DISEÑO E IMPLEMENTACIÓN DE UN MÓDULO EFICIENTE MODULADOR/ DEMODULADOR QAM PARA GNU RADIO

DESIGN AND IMPLEMENTATION OF AN EFFICIENT QAM MODULATOR/DEMODULATOR MODULE FOR GNU RADIO

AUTOR
ORLANDO ARBOLEDA MOLINA
Magister en Ingeniería
*Universidad Autónoma de Occidente
Docente
Departamento de Operaciones y Sistemas
oarboleda@uao.edu.co
COLOMBIA

AUTOR
HELMUT ALEXANDER RUBIO
Magister en Ingeniería
*Universidad Autónoma de Occidente
Docente
Departamento Automática y Electrónica
harubio@uao.edu.co
COLOMBIA

AUTOR
LEANDRO ANTONIO VILLA BARONA
Magister en Automática
**Universidad del Valle
Docente
Escuela de Ingeniería Eléctrica y Electrónica
Leandro.villa@correounivalle.edu.co
COLOMBIA

***INSTITUCIÓN**
Universidad Autónoma de Occidente
UAO
Carácter institucional
Cll 25 No. 115-85. Km 2 vía Cali-Jamundí
buzon@uao.edu.co
COLOMBIA

****INSTITUCIÓN**
Universidad del Valle
UNIVALLE
Carácter institucional
Cll 13 No 100 - 00
secretaria.eiee@correounivalle.edu.co
COLOMBIA

INFORMACIÓN DE LA INVESTIGACIÓN O DEL PROYECTO: Arquitectura abierta de Radio Definida por Software (SDR) para la implementación de técnicas de modulación digital empleadas en comunicaciones inalámbricas.

RECEPCIÓN: Julio 28 de 2016 **ACEPTACIÓN:** Noviembre 23 de 2016

TEMÁTICA: Sistemas inalámbricos y móviles, Comunicaciones Móviles, Telecomunicaciones, Acceso y Conectividad inalámbrica

TIPO DE ARTÍCULO: Artículo de Investigación Científica e Innovación

Forma de citar: Arboleda, Molina. O. (2016). Diseño e implementación de un módulo eficiente modulador/ demodulador QAM para GNU Radio. En R, Llamosa Villalba (Ed.). Revista Gerencia Tecnológica Informática, 15(43), 63-77. ISSN 1657-8236.

RESUMEN ANALÍTICO

GNU Radio es una plataforma muy popular de código libre y abierto, de desarrollo para SDR, que cuenta con bloques genéricos de procesamiento digital de señales para la modulación y demodulación QAM. En estos bloques, el mapeo se hace empleando constelaciones cuya relación símbolo–complejo se basa en la posición dentro de los arreglos suministrados, obligando al ingreso ordenado, de parte del usuario, de los complejos correspondientes a cada símbolo. Esto mejora la complejidad en el proceso de modulación pero la degrada en la demodulación. El presente artículo presenta el diseño e implementación de un módulo modulador/demodulador alternativo para GNU Radio que proporciona bloques para mapeo de símbolos a complejos, mapeo de complejo a símbolos y modulación/demodulación QAM de orden variable que aumenta la eficiencia en el proceso de demodulación y elimina el error potencial del usuario al ingresar los valores de la constelación en el arreglo de datos.

PALABRAS CLAVES: Radio definida por software (SDR), modulación digital, Gnu Radio, QAM.

ANALYTICAL SUMMARY

GNU Radio is a very popular free and open source development platform for SDR. It has generic digital signal processing blocks for QAM modulation and demodulation. In these blocks, mapping is done using constellations whose symbol-complex relationship is based on position within arrays provided, forcing user to insert each complex and symbol tidily. This improves the complexity in the modulation process but degrades it in demodulation. The present paper presents the design and implementation of an alternative modulator / demodulator module for GNU Radio, which provides symbol-to-complex mapping, complex-to-symbol mapping and variable order QAM modulation / demodulation blocks that increases efficiency in demodulation process and eliminates potential user error when constellation values are inserted in data array.

KEYWORDS: Software defined radio (SDR), digital modulation, GNU Radio, QAM.

INTRODUCCIÓN

Los sistemas de Radio Definida por Software (SDR) son sistemas de telecomunicaciones en los que gran parte del procesamiento de señal se lleva a cabo en procesadores de propósito general (GPP), lo que ofrece un alto grado de flexibilidad al permitir modificar, a voluntad, el procesamiento digital de las señales por medio de cambios a nivel de software y, por ello, se está convirtiendo en una de las tecnologías más empleadas para implementación de comunicaciones por radio. Una posible aplicación en la que los sistemas SDR podrían aprovecharse por su flexibilidad, sería en la implementación de terminales de telefonía móvil donde conviven diversos protocolos y tecnologías de transmisión inalámbrica que se están actualizando continuamente [1].

El alto nivel de reconfiguración y su bajo costo, entre otras ventajas, hacen que los sistemas SDR sean muy atractivos en el entorno académico [2], pues permiten el estudio y el aprendizaje del procesamiento digital de señal sin la necesidad de requerir grandes inversiones en hardware específico. Adicionalmente, se debe tener en cuenta que existen plataformas de software libres y abiertas tales como GNU Radio, que facilitan la investigación en el campo de las radiocomunicaciones ya que permiten implementar prototipos de manera rápida para experimentar y evaluar el funcionamiento de nuevas tecnologías de comunicaciones y procesamiento de señales.

En GNU Radio existen bloques para modulación digital que son genéricos, poco documentados, muy rígidos en las constelaciones aceptadas que les permiten ser eficientes en la modulación pero ineficientes en sus procesos de demodulación y que ofrecen poca flexibilidad respecto a las constelaciones que pueden utilizarse [3] y [4].

En la presente publicación se expone el diseño e implementación del módulo GNU Radio srd-UAO_UV, el cual proporciona bloques eficientes de mapeo de símbolos, mapeo de complejos, modulación

y demodulación QAM que permiten trabajar con constelaciones de diverso orden. Para comprobar su correcto funcionamiento, se emplearon constelaciones QAM con símbolos de 4 y 5 bits.

1. CONCEPTOS PRELIMINARES

1.1 GNU RADIO

GNU Radio es una caja de herramientas de desarrollo de software gratuito y de código abierto que proporciona bloques de procesamiento de señales para implementar radios por software, la cual puede usarse con hardware RF externo de bajo costo para crear radios definidos por software o sin hardware en un ambiente de simulación. Es ampliamente usado en ambientes aficionados, académicos y comerciales para soportar tanto la investigación de comunicaciones inalámbricas como los sistemas de radio del mundo real [5].

GNU Radio ejecuta todo el procesamiento de la señal, a través de bloques (ej. filtros, demoduladores, etc) interconectados entre sí. Igualmente, incluye un método para conectar los bloques y gestionar como los datos se pasan de un bloque a otro.

