC_RPM / MEMORIA".

La máquina permanece en el estado “PWM” durante 4 segundos (para darle tiempo al motor de estabilizar el giro), una vez transcurrido este tiempo pasa al estado REC_RPM generando el pulso “Motor_Listo” durante la transición.

La máquina permanece en el estado “REC_RPM” hasta que la señal “Per_Listo” del módulo “GPS” sea 1, es decir, la máquina permanece en este estado hasta que se han terminado tanto la recepción del módulo “Recep” como el cálculo de la señal “periodo” del módulo “GPS”. Posterior a esto la máquina pasa al estado “MEMORIA” generando el pulso “Guarda” durante la transición de un estado al otro.

Una vez en el estado “MEMORIA” permanece ahí durante 10 ciclos de reloj de 20ns (cnt2=”1010”), esto se hace para permitir que el bloque “Memoria R/W” tenga un pequeño tiempo para almacenar de forma correcta. Transcurridos los 10 ciclos de reloj, la máquina pasa de nuevo al estado “PWM” generando el pulso “IND” durante la transición.

Se repite el ciclo “PWM / REC_RPM / MEMORIA” 20 veces (men1 es puesto en ‘1’ durante los 20 ciclos para desplegar en el LCD los mensajes correspondientes).

Termina el Loop “PWM / REC_RPM / MEMORIA”.

Pasando estos 20 ciclos, la máquina cambia al estado “TRAN” generando el pulso “RST_INT” durante la transición.

La máquina permanece en el estado “TRAN” hasta que se termina de enviar los datos a LabVIEW y los mensajes 6 y 7 del módulo “Mensajes” terminan de desplegarse en el LCD (10 segundos).

Finalmente la máquina vuelve al estado “INICIO”.

2.5 Bloque VA (Interfaz de LabVIEW)

Los programas realizados en LabVIEW se llaman instrumentos virtuales "Vls", ya que tienen la apariencia de los instrumentos reales, sin embargo, poseen analogías con funciones provenientes de lenguajes de programación convencionales. LabVIEW el cual es un sistema de desarrollo basado en la programación gráfica, orientada a desarrollar aplicaciones para instrumentación, integra una serie de librerías que son utilizadas para el desarrollo de este proyecto.

Gracias a esto se puede realizar la interfaz por medio de comunicación serial entre el FPGA y la PC, cabe resaltar que dicha comunicación se realiza a través del puerto serial. Mediante esta interfaz se puede observar la respuesta entregada del MEMS y puede ser graficada, la gráfica es explicada más adelante. En la figura 2.5.1 se muestra la caratula con la cual el usuario puede interactuar.

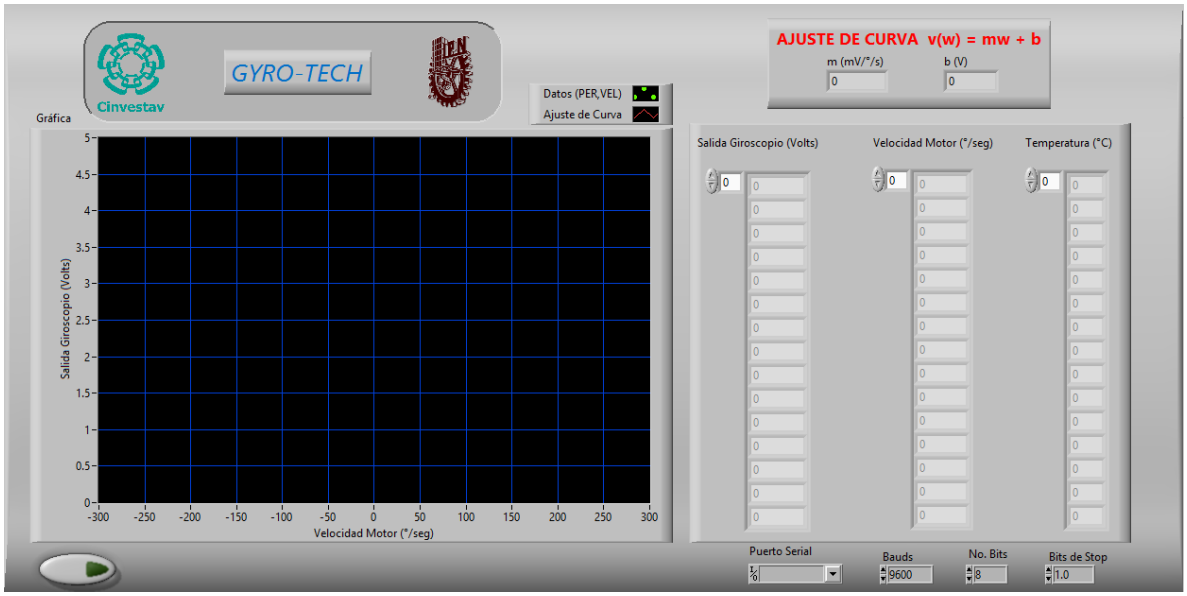


Figura 2.5.1: Interfaz del usuario en la PC

Para la elaboración de esta interfaz se requiere programar un VI en LabVIEW. En esta imagen, figura 2.5.2, se observa la programación que se requiere (diagrama a bloques), a gran escala se encuentra todo el programa dentro de un “Flat Sequence Structure”, el cual consiste en ejecutar de cuadro en cuadro empezando por el lado izquierdo, en este caso, primero el módulo de transmisión de datos, después la parte de acondicionamiento de nuestra información, posteriormente la parte esencial de generación de la gráfica y por último el módulo de "stop".

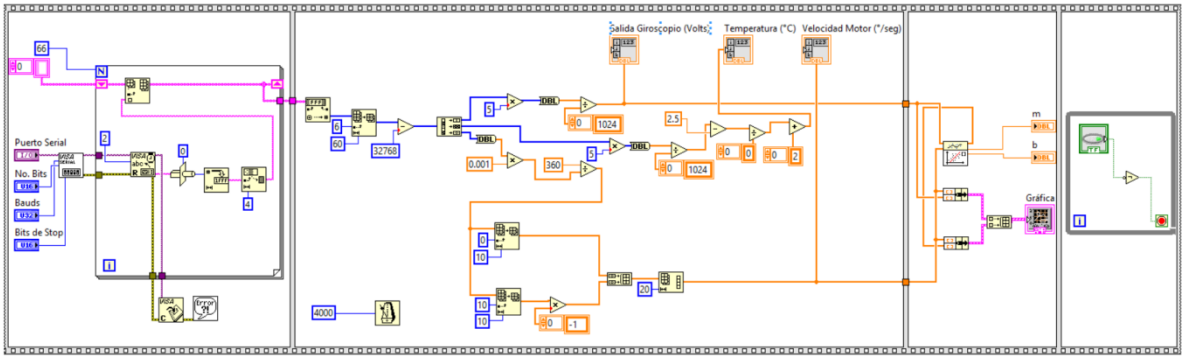


Figura 2.5.2: Panel de bloques

El primer cuadro, figura 2.5.3, cuenta con un bloque esencial para la comunicación serial, ya que sin este no se tiene conexión con la PC. El bloque llamado "VISA Configure Serial Port VI" es el encargo de configurar el puerto serial, configuraciones tales como: velocidad de transmisión, número de bits, bits de parada, bits de paridad, etc. En este caso sólo se configura lo siguiente: la Velocidad de Transmisión es de 9600 Bauds, el Numero de Bits se configura a 8 Bits y los Bits de Parada a 1 Bit.

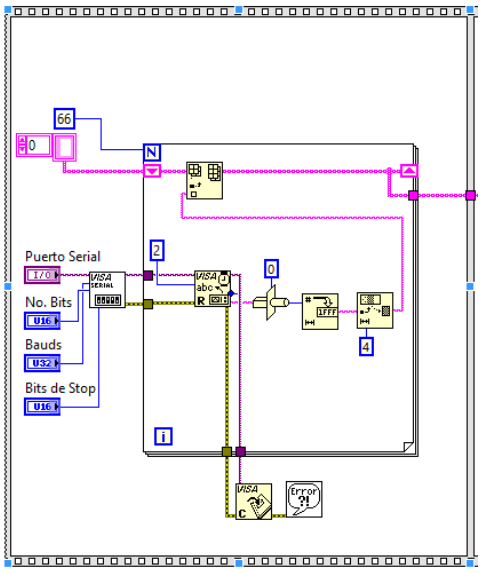


Figura 2.5.3: Cuadro 1 "comunicación serial"

Por consiguiente se necesita ahora la contraparte del primer bloque ("VISA Configure Serial Port VI"), el cual puede ser un bloque de Lectura o Escritura, ya que se necesita poder graficar los datos provenientes del FPGA. Se utiliza el bloque "VISA Read Function" el cual es el encargado de leer los datos provenientes del puerto.

De esta manera, teniendo los datos recibidos, se procede a convertirlos a hexadecimal mediante el bloque "Number To Hexadecimal String Function". Todos los bloques están dentro de un "for" con un valor constante de "66" ya que el proceso de recibir y guardar se realiza 66 veces, que es el número de valores que se reciben de parte del FPGA.

El siguiente cuadro, figura 2.5.4, es el encargado de acondicionar la señal para poder ser graficada posteriormente. Para esto es necesario primeramente convertir los valores entrantes hexadecimales a valores decimales para facilitar la realización de la gráfica, para esto se ocupa el bloque "Hexadecimal String To Numbre Function". Con dichos valores convertidos a Decimal se procede a crear el Arreglo de Valores de una Dimensión creándola mediante el bloque "Array Subset Function" a quien se le introduce valores de inicio (6) hasta el valor de tamaño del arreglo que se desea crear (60).

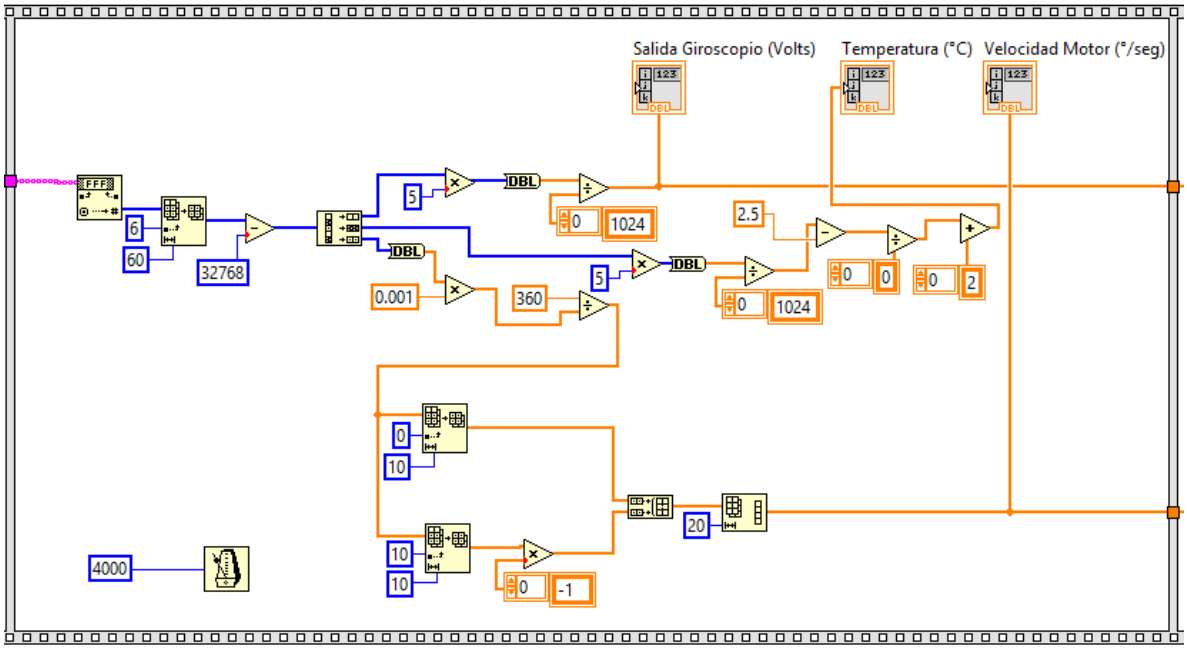


Figura 2.5.4: Cuadro 2 "acondicionamiento de valores"

Debido a problemas de lectura por parte de LabVIEW a la hora de mandar valores iniciando con "0", se toma la decisión de sumarle un "1" al inicio de toda la cadena de datos que se recibe, esto quiere decir que al llegar a el programa en LabVIEW es necesario restarle dicho valor el cual es: $2^{15} = 32768$. Logrando con esto poder leer correctamente los valores en decimal.

El siguiente bloque "Decimate 1D Array Function", figura 2.5.5, es de suma importancia ya que este es el encargo de separar los valores recibidos de parte de el arreglo que se crea en el FPGA, hay que recordar que dichos valores vienen intercalados (VEL-TEM-PER) y esto se logra con el bloque antes mencionado ya que divide la señal de entrada en salidas de n+1, logrando con esto tener solo valores de VEL en un arreglo, valores TEM en otro y PER en otro más.

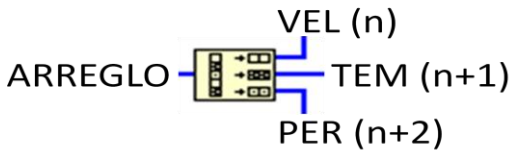


Figura 2.5.5: Bloque " Decimate 1D Array Function"

Teniendo separados los valores, se puede trabajar cada señal individualmente y hacer el acondicionamiento que necesite cada una, para después ser graficada. Se inicia con la señal TEM, esta señal es de Temperatura (25°C aproximadamente) y se logra calcular de la siguiente manera. Primero dicho valor proveniente del arreglo se tiene que convertir a un valor de *voltaje analógico*, ya que viene en un valor digital, para poder aplicar la fórmula de conversión a grados Celsius. (Recordar que este valor sólo servirá como valor extra que arroja el MEMS, es decir en esta fase del proyecto no se utiliza, pero sí se considera para poder utilizarlo, en un futuro, en la fase 2).

$$V_{Tout} = \frac{5\,V_{DIG}}{2^{10}} \qquad T = \frac{V_{Tout} - 2.5}{0.0084} + 25$$

Para el arreglo con valores de VEL, se parte de la siguiente ecuación, la cual se utiliza para convertir el valor digital en un *voltaje de salida analógico*, teniendo en cuenta que dichos valores del arreglo provienen del microcontrolador, con un voltaje de referencia de 5 Volts y una resolución de 10 bits:

$$V_{OUT} = \frac{5\,V_{DIG}}{2^{10}}$$

A continuación se muestra el diagrama de bloques en LabVIEW del acondicionamiento de la señal VEL, figura 2.5.6.

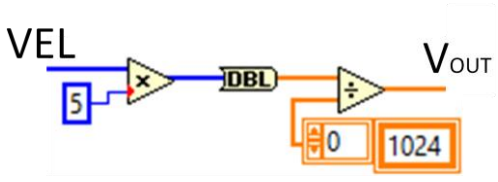


Figura 2.5.6: Diagrama de bloques "acondicionamiento VEL"

Un punto importante de atender en esta parte, es el hecho de transformar los valores enteros del arreglo a valores flotantes, ya que sin esto la operación siguiente (una división) no puede ser realizada.

Ahora sólo falta analizar el último valor llamado PER, mediante el cual se grafica la velocidad angular con dicho arreglo, pero antes se debe de acondicionar el arreglo con la siguiente fórmula:

$$w\ (°/seg) = \frac{360°}{("periodo" * 1\mu s)}$$

(“periodo” es el valor obtenido en el módulo GPS del FPGA)

Dicha fórmula se logra programar en LabVIEW con los bloques que a continuación se muestran, figura 2.5.7.

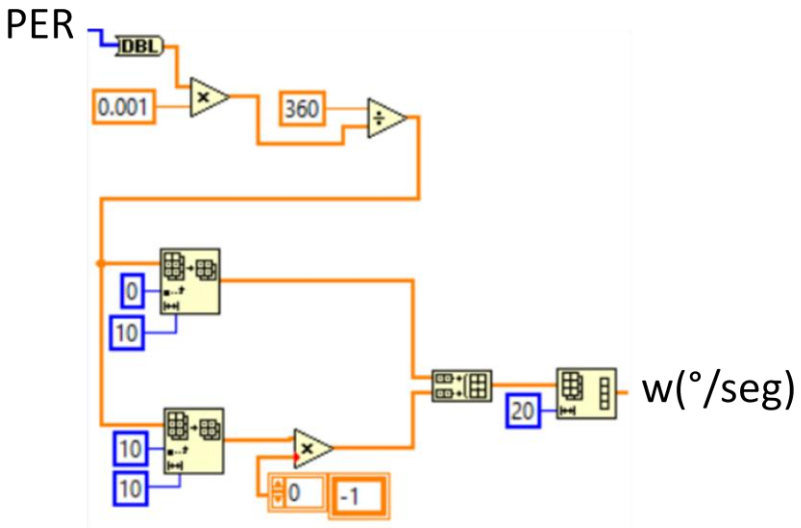


Figura 2.5.7: Diagrama de bloques "acondicionamiento PER"

Como se puede observar, los bloques iniciales del acondicionamiento es la aplicación de la formula antes mencionada, posteriormente el arreglo se divide en 2 para lograr poner a los valores de -300° a -30° el signo negativo (-), esto se logra utilizando el bloque "Array Subset Function". Ahora, de modo contrario a lo anterior, se procede a unir el arreglo para así poder graficar. El bloque para la tarea de crear el arreglo de nuevo es "Build Array Function".

Para el siguiente cuadro, finalmente se puede proceder a realizar la gráfica de datos de las señales (PER, VEL) y con esto poder realizar un ajuste de curva. Para realizar la tarea del ajuste de curva es necesario un solo bloque llamado "Linear Fit VI" y posteriormente se unen líneas y se grafican en el bloque "XY Graph", figura 2.5.8.

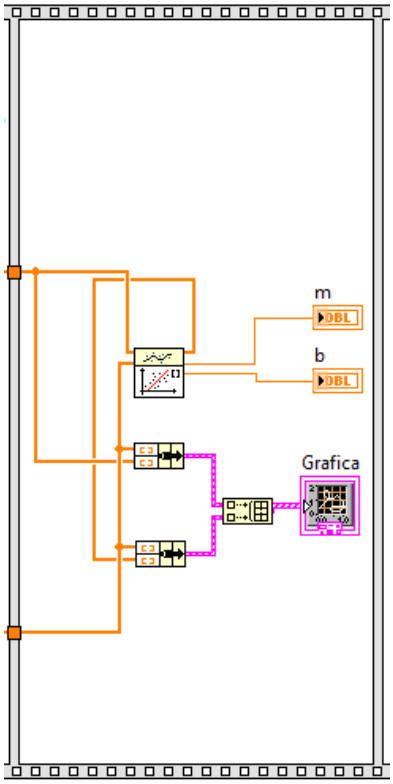


Figura 2.5.8: Cuadro 3 "gráfica de valores (PER, VEL) y ajuste de curva"

En el último cuadro se tiene el bloque de "Stop" cuya función es detener el programa cuando se desee.

CAPÍTULO 3

Pruebas y Resultados

Para las pruebas se utiliza el siguiente equipo de laboratorio

- Multímetro: Fluke 177
- Osciloscopio: Tektronix MSO 2012
- Fuente de alimentación: Tektronix TM5003
- Generador de señales: Tektronix AFG3021G
- Puntas de osciloscopio: Tektronix P2221
- Puntas probador lógico: Tektronix P6316

3.1 Bloque ACT (Adquisición, Conversión y Transmisión)

Si se aplica una tensión V_A a la entrada del convertidor analógico a digital, entonces el voltaje digital (V_{Dig}) de la conversión es:

$$V_{Dig} = \frac{(2^{10} - 1) * V_A}{V_{ref}}$$

Donde V_{ref} =voltaje de referencia del ADC

Finalmente este valor decimal se debe convertir a binario.

Para esta prueba se llevan a cabo los siguientes pasos:

1) Se mide el voltaje que entra a la terminal AREF del microcontrolador

$$V_{ref} = 5.032$$

2) Se aplica a la entrada de los dos canales del convertidor 0 Volts.

Se mide, con la ayuda de un osciloscopio, la trama que está siendo transmitida de forma serial en la terminal tres del ATMega48 (TXD). El resultado se muestra en la figura 3.1.1.

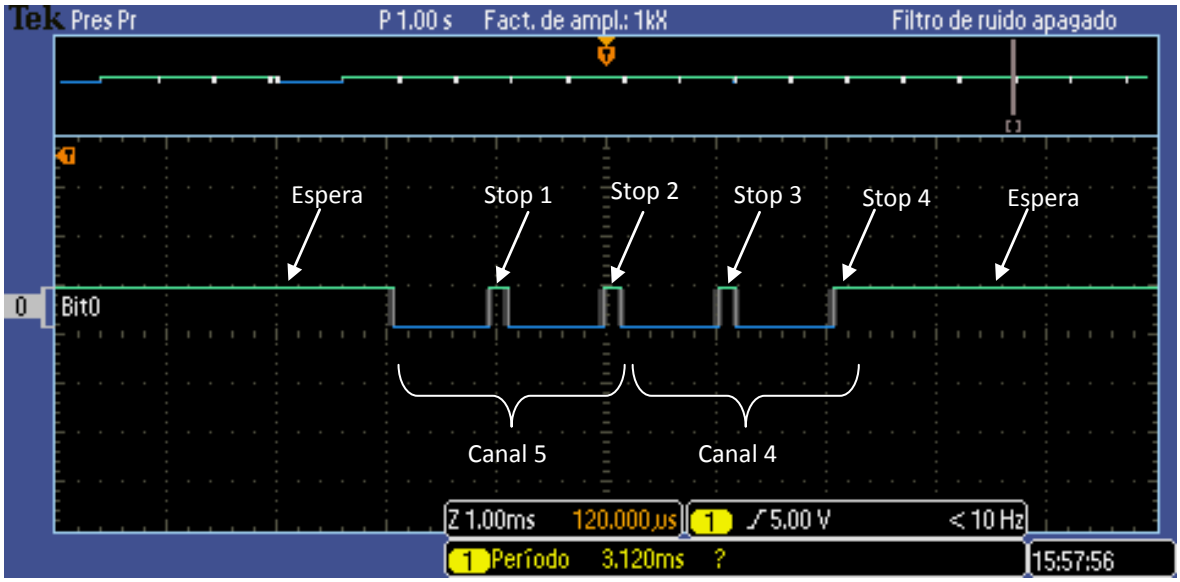


Figura 3.1.1: Se muestra una de las tramas transmitida por el microcontrolador al aplicarle 0 Volts a la entrada de los canales 4 y 5 del ADC.

Tal como se espera, todos los bits de datos de ambos canales se encuentran en cero.

3) Se realiza la misma prueba, ahora se le aplica 5.031 V a la entrada de los canales del ADC (la tensión de referencia del microcontrolador). La trama transmitida en ese momento es la mostrada en la figura 3.1.2.

Para este caso el valor digital esperado es

$$V_{Dig\ esperado} = \frac{(2^{10}-1) * V_A}{5.031\ Volts} = \frac{(2^{10}-1) * 5.031\ Volts}{5.031\ Volts} = 1023$$

Convertido a binario

$$V_{Dig\ esperado} = 11\ 11111111$$

Es decir, los diez bits de la conversión deben estar en estado alto.

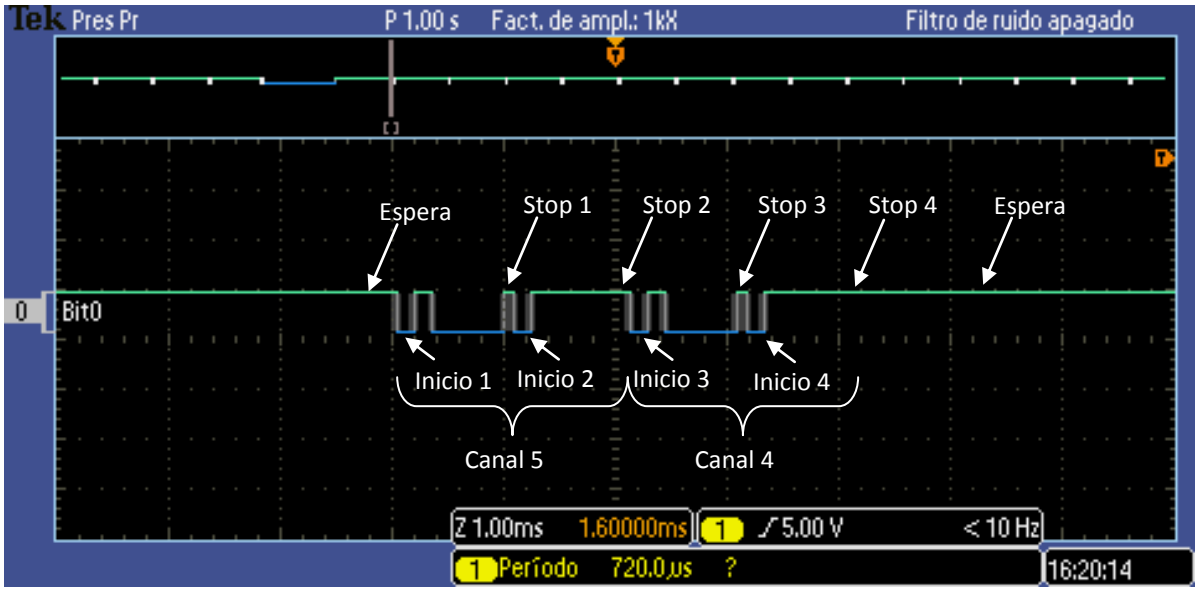


Figura 3.1.2. Se muestra una de las tramas enviada por el microcontrolador al aplicarle la tensión de referencia a la entrada de los canales 4 y 5 del ADC.

En la figura 3.1.2 se observa que después del bit de inicio de la primera trama (Inicio1) hay dos bits en estado alto (b8 y b9) y posterior a estos se encuentran seis ceros. Después del bit de inicio de la segunda trama (Inicio2) se encuentran los ocho bits menos significativos de la conversión del canal 5 en estado alto (b0 - b7), seguidos del bit de parada de la segunda trama (stop 2). Esto mismo se repite en el canal 4.

