UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

SOFTWARE DEL MODULO C RIO PARA LA LECTURA DE CUERDAS VIBRANTES

TESIS

Que para obtener el título de

Ingeniero Mecatrónico

P R E S E N T A

Jiménez Sandoval Kevin Ernesto

DIRECTOR DE TESIS

Ing. Enrique Ramón Gómez

Ciudad Universitaria, Cd. Mx., 2021

Índice general

Capítulo 1. Planteamiento del problema

1.1 Introducción

La república mexicana está situada en una de las regiones sísmicas más activas del mundo conocida como "Cinturón Circupacífico"[1], por lo cual existe una necesidad en aumento sobre instrumentos de monitoreo estructural que permitan conocer la salud de edificios o estructuras de manera precisa, eficiente y rápida.

A partir del sismo vivido el 19 de septiembre del 2017, el Instituto de Ingeniería de la UNAM (IIUNAM) propuso la integración de sistemas físicos y dinámicos para el monitoreo estructural al Instituto para la Seguridad de las Construcciones (ISC), por medio de un sistema que permite tomar datos tanto dinámicos como estáticos de diversos sensores empleados para el monitoreo estructural como son los sensores de voltaje y los lectores de cuerdas vibrantes.

En el presente trabajo se describe el desarrollo de un software, creado por el IIUNAM en su coordinación de electrónica, cuyo fin es obtener en forma simultánea y a través del uso de diversos sensores distintos parámetros estructurales que sean útiles para diagnosticar el estado que guarda una estructura, en determinado momento de su vida útil.

El sistema integra un módulo C creado por el IIUNAM que permite utilizar hasta 256 cuerdas vibrantes multiplexadas permitiendo la medición de las variables de temperatura y frecuencia de cada sensor, usando las características y las ventajas que ofrece un CompactRIO (cRIO) debido a su FPGA.

[1] Sismología de México, SGM, En línea, Fecha de consulta: 10/05/2020, Disponible en: https://www.sgm.gob.mx/Web/MuseoVirtual/Riesgos-geologicos/Introduccion-riesgos.html

1.2 Planteamiento del problema

En los últimos años en México han ocurrido sismos y temblores que han afectado la seguridad de diversas obras de infraestructura, por lo cual se le solicito a la coordinación de electrónica del IIUNAM la creación de un sistema de medición estructural que permitiera tomar medidas de manera precisa, reducir tiempos de medición y la cantidad de equipos empleados para ello, así como el espacio empleado por éstos, para la toma de las lecturas em determinadas obras civiles.

Como parte del equipo de desarrollo de dicho sistema se me encargo la creación del software necesario para que un módulo C (totalmente creado por el IIUNAM), funcionara en un cRIO de National Instruments (NI), además de generar un programa que permita analizar y procesar todas las señales obtenidas de los sensores conectados al módulo C.

1.3 Justificación

Actualmente existen muchos equipos o sistemas de medición para el monitoreo de estructuras, pero su costo suele ser muy alto o poseen limitantes en cuanto a las mediciones, sensores o características estructurales que analizan, por lo cual es muy común el hecho de comprar un equipo específico para una situación en particular. Por lo anterior, se creó un sistema que se enfocara en las necesidades actuales de medición estructural en el país, que disminuyera el numero de los equipos de medición necesarios y que entrega resultados técnicos de una manera clara y comprensible sin importar la cantidad y la variedad de sensores empleados en el análisis.

1.4 Alcance

Presentar los procesos de comunicación entre los diversos dispositivos que conforman el sistema de procesamiento y almacenamiento de los datos provenientes de los sensores de cuerda vibrante, así como los programas que permiten dicha interacción entre ellos. Enfocándome en el software para la comunicación módulo C – CompactRIO y CompactRIO – PC.

Crear una interfaz humano - maquina (HMI) en LabVIEW que permita analizar, procesar y entender en lenguaje técnico los datos obtenidos de los sensores conectados al módulo C.

1.5 Metodología

Para el desarrollo de este trabajo fue necesario emplear una metodología diseñada por NI que permitiera trabajar en LabVIEW y comunicarse con un módulo no creado por ellos. Esta metodología prácticamente consiste en 3 etapas, la primera, que ellos no mencionan o hablan de manera explícita, pero que denomine como identificación, es la que permite al sistema de LabVIEW reconocer un circuito electrónico conectado al compactRIO como un módulo C, está integrada por dos programas en formato XML y la programación de la memoria EEPROM del módulo C.

La segunda etapa, es llamada de desarrollo, y en esta la intención es crear todos los programas en LabVIEW necesarios para que el módulo funcione con su infraestructura sin que el usuario final interactúe con ella, como cuando se instalan drivers o librerías para reconocer un dispositivo en la PC. Esta etapa integra variables internas de comunicación, programas de manejo de datos, programas para la identificación de errores, programas para la calibración del módulo, programas para el empleo de diversos métodos de uso del módulo, y un programa general que integra a los anteriores y controla al módulo desde el FPGA.

La última la llaman de lanzamiento, y no es más que el instalador del módulo C y el programa final con el que el usuario interactúa con el módulo para obtener los datos provenientes de él, en este caso, las señales enviadas por los sensores conectados al mismo.

Estas etapas son consecutivas y sin la anterior no es posible el continuar con la siguiente, ya que cada etapa sustenta a la siguiente y permite o le da información necesaria para su desarrollo. Los programas necesarios en estas etapas se desarrollarán y explicarán durante todo el trabajo, aunque cabe mencionar que posterior a este trabajo se necesitó plantear una metodología para permitir la comunicación del módulo con el compactRIO que se desarrolló basándose en los modos de operación de los módulos C (explicados más adelante) y em el empleo del protocolo de comunicación SPI.

Capítulo 2. Objetivos e Hipótesis

2.1 Objetivo general

Desarrollar el software para un sistema basado en un módulo C (Nombrado como IIUNAM-9901) para cRIO, capaz de procesar y almacenar datos de sensores del tipo cuerda vibrante.

2.2 Objetivos específicos

- Realizar un programa capaz de procesar los datos obtenidos del módulo C utilizando LabVIEW.
- Construir uno o más programas que permitan comunicar el módulo C con el backplate de los cRio de NI.
- Crear la estructura de comunicación y el software de instalación del módulo C para que pueda funcionar en diversos equipos de cómputo.
- Crear una interfaz humano-máquina (HMI) en LabVIEW para poder controlar la toma de datos, el procesamiento de los mismos, su almacenamiento y determinar el estado del módulo C.

2.3 Hipótesis

Es posible hacer más eficiente, rápido y exacto la obtención de datos y procesamiento de los mismos, para analizar un sistema estructural a través de la integración de una PC, un cRIO y un módulo C, que adquiere los datos de diversos sensores electrónicos, especialmente los sensores de cuerdas vibrantes, usando las ventajas que ofrece un cRIO debido a su FPGA, siendo la característica más importante la capacidad de configurar y reprogramar su funcionamiento de la manera más eficiente y concreta posible, permitiendo adaptar de manera general el sistema a las características y condiciones que se requieran.

Capítulo 3. Sistema para el control de un módulo C

El sistema de control de un módulo C diseñado en el IIUNAM, en su coordinación de electrónica, está conformado por los siguientes componentes o sistemas:

- **Sensores para el análisis:** Fibras ópticas, sensores de voltaje, cuerdas vibrantes, entre otros.
- **Módulo C**
- **CompactRIO**
- **Equipo de computo**

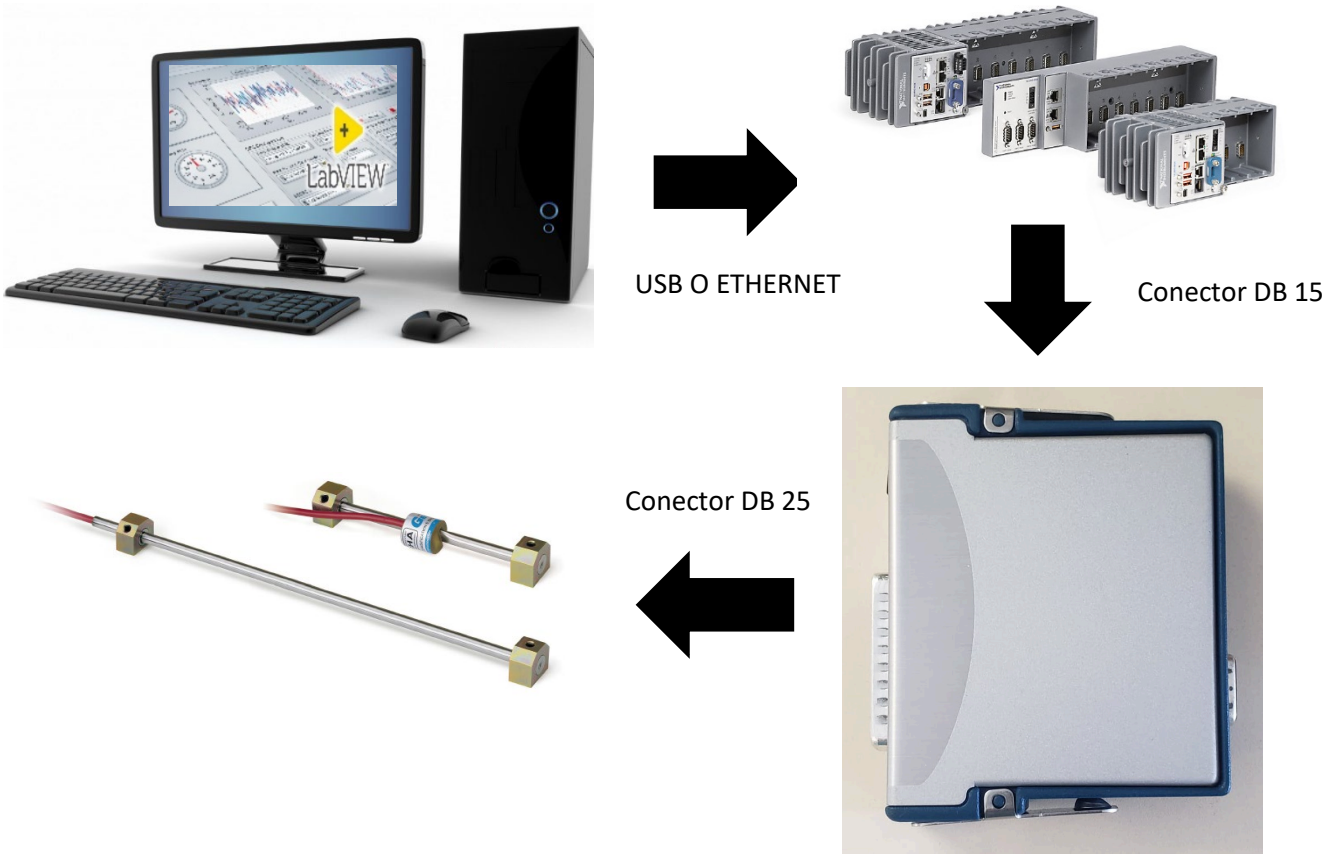


Figura 1. Diagrama del sistema de adquisición de datos

El sistema está diseñado para que al módulo C, creado desde cero por el IIUNAM, se le puedan conectar diversos sensores (hasta 256 cuerdas vibrantes multiplexadas) almacenando y realizando un pre-procesamiento de los datos obtenidos de dichos sensores en el CompactRIO (cRIO), contando con un pos-procesamiento de los datos al conectar el cRIO a una computadora que tenga instalado el programa en LabVIEW creado para obtener, procesar y analizar las variables técnicas que provienen de dicho modulo.

En este capítulo se dará una breve explicación de los componentes ya mencionados del sistema, como interactúan entre ellos y las ventajas que tienen emplear dichos componentes para ser parte de un sistema más completo que analiza el estado de una obra estructural.

3.1 Sensor de cuerda vibrante

El sistema está desarrollado especialmente para realizar lecturas de los sensores de cuerda vibrante, dichos sensores emplean el principio de resonancia para obtener la frecuencia de oscilación de una cuerda. De tal manera que en una cámara cerrada se tiene una cuerda de acero tensada y sujeta en uno de sus extremos a una membrana que percibe el cambio en la presión externa que influye en la frecuencia de vibración de la cuerda de forma proporcional a la presión que se desea medir.

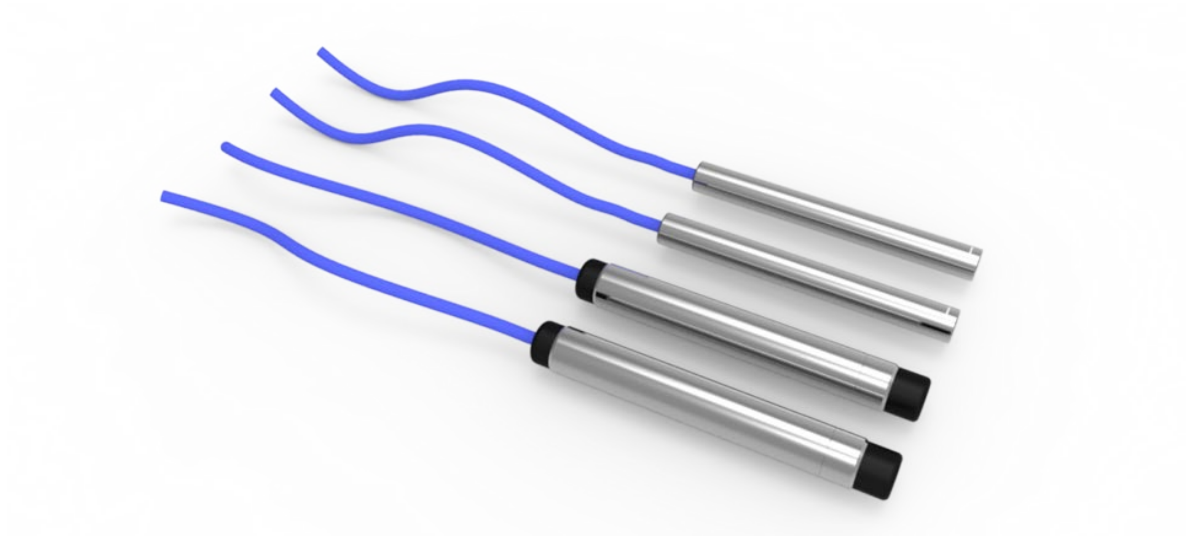


Figura 2. *Sensores de cuerda vibrante*

(SISGEO, 2020)

La presión que se ejerce en el sensor provoca que la membrana se desvié de su estado anterior, reduciendo la tensión en el cable y cambiando la frecuencia vibratoria medida por la bobina electromagnética, la señal leída se trasmite por un cable a un aparato para analizarla.

Las bobinas dentro del sensor excitan la cuerda mediante impulsos magnéticos y captan la frecuencia de vibración de la misma. El sensor posee dos bovinas electromagnética debido a que una bobina simple es muy perceptible a las interferencias magnéticas (ruido ambiental), por lo que al conectar las bobinas en fuera de fase (diferencia de fase de 180°) la señal de ruido se elimina drásticamente y las señales obtenidas por cada bovina se suma.

Su principio de funcionamiento radica en que la frecuencia a la que oscila la cuerda [F] es inversamente proporcional a la distancia entre sus extremos [L] que a su vez es función de la presión que se ejerce en la membrana del sensor.

Estos sensores son muy precisos y fiables, según los proveedores de estos sensores llegan a tener un grado de exactitud de un 0.2%, permiten trasmitir su señal a distancias muy largas sin perder precisión, posen un corto tiempo de respuesta, son relativamente nuevos, además de tener la versatilidad de emplearse en variados tipos de terrenos o zonas.

Esos sensores son conectados al módulo C por medio de un conector de 25 pines integrado en el mismo. A través de un multiplexaje externo al módulo C, es posible conectar hasta 256 sensores de cuerda vibrante sin termistor o 128 con termistor. Siendo empleada la lectura de la temperatura para mejorar la exactitud de los datos obtenidos por los sensores de cuerda vibrante.

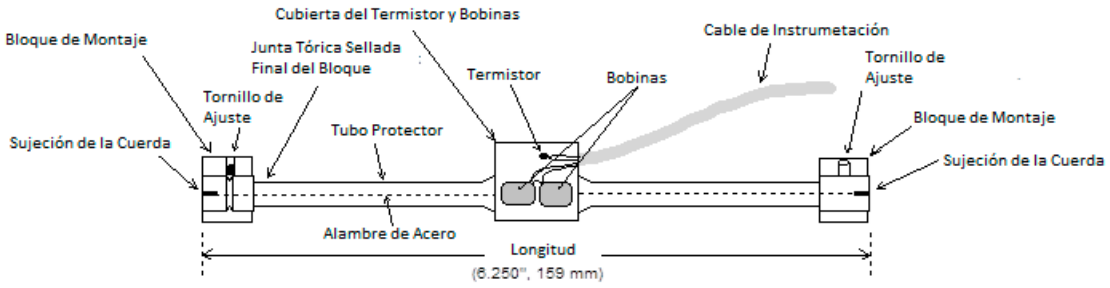


Figura 3. *Composición de un transductor de cuerda vibrante (Modelo 4000 de la marca GEOKON)* (Figueroa, 2016)

3.2 Módulo C

Un módulo C es una serie de componentes electrónicos que son empleados para interactuar con una variedad de conmutadores, transductores, sensores, dispositivos industriales, interfaces de comunicación y hasta controladores inteligentes, ofreciendo entradas y/o salidas digitales para el control y monitoreo de dichos sistemas electrónicos. Siendo estos módulos compactos y muy rápidos, al interactuar con el FPGA (Procesador Reprogramable de Silicio) que poseen los cRIO.

s

Figura 4. *Ejemplo de un módulo C. Módulo de interface EtherCAT.*

(Instruments, 2020)

Un módulo C, como ya se mencionó, integra ciertos componentes electrónicos que cada fabricante determina en una placa base (PCB) que debe cumplir con una serie de características o requisitos que marca NI para poder conectarse y comunicarse con el cRIO y tener un cierto grado de calidad y certificación que ellos avalan. Algunas de las características mencionadas que debe tener este PCB pueden verse en el anexo A2.

3.2.1 Electrónica dentro del módulo C

Un módulo C típico tiene la siguiente estructura:

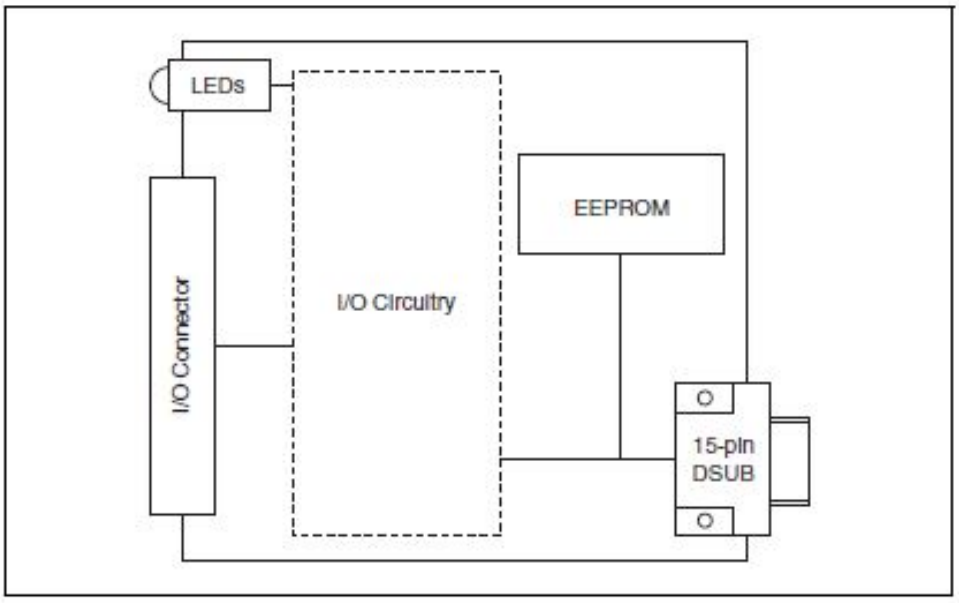


Figura 5. *Diagrama de bloques de un módulo C típico*

(National Instruments Corporation, 2011)

Los componentes que lo integran son:

- El *conector DSUB de 15 pines:* es el que permite la comunicación con el cRIO a través del protocolo de comunicación SPI.
- La *memoria EEPROM:* es la que permite almacenar la información de identificación y calibración del módulo para que pueda ser reconocido por el cRIO y LabVIEW.
- El apartado de *I/O Conector:* permite la función de comunicar el módulo con otros sistemas externos a través de entradas y/o salidas eléctricas.
- Los *LEDs:* se utilizan como medio de trasmisión visual del estado del módulo y de su comunicación con los sistemas externos.

- Finalmente, el *circuito interno*: es el conjunto de componentes electrónicos que permite realizar la tarea para la cual fue diseñado el módulo (manipulación, conversión o recepción de señales, sistemas de control, sistemas de monitoreo inteligentes, entre muchas otras cosas).

De manera particular para este trabajo el circuito interno del módulo C está integrado por resistencias, diodos, capacitores, interruptores, amplificadores operacionales, un microcontrolador PIC32MX, un transistor MOSFET, un controlador de líneas, aisladores digitales, comparadores de voltaje, multiplexores analógicos, un regulador de voltaje, un oscilador ASTX-H11-16 con una frecuencia de 16MHz, un ADS1118 siendo este un convertidor analógico digital que integra un amplificador de ganancia programable (PGA), referencias de voltaje y un sensor de temperatura de alta precisión.

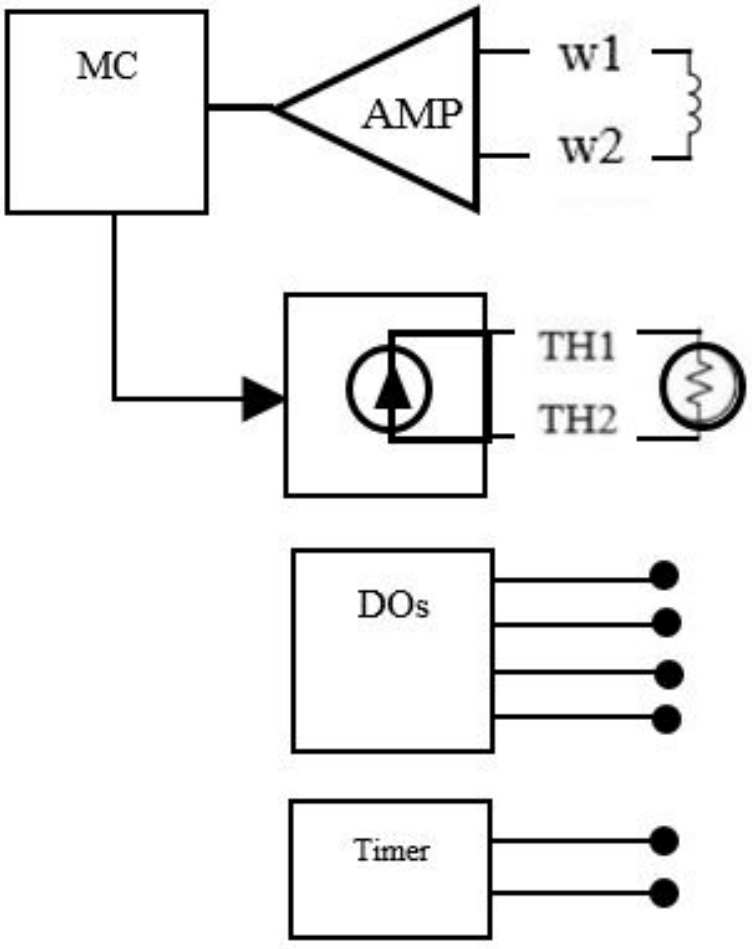


Figura 6. *Diagrama de bloques del módulo C.*

Todos estos componentes permiten que las señales de los sensores conectados al módulo C a través del conector de 25 pines, se organicen, procesen y trasmitan la información en variables técnicas al cRIO.

Interfaz de comunicación del módulo C con el CompactRIO

Para realizar la comunicación modulo C – cRIO se debe respetar la siguiente interfaz de señales/conexiones para el DSUB 15 del módulo:

	Modo de funcionamiento del módulo				
Pin	**Sleep**	**Idle**	**ID**	**Auxiliary Communication**	**Normal Operation**
14	GND				
4	V_{CC}				
8	SLEEP				
1	—	~ID_SELECT			
11	—	—	SPI_CLK	SPI_CLK	SPI_CLK
6	—	—	MOSI	MOSI/DIO7	MOSI/DIO7
12	—	—	MISO	MISO/DIO6	MISO/DIO6
2	—	—	~SPI_CS	~SPI_CS/DIO5	~SPI_CS/DIO5
7	—	—	SPI_FUNC=1*	SPI_FUNC=0*	SPI_FUNC/DIO4
9	—	—	—	DIO3	~CONVERT/DIO3
15	—	—	—	~DONE/DIO2	~DONE/DIO2
5	—	—	—	DIO1	TRIG_OUT/DIO1
10	—	—	—	DIO0	DIO0
3	Reserved				
13	Reserved				
* Impulsado por el operador para distinguir entre los modos de ID y Auxiliar de comunicación.					

Tabla 1. *Tabla de funcionamiento de las señales del módulo C*

(National Instruments Corporation, 2011)

Siendo las señales presentes en cualquier modulo las siguientes:

- **GND** - Referencia de tierra para todas las señales y potencia (VCC).

- **V_{CC}** - 5 V ± 5% en condiciones normales de funcionamiento, con un consumo máximo de corriente de 200 mA.

- **SLEEP** - Señal alta activa impulsada por el operador que pone un módulo en modo de Suspensión. Cuando "SLEEP" se conduce alto, el módulo entra en el modo de Suspensión y ya no puede comunicarse con el operador. Cuando "SLEEP" se baja, el módulo entra en uno de los otros modos definidos por las otras líneas de señal.

- **~ID_SELECT** - Esta señal proporciona dos funciones: detección de módulo y selección de modo. Las entradas tienen una resistencia débil de bajada en esta línea, mientras que los módulos tienen una fuerte resistencia de pull-up. Los operadores pueden monitorear esta línea para determinar si un módulo está presente (un valor alto indica que un módulo está presente y listo para la comunicación normal) o si se ha eliminado uno. Los operadores también pueden usar esta línea para seleccionar modos de operación. El usuario impulsa "ID_SELECT" bajo en modo ID y modo de comunicación auxiliar. Tenga en cuenta que, en estos modos, el operador no puede detectar si el módulo se elimina (ya que está activa la línea). El operador entra en el modo Inactivo o en el modo de Operación normal haciendo triestado "ID_SELECT" y dejando que las resistencias pull-up y pull-down surtan efecto.

- **RESERVED** - Las líneas reservadas están, como su nombre indica, reservadas para futuras funcionalidades y deben dejarse desconectadas tanto en operadores como en módulos.

- **~CONVERT** - Esta señal proporciona dos funciones: iniciar conversiones en módulos temporizados externamente y restablecer la fase de una adquisición en módulos autocontrolados. Para los módulos temporizados externamente, el transportista maneja el "CONVERT" la señal baja para iniciar una conversión e

inactiva la señal alta. El borde descendente de esta señal es el punto de referencia de temporización para la adquisición. Para los módulos auto-temporizados, el operador impulsa la señal "CONVERT" baja para restablecer la fase de la adquisición y puede sincronizar los módulos accionando "CONVERT" módulos de bajo a múltiple simultáneamente.

- **~DONE** - Esta señal proporciona dos funciones opcionales: indicar el progreso de una adquisición y la comunicación de interfaz periférica serial (SPI) de estrangulamiento genéricamente. Un nivel alto en esta línea indica que un módulo actualmente no puede aceptar la comunicación, mientras que un nivel bajo indica que la comunicación puede comenzar o continuar. Cuando se usa para indicar el progreso de una adquisición, un módulo conduce "DONE" alto en respuesta a un flanco descendente en "CONVERT". Una vez que el módulo está listo para la comunicación, conduce "DONE" bajo, y la transferencia de datos puede comenzar utilizando SPI. "DONE" también se puede usar para regular las transferencias de SPI en los límites de bytes. El módulo debe conducir "DONE" a un estado válido en los límites de bytes. Si "DONE" está bajo, la comunicación puede continuar; si es alto, la comunicación debe detenerse hasta que "DONE" vuelva a ser bajo.

- **TRIG_OUT** - Esta señal es generada por el módulo para actuar como un disparador para el resto del sistema. Los ejemplos podrían ser un canal de entrada digital o un comparador en un canal de entrada analógica. "TRIG_OUT" tiene los mismos requisitos que el DIO de propósito general que se usa como disparadores.

Para la trasferencia de datos por SPI:

- **MOSI** — Línea de datos SPI Master-Out, Slave-In.

- **MISO** — Línea de datos SPI Master-In, Slave-Out.

- **~SPI_CS** — Esta señal es impulsada por el operador para enmarcar transacciones SPI con el módulo. Un módulo debe ignorar "SPI_CLK" y "MOSI" cuando

"SPI_CS" se mantiene en alto, pero responderá a estas señales cuando "SPI_CS" esté bajo.

- **SPI_CLK** — Un reloj alto inactivo impulsado por el operador para controlar y temporizar las transferencias de datos SPI con un módulo. Esta es una señal clave para ciertos modos y debe controlarse cuidadosamente durante las transiciones de modo para evitar accesos inadvertidos de SPI.

- **SPI_FUNC** — Esta señal proporciona dos funciones: seleccionar entre el modo ID y el modo de comunicación auxiliar, y seleccionar entre dos puertos SPI en el modo de operación normal. Cuando "ID_SELECT" es bajo y "SPI_FUNC" es alto, se selecciona el modo ID. Cuando "ID_SELECT" es bajo y "SPI_FUNC" está bajo, se selecciona el modo de comunicación auxiliar. En el modo de funcionamiento normal, "SPI_FUNC" se puede utilizar para seleccionar entre dos puertos arbitrarios SPI, como un puerto de datos (SPI_FUNC = 0) y un puerto de configuración (SPI_FUNC = 1).

Señales de Entrada/Salida digitales genéricas:

- **DIO [0:7]** - Estas líneas pueden estar expuestas al usuario desarrollador o usarse para el funcionamiento interno del módulo. Los módulos pueden suponer que todos los operadores ofrecen soporte DIO estático (con cambios de señal programados por software).

Es importante mencionar que dichas señales básicas del módulo deben tener una serie de requisitos eléctricos únicos para cada señal, dichos requisitos pueden consultarse en el manual cRIO_MDK_Hardware_User_Manual del kit de desarrollo de módulos C de NI.

3.2.2 Software dentro del módulo C

En cuanto al software del módulo de manera particular se crearon muchos programas diferentes para que el módulo funcionara de la manera más óptima posible, comenzando con la creación de dos archivos XML, el primero con nombre “ModuleType” que permite, junto con la información almacenada en la memoria EEPROM, identificar al módulo y tener la información para la calibración del mismo; el segundo llamado “ModuleSupport” que permite configurar los modos y formas de comunicación del módulo con el cRIO.

Después se creó un programa en LabVIEW para poder grabar los datos de identificación y calibración del módulo en la memoria EEPROM al conectar el módulo al compact, sin la necesidad de usar equipos especializados y generalmente costosos para grabar memorias, antes de montarse en el módulo.

Figura 7. *Programa en LabVIEW para la escritura y lectura de datos de una memoria EEPROM conectada a un Módulo C.*

Se requirió también crear un programa en el microcontrolador (dentro del módulo C) que permitiera controlar todos los componentes eléctricos y que a la vez procesara las señales provenientes del exterior, para convertirlas en información digital que contuviera las variables provenientes de cada sensor (Eje. Voltaje, frecuencia, temperatura, etc.).

Finalmente se crearon dos programas en LabVIEW que interactúan entre ellos para lograr el procesamiento de información del módulo, el primero para el FPGA del cRIO donde se obtienen las variables generadas en el módulo de todos los sensores tipo cuerda vibrante conectados a él, a través del puerto de comunicación SPI, y el segundo es un programa que puede estar en el cRIO o en una PC que se emplea para visualizar, almacenar y procesar los valores obtenidos del módulo C en unidades ingenieriles de cada sensor empleado.

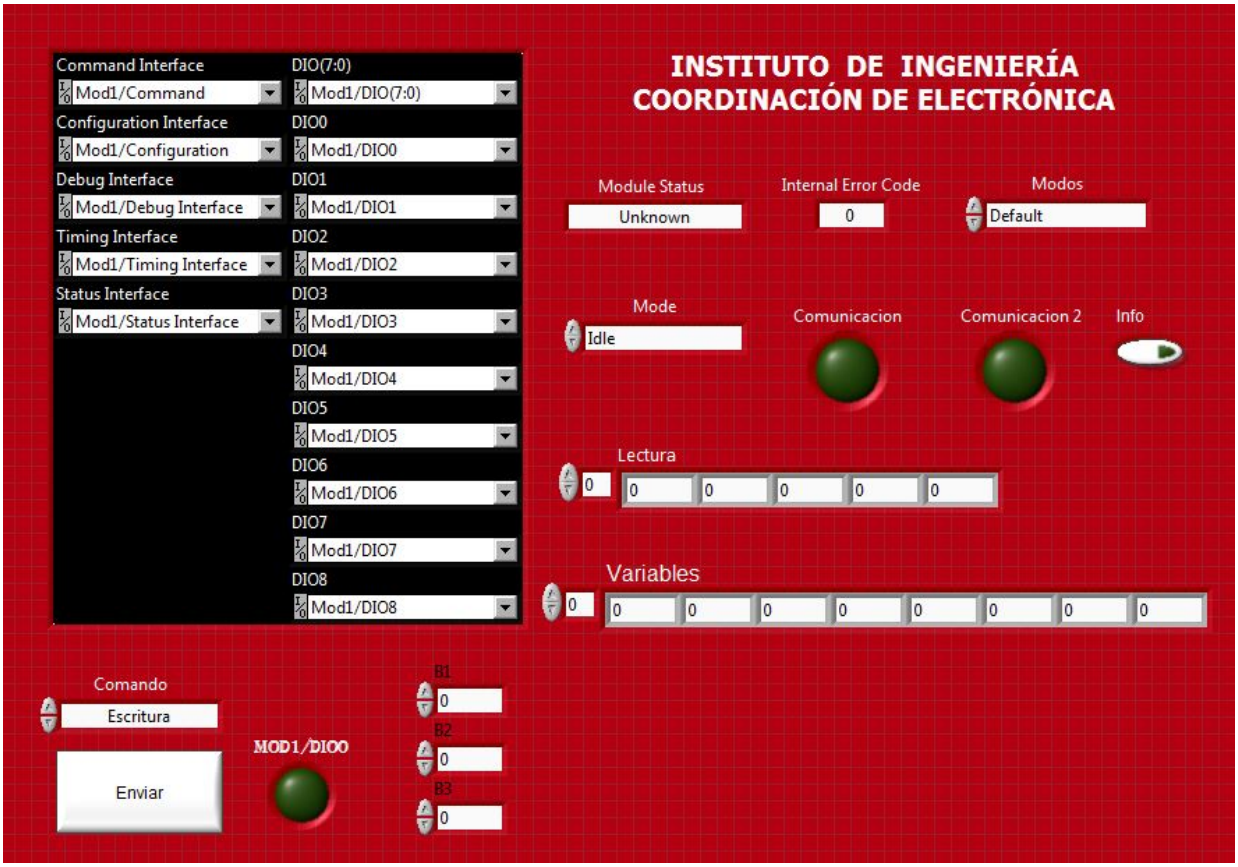


Figura 8. *Programa en LabVIEW para la programación del FPGA del cRIO.*

Figura 9. *Programa en LabVIEW para el procesamiento de variables de sensores de cuerda vibrante.*

El siguiente capítulo aborda de manera más específica cada uno de estos programas, así como explica su funcionamiento, la manera de crearse y los cambios o aspectos que pueden alterarse dada las especificaciones o requerimientos de uso del sistema como tal.

3.3 CompactRIO

Un CompactRIO (cRIO) es un controlador con un procesador y un FPGA programable por el usuario, ofreciendo atreves de sus módulos conectividad directa con sensores y funciones especializadas. Estos cRIO se venden por National Instruments contando con una gran variedad de tamaños que poseen diferentes características y especificaciones para diversas aplicaciones.[2]

Figura 10. *Ejemplos de CompactRIO con diversos módulos C*

(Instruments, 2020)

Las ventajas que tiene trabajar con un cRIO radican en la velocidad en la que se puede trabajar y procesar datos en tiempo real debido a su módulo FPGA que al combinarse con diversos módulos, permite una adaptabilidad, alta velocidad respuesta y compactibilidad para cualquier tipo de trabajo.

[2] CompactRIO, NATIONAL INSTRUMENTS, Disponible en: http://www.ni.com/es-mx/shop/compactrio.html, Fecha de consulta:18/05/2012

Capítulo 4. Configuración del módulo C

Para poder programar un módulo C es necesario entender que NI maneja 5 modos de operación que permiten comunicar el módulo con el cRIO, estos modos permiten la identificación, configuración y control del módulo en sí. Los modos de operación son los siguientes:

- **Sleep (Suspensión) -** El modo de suspensión se utiliza para poner el módulo en estado de espera. Todos los módulos deben mantenerse en este modo durante su encendido.

- **Idle (Inactivo) -** el modo Inactivo se utiliza para detectar la inserción o la presencia de un módulo.

- **ID (Identificación) -** El modo identificación se usa como su nombre lo dice para identificar el módulo y acceder al contenido de la EEPROM de identificación.

- **Auxiliary Communication (Comunicación auxiliar) -** El modo de comunicación auxiliar se utiliza para la comunicación SPI básica al módulo. Las líneas restantes pueden configurarse como E / S digitales sin tiempo para funciones de estado o control. En general, este modo se usa para admitir operaciones que no se ajustan al modo de Operación Normal.

- **Normal Operation (Funcionamiento normal) -** Se accede a la funcionalidad principal del módulo en este modo. Esto incluye señales de temporización, transferencia de datos y E / S digitales, entre otras cosas.

Estos modos siguen una serie de relaciones y transiciones que especifica NI para su funcionamiento valido que se pueden entender con el siguiente diagrama:

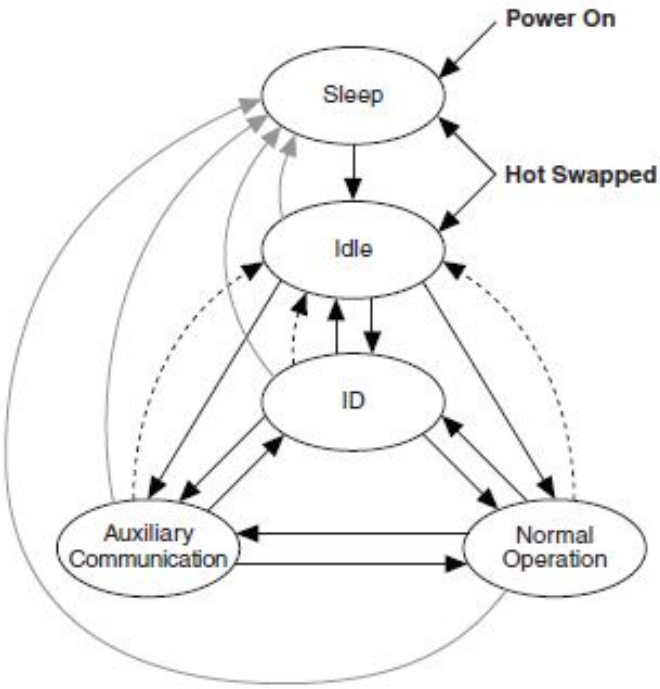


Figura 11. *Transiciones de modo válido*

(National Instruments Corporation, 2011)

Todos los módulos deben contar con el modo de suspender, inactivo e identificación, y dependiendo del funcionamiento del módulo se deben activar los modos de comunicación auxiliar y/o de operación normal.

Para poder cambiar de un modo a otro es necesario definir una serie de señal de control para cada estado de la siguiente manera:

- La señal “SLEEP” solo se utiliza para controlar si el módulo está dentro o fuera del modo de Suspensión.

- La señal “ID_SELECT” sirve para múltiples propósitos. Cuando “ID_SELECT” es conducido bajo por el operador, el módulo se coloca en el modo de comunicación Auxiliar o de Identificación. Cuando “ID_SELECT” está tristado, el módulo está en el modo de funcionamiento Inactivo o Normal. En el modo Inactivo, el operador está monitoreando “ID_SELECT” para determinar si un módulo está presente y listo para ser identificado. En el modo de Operación

Normal, el operador está monitoreando “ID_SELECT” para determinar si el módulo ha sido eliminado.

- La señal “SPI_FUNC” se usa para seleccionar entre el modo de identificación y el modo de comunicación auxiliar.

Visto de otra manera el control de los modos de un módulo se realiza como se muestra en la siguiente tabla:

Señal	**Modos de Operación**				
	Sleep	**Idle**	**ID**	**Auxiliary Communication**	**Normal Operation**
SLEEP	1	0	0	0	0
~ID_SELECT	Tristate	Tristate*	0	0	Tristate†
SPI_FUNC	Tristate	Tristate	1	0	Varies‡

* El operador monitorea ~ ID_SELECCIONAR para determinar si un módulo está presente y listo para ser identificado.

+ Los monitores de la operadora ~ ID_SELECT para determinar si el módulo se ha eliminado.

++ SPI_FUNC no se utiliza para la selección de modo aquí, y puede usarse para otros fines.

Tabla 2. *Señales de control de los modos de operación de un módulo C*

(National Instruments Corporation, 2011)

Las transiciones de modo las lleva a cabo el operador cambiando el estado de las señales de control de un modo a otro. Además, existen requisitos adicionales para las líneas de señal restantes que difieren dependiendo de los modos involucrados en la transición. Estos requisitos ayudan a evitar las condiciones de doble accionamiento entre el módulo y el operador, y proporcionan tiempo de configuración y mantenimiento para evitar problemas técnicos en las E / S del usuario. En general, todas las señales tienen comportamientos bien definidos mientras están en un modo particular, este comportamiento se define y programa en los archivos XML del módulo.