Al ser GNU Radio un software, sólo puede manipular datos digitales; por lo tanto, las aplicaciones que pueden ser construidas reciben datos desde flujos digitales o colocan datos en flujos digitales, los cuales pueden ser convertidos en señales analógicas de RF y posteriormente, transmitidos usando hardware como el periférico universal de radio definido por software USRP [6].

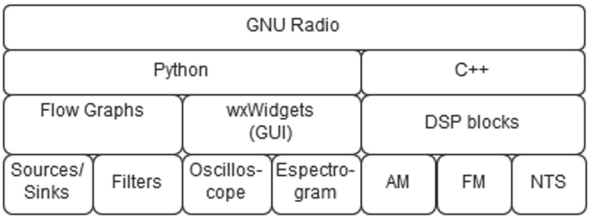
En la Figura 1 se indican los componentes principales de la plataforma GNU Radio, en la cual se pueden usar los lenguajes de programación Python o C++ para la construcción de aplicaciones o bloques de procesamiento de señales (DSP) respectivamente.

En GNU Radio, los bloques hacen parte de los módulos (similar a un paquete) y están pensados para realizar un único trabajo. Los bloques deben desarrollar tareas específicas de procesamiento de señales y en la mayoría de los casos, son construidos en C++ para tener un mejor desempeño. De esta manera, GNU Radio se conserva modular y flexible [7].

1.2 GRC

Como se puede apreciar en la Figura 1, la mayoría de aplicaciones en GNU Radio están compuestas por grafos de flujo (*Flow Graphs*) construidos en Phyton que conectan bloques para formar un software de radio funcional.

FIGURA 1. Vista abstracta de los mayores componentes de GNU Radio.



Fuente: Elaboración propia adaptada de [8].

GNU Radio Companion o GRC es la interfaz gráfica existente en GNU Radio, similar a Simulink, que permite crear las aplicaciones de procesamiento de señales o flujos de datos, mediante el mecanismo drag-and-drop, para generar el correspondiente código en Python del grafo de flujo deseado [9].

1.3 MODULACION DIGITAL EN GNU RADIO

Para modulación digital, GNU Radio ofrece los bloques **modulador** (*generic_mod*) [3] y **demodulador** (*generic_demod*) [4], genéricos, cuyo funcionamiento depende de la **constelación** (*constellation*) suministrada [10], ya que esta determina la técnica de modulación implementada.

La constelación, que es una de las entradas a los bloques moduladores y demoduladores, contiene el mapa de asignación entre los símbolos digitales, grupos de k bits representados mediante datos de tipo entero sin signo y los números complejos (puntos de la constelación) que conforman la envolvente compleja de la señal modulada. En el argumento *dimensionality* de los bloques moduladores y demoduladores se indica el número de complejos que se mapean a cada símbolo. Si se desea una nueva técnica de modulación digital en GNU Radio, se debe definir un objeto de tipo *constellation* con datos y métodos de procesamiento en C++ particulares para la técnica de modulación deseada.

Actualmente, en las constelaciones GNU Radio utilizadas para la modulación y demodulación QAM, no se hace uso de los valores de los símbolos suministrados y los procesos implementados se basan en que el valor de los símbolos ingresados corresponden a la posición en que son almacenados (ej. el símbolo 0 en la posición 0, el símbolo 1 en la posición 1, etc), obligando al ingreso ordenado de los complejos correspondientes a cada símbolo.

En GNU Radio, el modulador *generic_mod* incluye un bloque **mapeador de símbolos** (*chunks_to_symbols_bc*) a sus correspondientes complejos [11], con un

algoritmo de búsqueda de complejidad constante Θ(1) porque aprovecha que las constelaciones están ordenadas y retorna los complejos almacenados a partir de la posición correspondiente al símbolo.

El demodulador *generic_demod* incluye un bloque **mapeador de complejos** (*constellation_receiver_cb*) que reconstruye el símbolo de salida a partir del complejo recibido, comparándolo contra todos los complejos de la constelación para determinar cuál se encuentra más cercano y recuperar su posición (símbolo buscado). Esto le permite tener un algoritmo de búsqueda lineal Θ(n) con un recorrido sobre todos los complejos ingresados.

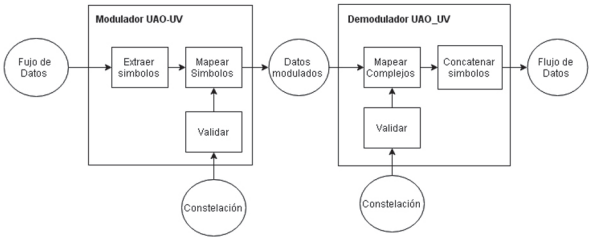
2. MÓDULO SDR-UAO_UV

El nuevo módulo GNU Radio desarrollado, que se denomina sdr-UAO_UV[1], proporciona bloques GNU Radio para los subprocesos de mapeo de símbolos y mapeo de complejos, a partir de los cuales se construyen los bloques para los procesos de modulación y demodulación para constelaciones QAM con tamaño de símbolo variable, no ordenadas, a las cuales se le aplica un subproceso de validación para evitar errores en el ingreso de las constelaciones y de ser necesario, las reorganiza para hacer más eficiente el proceso de demodulación.

2.1 MODELO

Como se indica en el diagrama de flujo de la Figura 2, el módulo sdr-UAO_UV está compuesto de los procesos modulador (moduladorUAO_UV) y demodulador (demoduladorUAO_UV), construidos a partir de la integración de los subprocesos: Validar, Extraer símbolos, Mapear símbolos (chunks_to_symbols_bc), Mapear complejos (constellation_receiver_cb) y Concatenar símbolos.

FIGURA 2. Procesos del sdr-UAO_UV



Los flujos de información procesados por los bloques son:

- **Flujo de datos:** muestra de 8 bits, que representa la información a enviarse.
- **Datos modulados**: complejo de 32 bits que representa la información modulada.
- **Constelación:** bloque Constellation Object de Gnu Radio, al que se le suministran los símbolos de 8 bits en el parámetro Symbol Map, sus complejos de 32 bits (16 bits para I, 16 bit para Q) en el parámetro Constellation Points, y la dimensión en el parámetro Dimensionality que debe ser definida en 1.

2.2 SUBPROCESO VALIDAR

Es el subproceso ejecutado al inicio del modulador y demodulador propuesto, el cual se encarga de verificar que la constelación ingresada sea correcta y si es necesario reorganiza sus datos para que el proceso de demodulación sea eficiente.

A diferencia de las constelaciones usadas para el modulador y demodulador genérico de GNU Radio, en la que los valores de los símbolos no son usados porque se asocian a la posición, se propone, como mejora, que las constelaciones para el módulo **sdr-UAO-UV** contengan símbolos con valores entre 0 y N-1 (donde N es el tamaño de la constelación) y que no se requiera que sean suministrados en forma ordenada, permitiendo que sus correspondientes complejos pueden ser ingresados sin estar agrupados por cuadrantes u organizados en un orden particular.