4) Se coloca el MEMS sobre una base giratoria cuya velocidad es de 7.2 °/s. Antes de hacerla girar se mide el voltaje entregado por el MEMS, en la terminal RATEOUT, cuando se encuentra estático.

$$V_{estático} = 2.35\ V$$

Teniendo este voltaje, se hace girar la base en el sentido opuesto a las manecillas del reloj. A dicha velocidad y en el sentido mencionado, se espera tener un voltaje de 2.314 V en la terminal RATEOUT del MEMS. Este valor se obtiene de:

$$V_{Rateout\ esperado} = V_{estático} - (5mv * 7.2°/s) = 2.35V - (5mv * 7.2°/s)$$

$$V_{Rateout\ esperado} = 2.314\ \text{V}$$

En la figura 3.1.3 se observa la captura de pantalla del osciloscopio del resultado de esta prueba.

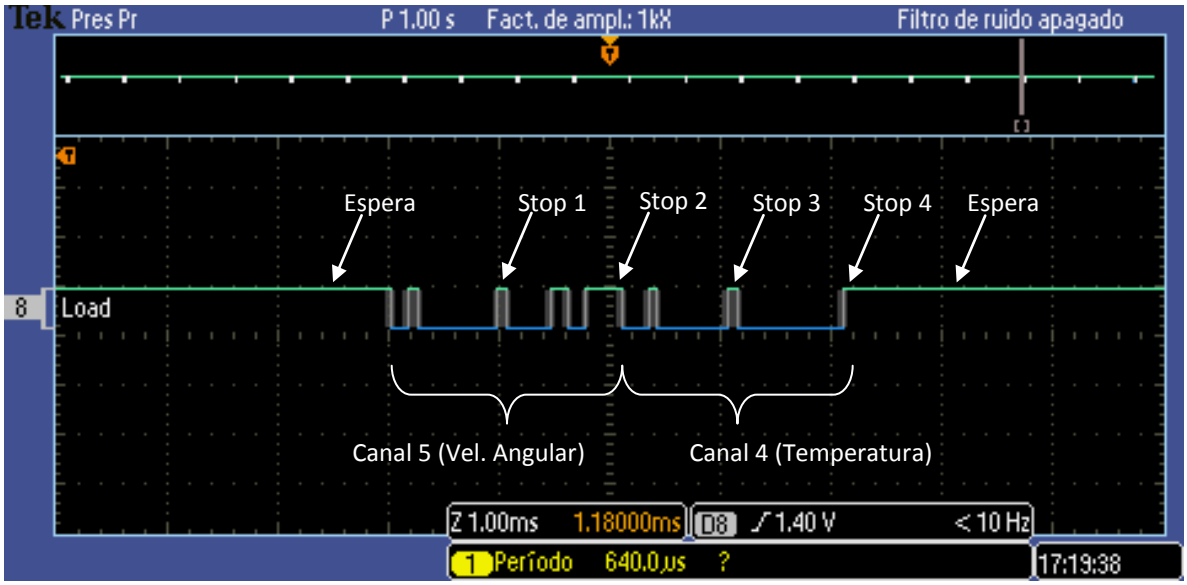


Figura 3.1.3: Se muestra una de las tramas enviada por el microcontrolador. En el canal 5 se envía la conversión de la velocidad (su equivalencia en volts). En el canal 4 se envía la conversión de la temperatura (su equivalencia en volts).

Los bits observados en la trama para el canal 5 son:

0 ***10000000*** 1 0 ***00011011*** 1

Los bits marcados en negritas son los bits de datos, los otros dos 1s son bits de parada (stop1 y stop2 de la figura 3.1.3) y el restante par de 0s son los bits de inicio 1 y 2.

Tomando en cuenta que las palabras de las tramas enviadas contienen la información acomodada tal como se muestra en la figura 3.1.4, se reacomodan los bits v9 a v0 medidos con el osciloscopio y se tiene que la conversión transmitida es

00000**01 11011000**

Transformando este valor, se tiene que el voltaje convertido y transmitido de la velocidad es

$$V_{Rateout} = 5.031\left(\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^6}+\frac{1}{2^7}\right)$$

$$V_{Rateout} = 2.318\,V$$

El cual es muy próximo al $V_{Rateout\ esperado} = 2.314\,\mathrm{V}$ que se aplica a la entrada del canal 5 del ADC. Sólo existe una diferencia de 4mV.

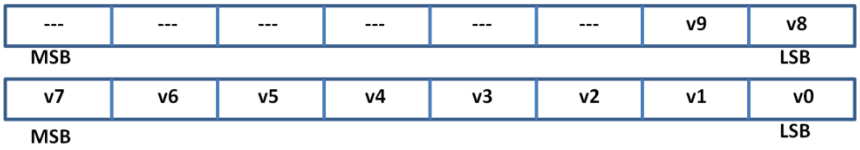


Figura 3.1.4. Se muestra la forma en que está acomodada la información de la conversión del ADC de la velocidad. Donde v9 es el bit más significativo de la conversión y v0 el de menor peso.

Debido a los resultados obtenidos en estas pruebas realizadas, se puede decir que el bloque de adquisición, conversión y transmisión está trabajando de forma correcta.

3.1.1 Módulos de RF

Se conecta el módulo transmisor a la terminal TXD del microcontrolador.

El módulo receptor se conecta en una protoboard. Sólo se conecta la alimentación de 5V y un cable en la terminal DATA para vigilar las señales recibidas.

Se aplican 0V a los canales 4 y 5 del ADC. Con la ayuda del osciloscopio se observa tanto la trama transmitida como la recibida en los módulos.

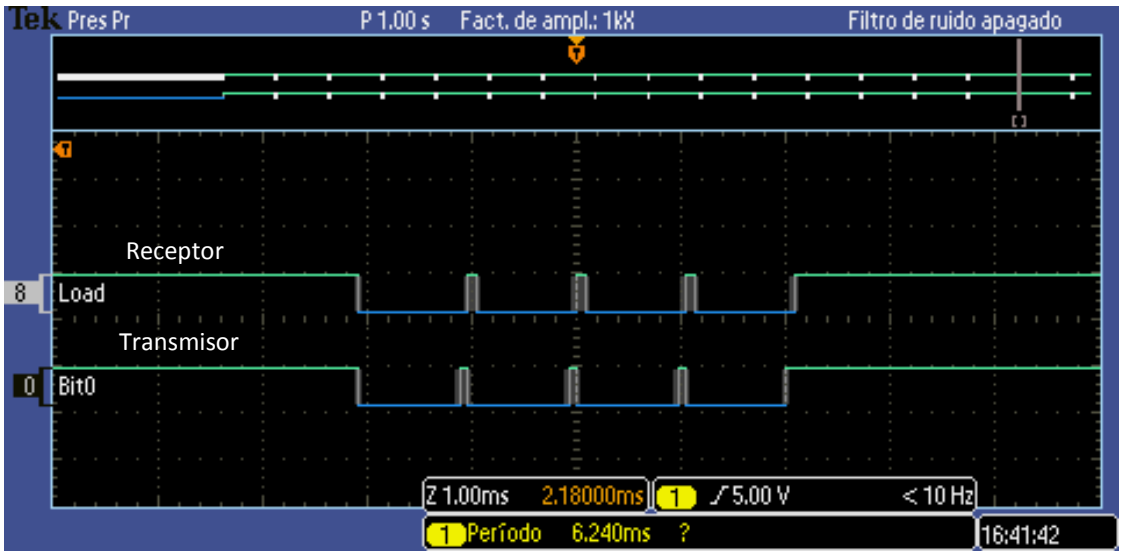


Figura 3.1.1.1: Se muestran las tramas transmitida y recibida para una conversión de 0 V a la entrada de los canales 4 y 5.

Se realiza otra prueba similar, sólo que ahora se aplican 5.031V a la entrada de ambos canales del ADC

El resultado se muestra en la figura 3.1.1.2.

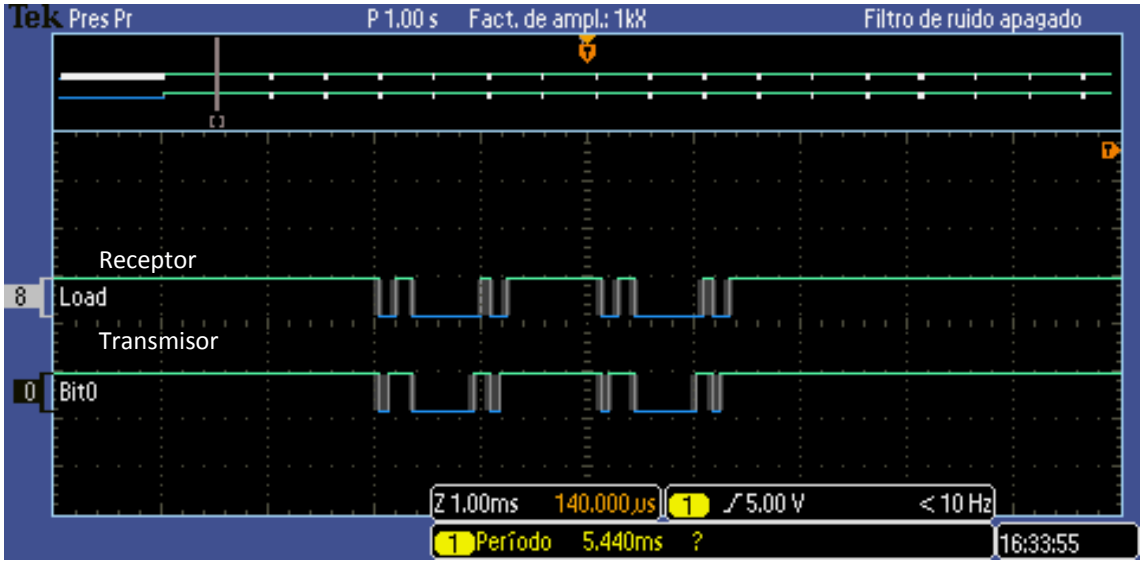


Figura 3.1.1.2: Se muestran las tramas transmitida y recibida para una conversión de 5.031V a la entrada de los canales 4 y 5.

En las figuras 3.1.1.1 y 3.1.1.2 se observa que tanto las tramas transmitidas como las recibidas son las mismas, sólo que un poco desfasadas (la recibida está atrasada con respecto a la transmitida), por lo cual se puede decir que el bloque “Módulos de RF” está funcionando correctamente.

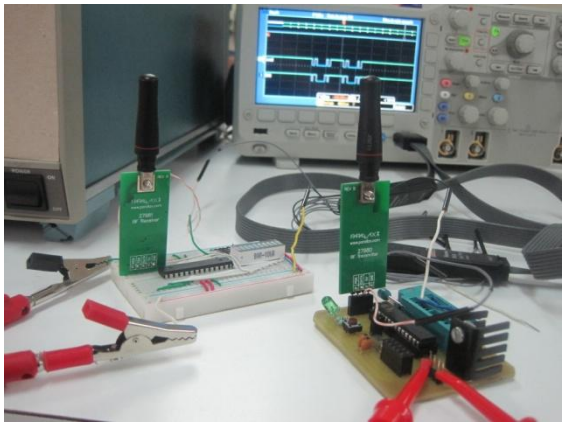


Figura 3.1.1.3. Fotografía tomada en el laboratorio de la prueba realizada para verificar el funcionamiento de los módulos receptor y transmisor

3.2 Bloque BG (Base Giratoria)

Se alimenta el puente H con 12 Volts y se hace un barrido de %C.U. del PWM aplicado. Con esto se tiene una idea de cómo incrementa la velocidad angular del motor con respecto al %C.U.. Se realiza la prueba para ambos sentidos de las manecillas del reloj y unaa frecuencia del PWM de 1kHz.

Para calcular la velocidad es necesario medir el tiempo que tarda en dar una vuelta el eje del motor y esto se logra con la ayuda del encoder.

Con el osciloscopio se mide el periodo de la señal entregada por el encoder. Teniendo este tiempo, se calcula la velocidad angular de la siguiente manera.

$$\omega = \frac{360°}{t}$$

En la tabla 3.3.1 se muestran los resultados a partir de 9 %C.U. ya que con menor porcentaje de ciclo útil el motor no logra girar.

Para 32 %C.U. el motor gira a aproximadamente 300°/s, que es la máxima velocidad a la que se puede hacer girar, ya que el MEMS no mide más allá de dicha velocidad.

De esta tabla se seleccionan 10 valores de %CU para ser utilizados en el proyecto (se busca que sean tales que la diferencia de velocidad a la que gira el motor sea similar entre cada %CU). Estos son: 9, 11, 14, 16, 19, 22, 24, 26, 28 y 30%CU (los subrayados en la tabla 3.2.1).

La corriente consumida por el motor para estas pruebas es:

$$I_{min} = 0.035\ Amp$$

$$I_{máx} = 0.3\ Amp$$

Estas mediciones son realizadas con la tarjeta del bloque ACT encima del disco que sostiene el eje del motor.

Giro en sentido de las manecillas del reloj			Giro en sentido opuesto a las manecillas del reloj		
%CU	t(s)	$\omega\ (°/s)$	%CU	t(s)	$\omega\ (°/s)$
9	**9.478**	**37.9827**	**9**	**10.04**	**35.85657**
10	7.47	48.19277	10	7.712	46.6805
11	**6.07**	**59.30807**	**11**	**6.43**	**55.98756**
12	5.18	69.49807	12	5.47	65.81353
13	4.46	80.71749	13	4.74	75.94937
14	**3.85**	**93.50649**	**14**	**4.18**	**86.1244**
15	3.43	104.9563	15	3.742	96.20524
16	**3.083**	**116.7694**	**16**	**3.37**	**106.8249**
17	2.808	128.2051	17	3.075	117.0732
18	2.585	139.265	18	2.86	125.8741
19	**2.364**	**152.2843**	**19**	**2.635**	**136.6224**
20	2.19	164.3836	20	2.44	147.541
21	2.04	176.4706	21	2.28	157.8947
22	**1.9**	**189.4737**	**22**	**2.12**	**169.8113**
23	1.767	203.7351	23	1.974	182.3708
24	**1.665**	**216.2162**	**24**	**1.836**	**196.0784**
25	1.565	230.0319	25	1.736	207.3733
26	**1.495**	**240.8027**	**26**	**1.64**	**219.5122**
27	1.42	253.5211	27	1.567	229.7384
28	**1.36**	**264.7059**	**28**	**1.5**	**240**
29	1.3	276.9231	29	1.446	248.9627
30	**1.25**	**288**	**30**	**1.33**	**270.6767**
31	1.225	293.8776	31	1.28	281.25
32	1.15	313.0435	32	1.23	292.6829

Tabla 3.2.1: Resultados obtenidos de la caracterización del motor

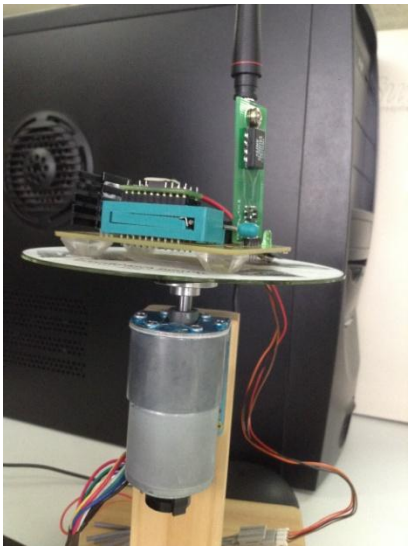


Figura 3.2.1. Fotografía del motor montado para realizar la caracterización.

3.2.1 Encoder (Interruptor óptico H21B1)

Para probar que el encoder funciona de manera correcta se arma el siguiente circuito.

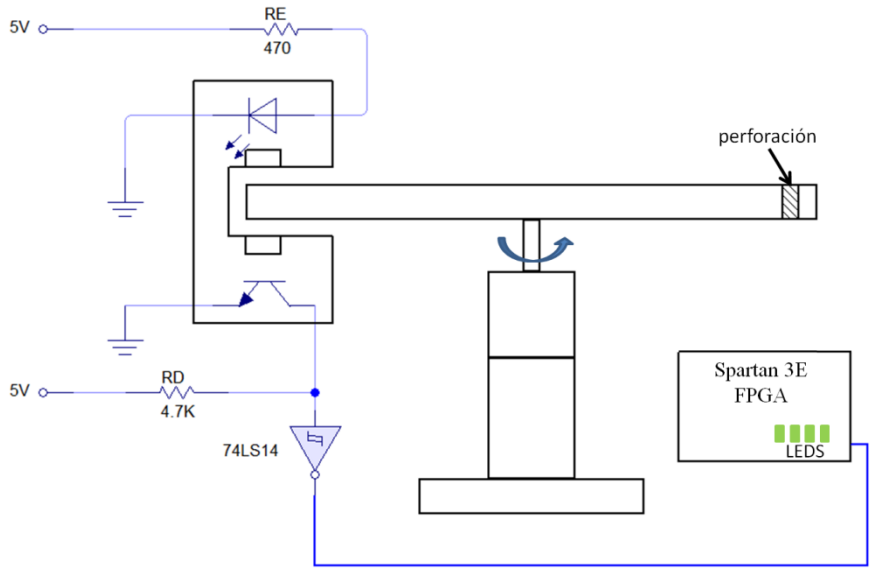


Figura 3.2.1.1: Se utiliza un disco con una perforación (para permitir el paso de la luz del LED hacia el fototransistor) montado en un motor que gira a velocidad constante.

Se utiliza la señal a la salida del inversor para hacer un contador en el FPGA, el cual se incrementa con flancos de subida. Las cuentas se observan en los LEDs. El proceso en VHDL es el siguiente

```
---CONTADOR
process (rst,SinReb)
begin
   if rst='1' then
      leds <= "0000";
   elsif SinReb'event and SinReb='1' then
         leds <= leds + 1;
   end if;
end process;
```

A la salida del inversor se vigila con el osciloscopio y pareciera no tener rebotes (véase figura 3.2.1.2), sin embargo el contador no cuenta de manera correcta, hay ocasiones en las que lo hace de uno en uno, pero en otras se salta cuentas, es decir, sigue habiendo rebotes.

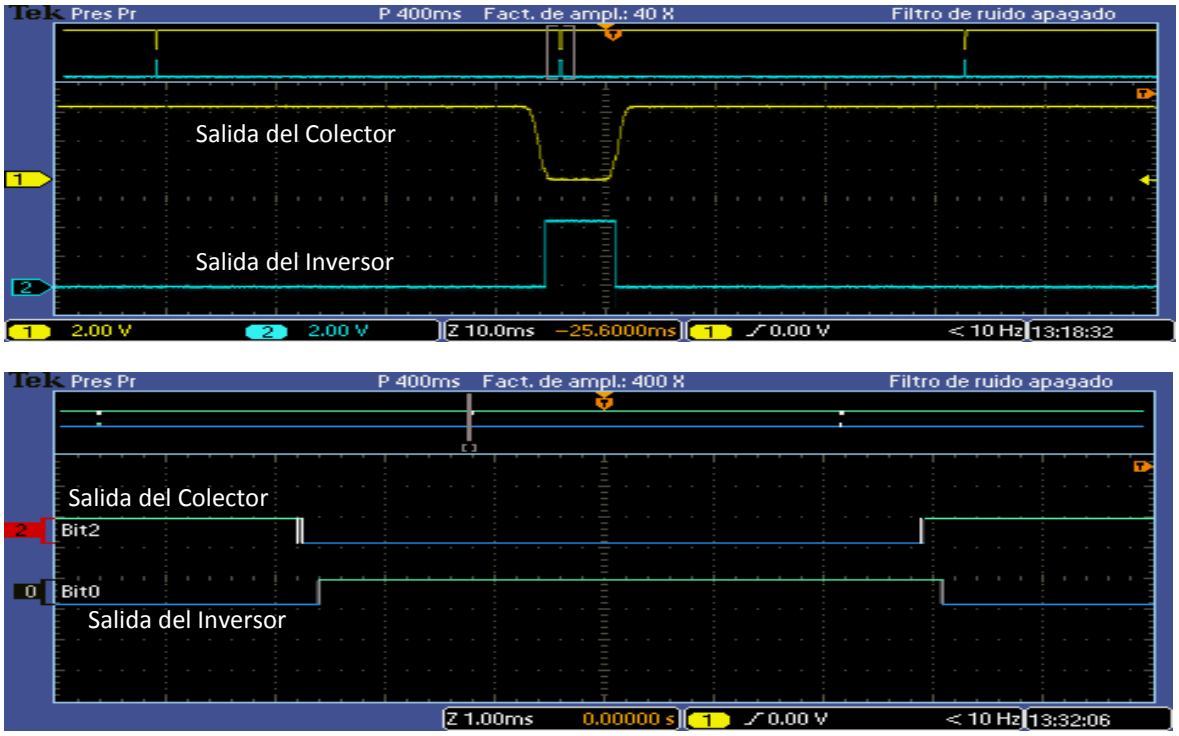


Figura 3.2.1.2: Capturas de pantalla del osciloscopio de las señales a la salida del encoder a) Analógico b) Digital.

Para eliminar los rebotes faltantes se emplea un anti rebote mediante software. El proceso anti rebotes en VHDL es el siguiente.

```
process (CLK,CNT,CAPTURA)
  begin
    if CAPTURA = '0' then
      CNT <= "000";
    elsif (CLK'event and CLK = '1') then
      if (CNT /= "010") then
      CNT <= CNT + 1;
      end if;
    end if;
    if (CNT = "001") and (CAPTURA = '1') then
      CAP <= '1';
    else
      CAP <= '0';
    end if;
end process;
```

Con esto el contador funciona de forma correcta, por lo cual se puede utilizar este circuito en el bloque GPS para la medición del tiempo que tarda en dar una vuelta el motor.

3.3 Bloque de control

Los resultados de cada módulo implementados en el FPGA se muestran a continuación.

3.3.1 Módulo Recep del FPGA

Con la ayuda del bloque ACT se envía de forma serial la conversión analógica a digital de voltajes constantes (el mismo voltaje para ambos canales) y se utilizan los LEDs con los que cuenta la tarjeta de desarrollo del FPGA para saber si se están almacenando bien estos valores. Como sólo cuenta con 8 LEDs, se utiliza uno de los interruptores, de la misma tarjeta, para seleccionar si se quiere ver los bits más significativos o los menos significativos de alguno de los arreglos (Velrx o Temrx), sin importar cuál de ellos sea ya que ambos están recibiendo la misma información.

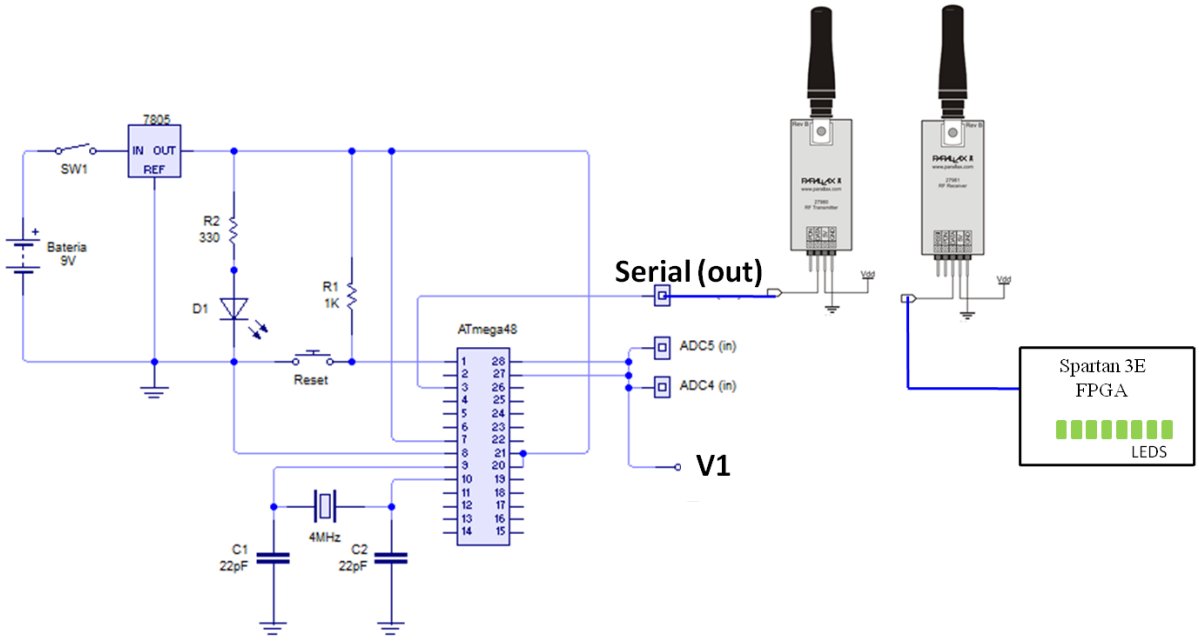


Figura 3.3.1.1: Circuito utilizado para probar el bloque Recep. *V1* es una fuente variable de voltaje que se aplica tanto a ADC4 como a ADC5. *Serial* se conecta al módulo transmisor de RF. La señal recibida por el modulo receptor se conecta a la entrada "rx" del FPGA.

Se utiliza un push button para emular el pulso *"Motor_Listo"* (proveniente de la máquina de estados) que se necesita para habilitar la recepción.

Los resultados obtenidos se encuentran registrados en la tabla 3.3.1.1.

