UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

CARRERA DE ELECTRÓNICA Y TELECOMUNICACIONES

TITULO DEL TRABAJO DE INVESTIGACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO PARA LA CANCELACIÓN DE RUIDO CON REDES NEURONALES UTILIZANDO DSP

AUTOR(ES):

LUIS MIGUEL SAMANIEGO CAMPOVERDE
VERÓNICA ELIZABETH SILVA GUADALUPE

AÑO

2016

Los miembros de Tribunal de Graduación del proyecto de investigación con el título: **DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO PARA LA CANCELACIÓN DE RUIDO CON REDES NEURONALES UTILIZANDO DSP** presentado por: **Luis Miguel Samaniego Campoverde, Verónica Elizabeth Silva Guadalupe** y dirigida por: **Ingeniero Fabián Gunsha.**

Una vez escuchada la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constatado el cumplimiento de las observaciones realizadas, remite el presente para uso y custodia en la biblioteca de la Facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

Ing. Paulina Vélez

Presidente del Tribunal **Firma**

Ing. Fabián Gunsha

Director del Proyecto **Firma**

Ing. Giovanny Cuzco

Miembro de Tribunal **Firma**

CERTIFICACIÓN DEL TUTOR

Certifico que el presente trabajo de investigación previo a la obtención del grado de Ingeniero en ELECTRONICA Y TELECOMUNICACIONES. Con el tema: **"DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO PARA LA CANCELACIÓN DE RUIDO CON REDES NEURONALES UTILIZANDO DSP"** ha sido elaborado por los estudiantes **Miguel Samaniego** y **Verónica Silva,** el mismo que ha sido revisado y analizado en un cien por ciento con el asesoramiento permanente de mi persona en calidad de Tutor por lo que se encuentran aptos para su presentación y defensa respectiva.

Es todo cuanto puedo informar en honor de la verdad.

Ing. Fabián Gunsha

C.I. 060260177-5

AUTORÍA DE INVESTIGACIÓN

La responsabilidad del contenido de este Proyecto de Graduación, nos corresponde exclusivamente a **Luis Samaniego, Verónica Silva** e **Ingeniero Fabián Gunsha;** y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

Luis Miguel Samaniego Campoverde

C.I 060437799-4

Verónica Elizabeth Silva Guadalupe

C.I 060425740-2

AGRADECIMIENTO

Este presente trabajo es muestra de haber culminado nuestra vida universitaria.

Le agradezco a Dios por haberme acompañado y guiado a lo largo de mi carrera, por ser mi fortaleza en los momentos de debilidad y por brindarme una vida llena de aprendizajes, experiencias y sobretodo felicidad.

A mi madre Guadalupe, por los valores que me ha inculcado, la oportunidad de estudiar la carrera de mis sueños, por el apoyo incondicional, por la paciencia de malas noches que paso por mí y por ser la mejor madre del mundo.

Agradezco a mi hermano José que aun siendo menor para mí me dio lecciones de vida, me enseñó a ver la realidad y estuvo a mi lado para escuchar mis miedos, mis llantos y mis dolores.

Agradezco a mi querida Erika que siempre me brindó su apoyo hasta en los peores momentos, para seguir adelante.

Agradezco la confianza, apoyo y dedicación de tiempo a mi tutor de Tesis Ing. Fabián Gunsha Docente

Agradezco el apoyo de mis amigos, familia y todos los seres queridos que me apoyaron en momentos críticos y pasaron a ser un recuerdo más en mi mente.

Luis Samaniego

AGRADECIMIENTO

A mis padres Wilson, Marlene y Rosita cuyo apoyo y ayuda diaria que me han llevado hasta donde estoy ahora, símbolos de trabajo y esfuerzo constante, siempre han estado pendientes de mi formación académica que nunca han dejado de confiar en mis capacidades, a mi compañero de tesis y amigo Miguel Samaniego con quien comparto el resultado del presente proyecto donde nos hemos esforzado y puesto a prueba nuestras habilidades, mi tutor de tesis Ing. Fabián Gunsha quien nos orientó con sus conocimientos, experiencia y paciencia para la culminación de este trabajo.

Verónica Silva

DEDICATORIA

Dedico este esfuerzo tan grande a la mujer que más amo en el mundo, Sra. Guadalupe Campoverde, que aparte de ser una excelente madre me dio fuerzas para seguir adelante, me enseñó a ser un luchador y alcanzar mis sueños.

También le dedico a todos mis hermanos que aun estando lejos me brindaron todo su apoyo y confianza.

Se la dedico a Erika por ser una gran mujer y estar conmigo siempre, dándome aliento para no rendirme y seguir adelante.

Luis Samaniego

DEDICATORIA

A mis padres ejemplo de esfuerzo y perseverancia, quienes han velado por mi bienestar y educación, pilares fundamentales en mi vida, entregándome su entera confianza y apoyo en todo momento y hermanas Joselin, Britany, Paola y Victoria que son mi motivación de día a día, quienes sentaron mi base de responsabilidad y deseos de superación, les dedico todo el trabajo y cariño que he puesto en la realización del proyecto de grado.

Verónica Silva

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Este trabajo presenta la implementación en hardware de un filtro adaptativo basado en redes neuronales artificiales (RNA) para la cancelación del ruido implementado en un DSP. El sistema es programado mediante lenguaje C en el Software Code Composer Studio versión 6.1.1, el algoritmo de aprendizaje empleado es la regla delta sobre una red Adaline, esta red tiene un parámetro conocido como alfa que es la razón de aprendizaje la cual modificada varía los tiempos de adaptabilidad del filtro, para implementar este filtro RNA se utilizó bloques de retraso de señal, que sirven para mantener los datos pasados mientras se los procesa. La implementación se realizó sobre el DSP TMS320C5515 de Texas Instruments ya que esta guiado al procesamiento digital de señales de alta velocidad. Las pruebas se realizaron con diferentes tipos de señales como triangular, sinusoidal y pulso, también ruidos por ejemplo ruido en la autopista y áreas recreativas pregrabados para comprobar su efectividad en cada área mencionada y en tiempo real. Los resultados obtenidos presentan un significante nivel de cancelación de ruido en la señal perturbada.

UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERIA
CENTRO DE IDIOMAS

Lic. Geovanny Armas 23 de Abril del 2016

SUMMARY

This work presents the hardware implementation of an adaptive filter based on artificial neural networks (ANN) for noise cancellation implemented in a DSP. The system is programmed using C Programming Language in the Software Code Composer Studio version 6.1.1, the learning algorithm used is the delta rule on an Adaline network, this network has a parameter known as alpha which is the reason of learning, which once it is modified, it varies the times for the adaptability of the filter, in order to implement the ANN filter, signal delay blocks were used, which serve to keep the passing data while they are being processed. The implementation was done on the TMS320C5515 DSP from Texas Instruments since it is directed for the high-speed signal digital processing. The tests were conducted with several types of signals such as triangular, sine and pulse, also noises such as prerecorded noise on the highway and recreational areas in order to check its effectiveness in each area mentioned and in real time. The results obtained show a significant level of noise cancellation in the disturbed signal.

CENTRO DE IDIOMAS
COORDINACION

INTRODUCCIÓN

El ruido es una perturbación no deseada que aparece en un sistema de comunicaciones, sin embargo no tiene relación alguna con la señal original, de manera que debe hacerse un tratamiento de filtrado adicional.

> El procesamiento de señales en la búsqueda de optimizar la transmisión de información ha desarrollado técnicas de procesamiento estadístico de señales con métodos de predicción de señal o patrones dinámicos. Sin embargo los procesos matemáticos que se usan como linealidad, sistema estacionario y sistema estadístico de segundo orden, aunque no todas las señales físicas con las que se trabaja en tiempo real son generadas por procesos dinámicos que son simultáneamente no lineales, no estacionario y no estadístico, entonces el sistema de procesamiento de señal resulta no ser eficiente. (Montero, 1999)

> La cancelación de ruido sobre la adquisición de señales siempre ha sido un tópico de interés, actualmente este proceso puede verse en la nueva generación de audífonos portátiles, los cuales tienen incluido un micrófono que captura el sonido ambiental y lo considera fuente de ruido. El usuario logra escuchar la canción de su reproductor más el sonido ambiental, por lo tanto el sistema de cancelación de ruido eliminará el sonido ambiental permitiendo al usuario escuchar únicamente el sonido deseado. Otro campo de gran auge sobre la cancelación de ruido es la robótica, un sistema robótico puede alterar su comportamiento ante la presencia de este, nuevamente se puede usar el sistema de cancelación para mejorar el funcionamiento de estos sistemas. (Santiago, 2015)

Existen diferentes métodos desarrollados para la extracción de ruido en señales contaminadas, como son:

- Métodos clásicos de cancelación de ruido aplicados esencialmente en la rama de las comunicaciones eléctricas.

- Métodos clásicos de procesamiento estadístico de señal.

- Métodos adaptativos de cancelación de ruido.

> Mientras más adaptativo se haga un sistema no lineal, más robusto será su comportamiento y mejor su operación en ambientes no estacionarios. Los filtros adaptativos tienen como propiedad ajustarse de forma automática a los parámetros de acuerdo a las variaciones estadísticas, si trabajan conjuntamente con redes neuronales tienen como ventaja presentar de forma interna una función de activación no lineal por lo tanto, las RNA tienen la capacidad de modelar las no linealidades. (Montero, 1999)

El trabajo utiliza el método adaptativo de cancelación de ruido empleando redes neuronales. “Una de las características importantes de las redes neuronales es su capacidad de aprender del entorno, y mediante el aprendizaje, mejorar en algún sentido su actuación” (Santiago, 2015). Se revisará la implementación de un filtro FIR para emplear una red neuronal ADALINE con la regla de aprendizaje delta, eligiendo una red neuronal de este tipo debido a su mayor inmunidad al ruido en comparación con un arreglo tipo PERCEPTRÓN.

El DSP TMS320C5515 fue escogido debido a sus características como velocidad de procesamiento, disponibilidad de memoria, ADC para la conversión de la señal, cuyos parámetros que son importantes para la recomposición de la señal deseada, descritos más delante de manera profunda.

El software de programación Code Composer Studio (CCStudio o CCS) es un entorno de desarrollo integrado (IDE) para desarrollar aplicaciones, para explicación del funcionamiento del dispositivo este documento cuenta con una descripción más profunda sobre la creación del programa y librerías que se requirieron en la elaboración del algoritmo de una red neuronal artificial.

CAPITULO I

1. FUNDAMENTACIÓN TEÓRICA

1.1 Filtros Digitales

1.1.1 Introducción

> Los filtros digitales se caracterizan, en términos generales, por ser sistemas predecibles, flexibles, simulables, consistentes y precisos. Por una parte, es posible cambiar sus especificaciones mediante la reprogramación, sin la adición de componentes discretos como capacitores, resistores o bobinas (normalmente con un tamaño considerable y con variaciones en el funcionamiento dependiente de la temperatura o la humedad). (Caldas, 2005)

Por otra parte, su propiedad digital calcula y simula su repuesta utilizando un procesador de uso general e implementa topologías no realizables mediante el uso de componentes físicos, estos sistemas integran las ventajas propias de los procesadores digitales.