Este subproceso recibe la constelación QAM y una variable booleana denominada *aplicarOrdenamiento* en la que se indica si se desea o no ordenar los datos de la constelación. Las tareas de verificación que realiza este subproceso son:

1. Que los símbolos estén en el rango [0,N-1].
2. Que todos los símbolos sean diferentes.
3. Que exista igual cantidad de complejos por cuadrante.

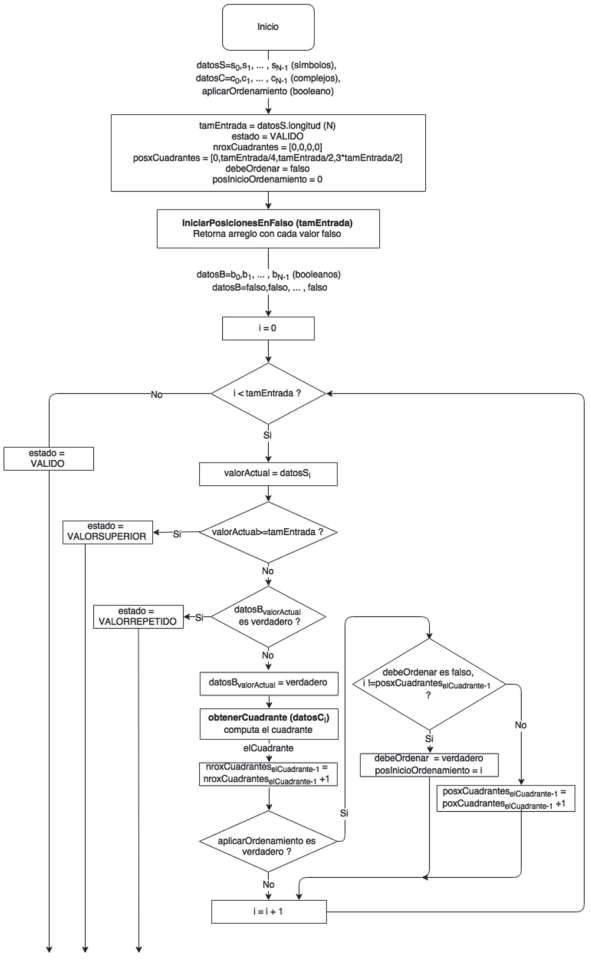
En caso de que **validar** sea invocado con la variable *aplicarOrdenamiento* en *verdadero*, también se valida si es necesario ordenar la constelación en los cuadrantes del plano cartesiano, según los signos de los complejos ingresados, para que se agrupen consecutivamente desde el cuadrante 1 hasta el 4.

La Figura 3 muestra el diagrama de flujo de la parte inicial del proceso **validar**, en la que se recorren los arreglos de símbolos y complejos de la constelación realizando las validaciones nombradas previamente. Dependiendo del tipo de validación que se esté realizando, si alguna verificación no se cumple, la variable estado cambia de "VALIDO" a "VALORREPETIDO" o "VALORSUPERIOR". Adicionalmente, cuando el validador es invocado con *aplicarOrdenamiento* en verdadero, en la variable

1 http://ingenieria.uao.edu.co/sdr/ImplementacionFinal_sdr_UAO_UV.zip

debeOrdenar se determina si realmente es necesario ordenar la constelación por cuadrantes.

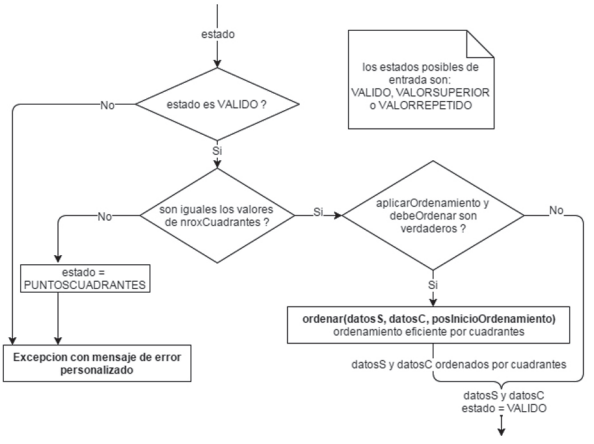
FIGURA 3. Diagrama de flujo del subproceso validar – recorrido de la constelación



Las acciones que realiza **validar**, según el valor de la variable estado, se muestran en la Figura 4. En caso de que alguna de las validaciones realizadas arroje un resultado negativo, el algoritmo genera una excepción con el mensaje de error correspondiente. Adicionalmente, según el estado de las variables aplicarOrdenamiento y debeOrdenar, se hace efectivo o no el ordenamiento de la constelación.

En las Tablas 1 y 2 se muestran tanto las variables como el algoritmo requerido en la implementación del subproceso validar. El único estado a retornar es VALIDO, porque, de encontrarse un error de la entrada, en la verificación se lanzará una excepción que detendrá al validador y al modulador o demodulador que lo invocó. Adicionalmente, este proceso puede reorganizar la constelación para que quede ordenada de forma creciente por cuadrantes según los complejos almacenados.

FIGURA 4. Diagrama de flujo del subproceso validar – acciones que se ejecutan según la validación



Para la comprensión del pseudocódigo del subproceso validar de la Tabla 2, se debe tener en cuenta que:

- Los arreglos son indexados desde la posición 0
- Se usan los cuatro cuadrantes del plano cartesiano.
- En el arreglo *nroxCuadrantes* se cuenta el número de complejos en cada cuadrante.
- El arreglo *posxCuadrantes* se usa para verificar que los complejos de la constelación sean clasificados en los cuadrantes (1,2,3,4) y almacenados en los rangos respectivos.
- El subproceso *iniciarPosicionesEnFalso* retorna un arreglo de tamaño tamEntrada con cada posición en falso, indicando que hasta ese punto del algoritmo, no ha sido procesado el símbolo i. Como requiere recorrer una sola vez el arreglo de datos, su complejidad es O(n).
- Se realiza un único recorrido por todos los datos de entrada (bloque Procesar-Repetir), verificando que los símbolos estén en el rango [0,N-1] y no se hayan repetido. Igualmente, logra determinar si en realidad *debeOrdenar* y desde que posición (posInicioOrdenamiento).
- Después del único recorrido por los datos de entrada, verifica que cada cuadrante tenga igual cantidad de puntos y si es necesario, posteriormente ejecuta el ordenamiento de los datos ingresados desde la posición computada. Finalmente, se lanza una excepción con el mensaje descriptivo o se retorna estado VALIDO.