V1 (Volts)	Datos observados en los LEDs	Conversión a analógico (Volts)
1	0011001001	0.98753027
1.1	0011011110	1.09070508
1.2	0011110010	1.1889668
1.3	0100000110	1.28722852
1.4	0100011100	1.39531641
1.5	0100101110	1.48375195
1.6	0101000011	1.58692676
1.7	0101010111	1.68518848
1.8	0101101100	1.78836328
1.9	0110000001	1.89153809
2	0110010101	1.9897998
2.1	0110101001	2.08806152
2.2	0110111101	2.18632324
2.3	0111010010	2.28949805
2.4	0111100111	2.39267285
2.5	0111111011	2.49093457
2.6	1000010000	2.59410938
2.7	1000100100	2.69237109
2.8	1000111000	2.79063281
2.9	1001001101	2.89380762
3	1001100001	2.99206934
3.1	1001110110	3.09524414
3.2	1010001010	3.19350586
3.3	1010011110	3.29176758
3.4	1010110011	3.39494238
3.5	1011000111	3.4932041
3.6	1011011100	3.59637891
3.7	1011110000	3.69464063
3.8	1100000100	3.79290234
3.9	1100011001	3.89607715
4	1100101111	4.00416504

Tabla 3.3.1.1. Resultados obtenidos del almacenado en un arreglo en el FPGA de los datos de las conversiones del bloque ACT.

La tercera columna de esta tabla contiene la conversión analógica de lo observado en los LEDs. Se puede ver que es muy similar al voltaje transmitido.

Para convertir un valor digital a analógico se utiliza la siguiente ecuación

$$V_{analógico} = V_{ref}\left(\frac{b_0}{2^{10}} + \frac{b_1}{2^9} + \frac{b_2}{2^8} + \frac{b_3}{2^7} + \frac{b_4}{2^6} + \frac{b_5}{2^5} + \frac{b_6}{2^4} + \frac{b_7}{2^3} + \frac{b_8}{2^2} + \frac{b_9}{2^1}\right)$$

Sólo se sustituye el valor binario de b0 a b9 en la ecuación, donde b9 es el bit más significativo y b0 el menos significativo.

Previamente el bloque ACT, el receptor y transmisor de RF ya estaban funcionando de forma correcta, así que en base a los resultados obtenidos en esta prueba, se concluye que el módulo Recep cumple con su función.

Figura 3.3.1.2: Se observa en los LEDs los 8 bits menos significativos del arreglo que contiene la conversión digital de 1 volt.

3.3.2 Módulo GPS del FPGA

Se aplica a la entrada “IR” del FPGA una señal cuadrada de frecuencia 1Hz, figura 3.3.2.1. El resultado obtenido de “periodo” se despliega en el LCD con el que cuenta la tarjeta del FPGA y es el mostrado en la figura 3.3.2.2.

Figura 3.3.2.1. Fotografía de la pantalla del generador de funciones que proporciona una señal cuadrada de 5V y frecuencia de 1Hz.

Figura 3.3.2.2. Se muestra en el LCD el valor de la señal periodo en hexadecimal para una señal de entrada “IR” de 1Hz.

Se realiza esta misma prueba cambiando la frecuencia de la señal cuadrada. Los resultados se encuentran registrados en la tabla 3.3.2.1. Para confirmar que la señal “periodo” tiene un valor correcto, se multiplica por 1ms (tercera columna de la tabla 3.3.2.1). Para emular la señal *“Motor_Listo”* proveniente de la máquina de estados se utiliza un push button, este se presiona cada vez que se desea habilitar el módulo GPS.

Periodo de la señal cuadrada (s)	Valor de la señal "periodo" en Hexadecimal	Periodo Calculado (s) T="periodo" x 1ms
1	3E8	1
1.5	5DC	1.5
2	7D0	2
2.5	9C4	2.5
3	BB8	3
3.5	DAC	3.5
4	FA0	4
4.5	1194	4.5
5	1388	5
5.5	157C	5.5
6	1770	6
6.5	1964	6.5
7	1B58	7
7.5	1D4C	7.5
8	1F40	8
8.5	2134	8.5
9	2328	9
9.5	251C	9.5
10	2710	10
10.5	2904	10.5

Tabla 3.3.2.1. Resultados de la prueba para diferentes valores de frecuencia de la señal "IR" proporcionada por el generador de funciones.

Como se puede observar en la tabla anterior, el cálculo de "periodo" es correcto.

La siguiente prueba realizada es utilizar el encoder como auxiliar para medir el tiempo que tarda el eje del motor en dar una vuelta. Se realiza la prueba para diferentes velocidades (diferentes %C.U. del PWM aplicado al puente h del motor). La señal del encoder es aplicada a la entrada "IR" del FPGA.

Los resultados se muestran en la tabla 3.3.2.2.

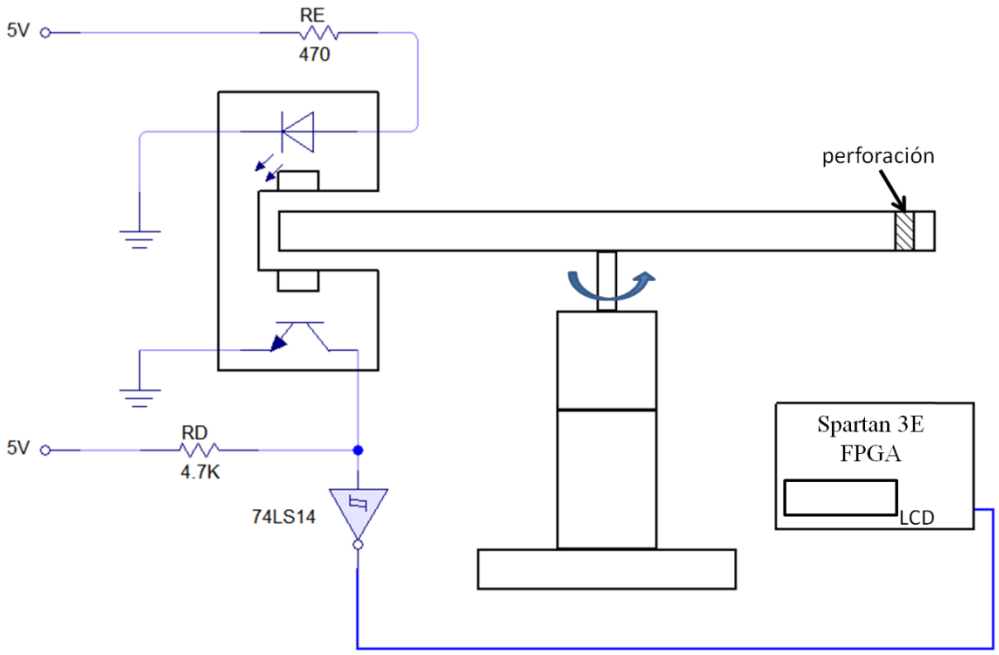


Figura 3.3.2.3. Circuito utilizado que une el encoder con el módulo GPS. La señal a la salida del encoder es conectada a la entrada "IR" del FPGA.

	Señal "periodo"					T="periodo" x 1ms				
%CU	T1 (hex)	T2 (hex)	T3 (hex)	T4 (hex)	T5 (hex)	T1 (seg)	T2 (seg)	T3 (seg)	T4 (seg)	T5 (seg)
9	291D	28F3	28CF	2825	2776	**10.525**	**10.483**	**10.447**	**10.277**	**10.102**
11	1A53	19C0	1986	197F	1999	**6.739**	**6.592**	**6.534**	**6.527**	**6.553**
14	10C1	10B3	10A7	10B4	10A8	**4.289**	**4.275**	**4.263**	**4.276**	**4.264**
16	D43	D46	D2F	D38	D24	**3.395**	**3.398**	**3.375**	**3.384**	**3.364**
19	9A8	9BD	99D	992	991	**2.472**	**2.493**	**2.461**	**2.45**	**2.449**
22	7B5	7BC	7A7	7A1	7A2	**1.973**	**1.98**	**1.959**	**1.953**	**1.954**
24	6C7	6CA	6DD	6D2	6D3	**1.735**	**1.738**	**1.757**	**1.746**	**1.747**
26	626	629	61D	628	61B	**1.574**	**1.577**	**1.565**	**1.576**	**1.563**
28	5A0	5A1	595	594	59A	**1.44**	**1.441**	**1.429**	**1.428**	**1.434**
30	533	531	52C	52D	52D	**1.331**	**1.329**	**1.324**	**1.325**	**1.325**

Tabla 3.3.2.2. Resultados obtenidos de la medición de "periodo" para diferentes velocidades del motor.

Comparando la tabla 3.3.2.2 con la de la caracterización del motor (tabla 3.2.1), se puede ver que el periodo calculado por el bloque GPS (unido con el encoder) funciona de forma correcta.

3.3.3 Módulo PWM del FPGA

Para probar este bloque se utiliza un push button (con un anti rebote mediante software) para emular el pulso *"IND"* proveniente de la máquina de estados. Se observa que el %CU del PWM se incrementa cada vez que se presiona el push button hasta llegar a 30%. Cuando llega a 30 comienza a decrementar el %CU, además la señal "Giro" (encargada de seleccionar el sentido de giro del motor) cambia de "01" a "10" (esto se observa en un par de LEDs a los cuales se manda la señal mencionada)

Las capturas de pantalla de las mediciones con el osciloscopio se muestran a continuación, figura 3.3.3.1.

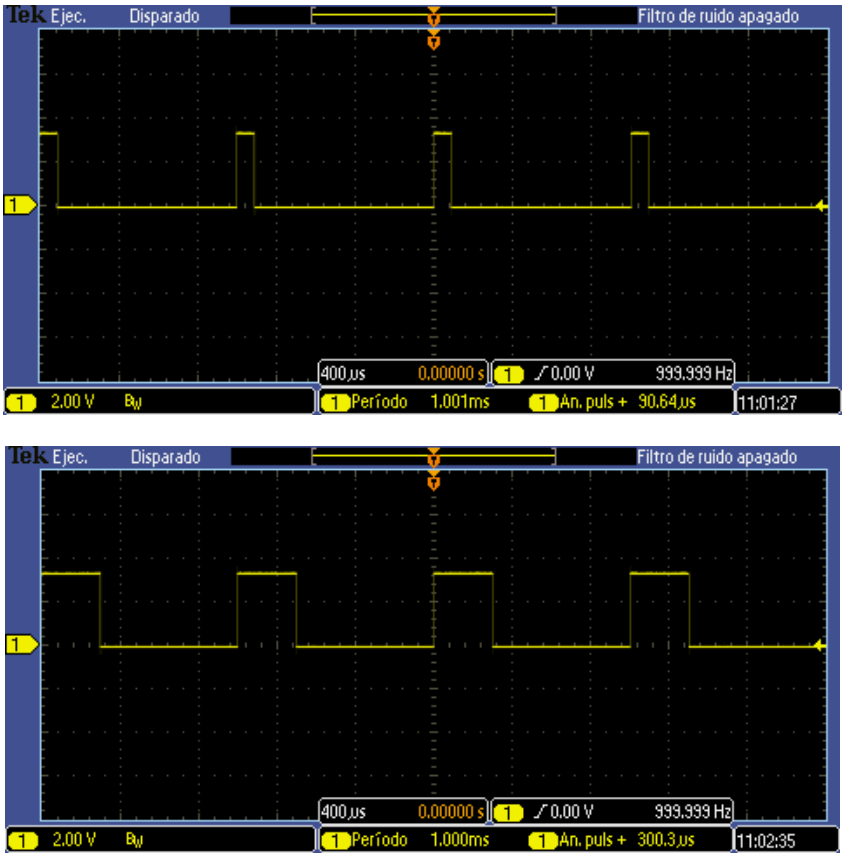
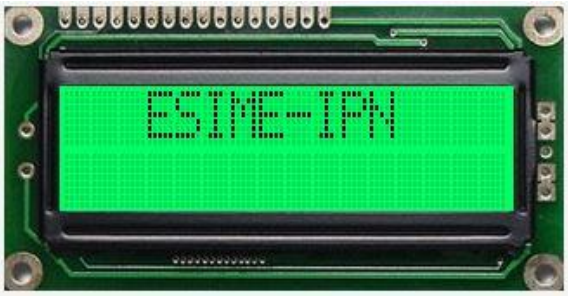
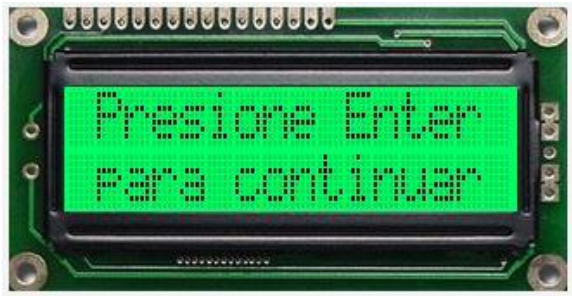


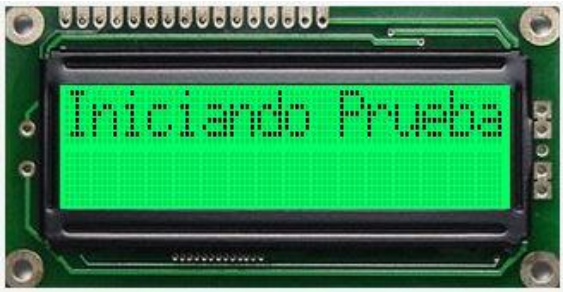
Figura 3.3.3.1: Se muestra el PWM tanto para 9 %C.U. como para 30 %C.U.

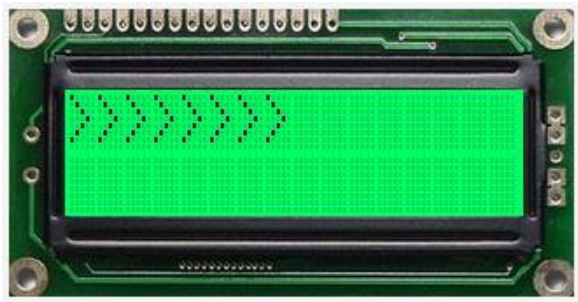
3.3.4 Módulo Mensajes del FPGA

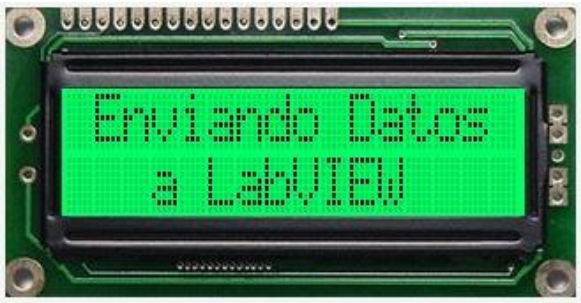
A continuación se presentan los resultados obtenidos del programa realizado en VHDL, estos resultados son el despliegue de los mensajes. Cabe resaltar que en dichas pruebas es imposible observar algunos efectos que se le dio a los mensajes tal como el de movimiento o desplazamiento.

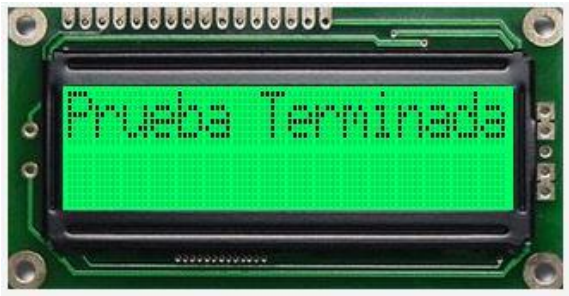












3.3.5 Módulo Control del FPGA (Máquina de Estados)

Debido a que los pulsos generados por la máquina de estados (IND, Motor_Listo, Guarda, RST_INT) tienen una duración de 20ns (es el tiempo que le toma en pasar de un estado a otro) es difícil realizar una prueba de la máquina en el laboratorio, así que para comprobar el correcto funcionamiento se utiliza el simulador Modelsim, en el cual sólo se programan las siguientes entradas al módulo: 1) reloj de 20ns, 2) un reset, 3) la señal *IR* proveniente del encoder y 4) la señal *Inicia* que proviene del push button presionado por el usuario. Teniendo estas cuatro señales, el simulador del código genera los siguientes resultados.

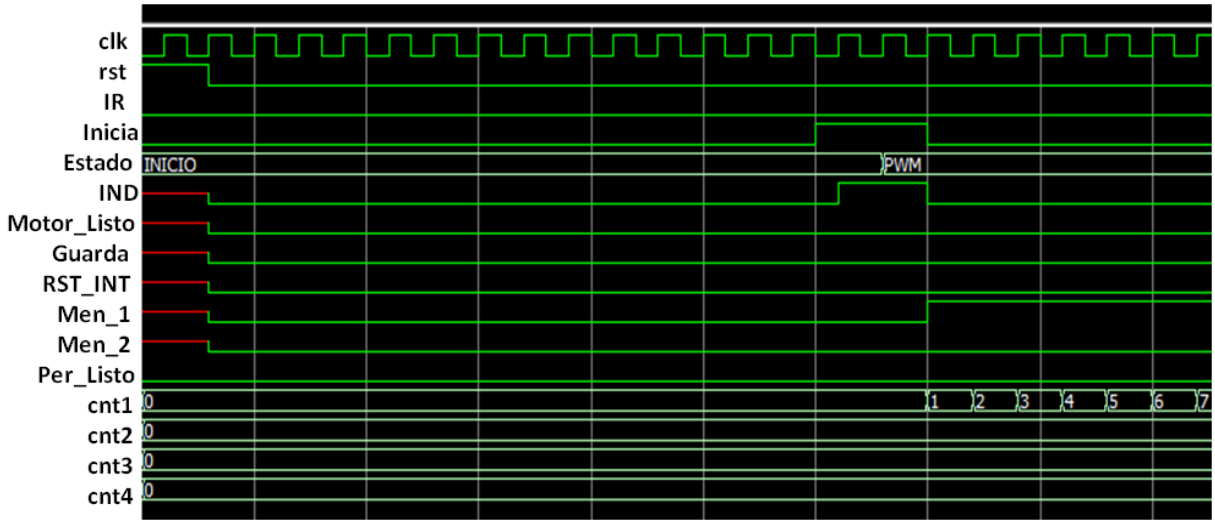


Figura 3.3.5.1. Se muestra el cambio del estado INICIO al estado PWM. IND se genera en la transición.

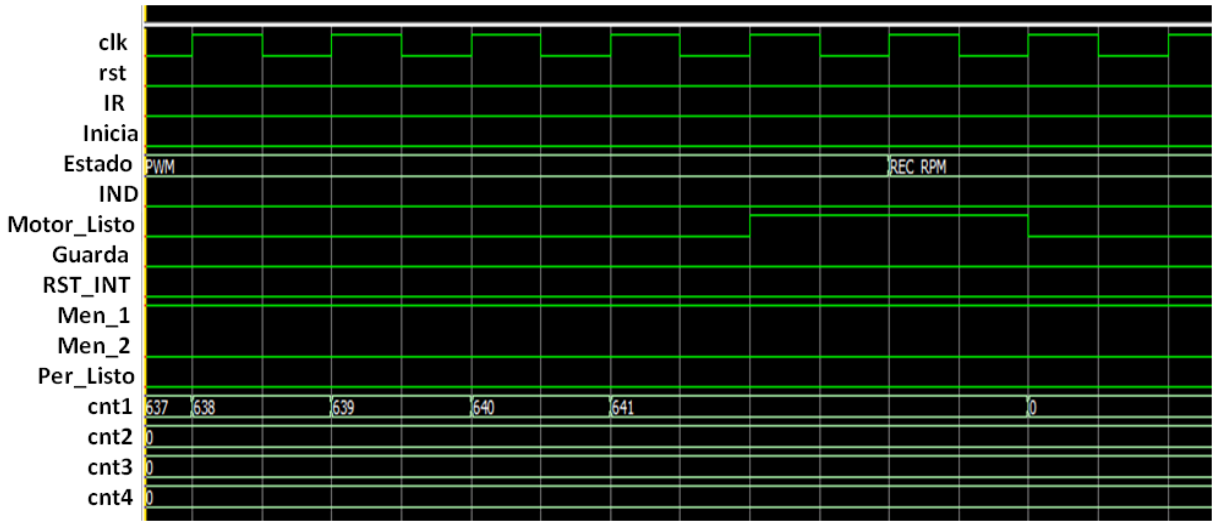


Figura 3.3.5.2. Se muestra el cambio del estado PWM al estado REC_RPM. Motor_Listo se genera en la transición.

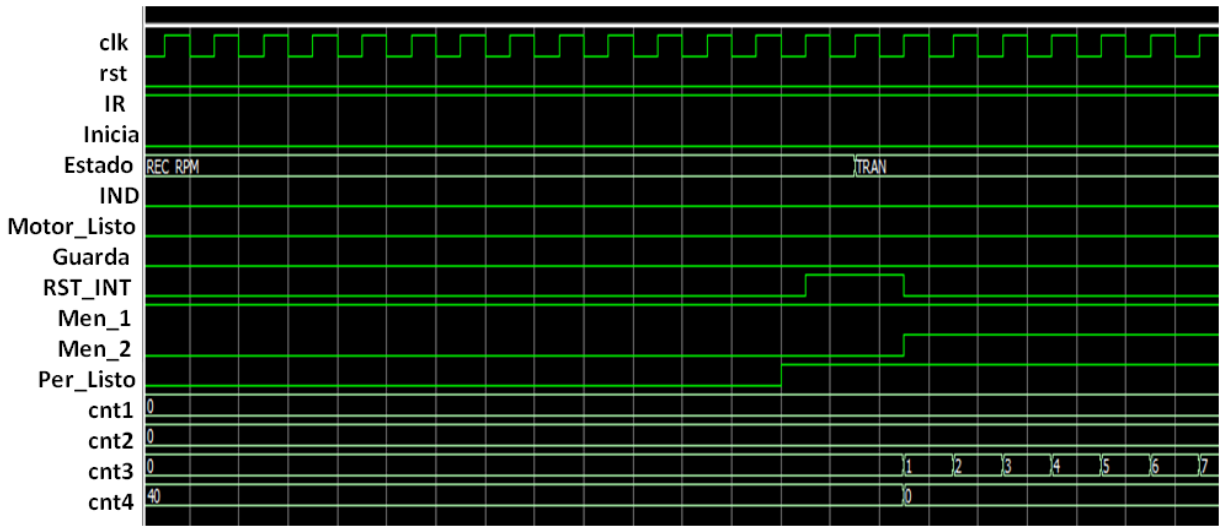


Figura 3.3.5.3. Se muestra el cambio del estado REC_RPM al estado TRAN. RST_INT se genera en la transición. Men_2 es puesto a 1 al llegar al estado TRAN.

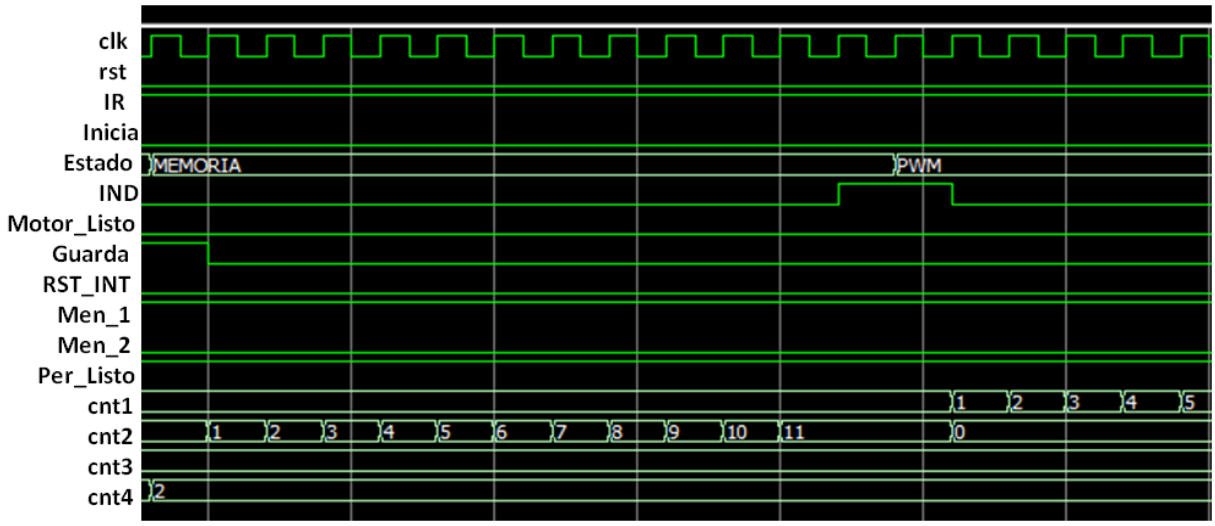


Figura 3.3.5.4. Se muestra el cambio del estado MEMORIA al estado PWM. IND se genera en la transición.

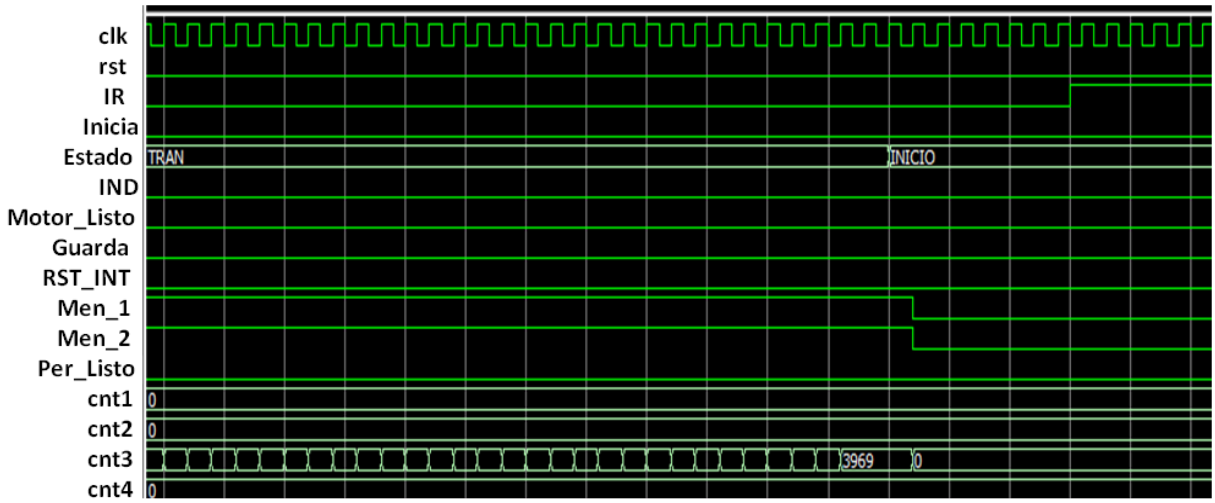


Figura 3.3.5.5. Se muestra el cambio del estado TRAN al estado INICIO.

