

V Conclusiones finales y propuestas de mejora

Concluidas las fases de experimentación y elaboración de la memoria, pasemos a exponer las **conclusiones** más importantes de este proyecto a modo de resumen final.

Para terminar, escribiremos un apartado donde se propondrán una serie de **líneas futuras de trabajo**, así como algunas posibles **propuestas de mejora**.

V.1.- CONCLUSIONES FINALES

Recordemos que nuestro objetivo es crear un **modulador/demodulador M-APK** para transmitir datos. La **fuentes** de estos datos son **ficheros** presentes en el disco duro de un PC, o bien en alguna unidad extraíble como pudieran ser un diskette de 3_{1/2}, ZIPs o CDROM.

Deberemos transmitirlos hacia otro PC cercano (o bien hacia el mismo PC, si la tarjeta es full duplex), teniendo como único medio de transmisión un **cable de audio** (mono o estéreo), terminado en dos conectores macho. Estos conectores se insertarán en las clavijas LINE-IN (PC Receptor) y LINE-OUT (PC Transmisor) de las tarjetas de sonido.

Para diseñar nuestro módem M-APK, necesitamos obtener nuestras variables de diseño del sistema, para ello empezamos haciendo un **estudio del canal**. Dicho estudio del canal nos proporciona una información muy valiosa, a saber:

- **Característica BP** (paso de banda) del canal.
- **Ancho de banda** del canal comprendido entre 20 hz. y 20 khz. Como consecuencia inmediata, situamos nuestra frecuencia portadora en 11025 hz.
- El canal introduce una **distorsión en amplitud**, por lo que habrá que ecualizar dicha respuesta impulsiva.
- El análisis del ruido nos indica que el nivel de éste no es importante como para considerarlo a la hora de diseñar el ecualizador de canal. Así, descarto usar filtros Wiener y realizamos un filtro igualador de cero forzado (**Zero-Forcing Equalizer**).
- Existen **problemas de sincronismo**, derivados del desajuste de frecuencias entre los relojes Transmisor y Receptor. Este es el problema más importante y el que hay que evitar o paliar a toda costa.

Ya en la fase de Diseño de nuestro sistema, y tras una breve introducción a la modulación digital, y a pesar de tener elegida de antemano la técnica de modulación a emplear, o sea, la Modulación Digital M-QAM, se creyó conveniente establecer una **justificación comparativa** de nuestra técnica con otros tipos de técnicas de modulación alternativas perfectamente eficaces. Todas estas comparaciones se discutieron desde varios puntos de vista:

- La **probabilidad de error de símbolo**, P_e .
- El **ancho de banda** de la señal.
- **Energía media** y **energía pico** del sistema.
- **Tasa de bits** del sistema, R_b .
- **Nº de puntos del mensaje** de la constelación, M .

Pero antes se realizó una **parametrización** del sistema, para conocer la relaciones existentes entre las distintas variables de diseño, y empezamos a **dimensionar** un poco el sistema, con valores típicos ($N=10$, $D=44100$ hz.), para hacernos una idea del rango numérico en el que se moverían cada una de estas variables. De aquí sacamos en claro algunos conceptos:

- El **ancho de banda** de nuestra señal depende de nuestra tasa de símbolos, D .
- **Para aumentar la velocidad de transmisión**, tenemos varias opciones:
 - **Aumentamos M** , de manera que la tasa de bits, R_b aumenta sin aumentar D . Pero esto aumentar la probabilidad de error del sistema, P_e . **Para mantener P_e constante**, podemos:
 - **Aumentar la energía de cada símbolo**, pero nuestra potencia está limitada al rango de nuestro canal y tenemos que evitar dicha saturación.
 - Aumentar M usando **otra técnica de modulación** que mantenga los símbolos repartidos en la constelación.
 - **Aumentar la frecuencia de muestreo, F_s** , teniendo en cuenta que D también aumenta y nuestro canal tiene una frecuencia de corte considerablemente baja.
- Establecimos un rango dinámico **el número de muestras por símbolo, N** . Al final obtuvimos un valor práctico para el caso más crítico (factor de Roll-off unidad y $M = 256$ símbolos) de $N=9$ muestras por símbolo.
- Además hallamos los **extremos teóricos de F_s** , para valores típicos de N :
 - $N=10 \rightarrow F_s|_{min} = 27563$ hz.
 - $N=10 \rightarrow F_s|_{max} = 69750$ hz.
- Decidimos 7 posibles valores de **M : 4,8,16,32,64,128 y 256 símbolos**.
- Por último, para dichos valores típicos, $N=10$ y $F_s=44100$ hz, establecimos el **rango de velocidades de nuestro sistema**, en función del rango de M :
 - $M = 4 \rightarrow R_b = 8820$ bits/sg.
 - $M = 256 \rightarrow R_b = 35280$ bits/sg.