Finalmente, para realizar la transición entre modos se requiere seguir una serie de reglas, condiciones eléctricas y tiempos definidos que permitan su funcionamiento correcto estipuladas por National Instruments, pero se dará por hecho en este trabajo que ya se toman en cuenta a la hora de diseñar su PCB, en cuanto al software en este asunto en LabVIEW ya

se realizan los cambios de modo de manera correcta y automática a través de los bloques de comando que se tienen en las paletas del MDK.

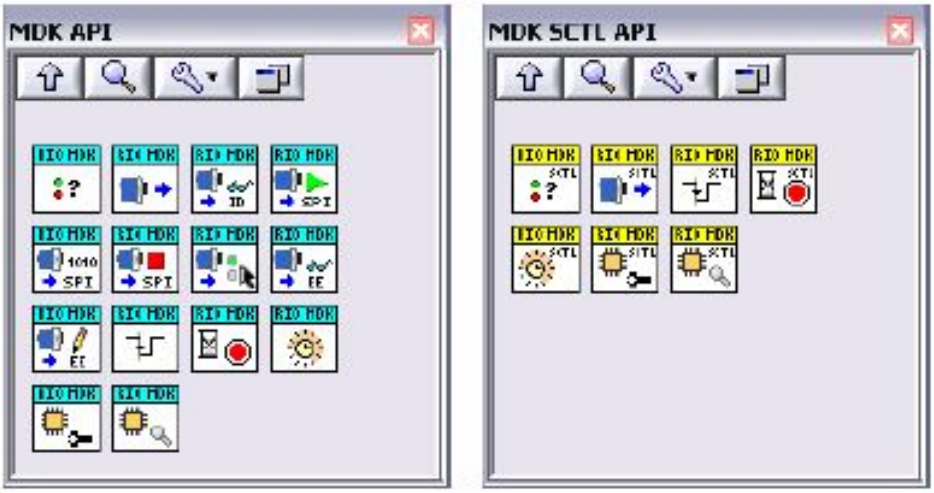


Figura 12. *Paletas del MDK*

(National Instruments Corporation, 2011)

Para la instalación y el manejo del MDK se hablará más adelante en la realización del programa en el FPGA. Por ahora solo es importante mencionar que el MDK (Module Devolopment Kit) es un paquete de desarrollo de módulos que vende NI, que cuenta con un pequeño complemento de software que permite controlar un módulo C de manera práctica y sencilla.

Con esta parte se concluye la información conceptual necesaria para desarrollar el software del módulo C, para realizar todos los procesos y/o programas que presenta este trabajo es necesario tener lo siguiente:

1. El hardware de su módulo C concluido.

2. Un programa que permita crear archivos XML, en este caso se empleara el Bloc de notas de Windows.

3. Cualquier versión de LabVIEW compatible con el MDK. En este caso será la versión de LabVIEW 2019.

4. El propio MDK, empleando la versión MDK 2.1 para este manual.

De preferencia, pero no de manera forzada:

1. Una versión de Microsoft Office para el uso de Excel (versión 2016 en este caso).
2. El Dispositivo llamado Interfaz I2C/SPI vendido por NI para la verificación de la comunicación SPI del módulo C.

4.1 Archivos XML

Para que LabVIEW pueda reconocer un módulo C es necesario que exista en la PC donde se trabaja y programa el módulo con el cRIO dos archivos XML, propios de cada módulo, que permitan identificarlo y configurar sus modos de uso. Cuando se está desarrollando el módulo es necesario crear dichos archivos XML, pero cuando ya se quiera comercializarlo NI tiene unos VIs que permiten generar un ejecutable de instalación del módulo, que ya copian estos XML en la computadora donde se quiera trabajar, sin que el usuario lo sepa.

Por lo tanto, primero hay que entender que XML (Lenguaje de Marcado Extensible) es un sistema o lenguaje de programación que permite intercambiar información entre diferentes plataformas, facilitando la organización de recursos y la configuración del programa. Para crear un archivo XML no se requieren programas muy sofisticados, puede hacerse desde un documento de texto, un bloc de notas, desde algunas páginas web o en programas especializados para la creación de este tipo de formato.

De manera general para programar un XML se deben seguir ciertas reglas como son:

1. Ningún elemento puede aparecer sin su correspondiente cierre. Eje. <elemento> …. </elemento>
2. XML es sensible a mayúsculas y minúsculas por lo que no es lo mismo <Kevin> que <kevin>.

3. Los elementos deben ser añadidos y cerrados de manera correcta. Eje. <Kevin> …. </Kevin>.

4. Los atributos deben ir entrecomillados. <VariableNombre=”Kevin”>.

5. Los espacios en blanco se preservan.

6. Los nombres de los elementos no pueden contener espacios.

7. Los elementos no pueden empezar con xml en ningún caso.

8. En XML los nombres de los elementos están encerrados por corchetes triangulares < >.

9. Un atributo especifica una propiedad para el elemento, utilizando un par nombre/valor. Un elemento XML puede tener uno o más atributos. Eje. </InternalChannel>

   <InternalChannel name="RangoDeFrecuencias">

   </InternalChannel>

Con dichas reglas básicas ya se puede programar un archivo XML, ahora para que se identifique y pueda operar un módulo C es necesario crear 2 archivos XML, el primero se llama **ModuleType** que permite identificar al módulo como tal, el segundo se llama **ModuleSupport** que permite configurarlo y programarlo para que funcione con las configuraciones que proporciona NI para los módulos, pero para que funcione hay que programar los archivos con comandos que LabVIEW y el cRIO entiendan.

Para el primer XML (ModuleType) su lenguaje es sencillo, tiene 5 elementos que son los siguientes:

XML Tag	Tipo de Dato	Descripción
ModuleName	String	Especifica el nombre del módulo. Normalmente consta de un acrónimo de dos letras seguido del código de modelo del módulo. Este nombre de módulo también se usa para nombrar los archivos y carpetas que conforman el soporte del módulo.
Description	String	Aparece en el cuadro de diálogo Nuevo módulo de la serie C al agregar un módulo a un proyecto de LabVIEW.
VendorID	Integer	Debe coincidir con la identificación de proveedor que se almacena en el módulo EEPROM. El "VendorID" se suele especificar con un 0x inicial para indicar que es hexadecimal.
ProductID	Integer	Debe coincidir con la identificación de producto que se almacena en el módulo EEPROM. El "ProductID" se suele especificar con un 0x inicial para indicar que es hexadecimal.
ModelCode	Integer	Debe coincidir con el código del modelo de módulo que se almacena en el módulo EEPROM. El "ModelCode" se suele especificar como decimal, ya que se corresponde con el número de modelo del módulo.

Tabla 3, *Tabla de datos para el XML Module Type*

(National Instruments Corporation, 2011)

Su estructura es la siguiente:

```
<ModuleType>
    <ModuleName>AA-9999</ModuleName>
    <Description>MDK Modulo --- fabricado por ---- </Description>
    <VendorID>0x9999</VendorID>
    <ProductID>0x0001</ProductID>
    <ModelCode>9999</ModelCode>
</ModuleType>
```

Donde las partes en rojo son las que se editan para cada módulo.

Para el módulo IIUNAM – 9901 que creo el Instituto de Ingeniería de la UNAM su programa quedo de esta manera:

<ModuleType>

<ModuleName>IIUNAM-9901</ModuleName>

<Description>MDK Instituto de Ingeniería de la UNAM</Description>

<VendorID>0x0001</VendorID>

<ProductID>0x0001</ProductID>

<ModelCode>9901</ModelCode>

</ModuleType>

Para crearlo solo hay que abrir un block de notas, llamarlo de la misma manera que se llama su módulo agregando el _ModuleType, además que terminarlo con .xml en vez de .txt, ejemplo: IIUNAM-9901_ModuleType.xml

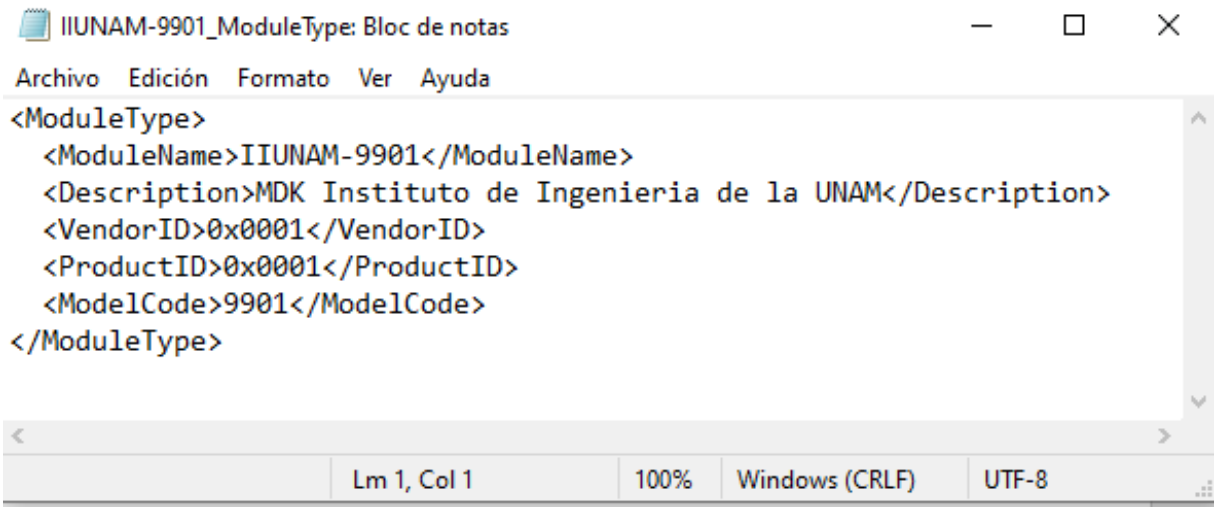


Figura 13. *Captura de pantalla del 1° XML (ModuleType) para el módulo IIUNAM-9901*

Así de simple tenemos el primer XML ahora para que pueda ser reconocido debe estar guardado en una ubicación específica de nuestra PC, dicha ubicación es la siguiente:

C:\Program Files\National Instruments\LabVIEW 2019\Targets\NI\FPGA\cRIO

Pero dicho proceso se realiza de manera automática una vez que se validan los XML y se manda el proyecto en LabVIEW del módulo C a su modo de desarrollo o de lanzamiento. Dicho proceso de validación se relatará más adelante, pero es importante verificar que en la ubicación anterior exista una carpeta llamada other.

En la carpeta de cRIO mencionada aparecen los módulos C que NI vende de manera comercial, para agregar el nuestro hay que crear o abrir una carpeta llamada other

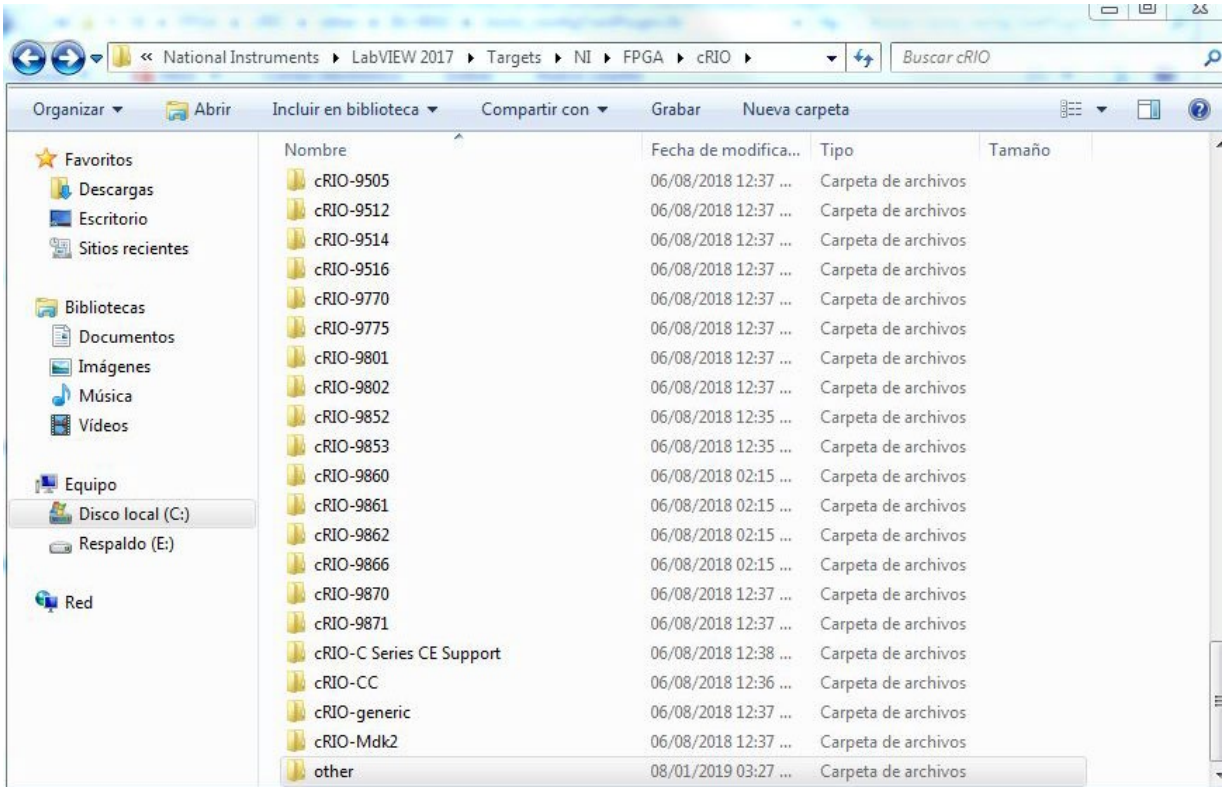


Figura 14. *Captura de pantalla de la ubicación de la carpeta other*

Es importante que nuestro archivo XML (ModuleType) este dentro de una carpeta con el nombre de **nicrio_configToolPlugin.llb**, para que podamos validar el XML:

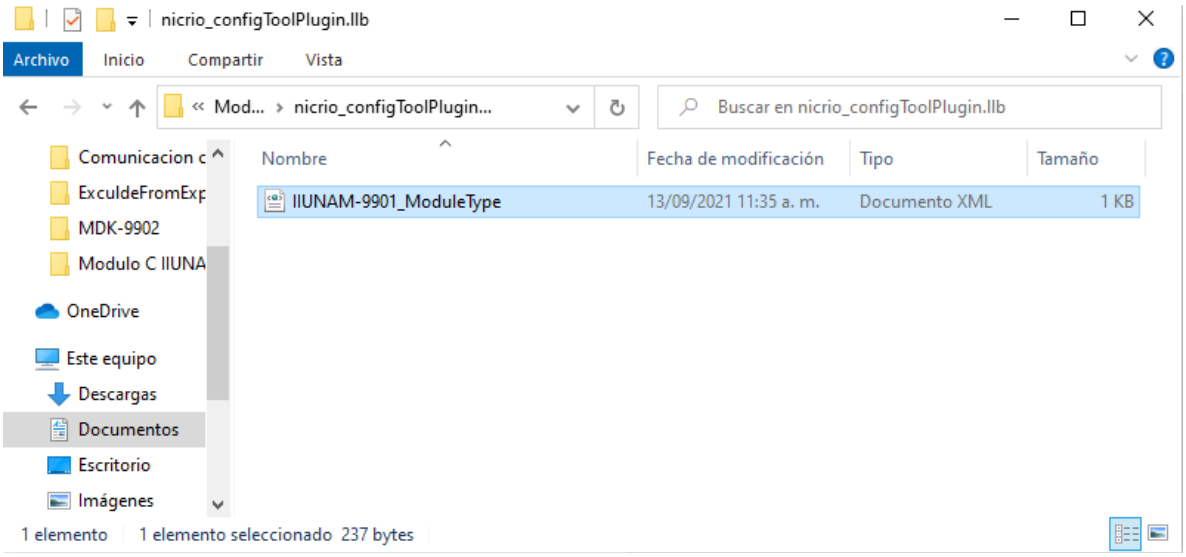


Figura 15. *Captura de pantalla de la ubicación del 1° XML (ModuleType) para el módulo IIUNAM-9901*

Ahora para la creación del 2° archivo XML (ModuleSupport) su lenguaje es mucho más complejo y se requieren muchos más elementos, en las siguientes páginas se describirá el proceso básico para la creación de este programa y en la parte de anexos se tiene una serie de tablas tomadas del manual cRIO_MDK_Software_User_Manual del kit de desarrollo de los módulos C de NI que permiten elaborar programas complejos y específicos para este XML. En dichas tablas se podrá ver el nivel de importancia que tiene el elemento para el programa, su nombre, su tipo de dato, si se requiere o no para la programación del módulo y una breve descripción de su uso.

Para el caso concreto del módulo del instituto crearemos el programa de XML (ModuleSupport) de tal manera que se permita entablar una comunicación SPI en su modo normal de operación para recibir los datos que procesa el módulo (IIUNAM-9901), permitiendo también recibir señales de Entrada/Salida a través de algunas líneas DIO.

Este ejemplo para el 2° archivo XML del módulo es muy básico, pero permite dar una idea clara de cómo se debe programar para que el módulo funcione de manera correcta con LabVIEW.

Como en el archivo anterior para crearlo solo hay que abrir un block de notas, llamarlo de la misma manera que se llama su módulo agregando el _ModuleSupport, además que terminarlo con .xml en vez de .txt, ejemplo: IIUNAM-9901_ModuleSupport.xml

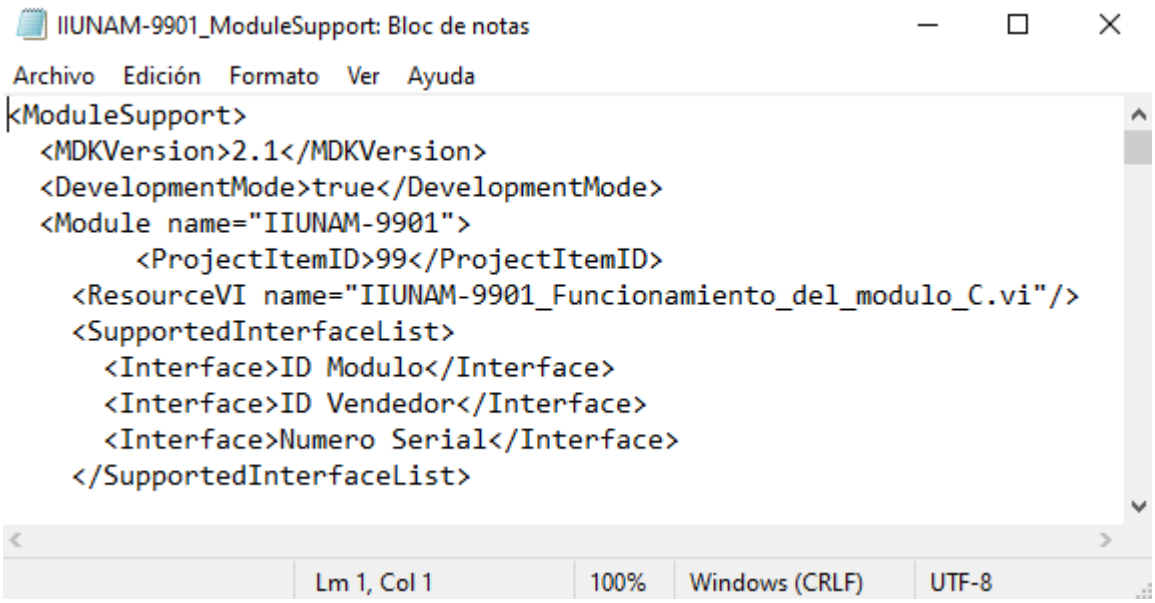


Figura 16. *Captura de pantalla del 2° XML (ModuleSupport) para el módulo IIUNAM-9901*

Este archivo también tiene que ser ubicado en una posición especifica en la computadora donde se está trabajando, la ubicación del 2° XML (ModuleSupport) tiene que estar en la misma carpeta del proyecto donde se encuentra la carpeta **nicrio_configToolPlugin.llb** mencionada anteriormente:

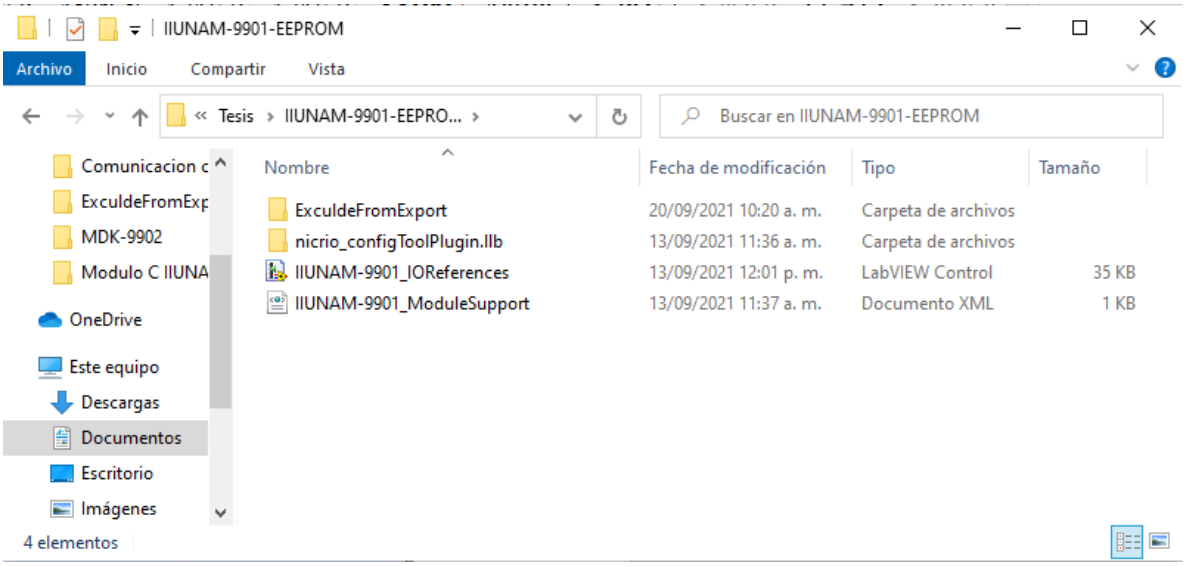


Figura 17. *Captura de pantalla de la ubicación del 2° XML (ModuleSupport) para el módulo IIUNAM-9901*

Como se puede observar es casi la misma ubicación que el archivo anterior, ubicando el archivo XML en la misma carpeta que creamos en el XML anterior (IIUNAM-9901), teniendo una razón e importancia que se verá un poco más adelante.

Ahora la estructura del archivo XML (ModuleSupport) es diferente dependiendo de las características que uno desee darle al módulo, pero todos los módulos C para NI comienzan de la misma manera:

```
<ModuleSupport>
        <MDKVersion>2.1</MDKVersion>
        <DevelopmentMode>true</DevelopmentMode>
        <Module name="MDK-9999">
            <ProjectItemID>99</ProjectItemID>
              <ResourceVI name="MDK-9902_ModuleResource.vi"/>

              <SupportedInterfaceList>
               <Interface>Update Error Status</Interface>
              </SupportedInterfaceList>

            <IOChannelList>
               <IOChannel name="AI0">
                <ProjectItemID>0</ProjectItemID>
                <SupportedInterfaceList>
                 <Interface>Analog Input Channel</Interface>
                </SupportedInterfaceList>
               </IOChannel>
              </IOChannelList>
```

</Module>

</ModuleSupport>

El número que va en el comando MDKVersion depende de la versión que se esté usando, en este caso se usa la versión cRIO MDK 2.1 pero se puede saber la versión que tienes instalada al correr el siguiente VI: *Mdk2Utility_GetInstalledMDKVersion.vi* ubicado en

C:\ProgramFiles\NationalInstruments\labview2019\vi.lib\LabVIEWTargets\FPGA\cRIO\shared\nicrio_Mdk2Utility.

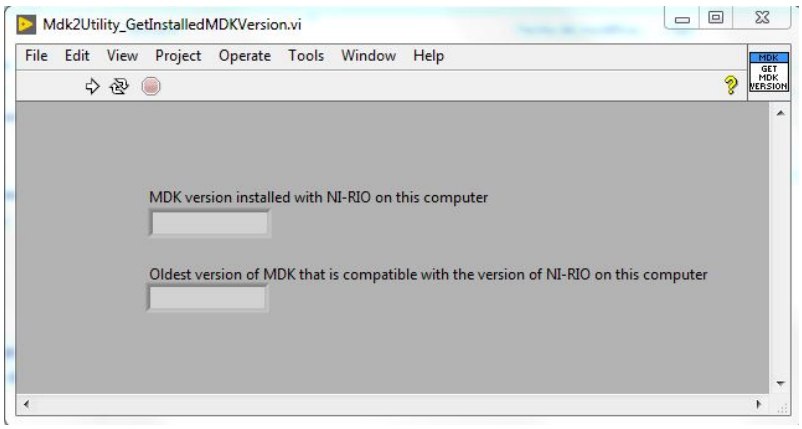


Figura 18. *Captura de pantalla del VI: Mdk2Utility_GetInstalledMDKVersion.vi*

(Instruments, 2020)

Al correrlo aparece en pantalla la versión que se tiene instalada. Eje:

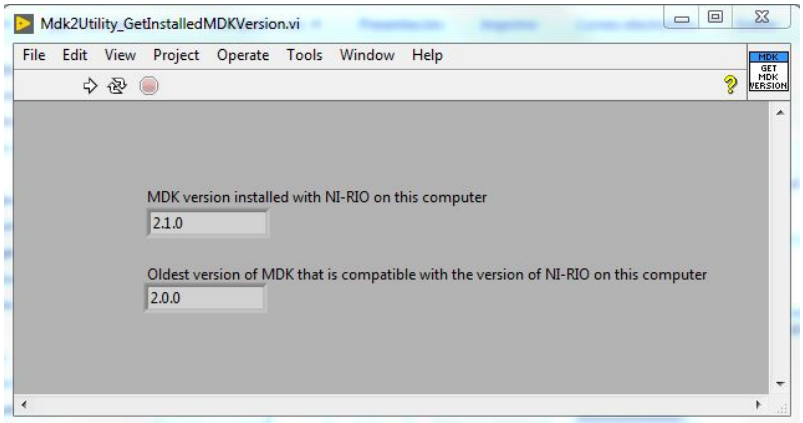


Figura 19. *Captura de pantalla del VI: Mdk2Utility_GetInstalledMDKVersion.vi*

(Instruments, 2020)

El segundo comando *DevolomentMode* sirve para saber si el módulo está en desarrollo o ya está en modo de lanzamiento al activarlo o desactivarlo. El tercero *ModuleName* es para saber que modulo se está programando y configurarlo. El *ProjectItemID* es un numero entero identificador del módulo. El *ResourceVIname* es el nombre del VI que se crea en el FPGA para controlar e indicar todas las funciones internas y/o externas del módulo C creado. Y finalmente el *IOChannelList* permite crear los canales dentro del módulo para permitir su funcionamiento. La estructura para el comienzo del 2° XML del módulo IIUNAM – 9901 que creo el Instituto de Ingeniería de la UNAM queda de esta manera:

```
<ModuleSupport>
        <MDKVersion>2.1</MDKVersion>
        <DevelopmentMode>true</DevelopmentMode>
        <Module name="IIUNAM-9901"/>
            <ProjectItemID>99</ProjectItemID>
            <ResourceVI name="IIUNAM-9901_Funcionamiento_del_modulo_C.vi"/>
              <SupportedInterfaceList>
                <Interface>ID Modulo</Interface>
                <Interface>ID Vendedor</Interface>
                <Interface>Numero Serial</Interface>
              </SupportedInterfaceList>

              <IOChannelList>

                <IOChannel name="DatoB0">
                  <ProjectItemID>0</ProjectItemID>
                  <SupportedInterfaceList>
                      <Interface>Dato1</Interface>
```

```
  </SupportedInterfaceList>
</IOChannel>

<IOChannel name="DatoB1">
  <ProjectItemID>1</ProjectItemID>
  <SupportedInterfaceList>
        <Interface>Dato2</Interface>
  </SupportedInterfaceList>
</IOChannel>

<IOChannel name="DatoB2">
  <ProjectItemID>2</ProjectItemID>
  <SupportedInterfaceList>
        <Interface>Dato3</Interface>
  </SupportedInterfaceList>
</IOChannel>

<IOChannel name="DatoB3">
  <ProjectItemID>3</ProjectItemID>
  <SupportedInterfaceList>
        <Interface>Dato4</Interface>
  </SupportedInterfaceList>
</IOChannel>

<IOChannel name="LecturaSPI">
```

```
                    <ProjectItemID>4</ProjectItemID>
                    <SupportedInterfaceList>
                        <Interface>DatosSPI</Interface>
                    </SupportedInterfaceList>
                  </IOChannel>

                </IOChannelList>
              </Module>
</ModuleSupport>
```

Para programar los canales de Entrada/Salida que tendrá el módulo, como ya se vio, se comienza con <IOChannelList> …. Se crea cada canal y se finaliza con …</IOChannelList>. La estructura para crear cada canal es la siguiente:

```
<IOChannel name="AI0">
        <ProjectItemID>0</ProjectItemID>
        <SupportedInterfaceList>
            <Interface>Analog Input Channel</Interface>
        </SupportedInterfaceList>
</IOChannel>
```

En *IOChannel name* se escribe el nombre de la Entada/Salida que uno desee darle. En *ProjectItemID* se escribe un número de identificación único para cada elemento que debe empezar desde 0 hasta el número de elementos que se tengan en el programa. Finalmente, en la parte que dice Interface se selecciona si este canal será E / S, método o un nodo de propiedad.

Para el módulo IIUNAM-9901 del Instituto de Ingeniería crearon 5 canales (variables internas) que permiten realizar la comunicación SPI, de la manera que se deseaba para este módulo, obteniendo a través de estos mismos la información que los sensores conectados al módulo C mandan de manera digital.

Lo siguiente a programar son los subelementos, siendo estos elementos en el módulo que no son consideradas señales de Entrada/Salida típicas como memorias SD, señales de comunicación externas (Serial, HDMI, Etc.), entre otras cosas. En este caso el módulo IIUNAM-9901 no contiene dichas señales por lo que se continua con la siguiente sección.

El siguiente paso es programar los canales internos que tendrá el módulo, para ello se comienza con el elemento <InternalChannelList> … Se crea cada canal y se finaliza con … </InternalChannelList>. La estructura para crear cada canal interno es la siguiente:

```
<InternalChannel name="Nombre">
    <InternalChannelType> Ocurrence </InternalChannelType>
  <DataType>U8</DataType>
</InternalChannel>
```

En *InternalChannel name* se escribe el nombre de la Entada/Salida que uno desee darle. En *InternalChannelType* se selecciona el tipo de canal que será, las opciones para esta etiqueta son Asincrónica (Asynchronous), Bloqueo (Blocking) y Ocurrencia (Ocurrence). Finalmente, en *DataType* se selecciona el tipo de dato que almacenara siendo sus opciones: I8, U8, I16, U16, I32, U32 y Boolean.

Para el módulo IIUNAM-9901 del Instituto de ingeniería se requirió 10 canales internos para permitir la comunicación SPI y obtener la información básica de la memoria EEPROM del módulo (ID del vendedor, ID del módulo y su número serial).

Lo siguiente a programar son las interfaces, los métodos y las propiedades de las E/S que se programaron, para el módulo IIUNAM-9901 se programaron 8 interfaces, 3 para obtener la identificación del vendedor, la del módulo y su número serial, el resto fueron empleadas

para permitir la comunicación SPI como se tenía planeado para este módulo, pero debido a su complejidad y al hecho de que se desarrollaron diversos subVI para su funcionamiento no se entrara en detalles con dicha programación, pero puede consultarse el ejemplo que da NI en su manual cRIO_MDK_Software_User_Manual del kit de desarrollo de los módulos C en su apéndice B para ver como se hace.

Llegamos a una de las partes más importantes de la programación del XML ModuleSupport que consiste en definir el hardware que posee el módulo y los modos que admite. Para programar esta parte se inicia con el elemento <ModuleModeDefinition> … Se configura y define el módulo y se termina con … </ModuleModeDefinition>.

Comenzaremos programando el Modo normal de operación, la estructura del programa es la siguiente:

```
<NormalOperationMode>
        <SPIConfiguration>
          <SPIHalfTauTicks>10</SPIHalfTauTicks>
        </SPIConfiguration>

        <ConvertPulseConfiguration>
          <ConvertPulseWidth>Long</ConvertPulseWidth>
        </ConvertPulseConfiguration>

        <DoneWaitConfiguration>
          <DoneWaitTimeoutTicks>500</DoneWaitTimeoutTicks>
        </DoneWaitConfiguration>

        <DigitalLines>
          <DIO0>BiDirectional</DIO0>
```

```
        <DIO1>BiDirectional</DIO1>
      </DigitalLines>
  </NormalOperationMode>
```

Esta estructura habilita el modo normal de operación a través del comando *NormalOperationMode*, el comando *SPIConfiguration* habilita la comunicación SPI en el módulo, por su parte *SPIHalfTauTicks* permite determinar el tiempo de espera para cada tick de reloj en tao (para saber cuánto vale un tao ver el manual cRIO_MDK_Hardware_User_Manual del kit de desarrollo de módulos C de NI), el *ConvertPulseConfiguration* habilita el método de conversión de impulsos, el *DoneWaitConfiguration* habilita el método de espera o hecho para el módulo y finalmente, *DigitalLines* habilita las líneas digitales, escribiendo la línea que se habilita (Eje. <DIO0>) y el tipo de función que tendrá, siendo las opciones para esta etiqueta:

- **BiDireccional:** la línea toma como valor predeterminado una entrada en la inserción del módulo y también se puede usar como salida digital.
- **ConstantOutputHigh:** la línea predeterminada es una salida ALTA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran.
- **ConstantOutputLow:** la línea predeterminada es una salida BAJA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran.
- **InputOnly:** la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital.
- **Unused:** la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital.

Para habilitar el modo Auxiliar de comunicación se usa la misma estructura que con el modo normal de operación, pero empleando el comando <AuxiliaryCommunicationMode> en vez de <NormalOperationMode>. Ejemplo:

```
<AuxiliaryCommunicationMode>
        <SPIConfiguration>
          <SPIHalfTauTicks>15</SPIHalfTauTicks>
        </SPIConfiguration>

        <DigitalLines>
          <DIO0>InputOnly</DIO0>
          <DIO1>Unused</DIO1>
        </DigitalLines>
  </AuxiliaryCommunicationMode>
```

Para finalizar el XML se programan las secciones de sincronización y de arbitraje que para el módulo IIUNAM-9901 del Instituto de ingeniería no fueron utilizadas.

El programa final del 2° XML (ModuleSupport) para el módulo IIU-9932 del Instituto de Ingeniería de la UNAM fue el siguiente:

```
<ModuleSupport>
<MDKVersion>2.1</MDKVersion>
  <DevelopmentMode>true</DevelopmentMode>
  <Module name="IIUNAM-9901">
        <ProjectItemID>99</ProjectItemID>
    <ResourceVI name="IIUNAM-9901_Funcionamiento_del_modulo_C.vi"/>
    <SupportedInterfaceList>
      <Interface>ID Modulo</Interface>
      <Interface>ID Vendedor</Interface>
      <Interface>Numero Serial</Interface>
```

```
</SupportedInterfaceList>

<IOChannelList>

  <IOChannel name="DatoB0">
    <ProjectItemID>0</ProjectItemID>
    <SupportedInterfaceList>
          <Interface>Dato1</Interface>
    </SupportedInterfaceList>
  </IOChannel>

  <IOChannel name="DatoB1">
    <ProjectItemID>1</ProjectItemID>
    <SupportedInterfaceList>
          <Interface>Dato2</Interface>
    </SupportedInterfaceList>
  </IOChannel>

  <IOChannel name="DatoB2">
    <ProjectItemID>2</ProjectItemID>
    <SupportedInterfaceList>
          <Interface>Dato3</Interface>
    </SupportedInterfaceList>
  </IOChannel>
```

```
    <IOChannel name="DatoB3">
      <ProjectItemID>3</ProjectItemID>
      <SupportedInterfaceList>
            <Interface>Dato4</Interface>
      </SupportedInterfaceList>
    </IOChannel>

    <IOChannel name="LecturaSPI">
      <ProjectItemID>4</ProjectItemID>
      <SupportedInterfaceList>
            <Interface>DatosSPI</Interface>
      </SupportedInterfaceList>
    </IOChannel>

  </IOChannelList>
</Module>

<InternalChannelList>

  <InternalChannel name="IniciarOperacion">
    <InternalChannelType>Blocking</InternalChannelType>
    <DataType>IIUNAM-9901_TipoDeOperacion.ctl</DataType>
  </InternalChannel>

  <InternalChannel name="OperacionFinalizada">
```

```
  <InternalChannelType>Occurrence</InternalChannelType>
</InternalChannel>

<InternalChannel name="DireccionEEPROM">
  <InternalChannelType>Blocking</InternalChannelType>
  <DataType>U16</DataType>
</InternalChannel>

<InternalChannel name="DatoEEPROM">
  <InternalChannelType>Blocking</InternalChannelType>
  <DataType>U8</DataType>
</InternalChannel>

<InternalChannel name="ErrorDelModulo">
  <InternalChannelType>Asynchronous</InternalChannelType>
  <DataType>IIUNAM-9901_TipoDeError.ctl</DataType>
</InternalChannel>

<InternalChannel name="SPIb0">
  <InternalChannelType>Asynchronous</InternalChannelType>
  <DataType>U8</DataType>
</InternalChannel>

<InternalChannel name="SPIb1">
  <InternalChannelType>Asynchronous</InternalChannelType>
```

```
    <DataType>U8</DataType>
  </InternalChannel>

  <InternalChannel name="SPIb2">
    <InternalChannelType>Asynchronous</InternalChannelType>
    <DataType>U8</DataType>
  </InternalChannel>

  <InternalChannel name="SPIb3">
    <InternalChannelType>Asynchronous</InternalChannelType>
    <DataType>U8</DataType>
  </InternalChannel>

  <InternalChannel name="SPILectura">
    <InternalChannelType>Asynchronous</InternalChannelType>
    <DataType>U8</DataType>
  </InternalChannel>
</InternalChannelList>

<PropertyNodeInterfaceList>

  <Interface name="ID Vendedor">
    <DataType>U16</DataType>
    <Direction>Read</Direction>
    <NodeIcon>AO</NodeIcon>
```

```
  <ReadVIScriptInfo name="VendorIDScriptInfo">
   <UseInstanceData>false</UseInstanceData>
   <VIList>
    <VI name="IIUNAM-9901_ID_Vendedor.vi">
     <SequenceOrder>0</SequenceOrder>
     <VIHasTerminalConnection>true</VIHasTerminalConnection>
    </VI>
    <VI name="IIUNAM-9901_Error.vi">
     <SequenceOrder>1</SequenceOrder>
     <ErrorHandling>true</ErrorHandling>
     <VIScope>NodeScoped</VIScope>
    </VI>
   </VIList>
  </ReadVIScriptInfo>
 </Interface>

<Interface name="ID Modulo">
  <DataType>U16</DataType>
  <Direction>Read</Direction>
  <NodeIcon>AO</NodeIcon>
  <ReadVIScriptInfo name="ModuleIDScriptInfo">
   <UseInstanceData>false</UseInstanceData>
   <VIList>
    <VI name="IIUNAM-9901_ID_Modulo.vi">
     <SequenceOrder>0</SequenceOrder>
```

```
      <VIHasTerminalConnection>true</VIHasTerminalConnection>
     </VI>
     <VI name="IIUNAM-9901_Error.vi">
      <SequenceOrder>1</SequenceOrder>
      <ErrorHandling>true</ErrorHandling>
      <VIScope>NodeScoped</VIScope>
     </VI>
    </VIList>
   </ReadVIScriptInfo>
  </Interface>

 <Interface name="Numero Serial">
   <DataType>U32</DataType>
   <Direction>Read</Direction>
   <NodeIcon>AO</NodeIcon>
   <ReadVIScriptInfo name="SerialNumberScriptInfo">
    <UseInstanceData>false</UseInstanceData>
    <VIList>
     <VI name="IIUNAM-9901_NumeroSerial.vi">
      <SequenceOrder>0</SequenceOrder>
      <VIHasTerminalConnection>true</VIHasTerminalConnection>
     </VI>
     <VI name="IIUNAM-9901_Error.vi">
      <SequenceOrder>1</SequenceOrder>
      <ErrorHandling>true</ErrorHandling>
```

```
        <VIScope>NodeScoped</VIScope>
       </VI>
      </VIList>
     </ReadVIScriptInfo>
    </Interface>

<Interface name="Dato1">
    <DataType>U8</DataType>
    <Direction>BiDirectional</Direction>
    <DefaultDirection>Read</DefaultDirection>
    <NodeIcon>AI</NodeIcon>

    <ReadVIScriptInfo name="CustomChannelRead1">
     <VIList>
      <VI name="IIUNAM-9901_Lectura_B0.vi">
       <SequenceOrder>0</SequenceOrder>
       <VIHasTerminalConnection>true</VIHasTerminalConnection>
      </VI>
      <VI name="IIUNAM-9901_Error.vi">
       <SequenceOrder>7</SequenceOrder>
       <ErrorHandling>true</ErrorHandling>
      </VI>
     </VIList>
    </ReadVIScriptInfo>
```

```
    <WriteVIScriptInfo name="CustomChannelWrite1">
     <VIList>
      <VI name="IIUNAM-9901_Escritura_B0.vi">
       <SequenceOrder>0</SequenceOrder>
       <VIHasTerminalConnection>true</VIHasTerminalConnection>
      </VI>
      <VI name="IIUNAM-9901_Error.vi">
       <SequenceOrder>3</SequenceOrder>
       <ErrorHandling>true</ErrorHandling>
      </VI>
     </VIList>
    </WriteVIScriptInfo>
   </Interface>

<Interface name="Dato2">
    <DataType>U8</DataType>
    <Direction>BiDirectional</Direction>
    <DefaultDirection>Read</DefaultDirection>
    <NodeIcon>AI</NodeIcon>

    <ReadVIScriptInfo name="CustomChannelRead2">
     <VIList>
      <VI name="IIUNAM-9901_Lectura_B1.vi">
       <SequenceOrder>0</SequenceOrder>
       <VIHasTerminalConnection>true</VIHasTerminalConnection>
```

```
        </VI>
        <VI name="IIUNAM-9901_Error.vi">
          <SequenceOrder>7</SequenceOrder>
          <ErrorHandling>true</ErrorHandling>
        </VI>
      </VIList>
    </ReadVIScriptInfo>

    <WriteVIScriptInfo name="CustomChannelWrite2">
      <VIList>
        <VI name="IIUNAM-9901_Escritura_B1.vi">
          <SequenceOrder>0</SequenceOrder>
          <VIHasTerminalConnection>true</VIHasTerminalConnection>
        </VI>
        <VI name="IIUNAM-9901_Error.vi">
          <SequenceOrder>3</SequenceOrder>
          <ErrorHandling>true</ErrorHandling>
        </VI>
      </VIList>
    </WriteVIScriptInfo>
  </Interface>

<Interface name="Dato3">
    <DataType>U8</DataType>
    <Direction>BiDirectional</Direction>
```