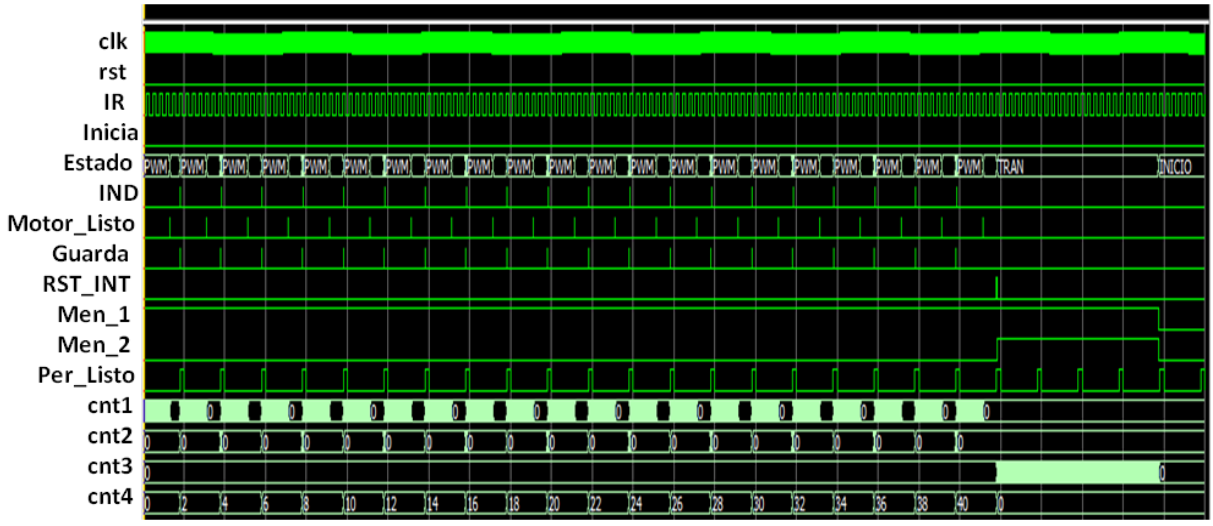


Figura 3.3.5.6. Se muestra la simulación completa del módulo control.

Cabe mencionar que la simulación se realiza para tiempos más pequeños que los reales, ya que con esto se obtienen resultados en el simulador en un periodo menor. Por ejemplo, en lugar de esperar 4 segundos en el estado PWM, sólo se espera 640 cuentas de 20ns (véase figura 3.3.5.2) y en lugar de esperar 10 segundos para pasar del estado TRAN al estado INICIO, se espera solamente 3969 cuentas de 20ns. (Figura 3.3.5.5).

En base a los resultados obtenidos en la simulación se puede decir que el módulo Control del FPGA funciona correctamente.

3.4 Bloque VA (Interfaz de LabVIEW)

Para esta prueba es necesario utilizar una tarjeta de desarrollo Spartan-3E, así como una conexión serial a la PC. A continuación se muestra una imagen de dichas conexiones, figura 3.4.1.

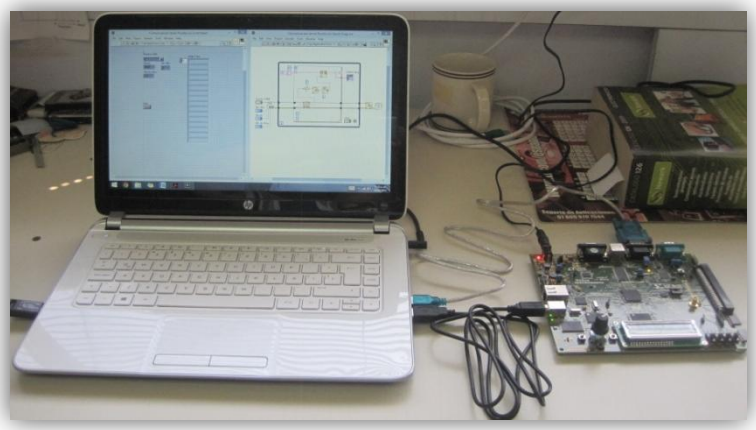


Figura 3.4.1: Conexión en laboratorio

Primeramente se prueba el módulo de configuración del puerto serial y recepción de datos enviados de parte del FPGA, así que se procede a armar y probar dicho módulo en LabVIEW, quedando de la siguiente manera, figura 3.4.2.

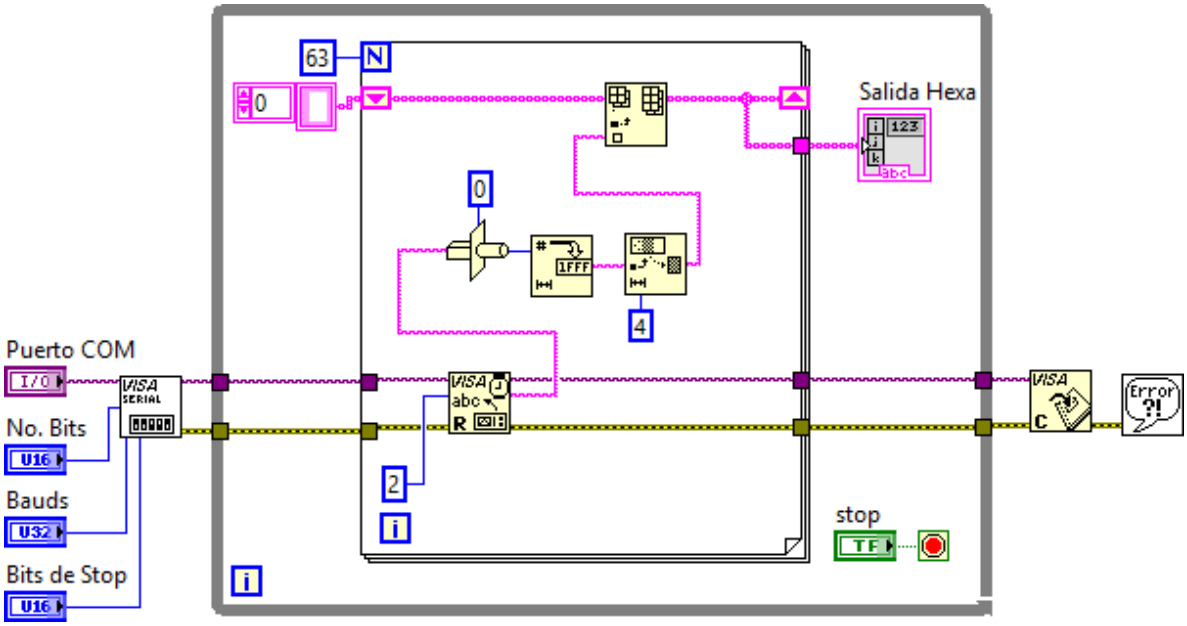


Figura 3.4.2: Módulo de configuración de puerto serial y recepción de datos.

En este punto se prueba los bloques " VISA Configure Serial Port VI " y "VISA Read Function". Con el primero bloque se prueba la correcta configuración del puerto serial para su comunicación entre el FPGA y la PC. Para observar los datos recibidos de parte de LabVIEW, se coloca un "Indicador" el cual despliega los valores recibidos de parte del FPGA y se visualizan como un arreglo de una sola dimensión (1D). En total se reciben 66 datos.

A continuación se muestra los valores obtenidos en LabVIEW, así como los valores introducidos en VHDL para corroborar el correcto funcionamiento de la transmisión y recepción mediante el puerto serial, figura 3.4.3.

Figura 3.4.3: Datos enviados desde el FPGA y recibidos en LabVIEW.

Cabe mencionar que los valores recibidos por parte de LabVIEW, se observa un ocho al inicio, esto es debido a problemas que se tuvieron en LabVIEW al recibir un valor igual a cero desde el inicio, así que ese valor se programa en VHDL. Pudiéndose observar que los valores recibidos son los correctos.

Ahora bien, se requiere realizar el acondicionamiento de los valores recibidos, cada uno de estos valores se acondiciona independientemente, para VEL y PER se utilizan fórmulas para lograr una gráfica correcta, TEM, por otro lado, sólo se maneja como una constante. A continuación se muestra el diagrama a bloque en LabVIEW para dicho acondicionamiento, figura 3.4.4.

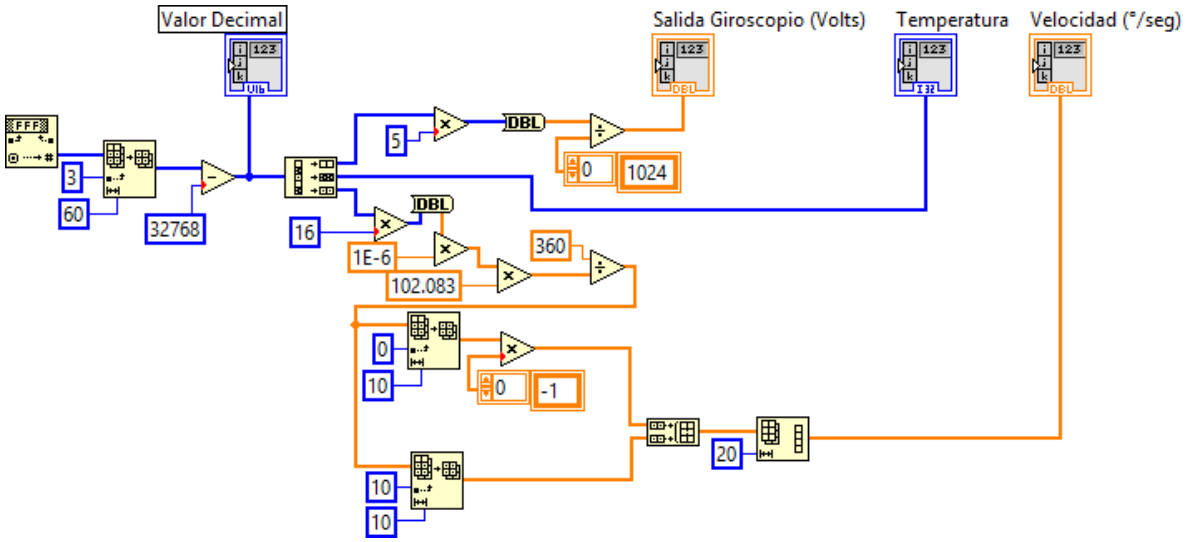


Figura 3.4.4: Modulo de acondicionamiento de señales (VEL, TEM, PER).

A cada salida de las señales se coloca un "Indicador", con el cual se puede observar el valor real a graficar, por parte de las señales (PER, VEL), figura 3.4.5.

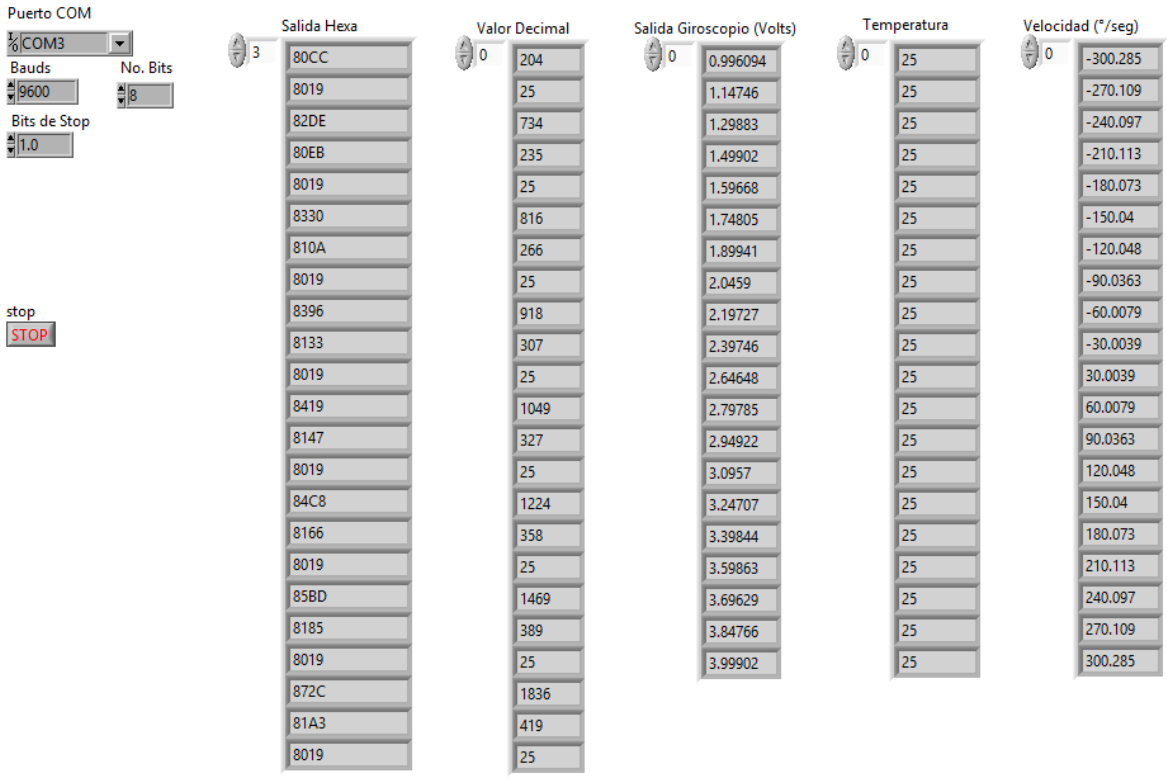


Figura 3.4.5: Valores acondicionados de las señales (VEL, TEM, PER).

Por último se prueba el funcionamiento del bloque "Linear Fit VI" con el cual se logra el ajuste de curva y a la par graficar los datos PER y VEL, que en realidad

se grafica Velocidad (°/seg) VS Volts. En la figura figura 3.4.6 se muestra los bloques armados para esta etapa.

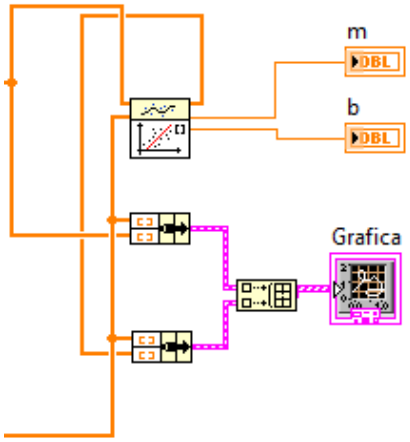


Figura 3.4.6: Modulo para la gráfica y ajuste de curva para las señales (PER, VEL).

Para el bloque "Linear Fit VI", se colocan dos "Indicadores" en sus salidas de la pendiente y ordenada al origen (m, b) y se grafica a la par de los valores de las señales VEL y PER. En la figura 3.4.7, se puede observar tanto los puntos de dichas señales como el ajuste de curva que da el bloque "Linear Fit VI".

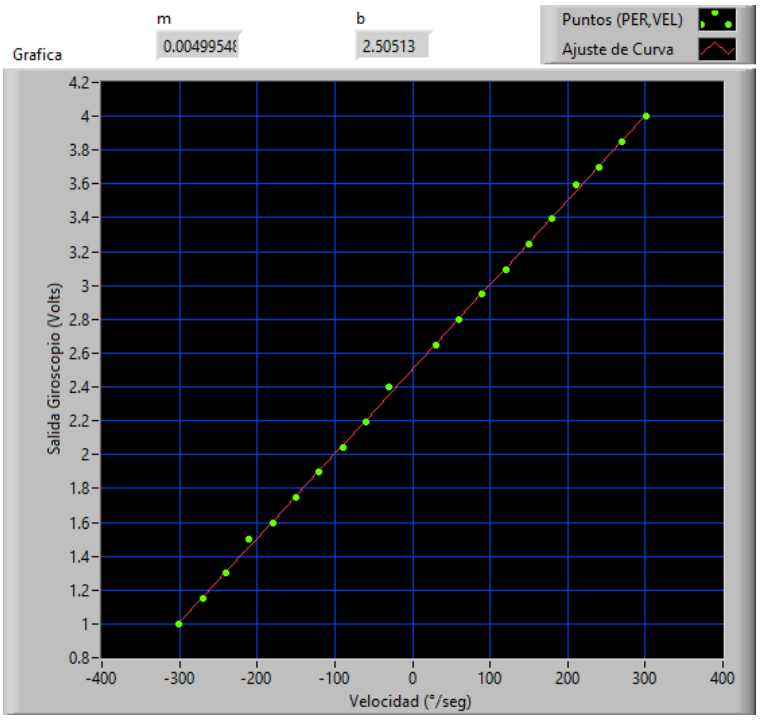


Figura 3.4.7: Gráfica y ajuste de curva para valores de VEL y PER.

3.5 Resultados obtenidos del proyecto

Teniendo todos los bloques funcionando, se unen y se realiza la caracterización de un lote de 6 giroscopios.

Pasos a seguir para la caracterización:

1) Se coloca el dispositivo a caracterizar en su base del bloque ACT.
2) Se enciende el bloque ACT localizado en la base giratoria.
3) Se presiona el botón de reset localizado en el gabinete del FPGA.
4) Se presiona el botón enter cuando lo indique la pantalla del LCD.
5) Una vez detenido el motor, en LabVIEW se selecciona el puerto COM correspondiente al caracterizador para recibir los datos y se ejecuta el programa.
6) Se presiona el botón "enviar" para transferir los datos del FPGA hacia LabVIEW.

Los resultados se muestran a continuación.

Giroscopio 1.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
34.9515	2.54883	34.9515	2.55216
59.6916	2.66602	59.6916	2.67584
88.9988	2.82227	88.9988	2.82234
119.008	2.97363	119.008	2.97237
147.179	3.11035	147.179	3.11319
175.61	3.25195	175.61	3.25532
204.545	3.39844	204.545	3.39997
230.769	3.56445	230.769	3.53107
256.41	3.66211	256.41	3.65925
287.54	3.80371	287.54	3.81487
-280.156	0.976562	-280.156	0.976904
-252.101	1.15234	-252.101	1.11715
-225.705	1.24512	-225.705	1.24911
-200.893	1.34766	-200.893	1.37315
-174.927	1.50879	-174.927	1.50295
-146.819	1.63086	-146.819	1.64347
-118.577	1.79688	-118.577	1.78465
-89.0869	1.94824	-89.0869	1.93208
-59.7114	2.06543	-59.7114	2.07893
-35.1975	2.18262	-35.1975	2.20148

Tabla 3.5.1: Resultados de la caracterización del giroscopio 1.

La primera columna muestra la velocidad a la que gira el motor. Esta se calculada en LabVIEW con los datos proporcionados por el módulo GPS del FPGA auxiliado del encoder.

La segunda muestra el voltaje proporcionado por el giroscopio cuando es excitado con la velocidad de la primera columna.

Los valores de la tercera y cuarta columna son los ajustes de la primera y segunda respectivamente.

Se realiza la gráfica en LabVIEW con los valores de la tabla 3.5.1 (véase figura 3.5.1). La gráfica punteada es realizada con los valores medidos de la velocidad del motor y el voltaje proporcionado por el MEMS. La línea continua es realizada con el ajuste.

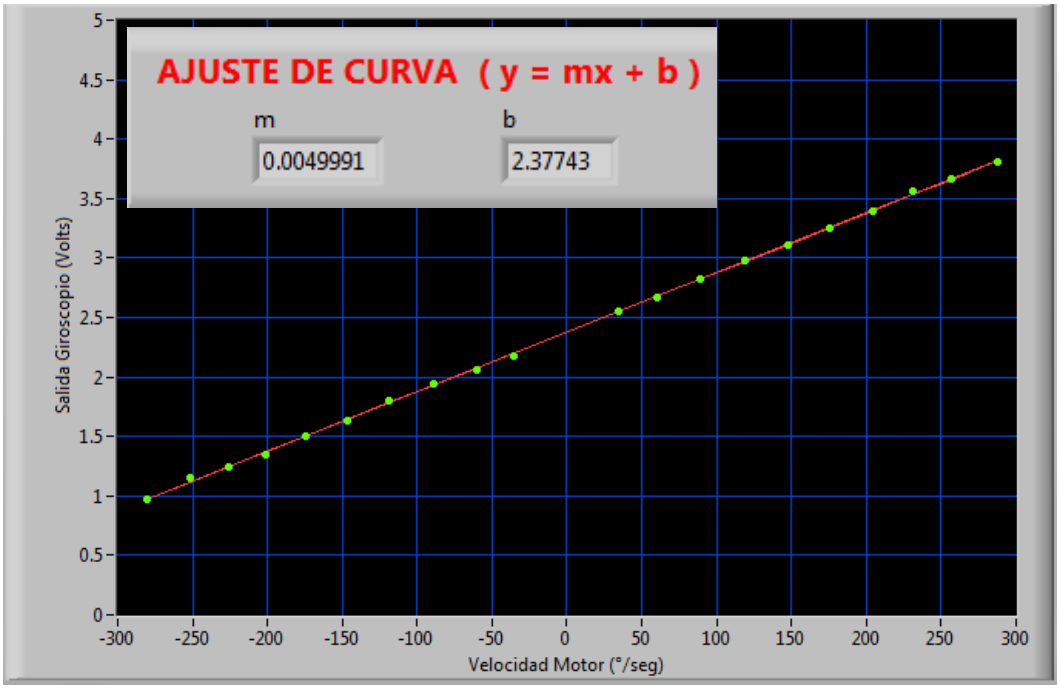


Figura 3.5.1: Gráfica de los valores de la caracterización del giroscopio 1.

En la figura 3.5.1 se observa que la sensitividad del giroscopio es de aproximadamente m = 5mV, mientras que el voltaje cuando se encuentra estático es de b = 2.377 V.

Para realizar otra caracterización, se cambia el dispositivo y se siguen todos los pasos anteriormente mencionados. Es recomendable apagar la tarjeta del bloque ACT antes de cambiar de dispositivo, así como también cerrar y volver a abrir el programa en LabVIEW.

Giroscopio 2.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
35.3461	2.73438	35.3461	2.72891
59.6718	2.85645	59.6718	2.85455
89.798	2.99805	89.798	3.01015
120.442	3.16895	120.442	3.16843
149.688	3.33008	149.688	3.31949
178.66	3.4668	178.66	3.46913
206.897	3.59863	206.897	3.61497
235.14	3.76465	235.14	3.76085
259.74	3.87695	259.74	3.88791
293.878	4.07227	293.878	4.06423
-283.688	1.08887	-283.688	1.08109
-252.809	1.18164	-252.809	1.24058
-229.153	1.34766	-229.153	1.36276
-202.817	1.53809	-202.817	1.49879
-176.817	1.65527	-176.817	1.63308
-149.75	1.77734	-149.75	1.77288
-120.04	1.92383	-120.04	1.92633
-90.1352	2.09961	-90.1352	2.08079
-60.6673	2.22656	-60.6673	2.233
-36.7459	2.3584	-36.7459	2.35655

Tabla 3.5.2: Resultados de la caracterización del giroscopio 2.

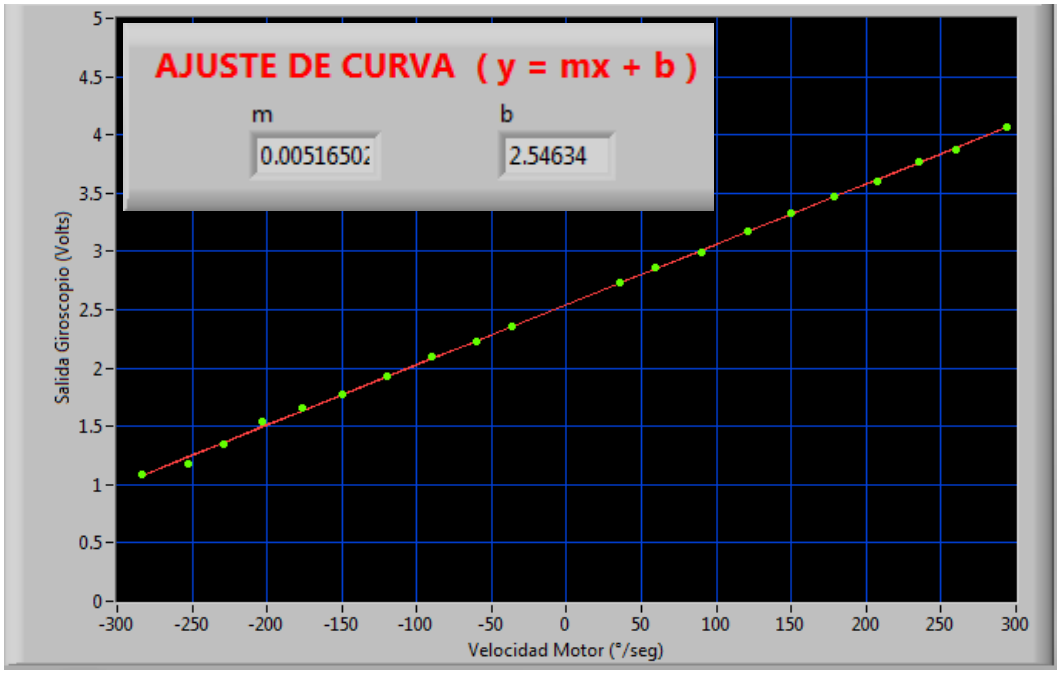


Figura 3.5.2: Gráfica de los valores de la caracterización del giroscopio 2.

