UNIVERSIDAD SAN FRANCISCO DE QUITO

Colegio de Ciencias e Ingeniería

Control de un generador de pulsos, una fuente de voltaje-corriente y matriz por medio del protocolo GPIB programado en Python

Mario Sebastián Celi Calderón

Daniel Fellig, M.Sc., Director de Tesis

Tesis de grado presentada como requisito para la obtención del título de Ingeniero en Sistemas

Quito, diciembre de 2013

Universidad San Francisco de Quito
Colegio de Ciencias e Ingeniería

HOJA DE APROBACION DE TESIS

Control de un generador de pulsos, una fuente de voltaje-corriente y matriz por medio del protocolo GPIB programado en Python

Mario Sebastián Celi Calderón

Daniel Fellig, M.Sc.
Director de Tesis ....

Fausto Pasmay, M.Sc.
Miembro del Comité de Tesis ....

Daniel Fellig, M.Sc.
Miembro del Comité de Tesis ....

Luis Miguel Procel, M.Sc.
Miembro del Comité de Tesis ....

Ximena Córdova, Ph.D.
Decana de la Escuela de Ingeniería ....
Colegio de Ciencias e Ingeniería

Quito, diciembre de 2013

Firma:

Nombre: Mario Sebastián Celi Calderón

C. I.: 1715309132

Fecha: Quito, diciembre de 2013

RESUMEN

El presente proyecto de tesis trata de la creación de dos aplicaciones en el lenguaje de programación Python, para controlar dos equipos del laboratorio de ingeniería electrónica de la Universidad San Francisco de Quito, y la configuración de un tercer equipo usando las herramientas de software del fabricante.

Las dos aplicaciones desarrolladas controlarán una fuente de corriente – voltaje y un generador de ondas, dos de los equipos presentes en el laboratorio de ingeniería electrónica. Para la aplicación de la fuente de corriente – voltaje se implementaron tres funcionalidades, configuración de valores fijos en el equipo, barrido de corriente o voltaje y barrido de corriente y voltaje de tipo maestro – esclavo. En la aplicación para el generador de ondas se implementaron las funcionalidades de configurar valores fijos en el dispositivo, hacer un barrido de frecuencia linear o logarítmico y generar ondas arbitrarias con el dispositivo

El tercer equipo es una matriz de conmutación de circuitos, que será configurada para funcionar en conjunto con un cuarto equipo, un caracterizador de semiconductores. La interacción entre los dos equipos será controlada desde la interfaz gráfica del caracterizador de semiconductores usando la aplicación y entorno de pruebas Keithley Interactive Test Environment (KITE).

ABSTRACT

The present thesis project consists in the development of two applications in the Python programming language, to control two devices of laboratory of electronic engineering laboratory of Universidad San Francisco de Quito, and the configuration of a third equipment using the manufacturer's software tools.

The two applications developed will control am electrical power supply and a wave generator, two of the equipment present in the electronic engineering laboratory. For the power supply application, three features were implemented, configure fixed values in the device, a current or voltage sweep and a master – slave current – voltage sweep. The application for the wave generator features setting up fixed values in the device, a linear or logarithmic frequency sweep and an arbitrary wave form generator tool.

The third equipment is a switching matrix, which will be configured to work in conjunction with a fourth equipment, a semiconductor characterization system. The interaction between the two devices will be controlled from the GUI of the semiconductor characterization system using the application and test environment Keithley Interactive Test Environment (KITE).

Tabla de Contenidos

RESUMEN 5
ABSTRACT 6
Figuras 9
Tablas 10
Capítulo 1 – Introducción 11
Antecedentes 11
Instrumentos Usados Para el Desarrollo de la Tesis 12
Agilent E3634A, Fuente de Poder 12
Agilent 33250A, Generador de Ondas 13
Keithley 707B, Matriz de Conmutación 15
Keithley 4200-SCS, Sistema Caracterizador de Semiconductores 16
Justificación del Proyecto 17
Objetivo General 19
Objetivos Específicos 19
Metas 20
Actividades 20
Capítulo 2 – Marco Teórico 23
Tecnología a Ser Utilizada 23
Lenguajes de Programación 23
Protocolos de Comunicación con Dispositivos 24
Librerías de Software 26
Metodología 26
Revisión Literaria 28
Capítulo 3 – Planificación e Implementación 31
Desarrollo de aplicaciones en Python 31
Aplicación Desarrollada Para la Fuente de Poder 34
Clase Wrapper 37
Clase Action 38
Clase Sweeper 38
Clase Definitions 38
Clase ValueException 39
Aplicación Desarrollada Para el Generador de Ondas 39
Clase Wrapper 42
Clase Device 43
Clase Definitions 43
Configuración de Matriz de Conmutación de Circuitos 43
Conexión con Caracterizador de Semiconductores 43

Manejo de Librerías en el Caracterizador de Semiconductores 47
Configuración del Ambiente de Pruebas KITE 49
Capítulo 4 – Análisis de Resultados 54
Descripción de los Resultados 54
Objetivo 1 55
Objetivo 2 59
Objetivo 3 63
Objetivo 4 65
Retroalimentación de los Usuarios 65
Capítulo 5 67
Conclusiones 67
Recomendaciones 69
Desarrollo Futuro 70
Referencias 73
Anexo 1: Detalle de Aplicación Para Fuente de Poder 76
Clase Wrapper 76
Clase Action 81
Clase Sweeper 83
Clase Definitions 85
Clase ValueException 85
Anexo 2: Detalle de Aplicación Para Generador de Ondas 86
Clase Wrapper 86
Clase Device 87
Clase Definitions 89
Anexo 3: Manual de Instalación Aplicaciones Para Fuente de Poder y Generador de Ondas 90
Paso 1 90
Paso 2 90
Paso 3 91
Paso 4 92
Paso 5 92

Figuras

Figura 1: MOSFET 18
Figura 2: Diagrama Componentes Fuente de Poder 32
Figura 3: Diagrama de Flujo Para Inicio de Aplicación 32
Figura 4: Código para iniciar el programa 33
Figura 5: Diagrama de Casos de Uso Fuente de Poder 35
Figura 6: Diagrama de clase Fuente de Poder 37
Figura 7: Diagrama de Casos de Uso Generador de Ondas 39
Figura 8: Diagrama de Clase Generador de Ondas 42
Figura 9: KCON 45
Figura 10: KCON 46
Figura 11: KCON 47
Figura 12: KULT 48
Figura 13: KITE 49
Figura 14: KITE 50
Figura 15: KITE 51
Figura 16: KITE 51
Figura 17: Diagrama de Casos de Uso KITE 52
Figura 18: KITE Configuración Final 53
Figura 19: Barrido de corriente 58
Figura 20: Barrido maestro esclavo 59
Figura 21: Configuración Estática 60
Figura 22: Onda Arbitraria 61
Figura 23: Onda Arbitraria Error 62
Figura 24: Barrido de Frecuencia 63

Tablas

Tabla 1: Metas Alcanzadas.. 55

Capítulo 1 – Introducción

Antecedentes

Es muy común que en laboratorios de física o ingeniería electrónica se necesite alguna clase de complemento de software para manejar los dispositivos que podemos encontrar en ellos. Existen ya herramientas de software como LabVIEW que permiten realizar una gran cantidad de tareas genéricas para varios de estos dispositivos, tales como fijar valores estáticos en los dispositivos, realizar mediciones de valores en los terminales del equipo, e inclusive se puede programar módulos cuya funcionalidad puede ser tan avanzada como lo permita el API específico para un dispositivo (Elliott, Vijayakumar, Zink, Hansen, 2007). Muchas necesidades en un laboratorio pueden ser resueltas por medio de esta herramienta o una similar, pero cuando se tienen requerimientos muy específicos o se trabaja con dispositivos muy complejos, es necesario desarrollar una aplicación a más bajo nivel. Cada uno de los dispositivos con los que se va a trabajar, proveen las herramientas necesarias de comunicación al más bajo nivel y esto da mucha flexibilidad en el desarrollo de una aplicación.

En cuanto a conectividad, herramientas como LabVIEW también facilitan la comunicación con los dispositivos, pero los mismos fabricantes de los equipos nos brindan otra serie de herramientas y documentación, con el fin de que el usuario genere sus propias herramientas para manejar los equipos, y esta es la intención que se tiene con el desarrollo de estas aplicaciones.

Fabricantes de instrumentación de laboratorio como *Agilent Technologies*, *National Instruments* y *Keithley Instruments*, son un buen ejemplo de fabricantes que entregan documentación y ciertas herramientas de software para facilitar el desarrollo de diferentes aplicativos que se ajustan a las necesidades de sus usuarios. Es importante notar que la mayoría de estos instrumentos utilizan librerías de software estándar para la comunicación

entre el equipo y la computadora personal que lo va a controlar. Cada fabricante puede tener su propia implementación de la librería de software, pero al basarse en estándares es posible controlar a todos usando la misma librería en algunos casos.

Puede que el fabricante proporcione las herramientas necesarias para aprovechar todas las funcionalidades del equipo, pero es responsabilidad del usuario configurar, conectar y de ser necesario desarrollar aplicaciones para el equipo. De esta manera se pueden integrar todos los componentes, para presentar una interfaz al usuario que tenga solamente el nivel de complejidad que pueda manejar un usuario específico, es decir, haciéndola tan simple o complicada como sea necesario. El desarrollo de esta tesis muestra cómo controlar equipos de laboratorio desde una interfaz gráfica, pero detrás de esta, tenemos la implementación de métodos y rutinas que se comunican con el equipo a un más bajo nivel.

El departamento de ingeniería electrónica de la Universidad San Francisco de Quito tiene a su disposición instrumentación de laboratorio muy avanzada y variada. Dentro de esta instrumentación tenemos equipos de uso diario para clases y laboratorios en los que participan los estudiantes con carreras afines a la materia. Pero también, podemos encontrar en este laboratorio instrumentación más avanzada y sólo disponible para aquellos que hacen investigación, como estudiantes de tesis y los profesores del departamento. Estos instrumentos son precisamente los más avanzados y más costosos, razón por la cual se les quisiera aprovechar en su totalidad.

Instrumentos Usados Para el Desarrollo de la Tesis

Agilent E3634A, Fuente de Poder

El Agilent E3634A es una fuente de poder DC, en este documento será llamado fuente de poder como referencia.

Este equipo tiene la capacidad de generar una corriente o voltaje dentro de los rangos permitidos, es decir desde 0 hasta 51.5 voltios y desde 0 a 7.21 amperios. En realidad, este equipo cuenta con dos fuentes de poder diferentes. Una, cuyo voltaje máximo es 25.75V y 7.21 amperios, y otra cuyo voltaje máximo es 51.5 voltios y 4.12 amperios (Agilent E3633A and E3634A DC Power Supplies User's Guide, 2012). Se puede seleccionar cualquiera de estas dos fuentes desde el panel frontal del dispositivo y es tan simple como presionar un botón. Este manejo de las dos fuentes es transparente desde la interfaz de la aplicación, automáticamente seleccionara la indicada dependiendo de los valores que el usuario ingrese.

Existen diferentes modos de operación para el dispositivo, son los siguientes:

- Voltaje Constante (Constant Voltage - CV)
- Corriente Constante (Constant Current - CC)

Para estos modos de operación se puede configurar desde el panel frontal una protección contra sobre voltaje o sobre corriente, Overvoltage protection (OVP) y overcurrent protection (OCP). Esta funcionalidad configura un límite de corriente o voltaje al que se puede llegar antes de que la fuente configure los valores mínimos de corriente y voltaje.

El resto de funcionalidad, como barridos de corriente y voltaje, no están disponibles desde el panel frontal de control del dispositivo, así que sólo se puede tener control sobre estas funciones de manera manual y enviando comandos a través de la interfaz remota GPIB.

Agilent 33250A, Generador de Ondas

El Agilent 33250A es un generador de funciones/ondas arbitrarias de 80 MHz y en este documento será llamado un generador de ondas como referencia.

El generador de ondas es otra de las muestras que tenemos de instrumentos no específicos para realizar mediciones que fabrica Agilent.

Desde el panel frontal del equipo se tiene acceso a la mayoría de funciones que este equipo dispone, pero para hacer que este se comporte de la manera específica en la que el usuario desea, se lo debe controlar por la interfaz remota GPIB. Tareas sencillas como configurar valores estáticos en el dispositivo pueden volverse tediosas ya que los botones en el panel frontal son limitados. Especialmente para la configuración de una onda arbitraria, la aplicación facilita mucho el ingreso de valores, ya que se pueden ingresar miles de valores de amplitud para una onda llenando una matriz y usando el teclado numérico del computador.

Este dispositivo tiene la capacidad de generar ondas, o funciones, con forma sinusoidal, cuadrada y rampa. Además es capaz de generar pulsos eléctricos, ruido, y un voltaje DC al que sólo se le debe indicar la distancia en voltios desde el 0, para generar una onda (Agilent 33250A 80 MHz Function / Arbitrary Waveform Generator User's Guide, 2003). Este tipo de ondas pueden ser configuradas desde el panel frontal usando valores fijos para frecuencia y amplitud. También se puede, desde el panel frontal, configurar barridos de frecuencia y ondas arbitrarias, las dos funciones que se implementaron en la aplicación. La razón de implementar una aplicación para realizar dos funciones que pueden ser configuradas desde el mismo panel del equipo, es que se desea especificar parámetros diferentes para el barrido de frecuencia, y además la interface para el ingreso de valores de la onda arbitraria es más amigable. Además de estas funciones, el dispositivo puede realizar modulaciones de frecuencia, modulaciones de amplitud y activar el modo de ráfaga. En este documento no se analizarán esas funciones, pues no son relevantes para el desarrollo de esta aplicación.

Keithley 707B, Matriz de Conmutación

El Keithley 707B es una matriz de conmutación de circuitos, en este documento será llamada matriz de conmutación como referencia.

Este equipo es muy diferente a la fuente de poder y generador de ondas. Además de que fue fabricado por Keithley, cumple una función completamente diferente a los otros dos equipos. Esencialmente, la función de una matriz de conmutación de circuitos, es la de conectar diferentes terminales eléctricas entre ellas, sin la necesidad de conectar y desconectar cables. Para usar este equipo, sólo se deben conectar los terminales (de fuentes de poder, generadores de ondas, instrumentos de medición, etc.) en una posición fija y luego, cambiando la configuración del dispositivo, se puede conectar internamente una terminal con otra, o una con varias a la vez. Su nombre (Matriz de conmutación), se debe a que visualmente podemos organizar los terminales del equipo en una matriz. Siendo columnas algunos de los terminales, y las filas otra sección de los mismos. De esta manera podemos conectar un terminal en una fila, con todos los terminales que se encuentran en las columnas, o cualquiera de los terminales de las columnas, con todos los terminales de las filas.

Desde el panel frontal se pueden configurar parámetros del dispositivo como la dirección GPIB, dirección IP para la interfaz web e inclusive se pueden realizar conexiones de la matriz, pero se deben realizar tales conexiones con pocos botones y usando los identificadores de cada terminal. Este equipo tiene una interfaz web de configuración, a la cual se puede acceder por medio de un explorador web usando su dirección IP. Desde esta interfaz se pueden configurar las conexiones de la matriz de una manera gráfica y muy intuitiva, pero esto no es lo que requieren los usuarios del departamento de ingeniería electrónica. Otra manera de comunicarse es enviando comandos al dispositivo, ya sea por medio de la interfaz GPIB o por la misma interfaz de red. A través de la interfaz GPIB se

pueden enviar comandos directamente como texto, pero este equipo también es compactible con otros dispositivos del mismo fabricante, y en este caso queremos controlar la matriz desde un caracterizador de semiconductores, específicamente desde la aplicación y ambiente de pruebas KITE (Keithley Interactive Test Environment) luego de configurar la comunicación entre los dos dispositivos. El ambiente de pruebas KITE es una aplicación provista por el fabricante Keithley junto con el caracterizador de semiconductores

Keithley 4200-SCS, Sistema Caracterizador de Semiconductores

Este equipo es secundario en el desarrollo de esta tesis, es simplemente el medio por el cual se va a controlar a la matriz de conmutación de circuitos. Pero, es importante notar que el fin de la configuración de la matriz es que trabaje de manera conjunta con el caracterizador de semiconductores y también con otros equipos de laboratorio, de la manera en que los usuarios lo necesitan.

Este es seguramente el equipo más complejo y avanzado que se va a tratar en el documento. Su funcionalidad completa no es relevante para este desarrollo, pero si podemos nombrar algunas de sus funciones.

Este equipo funciona como una fuente de corriente y voltaje, pero su funcionalidad principal es la de realizar mediciones de semiconductores. La interfaz de control de este dispositivo es la más completa, cuenta con una pantalla integrada y una instalación propia de Windows XP, sistema operativo donde se ejecuta el principal programa de control del dispositivo KITE (Keithley Interactive Test Environment). Además de la herramienta KITE, el sistema cuenta con una serie de herramientas de configuración e inclusive de programación del dispositivo mediante la herramienta KULT (Keithley User Library Tool).

Justificación del Proyecto

Este proyecto busca no sólo facilitar el uso más especializado de equipos de laboratorio, sino unir a todos ellos para un trabajo conjunto. De esta manera se podrán realizar proyectos más ambiciosos y que puedan ser controlados y monitoreados desde una sola estación.

Las aplicaciones desarrolladas para fuente de poder y generador de ondas pueden ser ejecutadas desde el sistema operativo (Windows XP) que está instalado en el caracterizador de semiconductores, cada uno con su propia interfaz gráfica, y tanto el caracterizador de semiconductores como la matriz de conmutación pueden ser controlados desde la aplicación KITE (Keithley Interactive Test Environment), una herramienta provista con el equipo caracterizador por el mismo fabricante.

Específicamente, se espera integrar el funcionamiento de estos 4 equipos de laboratorio con el fin de realizar mediciones sobre un MOSFET (metal–oxide–semiconductor field-effect transistor), el término en inglés para un transistor de efecto de campo metal-óxido-semiconductor. En la actualidad en el laboratorio de ingeniería electrónica de la Universidad San Francisco de Quito se realizan mediciones de un MOSFET utilizando solamente el caracterizador de semiconductores, pues se deben diseñar pruebas diferentes para usar una fuente de poder o generador de ondas externo en las mediciones. La integración de estos equipos en mediciones es un proyecto a futuro que será posible en parte gracias al desarrollo de este proyecto.

En este proyecto se configurarán 4 tipos de medición desde la aplicación KITE y usando solamente la matriz de conmutación y el caracterizador de semiconductores, todas para medir características eléctricas de un MOSFET. En la Figura 1 podemos ver el diagrama de un MOSFET. Cada tipo de medición requiere que se conecte una fuente de poder o de medición específica a uno de los terminales del MOSFET (Drain, Source, Gate

y Bulk). Esto quiere decir que para cada tipo de medición sería necesario conectar los cables de una manera específica, y esta es la principal función de la matriz de conmutación. Una vez que se hayan conectado todos los cables en la terminal eléctrica que les correspondan, se realizará una conexión entre cualquiera de estos terminales directamente desde el ambiente de pruebas KITE en la interfaz gráfica del caracterizador de semiconductores.

Las 4 mediciones que se realizarán son IV, CGC, CGB y CGA. La primera permite conocer características sobre la movilidad de electrones en un transistor, el voltaje de encendido y la transconductancia. La medición CGC quiere decir capacitancia Gate to cannel (capacitancia entre el terminal Gate y los dos terminales Source y Drain), refiriéndose a las partes de un MOSFET como se puede ver en la Figura 1. La medición CGB, mide la capacitancia de tipo Gate to Bulk (capacitancia entre el terminal Gate y el terminal Bulk) y por último CGA mide la capacitancia Gate to All (capacitancia entre el terminal Gate y los otros 3 terminales del transistor) de un MOSFET. Cada una de estas configuraciones mide características específicas de un MOSFET

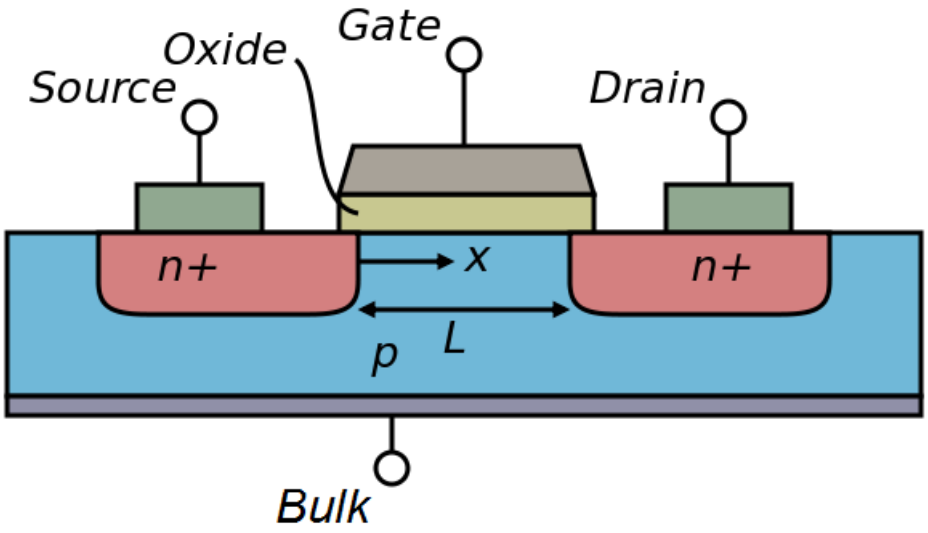

Figura 1: MOSFET

Por medio de las herramientas de software que se han sido desarrolladas en este proyecto, se espera aumentar la productividad de los equipos ya existentes en los laboratorios de ingeniería electrónica. Hay algunos equipos en este laboratorio que hasta ahora no han sido explotados en su totalidad, ya que muchas funciones pueden ser utilizadas solamente a través de su interfaz remota, y hasta ahora no se contaba con una aplicación que sea capaz de comunicarse con estos instrumentos utilizando dicha interfaz.