Como desenlace de las comparaciones entre la técnica de modulación M-APK, llegamos a las siguientes conclusiones:

- La técnica de modulación **M -QAM** es la que puede soportar, en las mismas condiciones de contorno, un **mayor número** de símbolos en el sistema, **M** , concretamente $M = 256$, debido entre otras cosas a que su constelación tiene

dos grados de libertad (amplitud y fase), y permite repartir mejor los puntos de mensaje en el plano.

- Esto sitúa a nuestra modulación como la **más versátil** en cuanto a número de **constelaciones** diferentes, tanto por la distribución de sus símbolos como por su número de constelaciones M-arias viables.
- También, y junto con la técnica M-PSK, es la que tiene **mejor eficiencia espectral, ρ** .
- Por el contrario, al estar contenida la información tanto en la fase como en la amplitud de la portadora, también es la **más vulnerable** a las **no linealidades del canal**, por lo que el Ecuador va a ser pieza clave en nuestro sistema, para eliminar esas no linealidades.

Tras dicha serie de comparaciones, dividimos nuestro sistema en dos bloques, **Bloque Transmisor** y **Bloque Receptor**, mostramos un dibujo genérico de cada uno de ellos, e hicimos una ruta muy detallada de cada tramo perteneciente en cada bloque, planteando los problemas y soluciones que les fuimos dando, incluyendo además dibujos aclarativos y aplicaciones prácticas cada vez que teníamos oportunidad.

Las observaciones más importantes de esta excursión descriptiva por nuestro sistema iban a ser las siguientes:

- Respecto a las **constelaciones**:
 - Para asegurar una **constelación variable**, tendremos **7 tablas** de constelación distintas, una para cada valor de M.
 - La tasa binaria más alta se dará para $M = 256$ símbolos. Sin embargo será el sistema con más tendencia al error en cuanto a la detección de los símbolos. Sin embargo, para $M = 4$ símbolos ocurre todo lo contrario.
 - Las constelaciones con un número de $M = 4, 16, 64$ ó 256 símbolos serán de **forma cuadrada**.
 - Las constelaciones con un número de símbolos $M = 32$ ó 128 serán constelaciones **“cuasi-cuadradas”**.
 - La **energía pico**, o energía del símbolo más alejado del sistema es igual a la **unidad**, debido al rango digital de la tarjeta de sonido, ubicado en el intervalo $[-1,1]$.
 - El código para codificar los puntos de mensaje de la constelación es el **código Gray**, para disminuir el número bits erróneos en un símbolo, ante un fallo en la detección de dicho símbolo.
 - Usamos tablas para aumentar la configurabilidad de las **constelaciones** y poder **dotarla de otra fisonomía**. Esto lo conseguimos modificando las coordenadas de los símbolos en dichas las tablas de mapeo.
 - Además, podríamos **cambiar** fácilmente el **tamaño** de las **constelaciones**, y la distancia, bien cambiando el valor de la mínima **distancia intersimbólica**, L, o bien escalando su valor por una constante.
- En relación al **Inicio y Fin de Trama**:

- Para devolver el rango dinámico de las muestras recibidas, al rango original donde se movían en transmisión, $[-1, 1]$, nos inventamos unos **símbolos ficticios** de Inicio de trama, **de valor 0.4**, que nos servirán para realizar el **reajuste de niveles**.
- Además los primeros 4 símbolos del Inicio de Trama, y los 4 símbolos de Fin de trama absorberán los **efectos del transitorio**, debido al retraso de grupo que introduce el Filtro Transmisor.

- En relación a los **Filtros Transmisor y Receptor**:
 - Para evitar la **Interferencia intersimbólica o ISI**, hacemos uso de la familia de **pulsos raíz de coseno alzado**.
 - El Filtro Transmisor ejerce la labor de **pulso conformador**, y el Filtro Receptor hace a su vez de **filtro matcheado**, y de **filtro Paso de Baja** para recuperar la envolvente de la señal una vez demodulada.
 - Ambos filtros son pulsos raíz de coseno alzado ,y en cascada (siempre suponiendo un canal ideal en conjunto con el ecualizador), forman un **pulso en coseno alzado**, que cumple las condiciones de Nyquist (en tiempo y en frecuencia) para evitar la ISI.
 - El factor de **Roll-off** será **modificable**, siendo manipulable de este modo la anchura de nuestra señal.
 - Ambos filtros, al convolucionar con la señal de entrada, generan unas **colas** denominadas **retraso de grupo**, y que hay que tener en cuenta en nuestro diseño. Dicho retraso de grupo será de tres tiempos de símbolos, esto es, de $3 \cdot N$ muestras.