Giroscopio 3.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
34.6021	2.54395	34.6021	2.54457
59.4845	2.66113	59.4845	2.67002
89.3744	2.8125	89.3744	2.82072
118.93	2.97363	118.93	2.96974
146.58	3.11523	146.58	3.10914
173.913	3.26172	173.913	3.24695
202.247	3.39844	202.247	3.38981
231.214	3.52051	231.214	3.53586
256.046	3.67188	256.046	3.66105
289.622	3.81348	289.622	3.83034
-284.36	0.947266	-284.36	0.936407
-252.278	1.05957	-252.278	1.09816
-227.273	1.21582	-227.273	1.22423
-202.361	1.35742	-202.361	1.34984
-175.268	1.52344	-175.268	1.48644
-147.059	1.62109	-147.059	1.62866
-117.532	1.77246	-117.532	1.77753
-89.7532	1.92871	-89.7532	1.91759
-60.5449	2.06543	-60.5449	2.06485
-36.84	2.18262	-36.84	2.18437

Tabla 3.5.3: Resultados de la caracterización del giroscopio 3.

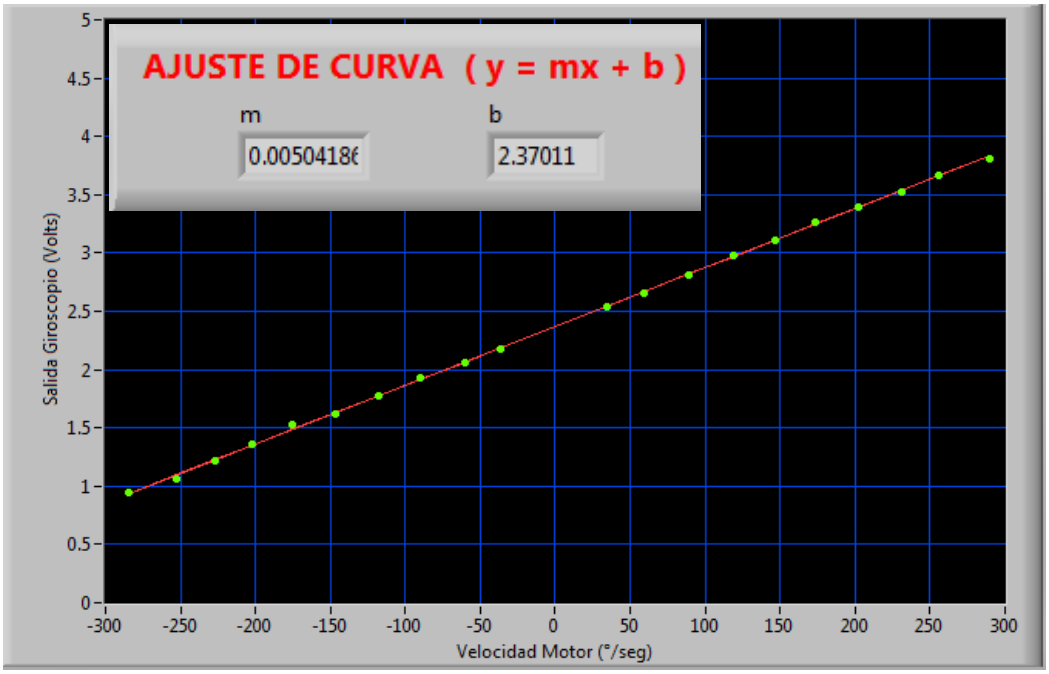


Figura 3.5.3: Gráfica de los valores de la caracterización del giroscopio 3.

Giroscopio 4.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
35.8352	2.67578	35.8352	2.67596
60.7698	2.80762	60.7698	2.8027
92.545	2.96875	92.545	2.96421
123.161	3.11523	123.161	3.11982
152.737	3.27637	152.737	3.27015
181.452	3.38379	181.452	3.4161
209.79	3.56934	209.79	3.56014
237.624	3.71582	237.624	3.70162
263.35	3.84277	263.35	3.83238
294.118	3.97949	294.118	3.98877
-285.261	1.03027	-285.261	1.04388
-253.7	1.19629	-253.7	1.20431
-230.032	1.32813	-230.032	1.32461
-205.245	1.48926	-205.245	1.45059
-177.778	1.58691	-177.778	1.5902
-150.88	1.72363	-150.88	1.72692
-122.158	1.86523	-122.158	1.87291
-92.3077	2.03125	-92.3077	2.02463
-61.8769	2.16797	-61.8769	2.17931
-37.2478	2.2998	-37.2478	2.30449

Tabla 3.5.4: Resultados de la caracterización del giroscopio 4.

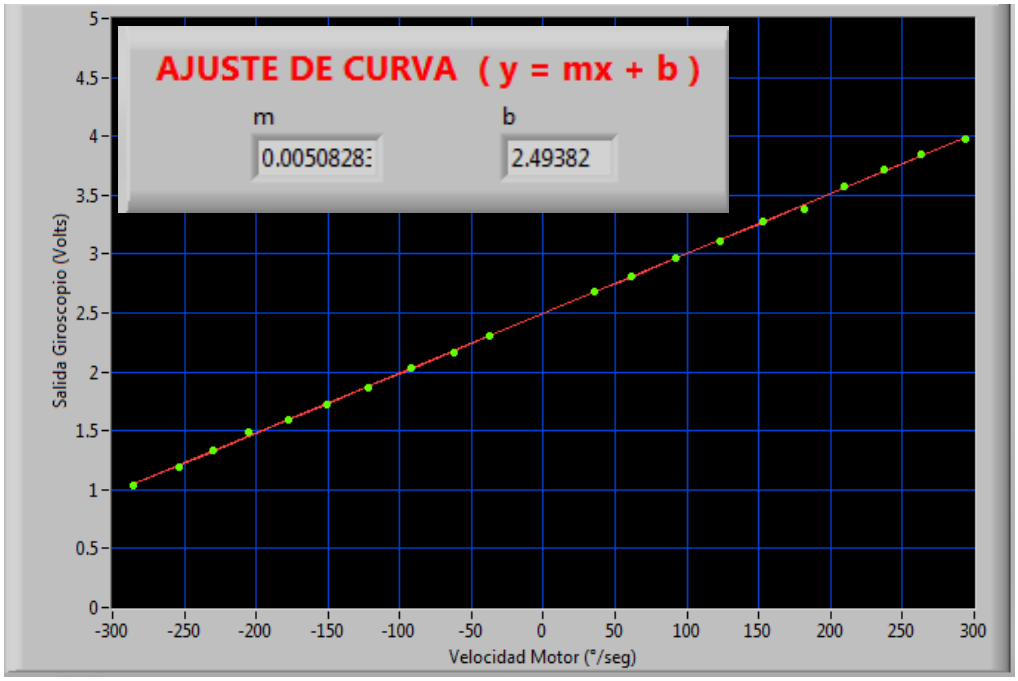


Figura 3.5.4: Gráfica de los valores de la caracterización del giroscopio 4.

Giroscopio 5.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
35.732	2.58301	35.732	2.59084
61.6333	2.72461	61.6333	2.72236
91.5332	2.87598	91.5332	2.87418
120.927	3.03223	120.927	3.02343
150.313	3.18359	150.313	3.17264
178.749	3.31543	178.749	3.31702
207.254	3.47168	207.254	3.46176
235.294	3.59863	235.294	3.60413
258.065	3.72559	258.065	3.71975
291.498	3.86719	291.498	3.88951
-284.585	0.952148	-284.585	0.964417
-254.237	1.10352	-254.237	1.11851
-229.738	1.24023	-229.738	1.2429
-203.62	1.37695	-203.62	1.37552
-178.218	1.52832	-178.218	1.5045
-150.691	1.66016	-150.691	1.64427
-120.765	1.79199	-120.765	1.79622
-91.954	1.94824	-91.954	1.94251
-61.845	2.08496	-61.845	2.09539
-37.0294	2.2168	-37.0294	2.22139

Tabla 3.5.5: Resultados de la caracterización del giroscopio 5.

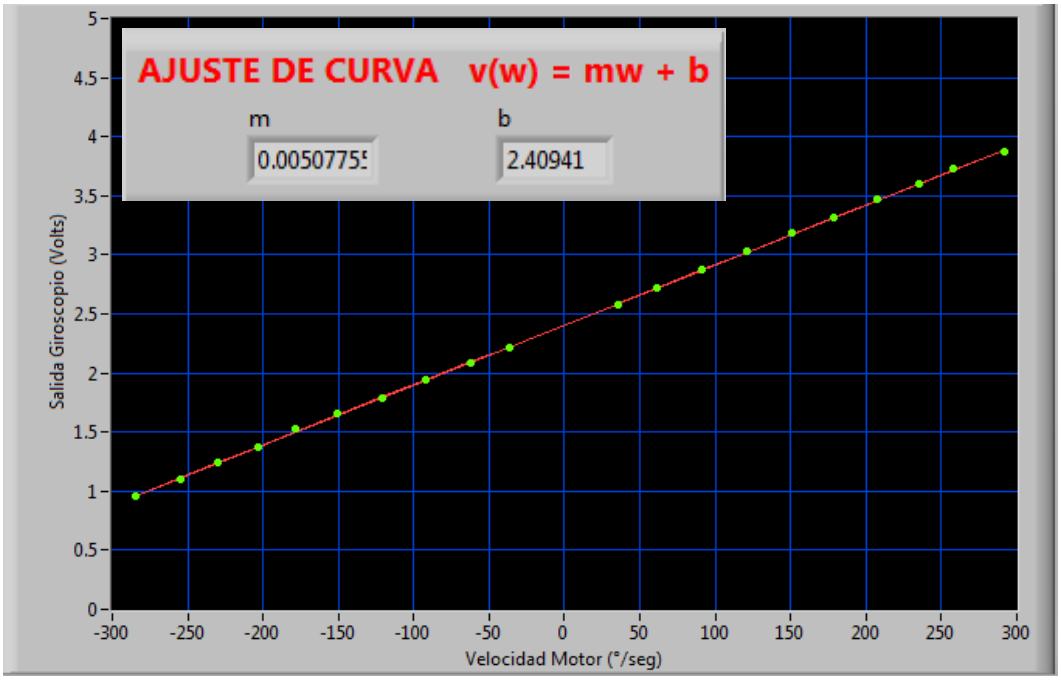


Figura 3.5.5: Gráfica de los valores de la caracterización del giroscopio 5.

Giroscopio 6.

Velocidad Motor (°/seg) - Medición	Salida Giroscopio (Volts) - Medición	Velocidad Motor (°/seg) - Ajuste de Curva	Salida Giroscopio (Volts) - Ajuste de Curva
35.0536	2.54395	35.0536	2.54705
60.6469	2.69043	60.6469	2.67744
92.1659	2.83203	92.1659	2.83803
121.049	2.98828	121.049	2.98519
151.197	3.13965	151.197	3.13879
180.27	3.30566	180.27	3.28692
209.181	3.42773	209.181	3.43422
236.998	3.58398	236.998	3.57595
263.158	3.70605	263.158	3.70923
293.878	3.84766	293.878	3.86575
-286.396	0.878906	-286.396	0.90927
-256.228	1.0791	-256.228	1.06298
-231.511	1.20117	-231.511	1.18891
-206.068	1.31348	-206.068	1.31854
-180.27	1.43066	-180.27	1.44998
-151.197	1.64063	-151.197	1.59811
-122.993	1.7334	-122.993	1.7418
-92.8314	1.89453	-92.8314	1.89548
-62.1654	2.0459	-62.1654	2.05172
-36.8475	2.17285	-36.8475	2.18071

Tabla 3.5.6: Resultados de la caracterización del giroscopio 6.

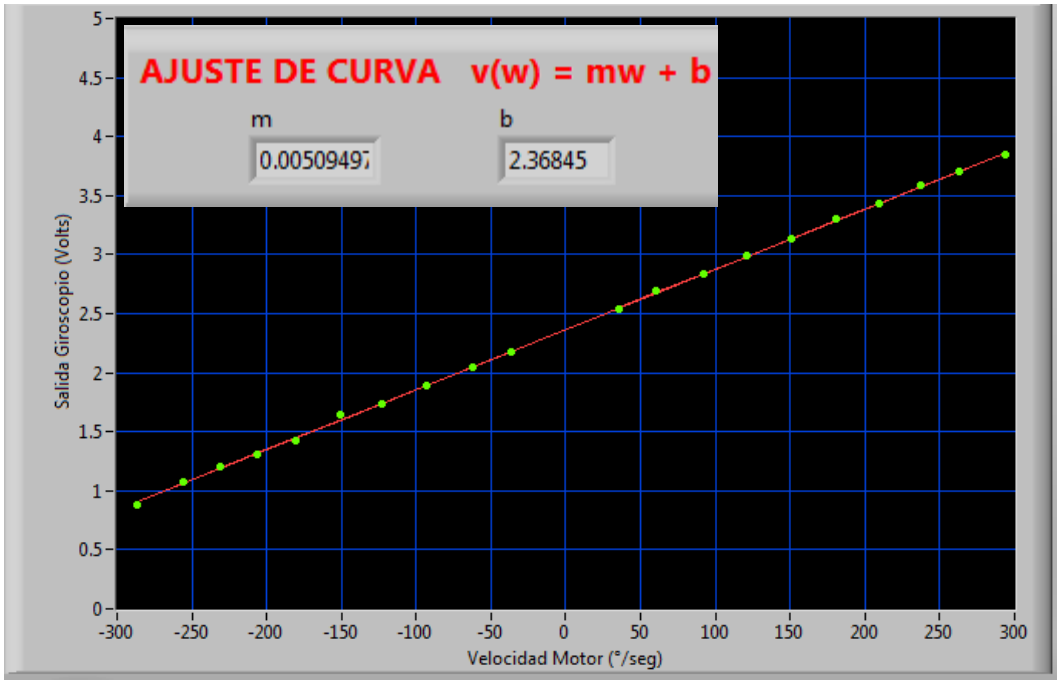


Figura 3.5.6: Gráfica de los valores de la caracterización del giroscopio 6.

Giroscopio	Sensitividad (m) $[mV/°/s]$	Null (b) $[Volts]$
1	4.99	2.37
2	5.16	2.54
3	5.04	2.37
4	5.08	2.49
5	5.07	2.41
6	5.09	2.37

Parameter	Conditions	Min[1]	Typ	Max[1]	Unit
SENSITIVITY	Clockwise rotation is positive output				
Dynamic Range[2]	Full-scale range over specifications range	±300			°/s
Initial	@25°C	4.6	5	5.4	mV/°/s
Over Temperature[3]	V_S = 4.75 V to 5.25 V	4.6	5	5.4	mV/°/s
Nonlinearity	Best fit straight line		0.1		% of FS
NULL					
Initial Null		2.3	2.50	2.7	V

Figura 3.5.7: Resultados obtenidos en los seis giroscopios ADXRS300 (arriba) y sección de la hoja de datos de los mismos (abajo).

En todas las caracterizaciones se tienen valores de $4.6[mV/°/s] < m < 5.4[mV/°/s]$ y $2.3[V] < b < 2.7[V]$, los cuales son similares a los esperados (véase figura 3.5.7). Con esto se puede decir que el caracterizador está funcionando de forma correcta.

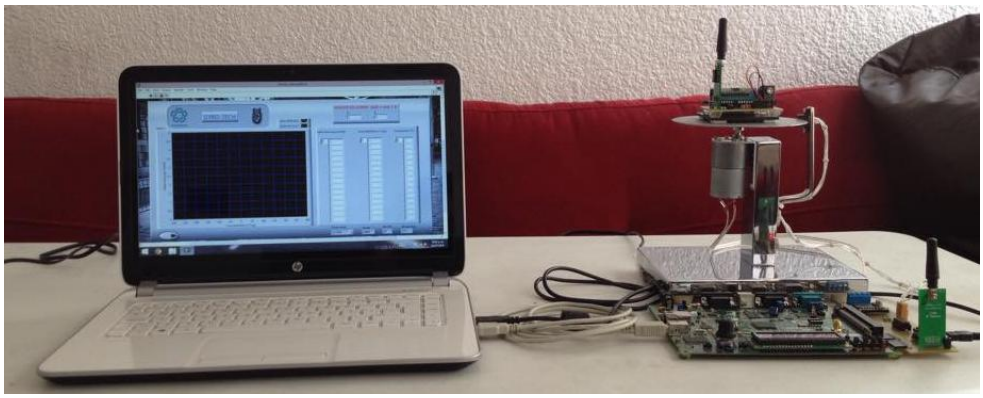


Figura 3.5.8: Fotografía del sistema de obtención de parámetros.

Conclusiones y recomendaciones

- Para tener una mejor caracterización del giroscopio es conveniente utilizar un sistema que incluya un bloque de control de temperatura del área de trabajo del dispositivo MEMS, para así poder realizar pruebas con diferentes valores de temperatura.

- Se puede reducir tiempos en la caracterización del giroscopio, para ello es recomendable utilizar un disco con más de una perforación, para así poder realizar más rápido el cálculo del tiempo que tarda en dar una vuelta el motor.

- Para cubrir un mayor intervalo de velocidades bajas, se recomienda utilizar un motor con otra relación de engranaje ya que el utilizado en este proyecto al colocarle una carga (la tarjeta que contiene al bloque ACT) sólo puede girar a una velocidad mínima de aproximadamente 30°/s, dejando así un intervalo de 60°/s sin analizar (de +30°/s a -30°/s).

- También para reducir la carga que el motor mueve se recomienda utilizar en la tarjeta del bloque ACT componentes de montaje superficial, además de otros módulos de RF ya que con esto se logra minimizar el tamaño de dicha tarjeta y con ello el tamaño de gabinete.

- Para la tarjeta del bloque ACT se utiliza una batería de 200mAh, la cual sólo dura cargada aproximadamente 1 hora, por lo cual si se desea utilizar el caracterizador por más tiempo se recomienda utilizar baterías de mayor cantidad de mAh.

Bibliografía

- Ronald J. Tocci Sistemas Digitales Principios y Aplicaciones, Prentice Hall, 1996

- David G. Máxinez, Jessica Alcalá, VHDL El arte de programar sistemas digitales, Grupo editorial patria, 2002

- Douglas L. Perry, VHDL Programming by Example, McGraw-Hill, 2002

- José Rafael Lajara Vizcaino y José Peligri Sebastiá, LabVIEW entorno grafico de programación, Alfaomega, 2007

- Joaquín del Rio Fernández, Shahram Sharlat- Panahi, David Sarriá Gandul y Antoni Mánuel Lázaro, LabVIEW programación para sistemas de instrumentación.

Referencias

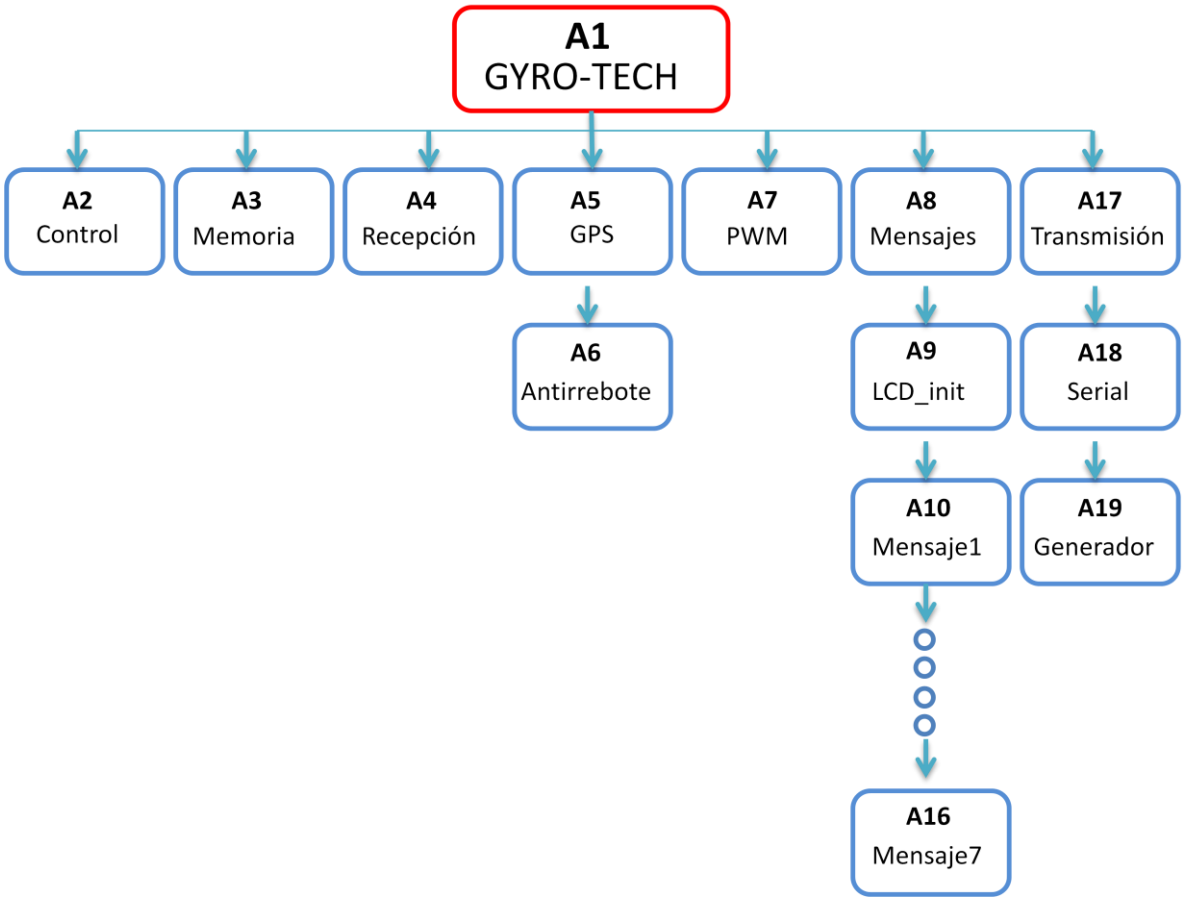
- Datasheet Atmel, 8-bit Microcontroller ATmega48

- Spartan-3E Starter Kit Board User Guide, Xilinx

Medios electrónicos

- http://www.dsi.fceia.unr.edu.ar/downloads/dda/vhdl_pardocarpio.pdf
- http://www.ing.unlp.edu.ar/islyd/Trabajo%20Final.pdf

Apéndice A



Códigos del FPGA

A1

```
--------------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               Gyro_Tech
-- Descripción:          Programa principal de enlace de módulos
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Componentes_pkg.ALL;

entity Gyro_Tech is
    Port( clk, rst, rstx, Rx, IR, inicia: in STD_LOGIC;
          data:  inout STD_LOGIC_VECTOR( 0 to 3 );
          giro: out     STD_LOGIC_VECTOR( 1 downto 0);
```

```
          LCD_E, LCD_RS, LCD_RW, per_listo:  inout   STD_LOGIC;
          SF_CE0, Tx, pwm: out STD_LOGIC );
end Gyro_Tech;

architecture Arq of Gyro_Tech is

signal IND, Motor_listo, men_1, men_2, rst_int, guarda, etx: std_logic;
signal Temrx, Velrx, periodo, V, T, P : std_logic_vector (15 downto 0);

begin

mod_1: Control  port map (clk, rst, inicia, per_listo, IND, men_1, men_2,
                                  rst_int, guarda, Motor_listo );
mod_2: Memoria  port map (Temrx, Velrx, periodo, rst, rst_int, rstx, IND,
                                  guarda, etx, V, T, P );
mod_3: Recep    port map (clk, rst, Rx, Motor_listo, Temrx, Velrx );
mod_4: GPS      port map (clk, rst, IR, rst_int, Motor_listo, periodo,
                                       per_listo);
mod_5: PWM      port map (clk, rst, IND, rst_int, giro, pwm);
mod_6: Mensajes port map (clk, rst, men_1, men_2, data, LCD_E, LCD_RS,
LCD_RW,                                SF_CE0 );
mod_7: Trans    port map (clk, rstx, Tx, etx, V, T, P );

end Arq;
```

A2

```
--------------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               Control
-- Descripción:          Máquina de estados para el control de los módulos
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Control is
      port (      clk, rst, inicia, per_listo: in std_logic;
                        IND,men_1,men_2,rst_int,Guarda,Motor_listo: out
std_logic
                  );
end Control;

architecture Arq of Control is

type estados is(INICIO, PWM, REC_RPM, MEMORIA, TRAN);

signal edo_presente,edo_futuro: estados;
signal cnt1: std_logic_vector(27 downto 0);
signal cnt2: std_logic_vector(3 downto 0);
signal cnt3:std_logic_vector(5 downto 0);
signal cnt4: std_logic_vector(31 downto 0);
```

```
begin


process(clk, rst, edo_presente, edo_futuro, inicia, Per_Listo, cnt1,
cnt2,             cnt3, cnt4)
begin

      if rst='1' then
            edo_futuro <= INICIO;
            cnt1<=(others=>'0');
            cnt2<=(others=>'0');
            cnt3<=(others=>'0');
            cnt4<=(others=>'0');
      elsif clk'event and clk = '1' then
            edo_presente <= edo_futuro;
            case edo_presente is
                  when INICIO  =>   IND<='0'; Motor_Listo<='0';
men_1<='0';                         men_2<='0'; rst_int<='0'; Guarda
<='0';cnt4<=(others=>'0');
                  if inicia='0' then
                  edo_futuro <= INICIO;
            else
                  edo_futuro <= PWM;
                  IND<='1';
            end if;

            when PWM           =>   men_1<='1'; cnt2<=(others=>'0');
IND<='0';

            if cnt1 <= x"BEBC200" then -- cnt1 <= 4seg
            cnt1 <= cnt1+1;
            edo_futuro <= PWM;
            else
            edo_futuro <= REC_RPM;
            Motor_Listo <= '1';
            end if;

      when REC_RPM =>   men_1<='1'; Motor_Listo<='0'; cnt1<=(others =>
'0');
            if Per_Listo='0' then
            edo_futuro <= REC_RPM;
            elsif Per_Listo='1' then
            if cnt3 <= "100111" then  -- cnt3 <= 39
            cnt3 <= cnt3+1;
            edo_futuro <= MEMORIA;
            Guarda <= '1';
            else
            edo_futuro <= TRAN;
            rst_int <= '1';
            end if;
end if;

when MEMORIA =>   men_1<='1'; Guarda<='0';
            if cnt2 <= "1010" then -- cnt2 <= 10
            cnt2 <= cnt2+1;
            edo_futuro <= MEMORIA;
```

```
            else
            edo_futuro <= PWM;
            IND<='1';
            end if;

when TRAN    =>   men_2<='1'; rst_int<='0'; cnt3<=(others => '0');
            if cnt4 <= x"1DCD6500" then --cnt4 <= 10seg
            cnt4 <= cnt4+1;
            edo_futuro <= TRAN;
            else
            edo_futuro <= INICIO;
            end if;
      end case;
      end if;
end process;

end Arq;
```