Específicamente en la fuente de poder se implementará 3 funcionalidades de barrido que antes no podían ser usadas, y en el generador de ondas se desea facilitar al usuario el uso de funciones que si estaban disponibles desde el panel frontal de configuración, pero eran muy difíciles o complicadas de utilizar.

Objetivo General

Permitir que los usuarios del laboratorio de ingeniería electrónica aprovechen toda la funcionalidad de sus equipos desarrollando dos aplicaciones en Python para el control de remoto de dos de ellos y configurando un tercero para ser controlado, también remotamente, desde la interfaz gráfica de un cuarto equipo de laboratorio.

Objetivos Específicos

1. Desarrollar una aplicación en Python que permita, controlar una fuente de corriente y voltaje.
2. Desarrollar una aplicación en Python que permita, controlar un generador de ondas
3. Configurar una matriz de conmutación de circuitos, para comunicarse y ser controlada desde un caracterizador de semiconductores por medio de una interfaz GPIB.

4. Configurar el equipo donde se va a correr la aplicación de fuente de poder y generador de ondas, dentro del laboratorio de ingeniería electrónica.

Metas

1. Diseñar una interfaz gráfica que permita al usuario controlar todas las funciones que han sido programadas para la fuente de poder.
2. Implementar en la aplicación desarrollada para la fuente de corriente y voltaje, las funcionalidades requeridas por el usuario.
3. Diseñar una interfaz gráfica que permita al usuario controlar todas las funciones que han sido programadas para el generador de ondas.
4. Implementar en la aplicación desarrollada para el generador de ondas, las funcionalidades requeridas por el usuario.
5. Configurar la conexión entre la matriz de conmutación y el caracterizador de semiconductores
6. Configurar la herramienta KITE para ejecutar las mediciones requeridas por los usuarios.
7. Preparar la computadora donde se va a instalar la aplicación, para que esta pueda ejecutar las aplicaciones programadas
8. Instalar los archivos de aplicación y verificar que esta funcione

Actividades

1. Usar QtDesigner para diseñar la interfaz de usuario gráficamente de las aplicaciones de fuente de poder y generador de ondas, haciendo drag and drop de los componentes gráficos.
2. Generar código en Python a partir de la interfaz creada en QtDesigner usando la aplicación pyuic (Parte de PyQt). El archivo generado interface.py es el que genera la interfaz gráfica del programa para la fuente de poder y generador de ondas.

3. Implementar la funcionalidad de conectarse al equipo, así como almacenar la dirección GPIB o alias en un archivo de configuración XML, para el generador de ondas y la fuente de poder.
4. Validar el ingreso de datos por el usuario en el programa de fuente de poder y generador de ondas. Los valores máximos y mínimos permitidos por el dispositivo se encuentran en el manual del equipo respectivo.
5. Implementar la funcionalidad de configurar valores fijos de corriente y voltaje en la fuente de poder.
6. Implementar la funcionalidad de iniciar barridos de corriente o voltaje en la fuente de poder.
7. Implementar la funcionalidad de iniciar barridos de corriente y voltaje (Maestro - Esclavo) en la fuente de poder.
8. Implementar la funcionalidad de configurar valores fijos para el generador de ondas, especificando la amplitud, frecuencia, offset y tipo de onda a ser generada.
9. Implementar la funcionalidad de iniciar barridos de frecuencia lineales o logarítmicos.
10. Implementar la funcionalidad de generar ondas arbitrarias.
11. Establecer la conexión con la matriz de conmutación desde la herramienta de configuración de equipos remotos en el caracterizador de semiconductores.
12. Configurar 4 modos de conexión en la matriz de conmutación.
13. Estructurar un proyecto en la herramienta de software KITE para realizar los 4 tipos de mediciones que necesita el usuario.
14. Instalar las librerías propietarias de los dispositivos y los controladores necesarios para el correcto funcionamiento de la aplicación.

15. Instalar Python, PyQt y PyVisa en el equipo para poder ejecutar la aplicación que se ha desarrollado.
16. Copiar los archivos generados en el computador de desarrollo al computador donde se va a ejecutar la aplicación.
17. Realizar varias pruebas para verificar el correcto funcionamiento de la aplicación en el nuevo computador.

Capítulo 2 – Marco Teórico

Tecnología a Ser Utilizada

Lenguajes de Programación

Definición

Los lenguajes de programación han evolucionado, desde el nivel más bajo de programación de hardware, hasta los más modernos lenguajes de alto nivel que podemos utilizar hoy en día. En un comienzo se daban instrucciones al computador mediante tarjetas perforadas, el programador debía escribir programas a nivel de unos y ceros que es lo único que un procesador de computador entiende. El siguiente nivel fue el lenguaje de ensamblador, en donde ya se definieron instrucciones que el procesador comprendía en unos y ceros, y su equivalente como palabras simples. De esta manera los programadores ya no perforaban unos y ceros en tarjetas, sino que escribían diferentes comandos con una lógica similar a la que se utiliza para lenguajes de alto nivel. Hoy en día, los programadores pueden utilizar estos lenguajes, que simplifica significativamente el proceso de programación. Para que esto sea posible, otro programa, llamado compilador, es el encargado de "Traducir" estas instrucciones de alto nivel, primero a código de ensamblador y luego se realiza el mismo proceso de transformación de las instrucciones a unos y ceros, el resultado final es el mismo ahora y hace décadas, la diferencia radica, en las instrucciones que entienden ahora los procesadores y las operaciones que son capaces de realizar (Pierce, 2002).

C++

Este lenguaje, desarrollado por Bjarne Stroustrup mientras trabajaba para Bell Labs, es una mejora del lenguaje C. Originalmente el lenguaje se denominó C con clases, y en 1983 toma el nombre de C++. Este lenguaje es orientado a objetos, es decir que se pueden crear clases, que tienen una estructura definida, y a partir de esta clase se instancian objetos

que son completamente independientes entre ellos, pero mantienen la misma estructura que su clase define. Este lenguaje compilado puede funcionar con llamadas a rutinas del lenguaje anterior C. Esto es útil cuando se intenta hacer llamadas a métodos o funciones de muy bajo nivel, generalmente para realizar acciones directamente sobre el hardware del sistema. Hoy en día sigue siendo un lenguaje muy utilizado, pues es seguramente el lenguaje más eficiente y de bajo nivel que por medio de librerías de software y permite realizar programas muy complejos. (Stroustrup, 1997).

Python

Python es un lenguaje de programación de alto nivel, interpretado en tiempo de ejecución. El proceso de ejecución de un programa en Python es similar a cualquier otro, es decir, primero se compilan las instrucciones de alto nivel a lenguaje de ensamblador, que en este caso se conoce como bytecode que son instrucciones propias de Python y que en tiempo de ejecución (mientras el usuario corre el programa) van a ser interpretadas por un programa interprete que está escrito en C (en la implementación más utilizada de Python). Al ser un lenguaje interpretado, el desempeño no es tan bueno como aquel de un lenguaje que en tiempo de ejecución no tiene que interpretar comandos (SANNER, 1975).

Protocolos de Comunicación con Dispositivos

Antecedentes

Los dispositivos electrónicos de procesamiento de datos siempre han gozado de cierta autonomía, salvo por la fuente de energía que en muchos casos debe ser externa. En computadores y dispositivos similares todo el procesamiento de información puede ser hecho localmente, pero ahora y desde que el internet se volvió algo tan importante en nuestras vidas estos mismos dispositivos se vieron en la necesidad de poder comunicarse entre ellos.

Ya sea por medio de una red de datos, o por medio de dispositivos conectados directamente a un ordenador, para que dos dispositivos electrónicos puedan comunicarse entre sí, deben hacerlo de tal manera que los 2 entiendan la información que están enviando y recibiendo. Por esta razón, se definen protocolos de comunicación tales como el probablemente más usado hoy en día, Ethernet, y TCP/IP que son los encargados de estandarizar la estructura de la información que se intercambia entre dispositivos conectados al internet (Bochmann, 1976).

GPIB

IEEE-488 es la especificación del protocolo GPIB, un protocolo de comunicación digital en bus de corto alcance. En 1960 el protocolo es desarrollado por Hewlett-Packard para controlar procesos automatizados en equipos de laboratorio como multímetros y analizadores lógicos. El conector tiene 24 pines, pero la comunicación se produce sobre 8 bits paralelos. Este estándar fue diseñado de tal manera que se pueden conectar hasta 15 dispositivos en un cable o bus de hasta 20 metros. El protocolo hasta el día de hoy se utiliza para instrumentos de laboratorio a pesar de que otros protocolos ya están reemplazándolo (IEEE Computer Society, 2004).

Ethernet

Este protocolo es el más utilizado en las redes de comunicaciones hoy en día. El número de dispositivos que usan este protocolo crece cada día más, no sólo para servir a clientes pequeños sino que es el protocolo de comunicación que conecta países enteros recolectando el tráfico de información de miles de millones de usuarios en todo el mundo. Este protocolo ha evolucionado desde velocidades al 1 Mbit/s hasta ahora que puede alcanzar velocidades de hasta 100 Gbit/s (IEEE Computer Society, 2008).

Librerías de Software

Librería PyVISA

Librería de Python que aumenta una capa de abstracción para la comunicación con dispositivos locales como instrumentos de laboratorio. También es la librería que nos permite emitir comandos y leer datos desde estos dispositivos.

Esta librería es universal, es decir funciona para una amplia gama de dispositivos de laboratorio, además de que trabaja sobre una variedad de protocolos como Ethernet, GPIB y USB. Lo único necesario para usar esta librería es hacer las importaciones necesarias en el código de Python y tener configurados los controladores de cada dispositivo, usando las librerías propietarias del fabricante. Para la conexión a los dispositivos PyVISA recibe como parámetros el tipo de conexión y el puerto en el que este se encuentra conectado.

La misma librería incluye módulos para presentación de datos y gráficos. La documentación de PyVISA es muy extensa en libros y manuales.

PyQt

Esta, es una implementación de la reconocida librería para desarrollo de interfaces gráficas, hecha en Python. Qt es una librería que facilita mucho el manejo de objetos que tienen una representación gráfica dentro de una interfaz de usuario. Los objetos manejan eventos y atributos que permiten programar rutinas que serán ejecutadas cuando suceda un evento, ya sea una acción externa de un usuario como un clic o un evento automático como notificaciones del sistema o temporizadores.

Metodología

En este proyecto se trabaja con el departamento de ingeniería electrónica de la Universidad San Francisco de Quito. Son ellos los que presentaron el requerimiento de un sistema como el que se va a desarrollar en este proyecto.

Como en cualquier proyecto de desarrollo donde se deben cumplir ciertos requerimientos, el primer paso es reunirse con aquellas personas que presentan los requerimientos. En esta reunión se debe obtener toda la información posible, es decir hasta el más mínimo detalles de lo que, en este caso el departamento de ingeniería electrónica, quiere para manejar sus dispositivos en el laboratorio. Es muy común que las personas que presentan los requerimientos no piensen en todo lo que necesitan en ese preciso momento, por esta razón, es importante realizar todo tipo de pregunta que podría resolver una duda más adelante en el proyecto.

Después de que se obtiene una primera idea para el proyecto, se comenzará con el desarrollo. En esta etapa ya se conoce en teoría todo lo que se quiere del programa, pero lo más seguro es que en futuras reuniones se realicen pequeños cambios en cosas que no fueron consideradas en reuniones anteriores. El proceso de desarrollo de software es, si se trabaja solo, una tarea de mucha dedicación y que involucra mucha investigación sobre la marcha. Luego de haber obtenido el conocimiento básico y practicado con las librerías de python, ya se puede comenzar a diseñar la interfaz gráfica, esta interfaz es lo que se va a poder apreciar como producto final. El primer diseño de la interfaz puede y será no totalmente funcional, es decir que todos los componentes gráficos (botones, campos de texto, etiquetas, etc.) se verán tal como si el programa estuviese terminado, pero en realidad muchos de estos componentes no harán nada cuando se haga clic sobre ellos (Seely Brown, Duguid, 1996).

Una vez que se tiene una idea de cómo va a manejar el usuario la interfaz del programa se debe implementar la lógica interna del mismo, es decir, toda la programación que responde a los eventos generados por la interfaz gráfica previamente diseñada. Esta es probablemente la etapa más larga y complicada del desarrollo. Detrás de todo lo que puede ver el usuario final, está trabajando mucho código de programación que se encarga de

comunicarse a bajo nivel con los dispositivos de laboratorio, procesar la información ingresada por los usuarios, actualizar la interfaz gráfica de acuerdo a las acciones que se tomen. En esta etapa, también se modificará la interfaz de acuerdo sea necesario, es muy poco común que un diseño inicial permanezca igual hasta el final del proyecto. Al ser esta etapa de desarrollo muy prolongada, se realizarán pruebas de navegación y de uso constantemente mientras se siga desarrollando la aplicación.

Una vez terminado el desarrollo, y luego de haber realizado todas las pruebas necesarias, se puede proceder a que los usuarios principales prueben la aplicación. Esta etapa es importante, pues desde el punto de vista del programador muchas cosas son claras y esto no es necesariamente igual para los usuarios que no desarrollaron la aplicación. Es importante que el usuario final pueda utilizar intuitivamente el programa. Luego de muchas pruebas se recolecta toda la información que los usuarios entregan con respecto al uso de la aplicación. Con esta información adicional se realizan cambios adicionales en la aplicación para que responda de mejor manera a las necesidades de los usuarios, y se puede repetir este proceso hasta que el usuario final esté satisfecho.

Inclusive después de que se ha terminado la aplicación y se la comienza a usar, lo más probable es que más adelante se deban hacer cambios a la aplicación, pues no es fácil darse cuenta de todas las cosas que se desean cambiar inmediatamente, sólo con el uso se podrá detectar todo aquello que debe ser mejorado o reparado.

Revisión Literaria

Para el desarrollo de esta aplicación de control se va a utilizar el lenguaje de programación Python, que es un lenguaje de alto nivel que recientemente ha sobresalido sobre otros lenguajes. Este es un lenguaje interpretado, es decir, que el código que escribimos no se compila, sino que en tiempo de ejecución un proceso intérprete

transforma nuestras instrucciones en otras de más bajo nivel. M. F. SANNER discute en su ensayo PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT, sobre qué tan cierto es que los lenguajes interpretados tiene un menor rendimiento que aquellos que son compilados. En general esto es cierto, si tiene menor rendimiento que una aplicación escrita en C o C++ por ejemplo, pero el rendimiento es bastante aceptable en Python de todas maneras y provee de muchas herramientas para un desarrollo avanzado de software en donde generalmente el rendimiento no es un inconveniente, el usuario no va a notar la diferencia (SANNER, 1975). Por esta y muchas otras razones, Python es un lenguaje muy utilizado entre programadores y es el lenguaje se utilizará para este proyecto.

El desarrollo de aplicaciones en ciencia es muy utilizado hoy en día. Ahora los programadores tienen herramientas a su disposición, Meloni, en su artículo Computational Materials Science application programming interface (CMSapi): a tool for developing applications for atomistic simulations, muestra una librería específica para desarrollo de aplicaciones orientadas a laboratorios. En el proyecto se usarán varias librerías, pero ninguna específica para desarrollo de aplicaciones científicas; algunas serán específicas de los dispositivos que se manejarán (Meloni, Rosati, Federico, Ferraro, Mattoni, Colombo, 2005). Existen también otras herramientas que permiten el control de estos equipos de laboratorio, pero al usar el protocolo GPIB para controlar los equipos, muchos comandos son específicos de estos (Sokoloff, 2002). Para desarrollar la aplicación se utilizará la documentación del dispositivo, los controladores y algunas librerías que el fabricante provee para cada uno de ellos.

La estructura del proyecto es comúnmente usada para desarrollo de software en laboratorios. Se define un computador host que va a ejecutar la aplicación y presentar la interfaz gráfica al usuario. A este computador se encuentran conectados los equipos de

laboratorio que se desea manejar y son controlados a bajo nivel por controladores y procesos propios del dispositivo, no de la aplicación que se va a desarrollar (Kodosky, 2001).

Además de la interfaz GPIB, se puede implementar la comunicación de los dispositivos sobre un protocolo estándar de red como Ethernet. Inclusive se puede implementar una red virtual de dispositivos GPIB sobre un BUS de datos (Rawnsley, 1997). En este proyecto se va a utilizar los 2 acercamientos, programar directamente dispositivos por su puerto GPIB y al menos uno de los dispositivos por el protocolo de red Ethernet. De todas maneras, todo el control de los dispositivos va a estar centralizado en un computador host, sin importar como se comunique cada uno de ellos. Una característica importante de las redes hoy en día es que a través del internet podemos manejar cualquier dispositivo remotamente, el único requisito es que esté conectado al internet. Lo mismo se aplica para instrumentos de laboratorio, se podría tener acceso al computador host y realizar peticiones a los dispositivos específicos (Gallardo, Toral, Duran, 2006).

Capítulo 3 – Planificación e Implementación

Desarrollo de aplicaciones en Python

Para el desarrollo de las dos aplicaciones (Fuente de poder y generador de ondas), se utilizó el lenguaje de programación Python en su versión 2.7, como ambiente de desarrollo o IDE (Integrated Development Environment) Eclipse, con el plugin para desarrollo de aplicaciones en Python PyDev.

Se mantuvo el proyecto bajo control de versiones usando el manejador de código fuente Git. El repositorio es público y se mantiene en línea en el sitio GitHub (https://github.com). La ruta al repositorio es https://github.com/mcelicalderon/Tesis. En el repositorio se pueden ver todos los cambios que se han realizado en el código de los dos programas desde la primera vez que se subió un cambio al sitio, también se puede descargar el código para usarlo y además, este fue el mecanismo por el que se compartió el código y actualizaciones con el usuario, en la etapa final de desarrollo.

Se utilizó la arquitectura de software MVC (Model-View-Controller) para el desarrollo de las dos aplicaciones. Cada aplicación cuenta con un archivo interface.py que contiene la clase Ui_MainWindow, clase que contiene todos los componentes gráficos que se van a mostrar en la ventana principal de la aplicación. Se utiliza la librería de interfaz gráfica PyQt para el desarrollo de estas interfaces.

En la Figura 2 se puede ver el diagrama de componentes para la aplicación de fuente de poder, la estructura a nivel de componentes de la aplicación para el generador de ondas es muy similar.