<DefaultDirection>Read</DefaultDirection>

<NodeIcon>AI</NodeIcon>

<ReadVIScriptInfo name="CustomChannelRead3">

<VIList>

<VI name="IIUNAM-9901_Lectura_B2.vi">

<SequenceOrder>0</SequenceOrder>

<VIHasTerminalConnection>true</VIHasTerminalConnection>

</VI>

<VI name="IIUNAM-9901_Error.vi">

<SequenceOrder>7</SequenceOrder>

<ErrorHandling>true</ErrorHandling>

</VI>

</VIList>

</ReadVIScriptInfo>

<WriteVIScriptInfo name="CustomChannelWrite3">

<VIList>

<VI name="IIUNAM-9901_Escritura_B2.vi">

<SequenceOrder>0</SequenceOrder>

<VIHasTerminalConnection>true</VIHasTerminalConnection>

</VI>

<VI name="IIUNAM-9901_Error.vi">

<SequenceOrder>3</SequenceOrder>

<ErrorHandling>true</ErrorHandling>

```
        </VI>
       </VIList>
      </WriteVIScriptInfo>
     </Interface>

<Interface name="Dato4">
     <DataType>U8</DataType>
     <Direction>BiDirectional</Direction>
     <DefaultDirection>Read</DefaultDirection>
     <NodeIcon>AI</NodeIcon>

     <ReadVIScriptInfo name="CustomChannelRead4">
      <VIList>
       <VI name="IIUNAM-9901_Lectura_B3.vi">
        <SequenceOrder>0</SequenceOrder>
        <VIHasTerminalConnection>true</VIHasTerminalConnection>
       </VI>
       <VI name="IIUNAM-9901_Error.vi">
        <SequenceOrder>7</SequenceOrder>
        <ErrorHandling>true</ErrorHandling>
       </VI>
      </VIList>
     </ReadVIScriptInfo>

     <WriteVIScriptInfo name="CustomChannelWrite4">
```

```
      <VIList>
       <VI name="IIUNAM-9901_Escritura_B3.vi">
        <SequenceOrder>0</SequenceOrder>
        <VIHasTerminalConnection>true</VIHasTerminalConnection>
       </VI>
       <VI name="IIUNAM-9901_Error.vi">
        <SequenceOrder>3</SequenceOrder>
        <ErrorHandling>true</ErrorHandling>
       </VI>
      </VIList>
    </WriteVIScriptInfo>
  </Interface>

<Interface name="DatosSPI">
    <DataType>IIUNAM-9901_CanalSPI.ctl</DataType>
    <Direction>BiDirectional</Direction>
    <DefaultDirection>Read</DefaultDirection>
    <NodeIcon>AI</NodeIcon>

    <ReadVIScriptInfo name="CustomChannelRead5">
      <VIList>
       <VI name="IIUNAM-9901_Lectura_SPI.vi">
        <SequenceOrder>0</SequenceOrder>
        <VIHasTerminalConnection>true</VIHasTerminalConnection>
       </VI>
```

```
      <VI name="IIUNAM-9901_Error.vi">
        <SequenceOrder>7</SequenceOrder>
        <ErrorHandling>true</ErrorHandling>
      </VI>
    </VIList>
  </ReadVIScriptInfo>

  <WriteVIScriptInfo name="CustomChannelWrite5">
    <VIList>
      <VI name="IIUNAM-9901_Escritura_SPI.vi">
        <SequenceOrder>0</SequenceOrder>
        <VIHasTerminalConnection>true</VIHasTerminalConnection>
      </VI>
      <VI name="IIUNAM-9901_Error.vi">
        <SequenceOrder>3</SequenceOrder>
        <ErrorHandling>true</ErrorHandling>
      </VI>
    </VIList>
  </WriteVIScriptInfo>

 </Interface>

</PropertyNodeInterfaceList>

<ModuleModeDefinition>
```

```
  <NormalOperationMode>
    <SPIConfiguration>
      <SPIHalfTauTicks>10</SPIHalfTauTicks>
    </SPIConfiguration>

    <ConvertPulseConfiguration>
      <ConvertPulseWidth>Long</ConvertPulseWidth>
    </ConvertPulseConfiguration>
  </NormalOperationMode>

  <AuxiliaryCommunicationMode>
    <SPIConfiguration>
      <SPIHalfTauTicks>15</SPIHalfTauTicks>
    </SPIConfiguration>

  </AuxiliaryCommunicationMode>
 </ModuleModeDefinition>

</ModuleSupport>
```

Lo siguiente que se debe hacer es confirmar que nuestros archivos XML estén bien programados y escritos, para eso NI tiene un VI que nos permite verificar su sintaxis, el VI es el Mdk2Utility_GenerateModuleSupportExport.vi ubicado en:

C:\ProgramFiles\NationalInstruments\LabVIEW2019\vi.lib\LabVIEWTargets\FPGA\cRIO\shared\nicrio_Mdk2Utility\ Mdk2Utility_GenerateModuleSupportExport.vi.

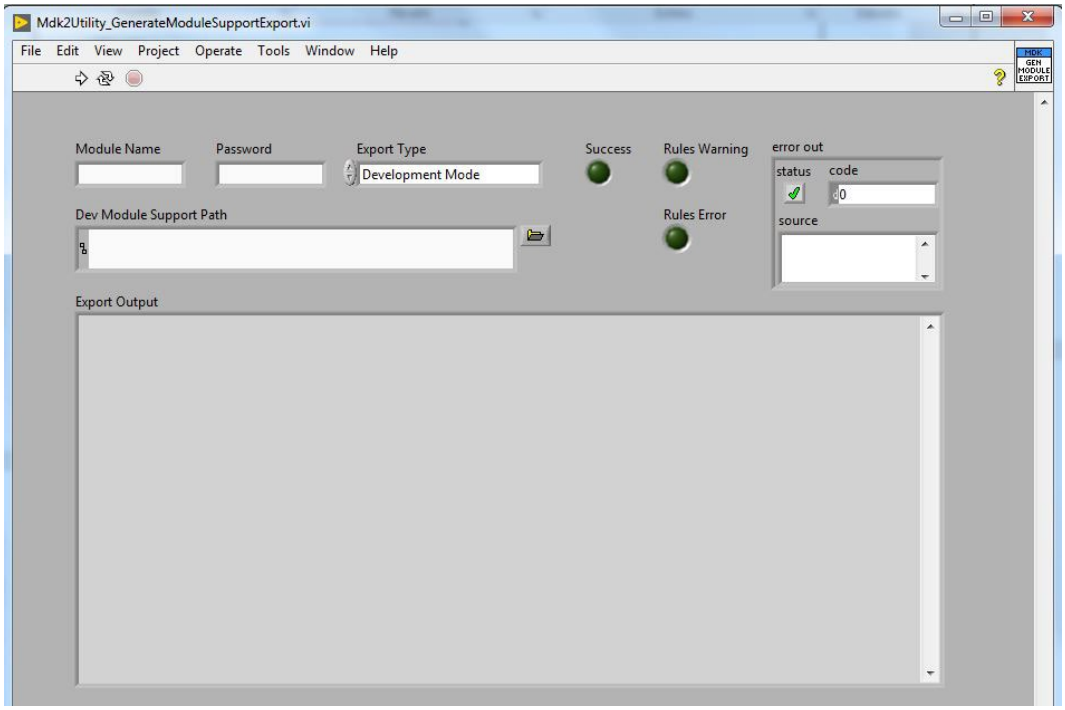


Figura 20. *Programa Mdk2Utility_GenerateModuleSupportExport.vi*

(Instruments, 2020)

Para verificar los archivos hay que escribir el nombre de nuestro modulo y cambiar en Export Type a la opción de *Valide XML Only (No Export)* como se muestra en la siguiente imagen:

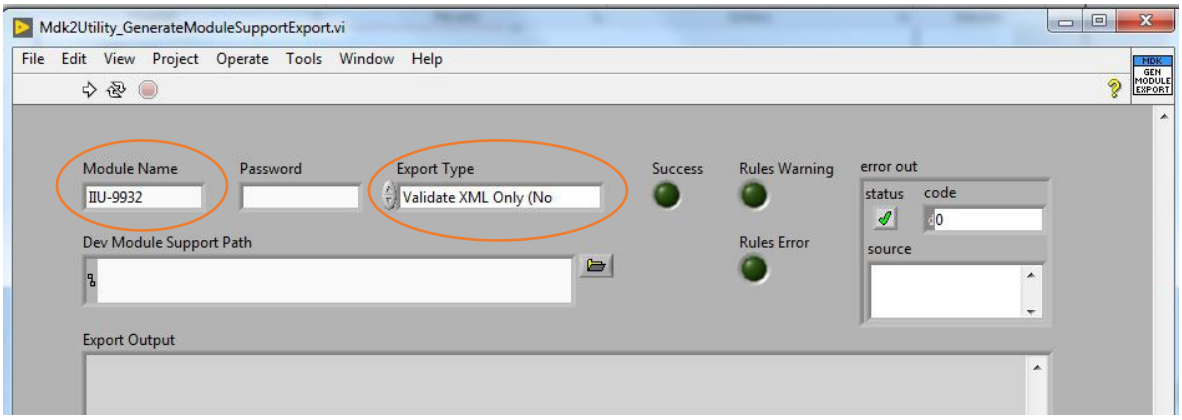


Figura 21. Ejemplo 1 - *Programa Mdk2Utility_GenerateModuleSupportExport.vi*

(Instruments, 2020)

El siguiente paso es cargar en Dev Module Support Path la carpeta donde están ubicados los archivos XML (En este caso en IIU-9932). La ubicación seria:

C:\ProgramFiles\NationalInstruments\LabVIEW2017\Targets\NI\FPGA\cRIO\other\IIU-9932

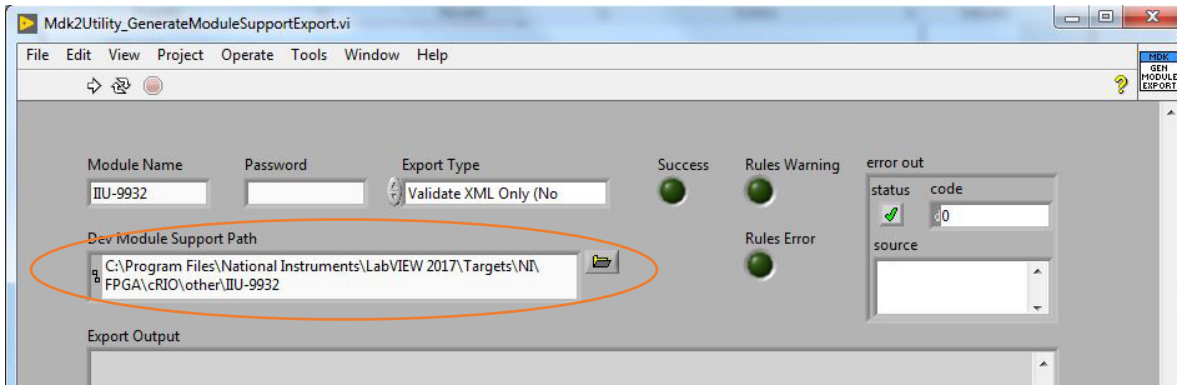


Figura 22. *Imágenes del programa Mdk2Utility_GenerateModuleSupportExport.vi*

(Instruments, 2020)

Es importante mencionar que el programa no permite cargar una carpeta, solamente archivos por lo que se recomienda escribir la dirección de manera manual o seleccionar el archivo de XML Module Support, para luego borrar el nombre del archivo y apuntar la dirección a la carpeta, ya que si no se apunta a la carpeta aparecerá un error.

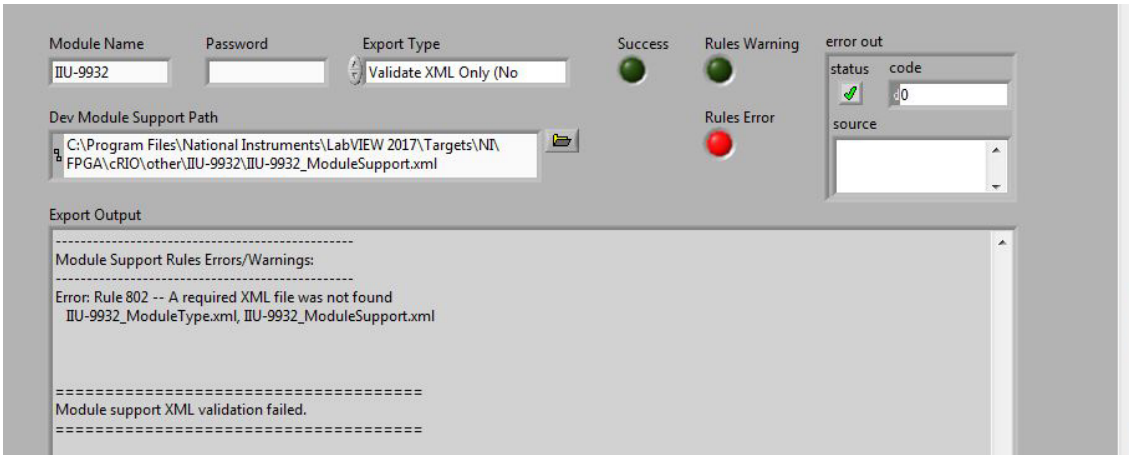


Figura 23. *Error del programa Mdk2Utility_GenerateModuleSupportExport.vi al apuntar a un archivo y no a una carpeta*

(Instruments, 2020)

Si al correr el programa todo está bien, aparecerá un mensaje de validación completa en el apartado de Export Output como se muestra en la siguiente imagen:

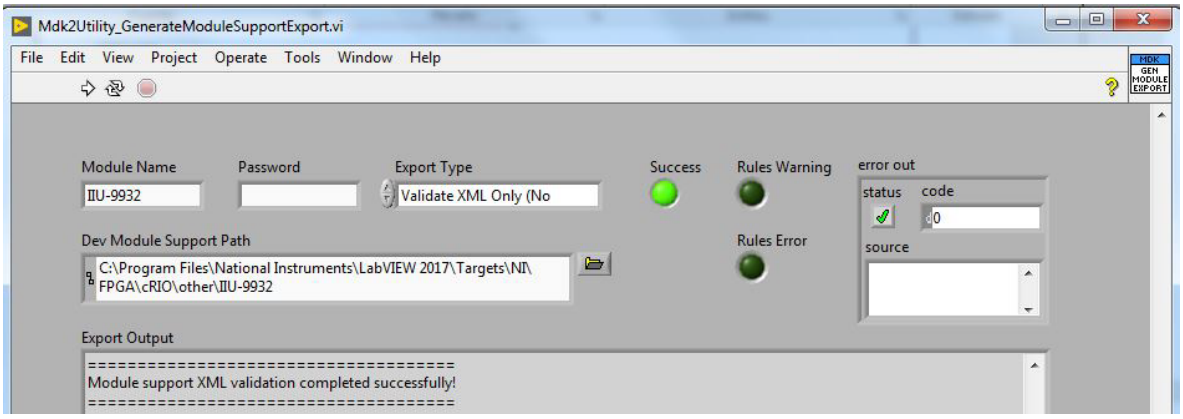


Figura 24. *Imagen del programa Mdk2Utility_GenerateModuleSupportExport.vi que muestra una validación de los archivos XML correcta*

(Instruments, 2020)

Si los archivos están mal programados se mostrará cual es el problema específico y en qué línea o parte del programa encontrarlo, en este mismo apartado (Export Output) para que puedan ser corregido.

Este mismo programa permite que nuestro proyecto se encuentre en modo desarrollo o en modo lanzamiento una vez que los archivos XML estén validados. El modo desarrollo nos permite configurar las entradas y/o salidas de nuestro modulo, programar las acciones a realizar en caso de errores, así como configurar y programar el funcionamiento del mismo, de tal manera que el usuario final no vea todos estos procesos que se hacen mientras se está usando. Finalmente, el modo lanzamiento verifica que todo lo programado en el modo desarrollo este bajo las reglas que tiene National Instruments en sus módulos C y que funcione con sus cRIOs, permitiendo cargar nuestro modulo como cualquiera de los que ellos venden. Para poder entender a más detalle estos modos de programación es necesario plantear algunas cuestiones básicas para la programación en FPGA y para el uso del MDK por lo mientras planteo dichas bases hablare a más detalle de dichos modos de programación.

4.2 Creación de un proyecto en LabVIEW para trabajar con módulos C

Una vez que se tengan los programas XML validados y programados para que su módulo trabaje como se diseñó, hay que generar uno o varios proyectos de LabVIEW para poder trabajar en la computadora y en el cRIO con el módulo.

Es importante mencionar que es necesario tener instalada alguna versión de LabVIEW compatible con su versión de MDK, además de tener el software o complemento necesario de NI para poder trabajar con RIO, también debe cargarse en el cRIO el software necesario para trabajar con el FPGA, estas condiciones necesarias para continuar dependen de los componentes y sistemas que se tengan por lo que se considerara para este trabajo que ya se tiene con dicha estructura, si se requiere más información sobre esto puede consultar la página oficial de NI: http://www.ni.com/es-mx.html.

Bien, entonces para crear un proyecto para trabajar con el módulo desde LabVIEW se abre el programa y se crea un proyecto nuevo, ya sea un proyecto en blanco o directamente un proyecto simple de cRIO, como ejemplo se creará desde uno en blanco:

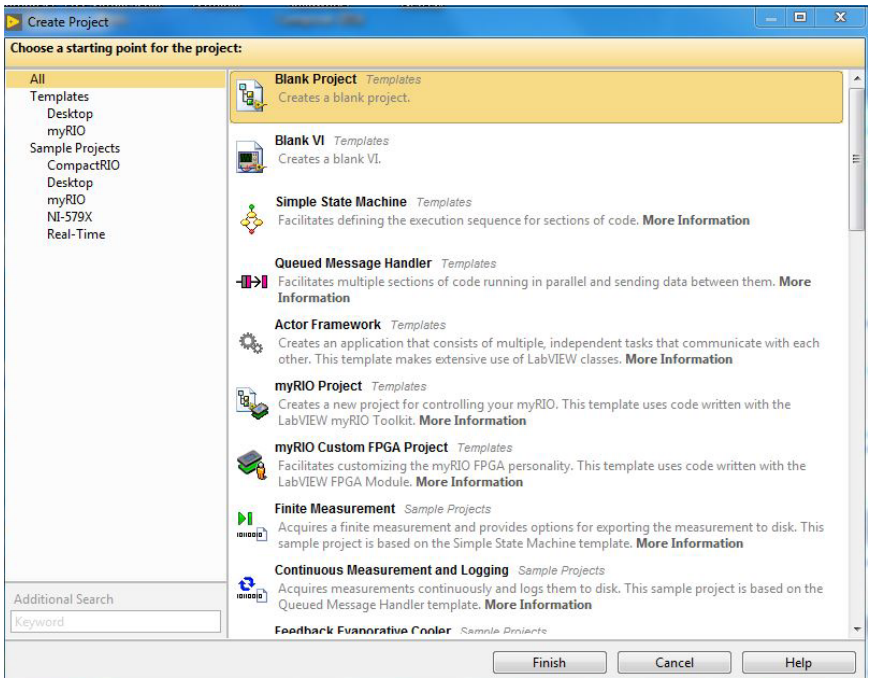


Figura 25. *Proyecto nuevo en LabVIEW 2017*

(Instruments, 2020)

Una vez creado se nombra y guarda en la ubicación de la computadora que uno quiera. Lo siguiente es seleccionar el proyecto con el botón derecho del mouse, seleccionar en new y luego en Targets and Devices …

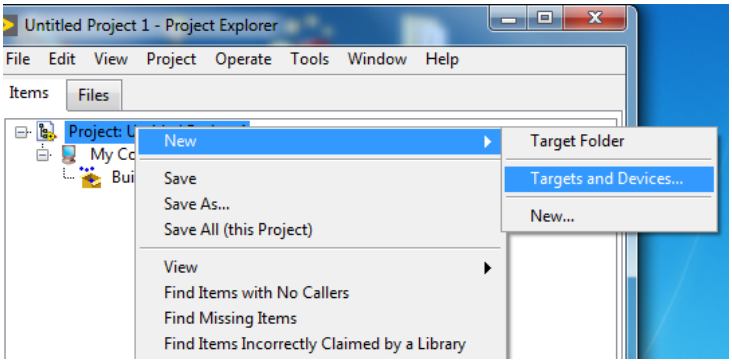


Figura 26. *Proyecto nuevo en LabVIEW 2017*

(Instruments, 2020)

Se mostrará un menú como se muestra en la siguiente imagen, donde buscaremos la carpeta de Real – Time CompactRIO y seleccionaremos el cRIO con el que vayamos a trabajar.

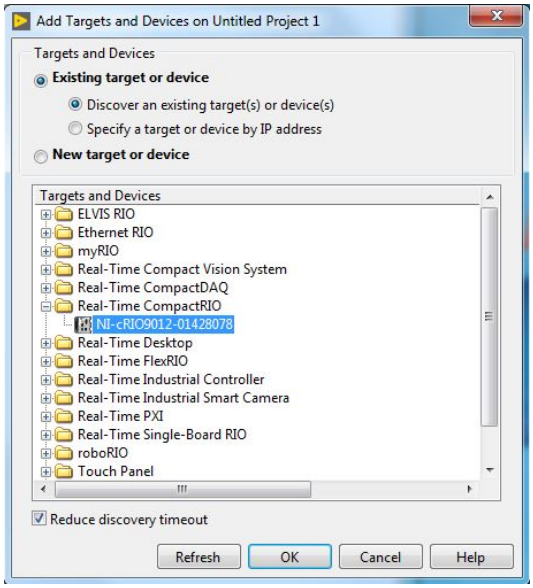


Figura 27. *Agregar una tarjeta o dispositivo al proyecto*

(Instruments, 2020)

Cuando se haya seleccionado el compact se abrirá la siguiente ventana:

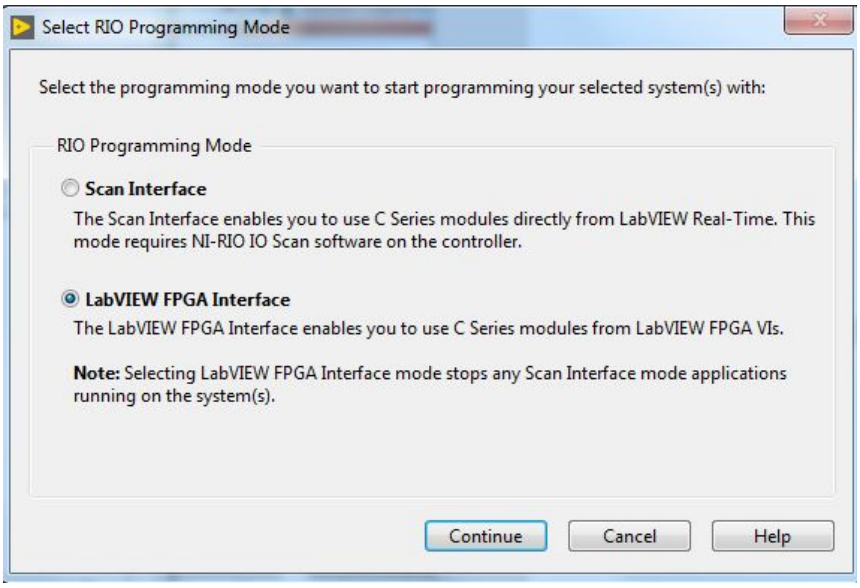


Figura 28. *Selector de modo de programación de RIO*

(Instruments, 2020)

Donde se selecciona la opción de LabVIEW FPGA Interface y se selecciona el botón de continuar. Una vez que haya cargado el cRIO y haya encontrado los módulos conectados a él se creara un proyecto como el siguiente:

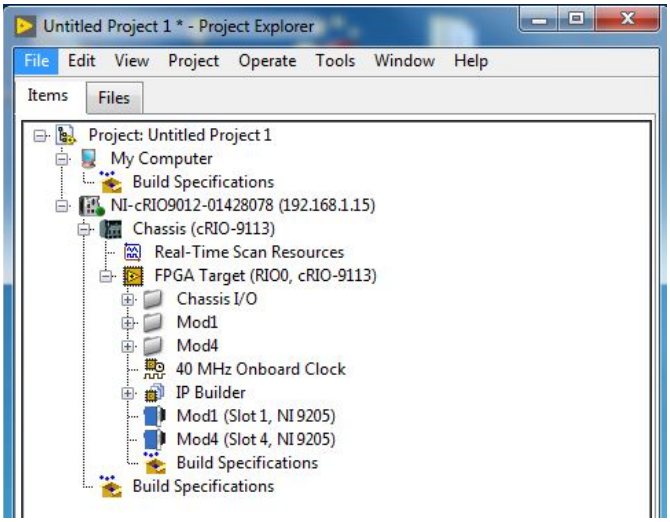


Figura 29. *Proyecto FPGA en LabVIEW 2017*

(Instruments, 2020)

El último paso es realizar la conexión con el cRIO para ello lo seleccionamos en el proyecto y con el clic derecho del mouse seleccionamos la opción de *conect*, aparecerá una

ventana como se muestra a continuación y si todo está bien el mensaje final será Deployment completed successfully, por lo que le podemos dar en *Close* y listo el cRIO está listo para trabajar en LabVIEW.

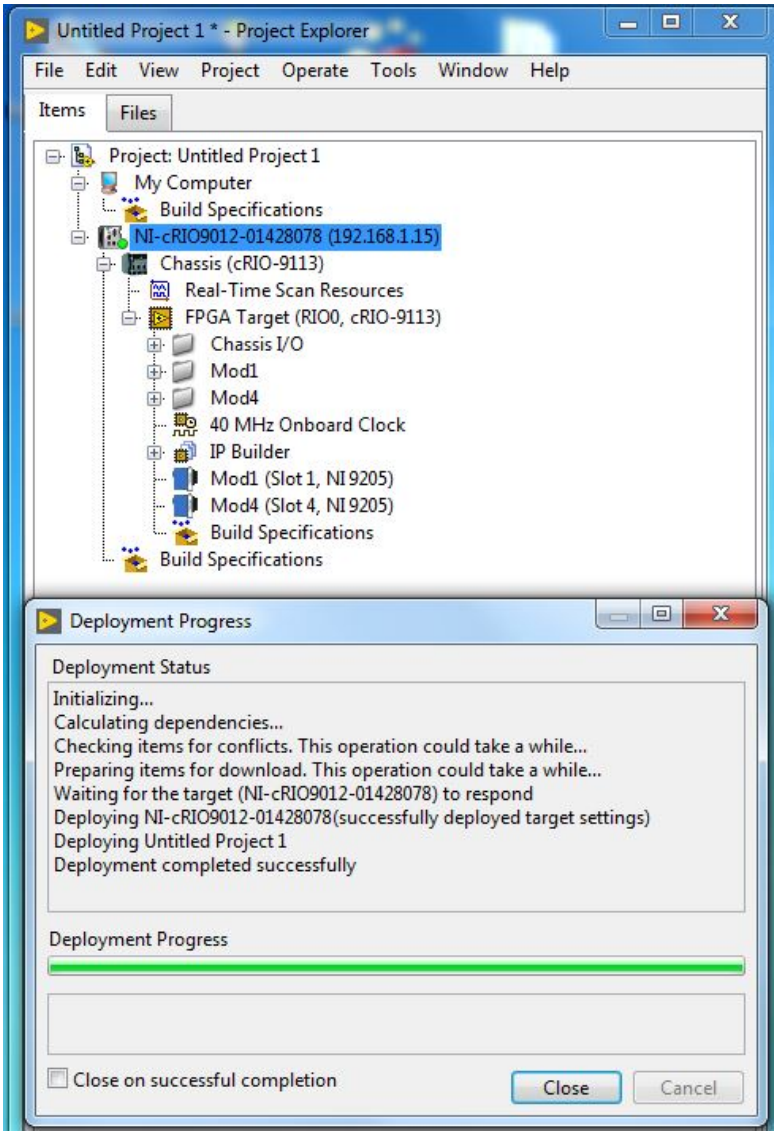


Figura 30. *Conexión exitosa del compactRIO con LabVIEW*

(Instruments, 2020)

Una vez que sepamos cómo crear proyectos para trabajar con el cRIO y el módulo en LabVIEW es hora de hacer que dicho proyecto pueda reconocer nuestro modulo para ello hay que generar un control de referencias de E/S específico para el módulo, eso se hace de manera automática por medio del programa de NI ubicado en:

C:\ProgramFiles\NationalInstruments\LabVIEW2019\vi.lib\LabVIEWTargets\FPGA\cRIO\shared\nicrio_Mdk2Utility\Mdk2Utility_CreateIOReferenceClusterControl.vi

Ejecutamos este VI y apúntalo a la carpeta de muestro modulo. Ejemplo:

C:\ProgramFiles\NationalInstruments\LabVIEW2019\Targets\NI\FPGA\cRIO\other\IIUNAM-9901

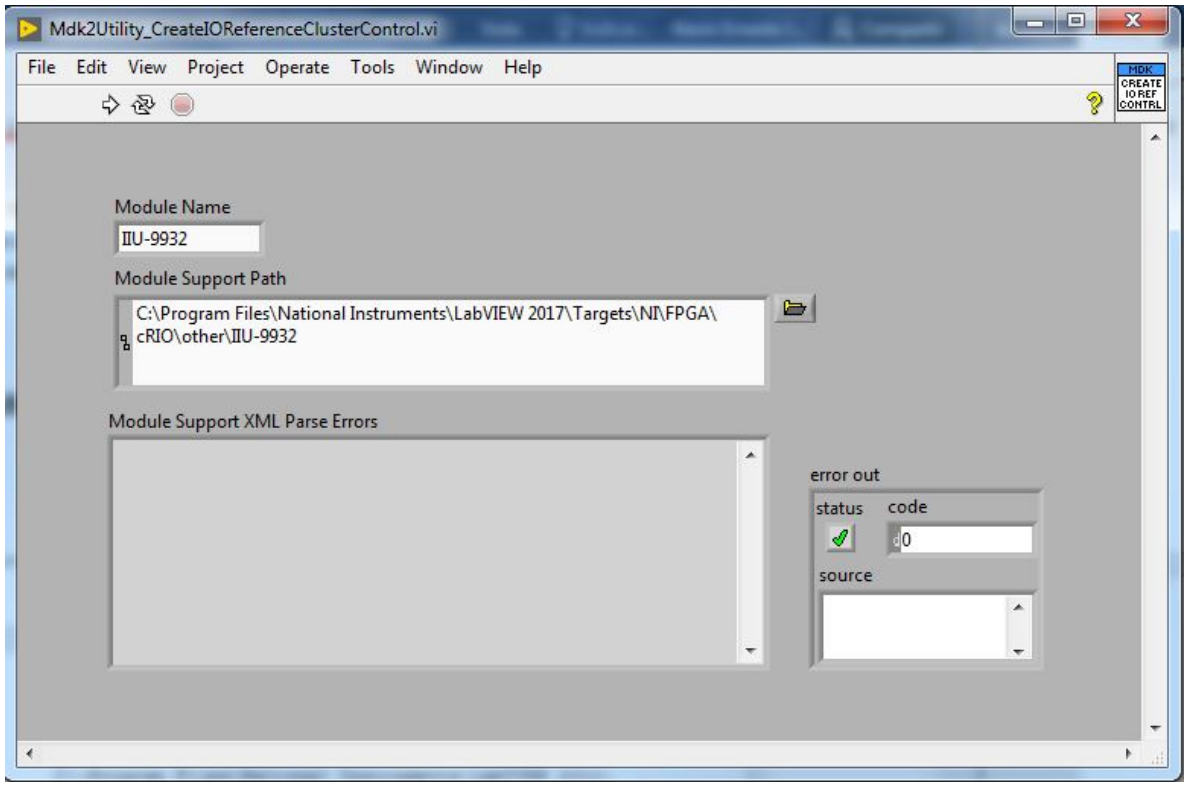


Figura 31. *Programa Mdk2Utility_CreateIOReferenceClusterControl.vi*

(Instruments, 2020)

Una vez ejecutado en la carpeta de nuestro modulo se creará un archivo llamado IIUNAM-9901_IOReferences.ctl como se muestra en la siguiente imagen:

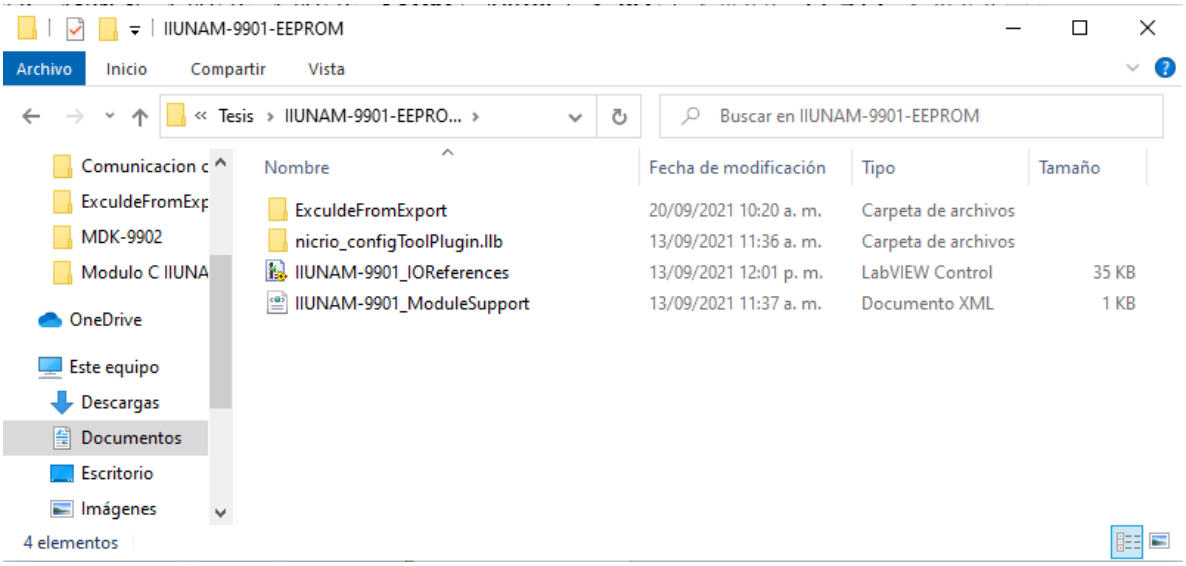


Figura 32. *Creación del control de referencias de E/S específico para el módulo IIUNAM-9901*

Una vez que lo tengamos el siguiente paso es agregar a nuestro proyecto de LabVIEW FPGA nuestro módulo de manera manual. Para ello creamos el proyecto como se vio anteriormente, hasta estar conectados con el cRIO, después damos clic derecho con el mouse en la opción de FPGA, seleccionamos la opción de nuevo y luego en la opción de módulos de la serie C, como se muestra en la siguiente imagen:

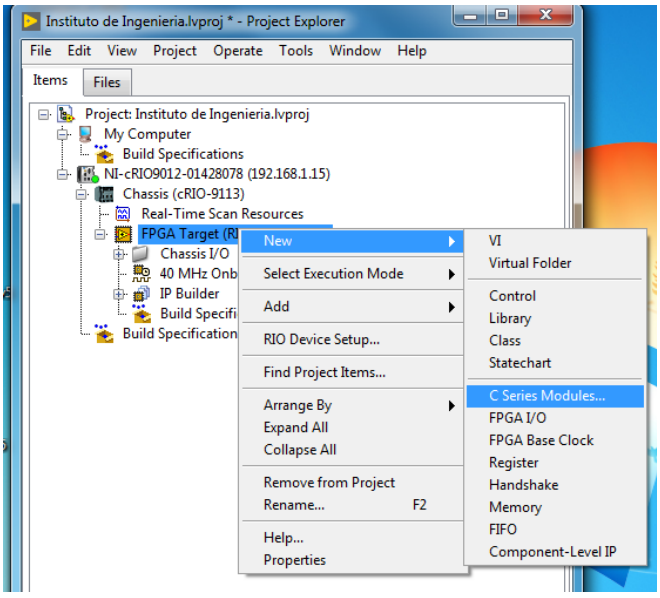


Figura 33. *Menú para agregar un control de E/S de un módulo propio a LabVIEW*

(Instruments, 2020)

En la ventana que se abre seleccionamos *New target or device*, después seleccionamos C Series Module y apretamos *OK*.

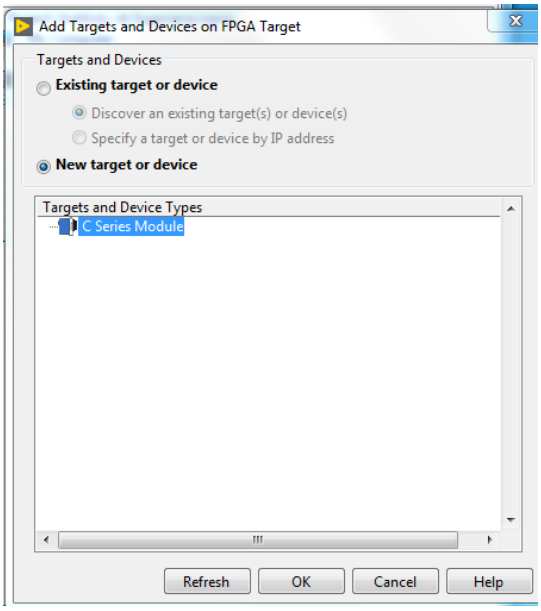


Figura 34. *Ventana emergente para seleccionar un nuevo módulo en LabVIEW*

(Instruments, 2020)

Aparece una nueva ventana donde nombramos nuestro modulo, seleccionamos el slot en el cRIO donde se va a conectar y buscamos nuestro modulo C

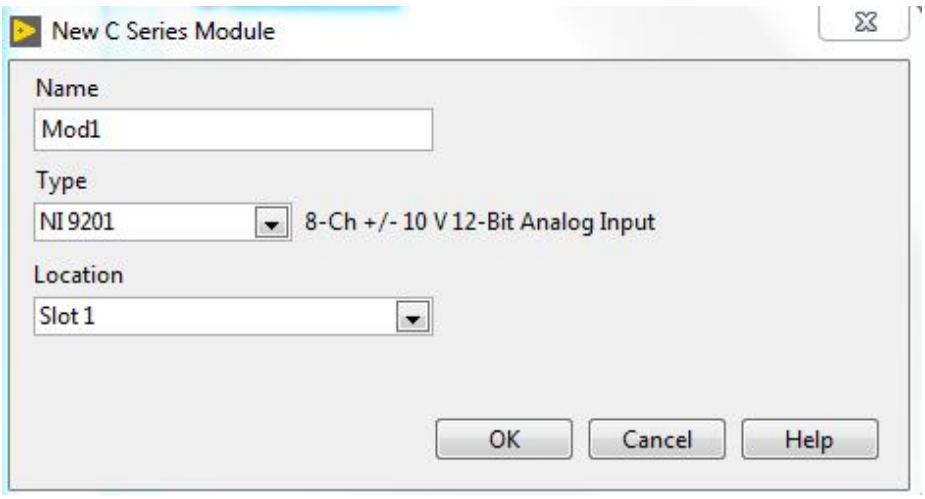


Figura 35. *Anexo manual de un nuevo módulo en el FPGA*

(Instruments, 2020)

Como ejemplo ilustrativo así queda la ventana emergente para ingresar al proyecto el módulo IIU-9932:

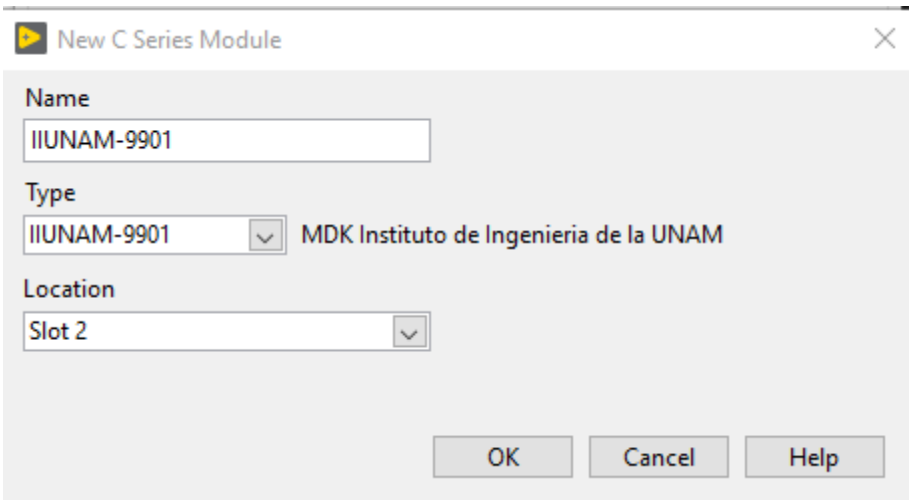


Figura 36. *Anexo manual del módulo IIUNAM-9901 en el FPGA*

Se da *OK* y se agrega el proyecto como se ve a continuación:

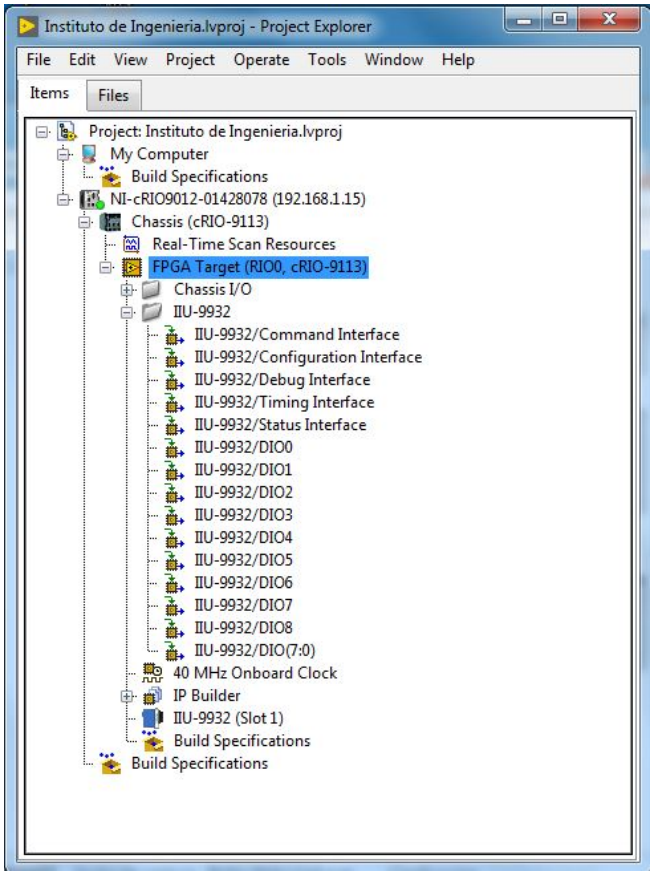


Figura 37. *Visualización de las E/S de un módulo C en un proyecto de LabVIEW*

El último paso para poder empezar a programar en LabVIEW con muestro modulo es agregar las paletas del MDK a LabVIEW, para ello hay que copiar los archivos ubicados en:

C:\Program Files\National Instruments\CompactRIO\CompactRIO MDK 2.1\LabVIEW API Palette_CrioMdk2Api

En la ubicación:

C:\ProgramFiles\NationalInstruments\LabVIEW2019\Targets\NI\FPGA\menus\FPGACategories\Programming_CrioMdk2Api.