A3

```
--------------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               Memoria
-- Descripción:          Lee o Escribe los datos de Temperatura, Velocidad
                         y Periodo
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity Memoria is
  port(     Temrx, Velrx, Periodo: in std_logic_vector (15 downto 0);
                  rst, rst_int, rstx, IND, Guarda, etx: in std_logic;
                  V,T,P: out std_logic_vector (15 downto 0) );
end Memoria;

architecture Arq of Memoria is

signal address : integer range 0 to 20 :=0;

type ram_t is array(0 to 20) of STD_LOGIC_VECTOR(15 downto 0);
                                    -- 21 datos de 16 bits cada uno
signal Tem: ram_t;
signal Vel: ram_t;
signal Per: ram_t;
signal Senal_1,Senal_2,WE:std_logic;


begin

process (rst_int, rst)
begin
```

```
	if rst='1'  then
		WE <= '1';
	elsif rst_int'event and rst_int = '0' then
		WE <= '0';
	end if;
end process;

process(Senal_1,WE)
begin

	if Senal_1'event and Senal_1='1' then
		if(WE='1') then		-- Almacena los datos en la memoria
			Tem(address) <= Temrx;
			Vel(address) <= Velrx;
			Per(address) <= Periodo;
		else				-- Lee los datos de la memoria
			T <= Tem(address);
			V <= Vel(address);
			P <= Per(address);
		end if;
	end if;

end process;

process(rst, rst_int, Senal_2, address,Tem)
begin

	if rst='1' or rst_int='1' or rstx='1' then
		address <= 0;
	elsif Senal_2'event and Senal_2='1' then --Cambio de dirección
		if address < Tem'high then
			address <= address + 1;
		else
			address <= 0;
		end if;
	end if;

end process;

-- Senal_1 y Senal_2 se asignan de acuerdo a si se lee o se escribe en la
memoria
with WE select
	senal_1<= Guarda when '1',
			etx	when '0',
			'0'	when others;
with we select
	senal_2<= IND	 when '1',
			etx	when '0',
			'0'	when others;

end Arq;
```

A4

```
----------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               Recep
-- Descripción:          Recepción a 9600 bauds
----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Recep is
    Port ( clk,rst,rx: in  std_logic;
                   Motor_Listo: in std_logic;
                   Temrx, Velrx:   out std_logic_vector(15 downto 0) );
end Recep;

architecture Arq of Recep is

signal      erx:    std_logic:='0';
signal      trx:    std_logic_vector(6 downto 0);
signal   cntrx:  std_logic_vector(10 downto 0);
constant baudrx: std_logic_vector(10 downto 0):= "11011001000"; -(9600*3)
                                                                  bauds
signal   sincro: std_logic_vector(27 downto 0);
signal      Listo: std_logic;


begin

--Detecta el pulso enviado por la maquina de estados
process(rst,Motor_Listo,trx)
begin
      if rst='1' or  trx="1111010" then
            Listo <= '0';
      elsif Motor_Listo'event and Motor_Listo='1' then
            Listo <='1';
      end if;
end process;

--Contador para el bit de sincronía
process (rst,clk,rx)
begin
      if rst='1' or rx='0' then
            sincro <=(others=>'0');
      elsif clk'event and clk='1' then
            sincro<=sincro+1;
      end if;
end process;

--Habilita o deshabilita la recepción (Listo, Bit de sincronía y bit de
inicio)
```

```
process (rst,rx,trx,sincro,Listo)
begin
      if rst='1' then
            erx <= '0';
      elsif Listo='1' and rx='0' and sincro>X"0DD40A0" then --  Detecta
            el bit de inicio y de sincronia
                  erx <= '1';       -- Habilita la recepción
      elsif trx="1111010" then      --Otra opcion: elsif Listo='0' then
            erx <= '0';             -- Deshabilita la recepción
      end if;
end process;

process (clk,rst,cntrx,erx)
begin
      if (rst='1' or erx='0')then
            cntrx <= (others=>'0');
            trx   <= (others=>'0');
      elsif (clk'event and clk='1' and erx='1')then
               cntrx <= cntrx + 1;
                  if (cntrx=baudrx)then
                        trx <= trx + 1;
                        cntrx<=(others=>'0');
                  end if;
      end if;
end process;

--Almacenando lo recibido en arreglos
process (clk,rst,trx)
begin
   if rst='1' then
            Velrx   <= (others=>'0');
            Temrx  <= (others=>'0');

      elsif (clk'event and clk='1') then
            case (trx) is
                  ---------------------------------------
                  -------Vel. Ang. MSB--------------------
                  when "0000100" =>  Velrx(8) <=      rx;-- b9 trx=4
                  when "0000111" =>  Velrx(9) <=      rx;-- b10 (Bit Más
                        significativo de la conversión de la Velocidad)
                  when "0001010" =>  Velrx(10) <=     rx;-- Trash
                  when "0001101" =>  Velrx(11) <=     rx;-- Trash
                  when "0010000" =>  Velrx(12) <=     rx;-- Trash
                  when "0010011" =>  Velrx(13) <=     rx;-- Trash
                  when "0010110" =>  Velrx(14) <=     rx;-- Trash
                  when "0011001" =>  Velrx(15) <=     rx;-- Trash trx=25
                  ---------------------------------------
                  -------Vel. Ang. LSB--------------------
                  when "0100010" =>  Velrx(0) <=      rx;-- b0 (lsb)trx=34
                  when "0100101" =>  Velrx(1) <=      rx;-- b1
                  when "0101000" =>  Velrx(2) <=      rx;-- b2
                  when "0101011" =>  Velrx(3) <=      rx;-- b3
                  when "0101110" =>  Velrx(4) <=      rx;-- b4
                  when "0110001" =>  Velrx(5) <=      rx;-- b5
                  when "0110100" =>  Velrx(6) <=      rx;-- b6
                  when "0110111" =>  Velrx(7) <=      rx;-- b7 trx=55
                  ---------------------------------------
```

```
                    -------Temp MSB-------------------------
                    when "1000000" =>  Temrx(8) <=      rx;-- b9 trx=64
                    when "1000011" =>  Temrx(9) <=      rx;-- b10 Bit Más
                          significativo de la conversión de Temperatura)
                    when "1000110" =>  Temrx(10) <=     rx;-- Trash
                    when "1001001" =>  Temrx(11) <=     rx;-- Trash
                    when "1001100" =>  Temrx(12) <=     rx;-- Trash
                    when "1001111" =>  Temrx(13) <=     rx;-- Trash
                    when "1010010" =>  Temrx(14) <=     rx;-- Trash
                    when "1010101" =>  Temrx(15) <=     rx;-- Trash trx=85
                    ------------------------------------
                    -------Temp LSB-------------------------
                    when "1011110" =>  Temrx(0) <=      rx;-- b0 (lsb)trx=94
                    when "1100001" =>  Temrx(1) <=      rx;-- b1
                    when "1100100" =>  Temrx(2) <=      rx;-- b2
                    when "1100111" =>  Temrx(3) <=      rx;-- b3
                    when "1101010" =>  Temrx(4) <=      rx;-- b4
                    when "1101101" =>  Temrx(5) <=      rx;-- b5
                    when "1110000" =>  Temrx(6) <=      rx;-- b6
                    when "1110011" =>  Temrx(7) <=      rx;-- b7 trx=115
                    ------------------------------------
                    ------------------------------------
                    when others => null;
            end case;
       end if;
end process;

end Arq;
```

A5

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              GPS
-- Descripción:         Calcula el Periodo del giro del motor
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Componentes_pkg.ALL;

entity GPS is
    Port (  clk,rst,IR,rst_int : in  STD_LOGIC;
                        Motor_Listo: in std_logic;
                        Periodo: out std_logic_vector (15 downto 0);
                        Per_Listo: inout std_logic
                );
end GPS;

architecture Arq of GPS is

signal clk1,cnt1:std_logic_vector(15 downto 0);
signal ref, Listo, SinReb, alto: std_logic;
```

```
begin

process (rst_int, rst)
begin
      if rst='1' then
            alto <= '0';
      elsif rst_int'event and rst_int = '0' then
            alto <= '1';
      end if;
end process;

--Detecta el pulso enviado por la maquina de estados
process(rst,Per_Listo, Motor_Listo)
begin
      if rst='1' or Per_Listo='1' then
            Listo <= '0';
      elsif Motor_Listo'event and Motor_Listo='0' then
            Listo <='1';
      end if;
end process;

--Reloj de 1ms
process(clk,rst,clk1)
begin
      if rst='1'  or clk1 = x"C350" then --110010
            clk1<=(others=>'0');
      elsif clk'event and clk='0' then
            clk1 <= clk1 + 1;
      end if;
end process;


-- Señal de Referencia
process (SinReb,rst,Listo,alto)
begin
      if rst='1' or alto='1' then
            ref <= '0';
      elsif SinReb'event and SinReb='1' and Listo='1' then
         ref <= not ref;
      end if;
end process;


-- Contador de pulsos de 1ms en una vuelta
process (clk1(15),rst,ref)
begin
      if rst='1' or ref='0' then
            cnt1 <= (others => '0');
      elsif clk1(15)'event and clk1(15)='0' and ref='1' then--cuando se
inicializa clk1
            cnt1 <= cnt1 + 1 ;
      end if;
end process;
```

```
-- Calcula el Periodo
process (ref,rst,Motor_Listo)
begin
      if rst='1' or Motor_Listo='1' then
            Periodo <= (others => '0');
            Per_Listo <='0';
      elsif ref'event and ref='0' then
         Periodo <= cnt1; -- hay que hacer en labview la multiplicacion:
Periodo x 1ms
            Per_Listo <='1';
      end if;
end process;

com1: Antirrebote port map (clk1(15),IR,SinReb);

end Arq;
```

A6

```
--------------------------------------------------------------------------------
-- Escuela:       E.S.I.M.E. IPN
-- Programadores: Luis Sánchez Márquez
--                Ricardo Flores Martínez
-- Asesor:        M. en C. Luis Martin Flores Nava
-- Proyecto:      TESIS
-- Modulo:        Antirrebote
-- Descripción:   Eliminación de los rebotes en los interruptores
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ANTIRREBOTE is
    Port ( CLK          : in  STD_LOGIC;
           CAPTURA      : in  STD_LOGIC;
                  CAP            : out STD_LOGIC);
end ANTIRREBOTE;

architecture Arq of ANTIRREBOTE is

SIGNAL CNT: STD_LOGIC_VECTOR (2 DOWNTO 0) := (OTHERS => '0');

begin

process (CLK,CNT,CAPTURA)
  begin
    if CAPTURA = '0' then
      CNT <= "000";
    elsif (CLK'event and CLK = '1') then
      if (CNT /= "110") then
            CNT <= CNT + 1;
            end if;
    end if;
    if (CNT = "101") and (CAPTURA = '1') then
            CAP <= '1';
       else
```

```
            CAP <= '0';
       end if;
end process;

end Arq;
```

A7

```
----------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               PWM
-- Descripción:          Módulo que proporciona los diferentes PWM de
                         prueba
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PWM is
    Port (  clk,rst,IND, rst_int: in  STD_LOGIC;
                         giro: out std_logic_vector(1 downto 0);
                   sal_pwm : out std_logic);
end PWM;

architecture Arq of PWM is


signal cnt: std_logic_vector (12 downto 0);
signal aux, a: std_logic_vector (11 downto 0);
signal clk1M, alto:  std_logic;
signal sel: std_logic_vector(4 downto 0);


begin

process (rst_int, rst)
begin
      if rst='1' then
            alto <= '0';
      elsif rst_int'event and rst_int = '0' then
            alto <= '1';
      end if;
end process;

-- Señal de Referencia (50% ciclo útil)
process (clk,rst,cnt)
begin
      if rst='1' or cnt=X"032" then -- 20ns * 50 ciclos = referencia 1us
         cnt <= (others => '0');
            clk1M <= '1';
```

```
      elsif clk'event and clk='1' then
            cnt <= cnt + 1 ;
            if cnt= X"019" then     --20ns * 25 ciclos = 1/2 Periodo
                  clk1M <= '0';
            end if;
      end if;
end process;


process (clk1M,rst,aux,a)
begin
      if rst='1' or aux=X"3E8" then  -- Tiempo (1000us) x064=100_d
         aux <= (others => '0');
            sal_pwm <= '1';
      elsif clk1M'event and clk1M='1' then
            aux <= aux + 1 ;
            if aux = a then
                  sal_pwm <= '0';
            end if;
      end if;
end process;

--Detecta la señal IND de la maquina de estados para incrementar el
contador "sel"
process(rst,IND,alto)
begin
      if rst='1' or alto='1' then
            sel <= (others => '0');
      elsif IND'event and IND ='1' then
            sel <= sel+1;
      end if;
end process;

--"Seleccona" el %CU del PWM
process (clk1M,rst,alto)
      begin
            if rst ='1' or alto='1' then
                  a <= x"012";            --valor con el que inicia al
resetear
                  giro <= "01";
            elsif clk1M'event and clk1M ='1' then

                  if sel="01011" then--Si ya llegó a la velocidad máxima
entonces cambia el sentido de giro
                        giro<="10";
                  end if;

                  case(sel) is
                        --Aumentando el PWM
                        when "00000" => a <= x"012";--pwm
                        when "00001" => a <= x"059";--pwm90us
                        when "00010" => a <= x"06D";--pwm110us
                        when "00011" => a <= x"08B";--pwm140us
                        when "00100" => a <= x"09F";--pwm160us
                        when "00101" => a <= x"0BD";--pwm190us
                        when "00110" => a <= x"0DB";--pwm220us
                        when "00111" => a <= x"0EF";--pwm240us
```

```
                        when "01000" => a <= x"103";--pwm260us
                        when "01001" => a <= x"117";--pwm280us
                        when "01010" => a <= x"12B";--pwm300us
                        --Disminuyendo el PWM
                        when "01011" => a <= x"12B";--pwm3000us
                        when "01100" => a <= x"117";--pwm280us
                        when "01101" => a <= x"103";--pwm260us
                        when "01110" => a <= x"0EF";--pwm240us
                        when "01111" => a <= x"0DB";--pwm220us
                        when "10000" => a <= x"0BD";--pwm190us
                        when "10001" => a <= x"09F";--pwm160us
                        when "10010" => a <= x"08B";--pwm140us
                        when "10011" => a <= x"06D";--pwm110us
                        when "10100" => a <= x"059";--pwm90us
                        when others => null;
                    end case;
                end if;
end process;

end Arq;
```

A8

```
--------------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               Mensajes
-- Descripción:          Inicialización, configuración y escritura de
                         mensajes
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use work.Componentes_pkg.ALL;

entity Mensajes is
    Port(  clk,rst,men_1,men_2: in STD_LOGIC;
           data:  inout STD_LOGIC_VECTOR( 0 to 3 );
           LCD_E,LCD_RS,LCD_RW:  inout   STD_LOGIC;
           SF_CE0: out STD_LOGIC );
end Mensajes;

architecture Arq of Mensajes is

signal listo: std_logic_vector (7 downto 0);
signal LCD_E_1, LCD_E_2, LCD_E_3,LCD_E_4,LCD_E_5,LCD_E_6,LCD_E_7,LCD_E_8
: std_logic;
signal LCD_RS_1, LCD_RS_2,
LCD_RS_3,LCD_RS_4,LCD_RS_5,LCD_RS_6,LCD_RS_7,LCD_RS_8 : std_logic;
signal LCD_RW_1, LCD_RW_2,
LCD_RW_3,LCD_RW_4,LCD_RW_5,LCD_RW_6,LCD_RW_7,LCD_RW_8 : std_logic;
signal DATA_1, DATA_2, DATA_3, DATA_4, DATA_5,DATA_6,DATA_7,DATA_8 :
std_logic_vector( 0 to 3 );
```

```
begin

SF_CE0 <= '1';  -- Inhabilitar acceso a StrataFlash

with listo select
LCD_E <= LCD_E_1 when "00000000",
                  LCD_E_2 when "00000001",
                  LCD_E_3 when "00000011",
                  LCD_E_4 when "00000111",
                  LCD_E_5 when "00001111",
                  LCD_E_6 when "00011111",
                  LCD_E_7 when "00111111",
                  LCD_E_8 when "01111111",
                  '0' when others;

with listo select
LCD_RS <= LCD_RS_1 when "00000000",
                   LCD_RS_2 when "00000001",
                   LCD_RS_3 when "00000011",
                   LCD_RS_4 when "00000111",
                   LCD_RS_5 when "00001111",
                   LCD_RS_6 when "00011111",
                   LCD_RS_7 when "00111111",
                   LCD_RS_8 when "01111111",
                   '0' when others;

with listo select
LCD_RW <= LCD_RW_1 when "00000000",
                   LCD_RW_2 when "00000001",
                   LCD_RW_3 when "00000011",
                   LCD_RW_4 when "00000111",
                   LCD_RW_5 when "00001111",
                   LCD_RW_6 when "00011111",
                   LCD_RW_7 when "00111111",
                   LCD_RW_8 when "01111111",
                   '0' when others;

with listo select
DATA <=  DATA_1 when "00000000",
                  DATA_2 when "00000001",
                  DATA_3 when "00000011",
                  DATA_4 when "00000111",
                  DATA_5 when "00001111",
                  DATA_6 when "00011111",
                  DATA_7 when "00111111",
                  DATA_8 when "01111111",
                  "0000" when others;

com0: LCD_init port map ( clk, rst, LCD_E_1,  LCD_RS_1,
LCD_RW_1,listo(0),DATA_1);
com1: mensaje1 port map ( clk, rst, listo(0), LCD_E_2, LCD_RS_2,
LCD_RW_2,listo(1),DATA_2);
com2: mensaje2 port map ( clk, rst, listo(1), LCD_E_3, LCD_RS_3,
LCD_RW_3,listo(2),DATA_3);
com3: mensaje3 port map ( clk, rst, listo(2), LCD_E_4, LCD_RS_4,
LCD_RW_4,listo(3),DATA_4);
com4: mensaje4 port map ( clk, rst, men_1,    LCD_E_5, LCD_RS_5,
```

```
LCD_RW_5,listo(4),DATA_5);
com5: mensaje5 port map ( clk, rst, listo(4), LCD_E_6, LCD_RS_6,
LCD_RW_6,listo(5),DATA_6);
com6: mensaje6 port map ( clk, rst, men_2,    LCD_E_7, LCD_RS_7,
LCD_RW_7,listo(6),DATA_7);
com7: mensaje7 port map ( clk, rst, listo(6), LCD_E_8, LCD_RS_8,
LCD_RW_8,listo(7),DATA_8);

end Arq;
```

A9

```
--------------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               LCD_init
-- Descripción:          Inicialización y configuración
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LCD_init is

    port( clk,rst:      in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
        listo: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end LCD_init;

architecture Behavioral of LCD_init is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 24 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (  -- Inicialización
                          X"380B71B0", -- 15ms
                          X"3000000C", -- 230ns
                          X"380320C8", -- 4.1ms
                          X"3000000C", -- 230ns
                          X"38001388", -- 100us
                          X"3000000C", -- 230ns
                          X"280007D0", -- 40us
                          X"2000000C", -- 230ns
                          -- Configuración
                          -- Function Set 0x28
                          X"280007D0", -- 40us
                          X"2000000C", -- 230ns
                          X"88000032", -- 1us
```

```
                        X"8000000C", -- 230ns
                        -- Entry Mode 0x06
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"68000032", -- 1us
                        X"6000000C", -- 230ns
                        -- Display On 0x0C
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"C8000032", -- 1us
                        X"C000000C", -- 230ns
                        -- Clear Display 0x01
                        X"080007D0", -- 40us
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C", -- 230ns
                        X"10014050" -- 1.64ms
                        );
begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
      listo <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
                  listo <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A10

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje1
-- Descripción:         ->Gyro-Tech<-
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje1 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
        listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje1;

architecture Behavioral of mensaje1 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 112 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (     -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"28000032", -- 1us
                        X"2000000C", -- 230ns

                        -- Escribiendo datos
                        X"7CFFFFFF", -- 335ms   --    ->
                        X"7400000C", -- 230ns
                        X"EC000032", -- 1us
                        X"E400000C",  -- 230ns

                        X"2C8FFFFF", -- 335ms   --    _
                        X"2400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"4CFFFFFF", -- 350ms   --    G
                        X"4400000C", -- 230ns
                        X"7C000032", -- 1us
                        X"7400000C", -- 230ns

                        X"7CFFFFFF", -- 350ms   --    y
                        X"7400000C", -- 230ns
                        X"9C000032", -- 1us
```

```
X"9400000C", -- 230ns

X"7CFFFFFF", -- 350ms   --    r
X"7400000C", -- 230ns
X"2C000032", -- 1us
X"2400000C", -- 230ns

X"6CFFFFFF", -- 350ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"BCFFFFFF", -- 350ms   --    -
X"B400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"5CFFFFFF", -- 350ms   --    T
X"5400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6CFFFFFF", -- 350ms   --    e
X"6400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns

X"6CFFFFFF", -- 350ms   --    c
X"6400000C", -- 230ns
X"3C000032", -- 1us
X"3400000C", -- 230ns

X"6CFFFFFF", -- 350ms   -- h
X"6400000C", -- 230ns
X"8C000032", -- 1us
X"8400000C", -- 230ns

X"2C8FFFFF", -- 335ms   --    _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns


X"7CFFFFFF", -- 350ms   -- <-
X"7400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns
```

```
X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift left
X"1000000C", -- 230ns
X"88000032", -- 1us
X"8000000C",  -- 230ns

X"08FFFFFF", -- 335ms   -- Finalmente Clear
```

```
                                                    Display 0x01
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C",  -- 230ns
                        X"10014050"  -- 1.64ms

                    );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
      listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A11

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje2
-- Descripción:         ESIME-IPN
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje2 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
```

```
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje2;

architecture Behavioral of mensaje2 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 92 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (      -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"38000032", -- 1us
                        X"3000000C", -- 230ns

                        -- Escribiendo datos
                        X"4CFFFFFF", -- 335ms   --    E
                        X"4400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"5CFFFFFF", -- 350ms   --    S
                        X"5400000C", -- 230ns
                        X"3C000032", -- 1us
                        X"3400000C", -- 230ns

                        X"4CFFFFFF", -- 350ms   --    I
                        X"4400000C", -- 230ns
                        X"9C000032", -- 1us
                        X"9400000C", -- 230ns

                        X"4CFFFFFF", -- 350ms   --    M
                        X"4400000C", -- 230ns
                        X"DC000032", -- 1us
                        X"D400000C", -- 230ns

                        X"4CFFFFFF", -- 350ms   --    E
                        X"4400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"BCFFFFFF", -- 350ms   --    -
                        X"B400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"4CFFFFFF", -- 350ms   --    I
                        X"4400000C", -- 230ns
                        X"9C000032", -- 1us
                        X"9400000C", -- 230ns

                        X"5CFFFFFF", -- 350ms   --    P
```

```
X"5400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"4CFFFFFF", -- 350ms   --    N
X"4400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
X"C000000C",  -- 230ns

X"18FFFFFF", -- 335ms   -- shift right
X"1000000C", -- 230ns
X"C8000032", -- 1us
```

```
                        X"C000000C",  -- 230ns

                        X"18FFFFFF", -- 335ms   -- shift right
                        X"1000000C", -- 230ns
                        X"C8000032", -- 1us
                        X"C000000C",  -- 230ns

                        X"18FFFFFF", -- 335ms   -- shift right
                        X"1000000C", -- 230ns
                        X"C8000032", -- 1us
                        X"C000000C",  -- 230ns

                        X"08FFFFFF", -- 335ms   -- Finalmente Clear
                                                   Display 0x01
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C",  -- 230ns
                        X"10014050"  -- 1.64ms

                    );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
      listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A12

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje3
-- Descripción:         Presione Enter para continuar
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje3 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje3;

architecture Behavioral of mensaje3 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 120 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (     -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C", -- 230ns

                        -- Escribiendo datos
                        X"5C8FFFFF", -- 335ms   --    P
                        X"5400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"7C8FFFFF", -- 350ms   --    r
                        X"7400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C", -- 230ns

                        X"6C8FFFFF", -- 350ms   --    e
                        X"6400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"7C8FFFFF", -- 350ms   --    s
                        X"7400000C", -- 230ns
                        X"3C000032", -- 1us
```

```
X"3400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    i
X"6400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    n
X"6400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    e
X"6400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns

X"2C8FFFFF", -- 350ms   --    _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"4C8FFFFF", -- 350ms   -- E
X"4400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns


X"6C8FFFFF", -- 350ms   -- n
X"6400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"7C8FFFFF", -- 350ms   --    t
X"7400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    e
X"6400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns

X"7C8FFFFF", -- 350ms   --    r
X"7400000C", -- 230ns
X"2C000032", -- 1us
X"2400000C", -- 230ns

-- Seleccionar DDRAM
X"10014050",  -- 1.64ms
X"C80007D0", -- 40us
X"C000000C", -- 230ns
```

```
X"18000032", -- 1us
X"1000000C", -- 230ns

-- Escribiendo datos
X"7C8FFFFF", -- 335ms   --    p
X"7400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"7C8FFFFF", -- 350ms   --    r
X"7400000C", -- 230ns
X"2C000032", -- 1us
X"2400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"2C8FFFFF", -- 335ms   --    _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    c
X"6400000C", -- 230ns
X"3C000032", -- 1us
X"3400000C", -- 230ns