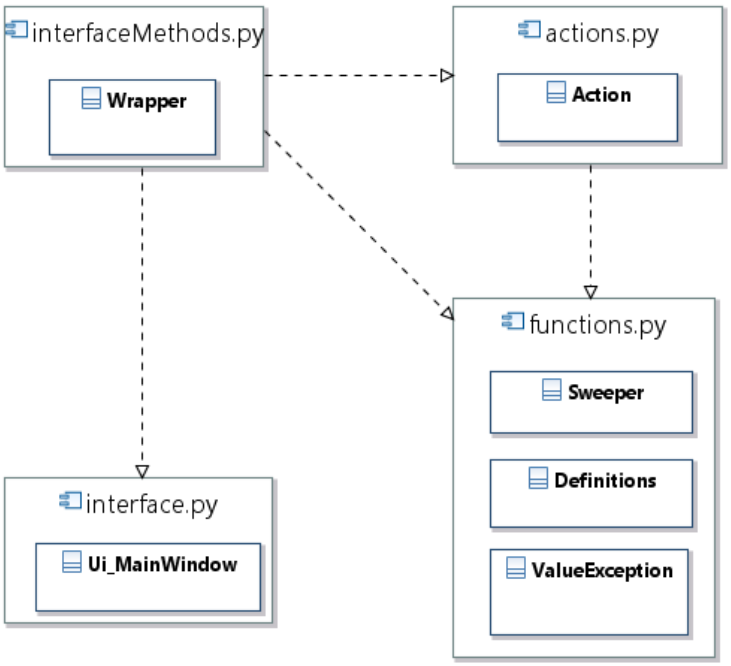

Figura 2: Diagrama Componentes Fuente de Poder

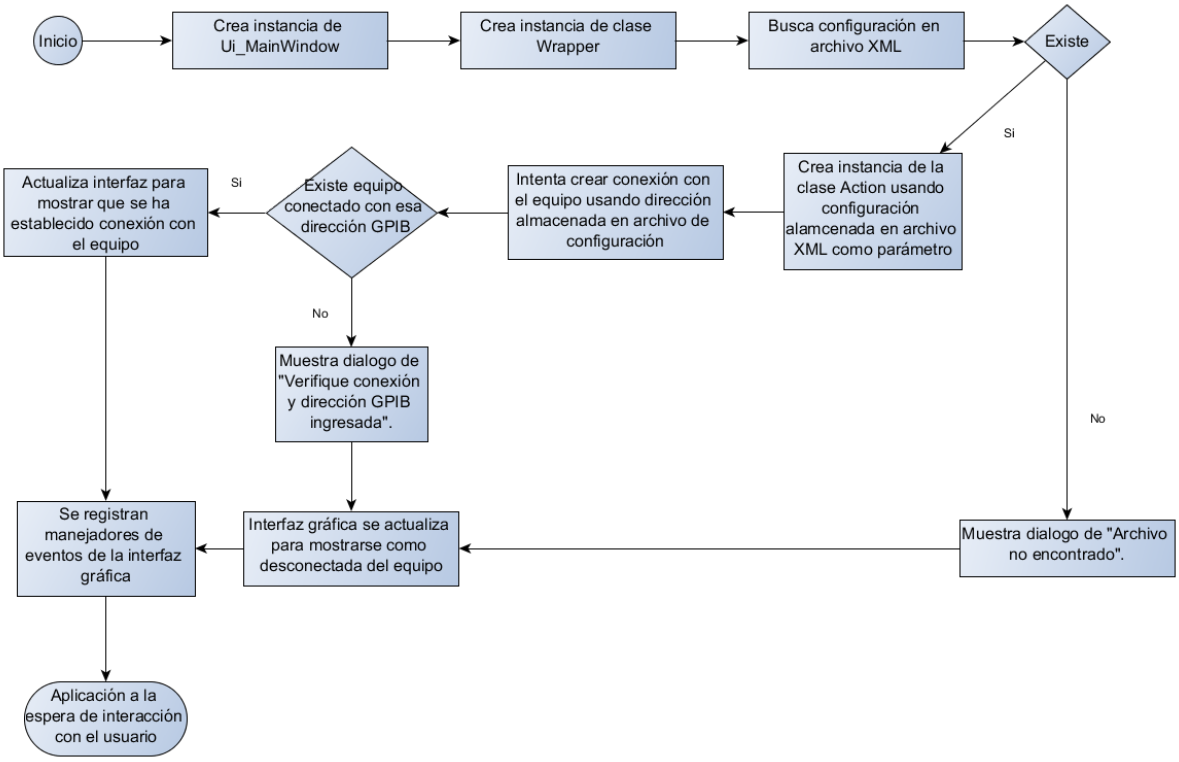

Figura 3: Diagrama de Flujo Para Inicio de Aplicación

Para los dos programas el procedimiento de desarrollo es similar. Primero se crea un diseño usando el programa QtDesigner, que mediante un archivo XML que genera, define la estructura, aspecto y ubicación de todos los componentes gráficos en la pantalla.

A partir de este archivo, y por medio de la aplicación *pyuic*, se genera el archivo interface.py, el equivalente en código Python al archivo generado en XML por el programa QtDesigner. Visualmente, el diseño creado en QtDesigner y la interfaz generada a partir del archivo interface.py en ejecución es exacta. En ocasiones existen ciertos detalles muy pequeños y corregibles que podrían presentarse de manera diferente a lo diseñado cuando la aplicación se está ejecutando. La sintaxis para generar el archivo de interfaz interface.py a partir de un archivo interface.ui de QtDesigner es la siguiente: *pyuic.py interface.ui > interface.py* ejecutándolo desde la línea de comandos de Windows.

Una vez generado el archivo con código en Python interface.py, ya se puede llamar a cada uno de los componentes desde el controlador del programa interfaceMethods.py (se debe pasar una referencia de la interfaz de la clase Ui_MainWindow). A continuación se muestra el fragmento de código para iniciar el programa.

```
if __name__ == "__main__":
   import sys
   app =
QtGui.QApplication(sys.argv)
   MainWindow = QtGui.QMainWindow()
   ui = Ui_MainWindow()
   ui.setupUi(MainWindow)
   MainWindow.show()
   wrapper = Wrapper(ui)
   wrapper.setListeners(ui)
```

Figura 4: Código para iniciar el programa

Las dos aplicaciones (fuente de poder y generador de ondas), comparten ciertas funcionalidades esenciales. Para las dos aplicaciones se usó el mismo código para conectarse al equipo y para encender y apagar el output (corriente en los terminales del equipo) de los respectivos dispositivos. En la Figura 4 vemos donde se crea una instancia de la clase Wrapper, en el constructor se asigna la referencia a la interfaz ui y además se realiza la conexión al dispositivo. El mismo constructor llama al método *createInstrument*

donde se prueba si el archivo de configuración donde se almacena la dirección GPIB del equipo existe. Si es que este archivo existe, se extrae la dirección GPIB o alias almacenado y se intenta conectar al dispositivo. De no existir este archivo, se muestra un mensaje que pide al usuario ingresar la dirección GPIB o alias del dispositivo conectado. Esta dirección debe ser configurada directamente desde el panel frontal del dispositivo, o el alias, desde el programa de administración de dispositivos de Agilent, Connection Expert. Una vez ingresada la dirección GPIB o alias, se presiona el botón “Conectar”. Cuando se presiona este botón, se intenta conectar al dispositivo usando la dirección o alias provisto por el usuario, si la conexión es exitosa, se cambia el estado del dispositivo a conectado y se almacena en el archivo de configuración la dirección o alias que se utilizó para la conexión. En la Figura 3 se muestra el diagrama de flujo para la ejecución inicial de la aplicación.

La interfaz gráfica del programa para los dos dispositivos también cuenta con un botón para cambiar el output del dispositivo (corriente en los terminales eléctricos) entre ON (encendido) y OFF (apagado). El output de los dispositivos se puede cambiar en cualquier momento, una vez que se haya realizado la conexión exitosamente, presionando el botón ON/OFF.

Aplicación Desarrollada Para la Fuente de Poder

La fuente de poder debe realizar 3 funciones. Fijar una corriente o voltaje fijo, especificado por el usuario dentro de los rangos permitidos. Hacer un barrido de corriente o voltaje (voltaje fijo y corriente variable, o corriente fija y voltaje variable), donde se valida que todos los valores ingresados estén dentro de los rangos permitidos. Y por último, se debe realizar un barrido de corriente y voltaje de tipo maestro - esclavo, donde por cada valor en el que varía el maestro (corriente o voltaje), se realiza un barrido

completo del esclavo (corriente o voltaje); el barrido termina cuando se han barrido todos los valores en el rango definido para el maestro. En la Figura 5 se muestra el diagrama de casos de uso para la aplicación desarrollada para la fuente de poder.

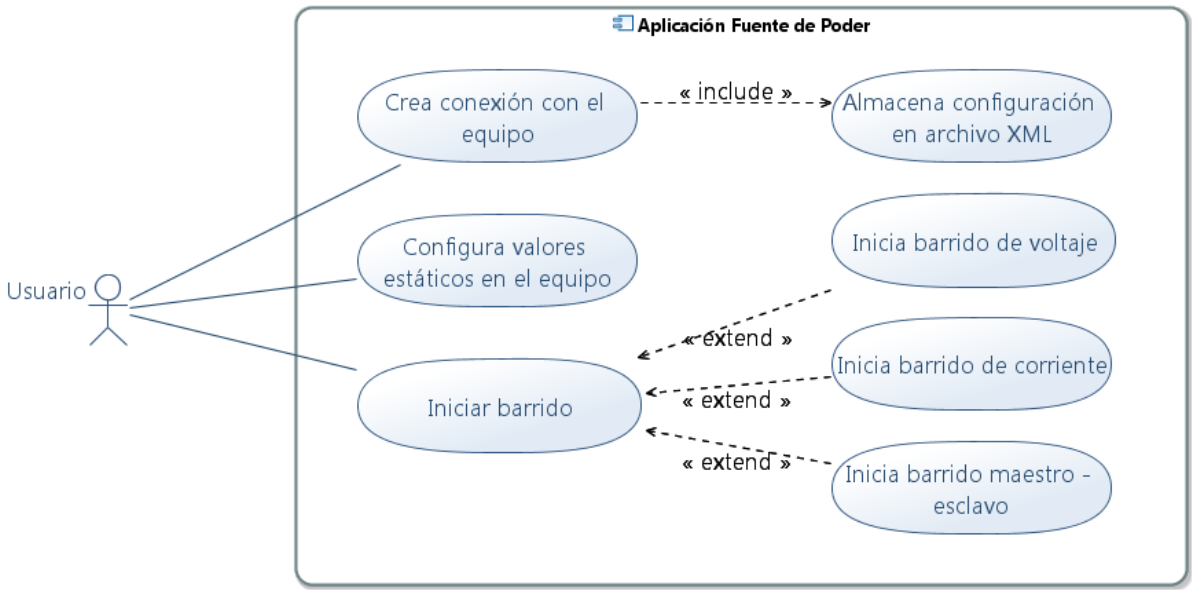

Figura 5: Diagrama de Casos de Uso Fuente de Poder

Para implementar la primera funcionalidad (configurar valores fijos de corriente y voltaje en el dispositivo), no se necesitó mucho código y este no es tan complejo. En esencia, se extraen cadenas de caracteres de los campos de texto en la interfaz gráfica, es decir los valores que se desean fijar en el equipo para corriente y voltaje, y se construye una nueva cadena de caracteres con el formato aceptado por el equipo en el estándar SCPI. Una vez que el comando está listo, se usa la clase *Instrument* de la librería VISA para enviar el comando al dispositivo; una instancia de la clase *Instrument* se crea previamente cuando se realiza la conexión al equipo. En el caso de la fuente de poder se llama a esta instancia *fPoder*, y dentro de la clase Action es este atributo y su método *write* los que se van a encargar de enviar comandos al dispositivo físico. Para crear una instancia de la clase *Instrument* se debe especificar la dirección GPIB o alias del dispositivo.

Una vez creada la instancia de *Instrument*, se puede emitir comandos usando el método *write* del mismo, y el único comando que se debe emitir para configurar un valor de voltaje y corriente fijos es *APPLy* (comando estándar para el manejo de instrumentos SCPI).

Se debe tomar en cuenta que como parte de la validación de valores ingresados, se controla que si el voltaje especificado por el usuario es mayor a 25.75 voltios, se debe cambiar el modo de operación de la fuente de poder a voltaje alto, donde el límite es 51.5 voltios y 4.12 amperios. Se utiliza el comando *"VOLTage:RANGe HIGH"* para cambiar el modo de operación, se puede realizar la acción inversa con el comando *"VOLTage:RANGe LOW"*.

La primera implementación del barrido en este equipo se realizó en la clase *Action* dentro del archivo *actions.py*. En la Figura 2 podemos ver el diagrama de componentes para la aplicación desarrollada para la fuente de poder, en este diagrama se puede apreciar cómo está estructurada la aplicación a nivel de paquetes.

En la Figura 6 se muestra el diagrama UML de clases de la aplicación para la fuente de poder, y en el Anexo 1: Detalle de Aplicación Para Fuente de Poder, se describe a detalle cada uno de los métodos que componen cada una de las clases de la aplicación desarrollada para la fuente de poder.

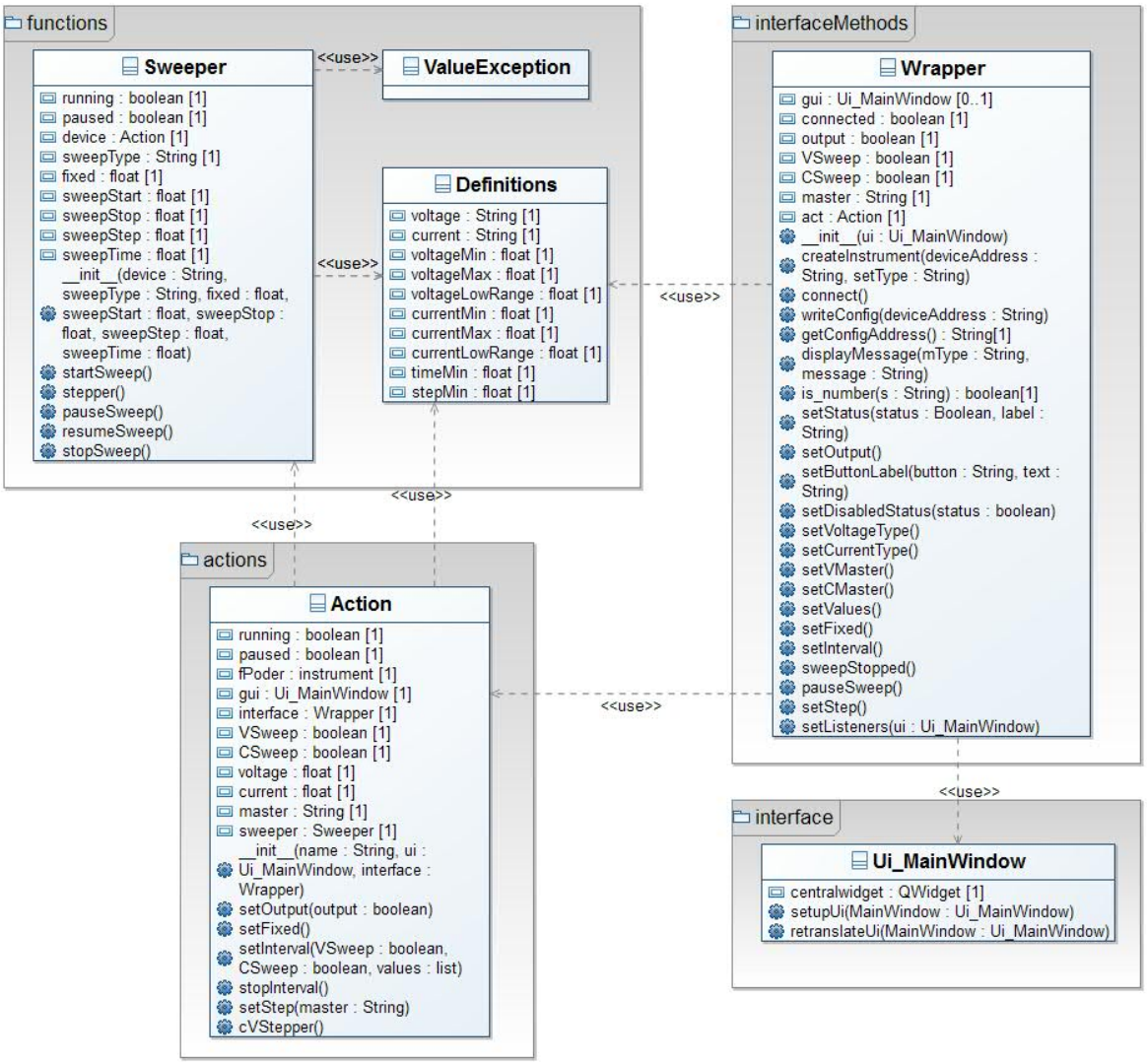

Figura 6: Diagrama de clase Fuente de Poder

Clase Wrapper

Clase que contiene los métodos para el manejo de la interfaz de usuario, es decir responde a todos los eventos generados por la misma y en donde se implementa toda la lógica que hace que la interfaz gráfica sea dinámica en tiempo de ejecución. Eventos tales como selección de un modo de operación por parte del usuario y los cambios que esto implica en la interfaz gráfica son manejados por métodos implementados en esta clase. También la validación de datos ingresados por el usuario está implementada en esta clase, así como el despliegue de mensajes de error con la razón del error que se muestra. Esta clase se encuentra en el archivo interfaceMethods.py del código fuente para la aplicación

de fuente de poder. En el Anexo 1: Detalle de Aplicación Para Fuente de Poder, se muestra el detalle de las clases usadas en la aplicación.

Clase Action

Clase que contiene los métodos para interactuar con el equipo físico. Usando una instancia de la clase *Instrument* que es parte de la librería PyVisa, se emiten comandos al dispositivo por medio de la interfaz GPIB. En esta clase también se implementan algunos procedimientos como el barrido de corriente y voltaje del tipo maestro – esclavo y la configuración de valores fijos en el equipo. Esta clase se encuentra en el archivo actions.py del código fuente para la aplicación de fuente de poder.

Clase Sweeper

Clase implementada para iniciar barridos de corriente o voltaje de manera sencilla, se encuentra en el archivo funtions.py del código fuente. Esta clase reemplazó a la implementación inicial de un barrido de corriente o voltaje para la fuente de poder. La clase *Sweeper* recibe información ingresada por el usuario para crear un objeto de barrido, la misma clase se encarga de validar que se hayan ingresado valores dentro de los rangos permitidos por los dispositivos y levanta una excepción, en caso de que uno de los valores ingresados sea incorrecto.

Clase Definitions

Esta clase simplemente contiene variables estáticas para ser usadas en la validación de datos y despliegue de mensajes en varios métodos de la aplicación. En esta clase se deben modificar los valores permitidos por el equipo, de ser necesario.

Clase ValueException

Clase que hereda de la clase *Exception*, que forma parte de la librería estándar de Python2.7. Se utiliza simplemente para levantar excepciones con un mensaje descriptivo específico de la razón por la que esta fue generada en tiempo de ejecución.

Aplicación Desarrollada Para el Generador de Ondas

La aplicación para el generador de ondas debe ser capaz de realizar 3 operaciones, configurar ondas con valores fijos, barridos de frecuencia y también configurar ondas arbitrarias. En la Figura 7 podemos ver el diagrama de casos de uso de la aplicación.

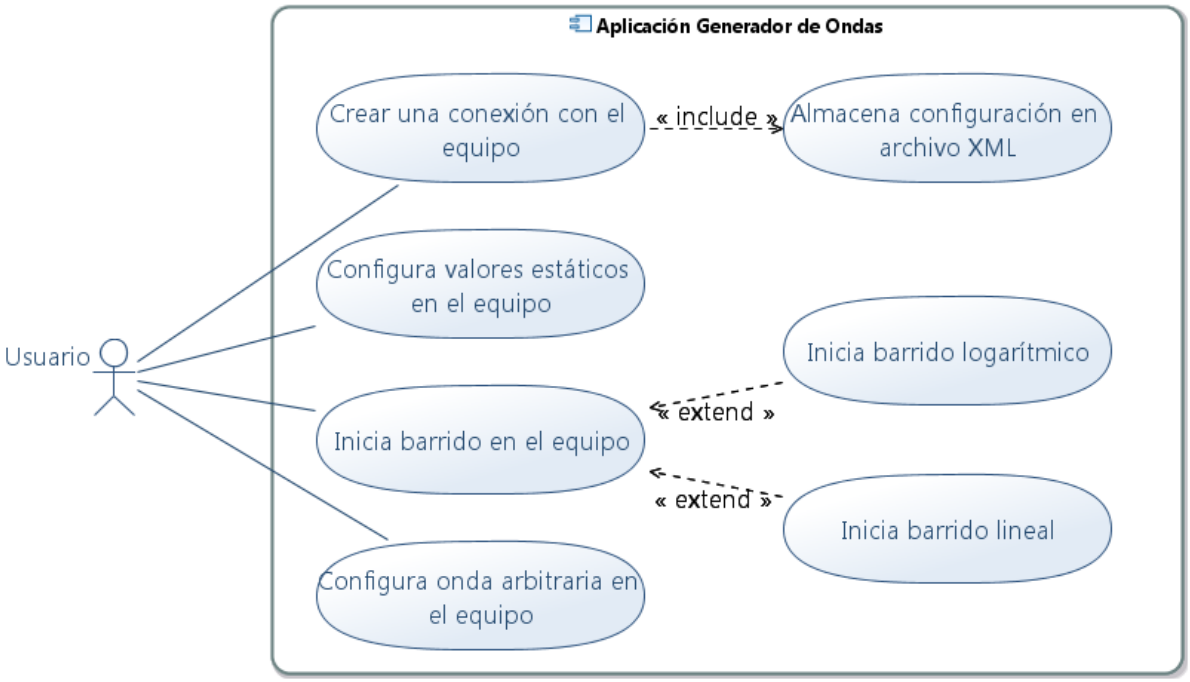

Figura 7: Diagrama de Casos de Uso Generador de Ondas

Primero, la aplicación debe ser capaz de configurar una onda fija con los valores seleccionados por el usuario de tipo de onda, frecuencia, amplitud y offset. Estos valores se deben encontrar dentro de los rangos permitidos por el equipo, está información será validada.