- En lo referente a la **señal Piloto**:
 - Tiene dos misiones igualmente importantes:
 - Establecer el **sincronismo de trama**.
 - Actualizar la respuesta impulsiva del canal antes de cada trama, para conseguir así un **ecualizador dinámico**.
 - Intentamos transmitir las tramas de una en una y nos resultó imposible debido a que no podíamos controlar los **tiempos de CPU**.
 - Así, con el ecualizador dinámico simulamos que transmitimos las tramas de una en una, por un canal distinto cada vez, y ecualizamos cada una de ellas de manera óptima.

- Respecto los **Convertidores Analógico-Digital y Digital-Analógico**:
 - Teniendo en cuenta el rango dinámico de F_s para valores típicos de N , y que las frecuencias de muestreo estándares para los dispositivos de audio para PC son **8000, 11025, 22050 y 44100 hz.**, llegamos a la conclusión de que **$F_s = 44100$ hz.** es la **máxima** y la **única frecuencia**

viable en nuestro sistema de transmisión, ya que inferiormente está acotada por el *efecto "aliasing"*.

- Nuestro proyecto utilizará siempre un número de bits de resolución $NB = 16$, ya que en el formato PCM que usaremos para transmitir, NB dependerá del tipo de datos que se transmitan. De todos modos dejamos explicado cómo transmitir a $NB=8$ muestras/símbolo, esto es, haciendo uso del comando 'uint', para cambiar de formato 'double' a 'uint' los datos a transmitir.
- El *retraso del ecualizador* habrá que tenerlo en cuenta en recepción.
- Respecto a la *modulación y demodulación*, decir simplemente que ambas han de hacerse, como es lógico a la misma frecuencia de portadora, que será fija e igual a **11025 hz.**, por el hecho de usar un submúltiplo de la frecuencia de muestreo, F_s , que será de 44100 hz.
- En relación al *Ecualizador*:
 - Ante la relación señal a ruido (*SNR*) tan *elevada* que presenta el sistema, *descartamos* el uso de *filtros Wiener*.
 - Al tratarse de un *sistema LTI* (aunque la falta de sincronismo lo camuflara como sistema LTV), *rechazamos* la necesidad de usar *filtros adaptativos*.
 - Usaremos un Igualador de cero forzado (*zero-forcing equalizer*), que en conjunto con el canal se comportarán como un sistema ideal. El igualador básicamente desplaza las muestras a su posición original, para paliar los efectos del desplazamiento de las muestras a causa de la falta de sincronismo.
 - Construiremos *Ecualizadores óptimos* para cada trama, gracias a la señal piloto que precede a cada una de ellas.
- El *Detector* que usaremos será el de *Máxima Verosimilitud*.
- El *Destino Digital* será un *fichero* que crearemos y guardaremos en el Disco duro o en un diskette. Dicho fichero, caso de no producirse errores, ha de ser idéntico al fichero procedente de la Fuente Digital.
- Respecto a las *anomalías del sistema*, enumeraremos las más frecuentes e influyentes:
 - La *distorsión en amplitud* del canal: la solucionamos con la ayuda del Ecualizador.
 - La *interferencia intersimbólica (ISI)*: Para remediarla hacemos uso de los filtros raíz de coseno alzado principalmente. El ecualizador también ayuda.
 - La *ausencia de total sincronismo*: Es el problema más grave.

- Se solucionó **acortando la longitud de las tramas**, para conseguir que el error de desplazamiento no superara el **1%**.
 - De este modo solo podemos mandar **882 muestras** totales por trama.
 - Así, para un caso crítico de probabilidad de error (M=256) y un valor típico de N=10 muestras / símbolo, establecemos el valor del número de bytes de datos por cada trama, **BDT**, que será de **60 a 70 bytes** aproximadamente.
- En relación a la **estructuración en tramas** de nuestra información:
 - Ésta surge como consecuencia de dos razones fundamentales:
 - **Facilitar la computación** de la CPU.
 - Para **controlar los trozos de información** en que se divide el fichero origen, como solución a la ausencia de sincronismo.
 - El formato de trama en el que nos inspiramos es el denominado **Ethernet**, es decir, el **IEEE 802.3**, en el que realizamos los siguientes cambios:
 - Eliminamos los campos Inicio, Preámbulo, Relleno, Dirección Origen y Dirección Destino, porque no los necesitábamos.
 - El CRC-32 lo cambiamos por el **CRC-16** por los siguientes motivos:
 - CRC-16 es ya bastante un código bastante potente.
 - El número de bytes del campo Datos es muy bajo (60 a 70 bytes < 1500 bytes permitidos en campo DATOS).
 - El CRC-16 introduce menor número de cálculos.
 - Así, cada una de las **tramas** adoptará el siguiente **aspecto**:



Fig. 119.- Campos de la trama

Respecto a la **compilación**, usamos una herramienta de Matlab, denominada **Compiler 2.1**, que nos ahorrará toda la labor de programación en C y de estudio de las APIs de Windows. Lo más importante a destacar en esta fase son los siguientes puntos:

- Para invocar al compilador utilizamos el **comando mcc**.

- Para crear los ejecutables ‘C stand alone’ para los Bloques Transmisor y Receptor, empleamos la **macro -m**, que equivale a todas las opciones necesarias para el comando mcc.
- Para la **compilación** era necesario:
 - El entorno de Matlab.
 - Compiladores de Matlab y de C.
 - La librería MATLAB C/C++ MATH library.
- Por otro lado, para la **portabilidad del código**, fue necesario:
 - Los ficheros ‘z_main.exe’ de ambos bloque Transmisor y Receptor.
 - Los archivos ‘playsnd.dll’ y ‘rcsnd.dll’ para la transmisión y recepción de las muestras a través de la tarjeta de sonido.
 - Todas las librerías dinámicas MATLAB Math run-time, contenidas en el archivo ‘mglinstaller.exe’.
- Los **problemas** que tuvimos en la **compilación** se debieron principalmente a las **limitaciones del compilador**, que no traducía correctamente algunas funciones, o bien no las admitía:
 - Para evitar usar la función ‘eval.m’, cambiamos la función ‘z_rcosflt.m’ de Matlab por nuestra función ‘z_filtro.m’.
 - Sustituimos la función ‘pause.m’ por la función ‘input.m’.
 - Tuvimos que transmitir silencio antes de la señal para evitar colisiones en el bus de datos.
 - Otros contratiempos sin importancia fueron que no se recogía la función ‘clc.m’ de Matlab, ni las tildes en las cadenas de caracteres.

En lo referente al capítulo de los **entornos de funcionamiento** es interesante comentar dos cosas únicamente:

- Reiterar que se necesitan **dos sesiones** para ejecutar nuestro proyecto, una para el **Bloque transmisor**, y otra para el **bloque Receptor**.
- El entorno usado en Windows es la **consola de MS-DOS**.

Tras la observación empírica del proyecto culminado en distintos PC’s, llegamos a la conclusión de que es necesario disponer de **PC’s relativamente potentes**, para que pueda soportar de una manera eficiente todo el procesamiento de la señal en ambos Bloques Transmisor y Receptor. Esto es así porque **nuestro módem** es eminentemente un **producto Software**, y no disponemos de un dispositivo Hardware que nos facilite la alguna tarea. Por este motivo, hay que tener paciencia en PC’s con micros bastante lentos, o poca memoria RAM, donde los Bloques de cálculo del CRC-16, detección de trama errónea y extracción del igualador óptimo se hacen notar bastante, porque ralentizan mucho la velocidad de proceso de cada trama. Esto supone un gran contraste frente a las **tarjetas de Ethernet**, por ejemplo, donde todos estos bloques se implementan **vía Hardware**, de la misma forma que en un módem para Internet.

Se trata de un *proyecto* que, tanto por sus conocidas *limitaciones de ancho de banda*, como por tratarse de un *sistema plesiócrono*, está enfocado a *ficheros de poca extensión* (sin contar además con las *limitaciones en velocidad de procesamiento* a las que aludimos en el párrafo anterior). Ya hemos apuntado que la máxima velocidad de transmisión la podemos alcanzar sin errores, para $N=10$ muestras y un número de símbolos $M=256$, es de $R_b= 35280$ bits/sg., ya que para $N=9$ aparecen algunas tramas erróneas esporádicamente.

V.2.- LÍNEAS DE MEJORA

La *propuesta de mejora más importante* que planteamos en esta sección está relacionada con la *rapidez del sistema*. Se trata de aprovechar de transmitir las tramas una a una. Con ello solapamos los tiempos de procesamiento de trama en transmisión y en recepción, incrementando notablemente la velocidad de ejecución de la transmisión, aspecto bastante importante en un sistema cuyo escaso ancho de banda apenas permite una tasa de unos cuantos Kbaudios.