Basados en un algoritmo mediante el cual la señal de entrada o secuencia numérica se transforma en una segunda la cual es llamada salida o secuencia numérica obtenida. Cumpliendo algún criterio que actué sobre las características en el dominio de la frecuencia o tiempo.

Las propiedades deseadas en el diseño de un filtro se establecen en el dominio de frecuencia en función de la salida del filtro en magnitud y fase.
El presente documento está enfocado en el diseño de filtros digitales y las redes neuronales artificiales, que básicamente se subdividen en dos tipos: filtros FIR e IIR.

Fundamentándose en los filtros FIR por su facilidad en el ajuste de parámetros, sencillez de entendimiento del algoritmo, simplicidad para el programador al momento de desarrollarlo en un lenguaje de programación, y su respuesta lineal, los filtros FIR tienen una respuesta impulso finita aunque ocupan una gran cantidad de memoria en el dispositivo.

1.1.2 Diseño de Filtros FIR

El filtro FIR, aparte de su linealidad, invariante, causal y estable, se caracteriza por tener en su salida una respuesta impulso de longitud finita, y la salida solo depende de los valores de la entrada, nunca de la salida.

Si se quiere obtener una fase lineal, la respuesta impulso debe ser simétrica o asimétrica. La estructura transversal de un filtro FIR, corresponde a la figura (1.1) Obsérvese que un filtro FIR solo utiliza muestras retardadas de la señal de entrada, Estructura directa de un filtro FIR de orden M.

> Un filtro FIR tiene una respuesta impulso de longitud finita. Asimismo, la respuesta en frecuencia tiene fase lineal siempre que se cumplan las condiciones de simetría. El hecho de que la fase sea lineal implica que el filtro no distorsione la salida a la banda de paso. Esta característica de no-distorsión suele ser el principal motivo para utilizar filtros FIR. (Ruiz, s.f)

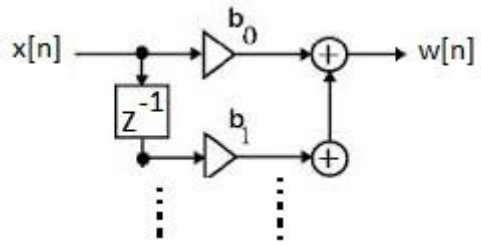


Figura 1.1 Estructura directa de un filtro FIR de orden M.
Fuente: M. Martínez – Universidad de Valencia

Los triángulos corresponden a multiplicadores y los elementos z^{-1}, con retardadores de una muestra.

1.1.2.1 Filtros FIR simétricos y Anti-simétricos

Un filtro FIR de longitud M se describe por la ecuación en diferencias:

$$y(n)=b_0x(n)+b_1x(n-1)+\cdots+b_{M-1}x(n-M+1)=\sum_{k=0}^{M-1}b_kx(n-k) \qquad (1.1)$$

o bien por la convolución:

$$y(n)=\sum_{k=0}^{M-1}h(k)x(n-k) \qquad (1.2)$$

A partir de ambas ecuaciones, se dice que: bk = h (k), k=0,1,2,...,M-1
El filtro también se caracteriza por una función de transferencia que es un polinomio de grado M-1 en la variable z-1.

$$H(z)=\sum_{k=0}^{M-1}h(k)z^{-k} \qquad (1.3)$$

Un Filtro FIR tiene una fase lineal si su respuesta impulso satisface la condición:

$$h(n)=\pm h(M-1-n) \qquad n=0,1,\dots,M-1 \qquad (1.4)$$

Teniendo en cuenta las condiciones de simetría y anti-simetría sería:

$$H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots + h(M-2)z^{-(M-2)} + h(M-1)z^{-(M-1)} \quad (1.4)$$

$$= z^{-(M-1)/2}\left\{h\left[\frac{M-1}{2}\right] + \sum_{n=0}^{(M-3)/2} h(n)\left[z^{(M-1-2k)/2} \pm z^{-(M-1-2k)/2}\right]\right\} \quad (1.5)$$

$M impar$

$$= z^{-(M-1)/2}\sum_{n=0}^{(M/2)-1} h(n)\left[z^{(M-1-2k)/2} \pm z^{-(M-1-2k)/2}\right] \quad (1.6)$$

$M\ par$

Ahora, reemplazando z-1 por z en la expresión de H(z) y multiplicamos ambos lados de la ecuación resultante por z-(M-1), tendríamos:

$$z^{-(M-1)}H(z^{-1}) = \pm H(z) \quad (1.7)$$

- Las respuesta en frecuencia de filtros FIR de fase lineal se adquiere evaluando H(z).

- Donde **h(n)=h(M-1-n)**, H(w) se puede expresar como:

$$H(w) = H_r(w)e^{j(\pi/2)} \quad (1.8)$$

Donde Hr(w) es una función real de w y se puede expresar como:

$$H_r(w) = h\left(\frac{M-1}{2}\right) + 2\sum_{n=0}^{(M-3)/2} h(n)\cos\left[w\left(\frac{M-1}{2} - n\right)\right] \quad (1.9)$$

$M\ impar$

$$H_r(w) = 2\sum_{n=0}^{(M-2)/1} h(n)\cos\left[w\left(\frac{M-1}{2} - n\right)\right] \quad (1.10)$$

$M par$

La característica de fase del filtro para M par y M impar es:

$$\theta(w)=\begin{cases} -w\left(\frac{M-1}{2}\right), si & H_\tau(w)>0 \\ \frac{3\pi}{2}-w\left(\frac{M-1}{2}\right), si & H_\tau(w)<0 \end{cases} \quad (1.11)$$

"El problema de diseño de filtros FIR es simplemente el de determinar M coeficientes h(n), n=0,1,...,M-1, a partir de una especificación de la respuesta en frecuencias deseada Hd(w) del filtro FIR." (CentOS, 2000)

1.1.3 Filtro Adaptativo

Los filtros Adaptativos operan en situaciones donde no es posible conocer la naturaleza exacta de la señal de entrada es decir, la señal de entrada siempre está expuesta variaciones de frecuencia donde entra en contacto con un bloque adaptativo, que es el que logra auto ajustar sus parámetros durante la operación a través de un algoritmo recursivo.

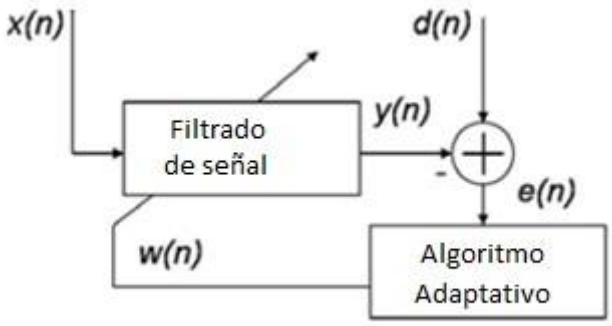


Figura 1.2 Estructura general de los filtros Adaptativos
Fuente: Redes Neuronales Artificiales - UTN

De la figura (1.2), ***x(n)*** es la señal de entrada del sistema, ***d(n)*** es la señal deseada ***y(n)*** es la salida del filtro, ***e(n)*** es la señal de error, que es semejante al filtro Wiener, la diferencia entre la señal deseada y la salida del filtro. El algoritmo de

adaptación busca corregir el valor cuadrático medio de la señal de error para alcanzar convergencia y así la adaptación.

1.1.3.1 Estructuras de filtrado adaptativo

Con las características antes mencionadas, los filtros adaptativos son útiles no sólo en el área del procesamiento digital de señales, también en aplicaciones de control. Las cuatro configuraciones de los sistemas adaptativos son:

- Identificador de sistemas
- Modelado inverso
- Predictor lineal
- Cancelador de ruido.

1.1.3.2 Identificador de Sistemas

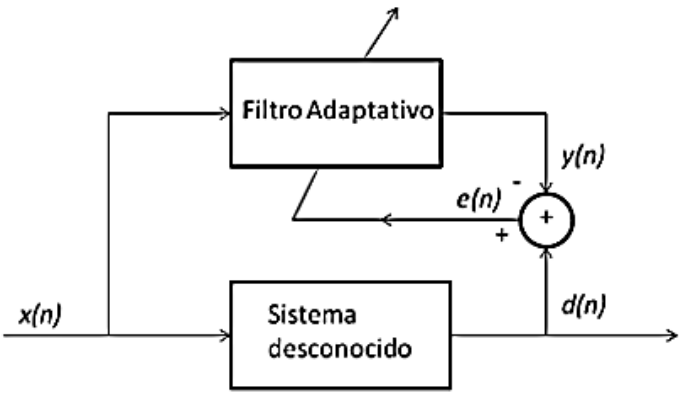


Figura 1.3 Filtro adaptativo como identificador de sistemas.

Fuente: Redes Neuronales Artificiales - UTN

Es muy útil al momento de obtener un modelo lineal en sistemas desconocidos donde la señal de entrada ***x(n)*** es la para el filtro adaptativo como para el sistema desconocido y de la salida del filtro ***y(n),*** esta actualiza los coeficientes del

sistema desconocido de la salida del filtro de forma se aproxima a la salida del sistema desconocido ***d(n)***.

1.1.3.3 Modelo Inverso

En el caso de sistemas lineales el filtro adaptativo provee el inverso de la función de transferencia del sistema, es decir *W(z)=1/H(z)*, con la combinación de la función de transferencia se tiene que *H(z)W(z)=1*. Esto se puede representar como un medio ideal de transmisión, esta técnica es utilizada para ecualizar e igualar los canales de transmisión de datos.

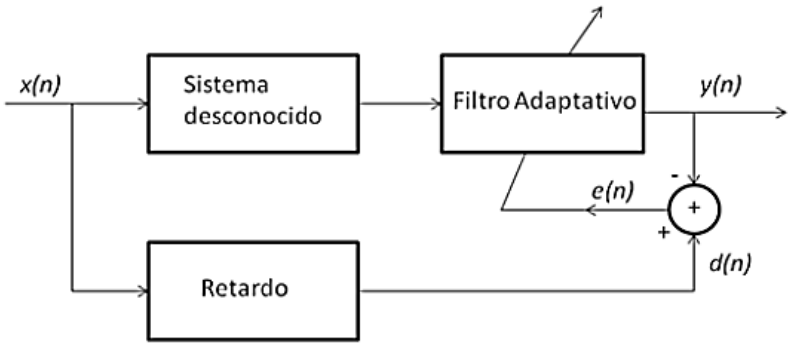


Figura 1.4 Filtro adaptativo como identificados de sistemas.
Fuente: José Hernández - ETSI

1.1.3.4 Predictor Lineal

Esta estructura, la señal deseada ***d(n)*** es la señal de entrada ***x(n),*** el error es la diferencia entre la señal deseada y la salida del filtro adaptativo ***y(n).*** La salida del filtro adaptativo es la predicción de la señal de entrada al mismo, esto con la finalidad de poder encontrar la representación paramétrica, por medio del error, de la señal aplicada al filtro.