La complejidad total del validador ($c_{validor}$), que depende de la complejidad para iniciar las posiciones en falso ($c_{inicPosFalso}$) y el procesamiento propiamente dicho de

validación ($c_{procesar}$), en el peor caso, que es cuando debe ordenar ($c_{ordenar}$), es:

$$C_{validor} = C_{inicPosFalso} + C_{procesar} + C_{ordenar}$$

$$C_{validor} = O(n) + O(n) + O(n) \quad (1)$$

$$C_{validor} = O(n)$$

2.3 PROCESO MODULADORUAO_UV

El bloque moduladorUAO_UV recibe el flujo datos binario (en paquetes de 8 bits) y entrega los datos modulados según la constelación recibida que se valida al inicio del proceso.

En su diagrama de flujo, indicado en la Figura 5, se aprecia que el modulador propuesto realiza la validación de la constelación una sola vez al inicio del proceso, sin solicitar que se aplique ordenamiento, porque solo son necesarias las tareas de verificación indicadas en la sección 2.2 y no que la constelación suministrada este ordenada, ni que deba ordenarse. Después de realizada la validación de la constelación, el modulador se encarga de extraer los símbolos continuamente del flujo de datos de entrada, haciendo uso del subproceso **extraerSimbolo** para luego realizar la conversión de cada símbolo obtenido a su complejo correspondiente por medio de la búsqueda en una estructura dinámica en el bloque **mapearSimbolo.**

TABLA 1. Variables de entrada, temporales y de salida del subproceso Validar

Subproceso: validar					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identif**
Símbolo[] Complejo[] Booleano	Símbolos de la constelación (datosS) Complejos de la constelación (datosC) Determina si se desea que el validador ordene (aplicarOrdenamiento)	Entero Entero[4] Entero[4] Booleano Entero Booleano[] Entero Símbolo Entero	Número de símbolos (tamEntrada) Número de complejos por cuadrante (nroxCuadrantes) Posición máxima complejos por cada cuadrante (posxCuadrantes) Si complejos deberían ser ordenados por cuadrante (debeOrdenar) Posición inicial del complejo fuera de su cuadrante (posInicioOrdenamiento) Arreglo para validar si el símbolo i ya fue procesado (datosB) Posición (i) Símbolo procesado (valor Actual) Cuadrante complejo procesado (elCuadrante)	Entero	estado

Como se observa en el pseudocódigo de la Tabla 3, la entrada inicial al bloque moduladorUAO_UV es un objeto constelación (Cu) que contendrá el arreglo de símbolos (Cu.S) y su correspondiente arreglo de complejos (Cu.C), el cual será enviado inicialmente al Validador. Si la validación se logra superar (estado retornado como VALIDO), el modulador queda disponible para procesar cada flujo de bytes, denominado Dm, en el que se han empaquetado varios símbolos, cada uno del tamaño bits por símbolo que se obtiene de la constelación (Cu.K).

2.3.1 Subproceso Extraer símbolos

Obtiene la cantidad de bits necesarios, según el tamaño del símbolo en la constelación, de cada muestra del flujo de datos para formar el símbolo binario que se entrega al mapeador de símbolos.

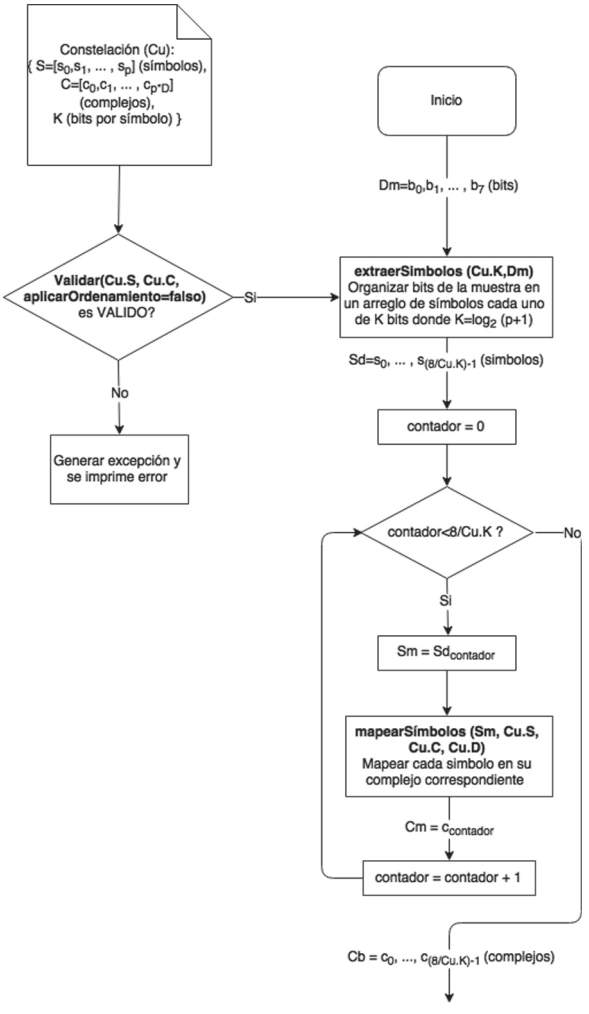
Para cada byte suministrado Dm, se debe aplicar un subproceso extraerSímbolos, con el fin de extraer cada uno de los símbolos que fueron representados en K bits y agrupados de izquierda a derecha para luego regresarlos en el arreglo Sd como se observa en la Tabla 4. Posteriormente, cada símbolo es mapeado al complejo correspondiente a través del subproceso **mapearSimbolos**.

En la implementación realizada se aprovechó el bloque *packed_to_unpacked_bb* existente en el módulo blocks suministrado por GNU Radio e indicó que la alineación correspondiente era también de izquierda a derecha (*gr. GR_MSB_FIRST*).