En caso de que alguna carpeta no exista, se crea. Después de copiarlos se reinicia la computadora y listo, al crear un VI en el FPGA y abrir la paleta de funciones podremos encontrar las paletas del MDK como se muestra en la siguiente imagen:

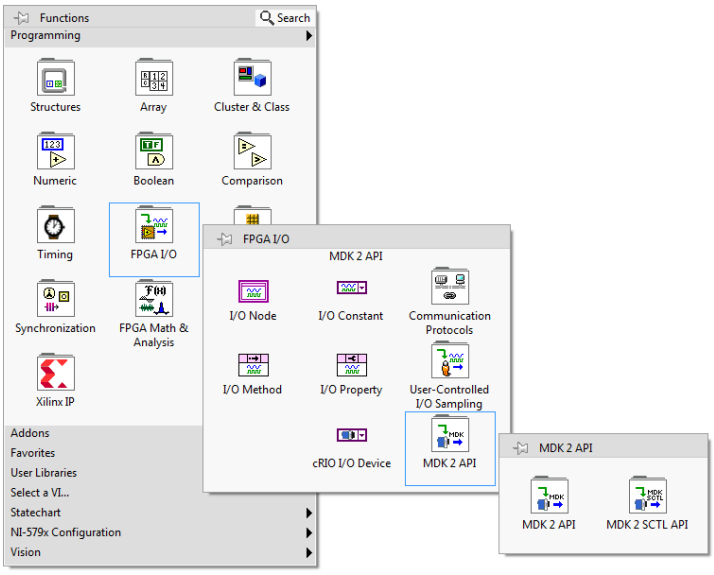


Figura 38. *Paleta de funciones del MDK en LabVIEW*

(Instruments, 2020)

En caso de que no aparezcan las paletas o marquen error, hay que agregarlas de manera manual en la ventana de *Tools* en la parte superior del VI, seleccionamos la opción de avanzado y luego en editar paletas, buscamos las del MDK y las agregamos. Para más información consulte la página oficial de NI.

Con los pasos anteriores ya tenemos un proyecto en LabVIEW para trabajar con un módulo C propio, el siguiente paso a seguir es escribir y leer en la memoria EEPROM de nuestro modulo, para ello se crearon dos maneras de escritura en la EEPROM del módulo C, la primera es usando la USB-845x de NI como intermediario para la comunicación entre LabVIEW y el módulo, para la segunda hay que conectar el módulo C en el slot que seleccionamos en el cRIO y crear un VI en el FPGA. Por último, se mostrará cómo usar un programa creado por NI para realizar el proceso de escritura de la EEPROM.

4.3 Lectura/Escritura de la EEPROM

Para poder escribir los datos en la EEPROM para que nuestro modulo sea reconocido primero hay que saber que datos se van a guardar, los datos para identificar un módulo tienen la siguiente estructura:

Tabla 4. *Identificación del módulo en la EEPROM.*

Dirección [Hex]	**Descripción**	**Tamaño en Bytes**
0x0000–0x0001	EEPROM Size / Tamaño de la EEPROM	2
0x0002–0x0003	Start Sentinel / Comando de inicio	2
0x0004–0x0005	Reserved (0x0000) / Reservado	2
0x0006–0x0007	Product ID / Identificación del Producto	2
0x0008–0x0009	Vendor ID / Identificación del vendedor	2
0x000A–0x000B	Module Model Code / Código del modelo del modulo	2
0x000C–0x000F	Extended Serial Number (for future use) / Número de serie ampliado	4
0x0010–0x0013	Serial Number / Número de serie	4
0x0014	Configuration Version Major / Versión de configuración mayor	1
0x0015	Configuration Version Minor / Versión de configuración menor	1
0x0016–0x0017	Module Descriptor Address / Dirección del descriptor del modulo	2
0x0018–0x002F	Reserved (0x00) / Reservado	24

Cada concepto de la tabla se describe de la siguiente manera:

- Tamaño de la EEPROM:

Dato: (ppps ssss 0000 0000) donde p = tamaño de página y s = tamaño EEPROM. Los primeros tres bits (p) indican el tamaño de la página, y los últimos cinco bits indican el tamaño EEPROM. El tamaño de página se representa como 0 para páginas de 8 bytes, 1 para páginas de 16 bytes, 2 para páginas de 32 bytes, hasta 7 para páginas de 2048 bytes. El tamaño EEPROM se representa como 7 para 128 bytes, 8 para 256 bytes, 9 para 512 bytes, y así sucesivamente. Por lo tanto, una memoria de 4 Kbytes con páginas de 32 bytes se representaría como 0x4C00. Los valores 0x0000 y 0xFF00 están reservados como valores no válidos para dar una indicación rápida de una parte no programada o un intento de lectura no válido.

Ejemplo. Para el módulo IIUNAM-9901 se seleccionó una memoria de 16K con 2048 páginas. Por lo que s = 111 b = 7 d, mientras que p = 14 d = 1110 b lo que nos da como dato = 1110 1110 0000 0000 b = EE 00 hex.

- Comando de Inicio:

Este es un valor conocido que se encuentra en todas las EEPROM de la Serie C. Se usa para verificar la comunicación con la EEPROM. El valor siempre es 0x3A 29

- Reservado:

Un campo reservado de 16 bits que se debe escribir con 0x0000.

- Identificación del Producto:

Es un numero único de 16 bits que se usa exclusivamente para su módulo. Se debe garantizar que cada módulo que se diseñe con el mismo ID de proveedor tenga una ID de modulo única. En el caso del módulo IIUNAM-9901 fue 0x0001.

- Identificación del vendedor:

Es una identificación única de 16 bits para cada fabricante que asigna NI. Para obtener dicho número hay que contactar con National Instruments. En el caso del módulo IIUNAM-9901 fue 0x9901 que en hexadecimal queda como 26AD.

- Código del modelo del módulo:

Es el número del modelo en decimal. Por ejemplo, para el IIUNAM-9901 es el 9901 en decimal que convertido en hexadecimal da 0x26CC.

- Número de serie ampliado:

Su valor debe ser 0x0000. Se reserva para el caso en que el número de serie rebase 8 dígitos hexadecimales.

- Número de serie:

Es un número de 32 bits que es único para cada copia del módulo. Por ejemplo, si el número de serie es 98C57, el valor es 0x00098C57. Como ilustración para el módulo IIUNAM-9901 se empleó el 0x1111.

- Versión de configuración mayor:

Esta es la versión de la tabla EEPROM. La versión que se describe aquí es 1.1. Por lo que el dato es 0x11.

- Versión de configuración menor:

Esta es la versión de la tabla EEPROM. La versión que se describe aquí es 1.1. Por lo que el dato es 0x11.

- Dirección del descriptor del modulo

Es un entero de 16 bits que indica la dirección absoluta en la EEPROM donde se inicia el descriptor del módulo. Si la EEPROM usa la Tabla de Calibración, el Descriptor del Módulo comienza inmediatamente después de la Tabla de Calibración. Si la EEPROM no

utiliza la Tabla de Calibración, el Descriptor del Módulo comienza inmediatamente después del Bloque de Identificación, y la dirección es 0x0030.

- Reservado

Esta serie de bytes está reservada para uso futuro en versiones de configuración posteriores. Estos bytes deberían estar escritos con 0x00, pero el software debería leerlos como valores que no importan.

Por lo que la tabla de datos de identificación para el módulo IIIUNAM-9901 es la siguiente:

Tabla 5. *Valores de la EEPROM para el módulo c IIUNAM-9901*

Address (hex)	Data (hex)	Address (hex)	Data (hex)	Address (hex)	Data (hex)	Address (hex)	Data (hex)
0	EE	C	0	18	0	24	0
1	0	D	0	19	0	25	0
2	3A	E	0	1A	0	26	0
3	29	F	0	1B	0	27	0
4	0	10	1	1C	0	28	0
5	0	11	1	1D	0	29	0
6	0	12	1	1E	0	2A	0
7	1	13	1	1F	0	2B	0
8	99	14	1	20	0	2C	0
9	32	15	1	21	0	2D	0
A	26	16	0	22	0	2E	0
B	CC	17	30	23	0	2F	0

Con dicha información ya podemos escribir los datos en la memoria EEPROM del módulo, solo es importante mencionar que en caso de querer realizar la tabla de calibración del módulo se empieza desde la dirección 0x0030, donde se sigue la siguiente estructura:

Tabla 6. *Calibración del módulo.*

Address	Short Description	Size in Bytes
0x0030–0x0057	Calibration Table Header / Encabezados de la tabla	40
0x0058–end	Factory Calibration Block / Factores de calibración de fabrica	Varios
	External Calibration Block / Factores de calibración Externos	Varios

Ya con los datos lo que procede es escribirlos en la memoria EEPROM del módulo C.

4.3.1 Usando NI – 845 para la escritura en la EEPROM

La USB-845x de NI es una interfaz maestra para conectarse y comunicarse entre dispositivos de circuitos inter-integrados (I2C), de bus de administración del sistema (SMBus) y de una interfaz periférica serial (SPI). Con conectividad USB plug-and-play, la USB-845X es una solución portátil para comunicar con electrónicos de consumo y circuitos integrados. Además, la interfaz proporciona +5 V y GND para energizar circuitos sin fuente de alimentación externa.

Figura 39. *NI 8451*

(Instruments, 2020)

Utilizando este dispositivo de NI podemos escribirle y leer en la memoria EEPROM de nuestro modulo todo un arreglo de valores, por lo que se puede mandar de una sola vez toda la tabla de identificación y de calibración del módulo, además de que podemos pre-grabar la memoria antes de instalarla en el módulo, además de que por medio de su interface SPI se puede comunicar el módulo con la computadora sin utilizar el cRIO.

Es necesario tener en cuenta que si se comunicara con la memoria directamente sin estar instalada en el módulo es necesario verificar las especificaciones técnicas de dicha memoria para garantizar que aguante la potencia del NI 845 además de seguir los comandos de comunicación que esta necesite para interactuar.

En caso de que se interactúe con ella ya instalada en el módulo es necesario activar o desactivar las señales de manera manual para que el módulo entienda que está en modo de identificación, eso se hace mandando a tierra los pines 1, 8 y el 14.

Para poder utilizar de manera correcta este dispositivo aquí está la tabla de asignación de señales digitales para el NI – 8451:

Module	Terminal	Signal	Module	Terminal	Signal
	1	GND		17	P0.0
	2	+5 V		18	P0.1
	3	SPI CS 7		19	P0.2
	4	SPI CS 6		20	P0.3
	5	SPI CS 5		21	P0.4
	6	NC		22	P0.5
	7	GND		23	P0.6
	8	GND		24	P0.7
	9	SPI CS 4		25	GND
	10	SPI CS 3		26	GND
	11	SPI CS 2		27	NC
	12	SPI CS 1		28	NC
	13	SPI CS 0		29	I²C SDA
	14	SPI MOSI (SDO)		30	I²C SCL
	15	SPI MISO (SDI)		31	+5 V
	16	SPI CLK (SCLK)		32	GND

Figura 40. *Tabla de asignación de señales digitales del NI 8451*

(Instruments, 2020)

El programa creado para grabar la memoria EEPROM del módulo IIUNAM-9901 del Instituto de Ingeniería usando el NI -8451 fue el siguiente programa:

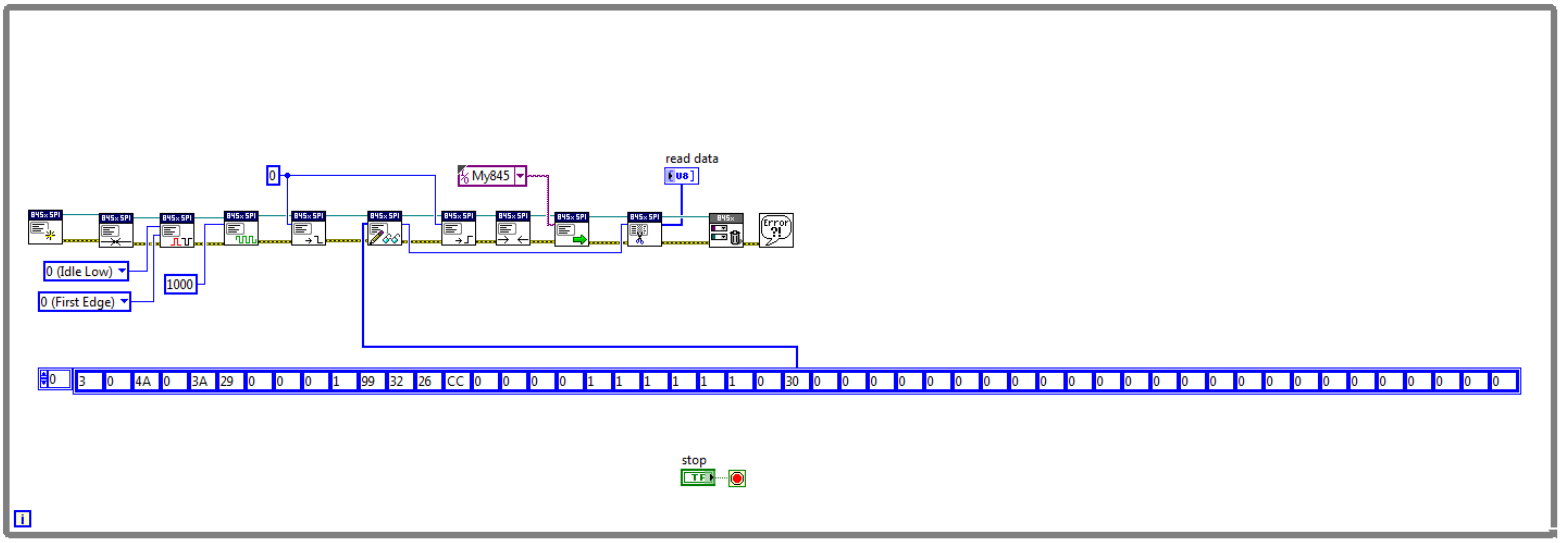


Figura 41. *Programa en LabVIEW del el NI-8451 para la escritura de datos en la EEPROM*

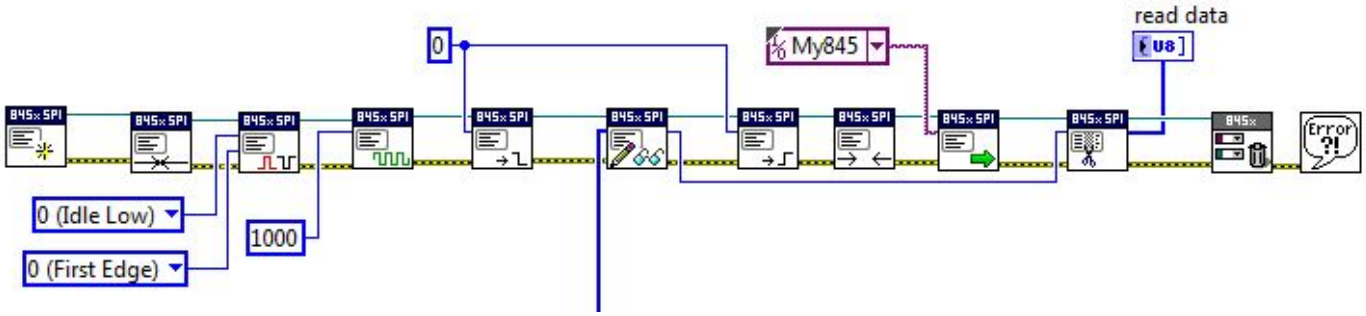


Figura 42. *Ampliación del programa en LabVIEW del NI-8451*

Es importante mencionar que para que funcione de manera correcta el primer byte del arreglo es el byte de comando, que dependiendo de la memoria EEPROM que se utiliza y de que operación se quiera realizar (Lectura, escritura, calibración, etc.) variara su valor.

4.3.2 Usando el MDK para la escritura en la EEPROM

Más adelante se detallará como emplear el MDK en LabVIEW pero para diseñar este programa se describirán algunos de sus bloques de comando con el fin de crear un programa que a través del cRIO se realice la escritura y lectura de la EEPROM del módulo para ello:

Change Mode VI

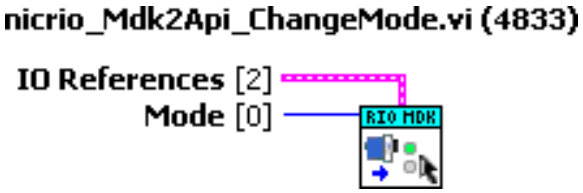


Figura 43. *Bloque de comando Change Mode VI*

(Instruments, 2020)

Este comando (Change Mode) nos permite cambiar de manera automática el estado de las líneas para cambiar a un modo de operación diferente en el módulo. Las opciones de cambio son las siguientes:

Tabla 7. *Modos de Operación del bloque de comando Change Mode VI*

Opción	**Modo de Operación**
0	Idle / Inactivo
1	ID / Identificación
2	Auxiliary Communication / Comunicación auxiliar
3	Normal Operation / Modo normal de operación

Para poder comunicarse con la EEPROM del módulo es necesario que se encuentre en modo de identificación (ID).

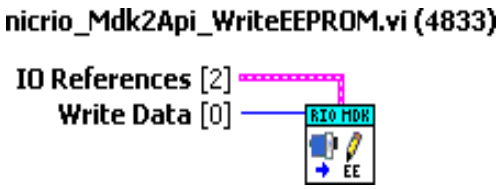


Figura 44. *Bloque de comando Write EEPROM VI*

(Instruments, 2020)

El siguiente comando es el (Write EEPROM VI) que como su nombre nos dice nos permite escribir información en la memoria byte por byte. Ahora para poder usar este comando también es necesario indicarle a la memoria en qué dirección va a guardar ese byte por lo que se usa el bloque de comando *Configuration Register VI*:

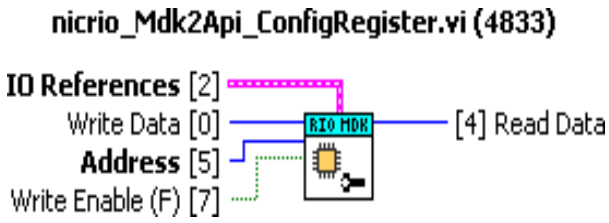


Figura 45. *Bloque de comando Configuration Register VI*

(Instruments, 2020)

Con el cual le indicamos la dirección de la memoria en la que se escribe el byte de información. Para poder leer un byte de la memoria EEPROM se hace lo mismo que en la escritura, pero se utiliza el bloque de comando *Read EEPROM VI*:

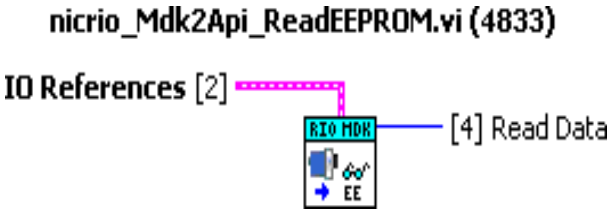


Figura 46. *Bloque de comando Read EEPROM VI*

(Instruments, 2020)

Una programación básica para escribir un byte en una dirección especificada de la memoria EEPROM del módulo seria la siguiente:

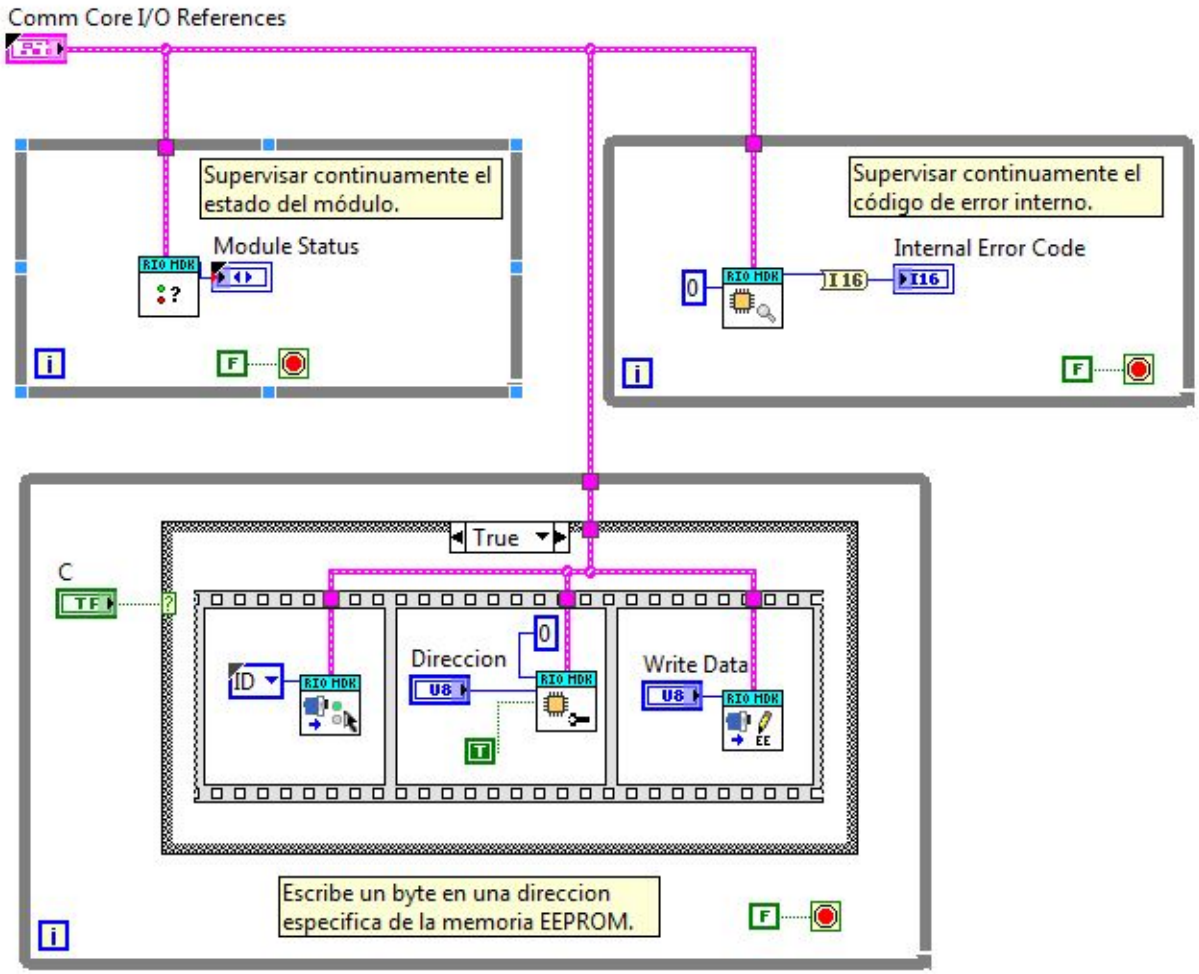


Figura 47. *Programa en LabVIEW para la escritura de la EEPROM*

Por otro lado, una programación básica para leer un byte en una dirección especificada la memoria EEPROM del módulo seria la siguiente:

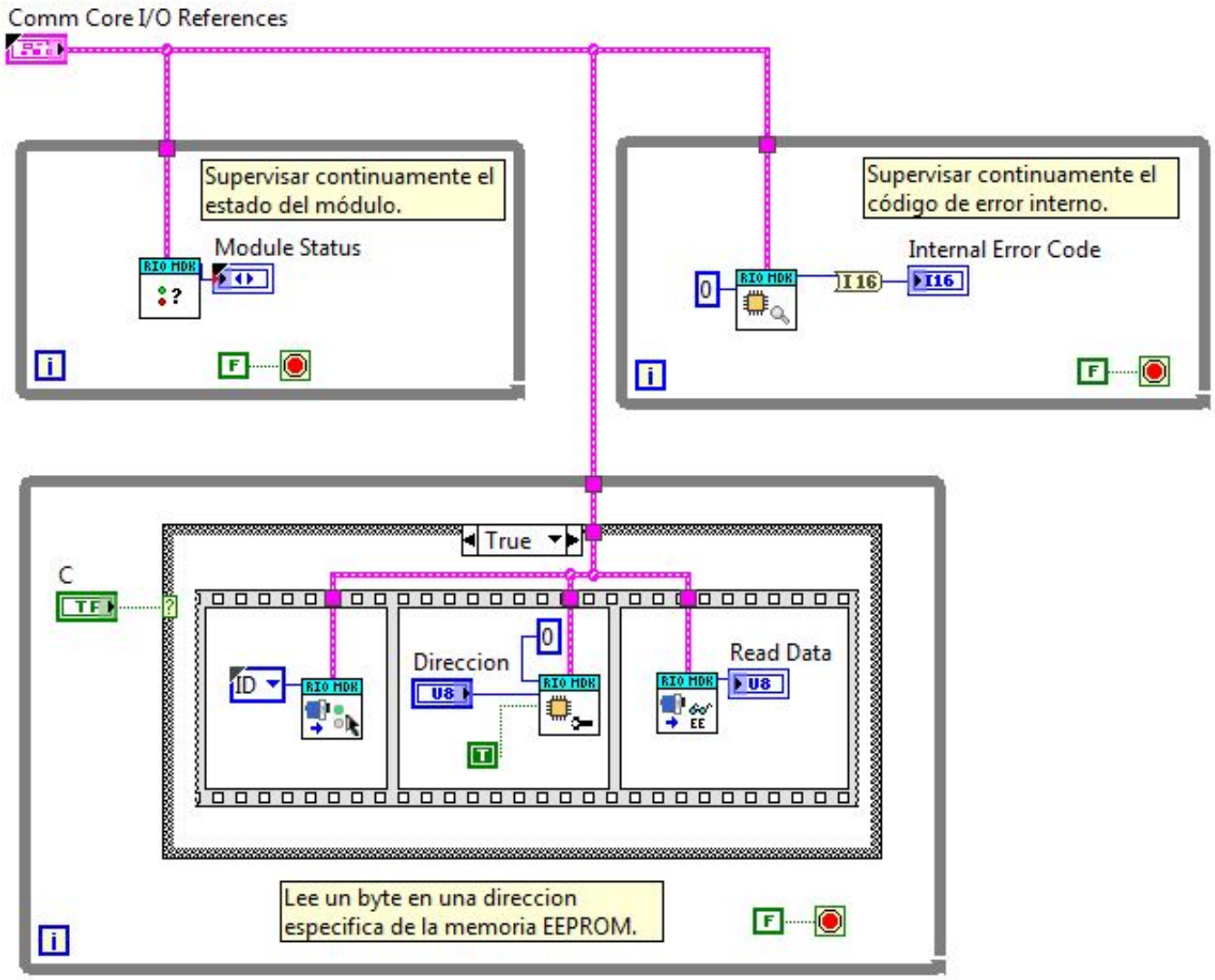


Figura 48. *Programa en LabVIEW para la lectura de la EEPROM*

Finalmente, el programa completo (Configuración EEPROM) en el FPGA para leer y escribir en la memoria del módulo C, byte por byte, es el siguiente:

Figura 49. *Interface gráfica del programa en LabVIEW para la escritura/lectura de la EEPROM*

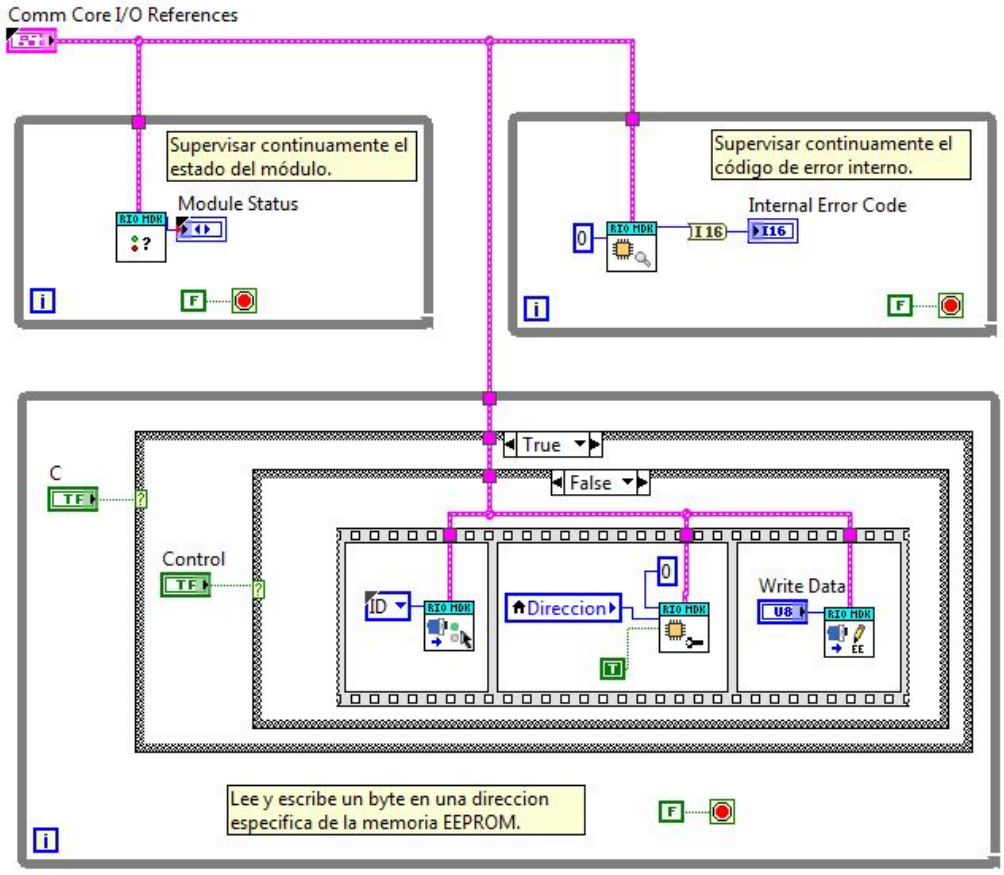


Figura 50. *Diagrama de bloques del programa en LabVIEW para la escritura/lectura de la EEPROM*

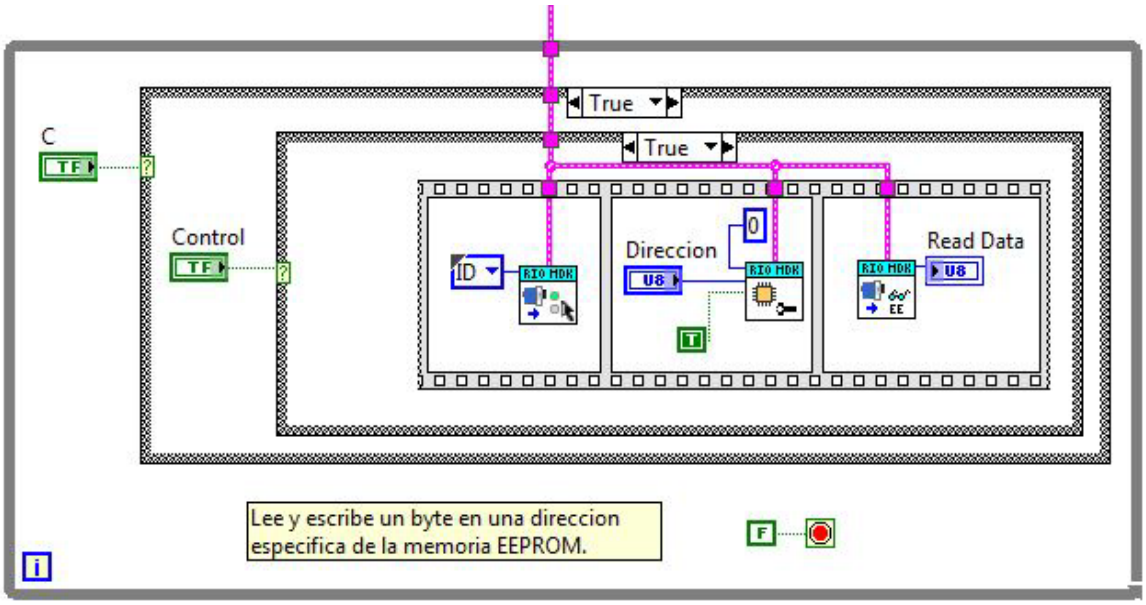


Figura 51. *Primer estado del programa en LabVIEW para la escritura/lectura de la EEPROM*

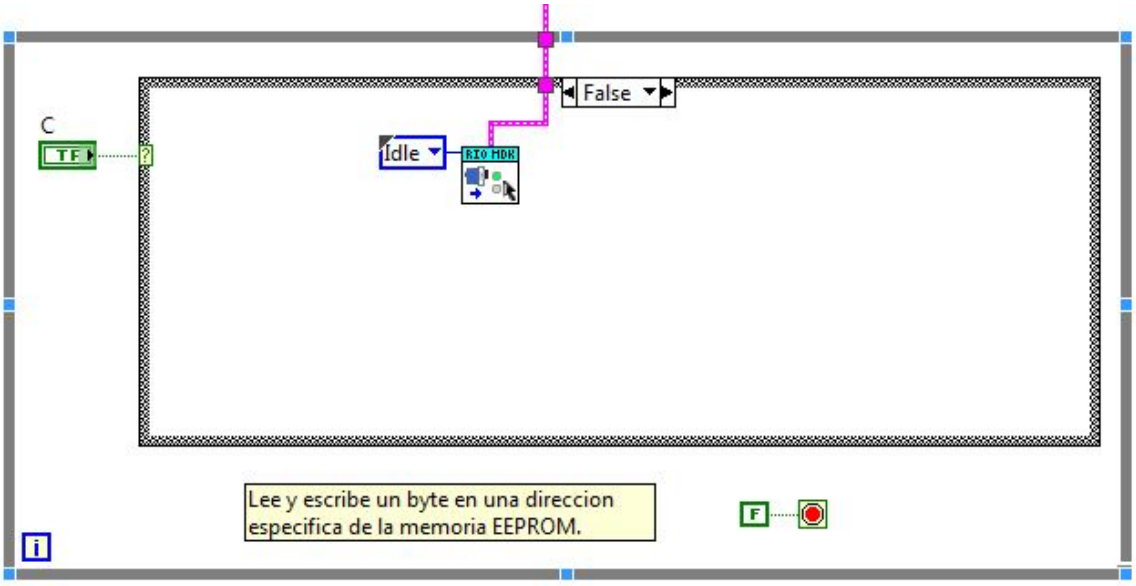


Figura 52. *Segundo estado del programa en LabVIEW para la escritura/lectura de la EEPROM*

Este programa es muy básico, pero pueden hacerse ligeros cambios para que en vez de enviar o recibir un byte, se envié de manera automática toda la tabla de identificación del módulo.

Una manera es emplear un ciclo For en dicho programa de tal manera que se envía de manera automática cada valor del arreglo uno por uno de manera muy rápida. Dicho programa se muestra en la siguiente imagen:

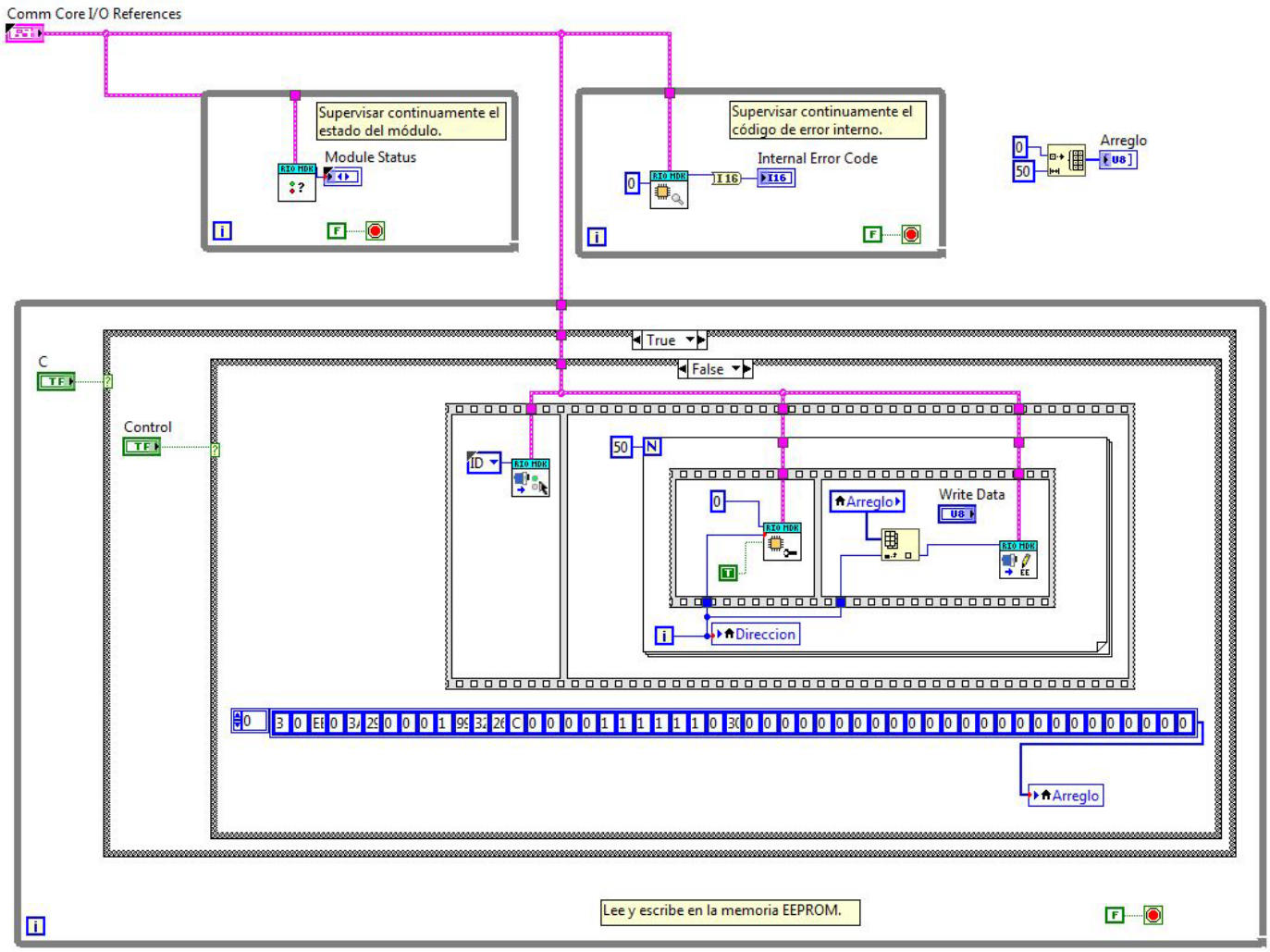


Figura 53. *Diagrama de Bloques Del VI – Configuración EEPROM*

4.3.3 Configuración de la memoria EEPROM con el VI de NI

Otra manera de leer y escribir la tabla de identificación y calibración en la memoria EEPROM de nuestro modulo es empleando una serie de VIs que vienen en el kit de desarrollo de módulos de NI que emplean un archivo Excel donde se escribe la información que se desea pasar a la memoria y otro Excel para leer los datos que tienen internamente almacenados la memoria.