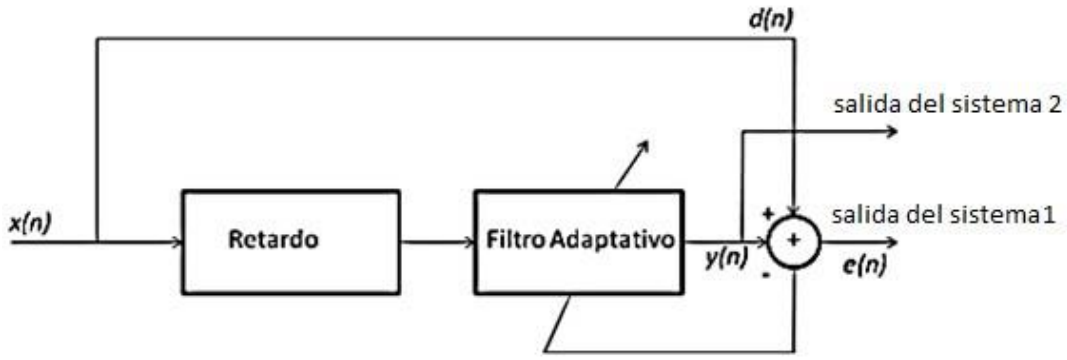


Figura 1.5 Filtro adaptativo como predictor lineal

Fuente: José Hernández - ETSI

1.1.3.5 Cancelador de Ruido

El cancelador de ruido es usado para eliminar una señal no deseada, llamada interferencia, o ruido de una señal de interés. En esta configuración la señal deseada ***d(n)*** es la señal de interés viciada por un ruido o por una señal de interferencia. La salida del filtro **e*(n)*** es la estimación del ruido, para lograr que de la diferencia entre ***d(n)*** y ***y(n)*** resulte en una señal libre de ruido o de interferencia.

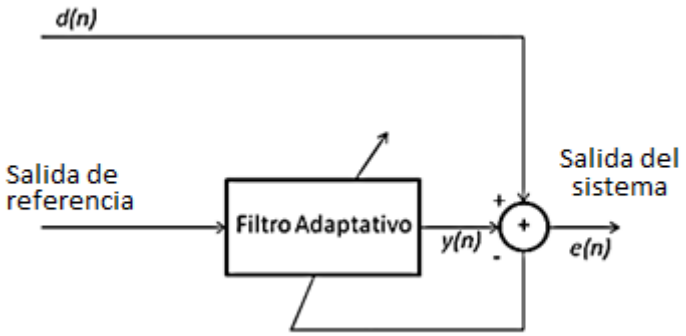


Figura 1.6 Filtro adaptativo como cancelador de ruido

Fuente: José Hernández - ETSI

1.1.4 Algoritmo de Mínimos Cuadrados Promediados LMS

Los algoritmos LMS están basados en la búsqueda del gradiente, es el máximo representante debido a su baja complejidad computacional, flexibilidad y robustez. Mencionaremos ventajas y desventajas del algoritmo.

Ventajas:

- Es considerado un algoritmo muy simple. No solicita medidas de las funciones de correlación, ni tampoco inversión de la matriz de correlación.
- El objetivo es que utilizando métodos estadísticos, reduzca el ruido que se ha introducido en la señal de entrada, para que de esta forma la señal de salida del filtro se aproxime lo más posible (sentido cuadrático medio) a la señal deseada (libre ruido).
- Posee características aleatorias en su sistema de actualización de pesos, en la acción llamada paso de adaptación utiliza una variable aleatoria y no un gradiente determinístico, el efecto es que los coeficientes del filtro pasan a ser una variable aleatoria cuya media es el filtro óptimo.
- Para el control y cálculo de la convergencia, estabilidad y robustez del algoritmo, incluye el parámetro de paso step size, el valor de la velocidad de convergencia μ.

Desventajas:

- La convergencia es lenta debido a que requiere un mayor número de iteraciones para lograr la convergencia, este defecto se recompensa con la velocidad del equipo debe ser más rápido.
- Alto número de coeficientes para funcionar, esto es un limitante.
- El cálculo del error es inferior al de otros algoritmos adaptativos.

- "Los algoritmos LMS internamente producen un ruido aproximado al 10%, sin contar otros ruidos externos". (Espol, 2010)
- "El algoritmo LMS al aumentar el número de muestras debería disminuir el porcentaje de error, cosa que no sucede sino que aumenta la frecuencia". (Espol, 2010)

El algoritmo de máximo descenso requiere conocer o estimar la matriz de auto correlación **Rx** y el punto de convergencia **p** para calcular el gradiente en un punto **w(n)**

$$\nabla J(w) = R_x w(n) - p \tag{1.12}$$

$$\nabla J(w) = E[x(n)x^T(n)]w(n) - E[x(n)d(n)] \tag{1.13}$$

$\boldsymbol{R_x}$ y **p** se puede estimar cuando tenemos $\boldsymbol{x_n}$ y de $\boldsymbol{d}[\boldsymbol{n}]$ de la siguiente forma:

$$\nabla J(w) = x_n x_n^T w(n) - x_n d[n] \tag{1.14}$$

Que puede interpretarse como una estima instantánea del gradiente.
El gradiente instantáneo puede estimarse puede reescribirse como:

$$\hat{\nabla} J(w) = x_n y(n) - x_n d[n] = x_n (y[n] - x_n d[n]) = -e[n]x_n \tag{1.15}$$

Es decir,

$$\nabla J(w) = -error \cdot x(n) \tag{1.16}$$

1.1.4.1 La ecuación fundamental del LMS es:

$$w(n+1) = w(n) + \mu e[n]x_n \tag{1.17}$$

1.1.4.2 Algoritmo LMS.

Inicializar w(0)

For n=0,…

$$e(n) = d(n) - w(n)^T x_n \tag{1.18}$$

$$w(n+1) = w(n) + \mu e[n] x_n \tag{1.19}$$

end

1.1.4.3 Convergencia del LMS (mínimos cuadrados promedios)

La convergencia del LMS ha de ser en media

$$\lim_{n \to \infty} E[w(n)] = w_{opt} \tag{1.20}$$

Y en MSE (error cuadrático medio)

$$\lim_{n \to \infty} E[J(n)] = Cte = J(\infty) \tag{1.21}$$

La condición de convergencia en media y MSE es:

$$0 < \mu < \frac{2}{\lambda_{max}} \tag{1.22}$$

1.1.4.4 LMS vs Máximo descenso

En la táctica de máximo descenso los coeficientes describen una trayectoria que acaba en la solución de Wiener.

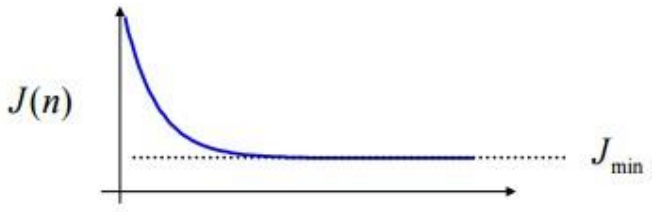


Figura 1.7 Solución de Wiener

Fuente: Redes Neuronales Artificiales – UTN

En el LMS describen un movimiento aleatorio alrededor de la solución de Wiener

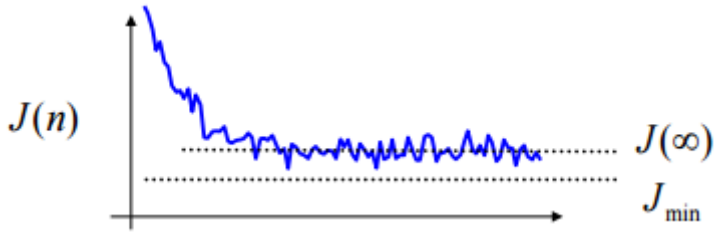


Figura 1.8Movimiento LMS

Fuente: Redes Neuronales Artificiales - UTN

1.1.4.5 Ruido de desajuste o MSE (error cuadrático medio)

- Debido a su convergencia ruidosa, el LMS siempre provoca un MSE en exceso del mínimo dado por el filtro de Wiener.
- El exceso de MSE es medido mediante el denominado ruido de desajuste D, que se define como:

$$D = \frac{J(\infty) - J_{min}}{J_{min}} \qquad (1.23)$$

Demostrando que para un μ suficientemente pequeño D viene dado por:

$$D = \frac{\mu}{2} M \sigma_x^2 \qquad (1.23)$$

Es decir,

"Para una velocidad de convergencia μ dada, el desajuste aumenta con el número de coeficientes del filtro *M*. El desajuste es inversamente proporcional a la velocidad de convergencia". (GTASE, s.f)

1.2 Redes Neuronales Artificiales

1.2.1 Introducción

Resulta irónico pensar que máquinas de cómputo capaces de realizar 100 millones de operaciones en coma flotante por segundo, no sean capaces de entender el significado de las formas visuales o de distinguir entre distintas clases de objetos. Los sistemas de computación secuencial, son exitosos en la resolución de problemas matemáticos o científicos, en la creación, manipulación y mantenimiento de bases de datos, en comunicaciones electrónicas, en el procesamiento de textos, gráficos y auto edición, incluso en funciones de control de electrodomésticos, haciéndolos más eficientes y fáciles de usar, pero definitivamente tienen una gran incapacidad para interpretar el mundo. Esta dificultad de los sistemas de cómputo que trabajan bajo la filosofía de los sistemas secuenciales, desarrollados por Von Neuman, ha hecho que un gran número de investigadores centre su atención en el desarrollo de nuevos sistemas de tratamiento de la información, que permitan solucionar problemas cotidianos, tal como lo hace el cerebro humano, este órgano biológico cuenta con varias características deseables para cualquier sistema de procesamiento digital. Tales como:

Altamente paralelo, pequeño, compacto y consume poca energía, diariamente mueren neuronas sin afectar su desempeño, flexible, se ajusta a nuevos ambientes de aprendizaje, no hay que programarlo. Puede manejar información difusa, con ruido o inconsistente. Es robusto y tolerante a fallas.

El cerebro humano emula un computador en la cabeza, es capaz de interpretar información imprecisa suministrada por los sentidos a un ritmo increíblemente veloz. Logra distinguir un rostro en la obscuridad, un susurro en una sala

ruidosa, lo más impresionante de todo, es que el cerebro aprende sin instrucciones de ninguna clase y crear las representaciones internas que hacen posibles estas habilidades.

Basados en la eficacia de los procesos llevados a cabo por el cerebro, algunos investigadores han desarrollado la teoría de las Redes Neuronales Artificiales (RNA), las cuales imitan las redes neuronales biológicas, que son utilizados para aprender tácticas y soluciones basadas en ejemplos de comportamiento típico de patrones, estos sistemas no requieren de la tarea de pre-programas, sino que generalizan y aprenden de la experiencia. (Tanco, 2012)

Las aplicaciones más exitosas de las RNA son:

- Procesamiento de imágenes y de voz
- Reconocimiento de patrones
- Planeamiento
- Interfaces adaptivas para sistemas hombre/máquina
- Predicción
- Control y optimización
- Filtrado de señales

Los computadores de hoy en día procesan la información de forma secuencial, un computador serial consiste de un solo procesador que manipula instrucciones, datos, archivos que se encuentran en la memoria, todo ocurre de forma secuencial.

Las RNA no realizan instrucciones, responden en paralelo a las entradas que se les somete. El resultado no es guardado en una posición de memoria sino es el estado de la red para el cual se logra equilibrio.