TABLA 2. Algoritmo del subproceso Validar

```
Leer datos
 Leer datosC
 Leer aplicarOrdenamiento

 Asignar datosS.longitud a tamEntrada
 Asignar VALIDO en estado
 Asignar [0, 0, 0, 0] a nroxCuadrantes
 Asignar [0, tamEntrada/4, tamEntrada/2, (3*tamEntrada)/4] a posxCuadrantes
 Asignar falso a debeOrdenar
 Asignar 0 a posInicioOrdenamiento
 // retorna arreglo de tamaño tamEntrada, con cada posición inicializada en falso
 Calcular datosB  mediante el subproceso: iniciarPosicionesEnFalso, pasando como valor: tamEntrada

 Asignar 0 a i
 Procesar para cada símbolo ingresado (i  < tamEntrada)
      Asignar datosS[i] en valorActual
      Si valorActual es mayor o igual a tamEntrada   // símbolo procesado excede el máximo permitido
           Asignar VALORSUPERIOR en estado
           Terminar Procesar
      Si datosB[valorActual] es verdadero   // ese símbolo ya había sido procesado
           Asignar VALORREPETIDO en estado
           Terminar Procesar
      Sino
           Asignar verdadero en datosB[valorActual]  // indica símbolo procesado por primer vez

      Calcular elCuadrante mediante el subproceso: obtenerCuadrante, pasando como valor: datosC[i]
      Incrementar 1 en nroxCuadrantes[elCuadrante-1]   // indica que hay un complejo más del cuadrante
      // Si se desea que el validador ordene. Solo se ejecuta en el demodulador, no en el modulador
      Si aplicarOrdenamiento es verdadero
           // se acaba de descubrir el primer complejo que no está en el cuadrante que le corresponde
           Si debeOrdenar es falso e i es diferente a posxCuadrante[elCuadrante-1]
                Asignar verdadero en debeOrdenar    // se debería ordenar
                Asignar i en posInicioOrdenamiento  // asigna posición a partir de la cual ordenar
           Sino
                Incrementar 1 en posxCuadrantes[elCuadrante-1]  // actualiza posición complejos procesado
      Incrementar 1 en i
 Repetir para el siguiente símbolo ingresado

 // Si no se detectó error de VALORSUPERIOR o VALORREPETIDO
 Si estado es igual a VALIDO
      // valida si hay igual cantidad de símbolos por cuadrante
      Si no son iguales: nroxCuadrantes[0], nroxCuadrantes[1], nroxCuadrantes[2], nroxCuadrantes[3]
           Asignar PUNTOSCUADRANTES en estado  // asigna símbolos por cuadrante no es igual
      // si es el demodulador y se determinó que deben ser ordenados
      Si aplicarOrdenamiento es verdadero y debeOrdenar es verdadero
           Ordenar datosS y datosC mediante el subproceso: ordenar, pasando como valores: datosS,
                     datosC y posInicioOrdenamiento  // ordenar datos símbolos y complejos

 Si estado es igual a VALORSUPERIOR  // Error detectado
      Abortar y disparar error ("… Hay símbolos con valor superior al tamaño de la entrada")
 Sino
      Si estado es igual a VALORREPETIDO   // Error detectado
           Abortar y disparar error ("… Hay símbolos repetidos en la entrada")
      Sino
           Si estado es igual a PUNTOSCUADRANTES   // Error detectado
                Abortar y disparar error ("… No hay igual cantidad de puntos en los cuadrantes")
           Sino
                Regresar estado  // retorna VALIDO
```

FIGURA 5. Diagrama de flujo del moduladorUAO_UV



2.3.2 Subproceso Mapear símbolos

Se encarga de realizar la conversión de cada símbolo binario, cuya longitud en bits depende del tamaño del símbolo, a su complejo correspondiente según la constelación que haya sido ingresada por el usuario.

En el actual bloque mapeador de símbolos en GNU Radio para QAM, se evita la búsqueda secuencial de los símbolos para recuperar su complejo correspondiente, ya que las constelaciones están ordenadas y se utiliza el símbolo a procesar como índice para recuperar su complejo. Como el presente modulador opera sobre constelaciones flexibles, con el fin de evitar la búsqueda secuencial y tener un método eficiente de mapeo, se propone el pseudocódigo de la Tabla 5, en el cual es necesario que se cree una estructura asociativa (para almacenar y recuperar elementos mediante claves de búsqueda) con los datos suministrados en los arreglos de símbolos y complejos y que sobre esta estructura, se realice el mapeo.

En la implementación del subproceso Mapear Simbolos se propuso una nueva versión del bloque chunks_to_symbols_bc para mapear los símbolos en sus complejos correspondientes [11], usando como estructura asociativa, la clase Map de C++, la cual se utiliza para almacenar y recuperar rápidamente a través de claves únicas (los símbolos) sus valores asociados (el complejo correspondiente) [12].

A diferencia de la búsqueda secuencial que es O(n), con la estructura asociativa Map, la complejidad teórica se reduce a O(1).

El bloque moduladorUAO_UV es jerárquico y se implementó en python. Para el análisis de complejidad de su seudocódigo (c_{modUAO_UV}), se debe tener en cuenta que k es el número de bits de la entrada Dm; y que depende de la complejidad del validador (c_{val}), obtener los símbolos (c_{obtS}) y mapear los símbolos (c_{mapS}), por ende:

$$C_{modUAO\_UV} = C_{val} + C_{obtS} + C_{mapS}$$

$$C_{modUAO\_UV} = O(n) + O(k) + O(n) \qquad (2)$$

$$C_{modUAO\_UV} = O(n)$$

Teniendo en cuenta que la validación se realiza sólo al inicio del proceso de modulación y que después sólo persiste la obtención de símbolos y el mapeo de símbolos, la complejidad operativa del modulador será O(n) porque es necesario guardar cada uno de los n datos de la constelación suministrada en la estructura asociativa.

2.4 PROCESO DEMODULADORUAO_UV

El proceso demoduladorUAO_UV recibe el flujo de datos modulados en formato complejo y según la constelación de entrada, obtiene los datos (símbolos) que originalmente fueron enviados.

En el diagrama de flujo mostrado en la Figura 6 se hace una validación (lado izquierdo), solicitando que se aplique ordenamiento, porque al estar los complejos ordenados por cuadrantes se pueden realizar procesos de mapeo de complejos sobre ¼ del número de complejos ingresados.

TABLA 3. Análisis del proceso moduladorUAO_UV

Proceso: moduladorUAO_UV					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Constelación Muestra	Constelación usada (Cu={S,C,K,..} Siendo: S= $s_0s_1...s_p$ (simbolos) C= $c_0c_1...c_{p*D}$ (complejos) K o bits por Símbolo (entero)) Datos a modular (Dm = $B_0B_1...B_{n-1}= b_0b_1... b_{n*8-1}$)	Símbolo[] Entero	Símbolos a mapear (Sm = $s_0s_1... s_{(n-1)*8/Bps}$) estado	Complejo[]	Complejos buscados (Cb)

Leer Cu y Dm

// Realiza validación sin intentar ordenamiento. Si hay falla en la validación aborta el modulador
Calcular estado mediante el subproceso: **validar**, pasando como valores: Cu.S, Cu.C y falso

Calcular Sm mediante el subproceso: **extraerSimbolos**, pasando como valores: Cu.K y Dm

Calcular Cb mediante el subproceso: **mapearSimbolos** pasando como valores: Sm, Cu.S y Cu.C

Regresar Cb
Terminar

TABLA 4. Análisis del subproceso ExtraerSimbolos

Subproceso: ExtraerSimbolos					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Entero Muestra	Bits por símbolo (K) Datos a modular (Dm)	Entero Bits[]	contador Bits extraídos (Be)	Simbolo[]	Símbolos a mapear (Sm)

Recibir K y Dm

Asignar 0 a contador
Procesar para cada agrupación de Bits por símbolos (contador < Dm.longitud*8/K)
Asignar Dm[contador*K…. contador*K+(K-1)] en Be