Estos programas pueden encontrarse en la siguiente dirección:

C:\Documents\National Instruments\CompactRIO\CompactRIO MDK 2.1\Examples

En esta carpeta se busca la carpeta ExculdeFromExport ubicada en:

C:\Documents\NationalInstruments\CompactRIO\CompactRIOMDK2.1\Examples\Module Support Development\MDK-MFG\ExculdeFromExport

Aquí encontraras un proyecto en LabVIEW FPGA que usan como ejemplo para mostrarte como leer y escribir en una memoria EEPROM de un módulo C, lo que tienes que hacer para usarlos es crear tu propio proyecto de LabVIEW FPGA, como vimos anteriormente, utilizando el cRIO con el que trabajaras, y luego copiar, exportar o rehacer los siguientes programas:

- MDK-MFG Write EEPROM Contents to Module (Host).vi
- MDK-MFG Read EEPROM Contents from Module (Host).vi
- MDK-MFG Development Mode (FPGA).vi

Es importante mencionar que el VI Devolopment Mode debe estar ubicado en el FPGA de tu Proyecto, debe haberse cargado manualmente tu modulo, como ya vimos en algún slot del cRIO, además debe generarse un Excel, como viene en esta carpeta de ejemplo, pero con las datos de tu tabla de identificación y de calibración de tu modulo, además de crear un Excel más vacío, pero que cuente con el formato de Excel del ejemplo, para poder leer lo que

tiene la memoria, sin la necesidad de borrar tu Excel de identificación y calibración a cada momento.

Una vez que se tenga todo lo anterior hay que copilar el VI del FPGA en el cRIO, cuando se haya copilado el VI del FPGA en el cRIO, ya podemos leer o escribir en la memoria, para escribir los datos de identificación y calibración en la EEPROM abrimos el VI – Write EEPROM:

VI - MDK-MFG Write EEPROM Contents to Module (Host).vi

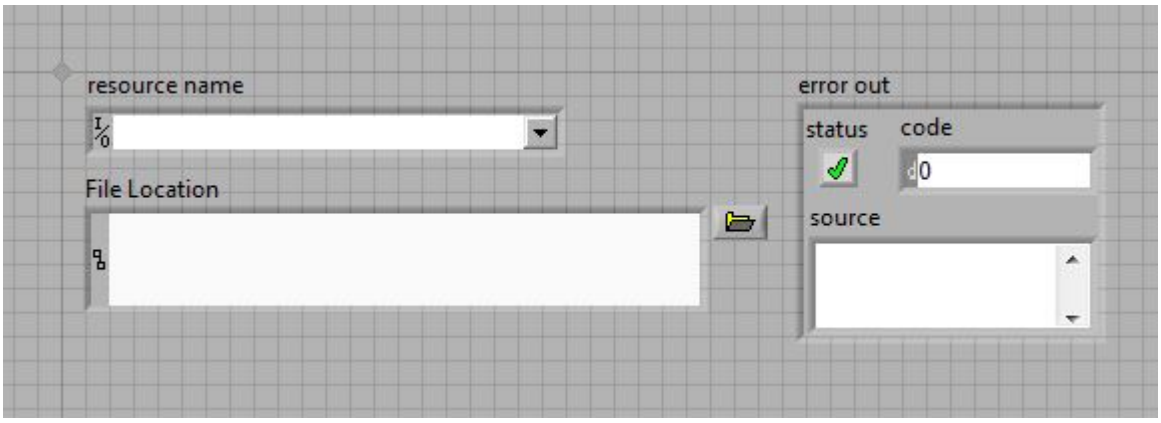


Figura 54. *Panel Frontal Del VI – Write*

(Instruments, 2020)

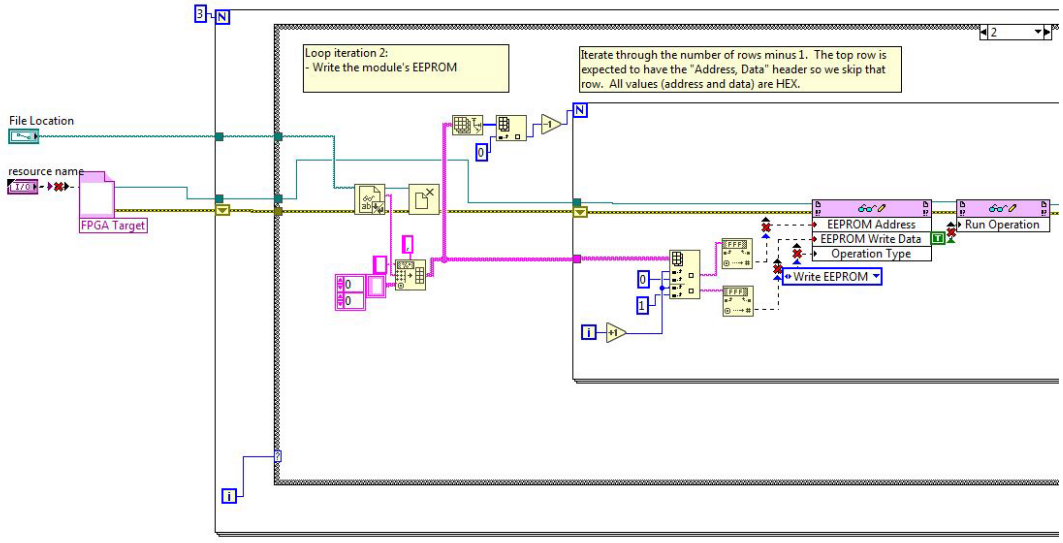


Figura 55. *Diagrama de Bloques Del VI – Write EEPROM con errores*

(Instruments, 2020)

Como se puede ver en la imagen anterior al solo copiar o rehacer el VI en un nuevo proyecto es necesario configurar la referencia del VI del FPGA, para que funcione de manera correcta, para ello damos clic derecho en el bloque de comando *OPEN FPGA VI REFERENCE* y seleccionamos la opción de configure *OPEN FPGA VI Reference*, como se muestra en la siguiente imagen:

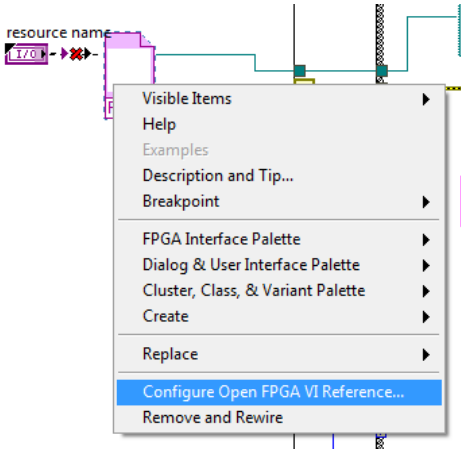


Figura 56. *Open FPGA VI Reference en LabVIEW*

(Instruments, 2020)

Esta opción nos abre la pestaña que se ve en la imagen siguiente (Figura 57), donde buscamos el VI del FPGA Development Mode que copilamos previamente, damos en ok y el programa ya no debe presentar problemas de compilación.

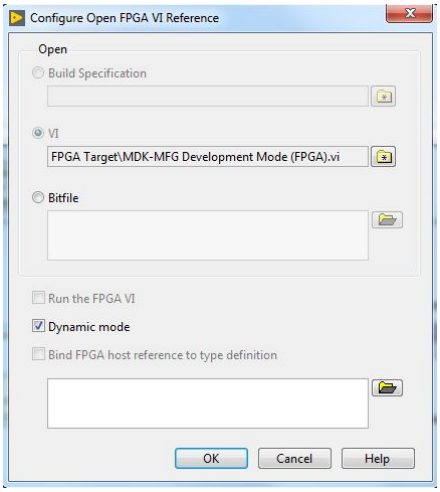


Figura 57. *Ventana emergente Configure Open FPGA VI Reference*

(Instruments, 2020)

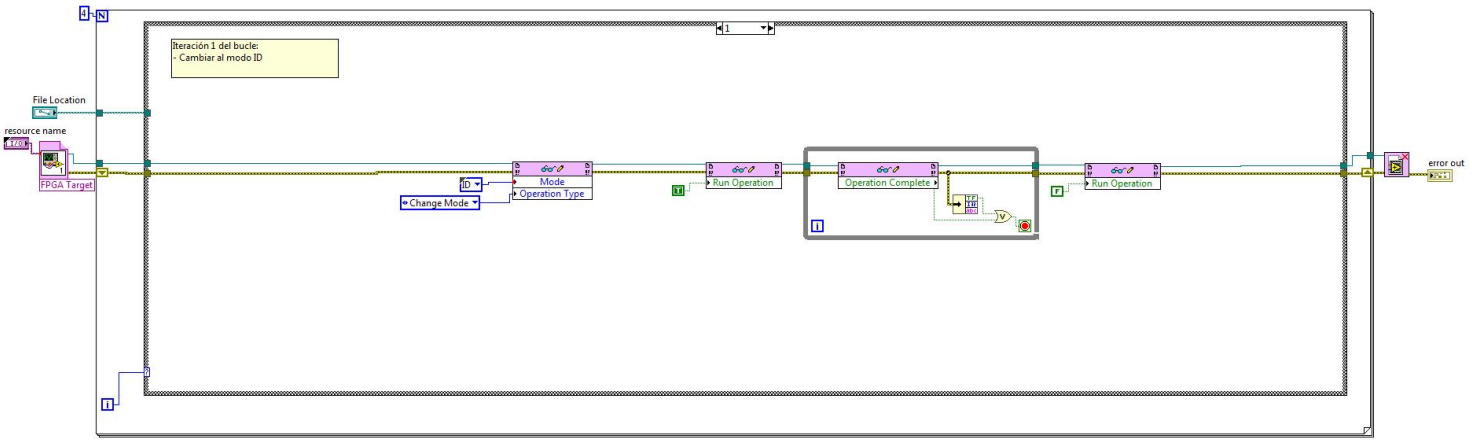


Figura 58. Diagrama de bloques del programa *Del VI – Write EEPROM – Parte 1*

(Instruments, 2020)

A este programa (VI) se le hicieron ligeros cambios para que se acoplara a la memoria EEPROM utilizada para el módulo IIU-9932, uno de ellos fue agregar un ciclo del For para que antes de escribir la tabla se pudiera mandar el código de activación de la escritura, como se muestra a continuación:

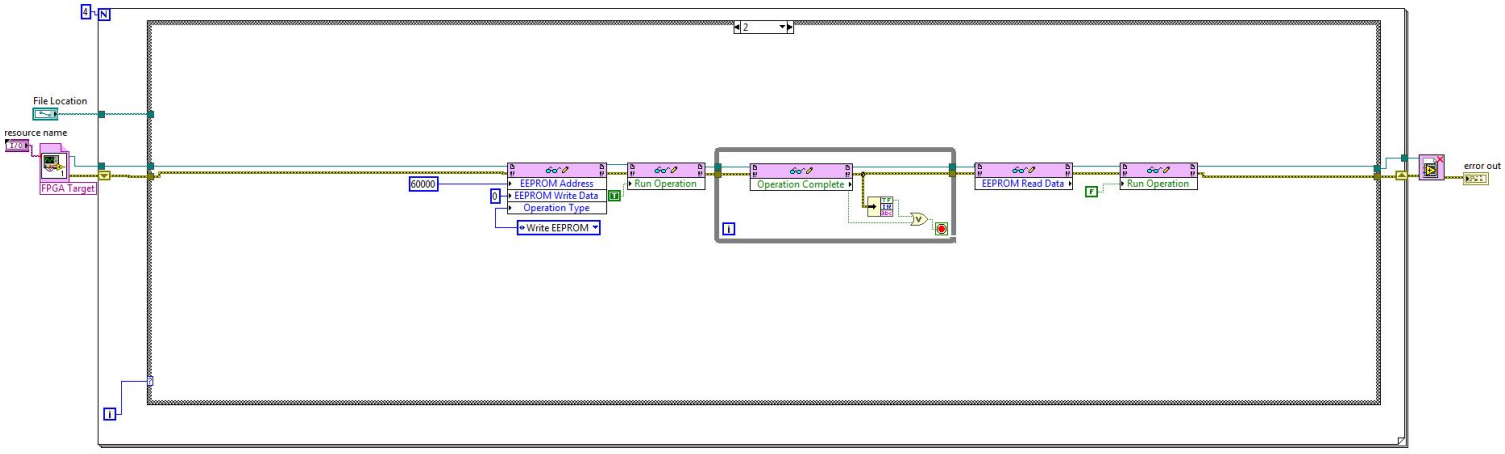


Figura 59. *Diagrama de Bloques Del VI – Write EEPROM – Parte 2*

De *National Instruments Corporation &* Del *Instituto de Ingeniería de la UNAM*

El siguiente cambio fue colocar indicadores para ver los datos que vienen del Excel y los datos que van a la memoria EEPROM.

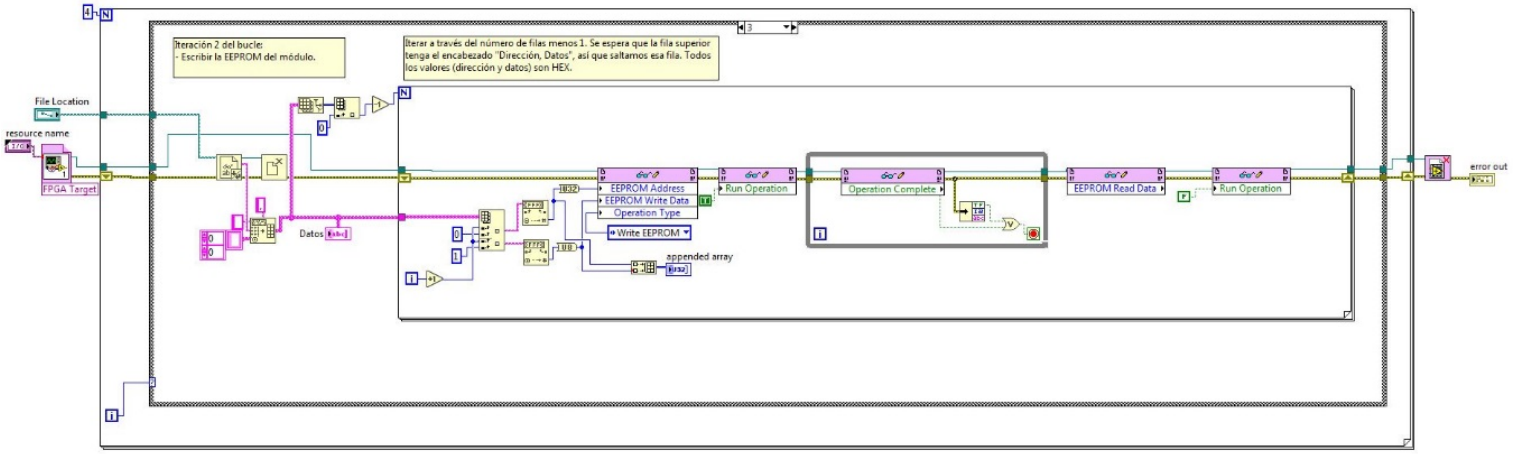


Figura 60. *Diagrama de Bloques Del VI – Write EEPROM – Parte 3*

De *National Instruments Corporation &* Del *Instituto de Ingeniería de la UNAM*

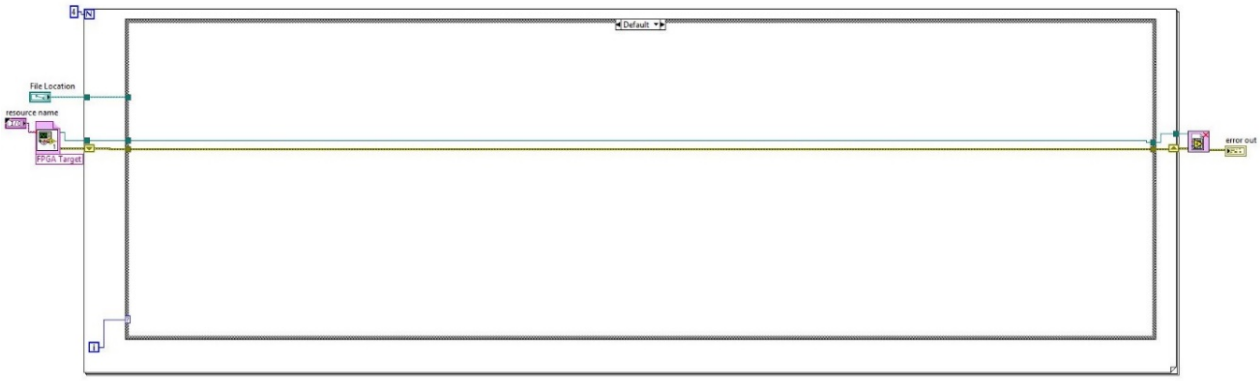


Figura 61. *Diagrama de Bloques Del VI – Write EEPROM – Parte 4*

De National Instruments Corporation & Del Instituto de Ingeniería de la UNAM

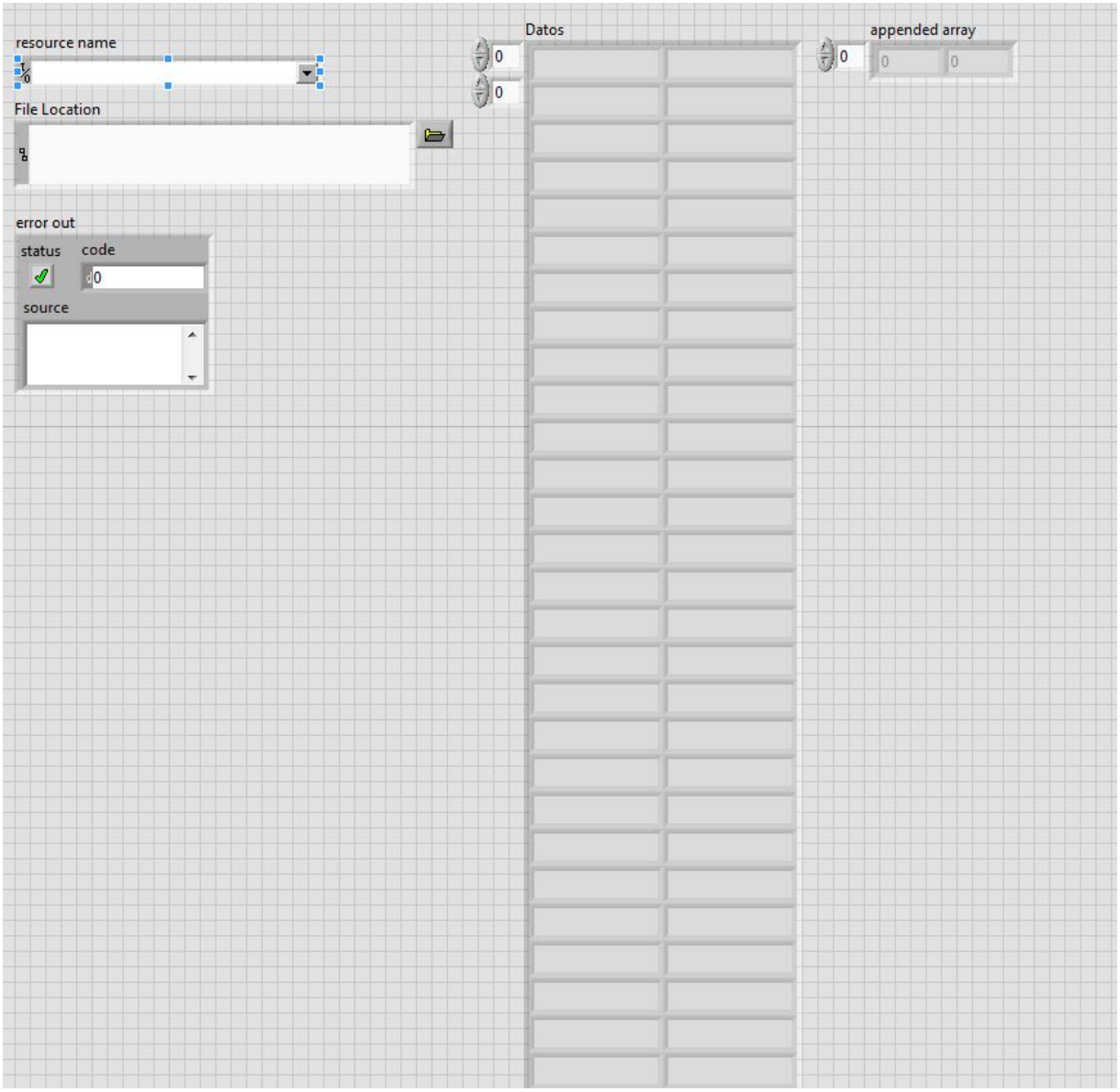


Figura 62. *Panel Frontal Del VI – Write EEPROM*

De National Instruments Corporation & Del Instituto de Ingeniería de la UNAM

Finalmente, para correr el VI y escribir en la memoria solo hay que seleccionar el cRIO que se está utilizando y el Excel cargado con la tabla de identificación y calibración como se muestra a continuación:

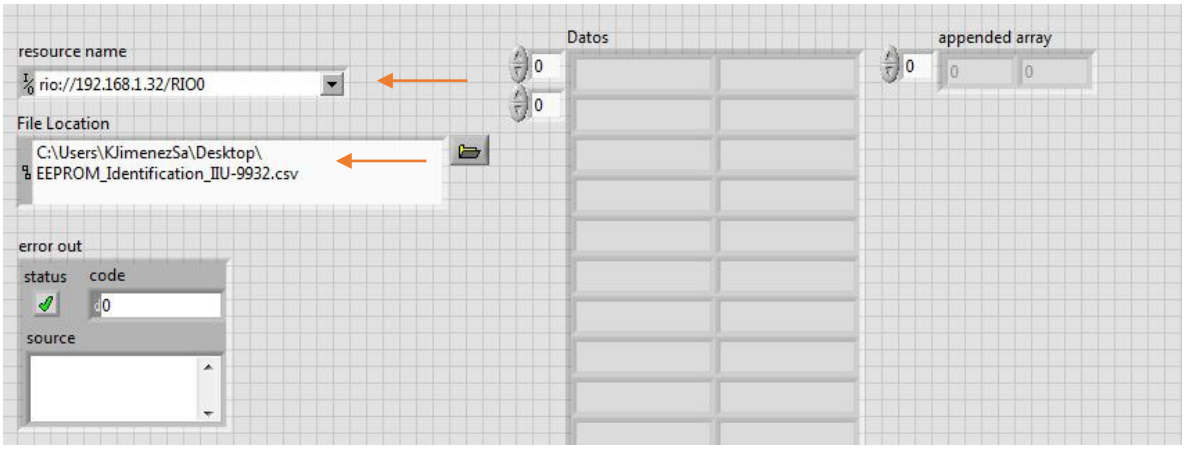
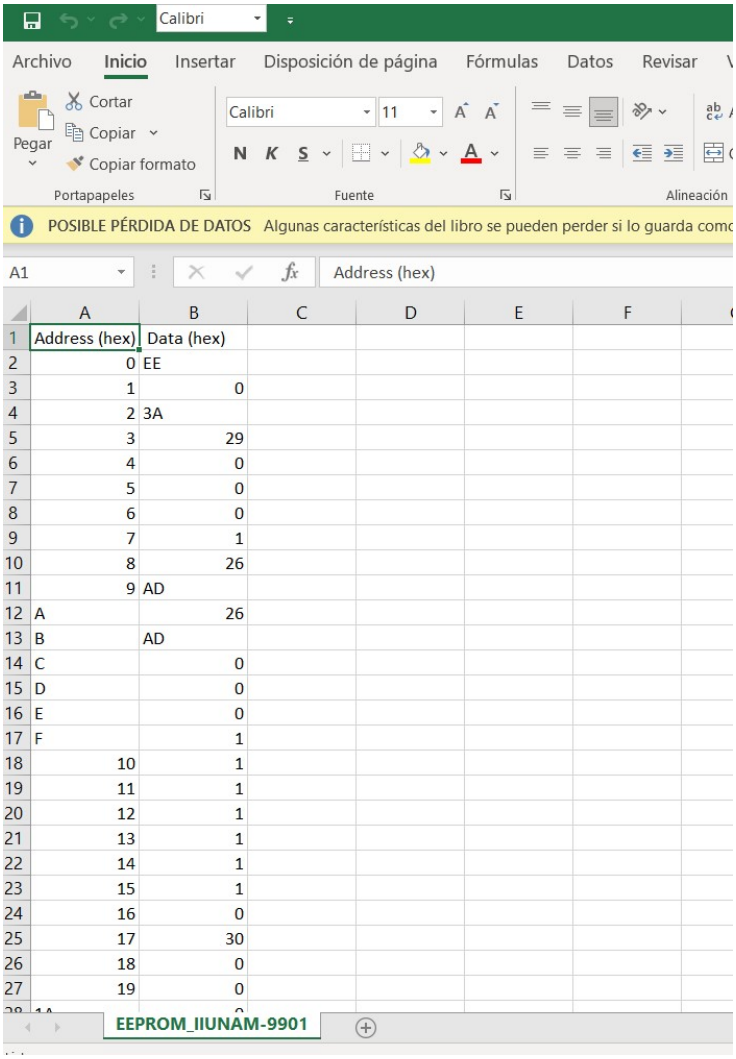


Figura 63. *Panel Frontal Del VI – Write EEPROM*

De National Instruments Corporation & Del Instituto de Ingeniería de la UNAM



	A	B	C	D	E	F
1	Address (hex)	Data (hex)				
2	0	EE				
3	1	0				
4	2	3A				
5	3	29				
6	4	0				
7	5	0				
8	6	0				
9	7	1				
10	8	26				
11	9	AD				
12	A	26				
13	B	AD				
14	C	0				
15	D	0				
16	E	0				
17	F	1				
18	10	1				
19	11	1				
20	12	1				
21	13	1				
22	14	1				
23	15	1				
24	16	0				
25	17	30				
26	18	0				
27	19	0				

Figura 64. *Excel con la tabla de identificación del módulo IIUNAM-9901*

VI - MDK-MFG Read EEPROM Contents from Module (Host).vi

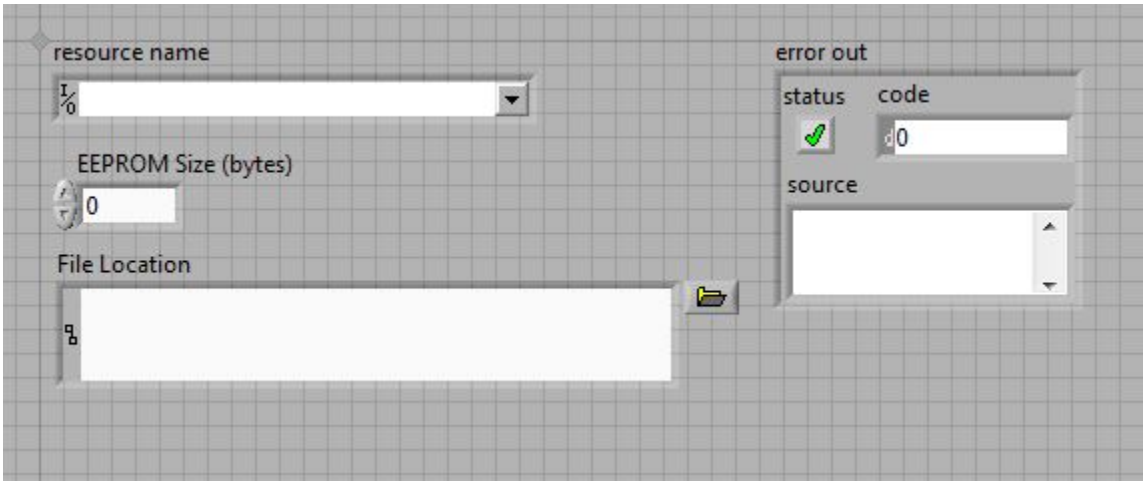


Figura 65. *Panel Frontal Del VI – Read EEPROM*

(Instruments, 2020)

Al igual que en el VI anterior hay que referenciar el VI del FPGA Development Mode, así que se repiten los mismos pasos anteriores para que no exista error de compilación.

En este programa el cambio que hay que hacer es agregar el código de lectura de la EEPROM, dicho cambio puede verse en el siguiente diagrama de bloques:

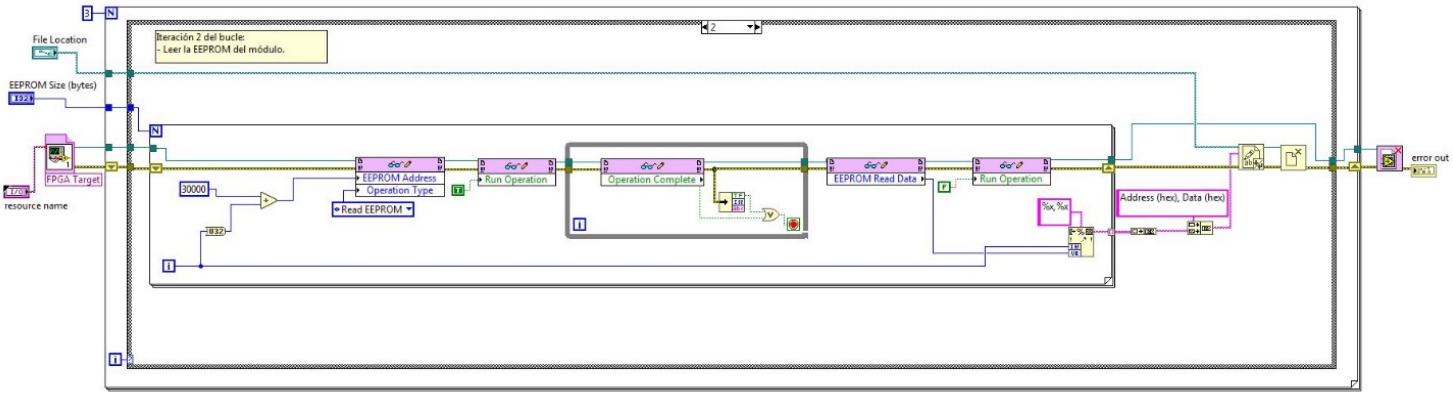


Figura 66. *Diagrama de Bloques Del VI – Read EEPROM*

De *National Instruments Corporation* & Del *Instituto de Ingeniería de la UNAM*

Finalmente, para correrlo y saber qué información tiene la memoria EEPROM almacenada, hay que seleccionar el cRIO con el que se está trabajando, el tamaño de datos (Bytes) que se quiere leer y el Excel vacío donde se van a almacenar dichos datos, y ejecutar el programa.

Figura 67. *Panel Frontal Del VI – Read EEPROM*

De *National Instruments Corporation &* Del *Instituto de Ingeniería de la UNAM*

Como se puede ver, estos VIs es una manera sencilla de escribir y leer datos en la memoria EEPROM de nuestro modulo, pero hay que tener cuidado con los comandos de lectura y escritura que maneja nuestra memoria, ya que cada proveedor programa sus memorias como más le convenga, variando el tipo de dato, el tamaño en bytes de dicho comando y la manera en la que se manda este valor a la memoria, para que pueda almacenar información y enviarla. Son cuestiones que hay que tener muy en cuenta para que funcionen estos programas de NI que ellos no mencionan y hay que modificar para su correcto funcionamiento.

Como dato importante puede darse el caso de que sea necesario configurar algunas cosas en el programa del FPGA dependiendo si la memoria EEPROM que se utilice es muy diferente a la que NI recomienda o si se va simular una EEPROM con el microcontrolador o microprocesador utilizado, ya que el proceso de comunicación entre los dispositivos varia ligeramente del mencionado en este trabajo.

Una vez que está cargada la información de identificación del módulo en la memoria EEPROM del módulo ya se puede desarrollar el software para que el módulo C realice las acciones por las que fue creado.

4.4 Software en LabVIEW para comunicación FPGA

Lo primero que se recomienda crear en todos los VIs que se diseñen en el FPGA para su módulo C es un programa que supervise continuamente si existe un error con el módulo y que determine en qué modo de operación se encuentra, para ello se debe crear un VI como el siguiente:

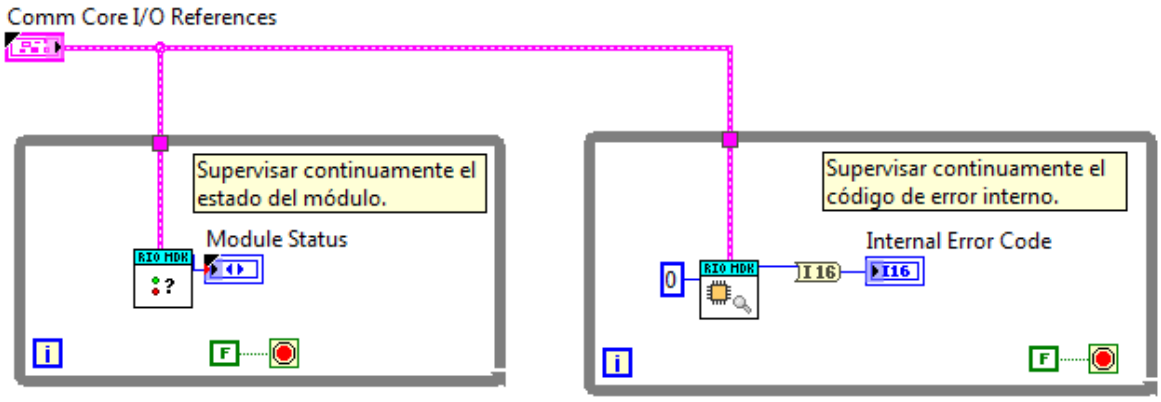


Figura 68. *Diagrama de Bloques del programa de supervisión del módulo C*

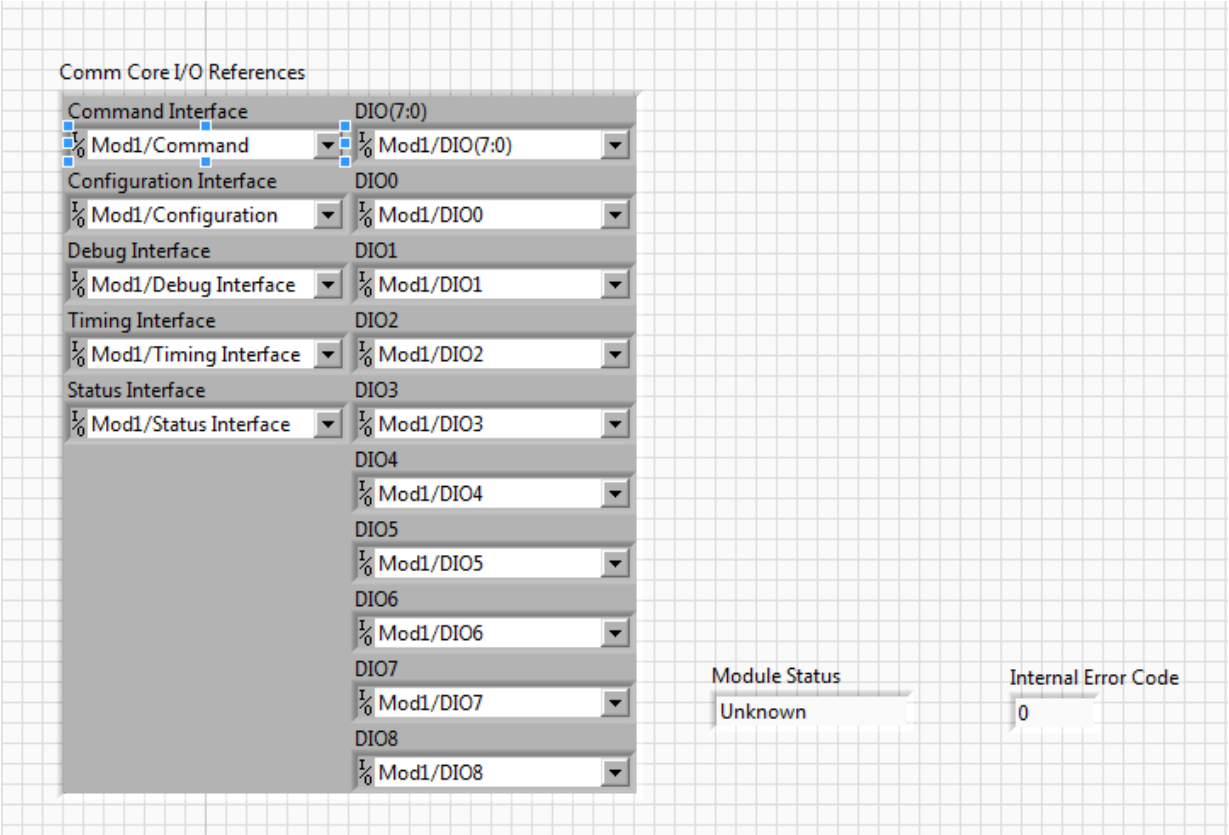


Figura 69. *Panel Frontal del programa de supervisión del módulo C*

Esta programación permite supervisar de manera constante el estado del módulo y detectar si se presentó un error a la hora de usarse.

Los estatus que pueden presentarse son los siguientes (*Tabla 8. Estado de un módulo C*):

Estatus	Descripción
Unknown (0)	El chasis se está encendiendo y la presencia del módulo aún no se ha determinado. El estado del módulo también cambia brevemente a desconocido durante un comando del módulo Identificar.
Correct (1)	El módulo se ha detectado como presente en el chasis y los contenidos de EEPROM coinciden con el identificador de proveedor esperado y el código de modelo del módulo.
Incorrect (2)	El módulo se ha detectado como presente en el chasis y los contenidos de EEPROM no coinciden con el identificador de proveedor esperado y el código de modelo del módulo.
No Module (3)	El módulo no ha sido detectado como presente en el chasis.
Invalid (4)	El módulo ha sido detectado como presente en el chasis y el centinela de inicio EEPROM no coincide con el valor esperado.

Por otro lado, los errores que pueden presentarse son los siguientes (*Tabla 9. Estado de un módulo C*):

Error	Código de Error	Descripción
None	0	No error.
Change Mode (invalid mode)	–1	Intentaste cambiar un modo que no es compatible.
Identify Module (not in ID mode)	–2	Intentó ejecutar la recomendación del módulo de identificación cuando no estaba en modo ID.
EEPROM Write (incorrect module)	–3	Intentó ejecutar un comando de escritura EEPROM cuando el módulo no era correcto.
SPI (not started)	–4	La transferencia SPI no se ha iniciado.
SPI (already started)	–5	Intentó iniciar una transferencia SPI cuando ya se había iniciado una.
SPI (not in correct mode)	–6	Intentó iniciar una transferencia SPI cuando estaba en un modo que no es compatible con SPI.
Invalid Command	–7	Intentaste ejecutar un tipo de comando inválido.
Change Mode and Output Enable	–8	Intentó ejecutar un comando de cambio de modo al ejecutar un método de habilitación de salida para DIO. Intentó ejecutar un tipo de comando no válido.
Invalid Configuration Register	–9	Intentó acceder a un registro no válido (o leyó / escribió en un registro que no era compatible con lectura / escritura).
Invalid Debug Register	–10	Intentó acceder a un registro de depuración no válido (o leyó / escribió en un registro que no admite lectura / escritura).
Pulse Convert (not in correct mode)	–11	Intentó pulsar ~ CONVERTIR en un modo que no admite el impulso ~ CONVERTIR.
Wait on Done (not in correct mode)	–12	Intentó esperar a que se haga en un modo que no admite esperar a que se haga.
EEPROM Read or Write (not in ID mode)	–13	Intentó una lectura / escritura de EEPROM cuando no estaba en modo ID.
EEPROM Write Timeout	14	Tiempo de espera de escritura de EEPROM.
SPI Divide Rate (accessed during SPI transfer)	–15	Se intentó escribir el registro de anulación de la tasa de división SPI durante una transferencia SPI.

Capítulo 5. Programa para el control del módulo C

Para que el módulo C se empleara de la manera que se diseñó se crearon muchos programas diferentes para que el módulo funcionara de la manera más óptima posible. Por el momento ya se han creado los dos archivos XML, el primero con nombre ModuleType que permite, junto con la información almacenada en la memoria EEPROM, identificar al módulo y tener la información para la calibración del mismo; el segundo llamado ModuleSupport que permite configurar los modos y formas de comunicación del módulo con el cRIO.

Después se crearon diversos programas en LabVIEW para poder grabar los datos de identificación y calibración del módulo en la memoria EEPROM al conectar el módulo al compact, si la necesidad de usar equipos especializados y generalmente costosos para grabar memorias, antes de montarse en el módulo.

Figura 70. *Programa en LabVIEW para la escritura y lectura de datos de una memoria EEPROM conectada a un Módulo C*

Se requirió también crear un programa en el microcontrolador que permitiera controlar todos los componentes eléctricos y que a la vez procesara las señales provenientes del

exterior, para convertirlas en información digital que contuviera las variables provenientes de cada sensor (Eje. Voltaje, frecuencia, temperatura, etc.). Dicho programa para este trabajo no será considerado ya que su elaboración fue hecha por otro compañero de la coordinación de electrónica.

Para llegar a emplear el módulo C en la forma que se planeó para el usuario final fue necesario desarrollar un proyecto en modo desarrollo con una serie de VI que permitieran tomar la información del microcontrolador del módulo y transformarla en una serie de canales que nos proporcionan la información requerida en el cRIO o en la PC (según fuera el caso).

Finalmente se detallaran los dos programas en LabVIEW que interactúan entre ellos para lograr el procesamiento de información del módulo, el primero para el FPGA del cRIO donde se obtienen las variables generadas en el módulo de todos los dispositivos conectados a él, a través del protocolo de comunicación SPI, y el segundo es un VI que puede estar en el compact o en una PC que se emplea para visualizar, almacenar y procesar las variables obtenidas del módulo en un lenguaje más técnico y especializado para dar parte de la información necesaria sobre la situación en la que se encuentra una zona estructural.