1.2.1.1 Las ventajas de las redes neuronales:

- **Aprendizaje adaptativo.-** Capacidad de aprender a realizar tareas basadas en un entrenamiento o una experiencia inicial.

- **Auto-organización.-** Una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.

- **Generalización.-** Facultad de las redes neuronales de responder apropiadamente cuando se les presentan datos o situaciones a los que no habían sido expuestas anteriormente.

- **Tolerancia a fallos.-** La destrucción parcial de una red conduce a una degradación de su estructura, sin embargo, algunas capacidades de la red pueden ser retenidos, incluso sufriendo gran daño. Con respecto a los datos, las redes neuronales pueden aprender a reconocer patrones con ruido, distorsionados o incompletos.

- **Operación en tiempo real.-** Los computadores neuronales pueden ser realizados en paralelo, y se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.

1.2.2 Panorama histórico

El desarrollo del algoritmo back-propagation fue dado a conocer por Rumelhart, Hinton y Williams. Ese mismo año, el libro Parallel Distributed Processing, fue publicado por Rumelhart y McClelland, siendo este libro la mayor influencia para la aplicación generalizada del algoritmo back-propagation. (Tanco, 2012)

1.2.3 Modelo Biológico

El cerebro tiene alrededor de 10^{11} elementos interconectados y 10^{4} conexiones por elemento, a estos se los conoce como neuronas.

1.2.3.1 Componentes de una neurona

Estas neuronas tienen tres componentes principales, las dendritas, el axón y el cuerpo de la célula o soma. Las dendritas, son el receptor de la red, que cargan de señales eléctricas el cuerpo de la célula. El cuerpo de la célula, realiza la suma de esas señales de entrada. El axón es una fibra larga que lleva la señal desde el cuerpo de la célula hasta otras neuronas.

1.2.3.2 Sinapsis

El punto de conexión entre el axón de la célula y una dendrita de otra célula se llama sinapsis, la longitud de la sinapsis se determinada por medio de la complejidad del proceso químico que estabiliza la función de la red neuronal.

Un esquema minimizado de esta interconexión de dos neuronas biológicas se observa en la figura (1.9).

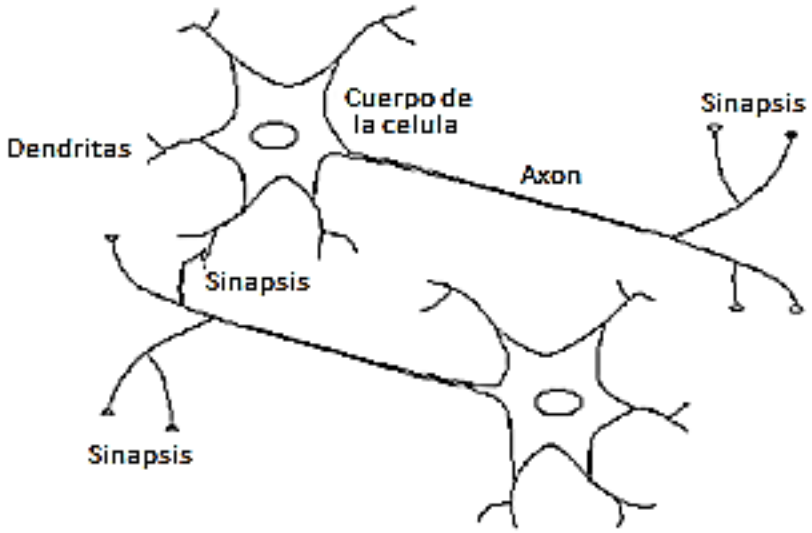


Figura 1.9 Interconexión de Neuronas biológicas

Fuente: ESCOM - IPN

1.2.3.3 Potenciales de acción

Todas las neuronas se comunican de la misma forma, esta viaja a lo largo de axones en breves impulsos eléctricos, llamados potenciales de acción, los potenciales de acción alcanzan una amplitud máxima de unos 100mV y duran un par de milisegundos, es el resultado del desplazamiento a través de la membrana celular de iones de sodio dotados de carga positiva, que van desde el fluido extracelular hasta el citoplasma intracelular, seguidos de un desplazamiento de iones de potasio carga negativa que se desplazan desde el fluido intracelular al extracelular.

En la figura (1.10) se observa la propagación del impulso eléctrico a lo largo del axón.

El potencial de acción, está constituido por señales de baja frecuencia canalizadas de forma muy lenta, no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinapsis.

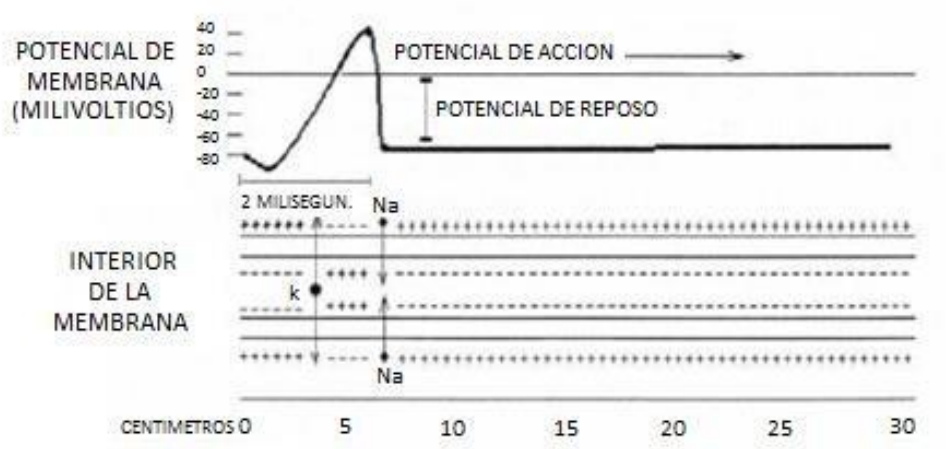


Figura 1.10 Concentración de Iones Sodio y Potasio en la membrana del axón

Fuente: Redes Neuronales Artificiales - UTN.

1.2.3.4 Tipos de sinapsis

a) *las sinapsis excitadoras*, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula pos-sináptica, facilitando la generación de impulsos a mayor velocidad.

b) *las sinapsis inhibidoras*, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos.

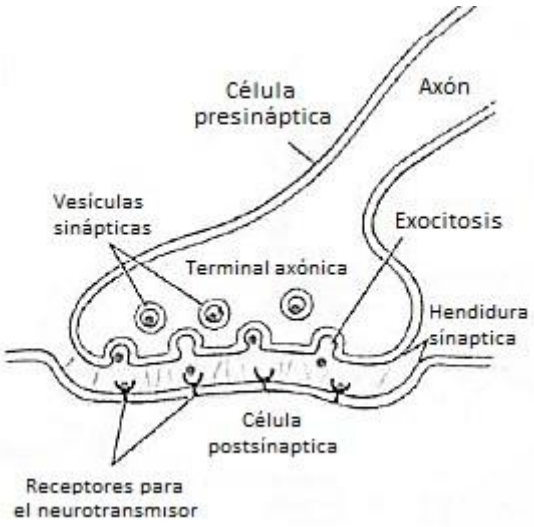


Figura 1.11 Esquema de sinápsis química, vista del botón y la membrana presináptico.

Fuente: Biologiacelularb

La neurona recibe las entradas provenientes de la sinapsis excitadora e inhibidora, es decir recibe impulsos amplificados o atenuados. La suma de éstos impulsos en el cuerpo de la célula determinará si será o no estimulada, dependiendo si el valor de la suma de señales es mayor o menor al umbral de accionamiento.

1.2.4 Neurona Artificial

El modelo de una RNA es una réplica del proceso de una neurona biológica, puede también asemejarse a un sumador hecho con un amplificador operacional tal como se ve en la figura (1.12).

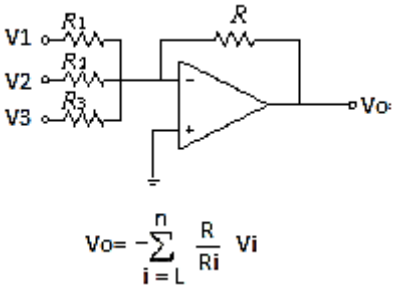


Figura 1.12 Amplificador Operacional Sumador

Fuente: Autores

Donde **(Vo)** es el voltaje de salida, **(Vi)** el voltaje de entrada, **(R/RI)** es la ganancia del sistema y **(n)** es el número de señales sumadas.

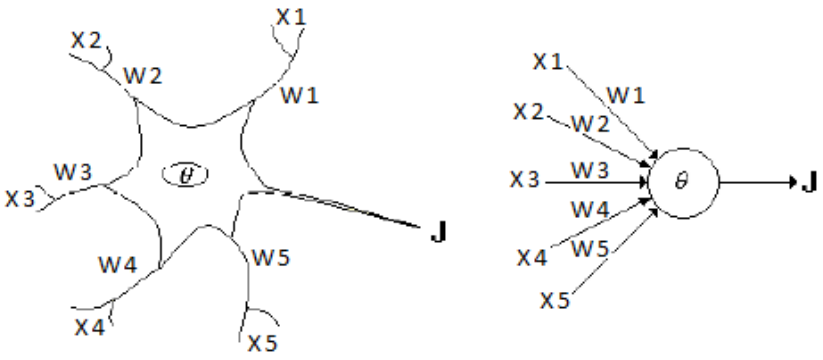


Figura 1.13 Neurona Biológica y Neurona Artificial

Fuente: Redes Neuronales Artificiales - UTN

Existen varias formas de nombrar una neurona artificial, es conocida comúnmente como nodo, neuro-nodo, celda, unidad o elemento de procesamiento (PE). En la figura (1.13) se observa un PE en forma general y su similitud con una neurona biológica:
Del proceso biológico se han hallado los siguientes análogos con el sistema artificial:

- Las entradas X_i representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.

- Los pesos W_i son la intensidad de la sinapsis que conecta dos neuronas, tanto X_i como W_i son valores reales.

- Θ es la función umbral que la neurona debe sobrepasar para activarse, este proceso ocurre biológicamente en el cuerpo de la célula.

Las entradas a una neurona artificial X_1, X_2,.., X_n son variables continuas en lugar de pulsos discretos, igual como sucede en una neurona biológica.
Toda señal que entra pasa a través de una ganancia o también llamado peso sináptico cuya función es análoga a la de la función sináptica de la neurona biológica. Los pesos pueden ser positivos o negativos, el nodo sumatorio almacena todas las señales de entradas multiplicadas por los pesos y las pasa a la salida a través de una función umbral. La entrada neta a cada unidad se escribe:

$$net_i = \sum_{i=1}^{n} W_i \mathrm{Xi} = \vec{X}\vec{W} \qquad (1.24)$$

En la figura (1.14) muestra el proceso en donde puede observarse el recorrido de un conjunto de señales que entran a la red.