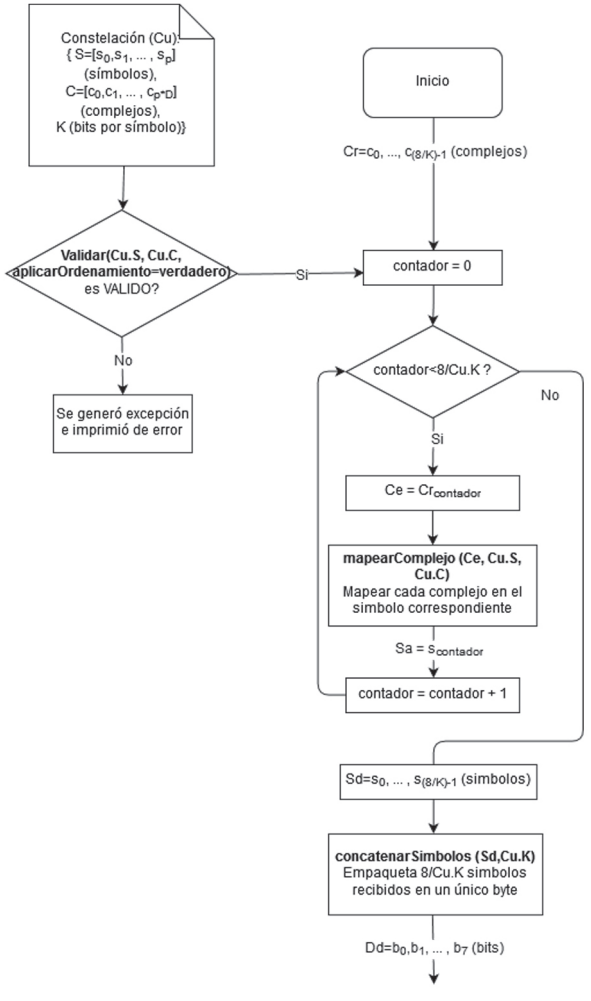
Calcular Sm[contador] mediante el subproceso: **completarSimbolo** pasando como valor: Be

Incrementar 1 en contador
Repetir para cada agrupación de Bits por símbolos
Regresar Sm
Terminar

TABLA 5. Análisis del subproceso MapearSimbolos

subProceso: MapearSimbolos					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Simbolo Simbolo[] Complejo[]	Símbolos a mapear (Sm) Símbolos constelación (Cu.S) Complejos de la constelacion (Cu.C)	Entero Estructura Asociativa	Posición buscada (Pb) Repositorio búsqueda (Rb)	Complejo	Complejo modulado (Cm)
Recibir Sm, Cu.S y Cu.C Calcular Rb mediante el subproceso: crearEstructuraAsociativa pasando como valores: Cu.S y Cu.C Calcular Cm mediante el subproceso: **buscarComplejo** pasando como valores: Rb y Sm Regresar Cm Terminar					

FIGURA 6. Diagrama de flujo del demoduladorUAO_UV



Como se indica en la Tabla 6, la entrada inicial al bloque demodulador es el mismo objeto constelación Cu que se le entregó al bloque modulador que contiene el arreglo de símbolos (Cu.S) y complejos (Cu.C), y es recibido por el **Validador**. Si la validación es exitosa (se retornó el estado VALIDO), el demodulador queda disponible para procesar cada complejo de entrada a demodular (Ce) y luego, los símbolos obtenidos tienen que ser de nuevo empaquetados hasta formar los bytes que son dados como respuesta.

2.4.1 Subproceso Mapear complejos

Se encarga de realizar la conversión de complejo a su símbolo correspondiente según la constelación que haya sido ingresada por el usuario.

En el actual bloque mapeador de complejos en GNU Radio para QAM, se recorre secuencialmente cada uno de los complejos, intentando encontrar aquel más próximo al complejo a demodular (Ce) y se retorna la posición en que este se encuentra. Como el presente demodulador opera sobre constelaciones no ordenadas, después de encontrar la posición del complejo más próximo, se retorna el símbolo almacenado en dicha posición. Con el fin de mejorar la búsqueda secuencial del complejo más próximo de la constelación, se realiza el ordenamiento en el subproceso Validar, para realizar una búsqueda secuencial sólo entre los N/4 complejos ubicados en el cuadrante al que pertenece Ce. Lo anterior se puede verificar en el pseudocódigo de la Tabla 7, en el cual se computan las posiciones iniciales (Pi) y finales (Pf) del cuadrante, para realizar el recorrido del fragmento de instrucciones Procesar-Repetir.

En la implementación correspondiente, se propone un bloque *constellation_receiver_cb* con las mismas entradas y salida del bloque existente en GNU Radio, pero mejorado, el cual invoca a una versión eficiente de la función C++ denominada *get_closest_point* en la que finalmente se implementa el pseudocódigo mostrado. Esta versión mejorada, al igual que la versión existente en GNU Radio, sigue siendo Θ(n) pero con una constante menor de la función que define su tiempo de ejecución, por realizarse la búsqueda sólo en el cuadrante respectivo.

2.4.2 subproceso Concatenar símbolos

Operación opuesta al proceso de extraer símbolos, ya que consiste en agrupar en una muestra de 8 bits los símbolos binarios obtenidos de mapear complejos.

Los símbolos a empaquetar (Se) obtenidos del mapeo de los números complejos del flujo de entrada, representados en muestras de K bits, se agregan en una sola muestra (Dd), como se observa en el subproceso **concatenarSimbolos** de la Tabla 8.

TABLA 6. Análisis del proceso demoduladorUAO_UV

Proceso: demoduladorUAO_UV					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Constelación Complejo	Constelación usada (Cu={S,C,K,..} Siendo: S= $s_0 s_1 ... s_p$ (simbolos) C= $c_0 c_1 ... c_{p*D}$ (complejos) K o bits por Símbolo (entero)) Complejo de entrada a demodular (Ce = a +bj)	Símbolo Símbolo[]	Símbolo demodulado actual (Sa) Símbolos demodulados (Sd)	Símbolo	Símbolos demodulados empaquetados (S)

```
Leer Cu y Ce

// Realiza validación intentando ordenamiento. Si hay falla en la validación aborta el modulador
Calcular estado mediante el subproceso: validar, pasando como valores: Cu.S, Cu.C y true

Asignar 0 a contador
Procesar para cada complejo (contador  < 8/Cu.K)
    Calcular Sa mediante el subproceso: mapearComplejo, pasando como valores: Ce, Cu.S
            y Cu.C
    Asignar Sa en Sd[contador]
    Incrementar 1 en contador
Repetir para cada complejo

Calcular S mediante el subproceso: concatenarSimbolos pasando como valores: Sd y Cu.K

Regresar S
Terminar
```

Para realizar la concatenación de símbolos se utilizó el bloque unpacked_to_packed_bb existente en el módulo blocks suministrado por GnuRadio [13] e indicó que la alineación correspondiente era la misma usada en el modulador, es decir, de izquierda a derecha (gr.GR_MSB_FIRST).