X"6C8FFFFF", -- 335ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"6C0FFFFF", -- 350ms   --    n
X"6400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"7C8FFFFF", -- 335ms   --    t
X"7400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    i
X"6400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"6C8FFFFF", -- 335ms   --    n
X"6400000C", -- 230ns
X"EC000032", -- 1us
```

```
                        X"E400000C", -- 230ns

                        X"7C8FFFFF", -- 350ms   --    u
                        X"7400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"6C8FFFFF", -- 335ms   --    a
                        X"6400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C", -- 230ns

                        X"7C8FFFFF", -- 350ms   --    r
                        X"7400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C" -- 230ns



                    );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
      listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A13

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje4
-- Descripción:         Iniciando Prueba
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje4 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje4;

architecture Behavioral of mensaje4 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 85 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (

                    -- Clear Display 0x01
           X"08FFFFFF", -- 335ms
                    X"0000000C", -- 230ns
                    X"18000032", -- 1us
                    X"1000000C",  -- 230ns
                    X"10014050",  -- 1.64ms

           -- Seleccionar DDRAM
                    X"880007D0", -- 40us
                    X"8000000C", -- 230ns
                    X"08000032", -- 1us
                    X"0000000C", -- 230ns

                    -- Escribiendo datos
                    X"4C8FFFFF", -- 335ms   --    I
                    X"4400000C", -- 230ns
                    X"9C000032", -- 1us
                    X"9400000C", -- 230ns

                    X"6C8FFFFF", -- 350ms   --    n
                    X"6400000C", -- 230ns
                    X"EC000032", -- 1us
                    X"E400000C", -- 230ns
```

```
X"6C8FFFFF", -- 350ms   --    i
X"6400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    c
X"6400000C", -- 230ns
X"3C000032", -- 1us
X"3400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    i
X"6400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    n
X"6400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    d
X"6400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6C8FFFFF", -- 350ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"2C8FFFFF", -- 350ms   -- _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns


X"5C8FFFFF", -- 350ms   -- P
X"5400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"7C8FFFFF", -- 350ms   --    r
X"7400000C", -- 230ns
X"2C000032", -- 1us
X"2400000C", -- 230ns

X"7C8FFFFF", -- 350ms   --    u
X"7400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns
```

```
                        X"6C8FFFFF", -- 350ms   --    e
                        X"6400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"6C8FFFFF", -- 350ms   --    b
                        X"6400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C", -- 230ns

                        X"6C8FFFFF", -- 350ms   --    a
                        X"6400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C", -- 230ns

                        X"2CFFFFFF", -- 350ms   --    _
                        X"2400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"2CFFFFFF", -- 350ms   --    _
                        X"2400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"088FFFFF", -- 335ms   -- Finalmente Clear
                                                   Display 0x01
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C",  -- 230ns
                        X"10014050"  -- 1.64ms

                    );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
        listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <=index;
                  listo_out <= '1';
```

```
            end if;
        else
            CNTR <= CNTR + 1;
        end if;
    end if;
end process;
end Behavioral;
```

A14

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje5
-- Descripción:         >>>>>>>>>>>>>>>>>
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje5 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje5;

architecture Behavioral of mensaje5 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 31 ) := X"00000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 67 ) of STD_LOGIC_VECTOR( 0 to 39 );

signal ROM: ROM_TYPE:= (

                        -- Seleccionar DDRAM
                        X"88000007D0", -- 40us
                        X"800000000C", -- 230ns
                        X"0800000032", -- 1us
                        X"000000000C", -- 230ns

                        -- Escribiendo datos
                        X"3C00FFFFFF", -- 335ms --    >
                        X"340000000C", -- 230ns
                        X"EC00000032", -- 1us
                        X"E40000000C", -- 230ns

                        X"3C1EAFFFFF", -- 335ms --    >
                        X"340000000C", -- 230ns
                        X"EC00000032", -- 1us
```

```
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns


X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns

X"3C1EAFFFFF", -- 335ms --     >
X"340000000C", -- 230ns
X"EC00000032", -- 1us
X"E40000000C", -- 230ns
```

```
                        X"3C1EAFFFFF", -- 335ms --    >
                        X"340000000C", -- 230ns
                        X"EC00000032", -- 1us
                        X"E40000000C", -- 230ns

                        X"3C1EAFFFFF", -- 335ms --    >
                        X"340000000C", -- 230ns
                        X"EC00000032", -- 1us
                        X"E40000000C", -- 230ns

                        X"3C1EAFFFFF", -- 335ms --    >
                        X"340000000C", -- 230ns
                        X"EC00000032", -- 1us
                        X"E40000000C"  -- 230ns

                    );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
        listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 39) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <= index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A15

```
----------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje6
-- Descripción:         Enviando Datos a LabVIEW
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje6 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje6;

architecture Behavioral of mensaje6 is

    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 105 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (

                        -- Clear Display 0x01
                        X"08FFFFFF", -- 335ms
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C",  -- 230ns
                        X"10014050",  -- 1.64ms

                        -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C", -- 230ns

                        -- Escribiendo datos
                        X"4C4FFFFF", -- 335ms   --    E
                        X"4400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    n
                        X"6400000C", -- 230ns
                        X"EC000032", -- 1us
                        X"E400000C", -- 230ns
```

```
X"7C4FFFFF", -- 350ms   --    v
X"7400000C", -- 230ns
X"6C000032", -- 1us
X"6400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    i
X"6400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    n
X"6400000C", -- 230ns
X"EC000032", -- 1us
X"E400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    d
X"6400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns

X"2C4FFFFF", -- 350ms   --    _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"4C4FFFFF", -- 350ms   -- D
X"4400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns


X"6C4FFFFF", -- 350ms   -- a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"7C4FFFFF", -- 350ms   --    t
X"7400000C", -- 230ns
X"4C000032", -- 1us
X"4400000C", -- 230ns

X"6C4FFFFF", -- 350ms   --    o
X"6400000C", -- 230ns
X"FC000032", -- 1us
X"F400000C", -- 230ns
```

```
X"7C4FFFFF", -- 350ms   --    s
X"7400000C", -- 230ns
X"3C000032", -- 1us
X"3400000C", -- 230ns

-- Seleccionar DDRAM
X"10014050",  -- 1.64ms
X"C80007D0", -- 40us
X"C000000C", -- 230ns
X"48000032", -- 1us
X"4000000C", -- 230ns

-- Escribiendo datos
X"6C4FFFFF", -- 335ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"2C4FFFFF", -- 335ms   --    _
X"2400000C", -- 230ns
X"0C000032", -- 1us
X"0400000C", -- 230ns

X"4C4FFFFF", -- 335ms   --    L
X"4400000C", -- 230ns
X"CC000032", -- 1us
X"C400000C", -- 230ns

X"6C4FFFFF", -- 335ms   --    a
X"6400000C", -- 230ns
X"1C000032", -- 1us
X"1400000C", -- 230ns

X"6C4FFFFF", -- 335ms   --    b
X"6400000C", -- 230ns
X"2C000032", -- 1us
X"2400000C", -- 230ns

X"5C4FFFFF", -- 335ms   --    V
X"5400000C", -- 230ns
X"6C000032", -- 1us
X"6400000C", -- 230ns

X"4C4FFFFF", -- 335ms   --    I
X"4400000C", -- 230ns
X"9C000032", -- 1us
X"9400000C", -- 230ns

X"4C4FFFFF", -- 335ms   --    E
X"4400000C", -- 230ns
X"5C000032", -- 1us
X"5400000C", -- 230ns

X"5C4FFFFF", -- 335ms   --    W
X"5400000C", -- 230ns
X"7C000032", -- 1us
X"7400000C"); -- 230ns
```

```
begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
        listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <= index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A16

```
--------------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Mensaje7
-- Descripción:         Prueba Terminada
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity mensaje7 is

    port( clk,rst,listo_in:   in      STD_LOGIC;
          LCD_E, LCD_RS, LCD_RW:  inout   STD_LOGIC;
          listo_out: out STD_LOGIC;
          DATA:  inout STD_LOGIC_VECTOR( 0 to 3 ) );

end mensaje7;

architecture Behavioral of mensaje7 is
```

```
    signal CNTR:    STD_LOGIC_VECTOR( 0 to 23 ) := X"000000";
    signal index:   integer:=0;
    -- Tengamos una definición tipo ROM :  D Y C0 C1 C2 C3 C4 C5
       -- Y contiene (LDC_E, LCD_RS, LCD_RW, 0)
    type ROM_TYPE is array( 0 to 72 ) of STD_LOGIC_VECTOR( 0 to 31 );

signal ROM: ROM_TYPE:= (

                        -- Clear Display 0x01
                        X"08FFFFFF", -- 335ms
                        X"0000000C", -- 230ns
                        X"18000032", -- 1us
                        X"1000000C",  -- 230ns
                        X"10014050",  -- 1.64ms


                        -- Seleccionar DDRAM
                        X"880007D0", -- 40us
                        X"8000000C", -- 230ns
                        X"08000032", -- 1us
                        X"0000000C", -- 230ns

                        -- Escribiendo datos
                        X"5C4FFFFF", -- 335ms   --    P
                        X"5400000C", -- 230ns
                        X"0C000032", -- 1us
                        X"0400000C", -- 230ns

                        X"7C4FFFFF", -- 350ms   --    r
                        X"7400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C", -- 230ns

                        X"7C4FFFFF", -- 350ms   --    u
                        X"7400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    e
                        X"6400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    b
                        X"6400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    a
                        X"6400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C", -- 230ns

                        X"2C4FFFFF", -- 350ms   --    _
                        X"2400000C", -- 230ns
                        X"0C000032", -- 1us
```

```
                        X"0400000C", -- 230ns

                        X"5C4FFFFF", -- 350ms   --    T
                        X"5400000C", -- 230ns
                        X"4C000032", -- 1us
                        X"4400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    e
                        X"6400000C", -- 230ns
                        X"5C000032", -- 1us
                        X"5400000C", -- 230ns

                        X"7C4FFFFF", -- 350ms   -- r
                        X"7400000C", -- 230ns
                        X"2C000032", -- 1us
                        X"2400000C", -- 230ns


                        X"6C4FFFFF", -- 350ms   -- m
                        X"6400000C", -- 230ns
                        X"DC000032", -- 1us
                        X"D400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    i
                        X"6400000C", -- 230ns
                        X"9C000032", -- 1us
                        X"9400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    n
                        X"6400000C", -- 230ns
                        X"EC000032", -- 1us
                        X"E400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    a
                        X"6400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    d
                        X"6400000C", -- 230ns
                        X"4C000032", -- 1us
                        X"4400000C", -- 230ns

                        X"6C4FFFFF", -- 350ms   --    a
                        X"6400000C", -- 230ns
                        X"1C000032", -- 1us
                        X"1400000C" -- 230ns


                );

begin

process(CLK,RST,CNTR,index) is
    begin
    if RST = '1' then
        INDEX <=0;
```

```
      listo_out <='0';
        CNTR <= (others=>'0');
    elsif rising_edge( CLK ) and listo_in = '1' then

            if CNTR >= ROM(index)(8 to 31) then

                CNTR <= (others=>'0');
                DATA(0 to 3) <= ROM(index)(0 to 3);
                LCD_E   <= ROM(index)(4);
                LCD_RS  <= ROM(index)(5);
                LCD_RW  <= ROM(index)(6);

                if index < ROM'high then
                        index <= index + 1;
                else
                        index <= index;
                  listo_out <= '1';
                end if;
            else
                CNTR <= CNTR + 1;
            end if;
    end if;
end process;
end Behavioral;
```

A17

```
--------------------------------------------------------------------------------
-- Escuela:             E.S.I.M.E. IPN
-- Programadores:       Luis Sánchez Márquez
--                      Ricardo Flores Martínez
-- Asesor:              M. en C. Luis Martin Flores Nava
-- Proyecto:            TESIS
-- Modulo:              Trans
-- Descripción:         Transmisión de datos del FPGA a LabVIEW
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use WORK.Componentes_pkg.ALL;

entity Trans is
port (      clk, rst: in  STD_LOGIC;
                tx : out  STD_LOGIC;
                etx : inout STD_LOGIC;
                V, T, P : in std_logic_vector(15 downto 0));
end Trans;

architecture Arq of Trans is


begin

com1: serial port map (clk,rst,V,T,P,etx,tx);
com2: generador port map (clk,rst,etx);

end Arq;
```

A18

```
----------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               serial
-- Descripción:          Transmite las tramas a una velocidad de 9600
                         baudios
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity serial is
  port (clk, rst: in  STD_LOGIC;
        V,T,P: in STD_LOGIC_VECTOR(15 downto 0);
        etx: in STD_LOGIC;
              tx: out  STD_LOGIC);
end serial;

architecture Arq of serial is

signal regtx:  STD_LOGIC_VECTOR (7 downto 0);
signal cnttx:     STD_LOGIC_VECTOR (15 downto 0);
signal ttx:       STD_LOGIC_VECTOR (3 downto 0);

signal cargador:  STD_LOGIC_VECTOR (3 downto 0);
signal V1,V2:     STD_LOGIC_VECTOR (7 downto 0);
signal T1,T2:     STD_LOGIC_VECTOR (7 downto 0);
signal P1,P2:     STD_LOGIC_VECTOR (7 downto 0);

-------------- VELOCIDAD DE TRANSMISIÓN ----------------------------
constant baudtx: STD_LOGIC_VECTOR(15 downto 0):="0001010001011000";
                                                 ---(1/9600)/20ns  = 5208


begin


-- Proceso para cargar los datos
 process(rst, etx)
  begin
      if rst='1' then
            V1<=(others=>'1');
            V2<=(others=>'1');
            T1<=(others=>'1');
            T2<=(others=>'1');
            P1<=(others=>'1');
            P2<=(others=>'1');

      elsif(etx'event and etx='0')then

            V1<= '1' & V(14 downto 8);
```

```
            V2<=V(7 downto 0);

            T1<='1' & T(14 downto 8);
            T2<=T(7 downto 0);

            P1<='1' & P(14 downto 8);
            P2<=P(7 downto 0);

      end if;
 end process;

-- Reloj de transmisión
 process (clk,rst,etx,cnttx,ttx)
  begin
      if (rst='1' or etx='0')then
             cnttx <= (others=>'0');
             ttx   <= "0000";
          cargador <= "0000";
      elsif (clk'event and clk='1' and etx='1')then
             cnttx <= cnttx+1;
            if (cnttx=baudtx)then
                  ttx<=ttx+1;
                  cnttx<=(others=>'0');
            if (ttx="1010")then
                  cargador <= cargador + 1;
                  ttx <= "0000";
            end if;
            end if;
      end if;
end process;

-- Asignacion de Caracter
with cargador select
      regtx <=    V1    when "0000",
                        V2    when "0001",
                        T1    when "0010",
                        T2    when "0011",
                        P1    when "0100",
                        P2    when "0101",
                        X"00" when others;

-- Protocolo de transmisión
with ttx select
            tx    <=    '1' when "0000",
                     '0' when "0001",     --bit de start
             regtx(0) when "0010",  --inicia transmision de dato
             regtx(1) when "0011",
             regtx(2) when "0100",
             regtx(3) when "0101",
             regtx(4) when "0110",
             regtx(5) when "0111",
             regtx(6) when "1000",
             regtx(7) when "1001",  --fin de transmision de dato
                       '1' when "1010",  --bit de stop
                       '1'         when others;
end Arq;
```

A19

```
----------------------------------------------------------------------------
-- Escuela:              E.S.I.M.E. IPN
-- Programadores:        Luis Sánchez Márquez
--                       Ricardo Flores Martínez
-- Asesor:               M. en C. Luis Martin Flores Nava
-- Proyecto:             TESIS
-- Modulo:               generador
-- Descripción:          Genera las señales de control de la transmisión
----------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity generador is
port (clk, rst: in std_logic;
           etx: inout std_logic);
end generador;

architecture Arq of generador is
signal contador: STD_LOGIC_VECTOR (19 downto 0) :=(others=>'0');
signal stop: STD_LOGIC_VECTOR (4 downto 0) :=(others=>'0');

begin

--- Proceso para generar la señal etx
process(clk,rst,contador,stop)
   begin
    if (rst='1' or contador <= X"0000A" or stop = "10111") then -- 11
                                                  ciclos de 50 MHz(220 ns)
           etx<='0';
       elsif (clk 'event and clk='1') then
           etx<='1';
       end if;
  end process;

--- Proceso para detener etx
process(etx,rst,stop)
   begin
      if rst='1' then
           stop <= (others=>'0');

   elsif (etx'event and etx='1')then
           stop<=stop+1;
      end if;
end process;

--Para 66 bits(trama 11 bits * # de variables V,T,P ) a 9600 baudios
6.875ms/20ns  contador de 0 a 343750 (X"53EC6")

process(clk,rst,contador)
   begin

      if (rst='1' or contador = X"53EC6" ) then
       contador <= (others=>'0');
    elsif (clk 'event and clk='0') then
```

```
            contador <= contador + 1;
        end if;
    end process;

end Arq;
```

Apéndice B

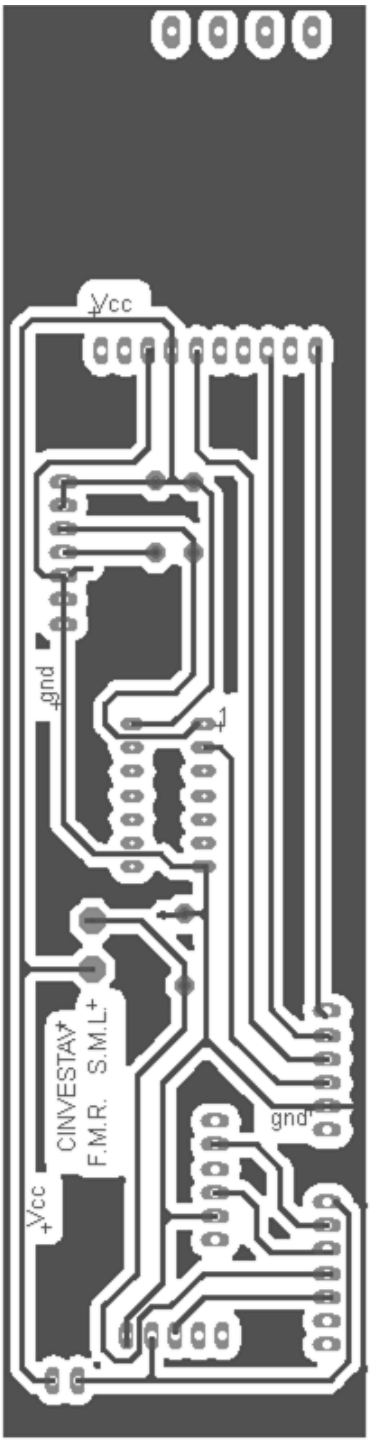
Código del AVR

```
#define F_CPU 4000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>


void init_ADC(void);
void init_USART(void);
void TX_USART(unsigned long int MS, unsigned long int LS);

unsigned int contador=0;

//Prom_Temp_LS: Se almacena el promedio de los bits menos significativos
de la temperatura
//Prom_Temp_MS: Se almacena el promedio de los bits más significativos de
la temperatura

//Prom_w_LS: Se almacena el promedio de los bits menos significativos de
la velocidad angular (w)
//Prom_w_MS: Se almacena el promedio de los bits más significativos de la
velocidad angular (w)
int main(void)
      {

            unsigned long int Temp=0;
            unsigned long int Prom_Temp=0;
            unsigned long int Prom_Temp_LS=0;
            unsigned long int Prom_Temp_MS=0;

            unsigned long int w=0;
            unsigned long int Prom_w=0;
            unsigned long int Prom_w_LS=0;
            unsigned long int Prom_w_MS=0;

            init_ADC();
            init_USART();

            do{

                  contador=0;
                  Temp=0;
                  Prom_Temp=0;
                  Prom_Temp_LS=0;
                  Prom_Temp_MS=0;
                  w=0;
                  Prom_w=0;
                  Prom_w_LS=0;
                  Prom_w_MS=0;

                  do{
```

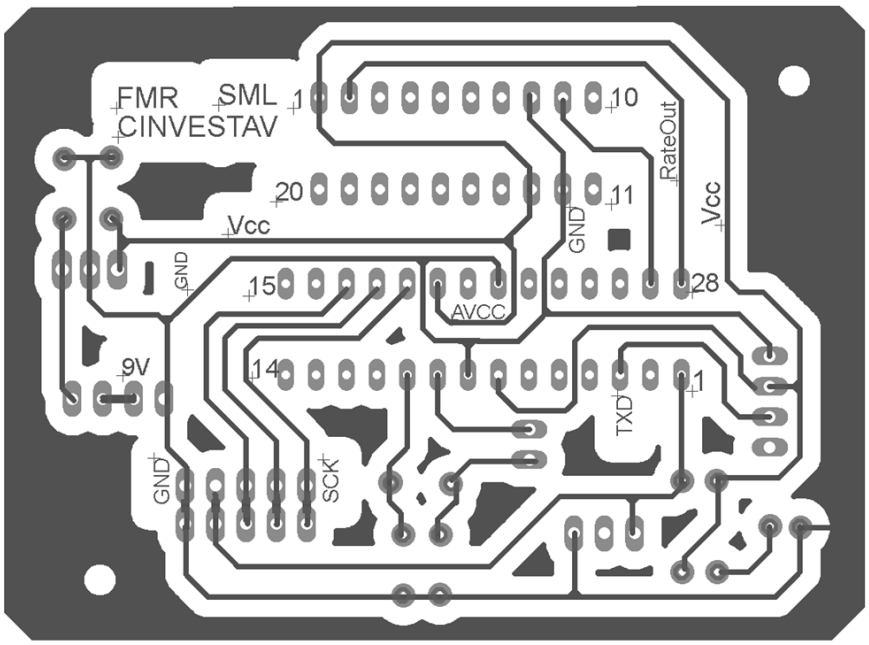
```
if(contador==0)   //Se coloca por un problema al simplemente seleccionar
                  ADMUX=Canal este problema se ve en simulación
ADMUX |= 0x05;
else
ADMUX |= _BV(MUX0); //Enciende el bit MUX0 (Selección del canal 5)

ADCSRA |= _BV(ADSC);    //Inicia la conversión


loop_until_bit_is_set(ADCSRA,ADIF);

ADCSRA |= _BV(ADIF);    //a ADCSRA asignale lo que ya contiene y además
                        borra al bit ADIF


w = w + ADC;


/****** Se repite para el canal 4 ********/
ADMUX &= ~(_BV(MUX0));  // Borra el bit MUX0 de ADMUX (Selección canal 4)
ADCSRA |= _BV(ADSC);    //Inicia la conversión

loop_until_bit_is_set(ADCSRA,ADIF);

ADCSRA |= _BV(ADIF);

Temp = Temp + ADC;

contador = contador + 1;

}while(contador<16);    //ciclo para las 16 mediciones de ambos canales


                  Prom_w = w/16;

                  // Separando en dos bytes
                  Prom_w_MS = Prom_w/256;
                  Prom_w_LS = Prom_w%256;

                  _delay_ms(1010);  //Retardo que servirá para saber
                              cuándo inicia la trama (sincronizar
                              Transmisor y Receptor)

                  TX_USART(Prom_w_MS,Prom_w_LS); // Transmite

                  Prom_Temp = Temp/16;

                  // Separando en dos bytes
                  Prom_Temp_MS = Prom_Temp/256;
                  Prom_Temp_LS = Prom_Temp%256;

                  TX_USART(Prom_Temp_MS,Prom_Temp_LS); // Transmite

            }while(1); //ciclo infinito
```

```
}//Fin del main()

void init_ADC()
	{
		DDRC = 0x00; //Para los canales ADC4 y ADC5 entradas
		DDRD=0xFF;
		ADCSRB = 0x00;
		ADCSRA = _BV(ADEN)|_BV(ADPS2)|_BV(ADPS0); //Modo conversión
		                                            simple
		//125 kHZ @4 MHZ
		ADMUX=_BV(REFS0);
		PORTD = 0x01;
	}


void init_USART(void)
	{
		UBRR0L=25;              //Configura USART para
		UCSR0B=_BV(TXEN0);      //baud rate 9600 a 4 Mhz
		UCSR0C=(3<<UCSZ00);     //Asíncrona, Paridad desactivada, 1
		                          bit de parada, 8 bits de datos

	}



	void TX_USART(unsigned long int MS, unsigned long int LS)
	{

		loop_until_bit_is_set(UCSR0A,UDRE0); //Espera que el buffer
		                                     de transmision este vacio

		UDR0=MS;   // Envia dato
		loop_until_bit_is_set(UCSR0A,UDRE0);

		UDR0=LS;
		loop_until_bit_is_set(UCSR0A,UDRE0);

	}
```

Apéndice C



Diseño del circuito impreso de la tarjeta hija.



Diseño del circuito impreso de la tarjeta para el bloque ACT.