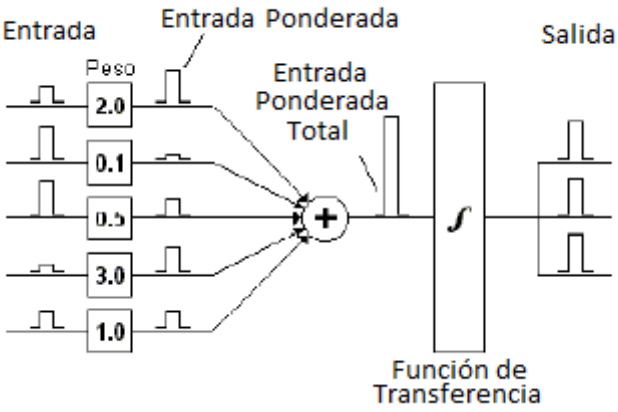


Figura 1.14 Ingreso de señales a la red

Fuente: Neuronal Network Toolbox [Tm] 7.

Una vez calculado la activación del nodo, el valor de salida equivalente será:

$$x_i = f_i(net_i) \tag{1.25}$$

Donde f_i es la función de activación para la unidad, que corresponde a la función escogida para transformar la entrada $neta_i$ en el valor de salida x_i y que depende de las características específicas de cada red.

1.2.5 Funciones de Transferencia

Un modelo más factible que facilita el estudio de una neurona, puede visualizarse en la figura (1.15):

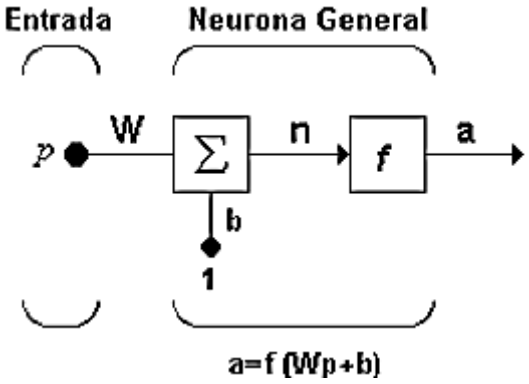


Figura 1.15 Vector p entrada a la neurona

Fuente: Neuronal Network Toolbox [Tm] 7.

Las entradas a la red ahora serán representadas en el vector (p), para el caso de una neurona que contiene solo un elemento, ***w*** sigue representando los pesos y la nueva entrada ***b*** es una ganancia que refuerza la salida neta de la red (**n)**, la salida total está determinada por la función de transferencia, la cual es una función lineal o no lineal, y que es escogida dependiendo de las características del problema que la neurona tenga que resolver.

Aunque las RNA están basadas en modelos biológicos, no existe ninguna limitación para realizar cambios o variaciones en las funciones de salida, así que se encontrarán modelos artificiales que no tienen nada en común con las características del sistema biológico.

1.2.5.1 Limitador fuerte (Hardlim)

La figura (1.16) muestra como esta función de transferencia aproxima la salida de la red a cero si la salida de la función es menor que cero y la lleva a uno si este argumento es mayor que uno. Esta función crea neuronas que clasifican las entradas en dos categorías diferentes, y le permite ser empleada en la red de tipo Perceptrón.

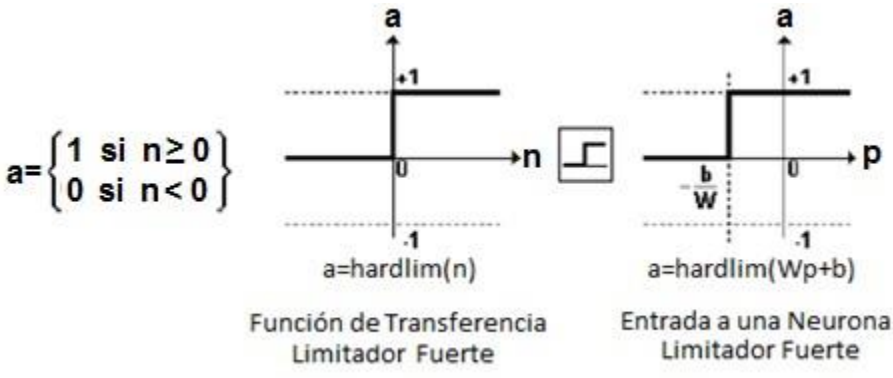


Figura 1.16 Función de Transferencia Limitador Fuerte

Fuente: Redes Neuronales Artificiales – UTN

Una modificación de esta función puede verse en la siguiente figura (1.17), la que indica la función de transferencia Hardlims que reduce el espacio de salida a valores entre 1 y –1.

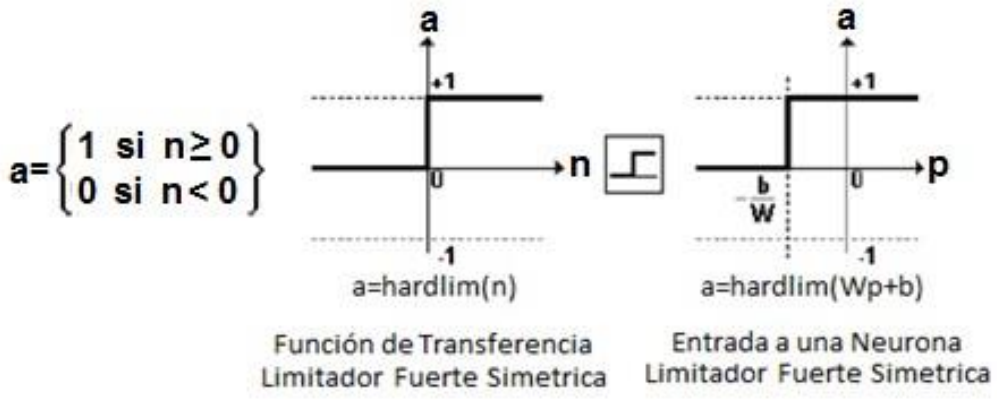


Figura 1.17 Función de Transferencia Limitador Fuerte Simétrica

Fuente: Redes Neuronales Artificiales – UTN

1.2.5.2 Función de transferencia lineal (purelin)

La salida de una función de transferencia lineal es igual a su entrada, como se puede apreciar en la figura (1.18).

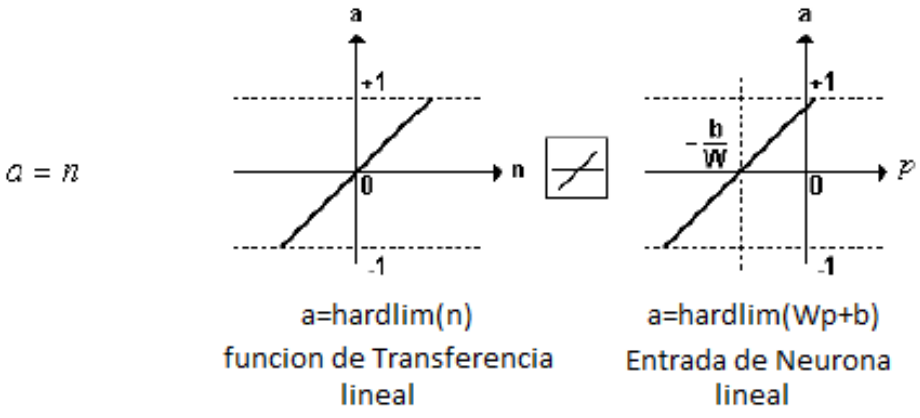


Figura 1.18 Función de Transferencia Lineal

Fuente: Grupo de Inteligencia Artificial – UTN

En la gráfica de la derecha puede verse las propiedades de la salida a de la red, comparada con la entrada (p), más un valor de ganancia (b), las neuronas que emplean esta función de transferencia son usadas en la red tipo Adaline.

1.2.5.3 Función de transferencia sigmoidal (logsig):

Nombre	Relación Entrada /Salida	Icono	Función
Limitado Fuerte	***a =0 n <0*** ***a=1 n>= 0***		***Hardürn***
Limitador Fuerte Simétrico	***a = -1 n <0*** ***a = +1 n >= 0***		***Hardlims***
Lineal Positiva	***a =0 n<0*** ***a= n 0<=n***		***Poslin***
Lineal	***a = n***		***Purelin***
Lineal Saturado	***a - 0 n <0*** ***a = n*** ***a =1 n >1***		***Satlin***
Lineal Saturado Simétrico	***a = -1 n <-1*** ***a =n -1<n<1*** ***a = +1 n>1***		***Satlins***
Sigmoidal Logarítmico	$a = \frac{1}{1+e^{-n}}$		***Logsig***
Tangente Sigmoidal Hiperbólica	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$		***TanSig***

Figura 1.19 Tabla Funciones de Transferencia en Redes Neuronales.

Fuente: Redes Neuronales Artificiales - UTN.

Esta función es utilizada en redes multicapa, como la Backpropagation, en parte porque la función logsig es diferenciable, la figura (1.19) hace una relación de las principales funciones de transferencia empleadas en el entrenamiento de redes neuronales, esta función toma valores de entrada los cuales pueden variar entre más y menos infinito, y disminuye la salida a valores entre cero y uno, de acuerdo a la expresión como se observa en la figura (1.19).

$$a = \frac{1}{1+e^{-n}}$$

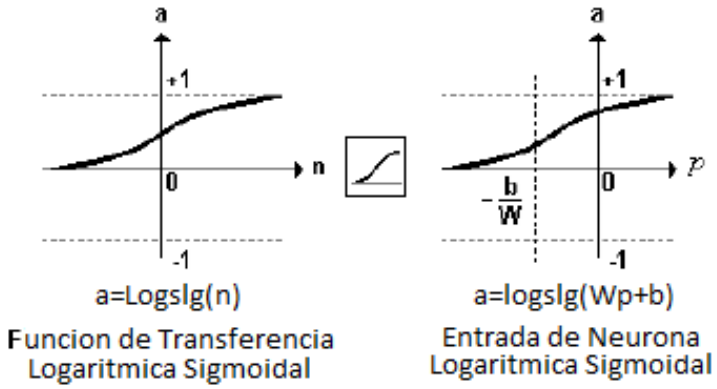


Figura 1.20 Función de transferencia sigmoidal

Fuente: inteligencia artificial- blogspot

1.2.6 Topología de una red

Típicamente una neurona tiene más de una entrada véase en la figura (1.21), una neurona con R entradas, las entradas individuales $p_1,p_2,...,p_R$ son multiplicadas por los pesos $w_{1,1}$, $w_{1,2},...w_{1,R}$ pertenecientes a la matriz de pesos $\boldsymbol{W}$.

La neurona tiene una ganancia b que mejora su salida y es la que llega al mismo sumador al que llegan las entradas multiplicadas por los pesos, para formar la salida n.

$$n=w_{1,1}p_1+w_{1,2}p_2+\cdots w_{1,R}p_R+b \qquad (1.26)$$

Esta expresión puede ser escrita en forma matricial de la siguiente manera:

$$n=Wp+b \qquad (1.27)$$

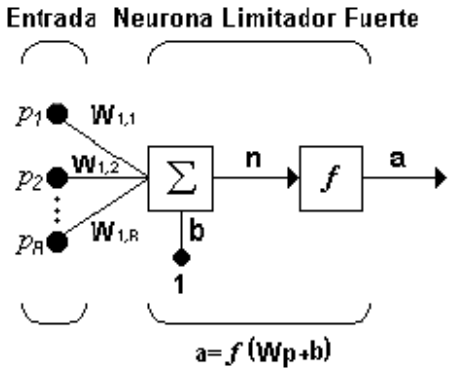


Figura 1.21Multiplicación de entradas por los pesos

Fuente: Neuronal Network Toolbox [Tm] 7.