El bloque demoduladorUAO_UV es jerárquico y se implementó en Python. Para el análisis de complejidad de su seudocódigo (c_{demUAO_UV}) se debe tener en cuenta que k es el número de complejos presentes en el flujo a procesar Cr; y que depende de la complejidad del validador (c_{val}), mapeo de complejos (c_{mapC}) y la

concatenación de los símbolos obtenidos (c_{concat}), por ende:

$$C_{demUAO\_UV} = C_{val} + k * C_{mapC} + C_{concat}$$

$$C_{demUAO\_UV} = O(n) + k * O(n) + O(1) \quad (3)$$

$$C_{demUAO\_UV} = O(n)$$

Teniendo en cuenta que la validación se realiza sólo al inicio del proceso de demodulación y que después sólo persiste el mapeo de complejos y la concatenación de símbolos, la complejidad operativa del demodulador será O(n) aportado por el mapeo de complejos, que es ejecutado también k veces, al igual que en el bloque demodulador de Gnu Radio. De la Tabla 9 que resume como opera el mapeo de complejos e indica la función del tiempo de ejecución, para el bloque existente en Gnu Radio y el bloque propuesto, resulta evidente que el desempeño del bloque de mapeo de complejos propuesto es mejor y por ende el bloque demoduladorUAO_UV tiene mejor desempeño que el generic_demod existente en Gnu Radio.

TABLA 7. Análisis del subproceso mapearComplejos

subProceso: mapearComplejos					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Complejo	Complejo de entrada (Ce)	Entero	cuadrante	Simbolo	Símbolo demodulado actual (Sa)
Simbolo[]	Símbolos constelación (Cu.S)	Entero	Posición inicial del cuadrante a buscar (Pi)		
Complejo[]	Complejos de la constelacion (Cu.C)	Entero	Posición final del cuadrante a buscar (Pf)		
		Real	Distancia al complejo más cercano (Dm)		
		Real	Distancia calculada (D)		

```
Recibir Ce, Cu.S y Cu.C

  Calcular cuadrante mediante el subproceso: obtenerCuadrante pasando como valor: Ce

  // define las posiciones en la que debe buscar
  Según (cuadrante)
     Es 1 : Asignar 0 en Pi
            Asignar longitud(Cu.S)/4 en Pf
     Es 2 : Asignar longitud(Cu.S)/4 en Pi
            Asignar longitud(Cu.S)/2 en Pf
     Es 3 : Asignar longitud(Cu.S)/2 en Pi
            Asignar longitud(Cu.S)*3/4 en Pf
     Es 4 : Asignar longitud(Cu.S)*3/4 en Pi
            Asignar longitud(Cu.S) en Pf

  // Calcula la distancia del complejo de entrada al primero del cuadrante a buscar
  Calcular Dm mediante el subproceso: obtenerDistancia pasando como valor: Pi, Ce, Cu.C
  Asignar Cu.S[Pi] en Sa

  // procesa los complejos restantes del cuadrante para encontrar el símbolo del complejo más próximo
  Asignar Pi+1 en contador
  Procesar para cada valor de contador (contador  < Pf)
      Calcular D mediante el subproceso: obtenerDistancia, pasando como valores: contador, Ce y Cu.C
      Si D es menor a Dm
          Asignar D en Dm
          Asignar Cu.S[contador] en Sa
      Incrementar 1 en contador
  Repetir para cada valor de contador
  Regresar Sa
Terminar
```

TABLA 8. Análisis del subproceso concatenar Símbolos

subProceso: concatenarSimbolos					
Entradas		**Temporales**		**Salidas**	
Tipo	**Identificador**	**Tipo**	**Identificador**	**Tipo**	**Identificador**
Simbolo[] Entero	Símbolos a empaquetar (Se) Bits por símbolos de la constelación (Cu.K)			Simbolo	Dato demodulado (Dd)

```
Recibir Se y Cu.K

Asignar 0 a contador
Procesar para cada símbolo (contador  < 8/Cu.K)
    Pegar los Cu.K bits de Se[contador] en S ( desde la posición contador*Cu.K )
    Incrementar 1 en contador
Repetir para cada símbolo
Regresar S
Terminar
```

TABLA 9. Comparación bloques mapeadores de complejos

Mapeador Gnu Radio	Mapeador Propuesto
Opera el complejo recibido contra todos los complejos de la constelación	Opera el complejo recibido solo contra complejos de la constelación que se encuentran en su cuadrante
Función del tiempo de ejecución F(n) = n	Función del tiempo de ejecución F(n)= (1/4)n

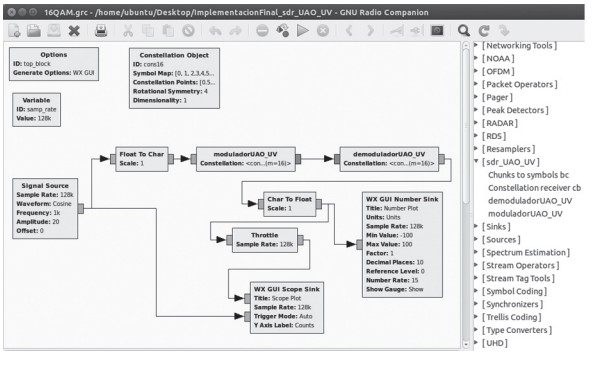
3. RESULTADOS OBTENIDOS

Los pseudocódigos de los subprocesos mapear símbolos (mejora del bloque ***chunks_to_symbols_bc***) y mapear complejos (mejora del bloque ***constellation_receiver_cb***) fueron implementados en C++, mientras que el subproceso validar y los procesos modular (nuevo bloque **moduladorUAO_UV**) y demodular (nuevo bloque **demoduladorUAO_UV**) fueron implementados en Python, haciendo uso de la herramienta gr_modtool. También se modificaron los archivos XML que permiten integrar los bloques nuevos a la herramienta GRC como muestra el ejemplo para el bloque ***modulador UAO_UV*** de la figura 7; esto permitirá usarlos no solamente para este proyecto en particular sino también en otras aplicaciones si se requieren. Como se observa en la línea 9, se requiere el ingreso de un parámetro constelación, en la línea 13 se define el flujo de entrada que corresponde a los bytes que representan los símbolos y en la línea 17 se define el flujo de salida que corresponde a los complejos que representan la señal modulada.