Segundo, el equipo debe poder realizar barridos de frecuencia sobre una onda eléctrica con características específicas definidas por el usuario, es decir forma, amplitud, offset, duración e intervalo de barrido. Este dispositivo cuenta con una función interna a la que se puede llamar para ejecutar barridos de frecuencia en el intervalo definido, con una duración específica y se puede inclusive escoger tipo de barrido lineal o logarítmico. Inicialmente se implementó esta funcionalidad obteniendo los parámetros para enviar el comando al equipo y dejar que este se encargue de la lógica del barrido. El inconveniente de usar este enfoque para realizar barridos, es que el usuario sólo definía la duración del barrido, más no la cantidad de veces que varía la frecuencia en el intervalo de tiempo definido. Esto es un problema ya que para algunos equipos de muestreo de estas ondas, se debe conocer este intervalo, para obtener una lectura precisa. Por esta razón, de la misma manera que en la fuente de poder, se implementó la lógica del barrido de corriente usando funciones de barrido y emitiendo comandos con cierta frecuencia al equipo. El barrido lineal de frecuencia se implementó en la lógica de la aplicación, pero por la naturaleza matemática del barrido de corriente logarítmico, en caso de que el usuario escoja este tipo de barrido, se utiliza la función interna del equipo.

Y la tercera y última función, la aplicación debe ser capaz de configurar el dispositivo para que genere ondas arbitrarias, es decir que el usuario ingresará una serie de valores de amplitud, para los cuales el equipo debe generar una onda. En este caso el usuario sólo debe escoger en qué frecuencia se quiere la onda y los valores de amplitud. Para el ingreso de valores de amplitud se construyó una matriz para mostrar al usuario en el que se escogen valores de voltaje para cada uno de los puntos en la onda que se quiere configurar.

Al igual que en la fuente de poder, se implementaron funciones básicas en la aplicación, como son las de conectarse al dispositivo, guardar la dirección GPIB o alias en

un archivo de configuración XML, prender y apagar el Output del equipo y actualizar el estado de conexión y Output en la interfaz gráfica de la aplicación.

Las aplicaciones fueron diseñadas para poder reutilizar mucho del código fuente en el desarrollo de aplicaciones similares. Por esta razón en el caso del generador de ondas, se describirán los métodos que sean diferentes a los de la fuente de poder, dentro de la descripción de cada clase de enlistarán aquellos métodos que ya fueron descritos en la aplicación de fuente de poder.

En la Figura 8 se muestra el diagrama UML de clases de la aplicación para el generador de ondas.

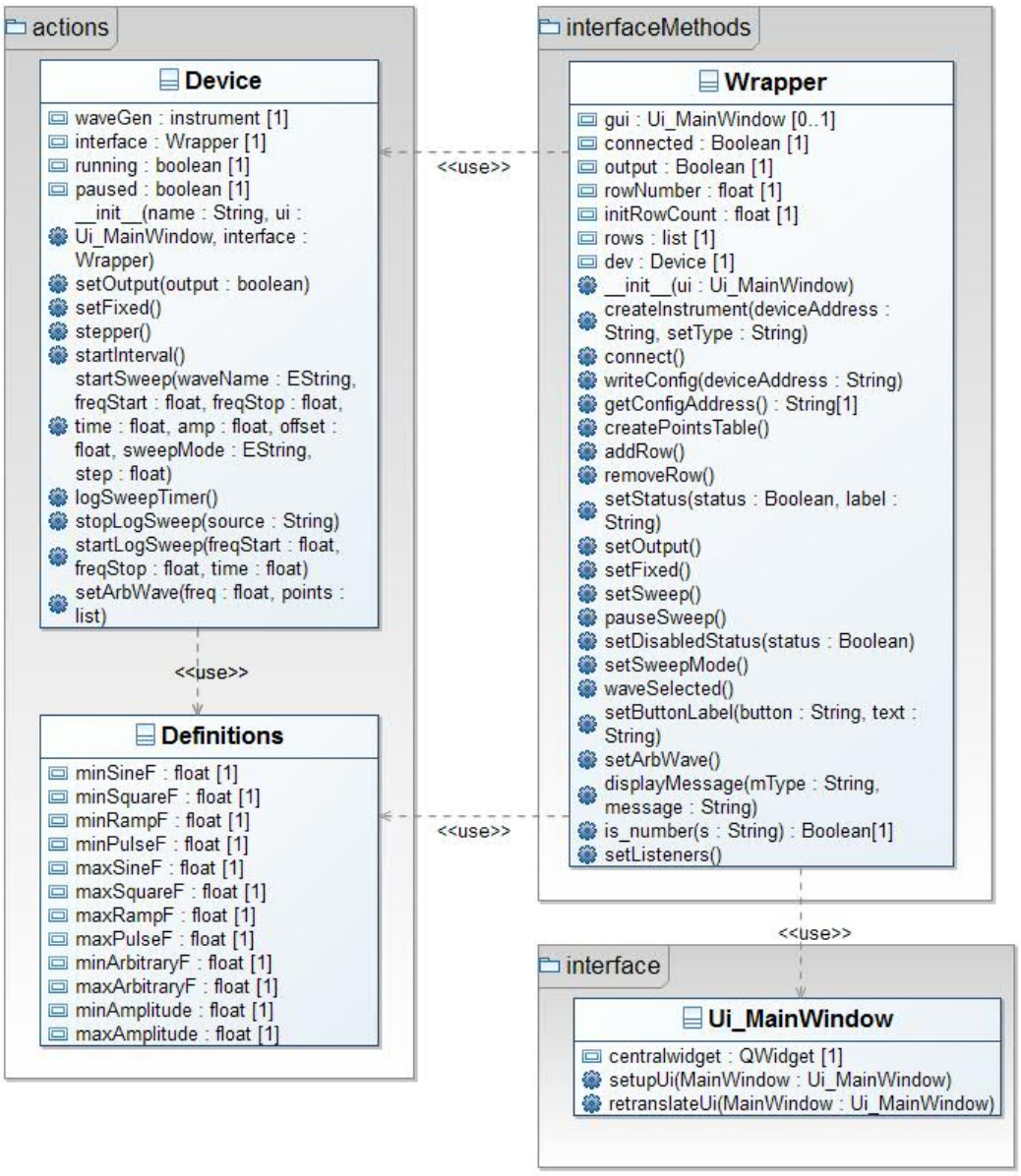

Figura 8: Diagrama de Clase Generador de Ondas

Clase Wrapper

De la misma manera que en la fuente de poder, la clase *Wrapper* es el controlador de la aplicación, es decir que desde ella se manipula la interfaz de usuario mediante referencias a la clase de interfaz gráfica y también se llama a métodos de la clase *Action* que se encarga de emitir comandos al dispositivo físico.

La clase *Wrapper* es más simple en el generador de ondas que en la fuente de poder, pues para el diseño de la interfaz se utilizó una vista con pestañas para mostrar cada

una de las funciones de la aplicación, a diferencia de la fuente de poder en la que sobre una sola vista, se muestran diferentes componentes mostrándolos y ocultándolos dinámicamente, y todo esto se controla en el código de la clase *Wrapper*.

El detalle de los métodos que componen cada una de las clases para la aplicación del generador de ondas se encuentra en el Anexo 2: Detalle de Aplicación Para Generador de Ondas.

Clase Device

Esta clase es el equivalente a la clase Action utilizada en la aplicación para la fuente de poder. Aquí se encuentran implementados los métodos que emitirán comandos al equipo físico por medio de la interfaz GPIB usando la clase Instrument, que es parte de la librería PyVisa. En estos métodos también se implementan procedimientos tales como el barrido de frecuencia. Esta clase se encuentra en el archivo actions.py del código fuente.

Clase Definitions

Al igual que en la aplicación para la fuente de poder, la clase *Definitions* contiene variables estáticas con valores que principalmente servirán para validar que los valores ingresados por el usuario estén dentro de los rangos permitidos por el equipo físico. Es en esta clase donde se deben manipular valores máximos y mínimos para cada una de las configuraciones que puede manejar un equipo de laboratorio.

Configuración de Matriz de Conmutación de Circuitos

Conexión con Caracterizador de Semiconductores

El primer paso para hacer funcionar a la matriz desde la interfaz del caracterizador, es configurar la comunicación entre estos dos equipos por medio de la interfaz GPIB que tiene cada uno de ellos. En el laboratorio de ingeniería electrónica cuentan con un adaptador de GPIB a USB, pero no contaban con un cable de GPIB en los dos extremos.

Para las pruebas se tuvo que solicitar este cable en el laboratorio de física de otro profesor de la Universidad San Francisco de Quito (Darío Niebieskikwiat).

Una vez que se conectó a los dos dispositivos por medio de su interfaz GPIB, se procede a configurar la conexión. Desde el panel frontal de la matriz de conmutación se debe configurar el tipo de mensajes que va a enviar y recibir el dispositivo, se debe fijar en la configuración que se usen mensajes con el formato de una matriz Keithley 707A, sólo de esta manera el caracterizador de semiconductores puede comunicarse de manera correcta con la matriz (Models 707B and 708B Switching Matrix User's Manual, 2010). El resto de configuraciones deben hacerse desde el caracterizador de semiconductores.

Para configurar la comunicación entre dispositivos, el caracterizador de semiconductores cuenta con una herramienta del fabricante llamada *Keithley CONfiguration Utility* (KCON). En la Figura 9, Figura 10 y Figura 11 vemos las pantallas de configuración de la herramienta.

En la Figura 9 se observa la pantalla principal de la herramienta de configuración. Aquí se pueden agregar todos los equipos que sean compatibles con el caracterizador de semiconductores. Para agregar la matriz de conmutación, se crea un nuevo dispositivo desde el menú de la aplicación, aquí se escoge el modelo de la matriz o en este caso una versión similar, la *Keithley 707A*. En la Figura 9 vemos la configuración básica del dispositivo, donde se especifica la dirección GPIB, el modo de operación y el modelo de tarjeta que está instalada en la matriz. Al seleccionar estas opciones se agrega un dispositivo más al árbol de configuración de la matriz como se muestra en la Figura 10.

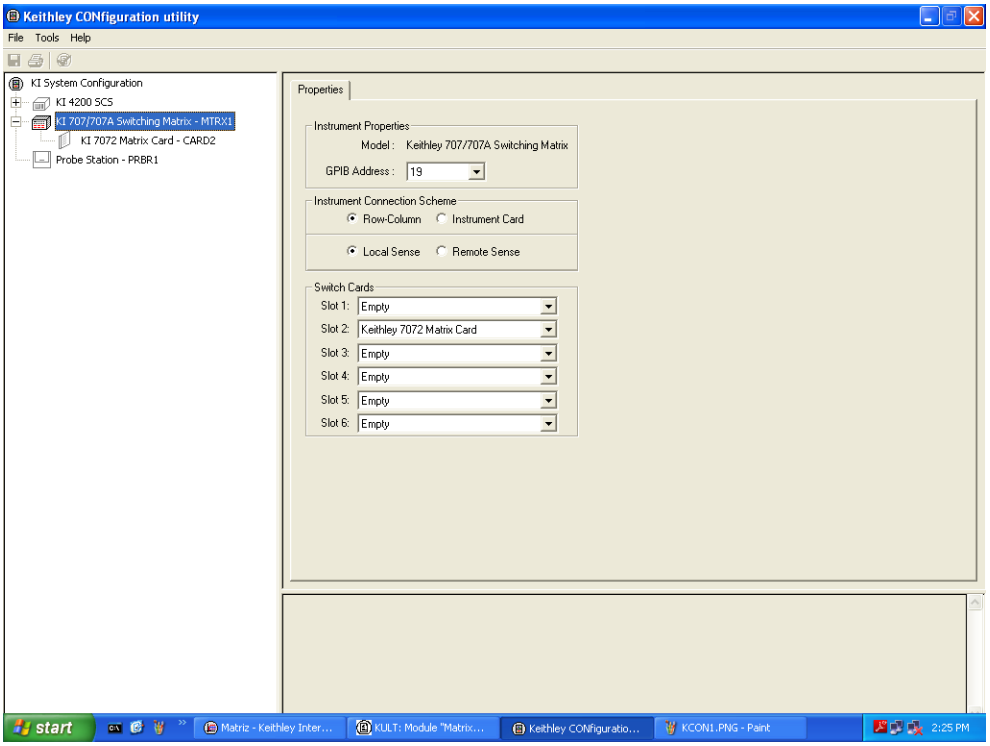

Figura 9: KCON

En la Figura 10 también se puede ver la configuración específica para la tarjeta que está instalada en la matriz de conmutación, una trajeta *Keithley 7072*. Las opciones que se seleccionaron para filas (rows) y columnas (columns) de la matriz dependen de lo que se desea conectar en los terminales de la matriz, en este caso se siguieron las indicaciones de Lionel Trojman pues se desea dejar conectada la matriz de tal manera que se puedan hacer diferentes tipos de mediciones simplemente seleccionando opciones en la aplicación KITE.

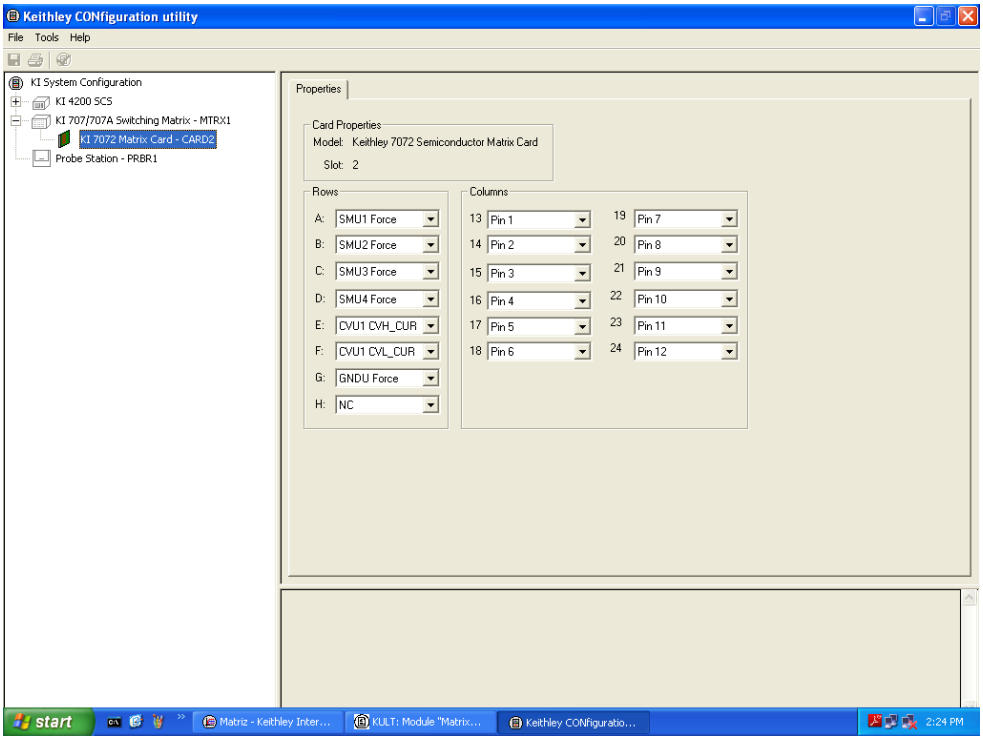

Figura 10: KCON

En la Figura 11 se puede ver como el caracterizador de semiconductores hace un mapeo de nombres a los instrumentos que están conectados a él. En esta página de configuración es donde se obtuvieron los nombres de cada uno de los conectores de la matriz, estos nombres son necesarios cuando se desea especificar que dispositivos se desean interconectar usando la aplicación KITE.

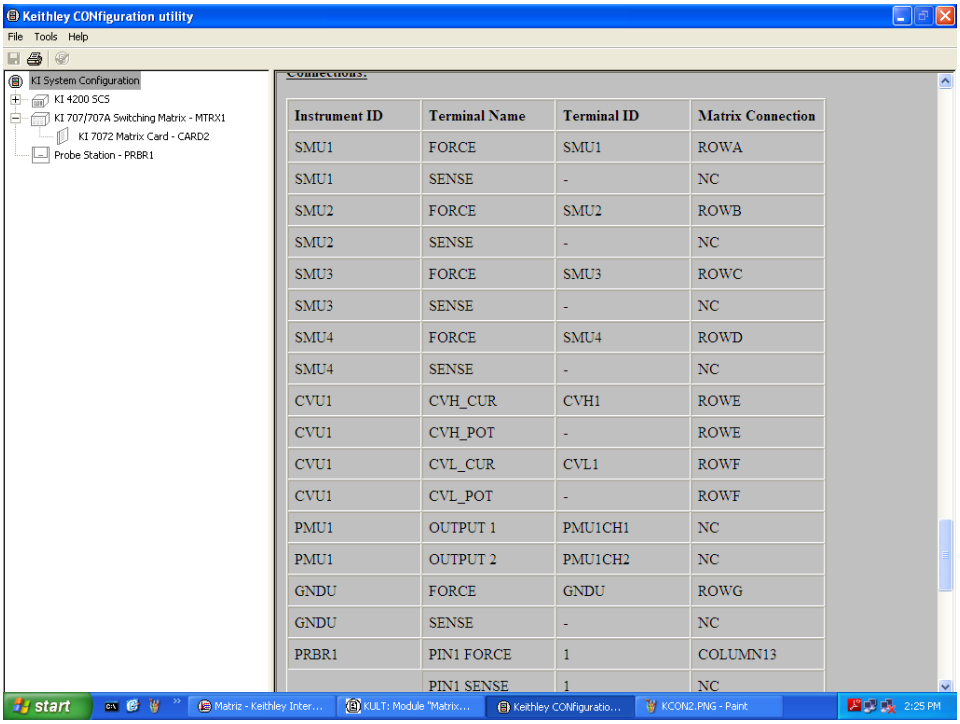

Figura 11: KCON

Manejo de Librerías en el Caracterizador de Semiconductores

El caracterizador de semiconductores además de la herramienta que ya describimos, *KCON*, cuenta con otras dos, *KULT* (Keithley User Library Tool) y *KITE* (Keithley Interactive Test Environment). Cada aplicación tiene un objetivo específico en el manejo del dispositivo. *KITE* es la aplicación de configuración en donde se van a programar las mediciones en última instancia, y *KULT* es la herramienta donde se programan las librerías de software que podrán ser utilizadas desde la aplicación *KITE*, estás librerías permiten al usuario programar todos los dispositivos compactibles con el caracterizador así como todas las funciones del caracterizador. La aplicación tiene una interfaz gráfica simple en donde se pueden declarar variables para el ingreso de datos y también variables que devolverán los métodos de una librería. Como se puede ver en la Figura 12, la definición de variables para el ingreso de datos se realiza de manera gráfica así como los valores que va a retornar la librería cuando termine su ejecución. Para la captura de datos que ingrese el usuario

vemos que se presenta una matriz en la que inclusive se puede validar los rangos de valores que puede ingresar el usuario, para el valor que retorna la función sólo se puede definir el tipo de dato, en el caso de este módulo está descrito como comentarios en el código el significado de cada valor que retorna la función, esto fue muy útil durante la configuración del dispositivo.