Los subíndices de la matriz de p representan los términos involucrados en la conexión. El primer subíndice representa la neurona destino y el segundo representa la fuente de la señal que alimenta a la neurona.

Por ejemplo, los índices de $w_{1,2}$ indican que este peso es la conexión desde la segunda entrada a la primera neurona. Esta conversión es útil cuando hay múltiples neuronas, o cuando se tiene una neurona con demasiados parámetros.
En este caso la notación de la figura (1.21), puede resultar inapropiada y se prefiere emplear la notación reducida representada en la siguiente figura (1.22).

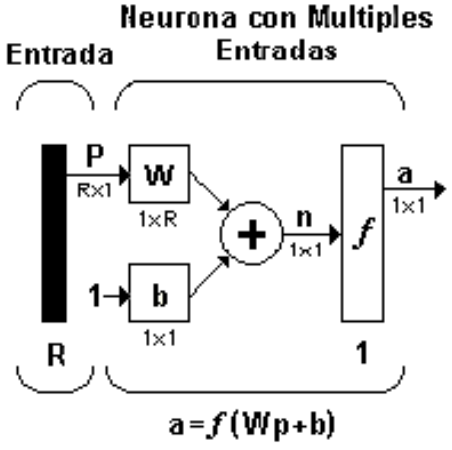


Figura 1.22Representación del vector p y matriz W

Fuente: Neuronal Network Toolbox [Tm] 7.

El vector de entrada ***p*** es sustituido por la barra sólida vertical a la izquierda. Las magnitud de p es mostrada en la parte inferior de la variable como Rx1, indicando que el vector de entrada es un vector fila de R elementos.

Las entradas son dirigidas a la matriz de pesos ***W***, la cual tiene R columnas y solo una fila para el caso de una sola neurona. La constante 1 entra a la neurona y es multiplicada por la ganancia escalar ***b***. La salida de la neurona ***a*** es en este caso un escalar, si la red tiene más de una neurona se convertiría en un vector.
En una red neuronal, los elementos de procesamiento se encuentran asociados por capas, una capa es una colección de neuronas, de acuerdo a la ubicación de la capa en la RNA, a esta se la conoce con diferentes nombres.

Capa de entrada: Adopta señales del exterior y las convierte en entradas para la red, algunos autores no consideran los vectores de entrada como una capa pues allí no se lleva a cabo ningún proceso.

Capas ocultas: Son aquellas capas que no tienen contacto con el medio exterior, y pueden tener diferentes conexiones, y son estas las que determinan las diferentes topologías de la red.

Capa de salida: Recibe la información de la capa oculta y transmite la respuesta al medio externo.

Una red de una sola capa con un número S de neuronas, como se observa en la figura (1.23) en la cual cada una de las R entradas es conectada a cada una de las neuronas, la matriz de pesos tiene ahora S filas.

La capa incluye la matriz de pesos, los sumadores, el vector de ganancias o proporcional, la función de transferencia y el vector de salida.

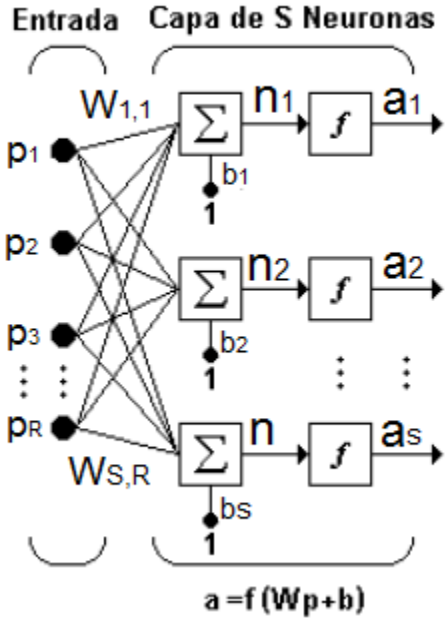


Figura 1.23 Capa con S número de neuronas

Fuente: Ariel Sabigue – Universidad de la República.

La capa de la figura (1.23) se observa en notación abreviada en la figura (1.24).

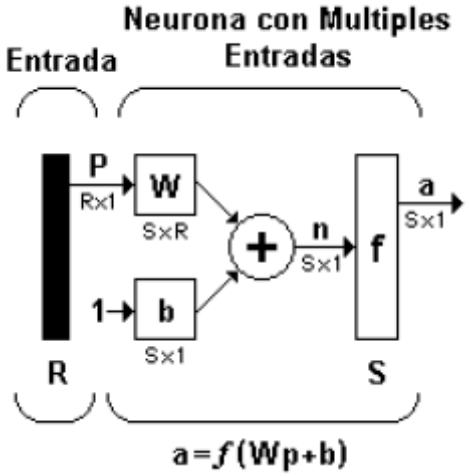


Figura 1.24 Notación abreviada

Fuente: Neuronal Network Toolbox [Tm] 7.

Los símbolos y las variables de la notación abreviada en la figura (1.24), tienen el siguiente significado:

La entrada a la red es el vector (**p)** cuya distancia (**R)** aparece en su parte inferior, (**W)** es la matriz de pesos con dimensiones (SxR) expresadas debajo del símbolo que la representa dentro de la red, **(a)** y (**b)** son vectores de longitud S, el cual representa el número de neuronas de la red.

Ahora, si se considera una red de varias capas o multicapa, cada capa tendrá su propia matriz de peso **W**, su propio vector de proporcionalidad **b**, un vector de entradas netas **n**, y un vector de salida **a**. La versión completa y la versión en notación abreviada de una red de tres capas, pueden ser visualizadas en las siguientes dos figuras (1.25) y (1.26) respectivamente.

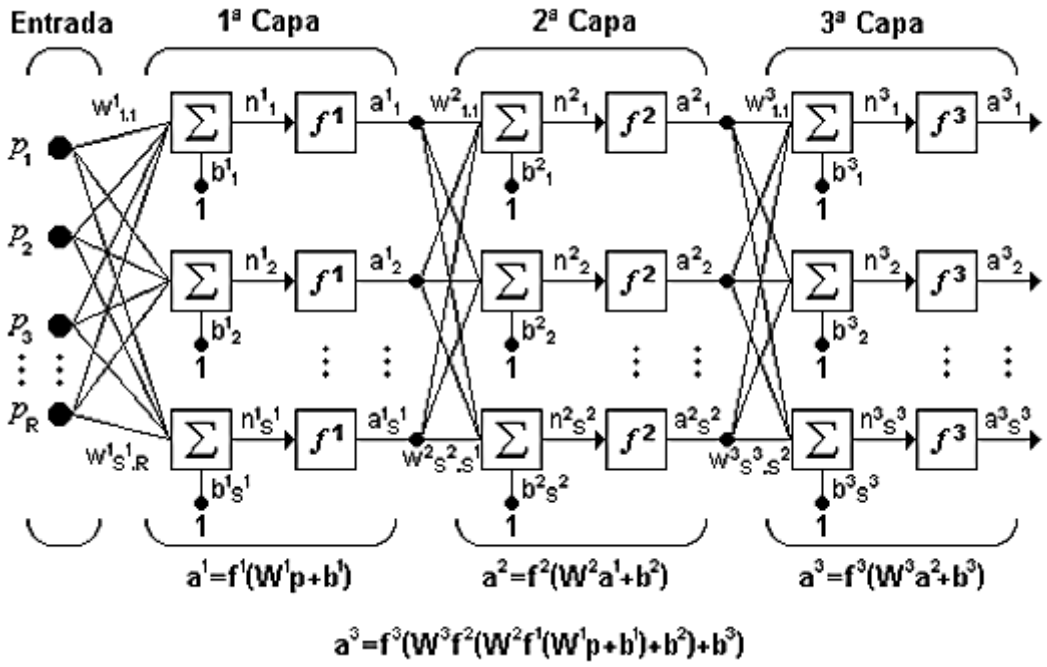


Figura 1.25 Tres capas con tres neuronas

Fuente: Neuronal Network Toolbox ™ 7.

Para esta red se tienen S^2 neuronas en la segunda capa, R entradas, S^1 neuronas en la primera capa, las cuales pueden ser diferentes, las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente, así la capa 2 puede ser vista como una red de una capa con $R=S^1$ entradas, $S^1=S^2$ neuronas y una matriz de pesos W^2 de dimensiones S^1xS^2.

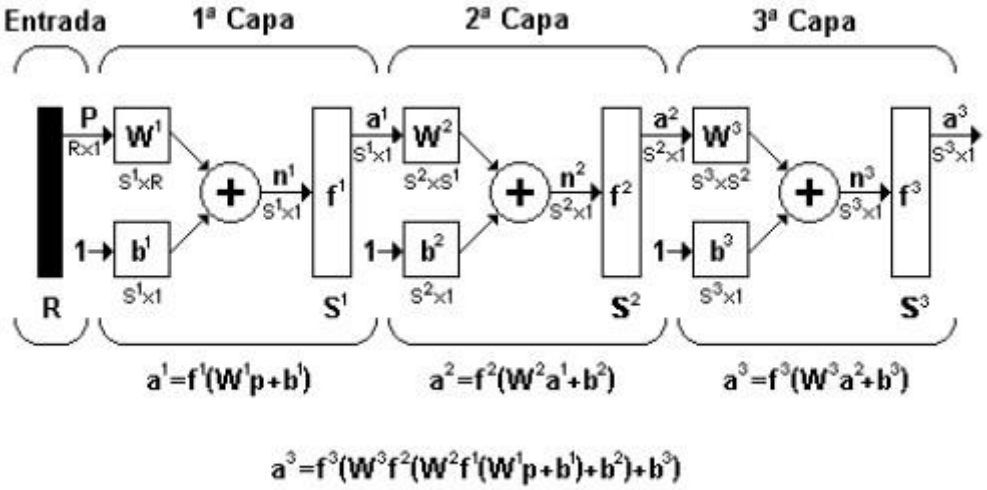


Figura 1.26 Notación abreviada de una red de tres capas
Fuente: Redes Neuronales Artificiales - UTN

Las redes de varias capas son más poderosas que las redes de una sola, por ejemplo, una red de dos capas que tenga una función sigmoidal en la primera capa y una función lineal en la segunda, puede ser entrenada y aproximar muchas funciones de forma aceptable.

En general las redes neuronales se clasifican de diversas maneras, según la topología, su forma de aprendizaje, función de activación, valores de entrada, un resumen de esta clasificación obsérvese en la figura (1.27).