Como ya comentamos durante la fase de Diseño, *intentamos sincronizar ambos Bloques Transmisor y Receptor*. Pero fue un *rotundo fracaso*, por varias razones:

- Ambos bloques estaban en *sesiones de Matlab diferentes*, y teníamos que jugar con los tiempos de CPU de sistemas para su sincronización, utilizando para ello funciones de Matlab que controlaran dichos tiempos.
- Además, el procesamiento en la cadena receptora era mucho más lento que en la transmisora, puesto que teníamos que incluir segundos de silencio en cada trama recibida para evitar la pérdida de muestras, y teníamos que controlar bien los tiempos de transmisión / recepción, para que no se perdieran muestras.
- Aún así, no contábamos con la *labor de multiprocesamiento de la CPU*, que no atendía instantáneamente nuestras peticiones de transmisión y recepción de las tramas, por lo que algunas se perdían.
- Así, para evitar pérdidas de muestras y transitorios indeseados, los *tiempos de espera* entre los procesamientos y las transmisiones se nos antojaban *bastante grandes*, lo cual provocó que esta técnica de transmisión en tiempo real fuera mucho más lenta que la técnica que finalmente usamos.

De todos modos dejamos esta opción en el aire, por si alguien se anima a *programar los drivers de sonido en tiempo real*, esto es, a permitir el acceso a los distintos buffers de la tarjeta de sonido receptora de modo que no hubiera que esperar a que terminara la transmisión del fichero para comenzar a procesarlo en recepción.

Otra forma de mejorar el proyecto para que éste gane en rapidez es implementar un *sistema de detección y corrección de errores*. De esta forma, podríamos aumentar un poco más la longitud de las tramas, aun a riesgo de cometer algún error en alguna que otra trama, ya que ésta sería corregida en recepción, gracias a dicho campo.

Una tercera propuesta podría ir dirigida a *mejorar la interfaz gráfica* de la aplicación, esto es, aprovechar las herramientas de Matlab de interacción con el programa *Visual Studio C++ 6.0*, para poder dotar a nuestra aplicación, por ejemplo, de:

- Ventanas de diálogo: para los Bloques Transmisor y Receptor.
- Etiquetas de texto: para enumerar las distintas opciones de diseño.
- Cuadros de texto: para escribir el nombre de los ficheros origen y destino.
- Grupos de texto: para agrupar las distintas opciones en un menú vistoso.
- Botones de opción: para mostrar los distintos grupos de opciones para cada variable de diseño.

- Barra de progreso: para mostrar el porcentaje de tramas procesadas.

Aunque no sea una propuesta de mejora en el estricto sentido de la ingeniería, sin embargo supondría una interfaz más agradable para el usuario.

Por último nos gustaría indicar una vía de **ampliación de nuestro proyecto**, integrando nuestra aplicación dentro de un **sistema de funcionamiento en red**. Es decir, varios PC's interconectados entre sí, en cualquier topología de red (en estrella, en anillo, interconexión total...), **unidos vía cable de audio**, y usando nuestro propio proyecto como base de la transmisión de los datos entre PC's. La ampliación de nuestro proyecto de comunicación punto a punto hacia una red de interconexión vía audio se reflejaría, entre otros aspectos, en los siguientes puntos:

- Tendríamos que implementar algún **protocolo de red** (bien orientado a conexión o bien servicio de Datagrama) para ,por ejemplo, evitar colisiones, encaminar las tramas, o para adquirir el testigo (caso de que se usara ese sistema), etc...
- Además, también sería necesario identificar cada uno de los PC's de alguna manera.
- Nos resultaría vital la recuperación de los **campos Dirección de Origen y Dirección de Destino** de la Trama Ethernet, desechados en un principio por su inutilidad, y que en este nuevo proyecto tendrían un papel decisivo para el enrutamiento de los datos. Incluso podríamos codificar estos campos para una transmisión en multidifusión.
- Podríamos además emplear alguna **codificación de fuente**, para compactar y encriptar los datos, lo cual serviría como ejercicio práctico muy instructivos para afianzar conocimientos teóricos.

Aquí concluye la exposición del proyecto. A quien pudiera interesarle continuar con lo desarrollado hasta aquí y quisiera consultar algún tipo de dudas, simplemente comentar que estaría encantado en colaborar en ese nuevo proyecto.

José Miguel Moreno Pérez
Dirección de correo electrónico: jmmoreno@linuxmail.org