Apéndice D

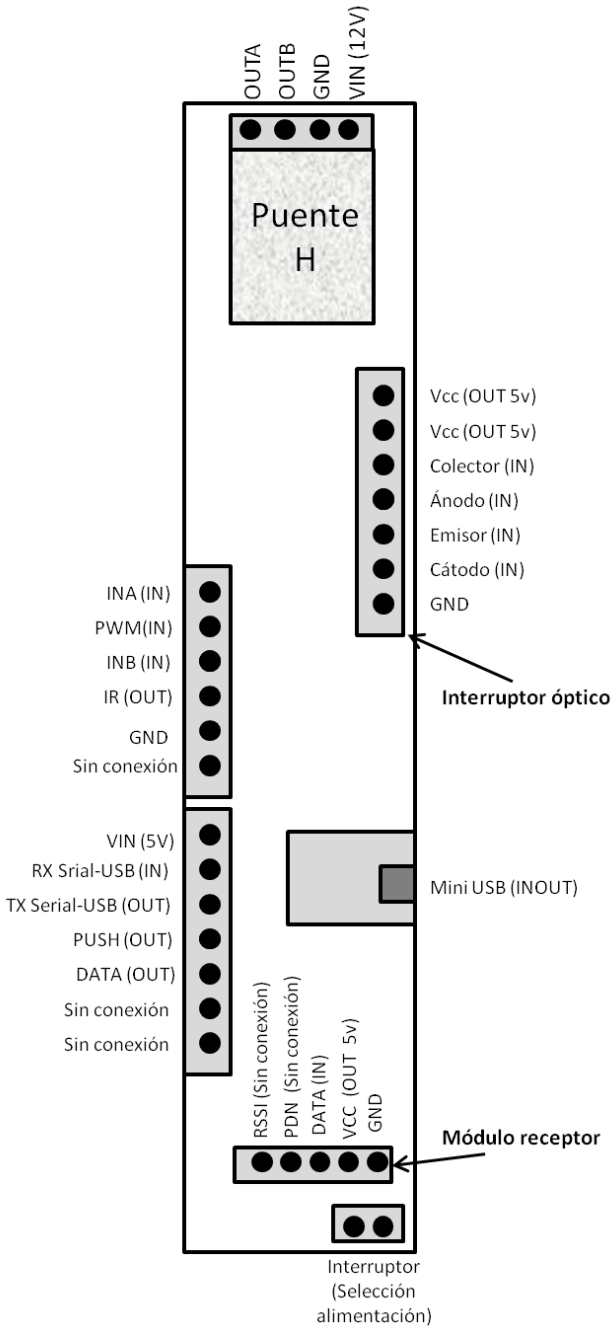


Diagrama de conexiones de entradas y salidas de la tarjeta hija.

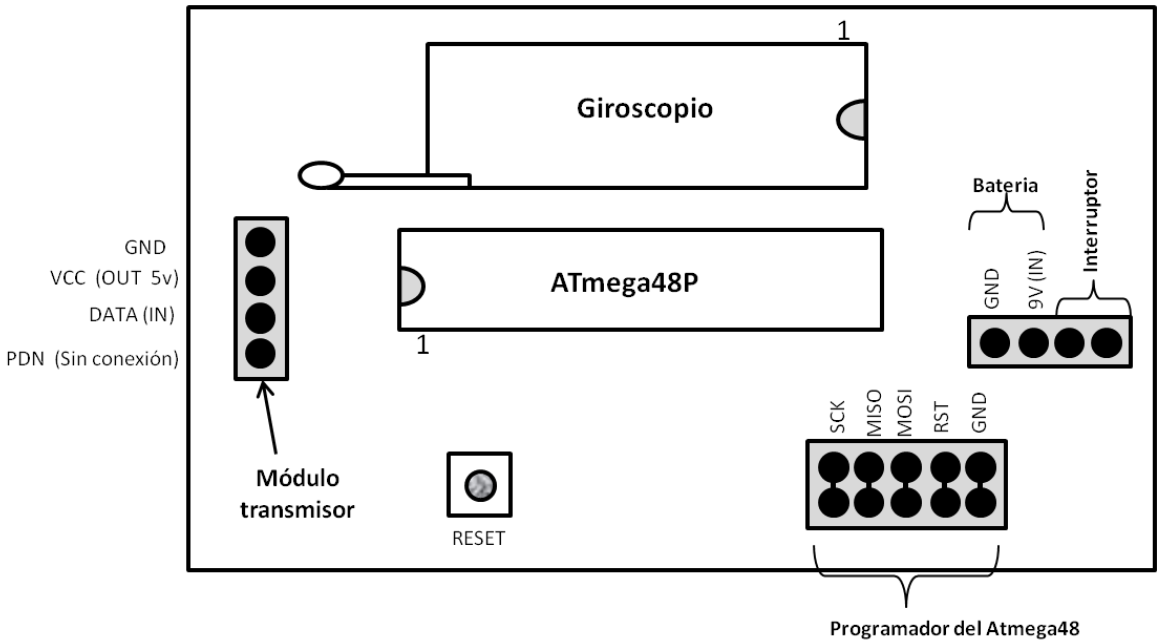


Diagrama de conexiones de la tarjeta para el bloque ACT

Apéndice E

Fuerza Coriolis

El efecto Coriolis, descrito en 1835 por el científico francés Gaspard-Gustave Coriolis, es el efecto que se observa en un sistema de referencia en rotación (y por lo tanto no inercial) cuando un cuerpo se encuentra en movimiento respecto de dicho sistema de referencia. Este efecto consiste en la existencia de una aceleración relativa del cuerpo en dicho sistema en rotación. Esta aceleración es siempre perpendicular al eje de rotación del sistema y a la velocidad del cuerpo. El efecto Coriolis hace que un objeto que se mueve sobre el radio de un disco en rotación tienda a acelerarse con respecto a ese disco según si el movimiento es hacia el eje de giro o alejándose de este. Debido a que el objeto sufre una aceleración desde el punto de vista del observador en rotación, es como si para este existiera una fuerza sobre el objeto que lo acelera. A esta fuerza se le llama fuerza de Coriolis, y no es una fuerza real en el sentido de que no hay nada que la produzca. Se trata pues de una fuerza inercial o ficticia, que se introduce para explicar, desde el punto de vista del sistema en rotación, la aceleración del cuerpo, cuyo origen está en realidad, en el hecho de que el sistema de observación esta rotando.

Apéndice F

ANALOG DEVICES

±300°/s Single Chip Yaw Rate Gyro with Signal Conditioning

ADXRS300

FEATURES

Complete rate gyroscope on a single chip
Z-axis (yaw rate) response
High vibration rejection over wide frequency
2000 *g* powered shock survivability
Self-test on digital command
Temperature sensor output
Precision voltage reference output
Absolute rate output for precision applications
5 V single-supply operation
Ultrasmall and light (< 0.15 cc, < 0.5 gram)

APPLICATIONS

Vehicle chassis rollover sensing
Inertial measurement units
Platform stabilization

GENERAL DESCRIPTION

The ADXRS300 is a complete angular rate sensor (gyroscope) that uses Analog Devices' surface-micromachining process to make a functionally complete and low cost angular rate sensor integrated with all of the required electronics on one chip. The manufacturing technique for this device is the same high volume BIMOS process used for high reliability automotive airbag accelerometers.

The output signal, RATEOUT (1B, 2A), is a voltage proportional to angular rate about the axis normal to the top surface of the package (see Figure 4). A single external resistor can be used to lower the scale factor. An external capacitor is used to set the bandwidth. Other external capacitors are required for operation (see Figure 5).

A precision reference and a temperature output are also provided for compensation techniques. Two digital self-test inputs electromechanically excite the sensor to test proper operation of both sensors and the signal conditioning circuits. The ADXRS300 is available in a 7 mm × 7 mm × 3 mm BGA chip-scale package.

FUNCTIONAL BLOCK DIAGRAM

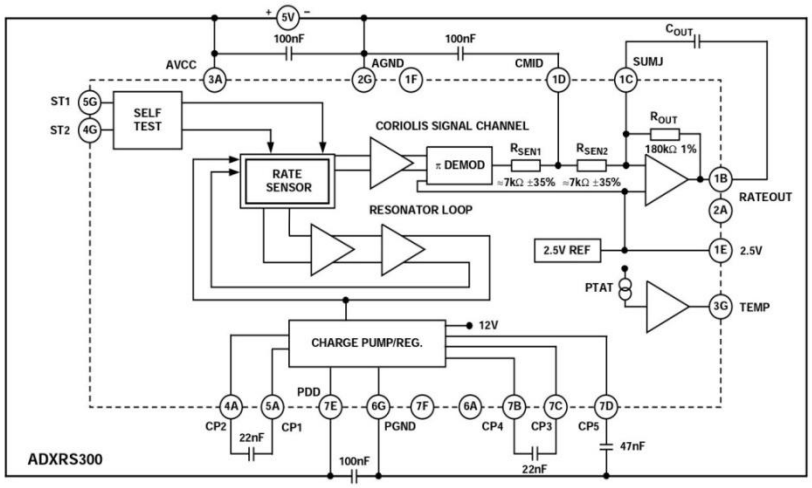


Figure 1.

Rev. B

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 **www.analog.com**
Fax: 781.326.8703

ADXRS300

TABLE OF CONTENTS

REVISION HISTORY

3/04—Data Sheet Changed from Rev. A to Rev. B
Updated Format ..... Universal
Changes to Table 1 Conditions 3
Added Evaluation Board to Ordering Guide 8

3/03—Data Sheet Changed from Rev. 0 to Rev. A
Edit to Figure 3 5

SPECIFICATIONS

@T_A = 25°C, V_S = 5 V, Angular Rate = 0°/s, Bandwidth = 80 Hz (C_{OUT} = 0.01 µF), ±1*g*, unless otherwise noted.

Table 1.

Parameter	Conditions	ADXRS300ABG			Unit
		Min[1]	Typ	Max[1]	
SENSITIVITY	Clockwise rotation is positive output				
Dynamic Range[2]	Full-scale range over specifications range	±300			°/s
Initial	@25°C	4.6	5	5.4	mV/°/s
Over Temperature[3]	V_S = 4.75 V to 5.25 V	4.6	5	5.4	mV/°/s
Nonlinearity	Best fit straight line		0.1		% of FS
NULL					
Initial Null		2.3	2.50	2.7	V
Over Temperature[3]	V_S = 4.75 V to 5.25 V	2.3		2.7	V
Turn-On Time	Power on to ±½°/s of final		35		ms
Linear Acceleration Effect	Any axis		0.2		°/s/*g*
Voltage Sensitivity	V_{CC} = 4.75 V to 5.25 V		1		°/s/V
NOISE PERFORMANCE					
Rate Noise Density	@25°C		0.1		°/s/√Hz
FREQUENCY RESPONSE					
3 dB Bandwidth (User Selectable)[4]	22 nF as comp cap (see the Setting Bandwidth section)		40		Hz
Sensor Resonant Frequency			14		kHz
SELF-TEST INPUTS					
ST1 RATEOUT Response[5]	ST1 pin from Logic 0 to 1	−150	−270	−450	mV
ST2 RATEOUT Response[5]	ST2 pin from Logic 0 to 1	+150	+270	+450	mV
Logic 1 Input Voltage	Standard high logic level definition	3.3			V
Logic 0 Input Voltage	Standard low logic level definition			1.7	V
Input Impedance	To common		50		kΩ
TEMPERATURE SENSOR					
V_{OUT} at 298°K			2.50		V
Max Current Load on Pin	Source to common			50	µA
Scale Factor	Proportional to absolute temperature		8.4		mV/°K
OUTPUT DRIVE CAPABILITY					
Output Voltage Swing	I_{OUT} = ±100 µA	0.25		V_S − 0.25	V
Capacitive Load Drive		1000			pF
2.5 V REFERENCE					
Voltage Value		2.45	2.5	2.55	V
Load Drive to Ground	Source		200		µA
Load Regulation	0 < I_{OUT} < 200 µA		5.0		mV/mA
Power Supply Rejection	4.75 V_S to 5.25 V_S		1.0		mV/V
Temperature Drift	Delta from 25°C		5.0		mV
POWER SUPPLY					
Operating Voltage Range		4.75	5.00	5.25	V
Quiescent Supply Current			6.0	8.0	mA
TEMPERATURE RANGE					
Specified Performance Grade A	Temperature tested to max and min specifications	−40		+85	°C

[1] All minimum and maximum specifications are guaranteed. Typical specifications are not tested or guaranteed.
[2] Dynamic range is the maximum full-scale measurement range possible, including output swing range, initial offset, sensitivity, offset drift, and sensitivity drift at 5 V supplies.
[3] Specification refers to the maximum extent of this parameter as a worst-case value of T_{MIN} or T_{MAX}.
[4] Frequency at which response is 3 dB down from dc response with specified compensation capacitor value. Internal pole forming resistor is 180 kΩ. See the Setting Bandwidth section.
[5] Self-test response varies with temperature. See the Self-Test Function section for details.

ABSOLUTE MAXIMUM RATINGS

Table 2.

Parameter	Rating
Acceleration (Any Axis, Unpowered, 0.5 ms)	2000 *g*
Acceleration (Any Axis, Powered, 0.5 ms)	2000 *g*
$+V_S$	−0.3 V to +6.0 V
Output Short-Circuit Duration (Any Pin to Common)	Indefininte
Operating Temperature Range	−55°C to +125°C
Storage Temperature	−65°C to +150°C

Stresses above those listed under the Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only; functional operation of the device at these or any other conditions above those indicated in the operational section of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Applications requiring more than 200 cycles to MIL-STD-883 Method 1010 Condition B (−55°C to +125°C) require underfill or other means to achieve this requirement.

Drops onto hard surfaces can cause shocks of greater than 2000 *g* and exceed the absolute maximum rating of the device. Care should be exercised in handling to avoid damage.

RATE SENSITIVE AXIS

This is a Z-axis rate-sensing device that is also called a yaw rate sensing device. It produces a positive going output voltage for clockwise rotation about the axis normal to the package top, i.e., clockwise when looking down at the package lid.

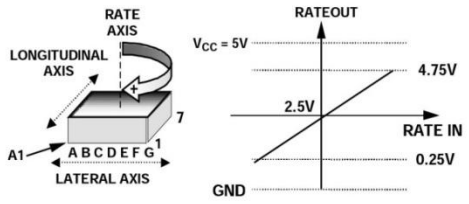


Figure 2. RATEOUT Signal Increases with Clockwise Rotation

ESD CAUTION

ESD (electrostatic discharge) sensitive device. Electrostatic charges as high as 4000 V readily accumulate on the human body and test equipment and can discharge without detection. Although this product features proprietary ESD protection circuitry, permanent damage may occur on devices subjected to high energy electrostatic discharges. Therefore, proper ESD precautions are recommended to avoid performance degradation or loss of functionality.

PIN CONFIGURATION AND FUNCTION DESCRIPTIONS

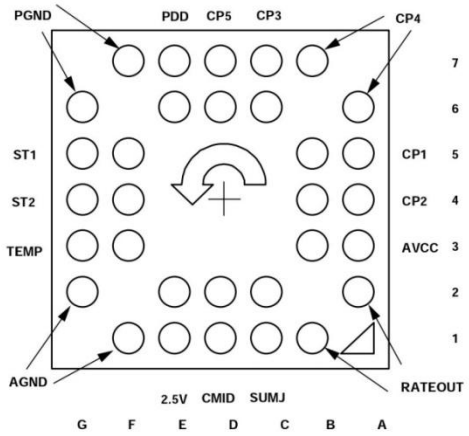


Figure 3. 32-Lead BGA (Bottom View)

Table 3. Pin Function Descriptions

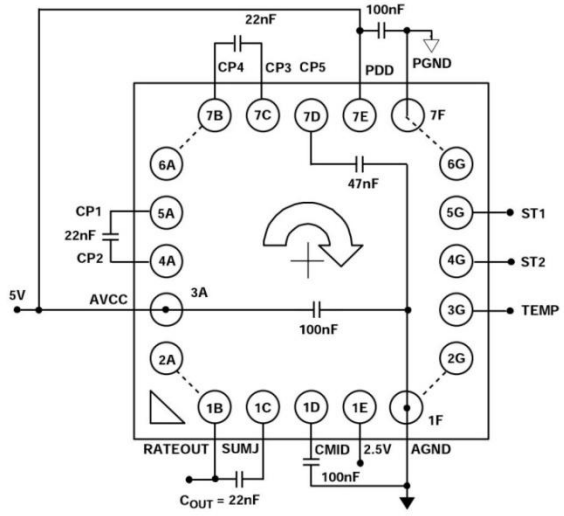
Pin No.	Mnemonic	Description
6D, 7D	CP5	HV Filter Capacitor—47 nF
6A, 7B	CP4	Charge Pump Capacitor—22 nF
6C, 7C	CP3	Charge Pump Capacitor—22 nF
5A, 5B	CP1	Charge Pump Capacitor—22 nF
4A, 4B	CP2	Charge Pump Capacitor—22 nF
3A, 3B	AVCC	+ Analog Supply
1B, 2A	RATEOUT	Rate Signal Output
1C, 2C	SUMJ	Output Amp Summing Junction
1D, 2D	CMID	HF Filter Capacitor—100 nF
1E, 2E	2.5V	2.5 V Precision Reference
1F, 2G	AGND	Analog Supply Return
3F, 3G	TEMP	Temperature Voltage Output
4F, 4G	ST2	Self-Test for Sensor 2
5F, 5G	ST1	Self-Test for Sensor 1
6G, 7F	PGND	Charge Pump Supply Return
6E, 7E	PDD	+ Charge Pump Supply

THEORY OF OPERATION

The ADXRS300 operates on the principle of a resonator gyro. Two polysilicon sensing structures each contain a dither frame, which is electrostatically driven to resonance. This produces the necessary velocity element to produce a Coriolis force during angular rate. At two of the outer extremes of each frame, orthogonal to the dither motion, are movable fingers that are placed between fixed pickoff fingers to form a capacitive pickoff structure that senses Coriolis motion. The resulting signal is fed to a series of gain and demodulation stages that produce the electrical rate signal output. The dual-sensor design rejects external *g*-forces and vibration. Fabricating the sensor with the signal conditioning electronics preserves signal integrity in noisy environments.

The electrostatic resonator requires 14 V to 16 V for operation. Since only 5 V is typically available in most applications, a charge pump is included on-chip. If an external 14 V to 16 V supply is available, the two capacitors on CP1–CP4 can be omitted and this supply can be connected to CP5 (Pin 7D) with a 100 nF decoupling capacitor in place of the 47 nF.

After the demodulation stage, there is a single-pole low-pass filter consisting of an internal 7 kΩ resistor (R_{SEN1}) and an external user-supplied capacitor (CMID). A CMID capacitor of 100 nF sets a 400 Hz ±3 5% low-pass pole and is used to limit high frequency artifacts before final amplification. The bandwidth limit capacitor, C_{OUT}, sets the pass bandwidth (see Figure 5 and the Setting Bandwidth section).



NOTE THAT INNER ROWS/COLUMNS OF PINS HAVE BEEN OMITTED FOR CLARITY BUT SHOULD BE CONNECTED IN THE APPLICATION.

Figure 4. Example Application Circuit (Top View)

SUPPLY AND COMMON CONSIDERATIONS

Only power supplies used for supplying analog circuits are recommended for powering the ADXRS300. High frequency noise and transients associated with digital circuit supplies may have adverse effects on device operation.

Figure 4 shows the recommended connections for the ADXRS300 where both AVCC and PDD have a separate decoupling capacitor. These should be placed as close to the their respective pins as possible before routing to the system analog supply. This minimizes the noise injected by the charge pump that uses the PDD supply.

It is also recommended to place the charge pump capacitors connected to the CP1–CP4 pins as close to the part as possible. These capacitors are used to produce the on-chip high voltage supply switched at the dither frequency at approximately 14 kHz. Care should be taken to ensure that there is no more than 50 pF of stray capacitance between CP1–CP4 and ground. Surface-mount chip capacitors are suitable as long as they are rated for over 15 V.

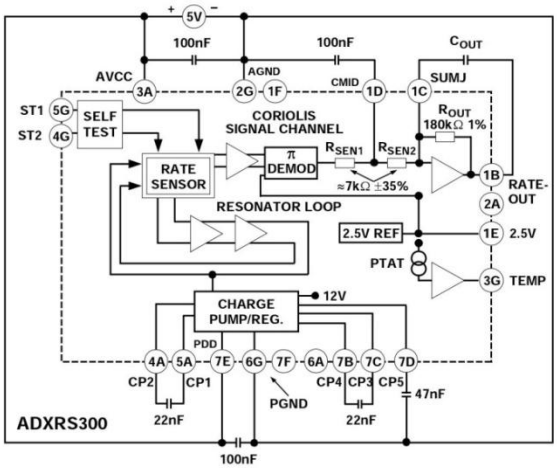


Figure 5. Block Diagram with External Components

SETTING BANDWIDTH

External capacitors CMID and C_{OUT} are used in combination with on-chip resistors to create two low-pass filters to limit the bandwidth of the ADXRS300's rate response. The −3 dB frequency set by R_{OUT} and C_{OUT} is

$$= 1 \left(2 \times \pi \times \quad \times \quad\right)$$

and can be well controlled since R_{OUT} has been trimmed during manufacturing to be 180 kΩ ± 1%. Any external resistor applied between the RATEOUT (1B, 2A) and SUMJ (1C, 2C) pins results in

$$= \left(180\ \text{k}\Omega \times \quad\right) \left(180\ \text{k}\Omega \times \quad\right)$$

The −3 dB frequency is set by RSEN (the parallel combination of R_{SEN1} and R_{SEN2}) at about 3.5 kΩ nominal; CMID is less well controlled since R_{SEN1} and R_{SEN2} have been used to trim the rate sensitivity during manufacturing and have a ±35% tolerance. Its primary purpose is to limit the high frequency demodulation artifacts from saturating the final amplifier stage. Thus, this pole of nominally 400 Hz @ 0.1 μF need not be precise. Lower frequency is preferable, but its variability usually requires it to be about 10 times greater (in order to preserve phase integrity) than the well-controlled output pole. In general, both −3 dB filter frequencies should be set as low as possible to reduce the amplitude of these high frequency artifacts and to reduce the overall system noise.

INCREASING MEASUREMENT RANGE

The full-scale measurement range of the ADXRS300 can be increased by placing an external resistor between the RATEOUT (1B, 2A) and SUMJ (1C, 2C) pins, which would parallel the internal R_{OUT} resistor that is factory-trimmed to 180 kΩ. For example, a 330 kΩ external resistor will give ~50% increase in the full-scale range. This is effective for up to a 4× increase in the full-scale range (minimum value of the parallel resistor allowed is 45 kΩ). Beyond this amount of external sensitivity reduction, the internal circuitry headroom requirements prevent further increase in the linear full-scale output range. The drawbacks of modifying the full-scale range are the additional output null drift (as much as 2°/sec over temperature) and the readjustment of the initial null bias (see the Null Adjust section).

USING THE ADXRS300 WITH A SUPPLY-RATIOMETRIC ADC

The ADXRS300's RATEOUT signal is nonratiometric, i.e., neither the null voltage nor the rate sensitivity is proportional to the supply. Rather they are nominally constant for dc supply changes within the 4.75 V to 5.25 V operating range. If the ADXRS300 is used with a supply-ratiometric ADC, the ADXRS300's 2.5 V output can be converted and used to make corrections in software for the supply variations.

NULL ADJUST

Null adjustment is possible by injecting a suitable current to SUMJ (1C, 2C). Adding a suitable resistor to either ground or to the positive supply is a simple way of achieving this. The nominal 2.5 V null is for a symmetrical swing range at RATEOUT (1B, 2A). However, a nonsymmetric output swing may be suitable in some applications. Note that if a resistor is connected to the positive supply, then supply disturbances may reflect some null instabilities. Digital supply noise should be avoided, particularly in this case (see the Supply and Common Considerations section).

The resistor value to use is approximately

$$\square_{\square} = (2\,5 \times 180\,000) \square (\square_{0} \quad \square\square_{1} \quad)$$

V_{NULL0} is the unadjusted zero rate output, and V_{NULL1} is the target null value. If the initial value is below the desired value, the resistor should terminate on common or ground. If it is above the desired value, the resistor should terminate on the 5 V supply. Values are typically in the 1 MΩ to 5 MΩ range.

If an external resistor is used across RATEOUT and SUMJ, then the parallel equivalent value is substituted into the preceding equation. Note that the resistor value is an estimate since it assumes V_{CC} = 5.0 V and V_{SUMJ} = 2.5 V.

SELF-TEST FUNCTION

The ADXRS300 includes a self-test feature that actuates each of the sensing structures and associated electronics in the same manner as if subjected to angular rate. It is activated by standard logic high levels applied to inputs ST1 (5F, 5G), ST2 (4F, 4G), or both. ST1 causes a voltage at RATEOUT equivalent to typically −270 mV, and ST2 causes an opposite +270 mV change. The self-test response follows the viscosity temperature dependence of the package atmosphere, approximately 0.25%/°C.

Activating both ST1 and ST2 simultaneously is not damaging. Since ST1 and ST2 are not necessarily closely matched, actuating both simultaneously may result in an apparent null bias shift.

CONTINUOUS SELF-TEST

The one-chip integration of the ADXRS300 gives it higher reliability than is obtainable with any other high volume manufacturing method. Also, it is manufactured under a mature BIMOS process that has field-proven reliability. As an additional failure detection measure, power-on self-test can be performed. However, some applications may warrant continuous self-test while sensing rate. Application notes outlining continuous self-test techniques are also available on the Analog Devices website.

OUTLINE DIMENSIONS

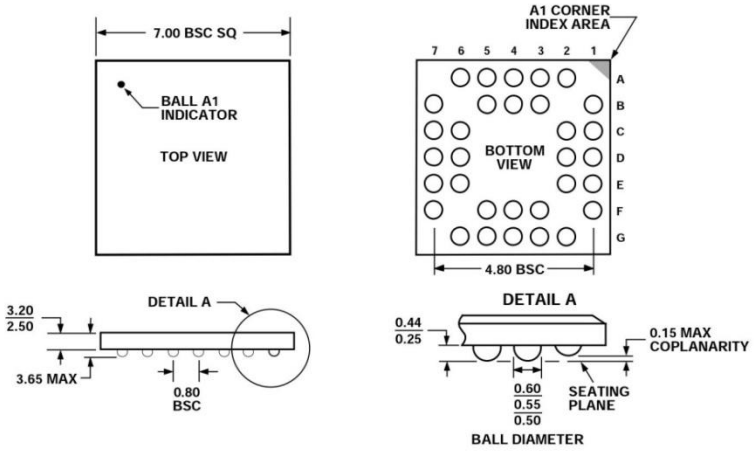


Figure 6. 32-Lead Chip Scale Ball Grid Array [CSPBGA]
(BC-32)
Dimensions shown in millimeters

ORDERING GUIDE

Model	Temperature Range	Package Description	Package Outline
ADXRS300ABG	–40°C to +85°C	32-Lead BGA	BC-32
ADXRS300ABG-Reel	–40°C to +85°C	32-Lead BGA	BC-32
ADXRS300EB		Evaluation Board	