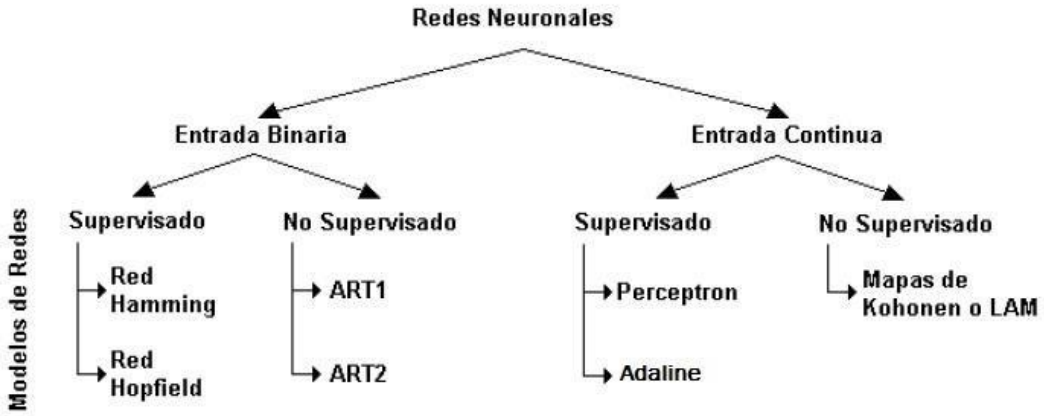


Figura 1.27 Esquema de topología de Redes Neuronales
Fuente: Grupo de Inteligencia Artificial – UTN

1.2.7 Perceptrón.

1.2.7.1 Comportamiento de una red neuronal.

Una técnica muy utilizada para analizar cómo actúan las redes es de tipo Perceptrón que representan en un mapa las regiones de decisión creadas en el espacio multidimensional de entradas a la red, en estas regiones se visualizan qué patrones pertenecen a una clase y cuáles a otra diferente, el Perceptrón separa las regiones por un híper plano cuya ecuación determina los pesos, conexiones y el valor umbral de la función de activación de la neurona, en este caso los valores de los pesos se fijan o adaptan empleando diferentes algoritmos de entrenamiento.

$$W = \begin{bmatrix} W_{1,1} & W_{1,2} \dots & W_{1,R} \\ W_{2,1} & W_{2,2} \dots & W_{2,R} \\ W_{S,1} & W_{S,2} \dots & W_{S,R} \end{bmatrix} \tag{1.27}$$

Los pesos para una neurona están representados por un vector compuesto de los elementos de la *i*-ésima fila de ***W***:

$$W = \begin{bmatrix} W_{i,1} \\ W_{i,2} \\ W_{i,R} \end{bmatrix} \quad (1.28)$$

De esta forma y aplicando la función de transferencia *hardlim* la salida de la neurona *i* de la capa de salida será:

$$a_i = hardlim(n_i) = hardlim(w_i^T p) \quad (1.29)$$

El Perceptrón, al constar de una sola capa de entrada y otra de salida con una única neurona, es capaz de representar respuestas bastante limitadas, este modelo solo puede comparar patrones sumamente sencillos, patrones linealmente separables, el caso más conocido es la dificultad al representar la función XOR (OR Exclusiva).

1.2.7.2 Estructura de la red.

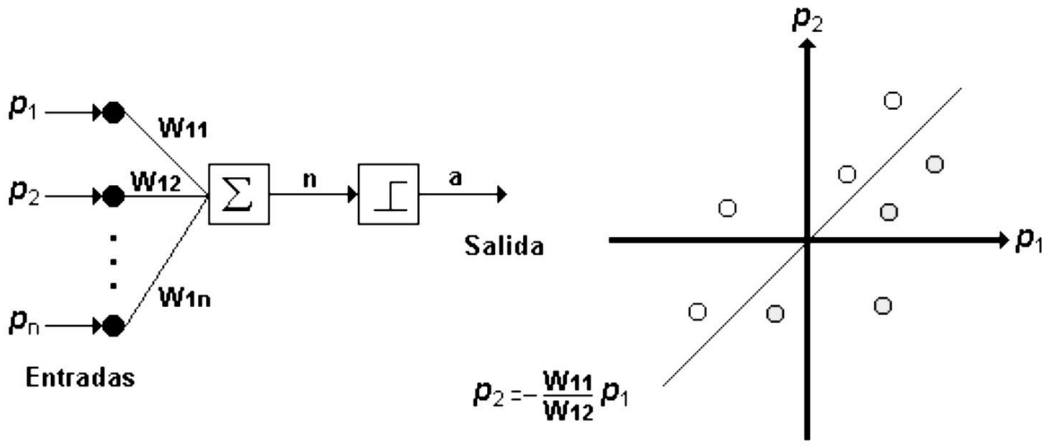


Figura 1.28 Estructura de una red Perceptrón.
Fuente: Modelo Perceptrón - UDG.

La neurona de salida del Perceptrón ejecuta la suma de las entradas, resta el umbral y pasa el resultado a una función de transferencia de tipo escalón. La decisión es responder +1 si el patrón presentado es de clase A, o –1 si el patrón es el perteneciente a la clase B, la salida depende de la entrada neta.

n = suma de las entradas pi ponderadas (1.30)

La red de tipo Perceptrón maneja principalmente dos funciones de transferencia, *hardlim* con salidas 1, 0 o *hardlims* con salidas 1, -1, su uso depende de la salida que se espera para la red, es decir si la salida de la red es unipolar o bipolar, pero la función *hardlims* es preferida sobre la *hardlim*, ya que el tener un cero multiplicando ocasiona que los pesos no se actualicen y que el aprendizaje sea más lento.

1.2.8 Adaline

El modelo para el aprendizaje de la red Adaline es llamada algoritmo LMS (Least Mean Square).

La red Adaline es parecida al de tipo Perceptrón, excepto en su función de transferencia, la cual es una función de tipo lineal en vez de un limitador fuerte como en el caso del Perceptrón. La red Adaline presenta también la limitación del Perceptrón en cuanto al problema que pueden resolver, ambas redes pueden resolver problemas linealmente separables nada más, sin embargo el algoritmo LMS es más potente que la regla de aprendizaje del Perceptrón minimizando el error medio cuadrático.

El Adaline es adaptivo en el sentido de que existe un procedimiento bien definido para modificar los pesos con objeto de hacer posible que el dispositivo proporcione el valor de salida correcto para la entrada dada, el significado de correcto para efectos del valor de salida depende de la función de tratamiento de señales que esté siendo llevada a cabo por el dispositivo. El Adaline es lineal porque la salida es una función lineal sencilla de los valores de la entrada. Es una neurona tan solo en el sentido (muy limitado) del PE. También

se podría decir que el Adaline es un Elemento Lineal, evitando por completo la definición como neurona. (Tanco, 2012)

1.2.8.1 Estructura de la red

La estructura general de una red tipo Adaline puede visualizarse en la figura (1.29).

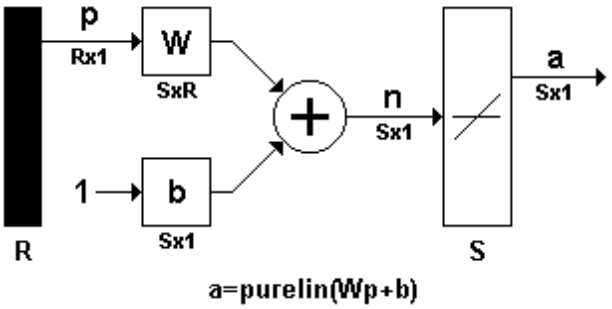


Figura 1.29 Estructura General Red Tipo Adaline
Fuente: Redes Neuronales Artificiales - UTN

La salida de la red está dada por:

$$a = purelin(Wp+b) = Wp+b \tag{1.31}$$

Para una red Adaline de una sola neurona con dos entradas el diagrama corresponde a la figura (1.30).

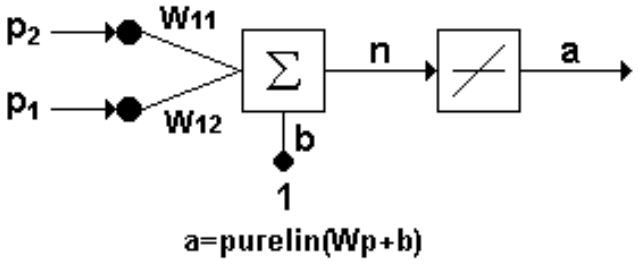


Figura 1.30 Red Adaline de una sola neurona
Fuente: Ariel Sabigue – Universidad de la República.

En similitud con el Perceptrón, el límite de la característica de decisión para la red Adaline se presenta cuando n = 0, por lo tanto:

$$w^T p + b = 0 \tag{1.32}$$

Específica la línea que separa en dos regiones el espacio de entrada, como se muestra en la figura (1.31).

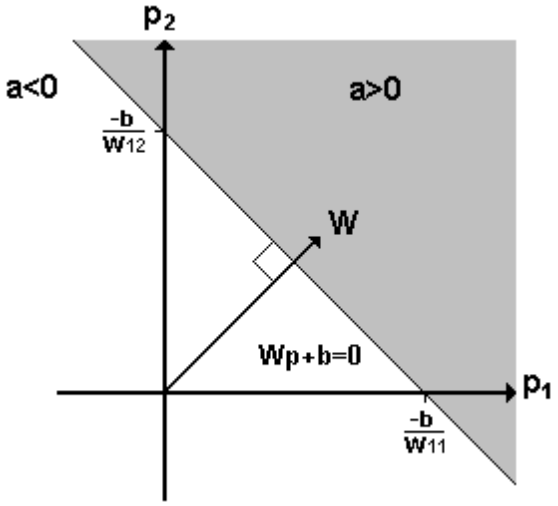


Figura 1.31 wp+b=0 es la línea que separa en dos regiones el espacio de entrada
Fuente: Redes Neuronales Artificiales - UTN

La salida de la neurona es mayor que cero en el área gris, en el área blanca la salida es menor que cero, la red de tipo Adaline puede clasificar correctamente patrones linealmente separables en dos categorías.

En este caso, la salida es la función al igual que la función de excitación, al utilizar la función identidad como función de salida y de activación establece que la salida es igual a la activación, que es la misma entrada neta al elemento. El procesamiento que ejecuta la suma total de los productos de los vectores conformados por las entradas y los pesos establecidos, se denomina combinado adaptivo lineal (ALC).

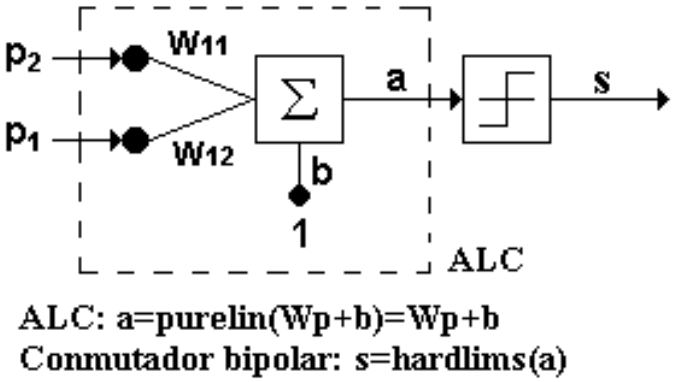


Figura 1.32 Conmutador Bipolar: s=hardlims(a)
Fuente: Ariel Sabigue – Universidad de la República.