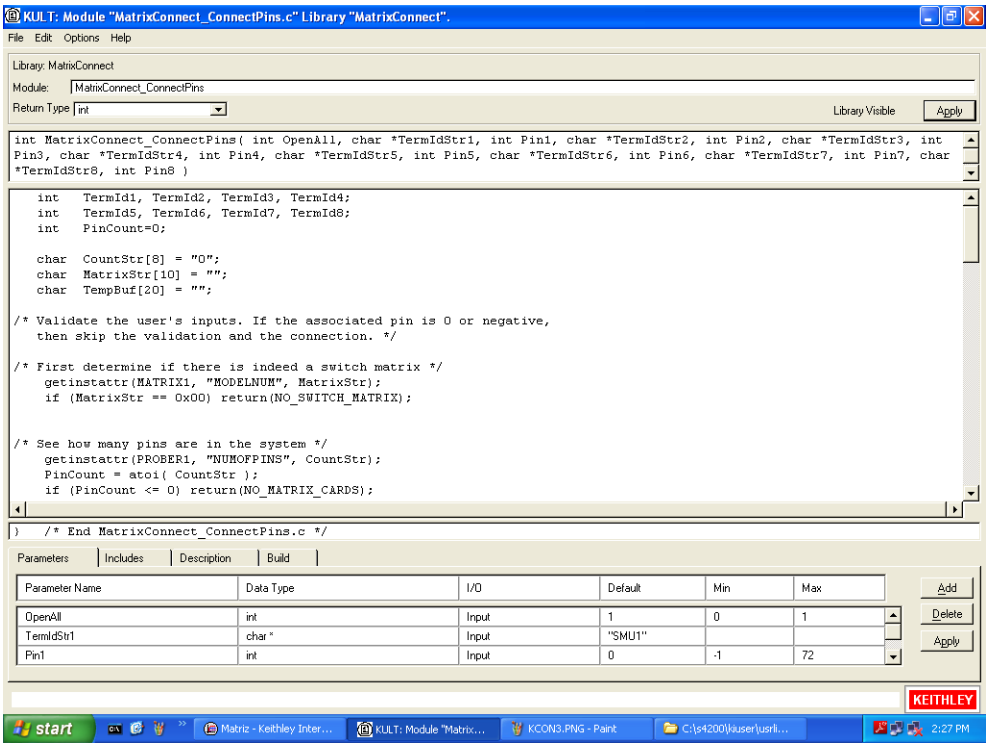

Figura 12: KULT

Para la configuración de la matriz vamos a usar la librería *MatrixConnect* y específicamente el módulo *MatrixConnect_ConnectPins*, en el anexo A se encuentra el código completo para esta librería.

Durante la configuración de los dispositivos se creó un módulo a partir de *MatrixConnect_ConnectPins* llamado *Custom_MatrixConnect_ConnectPins*. En este módulo se quería programar el funcionamiento de la matriz de tal manera que se ajuste a las necesidades de los usuarios del laboratorio de ingeniería electrónica. Después de entender todo lo que podía hacer la librería original se entendió que no era necesario implementar una nuevo sino sólo configurar la que ya existía.

Configuración del Ambiente de Pruebas KITE

KITE o *Keithley Interactive Test Environment* es la aplicación desde donde el usuario final va a configurar los equipos y diseñar mediciones que pueden involucrar a más de un dispositivo a la vez.

En la aplicación KITE se pueden crear proyectos para que permitan al usuario realizar mediciones de la manera que él lo necesita. Estos proyectos pueden ser almacenados y reusados cuantas veces sean necesarias, la aplicación tiene herramientas integradas para analizar los datos retornados en las mediciones e inclusive permite hacer un análisis estadístico de estos resultados.

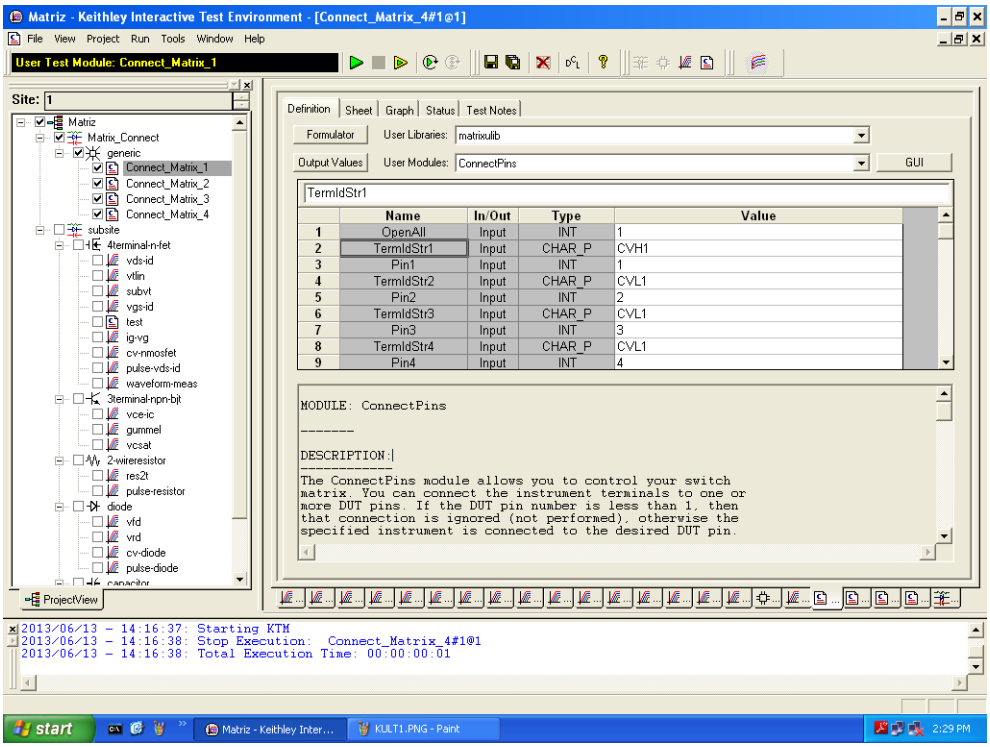

Figura 13: KITE

En la Figura 13, Figura 14, Figura 15 y Figura 16 podemos ver las 4 diferentes configuraciones para la matriz de conmutación (Mediciones IV, CGC, CGB, CGA), es decir que terminales se conectan entre ellas. En cualquiera de estas 4 figuras vemos como se muestra a un usuario una librería una vez que ha sido programada y compilada para ser usada dentro del ambiente de pruebas. En el caso del módulo *MatrixConnect_ConnectPins*

vemos como en la columna de valores (values) podemos ingresar el nombre del terminal ubicado en las filas de la matriz que deseamos conectar (definido por el dispositivo en la Figura 11) y el número del terminal que se encuentra en las columnas de la matriz como se definió en la Figura 9.

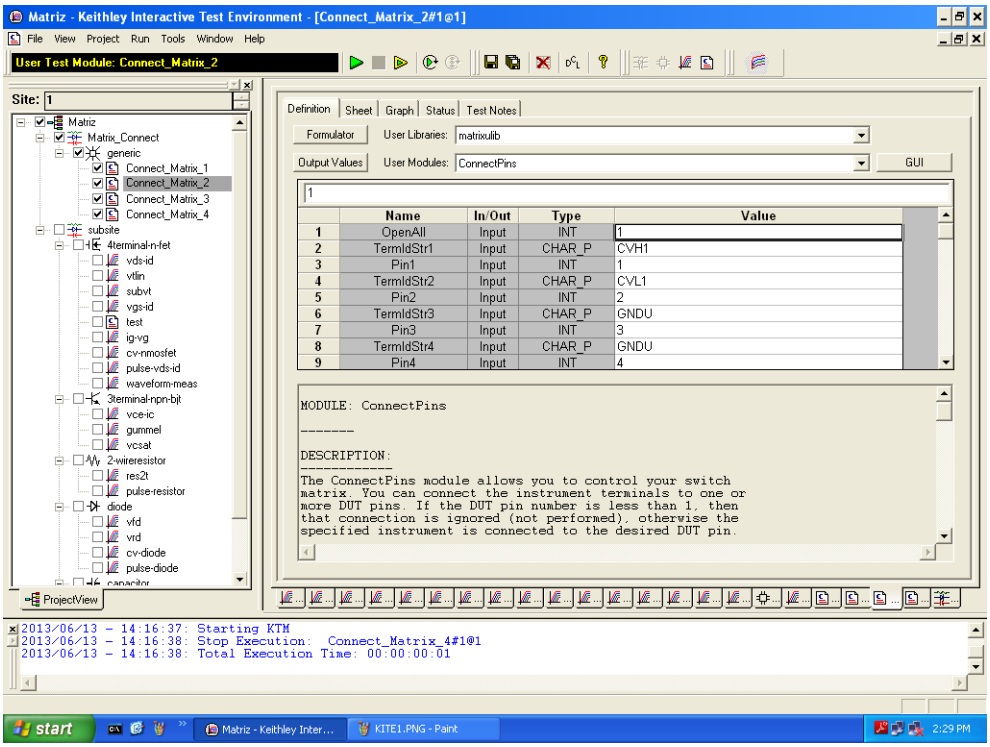

Figura 14: KITE

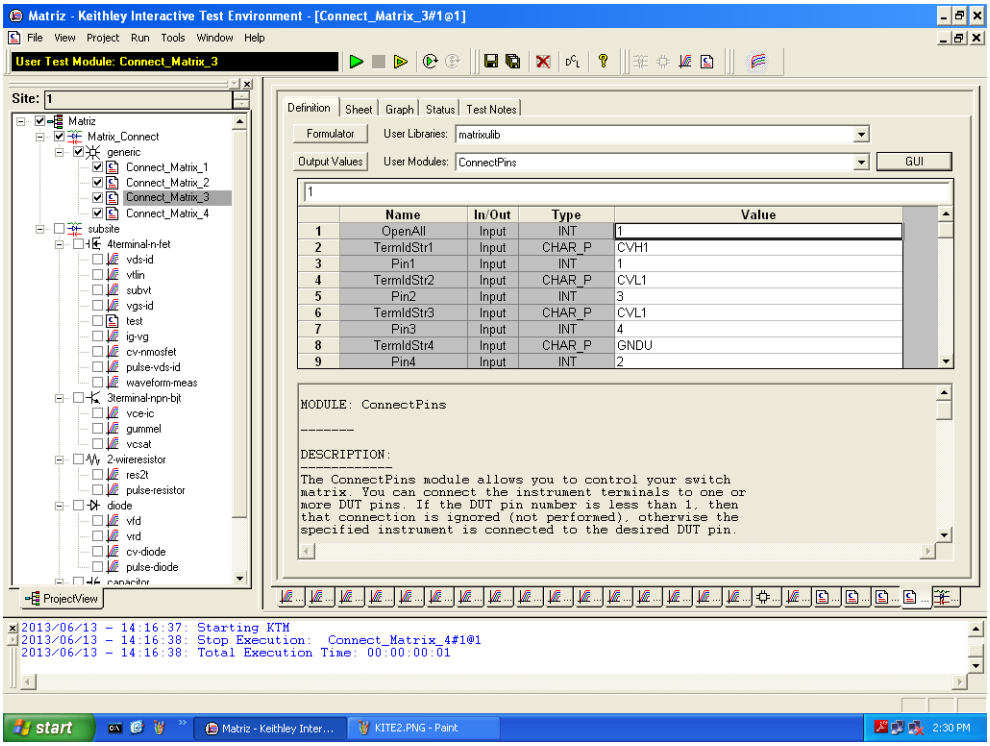

Figura 15: KITE

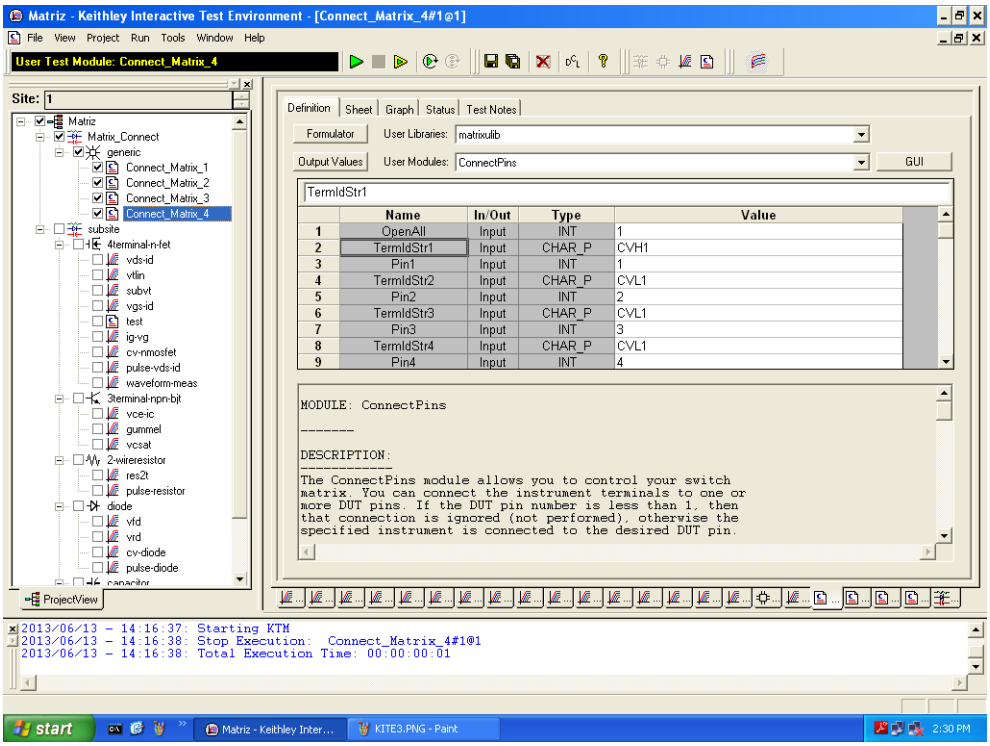

Figura 16: KITE

En la Figura 18 se observa la configuración final que se definió para el proyecto de medición. Se separaron las mediciones en 4 tipos:

1. IV
2. CGC
3. CGB
4. CGA

En la Figura 17 se muestra el diagrama de casos de uso para el proyecto configurado en el ambiente de pruebas *KITE*.

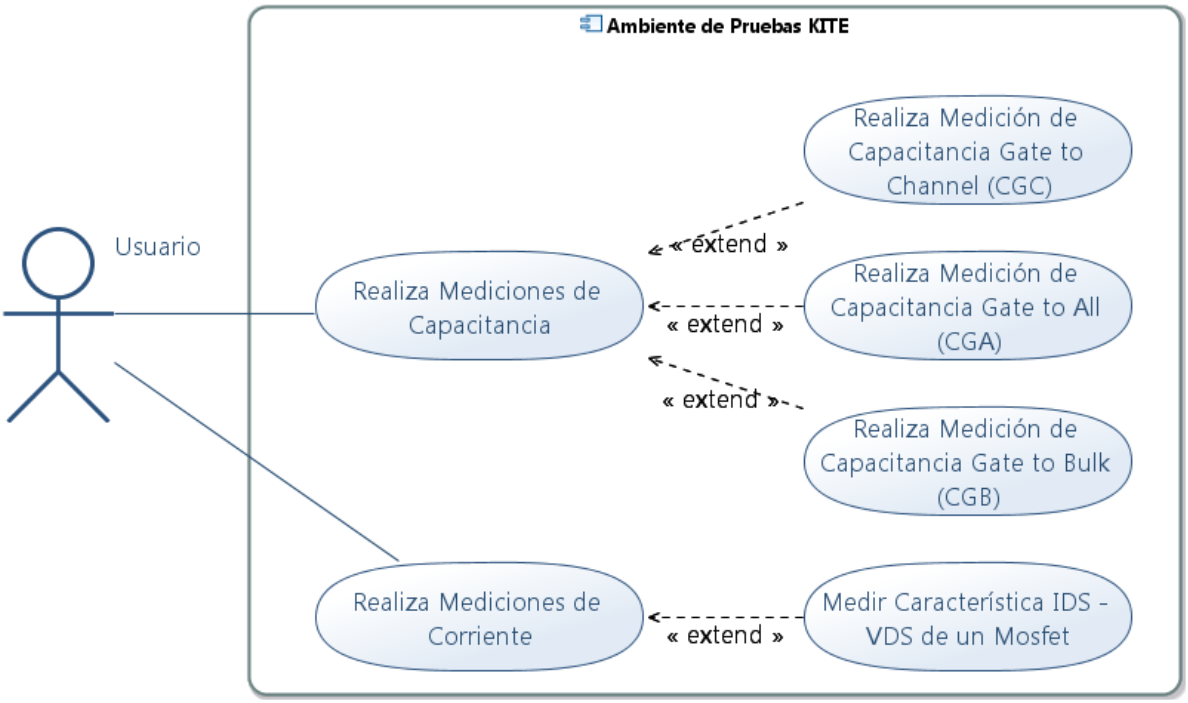

Figura 17: Diagrama de Casos de Uso KITE

Para cada tipo de configuración se tiene 3 módulos, el de conexión de la matriz, módulo de medición y módulo de desconexión de la matriz. Tomando como ejemplo al tipo de medición IV, tenemos los módulos *Conn_IV*, *Meas_IV* y *Disconn_IV*. El módulo de conexión para cada una de las mediciones, es el que se encarga de comunicarse con la matriz y configurarla de una manera específica para esa medición. El módulo de medición es aquel que asume que los terminales de los equipos están conectados de una determinada manera gracias a la configuración de la matriz y es capaz de realizar mediciones. El módulo de desconexión fue agregado sólo como una medida de precaución, ya que lo

único que hace es enviar la orden a la matriz de que desconecte todos los terminales una vez terminada la medición. De esta manera se evitarán accidentes debido a olvidos de conexiones establecidas por medio de la matriz.

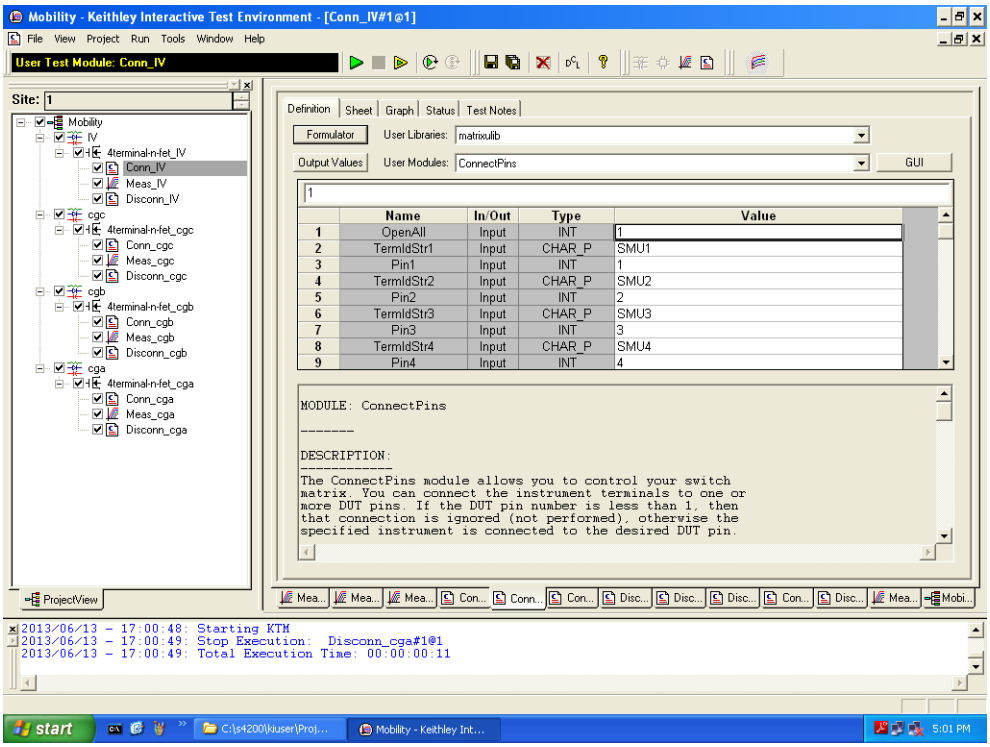

Figura 18: KITE Configuración Final

Capítulo 4 – Análisis de Resultados

Descripción de los Resultados

Después de terminada la implementación de las dos aplicaciones desarrolladas en Python, y de configurar la matriz de conmutación podemos ver que se han cumplido los objetivos planteados al comienzo de este documento. Estos objetivos fueron planteados en base a los requerimientos obtenidos de los usuarios del laboratorio de ingeniería electrónica, específicamente Luis Miguel Procel y Lionel Trojman, profesores de la Universidad San Francisco de Quito. A continuación repasamos los objetivos planteados para el desarrollo de este proyecto:

1. Desarrollar una aplicación en Python que permita, controlar una fuente de corriente y voltaje.
2. Desarrollar una aplicación en Python que permita, controlar un generador de ondas
3. Configurar una matriz de conmutación de circuitos, para comunicarse y ser controlada desde un caracterizador de semiconductores por medio de una interfaz GPIB.
4. Configurar el equipo donde se va a correr la aplicación de fuente de poder y generador de ondas, dentro del laboratorio de ingeniería electrónica.