5.1 Comunicación SPI

Para trasmitir información proveniente del módulo al cRIO o a la computadora de trabajo se decidió emplear la comunicación SPI para recibir y enviar bytes, para ello hay que seguir el siguiente diagrama básico para la transmisión por SPI en el FPGA.

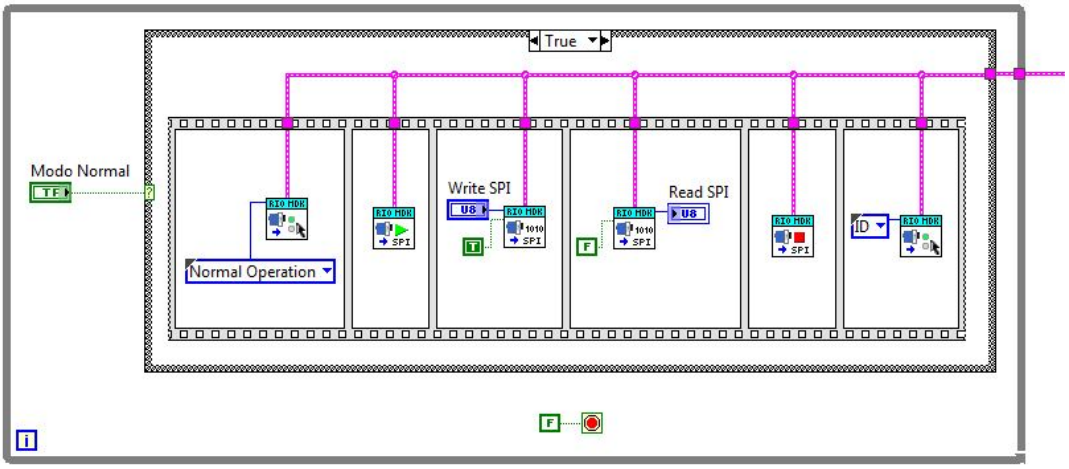


Figura 71. *Diagrama de Bloques para la comunicación SIP*

Para que se pueda dar la comunicación SPI hay que seleccionar por qué modo de operación se realizara dicha transacción de datos, ya sea por el Modo Normal o por el Modo Auxiliar, y ver que la comunicación SPI está habilitada en el XML de Module Support para dichos modos.

Después hay que construir una estructura en secuencia que siga los siguientes pasos:

1. Cambiar al módulo del modo de ID o Idle al modo donde se realizará la comunicación SPI (Modo normal o modo auxiliar). Este cambio se hace a través del bloque de comando Change Mode.

2. Empezar la comunicación SPI a través del bloque de comando SPI Start.

Figura 72. *SPI Start*

(Instruments, 2020)

3. Trasmitir y recibir los bytes de información. Para realizar este proceso se emplea el bloque de comando SPI Byte:

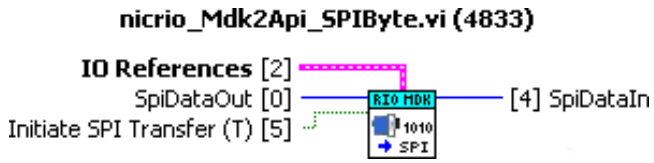


Figura 73. *SPI Byte*

(Instruments, 2020)

Este bloque permite enviar y recibir un byte por medio del SPI, pero hay que tener en cuenta ciertos detalles: como el hecho de que para comenzar una comunicación hay que activar la señal del SPI Transfer, también que este bloque de comando es canalizado, lo que significa que que la primera vez que se llama a este comando, los datos de escritura (n) se apagan en el bus SPI y los ceros se devuelven en el terminal de lectura de datos. En otras palabras, para enviar y recibir un byte hay que colocar dos bloques de comando SPI Byte donde el primero se activa y permite enviar un Byte al módulo, mientras que el segundo se desactiva y es el que recibe el byte que respondió el módulo.

Para enviar más de un byte hay que seguir estos detalles, pero en cadena, ejemplo para dos bytes:

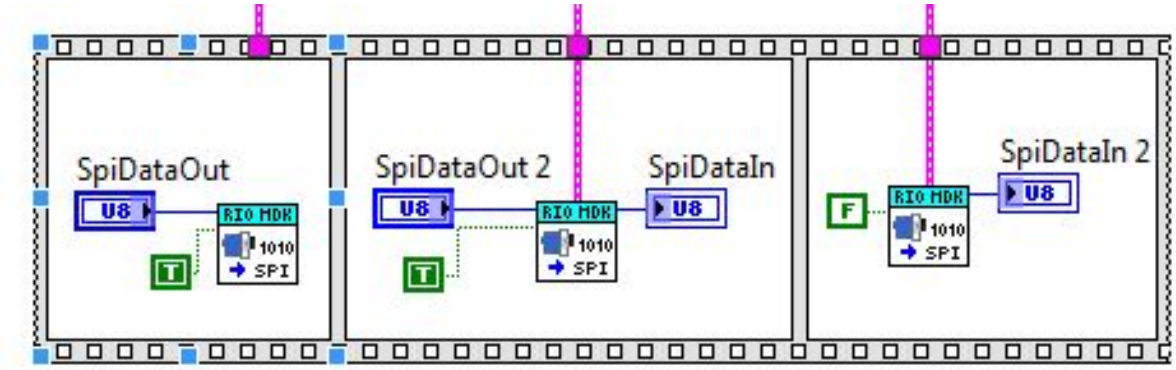


Figura 74. *SPI Byte para dos bytes*

Para tres Bytes:

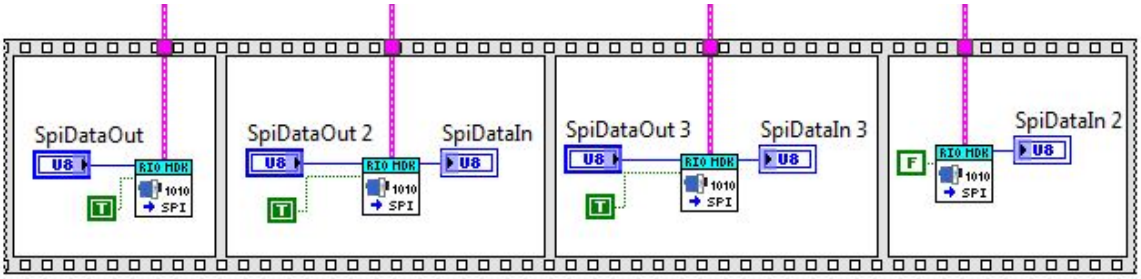


Figura 75. *SPI Byte para dos bytes*

Y así consecutivamente.

4. Después de enviar y recibir los bytes que se necesitaron hay que parar la comunicación SPI por medio del bloque de comando SPI Stop.

Figura 76. *SPI Stop*

(Instruments, 2020)

5. Cambiar el modo a ID o Idle.

6. Comenzar de nuevo todos los pasos para una nueva comunicación.

Es importante mencionar que se pueden enviar datos constantes por medio de la comunicación SPI, así como se puede enviar solo un byte y recibir el número de bytes que se quieran sin la necesidad de trasmitir más bytes, lo mismo a la inversa se puede enviar muchos Bytes sin recibir ningún byte del módulo como respuesta, y que los bytes que se envían o se reciben pueden almacenarse en un arreglo para almacenar o trabajar de mejor manera con estos datos en el cRIO o en un VI en la computadora.

5.2 Compilación de un VI en el FPGA

Para correr un programa en el FPGA de un cRIO hay que primero copilarlo para que el FPGA se programe para realizar las funciones que el VI le solicita y pueda realizar estos procesos de manera muy rápida y precisa.

Realizar este proceso es sencillo pero tardado, por lo que se recomienda primero revisar de manera razonable y analítica su VI para estar seguros de que no tiene errores y que realiza el proceso que quieres para tu modulo. Una vez que se estés seguro se corre el programa, donde aparece la siguiente ventana:

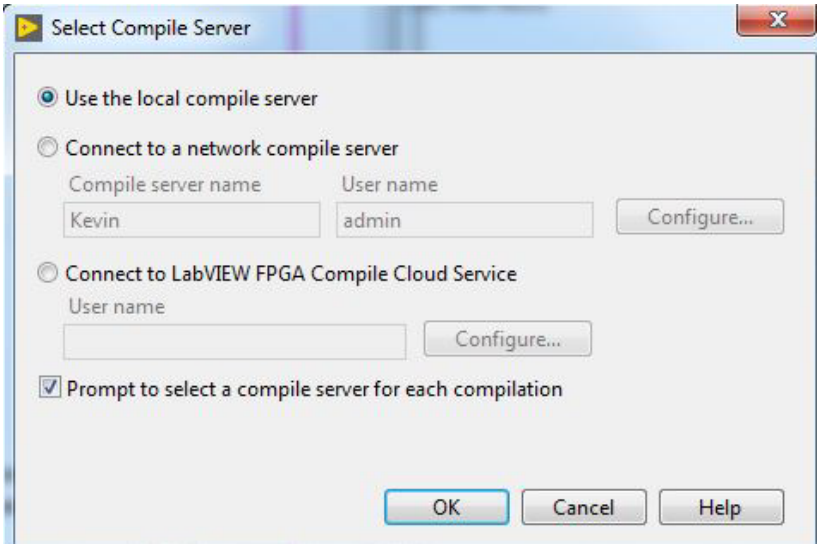


Figura 77. *Select Compile Server*

(Instruments, 2020)

Seleccionamos la opción de *local compile server*, y damos en *OK*. Es importante que el cRIO esté conectado al proyecto de LabVIEW y que el módulo se encuentre en el Slot seleccionado de manera manual. Después aparece una ventana de carga que verifica que el VI este bien programado, que todos los componentes estén presentes y que genera datos intermedios entre el cRIO y el FPGA.

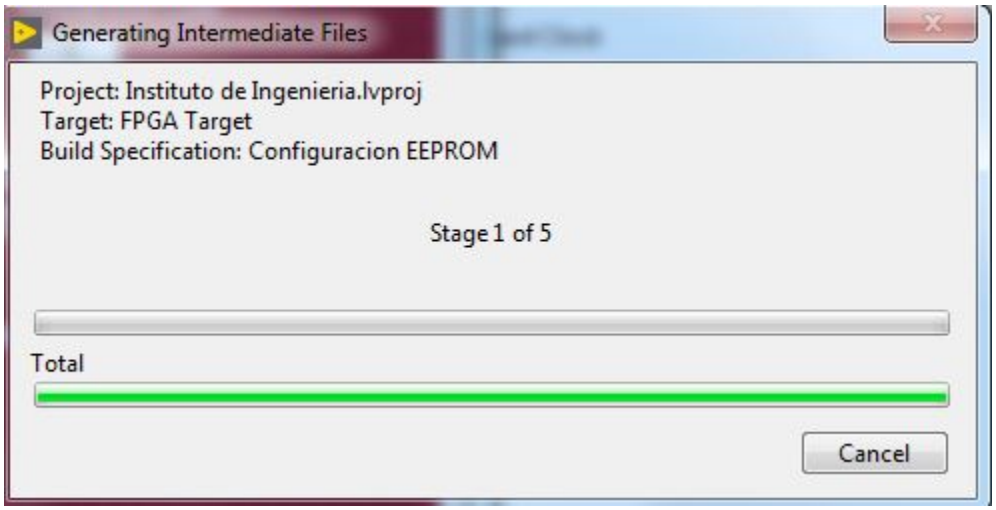


Figura 78. *Generating Intermediate Files*

(Instruments, 2020)

Si todo fue correcto abrirá el compilador Xilinx 14.7, sino abrirá una ventana donde mostrará los errores a corregirse para poder compilar el programa en el FPGA.

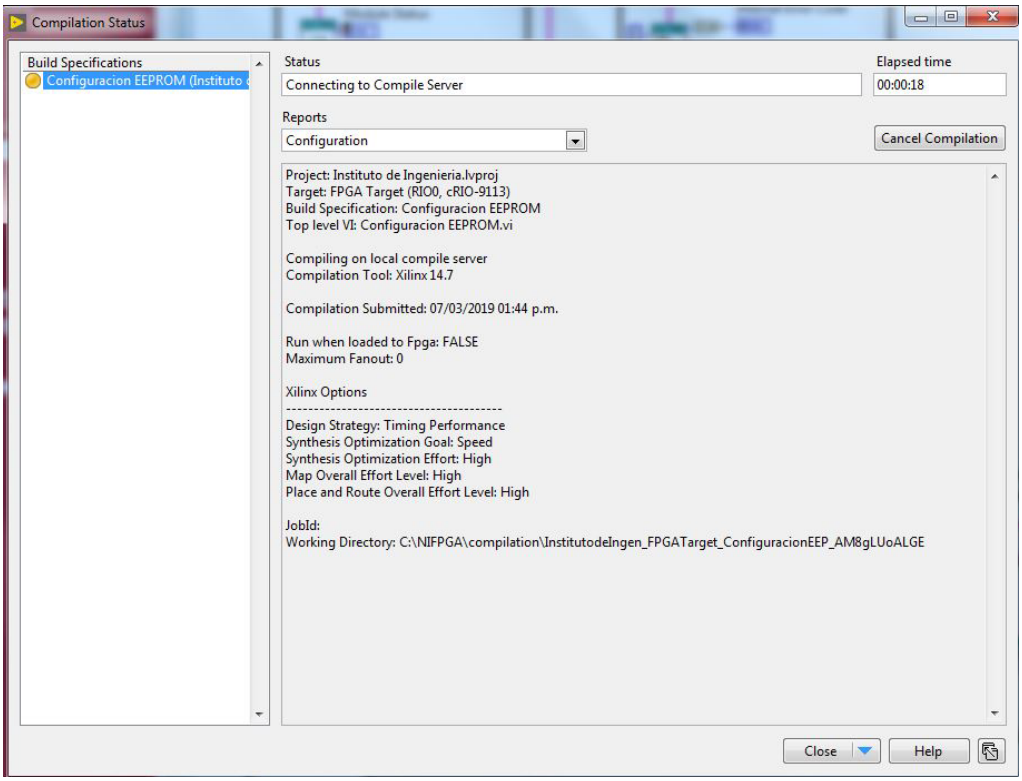


Figura 79. *Imagen del proceso de compilación en el Xilinx 14.7*

(Instruments, 2020)

El proceso es tardado (entre unos 20 a 30 minutos por cada vez que se copila) y pasa por muchos subprocesos, pero una vez que termina se puede cerrar el compilador y el VI debe aparece en estado de *run* para poder realizar el proceso programado. Mientras no se realicen cambios en el VI del FPGA con solo correr el programa este debería funcionar sin la necesidad de volverse a copilar, pero con el más mínimo cambio se tendrá que volver a pasar por todo este proceso y volver a copilar todo.

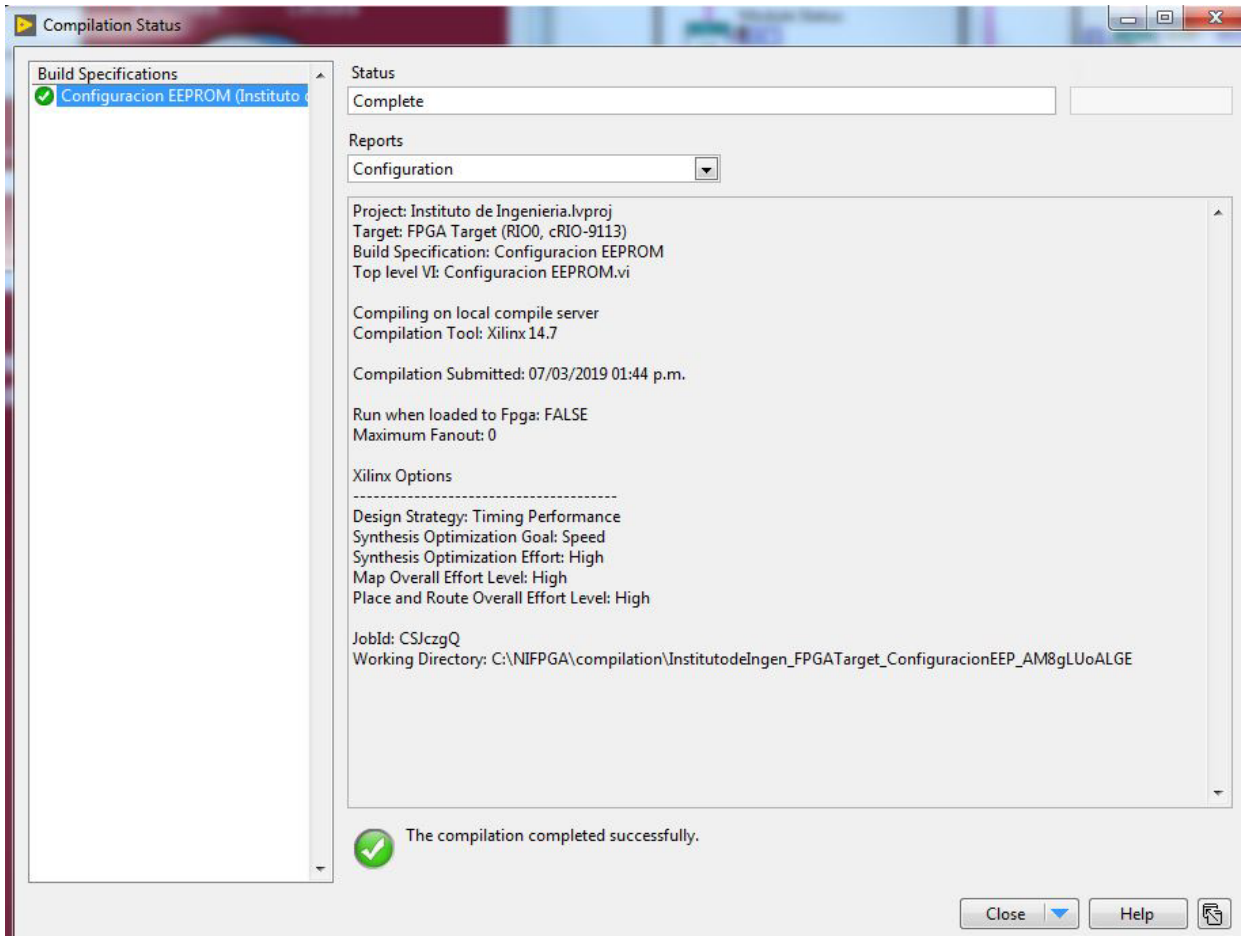


Figura 80. *Imagen de finalización correcta de la compilación en el Xilinx 14.7*

(Instruments, 2020)

El programa en el FPGA debe ser muy específico y con el único objetivo de enviar o recibir datos al o del módulo, ya que la intención es que este proceso se realice de manera muy rápida, eficiente y simple.

5.3 Módulo C en modo de desarrollo

Como ya se mencionó anteriormente el "modo de desarrollo" de un módulo C nos permite crear los VIs que en conjunto con lo programado en el XML de soporte permiten el adecuado funcionamiento del módulo dejando así al usuario final solo los canales de comunicación necesarios para realizar su función.

Debido a la complejidad, así como a la inmensa cantidad de programas creados y a la gran cantidad de información a tratar para que todo funcione correctamente en este modo, teniendo como objetivo llegar a su "modo lanzamiento" de manera correcta, solo detallaré los aspectos más importantes y daré un ejemplo para cada caso necesario, dejando claro que el proceso de desarrollo en general es el mismo solo cambia las variables y especificaciones que necesitemos para cada canal o proceso que realiza nuestro modulo C.

5.3.1 Datos y VI´s para el modo de desarrollo

Como se sabe LabVIEW maneja diversos tipos de datos de manera general como los enteros o los booleanos, pero para situaciones particulares se requiere que nuestro tipo de dato en algún canal sea una combinación de estos o simplemente que tenga una estructura determinada por nosotros diferente a la que LabVIEW tiene por defecto, en nuestro proyecto un ejemplo es el tipo de dato en el canal interno IniciarOperacion que definimos en el XML como IIUNAM-9901_TipoDeOperacion.ctl, claramente se puede apreciar que este tipos de dato tiene un formato diferente al convencional que necesitamos para que nuestro modulo funcione como lo desarrollamos.

Lo primero a resaltar es que estos datos diferentes tienen que ser de tipo .ctl que significa que son del tipo LabVIEW Control, para tener nuestra variable o dato en este formato solo hay que crear la variable en un VI como la necesitemos, en nuestro caso es un variable tipo Text Ring que cuenta con 5 opciones siendo estos los modos de Operación, propiedades, default, SPI y Auxiliar; después damos clic derecho sobre nuestra variable seleccionamos la opción de *Advanced* y luego en *Customize.* En la ventana que nos abre solo hay que

seleccionar que nuestra variable es de tipo control y guardarla en la carpeta de nuestro proyecto.

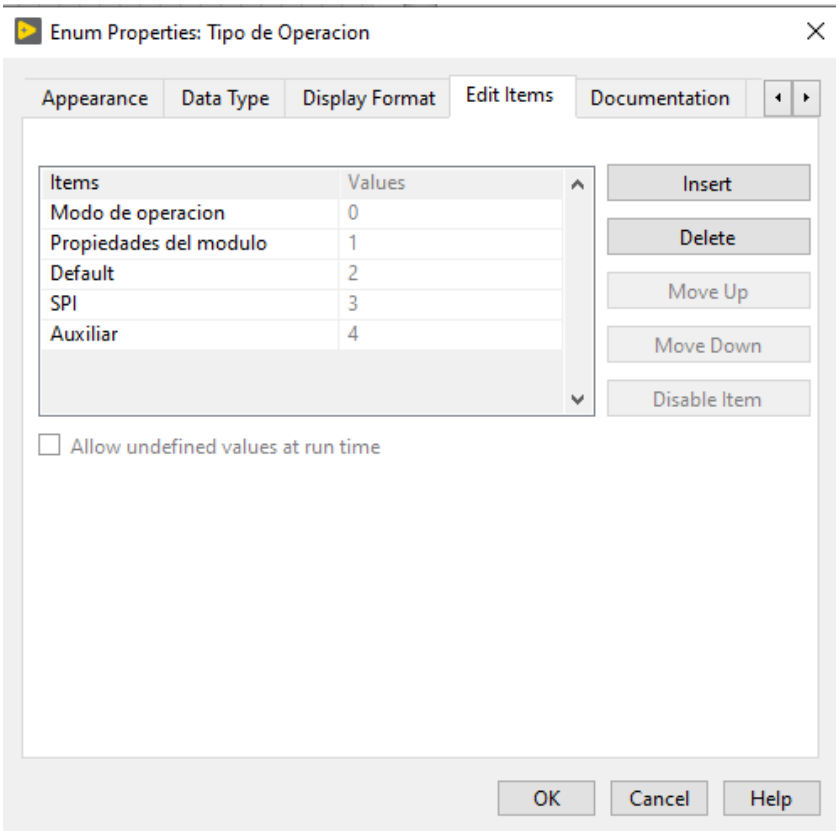


Figura 81. Propiedades de la variable IIUNAM-9901_TipoDeOperacion.ctl de nuestro modulo C.

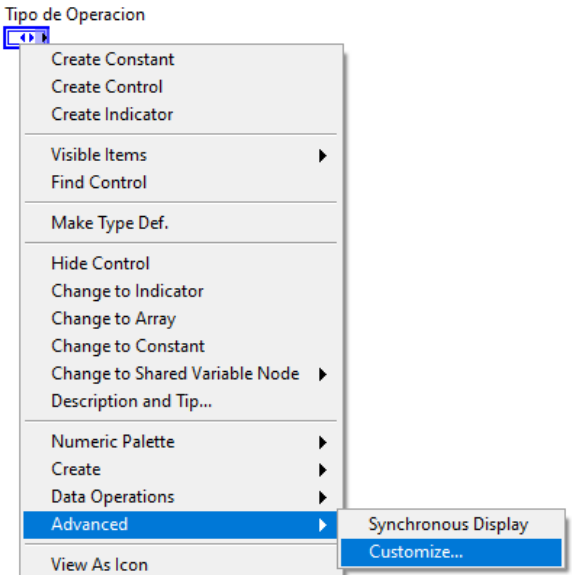


Figura 82. Creación de un tipo de variable LabVIEW Control.

Repetimos estos sencillos pasos para todas las variables necesarias para nuestro modulo C recalcando el hecho de que si el tipo de variable declarada en nuestro XML de soporte no es de un tipo común de LabVIEW o tipo control es muy seguro que al verificar nuestro "modo desarrollo" el programa que realiza el proceso nos marque un error y nos pida identificar esta variable de una manera específica.

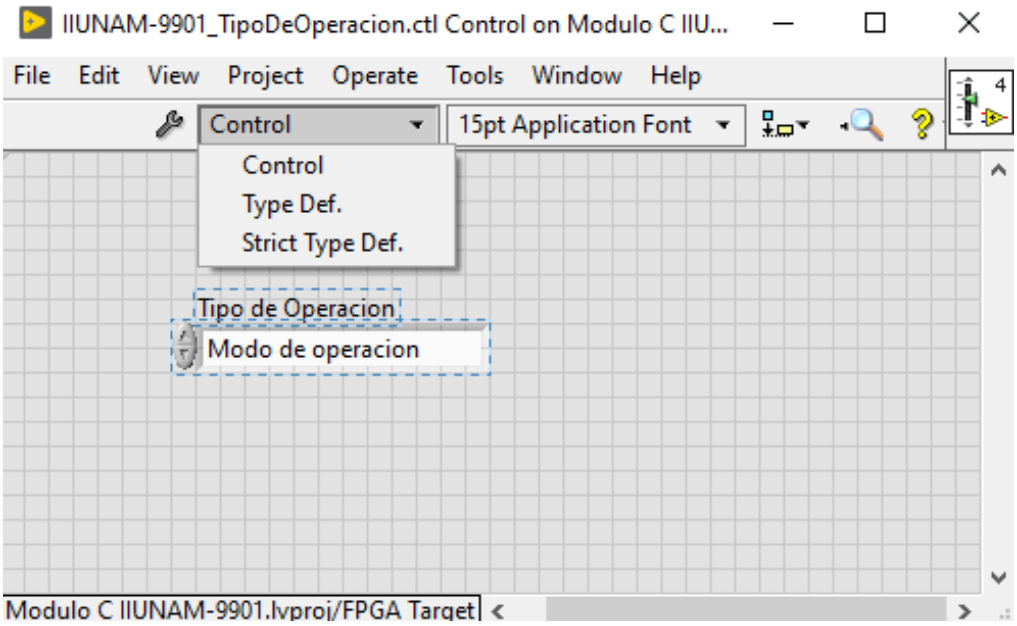


Figura 83. Variable IIUNAM-9901_TipoDeOperacion.ctl.

Otro aspecto que posee un formato especifico son los VI que empleamos en alguna interface en el archivo XML de soporte, como ejemplo tomaremos el VI que nombramos como IIUNAM-9901_ID_Vendedor.vi necesario en la interface de nombre ID Vendedor.

Como primer aspecto necesario para ser avalado adecuadamente es que este VI debe de tener un formato de terminales predeterminado por NI este formato puede verse en la siguiente imagen:

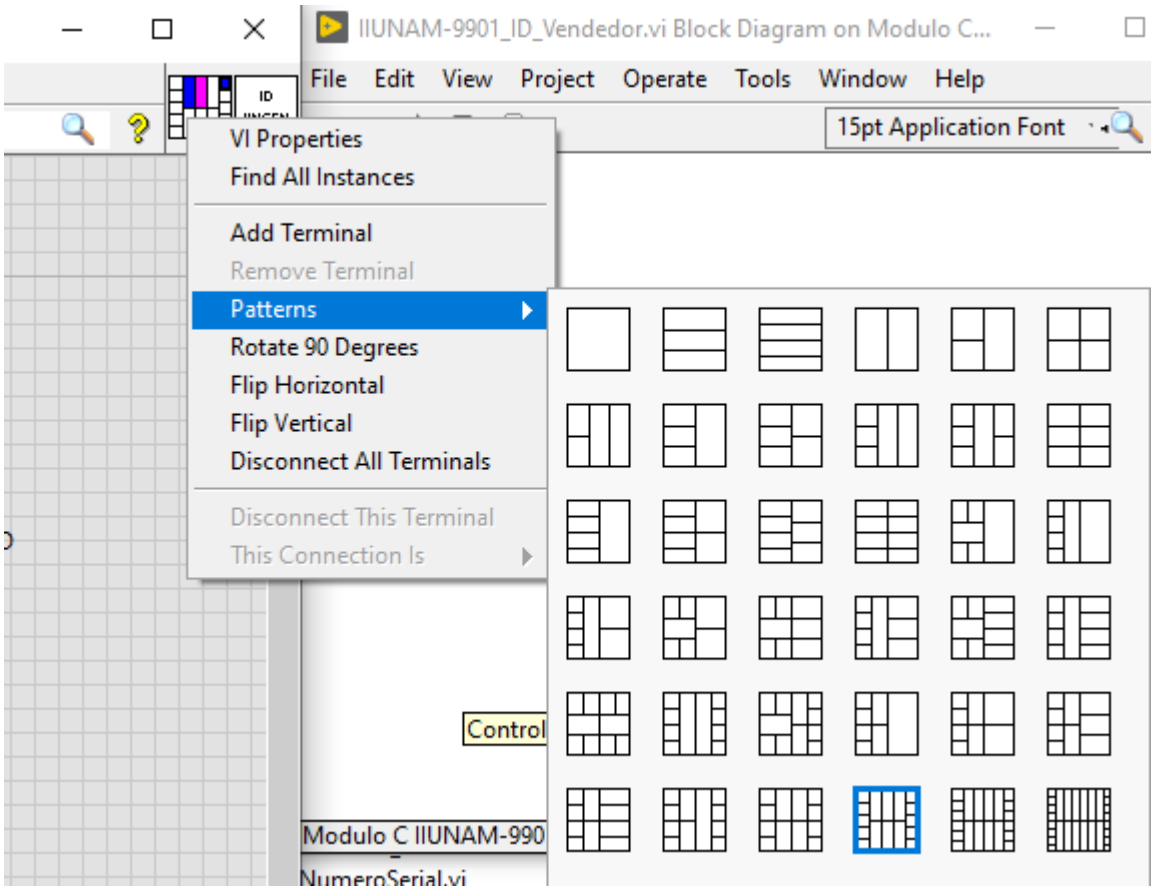


Figura 84. Formato del panel de terminales para un VI en modo de desarrollo.

Como se ve para seleccionarlo solo basta con dar clic derecho sobre el panel de terminales seleccionar la opción *Patterns* y luego seleccionar el formato predeterminado.

Es necesario que en dicho panel se respete la regla de que en la parte central del panel siempre se ubique la terminal de la variable de I/O References de nuestro modulo (IIUNAM-9901 I/O Reference) recordando el hecho de que esta variable fue creada a la hora de pasar nuestro proyecto a "modo desarrollo" por lo que es importante, aunque no necesario, que en todos los VI que sean requeridos en el archivo XML de soporte tengan esta misma referencia de entradas y salidas.

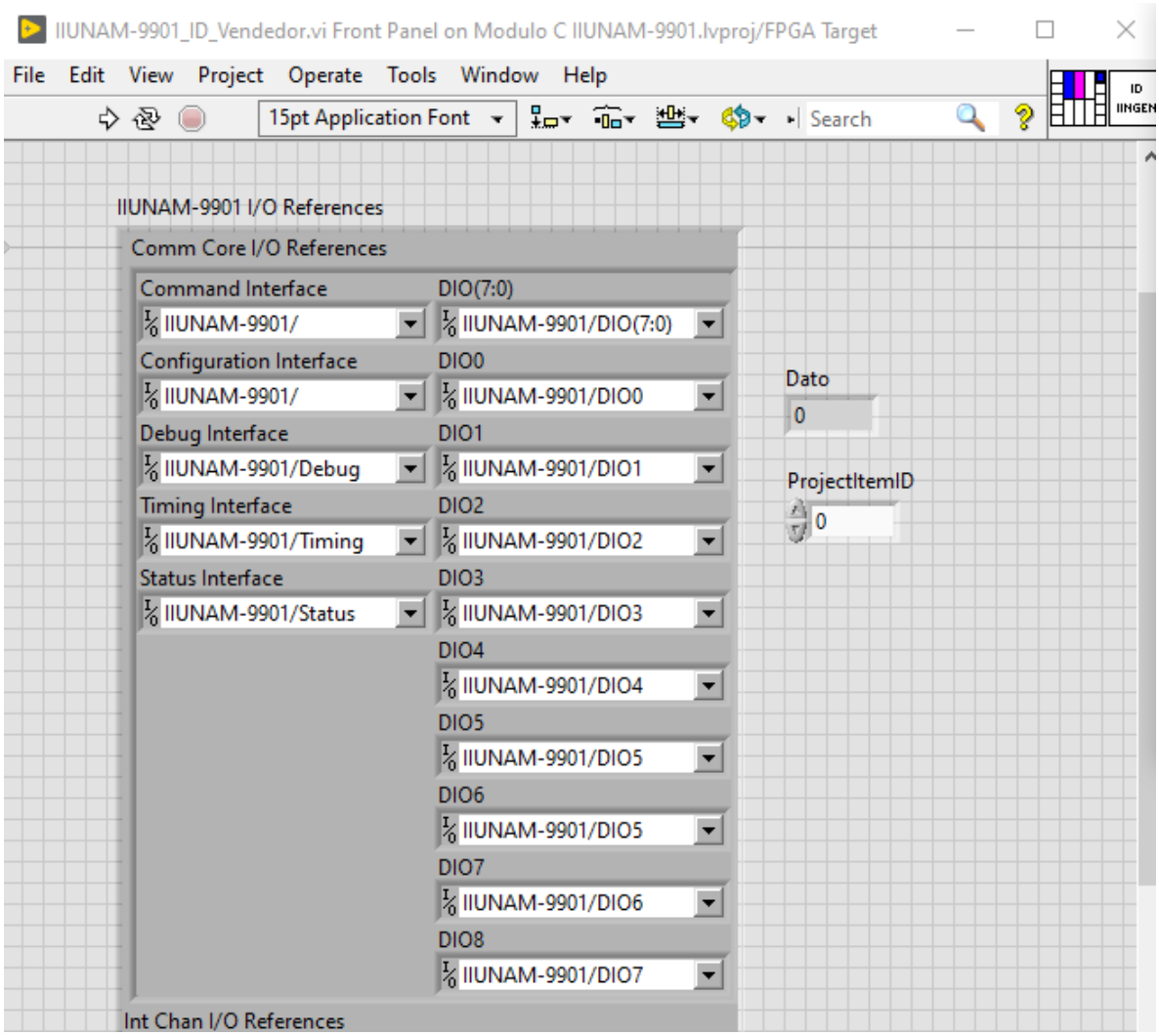


Figura 85. Panel del VI IIUNAM-9901_ID_Vendedor.

Por otro lado, como regla también para este panel es que nuestras terminales de entrada se encuentren de lado izquierdo mientras que las de salida se encuentren de lado derecho, con orden de prioridad de arriba hacia abajo.

El programa se muestra en la siguiente imagen y como se puede apreciar es relativamente simple pero debido a la condición de que cada proceso de lectura o escritura requiere de un sub VI que permita a NI realizar y entender su proceso, es necesario que todos tengan este formado de terminales y de variables.

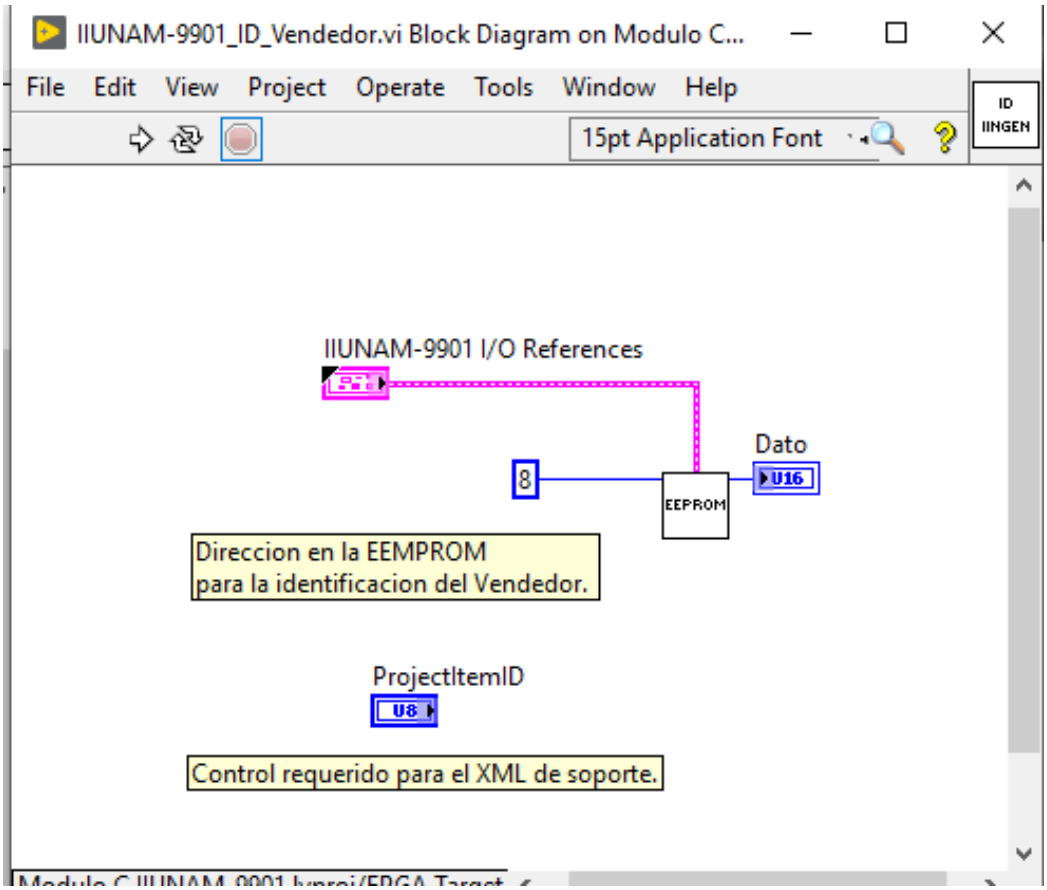


Figura 86. Diagrama de bloques del VI IIUNAM-9901_ID_Vendedor.

Con el objetivo de enmarcar las reglas del formato de los VI mostrare en las siguientes imágenes el VI IIUNAM-9901_Erro.vi que posee la misma estructura fundamental del anterior, pero muestra el proceso a realizar en caso de encontrar un error en los lugares donde este VI es llamado.

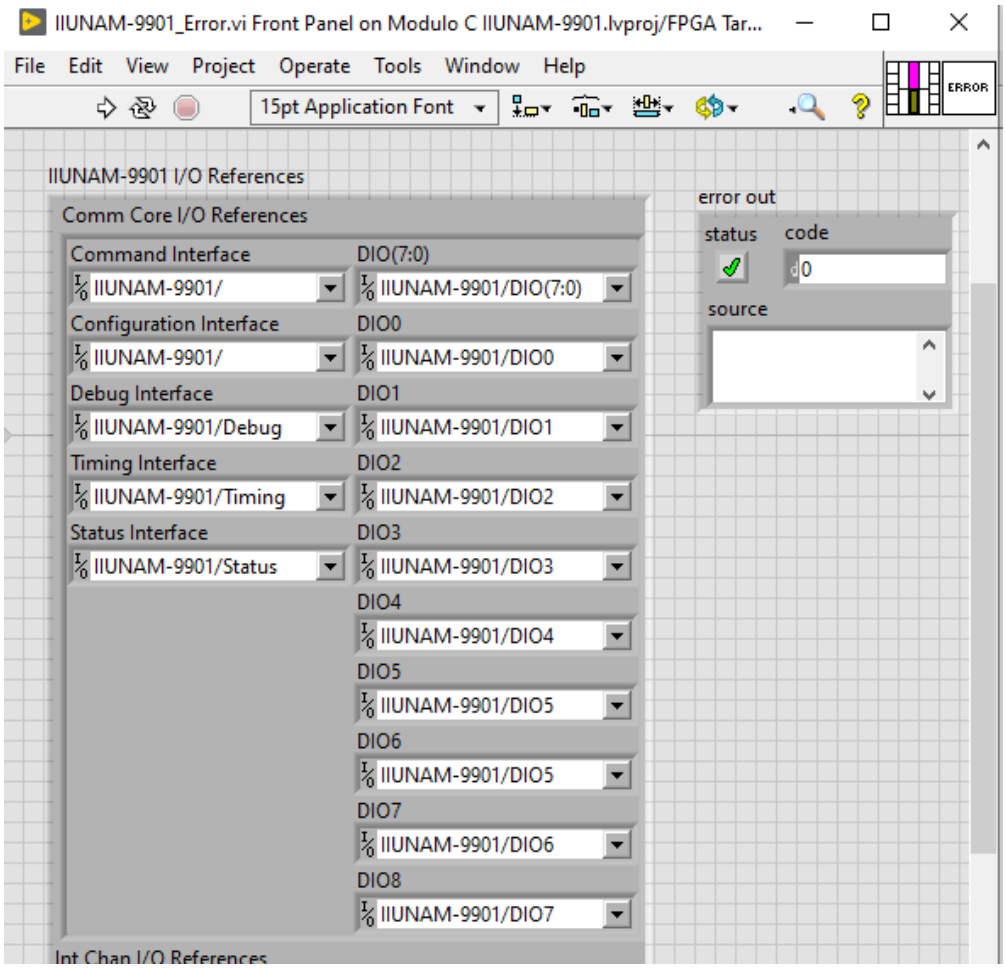


Figura 87. Panel del Vi llamado IIUNAM-9901_Error.