FIGURA 7. Archivo XML del bloque moduladorUAO_UV

```
1  <?xml version="1.0"?>
2  <block>
3    <name>moduladorUAO_UV</name>
4    <key>sdr_UAO_UV_moduladorUAO_UV</key>
5    <category>sdr_UAO_UV</category>
6    <import>import sdr_UAO_UV</import>
7    <make>sdr_UAO_UV.moduladorUAO_UV($constellation)</make>
8    <param>
9      <name>Constellation</name>
10     <key>constellation</key>
11     <type>raw</type>
12   </param>
13   <sink>
14     <name>in</name>
15     <type>byte</type>
16   </sink>
17   <source>
18     <name>out</name>
19     <type>complex</type>
20   </source>
21 </block>
```

FIGURA 8. Uso del sdr-UAO_UV para un 16QAM

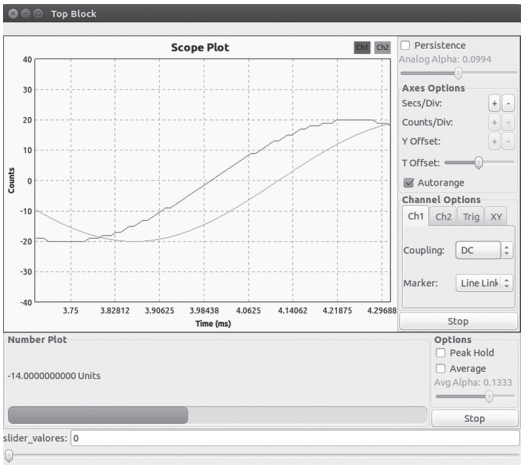


La Figura 8 muestra el montaje realizado para comprobar el correcto funcionamiento de la implementación e integración de los bloques moduladorUAO_UV y demoduladorUAO_UV en la herramienta GRC, en el cual se introduce una señal sinusoidal de frecuencia de 1KHz muestreada a una tasa de 128K muestras por segundo

para que sea modulada y luego demodulada usando 16QAM.

Para verificar que la operación de los bloques es correcta, se utiliza un bloque de despliegue numérico que muestra el valor actual demodulado y un bloque de despliegue gráfico similar a un osciloscopio que muestra la correspondencia entre la señal de entrada (Canal 2) y de salida (Canal 1), como muestra la Figura 9.

FIGURA 9. Verificación funcionamiento sdr-UAO_UV en 16QAM



Para comprobar que los bloques implementados funcionan con modulación QAM de orden variable, se utilizaron dos constelaciones distintas (16 QAM y 32QAM) como entrada en el bloque *Constellation Object* para modular la señal de entrada como se observa en la Figuras 10 y 11.

FIGURA 10. Constelación para 16QAM

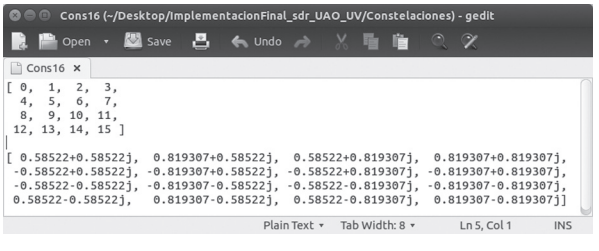
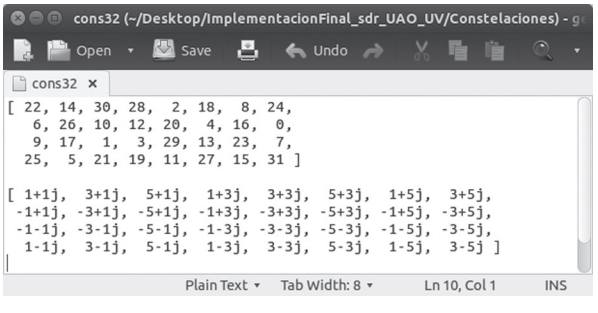


FIGURA 11. Constelación para 32QAM



4. CONCLUSIONES

El módulo sdr-UAO_UV proporcionado, compuesto por cuatro bloques funcionales, permite modular y demodular constelaciones QAM hasta de 128 símbolos con una mayor eficiencia que los bloques moduladores y demoduladores actuales, debido al uso de estructuras dinámicas asociativas que permiten ejecutar búsquedas no secuenciales en la etapa de modulación; y que en la etapa de demodulación evita la búsqueda secuencial del complejo recibido en toda la constelación (cuatro cuadrantes) y solo la efectúa en el cuadrante respectivo.

El subproceso validar proporcionado, permite verificar en los datos de la constelación suministrada por el usuario que: la cantidad de datos corresponda con su tamaño, los símbolos sean diferentes y la cantidad de complejos por cuadrante sea la misma; ya que son errores que el usuario puede cometer al ingresar la constelación. Adicionalmente, permite el reordenamiento de la constelación en caso de ser necesario.

GNU Radio es una alternativa interesante para la investigación de nuevos protocolos y tecnologías de transmisión para dispositivos inalámbricos, ya que convierte problemas hardware en problemas software, disminuye notablemente el tiempo de desarrollo e implementación y reduce costos al tratarse de una plataforma de código abierto.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Giannini, V., J. Craninckx & Baschirotto, A. (2008). *Baseband analog circuits for software defined radio.* Springer.

[2] Santana, J. (2012). *GNU-Radio en la enseñanza de comunicaciones inalámbricas.* Tesis de trabajo de grado no publicada. Universidad de Concepción, Concepción, Chile.

[3] Free Software Foundation. (2012). *Gnuradio. digital: Signal Processing Blocks. Generic_mod.* Recuperado (2015, enero 4) de http://gnuradio.org/doc/sphinx/digital/blocks.html#gnuradio.digital.generic_mod

[4] Free Software Foundation. (2012). Gnuradio. digital: Signal Processing Blocks. Generic_demod. Recuperado (2015, enero 4) de http://gnuradio.org/doc/sphinx/digital/blocks.html#gnuradio.digital.generic_demod

[5] Gnu Radio. (2013). *Welcome to GNU Radio.* Recuperado (2015, enero 4) de http://gnuradio.org/redmine/projects/gnuradio/wiki

[6] Gnu Radio. (2013). *What is GNU Radio and why do I want it?.* Recuperado (2015, enero 4) de http://gnuradio.org/redmine/projects/gnuradio/wiki/WhatIsGR

[7] Gnu Radio. (2013). *Out-of-tree modules*. 2015. Recuperado (2015, enero 4) de http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules

[8] Casey, D.,Tagliarini G.(2009). *Prototyping with GNU Radio and the USRP – Where to Begin* IEEE, Southeastcon, pp. 50-54.

[9] Gnu Radio. (2013). *GNU Radio Companion.* Recuperado (2015, enero 4) de http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

[10] Gnu Radio. (2015). *Constellation Class Reference.* Recuperado (2015, enero 4) de http://gnuradio.org/doc/doxygen/classgr_1_1digital_1_1constellation.html

[11] Gnu Radio. (2014). *chunks_to_symbols_bc File Reference*. Recuperado (2015, enero 4) de http://gnuradio.org/doc/doxygen-3.7.3/chunks__to__symbols__bc_8h.html

[12] Cplusplus.com (2015). *std::map*. Recuperado (2015, enero 4) de http://www.cplusplus.com/reference/map/map/

[13] Gnu Radio. (2014). *gr::blocks::unpacked_to_packed_bb Class Reference*. Recuperado (2015, enero 4) de http://gnuradio.org/doc/doxygen-3.7.3/classgr_1_1blocks_1_1unpacked__to__packed__bb.html