En la Tabla 1 podemos ver el resumen de las metas que se cumplieron en este proyecto

Tabla 1: Metas Alcanzadas

Meta	Cumplido
Diseñar una interfaz gráfica que permita al usuario controlar todas las funciones que han sido programadas para la fuente de poder.	Si
Implementar en la aplicación desarrollada para la fuente de corriente y voltaje, las funcionalidades requeridas por el usuario.	Si
Diseñar una interfaz gráfica que permita al usuario controlar todas las funciones que han sido programadas para el generador de ondas.	Si
Implementar en la aplicación desarrollada para el generador de ondas, las funcionalidades requeridas por el usuario.	Si
Configurar la conexión entre la matriz de conmutación y el caracterizador de semiconductores	Si
Configurar la herramienta KITE para ejecutar las mediciones requeridas por los usuarios.	Si
Preparar la computadora donde se va a instalar la aplicación, para que esta pueda ejecutar las aplicaciones programadas	Si
Instalar los archivos de aplicación y verificar que esta funcione	Si

Objetivo 1

Este objetivo se cumple, ya que se han implementado todas las funciones requeridas por el usuario en la aplicación que controla la fuente de poder. Se debieron hacer varias pruebas con el usuario para probar que no existan errores en la aplicación una vez que esta se encuentra en ejecución. En la última prueba realizada en un computador del laboratorio de ingeniería electrónica donde se instaló la aplicación, se resolvieron todos los problemas encontrados en ese momento, y a lo largo del desarrollo de este proyecto se han realizado pequeñas mejoras en la versión final de la aplicación, como respuesta a pequeños errores encontrados mientras se analiza el código. Todos estos cambios han sido subidos al repositorio en línea de Github, de donde los usuarios del laboratorio pueden descargar la última versión de la aplicación por medio del mecanismo que se dejó configurado en el computador donde se corre la aplicación.

A continuación se resume el funcionamiento de la aplicación programada para la fuente de poder. Esto incluye la capacidad de conectarse al dispositivo, fijar

configuraciones estáticas de corriente y voltaje, hacer barridos de corriente o voltaje y por último hacer barridos de corriente y voltaje de tipo maestro – esclavo.

Administración de la Aplicación

Con administración de la aplicación nos referimos a todas las funciones que se han programado en la aplicación, con el fin de permitir al usuario realizar las actividades señaladas en los objetivos del proyecto. La primera característica importante es la que tiene la aplicación para establecer una conexión con el dispositivo físico. Cuando el usuario inicia la aplicación por primera vez, esta detecta que no existe un archivo de configuración creado en el sistema, por lo que muestra una alerta indicando que se debe ingresar la dirección GPIB del dispositivo al que se desea conectar o el alias GPIB con el que se lo ha configurado dentro del sistema. Una vez que el usuario ha ingresado la dirección GPIB o alias y presiona el botón conectar, se intenta establecer una conexión con el dispositivo, de ser exitosa la conexión, se actualiza la interfaz gráfica para mostrar que la aplicación tiene una conexión con el dispositivo y además se almacena en un archivo de configuración la dirección GPIB o alias que se utilizó en la conexión exitosa para que la siguiente vez que se ejecute la aplicación no se deba realizar esta conexión manualmente.

Otra funcionalidad de soporte para la aplicación es la de en cualquier momento durante la ejecución de la misma permitir al usuario encender o apagar el “output” del dispositivo, es decir permitir el paso de corriente o no en los terminales eléctricos del equipo. Además de estas funciones que se han implementado, sólo queda por describir aquellas funciones que fueron desarrolladas como un requerimiento de la aplicación.

Configuración Estática de Corriente y Voltaje

La primera y la más simple de las 3 funciones es fijar una configuración estática en el dispositivo, al usuario se le presentan dos cuadros de texto donde debe ingresar un valor

para corriente y otro para voltaje, después se presiona el botón “Fijar Configuración”, y si los valores ingresados por el usuario están dentro de los rangos permitidos, se envían los comandos de configuración al dispositivo. Cada vez que se fija una configuración estática, el estado de “output” del dispositivo cambia a encendido.

Barrido de Corriente o Voltaje

La segunda función de la aplicación es la de realizar barridos de corriente o voltaje en el dispositivo. Se probó esta funcionalidad de la aplicación en muchos escenarios. La validación del ingreso de datos funciona en cualquiera de los campos para valores mayores o menores a los permitidos. Para ingresar a este modo de barrido, el usuario debe seleccionar la opción de barrido para uno de los dos valores (corriente o voltaje), como se puede ver en la Figura 19. Una vez elegida esta opción, el usuario debe ingresar el valor fijo para corriente o voltaje y también todos los valores necesarios para iniciar un barrido como son valor inicial, final, y los dos valores de “step”, el primero se especifica en amperios (A) o voltios (V), es decir cuánto se va a aumentar o disminuir el valor en cada iteración del barrido, y el segundo valor se especifica en segundos, es decir cuánto va a durar cada iteración del barrido, sea este de corriente o voltaje. Se probó la aplicación para barridos de corriente y voltaje, cada uno de estos en un barrido creciente o decreciente, y más que nada se probaron muchos escenarios en los que el usuario ingresa valores de manera errónea para verificar que la aplicación realice todas las validaciones necesarias.

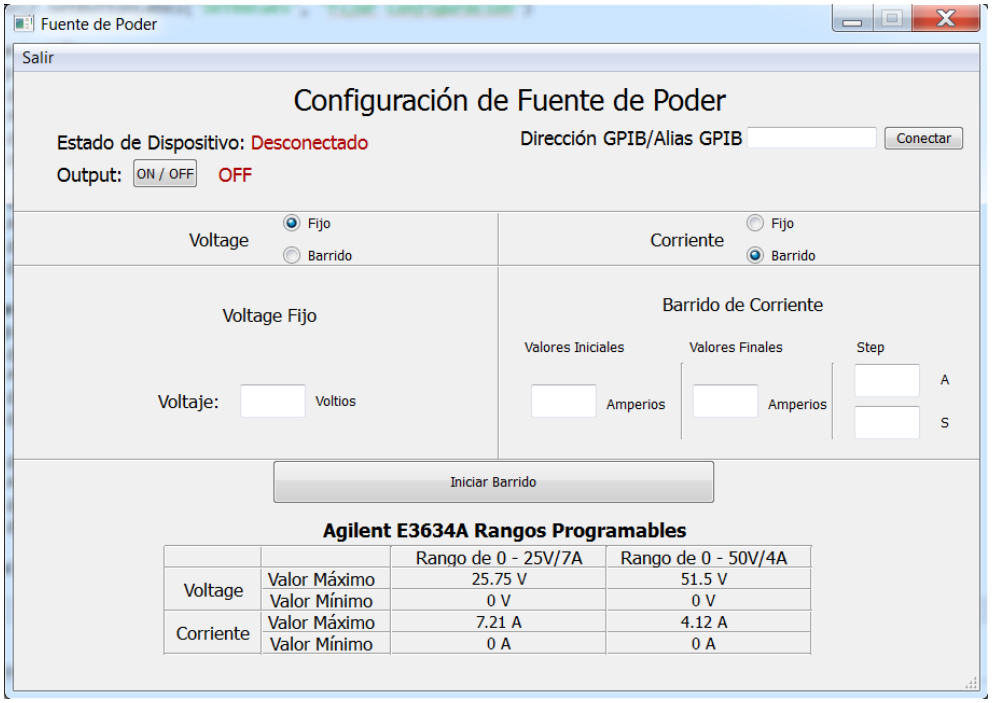

Figura 19: Barrido de corriente

Barrido de Tipo Maestro – Esclavo

Este modo de operación de la aplicación, al igual que los otros dos verifica el ingreso de datos por parte del usuario. Para acceder a este modo de operación, el usuario debe escoger para la configuración de corriente y voltaje el modo de barrido usando los botones de selección. Cuando se seleccionan estas opciones, la interfaz, además de mostrar los campos necesarios para ingresar datos de barrido para corriente y voltaje a la vez, muestra dos grupos de botones de selección que permiten configurar a la corriente o voltaje como Maestro o Esclavo, esto se muestra en la Figura 20. Aquel parámetro (corriente o voltaje) que ha sido fijado como esclavo, realizará un barrido completo por cada valor sobre el que el parámetro maestro itere. El barrido habrá terminado una vez que el parámetro maestro haya barrido todos los valores definidos dentro del intervalo configurado por el usuario. Este modo de operación fue probado en varios posibles escenarios de funcionamiento y se comprobó que funciona correctamente para todos ellos.

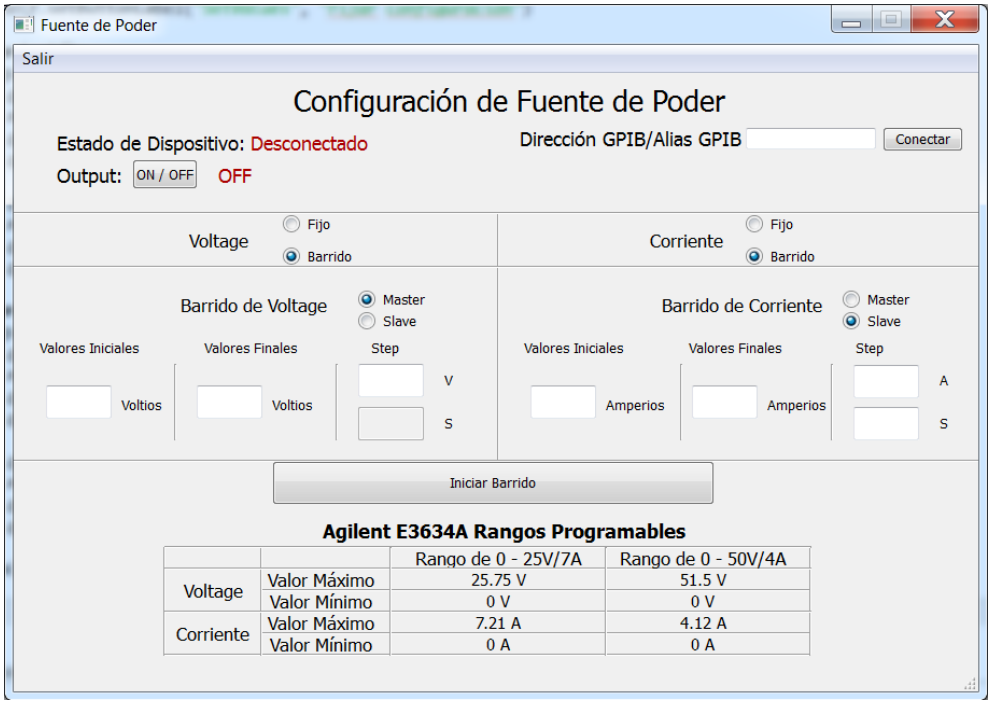

Figura 20: Barrido maestro esclavo

Objetivo 2

Al igual que con la aplicación para la fuente de poder, se cumplen todos los objetivos definidos para la aplicación que controla el generador de ondas. La aplicación cumple con las mismas funciones de soporte que la aplicación de la fuente de poder, es decir el control sobre el "output" del equipo y los mecanismos para conectarse al equipo y almacenar la configuración del dispositivo en un archivo XML. Además de esa funcionalidad básica, se cuenta con los modos de operación para fijar una configuración estática en el dispositivo, hacer un barrido de frecuencia lineal o logarítmico y configurar una onda arbitraria en el dispositivo.

Configuración Estática de una Onda

La configuración estática de este dispositivo es algo más compleja que la que tenemos en la fuente de poder. En este dispositivo debemos especificar 4 valores para fijar una configuración estática, tipo de onda, frecuencia, amplitud y offset. Durante las pruebas de la aplicación se probó que la aplicación valide el ingreso de datos para este modo de

operación. En un comienzo se detectaron pequeños errores de la aplicación cuando se seleccionaban valores de μHz por una conversión interna de notación que realizaba el intérprete de Python. Además de esto no se presentaron muchas complicaciones con este modo de operación de la aplicación. En la Figura 21 se puede ver una captura de pantalla de la aplicación en el modo de configuración estática. Se realizaron pruebas para todos los escenarios posibles en este modo de configuración y la aplicación no presentó problema alguno.

Figura 21: Configuración Estática

Configuración de Onda Arbitraria

En la Figura 22 se observa una captura de pantalla de la aplicación para el generador de ondas y su modo de configuración de una onda arbitraria. El usuario debe ingresar una frecuencia para la onda que se va a generar, así como los valores de amplitud que deben encontrarse en el rango de -10 a 10 voltios. Este modo de operación presentó algunos inconvenientes durante la implementación, pero luego de varias pruebas con el

equipo se pudieron corregir todos los errores y permitir que el usuario genere ondas arbitrarias de manera correcta. Este modo de operación valida cada uno de los valores de amplitud ingresados y muestra un mensaje de error al usuario indicando en que líneas se encuentra el error, como se muestra en la Figura 23.

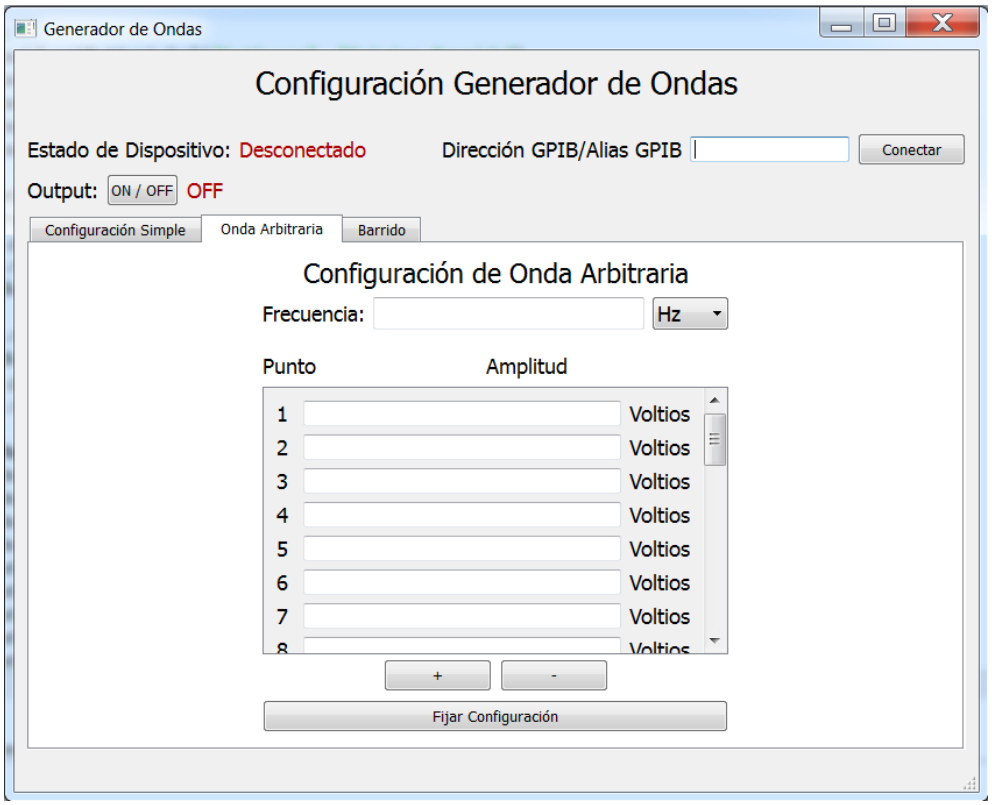

Figura 22: Onda Arbitraria

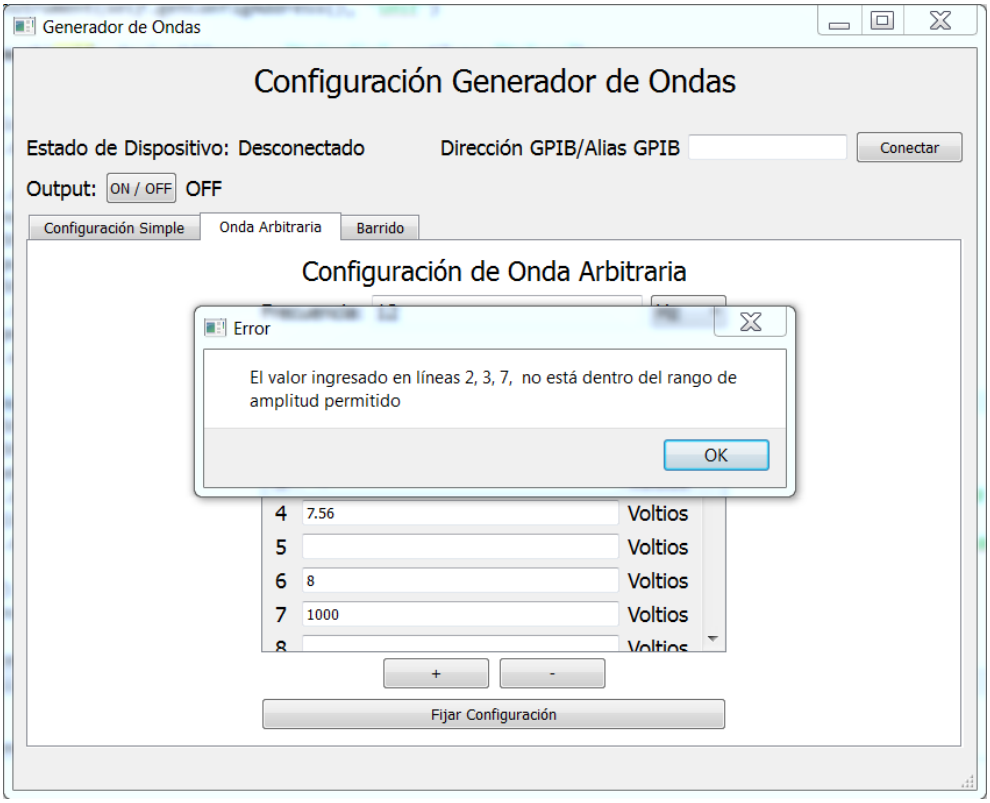

Figura 23: Onda Arbitraria Error

Barrido de Frecuencia

En la Figura 24 se puede ver el modo de operación para iniciar un barrido en el generador de ondas. Este fue el modo de operación donde se debió implementar el mayor número de validaciones. No sólo que el número de datos que debe ingresar el usuario es mayor, sino que existen validaciones condicionales de acuerdo a los mismos valores que ingresa el usuario, tales como que la amplitud de una onda nunca puede ser mayor a 10 voltios o menor que -10 voltios incluyendo el offset, además se debe validar que el rango de frecuencia elegido por el usuario esté dentro del permitido por el equipo.

Este modo de operación fue probado bajo diferentes condiciones y en todos los escenarios donde se realizaron las pruebas, los resultados fueron los deseados. Existen dos tipos de barrido, lineal y logarítmico. Inicialmente se realizó la implementación del barrido usando las funciones internas del generador de ondas, pero debido a que los usuarios desean tener control sobre el intervalo de tiempo que le toma al dispositivo cambiar un

valor de frecuencia, se debió implementar la lógica del barrido en la aplicación tal como se hizo en la aplicación para la fuente de poder. La implementación dentro del código de la aplicación fue para el barrido lineal, en el caso del barrido logarítmico si se utilizó la función interna del generador de ondas. Para el caso del barrido logarítmico también se debió implementar un método para detener el barrido una vez que haya transcurrido el tiempo especificado por el usuario, ya que al usar las funciones internas de barrido del generador de ondas, el barrido se repite sin fin hasta que el usuario lo detenga.

Figura 24: Barrido de Frecuencia

Objetivo 3

Este fue un objetivo más sencillo de cumplir que los otros dos primeros. En este caso no se tuvo que realizar desarrollo de ninguna clase, sino solamente configurar dos equipos de laboratorio para que funcionen en conjunto. Cada uno de estos dos dispositivos es programable independientemente, por una interfaz GPIB o Ethernet. De hecho, las pruebas iniciales con la matriz de conmutación se realizaron usando la aplicación web que

está integrada con el equipo y desde la línea de comandos en Windows enviando comandos a través de la interfaz Ethernet usando la misma librería PyVisa. Se planteó que se desarrolle una aplicación independiente como las desarrolladas para la fuente de poder y generador de ondas, pero los usuarios consideraron más conveniente configurar la matriz usando el caracterizador de semiconductores ya que de esta manera se puede manejar a los dos equipos de manera simultánea usando la aplicación KITE provista por el fabricante de los equipos *Keithley*.