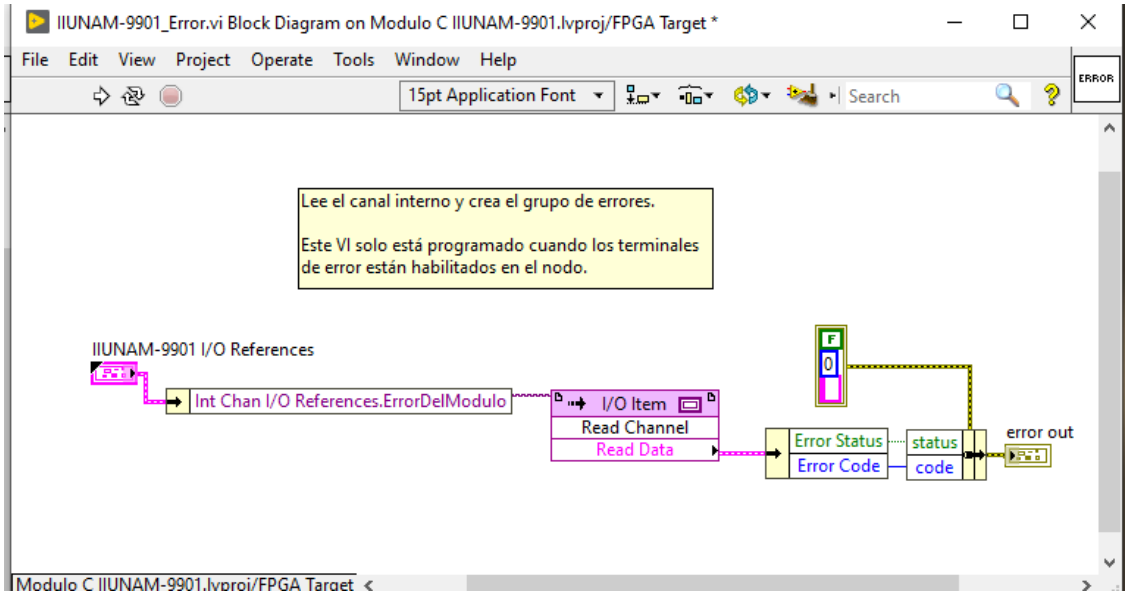


Figura 88. Diagrama de bloques del Vi llamado IIUNAM-9901_Error.

5.3.2 VI de Funcionamiento del módulo C

Finalmente, en la parte inicial de nuestro XML de soporte seleccionamos en el apartado de ResourceVI al Vi que controla todos los procesos, variables o procedimientos que realiza nuestro modulo C y este es el VI más importante para su funcionamiento por lo que merece un detalle más específico. En este caso se llamó IIUNAM-9901_Funcionamiento_del_modulo_C.vi este programa permite avalar el estado del módulo C, verificar si se encuentra algún error en funcionamiento, así como interactuar entre los distintos modos de funcionamiento y realizar los procesos necesarios para los canales definidos en su funcionamiento para que el usuario final solo vea el dato de salida o entrada, no su proceso.

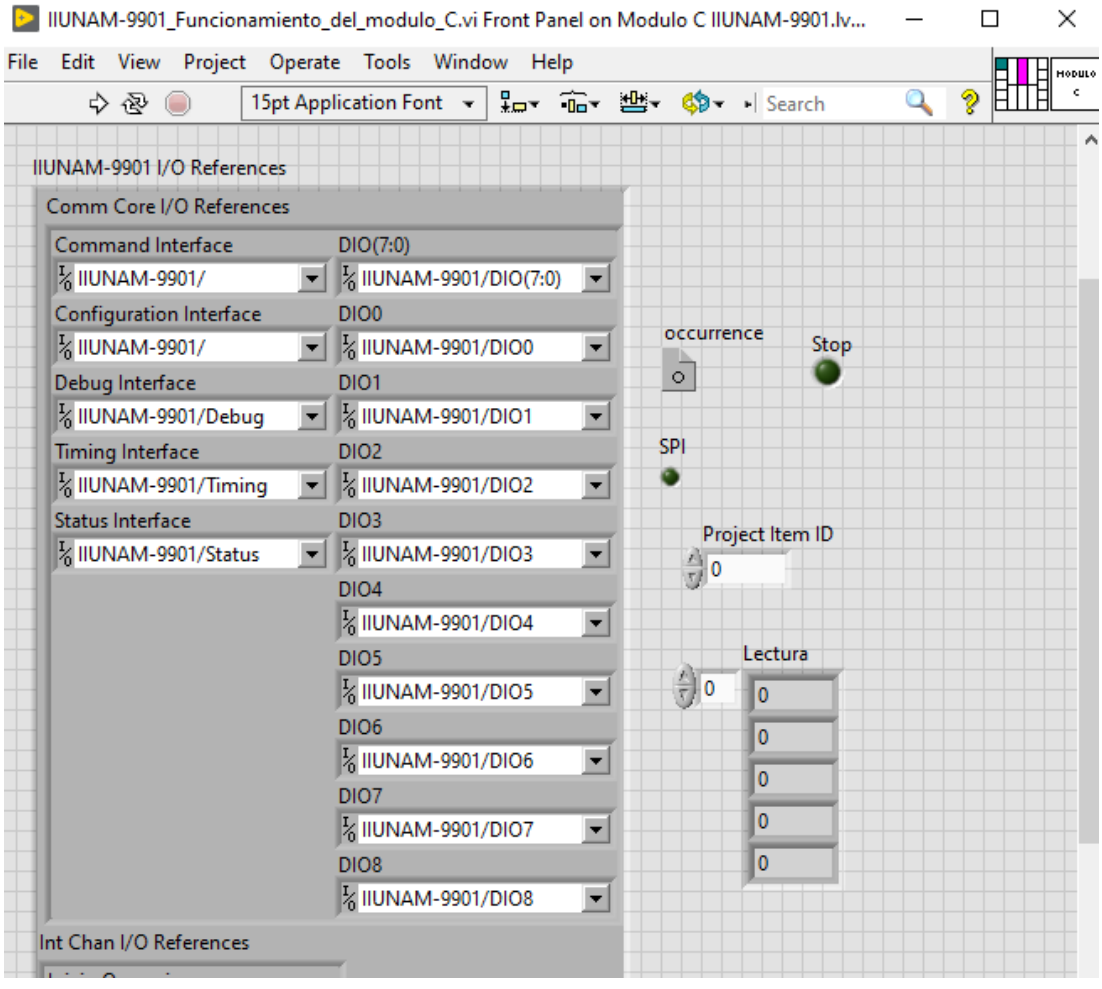


Figura 89. Panel del VI llamado IIUNAM-9901_Funcionamiento_del_modulo_C.vi

Para describir su funcionamiento a grandes rasgos presento el siguiente diagrama de flujo:

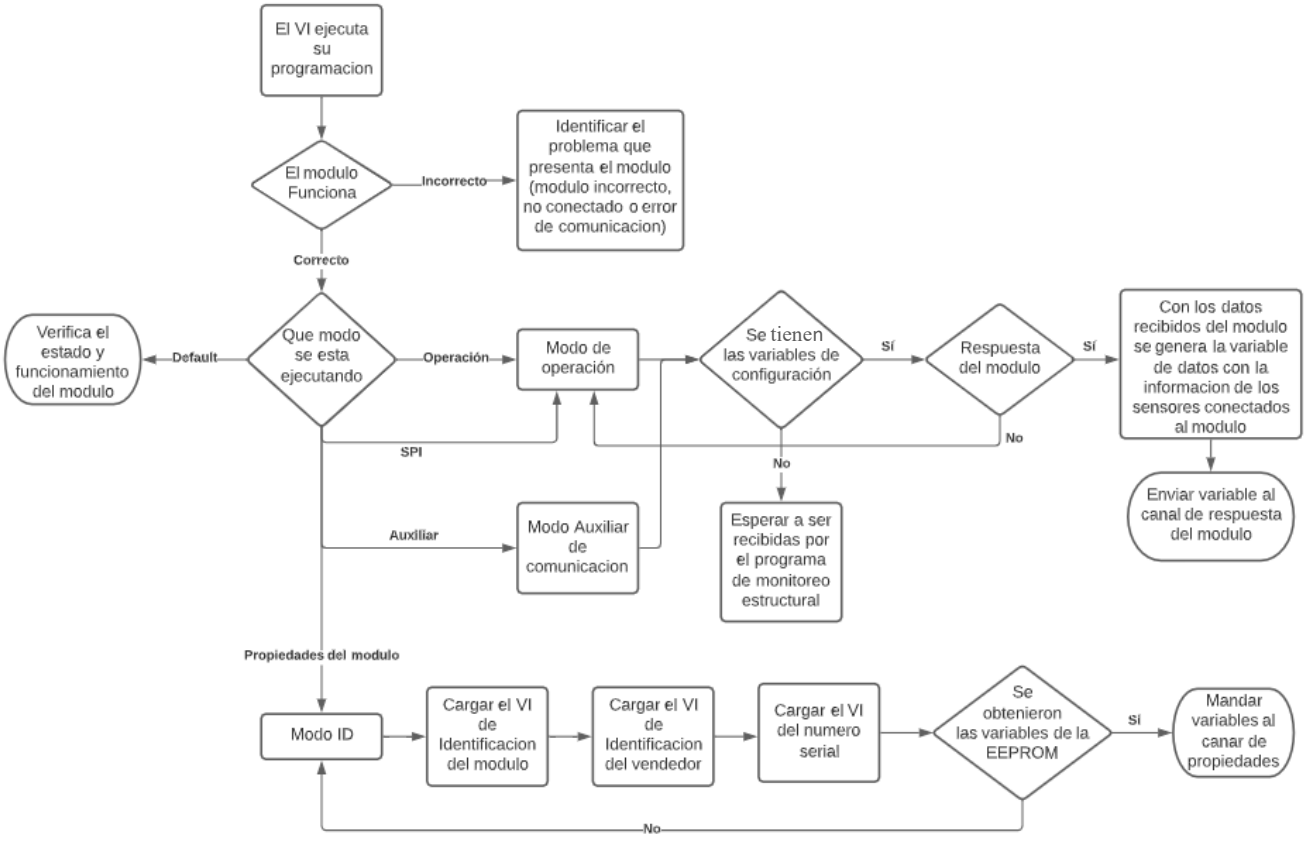


Figura 90. Diagrama de flujo del VI llamado IIUNAM-9901_Funcionamiento_del_modulo_C.vi
(Jimenez Sandoval, 2021)

Como se puede ver, el programa verifica si el módulo está conectado y si funciona correctamente, si es así se verifica qué modo de operación se quiere ejecutar en el mismo y dependiendo de la operación a realizar abre y ejecuta los VIs pertinentes, así como la comunicación SPI donde se obtienen los datos recibidos de los sensores conectados al módulo C; una vez realizados todos los procesos manda las variables obtenidas a través de los canales de comunicación pertinentes del módulo (Canales de propiedades o SPI).

5.4 Módulo C en modo lanzamiento

El “modo lanzamiento” del módulo C es en sí los programas creados para que el usuario final vea las características y especificaciones para lo que fue creado el módulo que se desea de manera particular para cada usuario, a través de una interface intuitiva y fácil de entender. En este caso en particular lo que se desarrollo es una serie de programas que permitieran tomar los datos del módulo C, que vienen de los canales creados en las secciones anteriores, y transformarlos en variables técnicas, que se almacenan en archivos para ser analizados cuando se desee, con el objetivo principal de analizar y monitorear una zona estructural en específico con sensores como las cuerdas vibrantes.

En si en este modo se crearon dos VI el primero se crea en el FPGA con el objetivo de enviar y recibir los datos necesarios para el funcionamiento del módulo C a través de sus canales creados, el segundo se crea en el cRIO o en la PC, interactuando con el anterior, teniendo la intención de trasformar la información obtenida del módulo en variables o datos técnicos específicos necesarios para su funcionamiento, así como el hecho de almacenar dichos datos para su uso posterior, recalcando que este programa se realiza de manera intuitiva y fácil de controlar para que el usuario realice las operaciones que necesite del módulo C.

Lo primero que se hace es cambiar de modo desarrollo a lanzamiento en nuestro proyecto por eso nos vamos a la ubicación:

C:\ProgramFiles\NationalInstruments\LabVIEW2019\vi.lib\LabVIEWTargets\FPGA\cRIO\shared\nicrio_Mdk2Utility\ Mdk2Utility_GenerateModuleSupportExport.vi

Ejecutamos o abrimos el programa, llenamos el apartado de *Modulo Name* con el nombre que le dimos a nuestro modulo, en este caso IIUNAM-9901, creamos una contraseña numérica en el apartado de Password (9968), seleccionamos la opción en *Export Type* de *Release Mode* y apuntamos el archivo a la carpeta donde tenemos el proyecto de nuestro modulo en *modo desarrollo.*

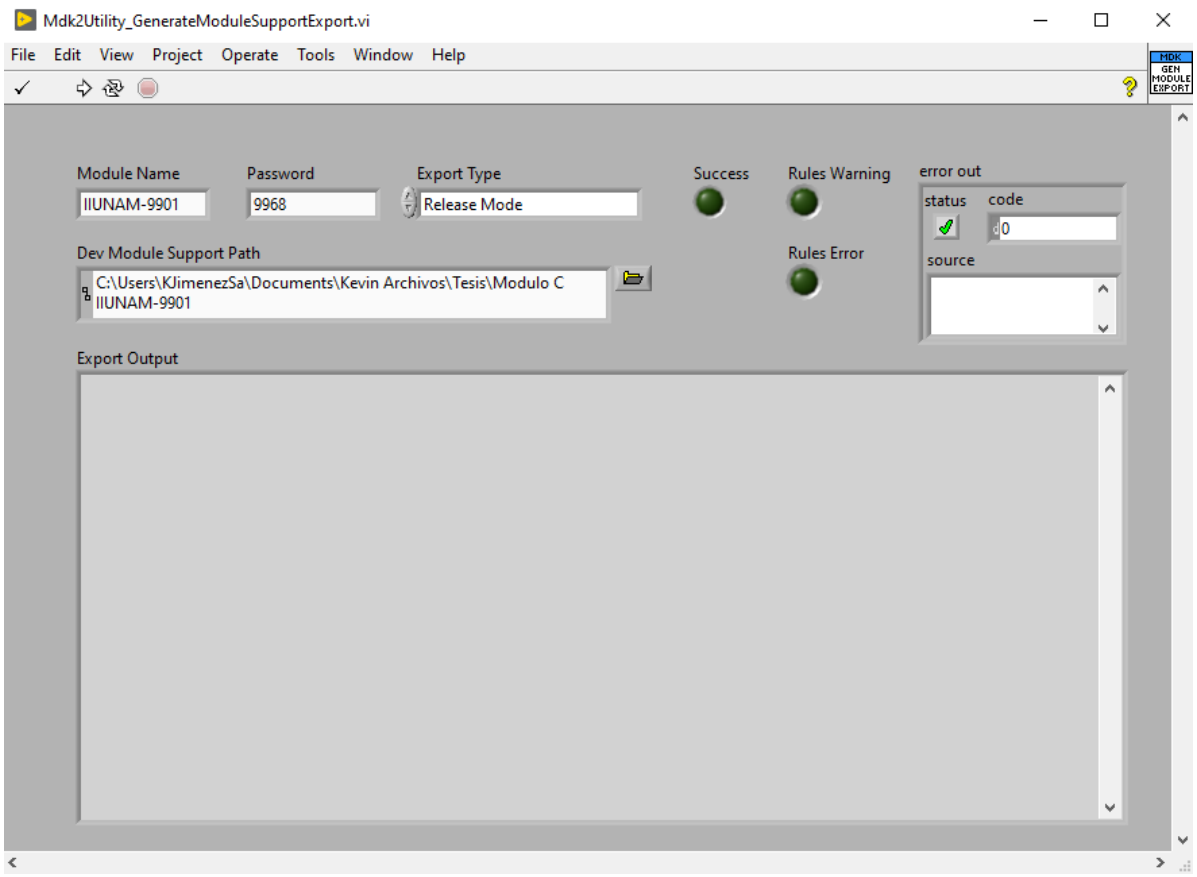


Figura 91. Configuración necesaria para pasar un módulo de desarrollo a lanzamiento.

Después solo ejecutamos el programa y esperamos hasta que nos muestre los errores que tenemos en nuestra programación según las especificaciones y reglas de National Instruments o a que nos muestre que todo es correcto como se muestra en la siguiente imagen:

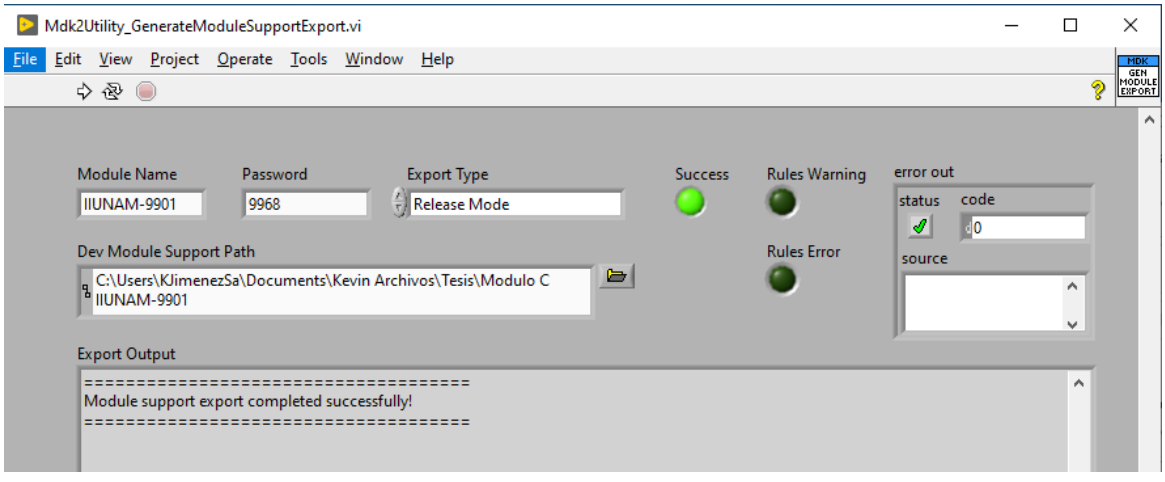


Figura 92. Mdk2Utility_GenerateModuloSupportExport modo lanzamiento correcto

Si se presenta un error en el apartado de *Export Output* nos indica en donde y a que se debe el error, se tiene que corregir y volver a ejecutar el programa hasta que nos muestre el mensaje de que el proceso se realizó correctamente. Cuando este correcto solo se crea un nuevo proyecto FPGA con nuestro cRIO con el módulo C conectado a él.

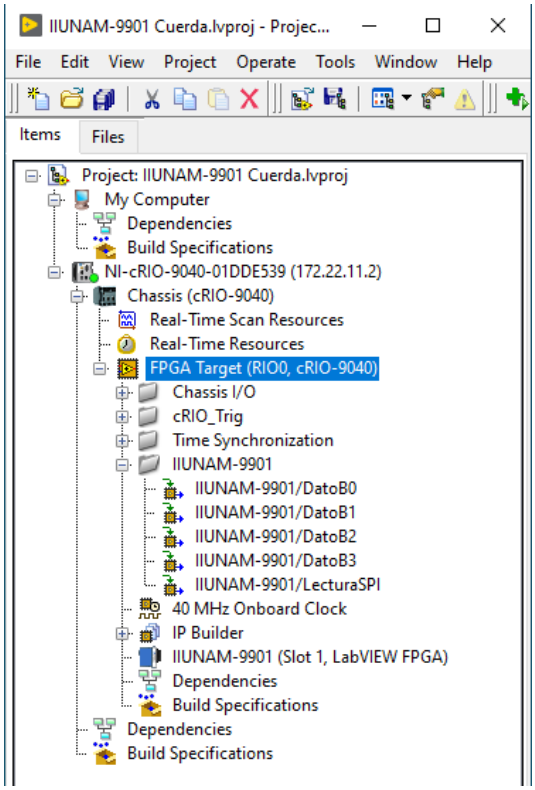


Figura 93. Modo de lanzamiento del módulo C IIUNAM-9901

Como se puede apreciar en la imagen anterior nuestro modulo ya reconoce el módulo C IIUNAM-9901 de manera automática y genera los canales de comunicación necesarios para su funcionamiento así que solo se crea un VI simple en el FPGA donde se emplea estos canales, de tal manera que empleamos los canales de DatoBO, DatoB1, DatoB2 y DatoB3 para mandar la configuración del módulo a través de comunicación SPI y en el canal de Lectura SPI esperamos la variable que contiene todos los datos de los sensores conectados al módulo C. Si se requieren los datos de identificación del módulo usamos los canales internos del mismo en su apartado de *Property Node* y creamos una variable para cada dato necesario (Id del vendedor, Id del módulo o número serial). Este conformaría el primer VI que interactúa con el siguiente de monitoreo que utiliza el usuario final para obtener la información que desea del mismo.

5.5 Programa de control y monitoreo dentro o fuera del CompactRIO

Para procesar los datos provenientes del módulo c de manera más sofisticada y compleja es necesario crear un VI en el cRIO o en la PC donde se trabaja, que a la vez nos permitirá el control y el monitoreo del módulo de manera tan simple o compleja como uno lo requiera.

Como realizar este programa ya depende de las características y necesidades que se tengan con el módulo creado, lo que hay que tener en cuenta es que este VI debe estar conectado con el VI en el FPGA para permitir la comunicación entre ellos y por ende la comunicación entre el cRIO, la PC y el módulo C diseñado.

Para realizar dicha conexión en el VI creado en el cRIO o en la PC se usa la paleta de comandos FPGA interface:

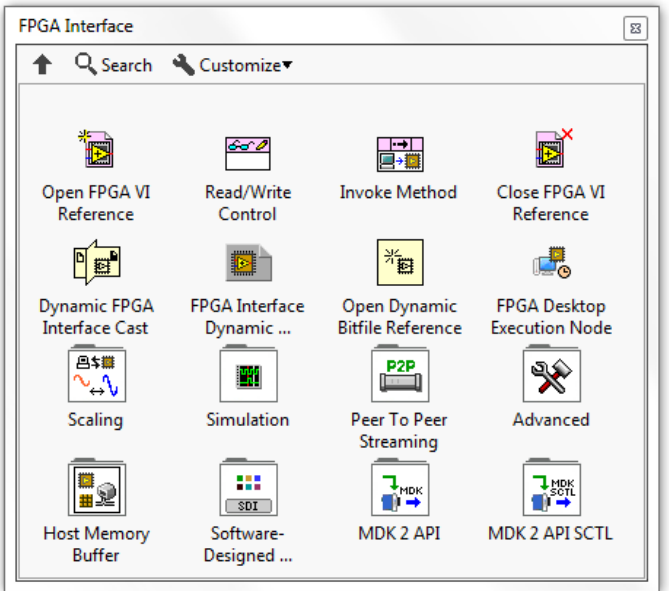


Figura 94. *FPGA Interface*

(Instruments, 2020)

Con los comandos de esta paleta podemos extraer datos y variables empleadas en el VI del FPGA, permitiendo emplear los recursos del cRIO o la PC para realizar un procesamiento más avanzado.

Los más importantes son los 4 comandos superiores que se ven en la paleta:

- El *Open FPGA VI Reference*: permite abrir y comunicar información al VI FPGA al que es referenciado.

- El *Closed FPGA VI Reference*: permite detener la comunicación y cerrar el programa del VI FPGA al que se referencio.

- El *Read/Write Control*: permite escribir o leer varíales de cualquier tipo empleadas en el VI FPGA de referencia, permitiendo obtener la información que tiene el FPGA y a la vez trasmitir nueva información al módulo desde este programa.

- El *Invoke Method*: permite mandar a llamar algún método empleado en el FPGA.

Esta paleta de comandos tiene muchas más opciones que permite una interacción mucho más compleja para programas más avanzados.

De manera ya concreta para nuestro caso se desarrolló un VI que emplea el módulo C IIUNAM-9901 del instituto de ingeniería que al indicarle el número de sensores (cuerdas vibrantes), el tipo de dato que se necesita y los rangos de frecuencia entrega valores específicos en valores ingenieriles como la resistencia [K ohms] y la frecuencia obtenida [Hz] de cada sensor.

Figura 95. *Panel Frontal del programa de configuración del Módulo IIUNAM-9901 en su primera versión*

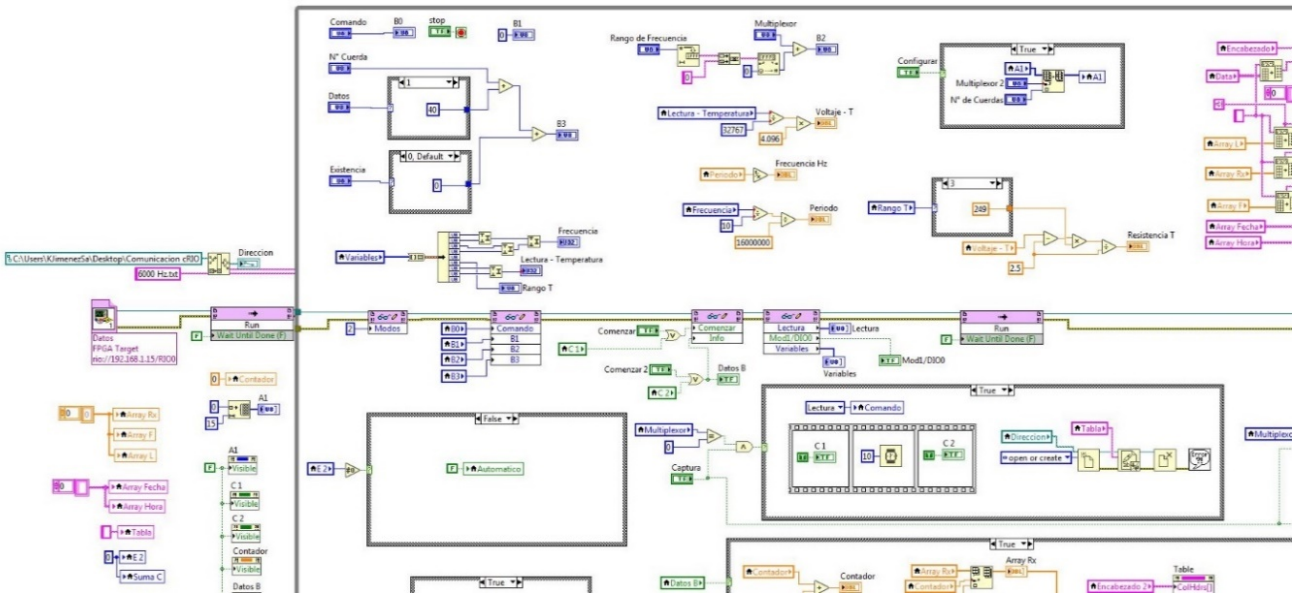


Figura 96. *Parte del Diagrama de Bloques del programa de configuración del Módulo IIUUAM-9901*

Como se puede ver el programa ya se puede volver tan complejo o sencillo como sea necesario para poder procesar los datos que manda y arroja el módulo C diseñado.

Es importante tener en cuenta que los procesos programados y/o la comunicación entre el FPGA y el módulo C es muy rápida, debido a las propiedades mismas del FPGA, por lo puede que muchos procesos se pueden realizar a tal velocidad que en la PC no puedan visualizarse o detectarse que se realizaron. Por lo que al programar el VI en el FPGA y este VI externo (PC o cRIO) debe considerarse que la comunicación entre dispositivos puede ser diferente y hay que diseñarlos de tal manera que puedan comunicarse entre sí, sin causar problemas entre ellos por diferencias de tiempo de ejecución. En mi caso lo solucione con variables empleadas como banderas para indicar entre los programas que algún proceso está en ejecución.

El último detalle a tener en cuenta en este apartado es que los VI de NI pueden ejecutarse en paralelo, lo que significa que el VI en el FPGA puede estar realizando procesos independientemente de lo que se haga o solicita en el VI externo, por lo que hay que tener cuidado cuando se requiera cambiar los procesos o modos de comunicación del módulo desde el VI externo, ya que puede provocar problemas de conexión, de funcionamiento o incluso estropear al módulo, por lo que siempre hay que estar monitoreando de manera externa los procesos que está realizando el módulo con el FPGA.

Capítulo 6. Resultados

Para verificar el funcionamiento del módulo C, se realizó una verificación de las lecturas obtenidas con los programas utilizando el cRIO y al módulo C “IIUNAM-9901” conectando a las entradas del lector de cuerda vibrante las señales de dos patrones o señales conocidas; una de frecuencia y otra de resistencia. Como primer patrón se empleó un calibrador multifunción (Fluke 741B), que puede generar salidas de tensión, corriente, frecuencia, resistencia, simulación de termopares, entre otras señales. Para mejorar la resolución y precisión en la generación de señal de frecuencia se utilizó un generador de funciones Tektronix AFG 3021B de 25 [MHz] y un potenciómetro de precisión para la resistencia.

Para comprobar la exactitud del módulo C en la medición de la frecuencia se tomaron 8 muestras para 9 diferentes frecuencias, que abarcaran el rango de frecuencia que el módulo puede medir (400 [Hz] – 6000 [Hz]), generando dicha frecuencia con los instrumentos de medición ya mencionados.

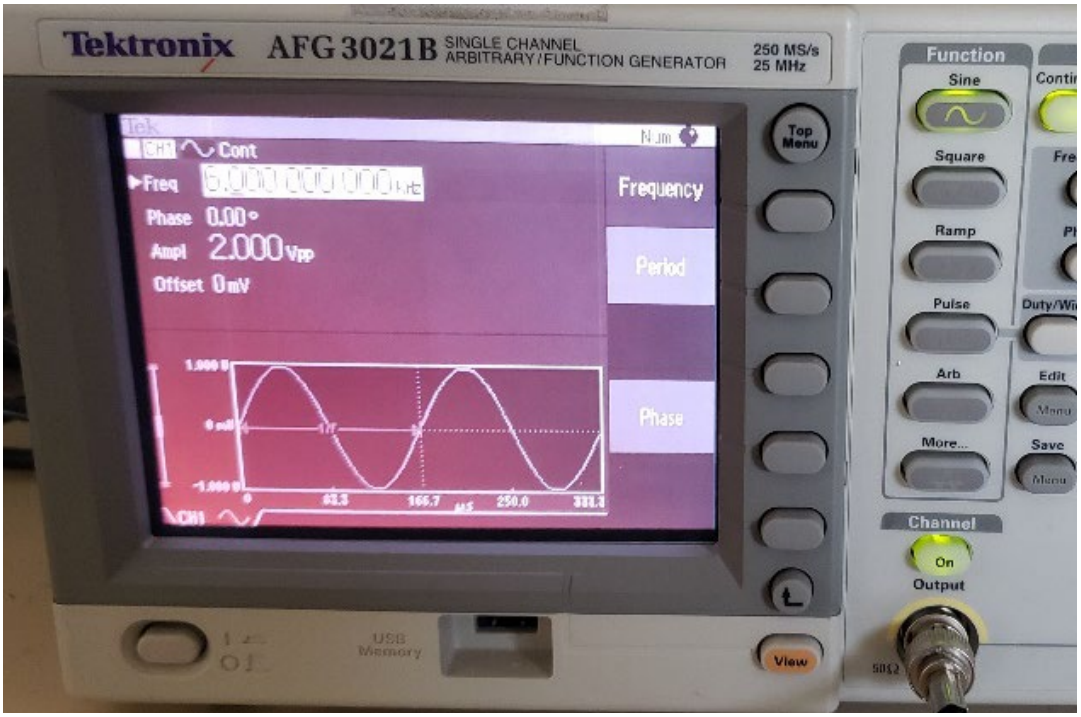


Figura 97. *Prueba a 6000 [Hz] con el generador de funciones Tektronix*

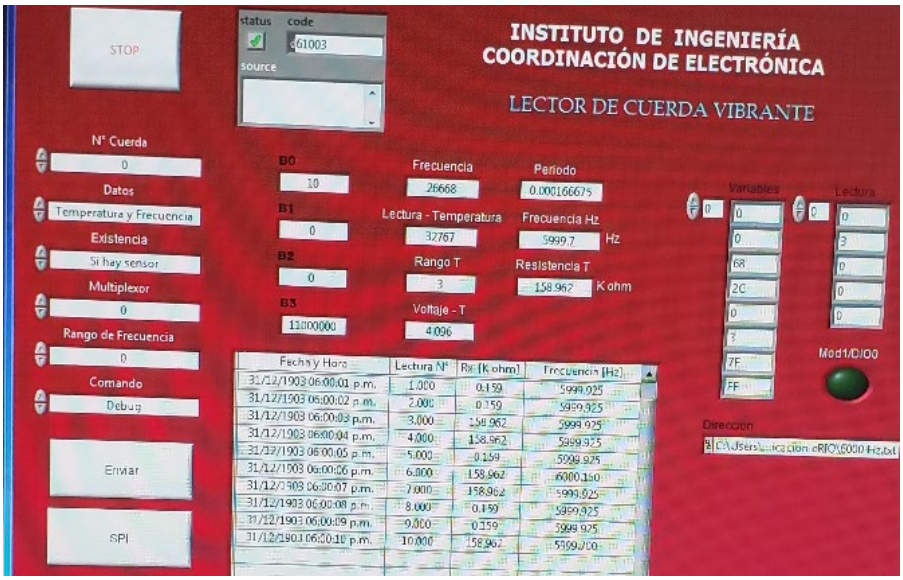


Figura 98. *Captura de la prueba a 6000 [Hz] con el generador de funciones Tektronix*

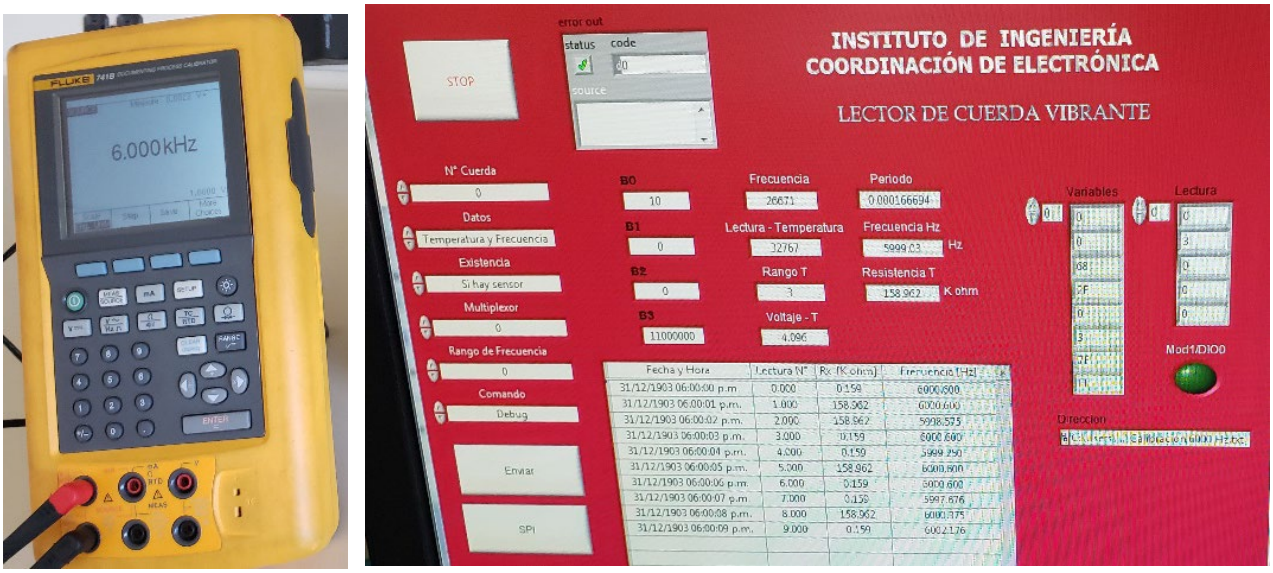


Figura 99. *Prueba a 6000 [Hz] con el calibrador multifunción Fluke*

Figura 100. *Captura de la prueba a 6000 [Hz] con el calibrador multifunción Fluke.*

Una vez realizadas todas las pruebas, los resultados obtenidos se procesaron en un Excel para identificar el ruido y resolución que tiene el módulo C en sus lecturas en frecuencia. Los resultados fueron los siguientes:

Lectura N°	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	
0	399.952	1099.997	1800.018	2500.039	3199.744	3898.920	4600.345	5300.119	6000.600	
1	399.926	1099.883	1800.322	2500.156	3199.552	3899.206	4600.610	5300.295	6000.600	
2	399.947	1099.747	1800.180	2499.844	3199.808	3899.586	4600.345	5300.119	5998.575	
3	399.949	1099.785	1800.140	2500.039	3199.744	3899.301	4600.610	5300.119	6000.600	
4	399.929	1100.072	1800.099	2499.844	3199.872	3899.681	4600.345	5300.470	5999.250	
5	399.991	1099.898	1800.200	2499.688	3199.552	3899.396	4600.610	5300.470	6000.600	
6	399.966	1099.929	1799.937	2500.117	3199.616	3899.015	4600.610	5300.119	6000.600	
7	399.926	1099.898	1800.140	2500.000	3199.744	3899.586	4600.610	5299.944	5997.676	
8	399.961	1099.981	1800.200	2499.805	3199.680	3899.586	4600.477	5300.119	6000.375	
Promedio	399.950	1099.910	1800.137	2499.948	3199.701	3899.364	4600.507	5300.197	5999.875	
Max	399.991	1100.072	1800.322	2500.156	3199.872	3899.681	4600.610	5300.470	6000.600	
Min	399.926	1099.747	1799.937	2499.688	3199.552	3898.920	4600.345	5299.944	5997.676	
Ruido MAX	0.041	0.162	0.185	0.208	0.171	0.317	0.103	0.273	0.725	
Ruido Min	0.024	0.163	0.200	0.260	0.149	0.443	0.162	0.254	2.199	Promedio
Adsoluto	0.041	0.162	0.185	0.208	0.171	0.317	0.103	0.273	0.725	0.243
	0.024	0.163	0.200	0.260	0.149	0.443	0.162	0.254	2.199	0.428
Ruido [Hz]	0.041	0.163	0.200	0.260	0.171	0.443	0.162	0.273	2.199	0.435
Valor Real [Hz]	400	1100	1800	2500	3200	3900	4600	5300	6000	

Figura 101. *Resultados obtenidos en la toma de frecuencia del módulo C empleando el Fluke.*

Lectura N°	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	Frecuencia [Hz]	
0	400.000	1099.997	1799.998	2500.000	3200.000	3899.966	4599.948	5299.944	5999.925	
1	400.000	1100.004	1799.998	2500.000	3200.000	3899.966	4600.081	5299.944	5999.925	
2	400.002	1099.997	1799.998	2500.000	3200.000	3899.966	4599.948	5299.944	5999.925	
3	400.001	1099.997	1799.998	2500.000	3200.000	3899.966	4599.948	5299.944	5999.925	
4	400.000	1099.997	1799.998	2499.961	3200.000	3899.966	4600.081	5299.944	5999.925	
5	400.000	1099.997	1799.998	2500.000	3200.000	3899.966	4600.081	5299.944	6000.150	
6	400.000	1099.997	1799.978	2500.000	3200.000	3899.966	4599.948	5299.944	5999.925	
7	400.000	1099.997	1799.998	2500.000	3200.000	3899.966	4599.948	5299.944	5999.925	
8	400.000	1100.004	1800.018	2499.961	3200.000	3900.061	4599.948	5299.944	5999.925	
Promedio	400.000	1099.998	1799.998	2499.991	3200.000	3899.976	4599.992	5299.944	5999.950	
Max	400.002	1100.004	1800.018	2500.000	3200.000	3900.061	4600.081	5299.944	6000.150	
Min	400.000	1099.997	1799.978	2499.961	3200.000	3899.966	4599.948	5299.944	5999.925	
Ruido MAX	0.002	0.006	0.020	0.009	0.000	0.085	0.088	0.000	0.200	
Ruido Min	0.000	0.002	0.020	0.030	0.000	0.011	0.044	0.000	0.025	Promedio
Adsoluto	0.002	0.006	0.020	0.009	0.000	0.085	0.088	0.000	0.200	0.045
	0.000	0.002	0.020	0.030	0.000	0.011	0.044	0.000	0.025	0.015
Ruido [Hz]	0.002	0.006	0.020	0.030	0.000	0.085	0.088	0.000	0.200	0.048
Valor Real [Hz]	400	1100	1800	2500	3200	3900	4600	5300	6000	

Figura 102. *Resultados obtenidos en la toma de frecuencia del módulo C empleando el generador Tektronix.*

Al analizar dichos resultados obtenemos un ruido promedio de 0.5 [Hz] en el valor de frecuencia obtenido por el módulo C.

Para medir la exactitud en las mediciones de temperatura, se verifico que el valor de resistencia necesario Rx [ohms] para la obtención de temperatura en el sensor de alta presión dentro del módulo C, fuera el correcto a través del Fluke y un potenciómetro de precisión. Las pruebas se realizaron de la misma forma que en frecuencia, tomando 10 muestras diferentes para 15 diferentes rangos de resistencia que estuvieran dentro del rango que el sensor tiene de 55.6 a 201 000 [ohms], tomado temperaturas de -50 [°C] a 150 [°C],

respectivamente, los resultados obtenidos fueron muy buenos con una precisión de 0.5 [ohms] y una resolución de 0.1 [ohms]en las lecturas, para valores de resistencia menores a 1 000 [ohms], y una precisión de 500 [ohms] con la misma resolución de 0.1 [ohms] para valores cercanos a los 200 [Kohms].