La configuración es sencilla si es que ya sabes cómo hacerlo, pero en un comienzo hay muchos factores que se deben considerar y comprender. Cosas como encontrar el identificador de cada terminal de la matriz dentro de la configuración del caracterizador son sencillas, pero se debió examinar el código de la librería de conexión de la matriz para comprender que con estos nombres se podía interconectar cualquier terminal desde la interfaz de la aplicación KITE. Internamente la librería de conexión a la matriz llama a una función (*getinstid*) que devuelve un identificador numérico de cada terminal en la matriz, cuando se le envía como parámetro la cadena de caracteres correspondiente a cada terminal y que podemos encontrar en la ventana principal de la aplicación KCON. Con estos identificadores numéricos se llama a otra función (*conpin*) y se envía como parámetros dos identificadores de terminales en la matriz, uno para la columna seleccionada y otro para la fila.

Como se puede ver en la Figura 13, Figura 14, Figura 15 y Figura 16, sólo se debe ingresar el identificador como cadena de texto en la columna value de la matriz donde se ingresan valores, un valor para cada pin que se desea conectar, y en otra fila de la matriz también se ingresa el número de pin al que se desea conectar. Todos estos identificadores a manera de texto y números de pin se deben definir primero en la herramienta de configuración del caracterizador de semiconductores KCON.

El resultado final de esta configuración fue satisfactorio. Se realizaron diferentes pruebas con el proyecto creado en la aplicación *KITE* y no se presentaron errores. Para estas pruebas se debieron conectar todos los cables a la matriz cuando normalmente se conectaban directamente entre el caracterizador de semiconductores y las agujas de medición cercanas a los semiconductores.

Objetivo 4

El objetivo 4 (Configurar el equipo donde se va a correr la aplicación de fuente de poder y generador de ondas, dentro del laboratorio de ingeniería electrónica), fue la etapa final de este proyecto. Una vez que las aplicaciones fueron probadas se pudo instalar en otro equipo que no fuera el de desarrollo las aplicaciones para la fuente de poder y el generador de ondas. En el Anexo 3: Manual de Instalación Aplicaciones Para Fuente de Poder y Generador de Ondas, se encuentra el detalle de los pasos que se deben seguir para instalar la aplicación en un equipo con sistema operativo Windows XP o superior de 32 o 64 bits.

Además de instalar la aplicación, se instaló TortoiseGit en el equipo, un programa que permite tener control de versiones sobre archivos en Windows usando GIT. Se configuro este mecanismo de tal manera que el sistema pueda recibir actualizaciones de la aplicación desarrollada por medio del repositorio en línea mantenido en Github (https://github.com/mcelicalderon/Tesis.git).

Retroalimentación de los Usuarios

Se obtuvo constante retroalimentación de la aplicación por parte de Luis Miguel Procel y Lionel Trojman, los dos principales usuarios de las aplicaciones, durante el desarrollo de la misma. Esto quiere decir, que la mayor parte de funcionalidades fueron aprobadas antes de que el desarrollo de las aplicaciones termine. Mucha de la

funcionalidad de las aplicaciones también fue desarrollada en base a la retroalimentación obtenida de estos dos usuarios principales de la aplicación.

Finalmente, se realizó una demostración de las aplicaciones y configuración de la matriz de conmutación el 29 de noviembre de 2013. En la demostración estuvieron presentes los dos usuarios de las aplicaciones Luis Miguel Procel y Lionel Trojman, además de Fausto Pasmay (Director del Departamento de Ingeniería en Sistemas) y Daniel Fellig (Lector de este proyecto de tesis y profesor a tiempo completo de la Universidad San Francisco de Quito).

Las observaciones que se obtuvieron sobre las aplicaciones en esta demostración fueron dos. La primera, que la aplicación debería mantener un log de todos los comandos que emite a los equipos de laboratorio. La segunda, las aplicaciones no están capturando errores generados al emitir un comando al equipo de laboratorio cuando luego de establecer una conexión, se desconecta el cable o se apaga el dispositivo. Es decir, en cualquier situación en la que la aplicación emite un comando y el dispositivo se tarda más del tiempo mínimo definido, en responder. Esta última funcionalidad fue implementada en la aplicación antes de la entrega final de la misma.

Capítulo 5

Conclusiones

- Los usuarios del laboratorio de ingeniería electrónica de la Universidad San Francisco de Quito ahora cuentan con una interfaz más simple para interactuar con dos de sus equipos, un generador de ondas y una fuente de corriente y voltaje.
- La fuente de poder usada para este proyecto ahora cuenta con dos nuevas funcionalidades, hacer barridos de corriente o voltaje, y hacer barridos de corriente y voltaje del tipo maestro – esclavo. Además la configuración fija de valores en el dispositivo ahora es más sencilla.
- El generador de ondas usado para este proyecto ahora cuenta con una interfaz más sencilla para el uso de 3 de sus funciones, configurar valores fijos en el dispositivo, hacer un barrido de frecuencia (lineal y logarítmico) y configurar una onda arbitraria.
- La configuración de una onda arbitraria en el generador de ondas es ahora una tarea mucho más sencilla gracias a la aplicación desarrollada. El usuario sólo debe ingresar valores de amplitud en una lista de cuadros de texto, elegir una frecuencia y presionar un botón para fijar esta configuración en el equipo.
- Las aplicaciones para fuente de poder y generador de ondas permiten que el usuario tenga retroalimentación sobre los valores que utiliza para configurar los equipos, en relación a los rangos que permite cada uno de ellos. En cuanto a validación de valores, la matriz de conmutación ya cuenta con funciones integradas que validan los datos ingresados por el usuario.
- Al configurar la matriz de conmutación para que pueda ser controlada desde el caracterizador de semiconductores, se facilita la realización de mediciones. El beneficio principal de esta integración es que los usuarios ya no deben conectar y

desconectar cables para realizar mediciones específicas, de esta manera evitan la manipulación de estos cables que son delicados y podrían deteriorarse con la continua conexión/desconexión. De esta manera se puede realizar diferentes tipos de mediciones simplemente configurándolas desde la aplicación KITE y además se evita la manipulación del hardware.

— Validar las acciones que realiza el usuario a través de la interfaz gráfica es indispensable, de otra manera la aplicación podría presentar comportamiento inesperado debido a eventos para los cuales no está programada.

— Presentar retroalimentación al usuario por medio de la interfaz gráfica es importante. De otra manera, si el usuario comete un error, no hay manera de que este identifique dicho error y pueda corregirlo.

— El lenguaje de programación Python es una muy buena elección para el manejo de dispositivos por interfaces remotas gracias a la librería PyVisa. Además el simple desarrollo de interfaces gráficas usando PyQt nos permite crear interfaces intuitivas y de fácil uso para el usuario sin un nivel de complejidad muy elevado. Otro beneficio de este lenguaje es que puede ser ejecutado en múltiples plataformas, de hecho, de no ser por las librerías propietarias que manejan la comunicación con los equipos, las aplicaciones desarrolladas en este proyecto podrían ser controladas no sólo desde un sistema operativo Windows, sino en cualquiera que pueda tener un intérprete de Python instalado.

— Obtener requerimientos detallados para las aplicaciones que se van a desarrollar antes de comenzar el desarrollo es muy importante. De lo contrario, se puede perder tiempo con implementaciones innecesarias o erróneas de acuerdo a la percepción que pueda tener el usuario y el programador de dichas aplicaciones.

Recomendaciones

- Una de las mejores prácticas para el desarrollo de aplicaciones e implementación de sistemas en cuanto a definir el alcance, es siempre obtener los requerimientos con el usuario antes de realizar cualquier tipo de desarrollo. Pero, sabiendo esto, es muy común que nos apresuremos y comencemos con el desarrollo antes de que se hayan cerrado los requerimientos de un proyecto. Por esta razón, me parece muy importante documentar todo requerimiento que se obtenga con el cliente para tener un respaldo cuando se deba hacer una entrega completa o parcial de la aplicación o sistema.
- Python es el lenguaje que fue escogido para el desarrollo de las aplicaciones para la fuente de poder y generador de ondas por Luis Miguel Procel y Lionel Trojman por su portabilidad y facilidad para ejecutar aplicaciones escritas en este lenguaje en múltiples plataformas. Por esta razón, para el desarrollo de este proyecto aprendí el lenguaje y desarrolle las aplicaciones usándolo. Pero hay ciertas cosas sobre el lenguaje que aprendí durante el desarrollo y que debí conocerlas antes de comenzar. Estándares de programación propios del lenguaje, patrones de diseño propios de las librerías que utilicé e inclusive la manera de nombrar métodos y variables son cosas que fui descubriendo a lo largo del desarrollo, y muchas cosas tuvieron que ser re implementadas una vez que aprendía algo nuevo. Por esta razón, antes de empezar el desarrollo, es recomendable aprender más sobre el lenguaje antes de empezar la implementación de aplicaciones.
- En las aplicaciones desarrolladas se consideraron la mayor cantidad de escenarios posibles, en donde el usuario puede no dar un uso ideal a las aplicaciones. Es decir, escenarios, donde las cosas no suceden de la manera en la que pensamos que van a suceder mientras esta es programada. Aun así, es difícil considerar todos los

escenarios, y uno de estos fue descubierto durante la demostración final de las aplicaciones. No se estaba considerando el caso en el que, después de establecer la conexión inicial con el equipo, se pierda esta conexión por una desconexión de cable o que se apague el equipo. Esta corrección es importante, por lo que si se incluyó en la versión final de la aplicación. Ahora si se emite un comando y el dispositivo no estaba listo, se presenta un mensaje de error y se cambia la interfaz gráfica para mostrarla como desconectada del equipo. Por este y otros escenarios que posiblemente no fueron considerados, se debería tener una etapa de pruebas con los usuarios con cada uno de los módulos desarrollados, es en estas etapas de uso donde se podrán encontrar la mayor cantidad de errores en las aplicaciones.

— En cuanto a la utilización de librerías en la aplicación KITE, es recomendable analizar el código ya escrito en las librerías incluidas con el programa antes de pensar en desarrollar una implementación propia de librería para interactuar con los equipos. En este proyecto se había comenzado a desarrollar una librería específica para el control de la matriz de conmutación antes de notar que la que ya estaba implementada para la matriz de conmutación era perfectamente capaz de realizar las funciones que se necesitaban para cumplir con los requerimientos de los usuarios.

Desarrollo Futuro

Como parte de la iniciativa de desarrollo de aplicaciones para el uso de alumnos y profesores en el departamento de ingeniería electrónica de la Universidad San Francisco de Quito, se espera que este proyecto sea solamente uno de los primeros. Paralelo al desarrollo de este proyecto, otro alumno de la universidad está desarrollando una aplicación para controlar el mismo caracterizador de semiconductores (Keithley 4200-

SCS), la aplicación se desarrolla en Python, usando PyQT y PyVisa, de la misma manera que se hizo en este proyecto.

Todas estas aplicaciones que serán desarrolladas para el departamento de ingeniería electrónica servirán no sólo para la función específica para la que fueron desarrolladas, sino como la base para el desarrollo de otras aplicaciones o extensiones en funcionalidad de las ya existentes. Es por esta razón, que el código fuente de las aplicaciones desarrolladas en este proyecto es abierto y está disponible para ser reutilizado en otros proyectos.

Específicamente en este proyecto, se puede hablar de algunas funcionalidades que no pudieron ser implementadas por razones de tiempo.

Para las dos aplicaciones (Para fuente de poder y generador de ondas), se habló durante la demostración de mantener un log de los comandos emitidos a los dispositivos por la aplicación. Es decir, con el fin de poder auditar la aplicación en caso de fallos o funcionamiento inadecuado de la misma, se quisiera poder tener un archivo de texto en el que se almacene con fecha y hora, todos los comandos que se envían a los dispositivos por medio de la interfaz GPIB. Esta funcionalidad no es muy difícil de implementar pero simplemente está fuera del alcance del proyecto y por razones de tiempo no fue implementada.

Otra funcionalidad que fue sugerida por uno de los usuarios durante el desarrollo, fue que se pueda configurar ciertos parámetros adicionales de los dispositivos desde la aplicación. Específicamente, esta recomendación se trataba de que en la aplicación del generador de ondas, el usuario pueda escoger la impedancia que se quiere usar en el dispositivo. Actualmente la aplicación utiliza por defecto alta impedancia, pues de esta manera sin configuraciones adicionales, se podía configurar al dispositivo dentro de los

rangos de valores que se han programado en esta aplicación. De otra manera estos rangos serían afectados por el valor de impedancia que se configure en el equipo.

Actualmente el único valor de configuración que se almacena en las aplicaciones es el de la dirección GPIB del equipo. Esta funcionalidad se podría extender para almacenar otros valores de configuración de los equipos como impedancia en el generador de ondas, y configuraciones más complicadas como por ejemplo la posibilidad de escoger ondas arbitrarias definidas por el usuario. En la fuente de poder no hay mucho que se pueda almacenar más que el estado de la aplicación la última vez que se ejecutó, pero en el generador de ondas podríamos pensar en más funcionalidades como generar ondas basadas en funciones matemáticas que defina el usuario escribiendo expresiones matemáticas como texto.

Estas y muchas otras funciones podrían implementarse con estos mismo equipos de laboratorio, y esto será un trabajo que probablemente seguirá siendo desarrollado por estudiantes en el laboratorio de ingeniería electrónica.

También se discutió con los usuarios de las aplicaciones las posibles pruebas y experimentos que se podrían diseñar para integrar más de uno de estos equipos. Al momento esto está en sus planes, pero al menos por ahora no saben exactamente que experimentos y mediciones van a realizar. Pero, se considerarán escenarios en los que por ejemplo se pueda conectar una fuente de poder adicional al caracterizador de semiconductores o también el generador de ondas.

Referencias

Agilent 33250A 80 MHz Function / Arbitrary Waveform Generator User's Guide, Segunda Edición. Agilent Technologies, Loveland, Colorado, 2003.

Agilent E3633A and E3634A DC Power Supplies User's Guide, Cuarta Edición. Agilent Technologies, Loveland, Colorado, 2012.

Agilent Technologies (2011). TECHNOLOGY SYNERGIES ACROSS AGILENT. [ONLINE] Disponible en: http://www.agilent.com/labs/agilent_synergies2011.pdf. [Último Acceso 3 de agosto de 2013].

Bochmann, Gregor V.. Finite state description of communication protocols, Computer Networks (1976), Volume 2, Issues 4–5, September–October 1978, Pages 361-372, ISSN 0376-5075, 10.1016/0376-5075(78)90015-6.

Elliott, C., Vijayakumar, V., Zink, W., & Hansen, R. (2007). National instruments LabVIEW: a programming environment for laboratory automation and measurement. Journal of the Association for Laboratory Automation, 12(1), 17-24.

Gallardo, S. Dept. of Electron. Eng., Seville Univ. Barrero, F. ; Toral, S.L. ; Duran, M.J., (2006). eDSPlab: A remote-accessed instrumentation laboratory for digital signal processors training based on the Internet . IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on. 1 (1), pp.4656 – 4661.

IEEE Computer Society, (2008). Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. 3rd ed. New York: The Institute of Electrical and Electronics Engineers, Inc..

IEEE Computer Society, (2004). Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation. 1st ed. New York: The Institute of Electrical and Electronics Engineers, Inc..

Kodosky, Jeffrey L. (Austin, TX), Shah, Darshan (Round Rock, TX), Dekey, Samson (Austin, TX), Rogers, Steven (Austin, TX) 2001 Embedded graphical programming system United States National Instruments Corporation (Austin, TX) 6173438 http://www.freepatentsonline.com/6173438.html

Meloni, S., Rosati, M., Federico, A., Ferraro, L., Mattoni, A., & Colombo, L. (2005). Computational Materials Science application programming interface (CMSapi): a tool for developing applications for atomistic simulations. Computer Physics Communications, 169(1-3), 462-466. doi:10.1016/j.cpc.2005.03.102.

Models 707B and 708B Switching Matrix User's Manual, Rev. A. Keithley Instruments, Inc, Cleveland, Ohio, 2010.

Models 707B and 708B Switching Matrix Reference Manual, Rev. A. Keithley Instruments, Inc, Cleveland, Ohio, 2010.

Pierce, Benjamin C., (2002). Types and Programming Languages. 1st ed. Estados Unidos de América: e.g. Houghton Mifflin.

Rawnsley, D.J., (1997). A virtual instrument bus using network programming . Instrumentation and Measurement Technology Conference, 1997. IMTC/97. Proceedings. Sensing, Processing, Networking., IEEE. 1 (1), pp.694 – 697.

Sanner, M. F. (e.g. 1975). PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT. The Scripps Research Institute. e.g. 32 (e.g. 2), pp.7.

Seely Brown, John, Paul Duguid, (1996). 'Keeping it Simple'. In: John Seely Brown and Paul Duguid (ed), Bringing design to Software. 1st ed.

Sokoloff, Leonard (2002). GPIB Instrument Control. DeVry College of Technology. Session 2559 (1), pp.16

Stroustrup, Bjarne , (1997). The C++ Programming Language. 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Anexo 1: Detalle de Aplicación Para Fuente de Poder

Clase Wrapper

Clase que contiene los métodos para el manejo de la interfaz de usuario, es decir responde a todos los eventos generados por la misma. Se encuentra en el archivo interfaceMethods.py del código fuente para la aplicación de fuente de poder. A continuación se describe la funcionalidad de cada uno de los métodos en la clase.

__init__(self,ui)

Este es el constructor de la clase Wrapper, y recibe como argumento una referencia a la instancia que se ha creado de la clase (*self*) y a la ventana principal donde se encuentran todos los componentes gráficos de la aplicación (*ui*). Este método inicializa todas las variables de la clase, y además hace varias llamadas a métodos para fijar la apariencia inicial por defecto de la interfaz gŕáfica. El constructor también llama al método *createInstrument(self, deviceAddress = "NoConfig", setType="NoType")*, encargado de crear la conexión con el dispositivo físico.

createInstrument(self, deviceAddress = "NoConfig", setType="NoType")

Método que se encarga de crear una conexión al dispositivo y una instancia de la clase Action (Encargada de enviar comandos al dispositivo). Recibe la dirección GPIB del equipo o alias, además de una variable que indica si se está llamando al método desde el constructor de la clase o no. Si la conexión es exitosa con el dispositivo, se modifica la interfaz para mostrar que la conexión fue exitosa, además, si se llamó a este método desde el manejador de eventos del botón conectar, en caso de tener una conexión exitosa, se almacena en el archivo de configuración la dirección o alias que se utilizó para conectarse al equipo.

connect(self):

Manejador del evento generado al presionar el botón “Conectar” en la interfaz gráfica. Se encarga de llamar al método *createInstrument* indicando que se lo llama al presionar el botón “Conectar” con el fin de almacenar en el archivo de configuración la dirección GPIB o alias utilizado en caso de tener una conexión exitosa con el dispositivo.

writeConfig(self, deviceAddress)

Método al que se llama en caso de tener una conexión exitosa con el dispositivo si se presionó el botón “Conectar” desde la interfaz gráfica. Recibe la dirección GPIB o alias utilizado en la conexión y lo almacena en un archivo de configuración XML (*config.xml*) en el directorio de instalación de la aplicación.

getConfigAddress(self)

Método que no recibe argumentos. Al ser llamado puede retornar una dirección GPIB o alias almacenado en el archivo de configuración *config.xml*, de no existir este archivo, se retorna la cadena de caracteres *“NoFile”*.

displayMessage(self, mType, message)

Debido a que se desean mostrar varios mensajes de error e informativos en la pantalla, se creó este método que recibe como argumentos el título de la ventana de error, así como el mensaje que se quiere mostrar. Con estos argumentos se despliega un mensaje al usuario que contiene los argumentos recibidos.

is_number(self, s)

Método simple utilizado para probar si el argumento recibido es un número o no.

setStatus(self,status, label)

Método encargado de actualizar la interfaz gráfica para mostrar si el dispositivo está conectado, desconectado, con Output ON o Output OFF. Recibe como argumentos una variable booleana (verdadero o falso) y en texto la etiqueta a la cual se quiere modificar el estado.

setOutput(self)

Método que cambia el estado del Output del dispositivo entre ON y OFF. Envía comandos al equipo conectado usando una instancia de la clase Action. Este método también actualiza la interfaz gráfica de la aplicación para mostrar el cambio que se ha realizado.

setButtonLabel(self, button, text)

Método utilizado para cambiar el texto que se muestra en los botones de la interfaz gráfica. Recibe como argumentos el nombre del botón al que se quiere modificar y el texto que se quiere mostrar en este.

setDisabledStatus(self,status)

Método utilizado para deshabilitar la edición de campos donde el usuario ingresa valores una vez que se empieza un barrido de corriente, voltaje o de corriente-voltaje. Su único parámetro es un booleano que define si se habilita la edición de campos o se la deshabilita.

setVoltageType(self)

Método que maneja la selección de configuración de voltaje (fijo o barrido). En base a esta selección se modifica la interfaz para mostrar los campos de ingreso de datos en base a la selección del usuario. Este método responde al evento generado al presionar los botones de tipo de configuración de voltaje.

setCurrentType(self)

Método que maneja la selección de configuración de corriente (fijo o barrido). En base a esta selección se modifica la interfaz para mostrar los campos de ingreso de datos en base a la selección del usuario. Este método responde al evento generado al presionar los botones de tipo de configuración de corriente.

setVMaster(self)

Método que se ejecuta al presionar el botón de selección de maestro o esclavo para los valores de voltaje en un barrido de corriente-voltaje. En base a esta selección se deshabilita el campo de tiempo de step en el barrido del maestro

setCMaster(self)

Método que se ejecuta al presionar el botón de selección de maestro o esclavo para los valores de corriente en un barrido de corriente-voltaje. En base a esta selección se deshabilita el campo de tiempo de step en el barrido del maestro

setValues(self)

Método manejador del evento generado al presionar el botón principal de configuración del dispositivo. Este botón muestra el texto “Fijar Configuración” cuando se inicia la aplicación, pero dependiendo de qué tipo de configuración se escoge para corriente y voltaje, este texto puede cambiar a iniciar barrido. También, una vez comenzado un barrido, el texto cambia a detener barrido, y en base a al texto del botón, este método va a realizar diferentes acciones.