Figura 103. *Resultados obtenidos del módulo C teniendo una resistencia de 160 [K ohms] fue empleando el calibrador Fluke como patrón*

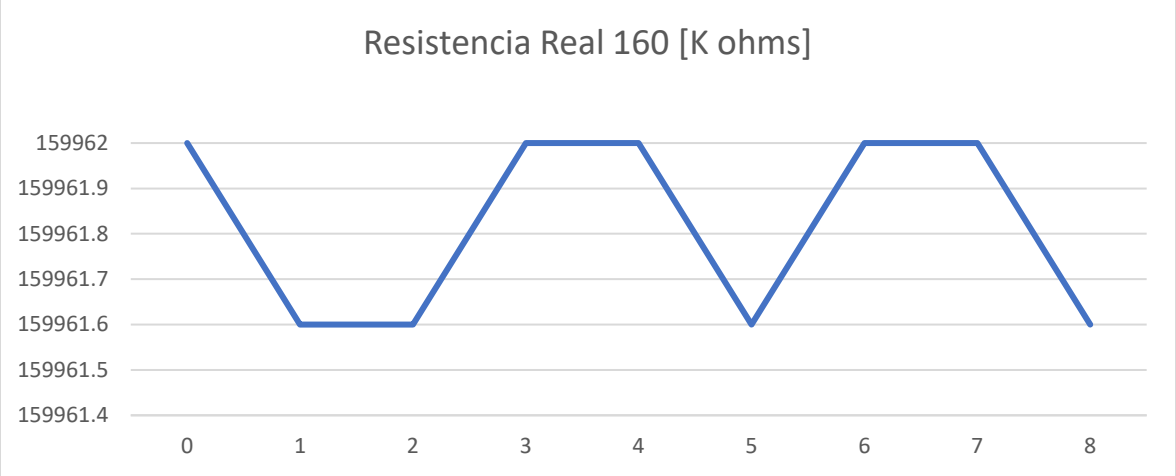


Figura 104. *Resultados obtenidos del módulo C teniendo una resistencia de 160 [K ohms] empleando el potenciómetro de precisión*

Capítulo 7. Conclusiones

Los resultados obtenidos en la verificación con el generador de señales y el calibrador Fluke muestran errores máximos de 2.3 Hz en las frecuencias altas y de 0.07 Hz en frecuencias bajas. Estos errores afectaran en un porcentaje máximo de 0.038 la lectura del sensor de cuerda vibrante. Lo cual se considera como aceptable para el equipo; ya que se presentan un mayor error en las lecturas de cuerdas vibrantes por los ruidos eléctricos inherentes en las conexiones de los sensores.

En lo que se refiera a las lecturas de resistencia para el termistor los resultados también se consideran muy aceptables; ya que el lector puede mejorar la resolución de la lectura con una función de auto escala que se configura para mejorar la lectura de la resistencia en todo su intervalo de 56 a 211 000 ohms. Por lo que el porcentaje de error en cualquiera de las escalas es similar.

Estos resultados nos indican que el equipo funciona de manera aceptable, como ya se mencionó, pero a la vez me permiten concluir que los programas diseñados y creados para su funcionamiento funcionan de manera correcta. Aunque el proceso es muy largo y complejo, la programación de cada parte que lo conforma a la vez es simple y estructurada, por lo que al tener claro el funcionamiento para el que se diseñó el módulo y obteniendo la información necesaria sobre las reglas y estructura que marca National Instruments para crear un módulo C solo fue necesario probar cada programa en la etapa de desarrollo en la que se estaba, para obtener este resultado satisfactorio.

De manera general considero haber cumplido mis objetivos con los programas creados en este trabajo y confirmar, hasta donde mis recursos me permiten, que mi hipótesis es cierta al crear un sistema que lee los datos de sensores tipo cuerda vibrante entregando información necesaria para el análisis de una estructura parecido a lo que entregan equipos profesionales con buena exactitud, y con la ventaja de adaptar la información y el procesamiento de los datos según las necesidades del usuario final.

Me gustaría mencionar que el sistema en general tiene posibilidades de mejorar sobre todo en la manera en la que se reciben los datos de los sensores una vez que se programa al

módulo, también en la parte de los tiempos de comunicación entre el programa de monitorea y el FPGA debido a que en ocasiones se presentan problemas a causa de esto. Además de que por experiencia de uno de mis sinodales (Miguel A. Mendoza G.) se me indico que a la hora de almacenar datos dentro de un cRIO este posee un tamaño de memoria pequeño y determinado por el modelo, lo que requiere que cada cierto tiempo estos datos sean trasferidos a un dispositivo externo para ser almacenados y borrados del cRIO, lo que mi programa no tiene.

También consideró importante mencionar como parte de mi conclusión que mi tesis tiene un formato muy parecido a un manual de cómo realizar los procesos paso por paso debido a que el problema más grande que afronte en el desarrollo del proyecto fue la obtención de información necesaria para llevarlo a cabo, debido a los requerimientos que National Instrument solicita para que funcione con sus producto, los cuales son súper escasos y difíciles de obtener si no es a través de una comunicación directa con ellos.

Este trabajo consideró que tuvo diversos contratiempos, desde situaciones externas, a internas como el hecho de que me vi en la necesidad de cambiar el cRIO con el que trabajaba en el desarrollo del sistema, debido a que se requirió para otro proyecto y se adquirió uno nuevo, que al final me impidió tomar más resultados y mejorar su funcionamiento, debido a que NI tenía problemas graves de funcionamiento en el modelo cRIO 9040 que fue el último modelo de cRIO con el que tuve que trabajar que no me permitió el funcionamiento correcto del módulo C.

Finalmente, para mí fue un proyecto largo y complejo, pero creo que logre hacer lo que se me solicito y plantee en este trabajo, cumpliendo así mis objetivos, por lo que solo resta agradecer a mis compañero y asesores dentro del instituto de ingeniería que hicieron posible este proyecto.

Referencias

ESPAÑA, R. (2017). *RANKIA*. Obtenido de ESPAÑA: https://www.rankia.com/blog/mundodelaempresa/1110194-marketing-ciclo-vida-productos-importancia-innovar

Figueroa, J. B. (2016). *Desarrollo del firmware y software de control para un dispositivo lector y almacenador de datos de cuerda vibrante con comunicación ethernet y/o USB.* Mexico D.F.: UNAM.

Instruments, N. (2020). *NI*. Obtenido de National Instruments: https://www.ni.com/es-mx.html

Jimenez Sandoval, K. E. (2021).

National Instruments Corporation. (2011). *cRIO MDK Hardware User Manual.* EU: NI.

National Instruments Corporation. (2011). *cRIO MDK Software User Manual.* EU: NI.

SISGEO. (2020). *SISGEO* . Obtenido de Trazamos soluciones: https://www.sisgeo.com/es/productos/piezometros/item/Piezometros-de-cuerda-vibrante.html#:~:text=C%C3%B3digos%20de%20referencia%20de%20los%20productos:%20PK45S,%20PK45A,%20PK45H,%20PK20A,

Ulrich. (2013). *Diseño y Desarrollo de Productos.* Mc Graw Hill.

ANEXOS

A1. Tablas del ModuleSupport XML

Tabla 1. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
1	MDKVersion Tag	String	Requerido	Especifica la versión del kit de desarrollo del módulo utilizado para desarrollar el soporte del módulo.
1	DevelopmentMode Tag	Boolean	Opcional	Especifica en qué modo (versión o desarrollo) aparece el módulo en el proyecto de LabVIEW. Si esta etiqueta no está presente, el soporte del módulo estará en modo *Release*.
1	Module Section	—	Requerido	Define las API de usuario final del módulo. Cada uno de los elementos enumerados en la sección del módulo del XML corresponde a un elemento del proyecto LabVIEW que aparece en el proyecto cuando se agrega el módulo. Cada uno de estos elementos del proyecto puede especificar qué interfaces (E / S, método y nodos de propiedad) se pueden usar con ellos. El módulo en sí es también un elemento del proyecto.
2	Name Attribute	ProjectItemName	Requerido	Especifica el nombre del módulo. Esto debe coincidir con el nombre del módulo definido en el archivo XML *ModuleType*.
2	ProjectItemID Tag	Integer	Opcional	Especifica el valor utilizado por el sistema del módulo de soporte VI. Cuando sus VIs de soporte de módulo están programados, una de las entradas de su VI es el valor especificado por *ProjectItemId*. Esto identifica en qué artículo del proyecto está operando el VI particular.
2	SupportedInterface List Section	—	Opcional	Contiene una lista de interfaces, definidas en otra parte del XML, que están disponibles en el módulo. Para el módulo, estas interfaces pueden ser nodos de método o propiedad. Los nodos de E/S no son compatibles directamente en el módulo. Cada elemento de *SupportedInterfaceList* se coloca dentro de las etiquetas XML de la interfaz.

Tabla 2. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
2	ResourceVI Section	—	Opcional	Especifica un VI que se incluirá en el diagrama FPGA en el momento de la compilación. No corresponde a ningún elemento de la API que se coloca en el diagrama de bloques. En su lugar, este VI se crea una instancia por módulo en el proyecto de LabVIEW.
3	Name Attribute	Simple Name	Requerido	Especifica el nombre del VI que se incluirá en el diagrama FPGA.
2	ModuleSubItemList Section	—	Opcional	Contiene una lista de subelementos del módulo. Los módulos que tienen una funcionalidad que no es E/S típica pueden usar el módulo subelementos en lugar de canales de E / S. Por ejemplo, el NI 9802 usa subelementos en el proyecto de LabVIEW para dos ranuras para tarjetas SD y el NI 9871 usa subelementos en el proyecto de LabVIEW para dos puertos seriales.
2	ModuleSubItem Section	—	Opcional	Aparece en el proyecto de LabVIEW bajo el ítem del módulo.
3	Name Attribute	ProjectItemName	Requerido	Especifica el nombre del subelemento del módulo que aparece en el proyecto de LabVIEW
3	ProjectItemId Tag	Integer	Opcional	Especifica el valor utilizado por los scripters del módulo de soporte VI. El rango para esta etiqueta es 0–255. Cuando sus VIs de soporte de módulo están programados, una de las entradas de su VI es el valor especificado por ProjectItemId. Esto se utiliza para identificar en qué elemento del proyecto está operando el VI particular.
3	SupportedInterface List Section	—	Opcional	Contiene una lista de interfaces, definidas en otra parte del XML, que están disponibles en el subelemento del módulo. Para los subelementos del módulo, estas interfaces pueden ser nodos de método o propiedad. Los nodos de E / S no son compatibles con los subelementos del módulo. Cada elemento de SupportedInterfaceList se coloca dentro de las etiquetas XML de la interfaz.

Tabla 3. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
4	Interface Tag	String	Opcional	—
3	ModuleSubItemIcon Tag	SimpleName (.png)	Opcional	Especifica el icono utilizado en el proyecto para mostrar el subelemento del módulo.
2	IOChannelList Section	—	Opcional	Contiene una lista de los canales de E / S que estarán disponibles en su módulo.
3	IOChannel Section	—	Opcional	Aparece como un elemento de E / S en el proyecto de LabVIEW. Al igual que los canales de E / S de NI, los elementos de E / S se colocarán automáticamente en una carpeta de módulo en el proyecto de LabVIEW cuando el módulo se agregue al objetivo FPGA.
4	Name Attribute	ProjectItemName	Requerido	Especifica el nombre del canal de E / S que aparece en el proyecto de LabVIEW.
4	ProjectItemId Tag	Integer	Requerido	Especifica el valor utilizado por los scripters del módulo de soporte VI. El rango para esta etiqueta es 0–255. Cuando sus VIs de soporte de módulo están programados, una de las entradas de su VI es el valor especificado por ProjectItemId. Esto se utiliza para identificar en qué elemento del proyecto está operando el VI particular.
4	SupportedInterface List Section	—	Opcional	Contiene una lista de interfaces, definidas en otra parte del XML, que están disponibles en el módulo. Para los elementos del canal de E / S, estas interfaces pueden ser E / S, método o nodos de propiedad. Cada uno de estos elementos en SupportedInterfaceList se coloca dentro de las etiquetas XML de la interfaz.
5	Interface Tag	RestrictedString	Opcional	—

Tabla 4. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
5	ParallelDigitalInterface Tag	Enumerated	Opcional	Contiene interfaces para líneas de E / S digitales paralelas. Las opciones para esta etiqueta son DIO0-DIO7, DI0-DI7, DO0-DO7. El soporte para estas interfaces es proporcionado por el software MDK 2.0. Al usar líneas de E / S digitales paralelas, no necesita especificar VIs que se guarden debajo de los nodos de E / S como lo hace para otras interfaces. Cada línea de E / S digital paralela se coloca dentro de las etiquetas XML ParallelDigitalInterface.
1	PropertyNodeInterface List Section	—	—	Contiene los detalles de cada una de las interfaces del nodo de propiedad que se especificaron en las secciones SupportedInterfaceList de los elementos del proyecto.
2	Interface Section	—	—	—
3	Name Attribute	RestrictedString	Requerido	Especifica el nombre del nodo de propiedad que está disponible en el diagrama de bloques de LabVIEW.
3	DataType Tag	Enumerated	Requerido	Especifica el tipo de datos del nodo de propiedad. Las opciones para esta etiqueta son I8, U8, I16, U16, I32, U32 y Boolean. También puede especificar un nombre de control en esta etiqueta. Este control puede ser un clúster, una matriz, un punto fijo o cualquier otro tipo de datos permitido en LabVIEW FPGA. El nombre del control se debe especificar con la extensión .ctl, por ejemplo, MDK-1234_MyControl.ctl. El control debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.
3	Direction Tag	Enumerated	Requerido	Especifica la dirección del nodo de propiedad. Las opciones para esta etiqueta son Read, Write, BiDirectional.

Tabla 5. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	DefaultDirection Tag	Enumerated	Requerido cuando la dirección es bidireccional.	Especifica la dirección del nodo cuando se coloca por primera vez en el diagrama de bloques. No está permitido cuando la dirección es Lectura o Escritura. Las opciones para la Dirección predeterminada son Lectura y Escritura.
3	NodeIcon Tag	Enumerated	Requerido	Especifica el icono que aparece en el nodo en el diagrama de bloques. Las opciones para esta etiqueta son AI, AO, DI, DO, DIO y Puerto.
3	WriteVIScriptInfo Section	—	Se requiere cuando la dirección del nodo de propiedad es Escritura o BiDireccional	Especifica los VIs de soporte de módulo para el nodo de propiedad y cómo deben conectarse.
4	Name Attribute	SimpleName	Requerido	Especifica el nombre de la sección WriteVIScriptInfo. Este nombre no aparece en la API del usuario final y solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe usarse para ninguna otra sección de VIScriptInfo.
4	UseInstanceData Tag	Boolean	Opcional	Especifica si el terminal de datos de instancia se utilizará en los VIs de soporte de módulo.
5	VIList Section	—	—	Contiene una lista de VIs de soporte de módulo que se incluirán en el script debajo del nodo de propiedad. Para los nodos de propiedades, solo puede especificar un VI además del VI opcional de manejo de errores.
5	VI Section	—	—	—

Tabla 6. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
6	Name Attribute	Simple Name (.vi)	Requerido	Especifica el nombre del VI que se incluirá en el script debajo del nodo de propiedad. Especifique el nombre de VI con la extensión .vi, por ejemplo, MDK-1234_MyVI.vi. El VI debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.
6	SequenceOrder Tag	Integer	Requerido	Especifica en qué parte de la secuencia de cuadros aparecerá este VI en particular cuando se escriban múltiples VIs en secuencia debajo del nodo de propiedad. El rango para esta etiqueta es 0–255.
6	VIHasTerminalConnection Tag	Boolean	Opcional	Indica que este VI en particular contiene un terminal que se conectará al terminal de entrada o salida del nodo de propiedad. Esta etiqueta es opcional, pero uno y solo un VI en VIList debe contener esta etiqueta establecida en verdadero.
6	ErrorHandling Tag	Boolean	Opcional	Indica que este VI en particular contiene un terminal que se conectará a la salida de error del nodo de propiedad. Para que su nodo de propiedad produzca códigos de error, uno y solo un VI en VIList debe contener esta etiqueta establecida en verdadero.
3	ReadVIScriptInfo Section	—	Se requiere cuando la dirección del nodo de propiedad es de lectura o bidireccional	Especifica los VIs de soporte de módulo para el nodo de propiedad y cómo deben conectarse.
1	MethodNodeInterfaceList Section	—	—	Contiene los detalles para cada una de las interfaces del nodo de método que se especifican en las secciones MethodInterfaceList de los elementos del proyecto.

Tabla 7. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
2	Interface Section	—	—	—
3	Name Attribute	RestrictedString	Requerido	Especifica el nombre del nodo de método que está disponible en el diagrama de bloques de LabVIEW. Este nombre debe ser único y no debe utilizarse para ninguna otra sección de interfaz.
3	MethodNodeTerminalList Section	—	Opcional	Define los terminales que se mostrarán en el nodo de método en el diagrama de bloques de LabVIEW. Si esta sección no está definida, el nodo de método no tendrá ningún terminal.
4	MethodNodeTerminal Section	—	—	—
5	Name Attribute	RestrictedString	Requerido	Especifica el nombre del terminal de nodo de método que aparece en el diagrama de bloques de LabVIEW.
5	DataType Tag	Enumerated o Simple Name (.ctl)	Requerido	Especifica el tipo de datos del nodo de método. Las opciones para esta etiqueta son I8, U8, I16, U16, I32, U32 y Boolean. También puede especificar un nombre de control en esta etiqueta. Este control puede ser un clúster, una matriz, un punto fijo o cualquier otro tipo de datos permitido en LabVIEW FPGA. El nombre del control se debe especificar con la extensión .ctl, por ejemplo, MDK-1234_MyControl.ctl. El control debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.
5	Direction Tag	Enumerated	Requerido	Especifica la dirección del terminal del nodo de método. Las opciones para esta etiqueta son Leer y Escribir.
5	Required Tag	Boolean	Opcional	Especifica el cableado en un terminal de escritura. Esta etiqueta solo se puede utilizar con terminales de escritura.

Tabla 8. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
				Cuando se establece en verdadero, el terminal de escritura en el nodo del método debe tener algo conectado al diagrama de bloques. Cuando se establece en falso, el terminal de escritura se puede dejar desconectado y se usará un valor predeterminado.
5	TerminalOrder Tag	Integer	Requerido	Especifica el orden en el nodo del método en el que aparece el terminal. El orden comienza con el terminal más alto que tiene una orden de terminal de cero. El rango para esta etiqueta es 0–255.
3	NodeIcon Tag	Enumerated	Requerido	Especifica el icono que aparece en el nodo en el diagrama de bloques. Las opciones para esta etiqueta son AI, AO, DI, DO, DIO y Puerto.
3	MethodVIScriptInfo Section	—	Requerido	Especifica los VIs de soporte de módulo para el nodo de método y cómo deben conectarse.
4	Name Attribute	SimpleName	Requerido	Especifica el nombre de la sección MethodVIScriptInfo. Este nombre no aparece en la API del usuario final y solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe usarse para ninguna otra sección de VIScriptInfo.
4	UseInstanceData Tag	Boolean	Opcional	Especifica si el terminal de datos de instancia se utilizará en los VIs de soporte de módulo.
5	VIList Section	—	—	Contiene una lista de VIs de soporte de módulo que se incluirán en el guión debajo del nodo del método. Para los nodos de método, solo puede especificar un VI además del VI de manejo de errores opcional.
5	VI Section	—	—	—
6	Name Attribute	SimpleName (.vi)	Requerido	Especifica el nombre del VI que se creará en el script debajo del nodo del método. Especifiqué el nombre de VI con la extensión .vi, por ejemplo, MDK-1234_MyVI.vi. El VI debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.

Tabla 9. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
6	SequenceOrder Tag	Integer	Requerido	Especifica en qué parte de la secuencia de cuadros aparecerá este VI en particular cuando se escriban múltiples VIs en secuencia debajo del nodo del método. El rango para esta etiqueta es 0–255.
6	TerminalConnection List Section	—	Opcional	Esta etiqueta es opcional, pero si el nodo del método contiene alguno, uno y solo un VI en la lista VIL debe incluir esta sección.
7	TerminalConnecti on Section	—	Opcional	Cada terminal del nodo de método debe tener una conexión de terminal correspondiente en la TerminalConnectionList.
7	Name Attribute	SimpleN ame	Requerido	Especifica el nombre de la sección TerminalConnection. Este nombre no aparece en la API del usuario final y solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe utilizarse para ninguna otra sección de TerminalConnection.
7	NodeTerminalName Tag	Restricte dString	Requerido	Especifica el terminal de nodo de la sección MethodNodeTerminalList que se está conectando.
7	ConnectorPaneTer minal Tag	Integer	Requerido	Especifica el terminal en el panel del conector del módulo de soporte VI que se está conectando al terminal del nodo de método. El rango para esta etiqueta es 0–255.
6	ErrorHandling Tag	Boolean	Opcional	Indica que este VI en particular contiene un terminal que se conectará a la salida de error del nodo de método. Para que su nodo de método produzca códigos de error, uno y solo un VI en VIList debe contener esta etiqueta establecida en verdadero.
1	IONodeInterfa ceList Section	—	—	Contiene los detalles de cada una de las interfaces de nodo de E / S que se especificaron en las secciones IONodeInterfaceList de los elementos del proyecto.
2	Interface Section	—	—	—

Tabla 10. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	Name Attribute	SimpleName	Requerido	Especifica el nombre de la sección de interfaz, Este nombre no aparece en la API del usuario final, porque los nodos de E / S heredan nombres de los elementos del proyecto con los que están asociados. Por ejemplo, el nodo de E / S para el proyecto AI0 se llamará AI0. Este nombre solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe utilizarse para ninguna otra sección de la interfaz.
3	DataType Tag	Enumerated or SimpleName (.ctl)	Requerido	Especifica el tipo de datos del nodo de E / S. Las opciones para esta etiqueta son I8, U8, I16, U16, I32, U32 y Boolean. También puede especificar un nombre de control en esta etiqueta. Este control puede ser un clúster, una matriz, un punto fijo o cualquier otro tipo de datos permitido en LabVIEW FPGA. El nombre del control se debe especificar con la extensión .ctl, por ejemplo, MDK-1234_MyControl.ctl. El control debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.
3	Direction Tag	Enumerated	Requerido	Especifica la dirección del terminal de nodo de E / S. Las opciones para esta etiqueta son Lectura, Escritura y BiDireccional.
3	DefaultDirection Tag	Enumerated	Requerido cuando la dirección es bidireccional.	Especifica la dirección del nodo cuando se coloca por primera vez en el diagrama de bloques. No está permitido cuando la dirección es Lectura o Escritura. Las opciones para la Dirección predeterminada son Lectura y Escritura.
3	NodeIcon Tag	Enumerated	Requerido	Especifica el icono que aparece en el nodo en el diagrama de bloques. Las opciones para esta etiqueta son AI, AO, DI, DO, DIO y Puerto.

Tabla 11. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	WriteVIScriptInfo Section	—	Se requiere cuando la dirección del nodo de E / S es Escritura o BiDireccional	Especifica los VIs de soporte del módulo para el nodo de E / S y cómo deben conectarse.
4	Name Attribute	SimpleName	Requerido	Especifica el nombre de la sección WriteVIScriptInfo. Este nombre no aparece en la API del usuario final y solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe usarse para ninguna otra sección de VIScriptInfo.
4	UseInstanceData Tag	Boolean	Opcional	Especifica si el terminal de datos de instancia se utilizará en los VIs de soporte de módulo.
5	VIList Section	—	—	Contiene una lista de VIs de soporte de módulo que se incluirán en el guión debajo del nodo de E / S. Para los nodos de E / S, puede especificar varios VIs de soporte de módulo además del VI de manejo de errores.
5	VI Section	—	—	—
6	Name Attribute	SimpleName (`.vi`)	Requerido	Especifica el nombre del VI que se creará en el script debajo del nodo de E / S. Especifiqué el nombre de VI con la extensión .vi, por ejemplo, MDK-1234_MyVI.vi. El VI debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta.
6	SequenceOrder Tag	Integer	Requerido	Especifica en qué parte de la secuencia de cuadros aparecerá este VI en particular cuando se escriban múltiples VIs en secuencia debajo del nodo de E / S. El rango para esta etiqueta es 0–255.

Tabla 12. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
6	VIScope Tag	Enumerated	Opcional	Especifica cuántas veces este VI se incluirá en un nodo de E / S en crecimiento. Las opciones para esta etiqueta son NodeScoped y ChannelScoped. Cuando se establece en NodeScoped, el VI solo se ejecutará una vez debajo del nodo de E / S, sin importar cuántos canales haya en el nodo en crecimiento. Cuando se establece en ChannelScoped, el VI se creará una secuencia de comandos para cada canal en el nodo de E / S en crecimiento. Cuando la etiqueta no está presente, el VI se establecerá de forma predeterminada en NodeScoped.
6	VIHasTerminalConnection Tag	Boolean	Opcional	Indica que este VI en particular contiene un terminal que se conectará al terminal de entrada o salida del nodo de E / S. Esta etiqueta es opcional, pero uno y solo un VI en VIList debe contener esta etiqueta establecida en verdadero.
6	ErrorHandling Tag	Boolean	Opcional	Indica que este VI en particular contiene un terminal que se conectará a la salida de error del nodo de E / S. Para que su nodo de E / S produzca códigos de error, uno y solo un VI en VIList debe contener esta etiqueta establecida en verdadero.
3	ReadVIScriptInfo Section	—	Se requiere cuando la dirección del nodo de E / S es de lectura o bidireccional	Especifica los VIs de soporte del módulo para el nodo de E / S y cómo deben conectarse.
1	MergedIONodeVISCriptInfoList Section	—	—	Especifica cómo se fusionarán las interfaces particulares del nodo de E / S cuando diferentes elementos de E / S están en el mismo nodo de E / S en crecimiento.

Tabla 13. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
2	MergeIONodeVIScriptInfo Section	—	Opcional	Especifica VIScriptInfos que se pueden fusionar cuando se usan en el mismo nodo de E / S en crecimiento.
3	Name Attribute	Simple Name	Requerido	Especifica el nombre de la sección MergedIONodeVIScriptInfo. Este nombre no aparece en la API del usuario final y solo se usa internamente dentro del archivo XML. Este nombre debe ser único y no debe usarse para ninguna otra sección de MergedIONodeVIScriptInfo.
3	VIScriptInfoName Tag	Simple Name	—	Dos o más etiquetas VIScriptInfoName pueden aparecer en la sección MergedIONOdeVIScriptInfo. Estos nombres corresponden a los nombres dados a las secciones particulares de VIScriptInfo (WriteVIScriptInfo, ReadVIScriptInfo) que se enumeraron en las diferentes interfaces.
1	InternalChannelList Section	—	Opcional	Enumera los diferentes canales internos que se utilizarán dentro de los VIs de soporte de módulos.
2	InternalChannel Section	—	Opcional	Especifica cómo funcionará el canal interno.
3	Name Attribute	SimpleName	Requerido	Especifica el nombre de la sección InternalChannel. Este nombre no aparece en la API del usuario final, ya que los canales internos están ocultos cuando el módulo se usa en el modo Release. Todos los nombres de los canales internos deben ser únicos.
3	InternalChannelType Tag	Enumerated	Requerido	Especifica qué tipo de funcionalidad tendrá el canal interno. Las opciones para esta etiqueta son Asincrónica, Bloqueo y Ocurrencia.

Tabla 14. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	DataType Tag	Enumerated	Requerido cuando el tipo es asíncrono o bloqueo	Especifica el tipo de datos del canal interno. Las opciones para esta etiqueta son I8, U8, I16, U16, I32, U32 y Boolean. También puede especificar un nombre de control en esta etiqueta. Este control puede ser un clúster, una matriz, un punto fijo o cualquier otro tipo de datos permitido en LabVIEW FPGA. El nombre del control se debe especificar con la extensión .ctl, por ejemplo, MDK-1234_MyControl.ctl. El control debe estar ubicado directamente dentro de la carpeta de soporte del módulo y no dentro de una subcarpeta. Esta etiqueta es opcional porque el tipo de canal interno de ocurrencia no especifica un tipo de datos. Si está utilizando los tipos de canales internos de bloqueo o asíncronos, debe especificar un tipo de datos.
1	ModuleModeDefinition Section	—	Requerido	Especifica cómo el núcleo de comunicación de la serie C operará en sus diferentes modos de operación.
2	NormalOperationMode Section	—	Opcional	Esta sección es opcional, pero todos los módulos deben contener una sección NormalOperationMode ya que este es el modo en el que se produce la comunicación primaria con el módulo.
3	SPIConfiguration Section	—	Opcional	Reserva las siguientes líneas de bus CompactRIO para el motor SPI en modo de funcionamiento normal: SPI_CLK, ~ SPI_CS, MISO, MOSI. Cuando está reservado por el motor SPI, estas líneas no se pueden usar para E / S digital en el modo de operación normal. Esta sección debe estar presente para poder utilizar las interfaces SPI del núcleo de comunicación de la serie C cuando el módulo está en el modo de operación normal.

Tabla 15. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
4	SPIHalfTauTicks Tag	Integer	Requerido	Especifica qué tan amplio es el período SPI Clock 1/2 Clock en tics de reloj base de 25 ns. El rango para esta etiqueta es 2–65535. Si su módulo tiene un SPI Tau de 150 ns, la etiqueta SPIHalfTauTicks se establecerá en El período del reloj SPI de 150 ns será de 6 tics de reloj base de ancho. La señal del reloj SPI debe ser simétrica, por lo que tendrá 3 tics altos y 3 tics bajos.
3	Convert PulseConfiguration Section	—	Opcional	Reserva la línea ~ CONVERTIR del bus CompactRIO para la lógica de conversión de impulsos cuando está presente. Esta sección debe estar presente para poder utilizar la interfaz de conversión de impulsos del núcleo de comunicación de la serie C.
4	ConvertPulseWidth Tag	Enumerated	Requerido	Especifica cuánto tiempo será el pulso de conversión. Las opciones para esta etiqueta son Corto, Medio y Largo. Estos tres anchos de pulso corresponden a los anchos de pulso de conversión descritos en el Manual del usuario del hardware del Kit de desarrollo de módulos CompactRIO.
3	DoneWaitConfiguration Section	—	Opcional	Reserva la línea ~ DONE del bus CompactRIO para la lógica de espera hecha cuando está presente. Esta sección debe estar presente para poder utilizar la interfaz Esperar en lo hecho del núcleo de comunicación de la serie C.
4	DoneWaitTimeoutTicks Tag	Integer	Requerido	Especifica cuánto tiempo esperará el núcleo de comunicación de la serie C para ver el ~DONE la línea tan baja antes de completar con un error de Tiempo de espera agotado. El rango para esta etiqueta es 0–4294967294. Este valor se especifica en unidades de tics de reloj base de 25 ns.

Tabla 16. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	DigitalLines Section	—	Opcional	Enumera las líneas de E / S digitales del bus CompactRIO que serán utilizadas por los VI de soporte de módulo para realizar el DIO básico.
4	DIO0–DIO8 Tags	Enumerated	Opcional	Especifica la función de las líneas DIO0 – DIO8. Las opciones para estas etiquetas son: • BiDireccional: la línea toma como valor predeterminado una entrada en la inserción del módulo y también se puede usar como salida digital. • ConstantOutputHigh: la línea predeterminada es una salida ALTA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran. • ConstantOutputLow: la línea predeterminada es una salida BAJA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran. • InputOnly: la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital. • Unused: la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital. Si una línea DIO está reservada por SPI Engine, Convert Pulse logic o Done Wait logic, es posible que no aparezca en la sección DigitalLines. Si una línea DIO no está reservada o especificada en la sección DigitalLines, se establecerá de forma predeterminada en No utilizado.
2	AuxiliaryCommunication Section	—	Opcional	—

Tabla 17. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
3	SPIConfiguration Section	—	Opcional	Reserva las siguientes líneas de bus CompactRIO para el motor SPI en modo auxiliar: SPI_CLK, ~ SPI_CS, MISO, MOSI. Cuando está reservado por el motor SPI, estas líneas no se pueden usar para E / S digital en modo Auxiliar. Esta sección debe estar presente para poder utilizar las interfaces SPI del núcleo de comunicación de la serie C cuando el módulo está en el modo de operación normal.
4	SPIHalfTauTicks Tag	Integer	Requerido	Especifica qué tan ancho es el período de la mitad de Tau del reloj SPI en tics de reloj base de 25 ns. El rango para esta etiqueta es 2–65535. Si su módulo tiene un SPI Tau de 150 ns, la etiqueta SPIHalfTauTicks se establecerá en el período del reloj SPI de 150 ns y será de 6 tics de reloj base de ancho. La señal del reloj SPI debe ser simétrica, por lo que tendrá 3 tics altos y 3 tics bajos.
3	DigitalLines Section	—	Opcional	Enumera las líneas de E / S digitales del bus CompactRIO que serán utilizadas por los VI de soporte de módulo para realizar el DIO básico.

4	DIO0–DIO8 Tags	Enumerated	Opcional	Especifica la función de las líneas DIO0 – DIO8. Las opciones para estas etiquetas son: BiDireccional: la línea toma como valor predeterminado una entrada en la inserción del módulo y también se puede usar como salida digital. ConstantOutputHigh: la línea predeterminada es una salida ALTA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran. ConstantOutputLow: la línea predeterminada es una salida BAJA en la inserción del módulo. Las operaciones de salida digital en esta línea se ignoran. InputOnly: la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital. Unused: la línea toma como valor predeterminado una entrada en la inserción del módulo y no se puede utilizar como salida digital.

Tabla 18. ModuleSupport XML

XML Level	XML Tag	Data Type	Required/ Optional	Description
				Si una línea DIO está reservada por el motor SPI, es posible que no aparezca en la sección DigitalLines. La Especificación de la Serie C dicta que la línea FUNC se usa para poner el módulo en el modo de Comunicación Auxiliar. Esto significa que la línea FUNC está reservada y no se puede utilizar como DIO en el modo Auxiliar. Si una línea DIO no está reservada o especificada en la sección DigitalLines, se establecerá de forma predeterminada en No utilizado.

2	DigitalLineInfo Section	—	Opcional	Especifica la configuración avanzada en las interfaces DIO del núcleo de comunicación de la serie C.
3	SCTLOutputSyncRegs Section	—	—	Especifica cuántos registros de sincronización se colocan en el FPGA entre el nodo de E / S de salida digital en el diagrama de bloques y el pin del bus CompactRIO.
4	DIO0–DIO8 Tags	Enumerated	Opcional	Especifica el valor que establece el número de registros de salida para esa línea DIO. Las opciones para estas etiquetas son 0 y 1. Si una línea no aparece en la sección SCTLOutputSyncRegs, se establecerá de manera predeterminada en 1 registro de sincronización.
3	Arbitration Section	—	—	Especifica cómo el FPGA arbitrará entre múltiples nodos de E / S de salida digital que se colocan en el diagrama de bloques.
4	DIO0–DIO8 Tags	Enumerated	Opcional	Especifica el valor que establece la opción de arbitraje para esa línea DIO. Las opciones para estas etiquetas son NeverArbitrate, AlwaysArbitrate y ArbitrateIfMultipleRequestorsOnly. Si una línea no aparece en la sección de Arbitraje, se establecerá de forma predeterminada en NeverArbitrate.

A.2 Características de un módulo C

Entre las características que debe contar el hardware del módulo están las siguientes:

- Dimensiones:

El circuito impreso (PCB) del módulo debe tener un grosor de 1,57 mm (0,062 pulg.) y debe tener unas dimensiones exteriores de 73,38 x 66,04 mm (2,888 x 2,600 in). Además de que el conector DSUB hembra de 15 pines debe estar ubicado en una posición exacta que se muestra en la siguiente imagen:

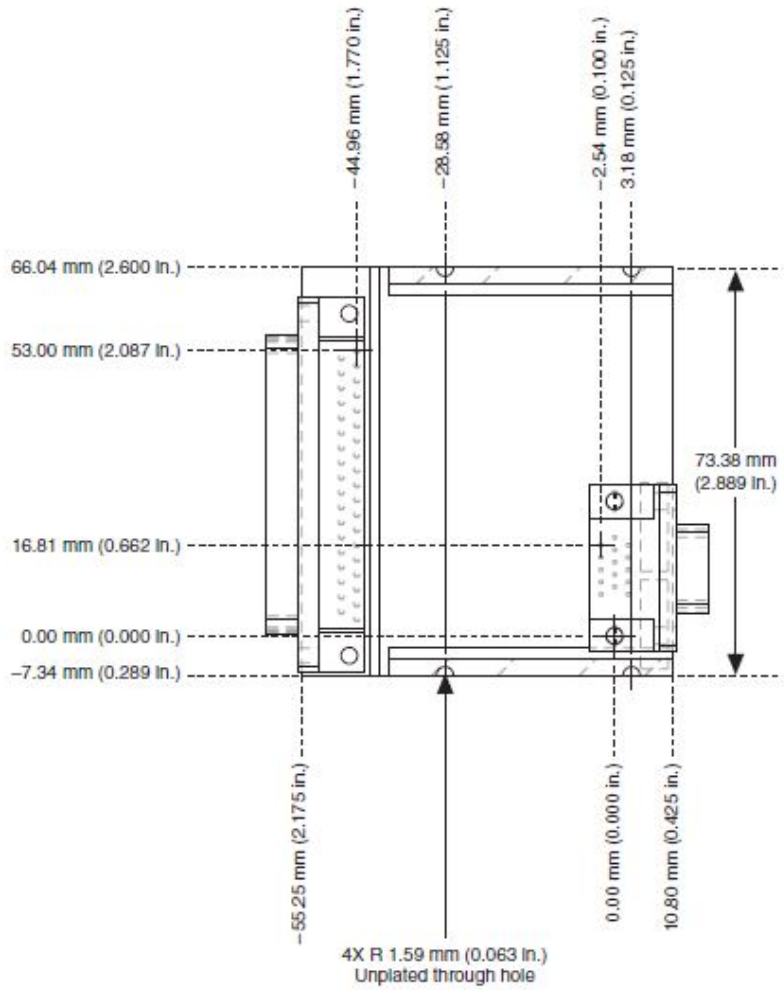


Figura 105. *Ejemplo de las dimensiones de un módulo C con un conector DSUB 37 pines*

(National Instruments Corporation, 2011)

Además, se debe tener unas zonas de retención (libre de componentes eléctricos) donde el aluminio del módulo está en contacto con el PCB, como se muestra en la siguiente imagen:

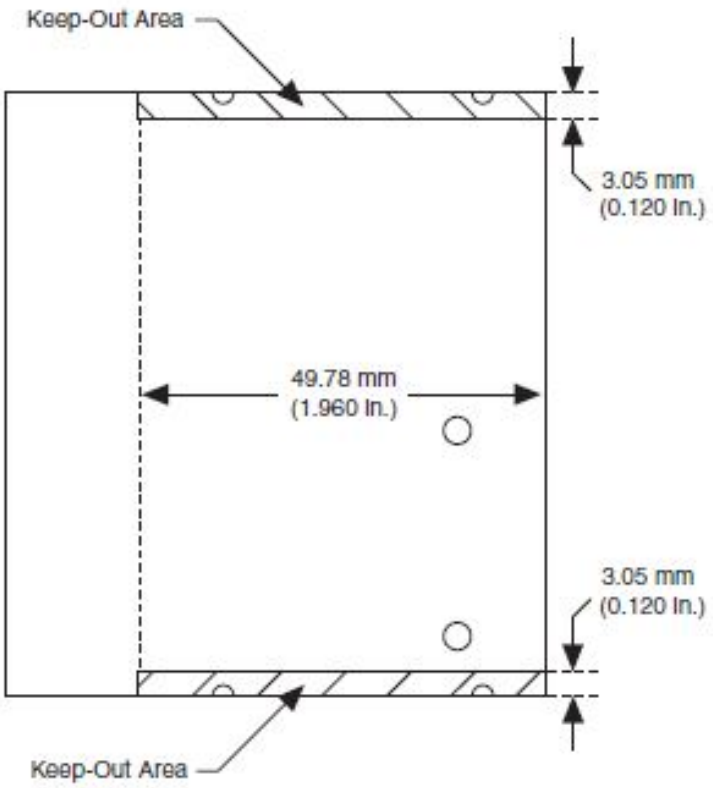


Figura 106. *Áreas de retención para el módulo*

(National Instruments Corporation, 2011)

- Estándares de seguridad:

Para mantener la coherencia con los módulos de la Serie C, NI requiere que los módulos personalizados cumplan con los requisitos de los siguientes estándares de seguridad para equipos eléctricos de medición, control, y uso de laboratorio:

IEC 61010-1, EN 61010-1

UL 61010-1, CSA 61010-1

En cuanto a la compatibilidad electromagnética se recomienda que cumplan con los requisitos de las siguientes normas EMC para equipos eléctricos para medición, control y uso en el laboratorio:

EN 61326 (IEC 61326): Class A emissions; Industrial immunity

EN 55011 (CISPR 11): Group 1, Class A emissions

AS/NZS CISPR 11: Group 1, Class A emissions

FCC 47 CFR Part 15B: Class A emissions

ICES-001: Class A emissions

En cuanto a choque se debe contar con una carcasa o panel que proteja los equipos electrónicos:

Figura 107. *Carcasa de Módulo de la Serie C*

(National Instruments Corporation, 2011)

En cuanto a vibración de funcionamiento debe están en los siguientes rangos:

- Aleatorio (IEC 60068-2-64)
- Sinusoidal (IEC 60068-2-6)
- Impacto (IEC 60068-2-27)

Finalmente, algunas especificaciones ambientales son:

I. Temperatura de funcionamiento: (IEC 60068-2-1, IEC 60068-2-2) … 40 a 70 °C

II. Temperatura de almacenamiento: (IEC 60068-2-1, IEC 60068-2-2) … 40 a 85 °C

III. Humedad de funcionamiento: (IEC 60068-2-56) ... 10 a 90% RH

IV. Humedad de almacenamiento (IEC 60068-2-56) … 5 a 95% RH

V. Altitud máxima: 2,000 m

Todas estas características de hardware son las que NI recomienda para que cualquier modulo C pueda funcionar perfectamente en sus cRIO, además de permitir que la misma compañía pueda avalar el funcionamiento de tu módulo para la comercialización del mismo con su sello de garantía.

Para más información se puede consultar a través de sus diversos medios de comunicación a National Instruments Corparation o su manual del hardware para la creación de módulos C titulado:

- cRIO_MDK_Hardware_User_Manual