Sí corriente y voltaje se encuentran configurados para valores fijos, se llama al método *setFixed()* para configuraciones estáticas del equipo. Si se va a hacer un barrido sólo de corriente o voltaje, se llama al método setInterval(). Cuando se ha seleccionado hacer un barrido de corriente-voltaje maestro esclavo, el texto del botón es el mismo que

para el caso anterior, pero en este caso se llama al método *setStep()*. Y por último, si el texto es “Detener Barrido”, se detiene el barrido que se esté realizando en cualquier momento.

Además, este método valida el ingreso de datos para valores fijos de corriente y voltaje, como también para valores ingresados en un barrido de corriente-voltaje del tipo maestro esclavo. La validación del barrido simple se realiza en la clase *Sweeper* descrita más adelante.

setFixed(self)

Método cuya función es fijar valores fijos de corriente y voltaje en el dispositivo. Antes de llamar a esta función, ya se comprobó que los valores ingresados por el usuario sean números y que estos se encuentran dentro de los rangos permitidos por el equipo.

setInterval(self)

Método que almacena toda la información proporcionada por el usuario, para configurar un barrido de corriente o voltaje, en una lista. Una vez recolectada toda la información se llama al método *setInterval* de la clase Action y se envían como parámetros estos valores.

sweepStopped(self)

Método utilizado para restaurar la interfaz gráfica a su estado original cuando un barrido se detiene. Este método es llamado cuando se quiere detener el barrido, o este ha terminado al cumplir su ciclo.

pauseSweep(self)

Método que maneja el evento generado al presionar el botón de “Pausar Secuencia” que se muestra sólo cuando un barrido está en progreso. El método pausa o reanuda un barrido de corriente o voltaje.

setStep(self)

Método llamado después de comprobar que todos los datos ingresados por el usuario sean números y se encuentren dentro del rango permitido para el dispositivo en un barrido de corriente - voltaje de tipo maestro - esclavo. Se llama al método *setStep* de la clase *Action*, y se envía como parámetro que valor es el maestro (corriente o voltaje).

setListeners(self, ui)

Método llamado al comienzo de la ejecución del programa. Recibe como argumento, una referencia a la ventana principal que contiene todos los elementos de la interfaz gráfica, y se asignan manejadores de eventos para cada una de las acciones que se quieren capturar del usuario, principalmente clics sobre botones.

Clase Action

Clase que contiene los métodos para interactuar con el equipo físico. Usando una instancia de la clase *Instrument* que es parte de la librería PyVisa, se emiten comandos al dispositivo por medio de la interfaz GPIB. Esta clase se encuentra en el archivo actions.py del código fuente para la aplicación de fuente de poder. A continuación se describe la funcionalidad de cada uno de los métodos en la clase.

__init__(self, name, ui, interface)

Este es el constructor de la clase *Action*, y recibe como argumento una referencia a la instancia que se ha creado de la clase (*self*), el alias o dirección GPIB del dispositivo, referencia a la ventana principal donde se encuentran todos los componentes gráficos de la aplicación (*ui*) y por último, una referencia a la instancia de la clase *Wrapper* desde donde se llama a este constructor. Este método inicializa todas las variables de la clase, y también cambia el modo de operación de la fuente de poder a bajo voltaje, es decir un máximo de 25.75 voltios.

setOutput(self, output)

Método que emite dos comandos al dispositivo dependiendo si el argumento recibido es verdadero o falso. Si es verdadero, activa el Output del equipo y si es falso, lo desactiva. Es decir, activa o desactiva el paso de corriente por los terminales de la fuente de poder.

setFixed(self)

Este método configura una corriente y voltaje fijos en el dispositivo. Antes de emitir el comando de configuración verifica en qué rango de voltaje se debe configurar a la fuente (alto o bajo).

setInterval(self, VSweep, CSweep, values)

Este método recibe como parámetros, si se ha seleccionado barrido de corriente o voltaje, además de una lista con los valores necesarios para iniciar un barrido, que se recogieron y validaron de la interfaz gráfica. Esta información se recoge en el método *setValues* de la clase Wrapper.

stopInterval(self)

Método llamado para detener un barrido de corriente o voltaje que se encuentra en ejecución. Primero comprueba que se haya creado una instancia de barrido en la clase, para después ordenar al objeto de barrido de la clase *Sweeper* que se detenga.

setStep(self, master)

Método encargado principalmente de iniciar un nuevo hilo de ejecución que va a manejar el barrido de corriente-voltaje de tipo maestro-esclavo. Desde aquí se llama al método *cVStepper* donde se encuentra toda la lógica del barrido.

cVStepper(self)

En este método se implementó toda la lógica del barrido de corriente-voltaje de tipo maestro-esclavo. El método inicializa variables y controla que se emitan comandos al dispositivo en intervalos de tiempo que el usuario especifica desde la interfaz gráfica. También, una vez que se llega a los valores finales del barrido, se actualiza la interfaz gráfica para mostrar que el barrido ha terminado.

Clase Sweeper

Clase implementada para iniciar barridos de corriente o voltaje de manera sencilla, se encuentra en el archivo funtions.py del código fuente. Esta clase reemplazó a la implementación inicial de un barrido de corriente o voltaje para la fuente de poder. La clase *Sweeper* recibe información ingresada por el usuario para crear un objeto de barrido, la misma clase se encarga de validar que se hayan ingresado valores dentro de los rangos permitidos por los dispositivos y levanta una excepción, en caso de que uno de los valores ingresados sea incorrecto.

__init__(self, device, sweepType, fixed, sweepStart, sweepStop, sweepStep, sweepTime)

El constructor de esta clase en particular es el que maneja gran parte de la lógica de la misma. Se reciben como parámetros una referencia al objeto de la clase llamante (device), el tipo de barrido (corriente o voltaje), y una serie de valores necesarios para configurar el barrido, sea este de corriente o voltaje. En el mismo constructor se valida que la información ingresada por el usuario sea correcta, de encontrarse algún error, se levanta una excepción capturada por el controlador de la aplicación. Las últimas acciones que realiza este método son fijar los valores iniciales para el barrido, es otro método el que inicia este proceso en un hilo de ejecución diferente.

startSweep(self)

Método que inicia el método stepper en un hilo de ejecución diferente al de la interfaz gráfica. Este método es llamado cuando se inicia un barrido, o cuando se reanuda uno que se encontraba en pausa.

stepper(self)

Método donde se implementa la lógica del barrido, es decir, donde se emiten comandos al equipo en los intervalos definidos para los valores de configuración seleccionados por el usuario. Al finalizar el barrido se actualiza la interfaz gráfica para mostrar que el barrido ha terminado

pauseSweep(self)

Método que cambia el estado de dos variables. Esto ocasiona que el método *stepper* que estaba corriendo en otro hilo de ejecución se detenga y no envíe más comandos al equipo. Además, una de las variables permite que este barrido sea reanudado.

resumeSweep(self)

Cambia el estado a las variables que indican que el barrido está en pausa y además llama al método para iniciar el barrido, como el estado del objeto *Sweeper* es el mismo que cuando se puso pausa, el barrido continúa desde donde se detuvo por última vez.

stopSweep(self)

Método que puede ser llamado en cualquier momento durante un barrido de corriente o voltaje. Detiene el hilo de ejecución que emitía los comandos de barrido periódicamente y también actualiza la interfaz gráfica para mostrar que se ha detenido el barrido.

Clase Definitions

Esta clase simplemente contiene variables estáticas para ser usadas en la validación de datos y despliegue de mensajes en varios métodos de la aplicación. En esta clase se deben modificar los valores permitidos por el equipo, de ser necesario.

Clase ValueException

Clase que hereda de la clase *Exception*, que forma parte de la librería estándar de Python2.7. Se utiliza simplemente para levantar excepciones con un mensaje descriptivo específico de la razón por la que esta fue generada en tiempo de ejecución.

Anexo 2: Detalle de Aplicación Para Generador de Ondas

Clase Wrapper

De la misma manera que en la fuente de poder, la clase *Wrapper* es el controlador de la aplicación, es decir que desde ella se manipula la interfaz de usuario mediante referencias a la clase de interfaz gráfica y también se llama a métodos de la clase *Action* que se encarga de emitir comandos al dispositivo físico.

La clase *Wrapper* es más simple en el generador de ondas que en la fuente de poder, pues para el diseño de la interfaz se utilizó una vista con pestañas para mostrar cada una de las funciones de la aplicación, a diferencia de la fuente de poder en la que sobre una sola vista, se muestran diferentes componentes mostrándolos y ocultándolos dinámicamente, y todo esto se controla en el código de la clase *Wrapper*.

Los siguientes métodos son similares a los que se utilizaron en la aplicación de fuente de poder, o al menos realizan la misma función para la aplicación:

- __init__(self, ui)
- createInstrument(self, deviceAddress = "NoConfig", setType="NoType")
- connect(self)
- writeConfig(self, deviceAddress)
- getConfigAddress(self)
- setStatus(self,status, label)
- setOutput(self)
- setDisabledStatus(self,status)
- setButtonLabel(self, button, text)
- displayMessage(self, mType, message)
- is_number(self, s)
- setListeners(self)

A continuación se describe la función de cada uno de los métodos de esta clase que tienen un comportamiento diferente al que vimos en la aplicación de fuente de poder.

Clase Device

Esta clase es el equivalente a la clase Action utilizada en la aplicación para la fuente de poder. Aquí se encuentran implementados los métodos que emitirán comandos al equipo físico por medio de la interfaz GPIB usando la clase Instrument, que es parte de la librería PyVisa. En estos métodos también se implementan procedimientos tales como el barrido de frecuencia. Esta clase se encuentra en el archivo actions.py del código fuente.

Los siguientes métodos son similares a los que se utilizaron en la aplicación de fuente de poder, o al menos realizan la misma función dentro de la aplicación:

- __init__(self, name, interface)
- setOutput(self, output)

setFixed(self, waveName, freq, amp, offset)

Método que recibe información que ya ha sido validada en la clase controladora *Wrapper*. Verifica dos condiciones simples para emitir comandos al dispositivo, y una vez construida la cadena de texto con el formato específico se fija una configuración estática en el dispositivo.

startSweep(self, waveName, freqStart, freqStop, time, amp, offset, sweepMode, step=0)

Método que recibe información validada en la clase controladora. Primero fija los valores iniciales del barrido en el dispositivo, y luego, dependiendo si el barrido es lineal o logarítmico, llamará al método que inicia cada uno de estos barridos con los parámetros que recibe desde este método.

startInterval(self)

Método al que se llama en caso de que se esté configurando un barrido lineal. El método *startSweep* configura todos los parámetros del barrido lineal modificando parámetros de la clase *Device*, y al llamar a este método, se inicia la secuencia de barrido en un hilo de ejecución diferente al de la interfaz gráfica.

stepper(self)

Método que se ejecuta en un hilo separado al hilo de ejecución de la interfaz gráfica. Aquí se implementa la lógica del barrido lineal de frecuencia en el generador de ondas. Una vez que se cumplen las condiciones de terminación del barrido, se detiene el barrido y se actualiza la interfaz gráfica para mostrar que este ha terminado.

startLogSweep(self, freqStart, freqStop, time)

Método llamado desde *startSweep* en caso de que se haya escogido un barrido de frecuencia logarítmico. Aquí se emite una serie de comandos que configuran al equipo generador de ondas para entrar en modo de barrido. Este barrido está implementado en el mismo dispositivo y se repetirá hasta que se lo detenga. Por esta razón, se inicia un hilo de ejecución separado de la interfaz gráfica, que una vez cumplido el tiempo de duración del barrido especificado por el usuario, detiene el barrido en el equipo.

logSweepTimer(self)

Método desde donde se inicia un hilo de ejecución separado de la interfaz gráfica con el fin de detener el barrido de frecuencia logarítmico una vez que este haya alcanzado la duración definida por el usuario.

stopLogSweep(self)

En este método se detiene el hilo de ejecución que se creó separado de aquel de la interfaz gráfica durante el tiempo especificado por el usuario. Una vez que ha pasado este

tiempo, se emite un comando al equipo generador de ondas para detener el barrido que está en ejecución en ese momento

setArbWave(self, freq, points)

Este método recibe como parámetros la lista de puntos recolectados en la clase controladora *Wrapper*, para configurar una onda arbitraria en el dispositivo. La lista de puntos, junto con la frecuencia escogida por el usuario son suficientes para calcular los valores que se deben enviar al dispositivo, con el fin de configurar la onda arbitraria que el usuario escogió.

Clase Definitions

Al igual que en la aplicación para la fuente de poder, la clase *Definitions* contiene variables estáticas con valores que principalmente servirán para validar que los valores ingresados por el usuario estén dentro de los rangos permitidos por el equipo físico. Es en esta clase donde se deben manipular valores máximos y mínimos para cada una de las configuraciones que puede manejar un equipo de laboratorio.

Anexo 3: Manual de Instalación Aplicaciones Para Fuente de Poder y Generador de Ondas

Paso 1

El primer instalador que se debe ejecutar en el proceso de instalación es el de la versión 2.7 de Python. Este instalador puede ser descargado para sistemas de 32 o 64 bits en la página http://www.python.org/download/releases/2.7.6/

En los recursos digitales de este documento se encuentra una carpeta de instaladores con las versiones que se utilizaron para probar las aplicaciones, pero no existe razón para que estas no funcionen con una versión más reciente de cualquiera de ellos.

En los recursos digitales se debe ejecutar el instalador de Windows *python-2.7.6.msi* (32 bits) o *python-2.7.6.amd64.msi* (64 bits). En la primera pantalla del instalador, se presenta la opción al usuario de instalar Python para todos los usuarios del equipo, o sólo para el actual. Debido a un error en el instalador de Python, se debe escoger la opción instalar sólo para el usuario actual, de tal manera que los demás instaladores puedan detectar la instalación de Python en el sistema. Después de esta pantalla se deben escoger las opciones por defecto de la instalación hasta que esta termine. Es importante, durante la instalación de Python, recordar el directorio en donde se realiza la instalación. En caso de no escoger el directorio por defecto (C:\Python27), se deberá seleccionar este mismo directorio en la instalación de PyQt y PyVisa.

Una vez instalado Python, se puede continuar con la instalación del resto de librerías necesarias para la instalación de la aplicación.

Paso 2

El siguiente instalador que se debe ejecutar es *PyQt4-4.10.3-gpl-Py2.7-Qt4.8.5-x32.exe* (32 bits) o *PyQt4-4.10.3-gpl-Py2.7-Qt4.8.5-x64.exe* (64 bits).

En este instalador se debe especificar el directorio de la instalación de Python que se escogió en el paso 1. Si no se cambiaron las opciones por defecto, este directorio es C:\Python27. Luego de aceptar todas las opciones por defecto y llegar al último paso de la instalación, esta está terminada y se puede continuar con la instalación de los drivers de Agilent y la librería PyVisa. En los recursos digitales de este documento se encuentra la versión probada de estos archivos, pero diferentes versiones pueden ser descargadas del sitio http://www.riverbankcomputing.com/software/pyqt/download

Paso 3

Antes de instalar la librería PyVisa para el manejo de conexiones con dispositivos por medio de una interfaz GPIB, se debe instalar primero los drivers para el cable USB/GPIB que se usó en este proyecto. En el caso del controlador del cable de marca Agilent, este incluye una implementación de la librería de comunicación estándar VISA (Virtual Instrument Software Architecture). PyVisa simplemente actúa como una interfaz para el intérprete de Python con esta librería, que debe estar instalada antes de poder hacer llamadas a la librería de Python, caso contrario se produce un error del sistema en donde no se encuentra el componente VISA.

En los recursos digitales de este documento podemos encontrar una versión de la librería y controlador para el cable de Agilent USB/GPIB, pero este contenido puede ser descargado de la página http://www.agilent.com/find/iosuiteproductcounter.

Para instalar el controlador se debe ejecutar el archivo IOLibSuite_16_3_17914.exe de los recursos digitales. Se deben seguir todos los pasos de la instalación con las opciones por defecto, en caso de que ya exista instalada en el equipo una versión de la librería VISA, se preguntará si se quiere usar la de Agilent como principal o primaria. Para el uso de estas aplicaciones se recomienda usar la de Agilent como primaria.

Una vez terminada la instalación, se puede conectar el cable USB/GPIB al equipo, la primera vez se instalará el controlador. Luego se debe abrir uno de los aplicativos instalados en el sistema *Agilent Connection Expert*. En esa aplicación se pueden manejar todos los dispositivos conectados por medio de su interfaz GPIB al computador. Cuando se conecte el equipo a la fuente de poder o generador de ondas por primera vez, se puede acceder a esta aplicación para verificar la dirección GPIB en el sistema o para designar un alias de conexión a cualquiera de los dos equipos. Si se conoce la dirección GPIB del equipo, no es necesario ejecutar esta aplicación para que las aplicaciones funcionen.

Paso 4

La última librería que se debe instalar para el funcionamiento de las aplicaciones desarrolladas en este proyecto, es PyVisa. Para esto se debe ejecutar el instalador *PyVISA-1.4.win32.exe* (32 ó 64 bits). Este instalador se encuentra en el contenido digital de este documento, pero fue descargado del sitio oficial del proyecto en http://sourceforge.net/projects/pyvisa/.

En el instalador se debe verificar que el directorio de instalación escogido sea el mismo que se utilizó en la instalación de Python, el cual por defecto es C:\Python27. Es en esta librería donde si no se escogió instalación únicamente para el usuario actual en el paso 1, no se encontrará una instalación válida de Python en el sistema. Luego de seguir los pasos de instalación, esta termina y las librerías están listas para ejecutar el programa.

Paso 5

Ahora el sistema está listo para ejecutar los archivos de la aplicación desarrollada en este proyecto. Python es un lenguaje que permite ejecutar aplicaciones como scripts directamente desde el código fuente de la aplicación, o también desde archivo previamente compilados. En los recursos digitales entregados como parte del documento, encontramos

dentro de la carpeta *Instaladores Aplicaciones* dos carpetas, una llamada "*compilados*" y otra llamada "*fuente*".

Para evitar que cualquier usuario modifique el código fuente de la aplicación que se instale en los equipos, es recomendable usar los archivos dentro de la carpeta compilados para la instalación. Pero, Python permite que se ejecute la aplicación directamente desde el código fuente si el usuario así lo desea. En este caso tomaremos como ejemplo los archivos dentro de la carpeta compilados para la instalación.

Se debe copiar todo el contenido de la carpeta instaladores/compilados en un directorio escogido por el usuario dentro del sistema donde se instala la aplicación, es decir las dos carpetas contenidas en este directorio (fuente de poder y generador de ondas). Una vez que se han copiado los archivos, ejecutar el programa es tan simple como ejecutar el archivo *interfaceMethods.pyc* por medio de la línea de comandos o simplemente haciendo doble clic en él para cada una de las aplicaciones (fuente de poder y generador de ondas), cada archivo está dentro del directorio correspondiente a cada equipo. Se recomienda crear un acceso directo al archivo *interfaceMethods.pyc* de cada aplicación en un lugar más accesible para el usuario, de tal manera que sólo se deba hacer doble clic en el acceso directo para ejecutar la aplicación.

Nota: En caso de usar el código fuente para la ejecución del programa, se deben copiar todos los archivos dentro de la carpeta fuente a un directorio del sistema, y el acceso directo debe ser creado para el archivo *interfaceMethods.py*.