1.2.8.2 Regla de aprendizaje

El combinador adaptativo lineal realiza el cálculo de la suma total de los productos entre las *N* señales de entrada y los pesos.

$$a = b + \sum_{j=1}^{N} w_j p_j \tag{1.33}$$

Para ejecutar la simplificación en función de la salida, podemos considerar el valor de umbral ***b*** como una conexión imaginaria de peso *w0*. Teniendo en cuenta que para esta entrada el valor de p_0=1, la ecuación (1.33) se puede escribir de la siguiente forma:

$$a = w_0 p_0 + \sum_{j=1}^{N} w_j p_j = \sum_{j=0}^{N} w_j p_j \tag{1.34}$$

La regla de aprendizaje LMS (*Least Mean Squared*) es llamada también *regla delta,* porque minimiza un delta o diferencia entre valor observado y deseado en la

salida de la neurona. La regla delta, tiene un procedimiento para calcular el vector de pesos ***W*** deseado, el cual deberá ser único y asociado con cada vector del conjunto o patrones de entrada {*X1, X2, X3, ..., XQ*} con su correspondiente valor de salida correcto o deseado {*t1, t2, t3, ..., tQ*}. La regla de aprendizaje LMS minimiza el error cuadrado medio, definido como:

$$\langle \varepsilon_k^2 \rangle = \frac{1}{2} \sum_{k=1}^{Q} \varepsilon_k^2 \tag{1.35}$$

Donde *Q* es el número de vectores de entrada es decir las entradas, que forman el grupo de entrenamiento para la neurona, y *εk* la diferencia entre la salida deseada y la obtenida cuando se introduce las entradas *k*-ésimo, que en la red de tipo Adaline, se expresa como $\varepsilon k = (tk - ak)$, siendo *ak* la salida del ALC, entonces:

$$a_k = P_k W = \sum_{j=0}^{N} w_j p_{k_j} \tag{1.36}$$

La función error es matemáticamente definida en el espacio de pesos multidimensionales para un conjunto de patrones dados. Es una superficie que tendrá muchos mínimos (Global y locales), y la regla de aprendizaje va en busca del punto en el espacio de pesos donde se encuentra el mínimo global de esta superficie. Aunque la superficie de error es desconocida, el método de gradiente decreciente consigue obtener información local de dicha superficie a través del gradiente. Con esta información, decide qué dirección tomar para llegar al mínimo global de dicha superficie. Entonces las modificaciones de los pesos son proporcionales al gradiente decreciente de la función error $\Delta wj = -\alpha (\partial \langle \varepsilon_k^2 \rangle / \partial wj)$. Por lo tanto, se deriva la función error con respecto a los pesos para ver cómo varía el error con el cambio de los pesos. Aplicando la regla de la cadena para el cálculo de dicha derivada:

$$\Delta w_i = -\propto \frac{d<\varepsilon_k^2>}{dw_i} = -\propto \frac{d<\varepsilon_k^2>}{da_k} \cdot \frac{da_k}{dw_i} \quad (1.37)$$

Los cambios en los pesos son proporcionales al gradiente descendente del error:

$$\Delta w_i = -\propto(-\varepsilon_k p_{ki}) = \propto \varepsilon_k p_{ki} = \propto(-t_k - a_k) p_{ki} \quad (1.38)$$

$$w_i(t+1) = w_i(t) + \propto(-t_k - a_k) p_{ki} \quad (1.39)$$

Siendo α la constante de proporcionalidad o tasa de aprendizaje. En notación matricial quedaría:

$$W(t+1) = W(t) + \propto \varepsilon_k P_k = W(t) + \propto(-t_k - a_k) P_k \quad (1.40)$$

Esta expresión representa los cambios de los pesos obtenidos a partir del algoritmo LMS, y es semejante a la obtenida para el caso de la red de tipo Perceptrón. **α** es el que determina la estabilidad y la velocidad de convergencia del vector de pesos hacia el valor de error mínimo. Las modificaciones en dicho vector deben hacerse relativamente pequeños en cada iteración, de lo contrario podría ocurrir que nunca encuentre un mínimo, o se encuentre solo por accidente, en lugar de ser el resultado de una convergencia sostenida hacia él.

Aunque a simple vista no exista gran diferencia entre las características de aprendizaje del Perceptrón y del Adaline, Adaline mejora al del Perceptrón ya que va a calcular de una manera más fácil el mínimo error, simplificando la convergencia del proceso de entrenamiento. En otras palabras el algoritmo de aprendizaje puede representarse como el siguiente proceso:

- Inicializa la matriz de pesos y el valor de la ganancia, por lo general asignar valores aleatorios pequeños a cada uno de los pesos *wi* y al valor *b*.k El valor de bias *b* puede tomarse como el valor de un peso adicional *w0* asociado con una entrada adicional siempre en 1.

- Aplicando un vector o patrón de entrada (*pk*) a las entradas del Adaline.
- Obteniendo la salida lineal se calcula la diferenciación respecto a la deseada.

$$a_k = f_k\left(\sum_{j=0}^{N} w_j p_{k_j}\right) \qquad \varepsilon_k = (t_k - a_k) \tag{1.41}$$

- Actualizando los pesos de esta manera:

$$W(t+1) = W(t) + \propto \varepsilon_k p_k \tag{1.42}$$

- Se repiten los pasos del 2 al 4 con todos los vectores de entrada (*Q*).
- Si el error cuadrático medio es un valor reducido aceptable, termina el proceso de aprendizaje caso contrario, repite otra vez desde el paso 2 con todos los patrones.

$$< \varepsilon_k^2 > = \frac{1}{2}\sum_{k=1}^{Q} \varepsilon_k^2 \tag{1.43}$$

1.2.8.3 Aplicación

Una aplicación comúnmente para una red de tipo Adaline es el *filtro adaptativo*, para eliminar el ruido de una señal.

Cuando se diseña un filtro digital por software, con un programa, el diseñador debe saber exactamente como especificar el algoritmo de filtrado y cuáles son los detalles de las señales, si se necesitaran modificaciones, o si cambian las características de la señal, es necesario volver a programarlo cambiado o modificando el código, para realizar una red tipo Adaline, el problema cambia, la red ahora será capaz de especificar la señal de salida deseada, dada una señal de entrada.

La red Adaline toma la entrada y la salida deseada, y se auto-ajusta para ser capaz de llevar a cabo la transformación. Si cambian las características de la señal, la red Adaline se adapta automáticamente.

Para implementar un filtro adaptivo, debe aplicarse los llamados bloques de retraso el cual es visible en la figura (1.33).

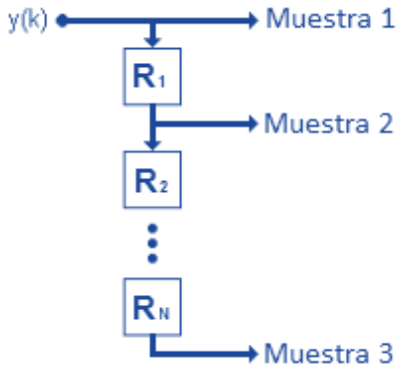


Figura 1.33 Bloques de Retardo

Fuente: Autores.

Si combinamos la red Adaline con un bloque de retardos en línea, se ha creado un filtro adaptivo como el de la figura (1.34), cuya salida está dada por:

$$a(k) = purelin(Wp + b) = \sum_{i=1}^{R} W_{1,i}\, y(k-i+1) + b \qquad (1.44)$$

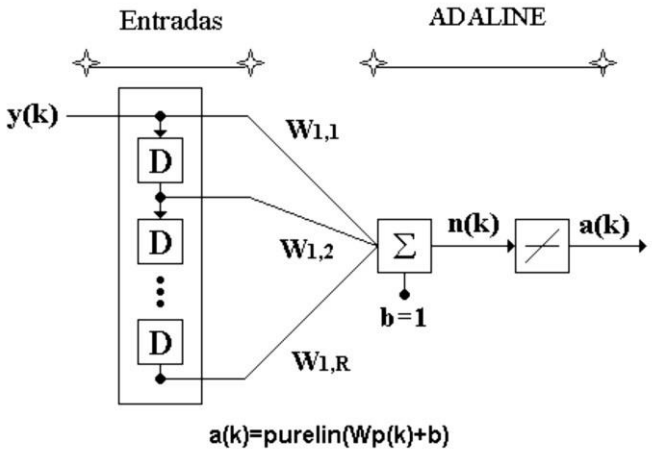


Figura 1.34 Filtro Adaptativo

Fuente: Neuronal Network Toolbox ™ 7.

1.3 Procesamiento Digital de Señales

1.3.1 Introducción

La tecnología de DSP involucra la representación digital, la transmisión y la manipulación de señales, aprovechando la capacidad y velocidad en la ejecución de algoritmos con alto número de operaciones matemáticas especializadas, que caracterizan a los procesadores digitales de señal DSPs. Se busca con ello analizar, modificar o extraer información de señales que resulten de interés en un determinado campo de la ingeniería, que normalmente son de tipo analógicos y luego son introducidas en el "mundo" digital mediante procesos de conversión apropiados. El desarrollo de la tecnología DSP ha sido sostenido por la creación de mejores algoritmos, y por el desarrollo de herramientas o plataformas de hardware más elaboradas para implementar estos últimos. Gracias a los avances logrados en la ciencia de semiconductores, se dispone ahora de sistemas completos integrados en un sistema single-chip. (Lin, 1987)

En términos generales, las soluciones de ingeniería basadas en DSP está orientada al procesamiento o análisis de señales, mayor mente de tipo análogo, partiendo de valores numéricos adquiridos de un proceso previo de muestreo ADC.

Figura 1.35 Esquema de trabajo simplificado usando DSP

Fuente: Filtro Adaptativo Difuso - IPN.

Una vez representado en forma digital, es realizado el procesamiento correspondiente, el resultado es enviado a una etapa DAC, que devuelve un equivalente análogo de la señal digital procesada. Es evidente que se puede tener señales digitales del mismo bloque de Tiempo Discreto útiles en algún proceso digital, como un lazo de control digital.

1.3.2 DSP

Las tecnologías empleadas para el procesamiento de información de manera electrónica son analógica y la digital. Donde el avance tecnológico en la computación y en la fabricación de circuitos integrados, los procesadores digitales con unas grandes velocidades de procesamiento obtenido atributos sobre los análogos haciendo de ellos una buena opción en aplicaciones que tienen que ver con señales en tiempo continúo. Las características principales por las que se utiliza procesadores digitales en lugar de analógicos son:

- Programables, permitiendo que estos sean flexibles y funcionales sin tener que agregar o quitar hardware.
- Presentan más resistencia a cambios a temperatura y de envejecimiento haciéndoles más exactos y confiables.
- Fáciles de desarrollar gracias a que son programados mediante lenguajes de programación de alto y bajo nivel.
- Las señales digitales sufren menor perdidas en transferencia a larga distancia.

Pero dentro de la categoría de procesadores digitales hay un conjunto de opciones a considerar como los microcontroladores, procesadores de propósito general, DSP’s y FPGA’s. Los tres primeros están conformados de una unidad central de procesamiento CPU con unidades funcionales como el aritmético, lógico y de

control y entrelazado con dispositivos de memoria y de entrada/salida. La diferencia entre ellos es el tamaño, la arquitectura, poder y capacidad del CPU así como del número y tamaño de sus dispositivos.

1.3.2.1 Características DSP

Hoy en día tenemos varios fabricantes de DSP's a nivel mundial, Texas Instruments y FreeScale son los nombres más comunes al preferir algunos, por lo consecuente existen DSPs con diversas características a considerar:

Formato de los datos.- Forma de manejo de datos, los DSP se clasifican en punto flotante y punto fijo. La diferencia entre estos, es el rango dinámico que es útil en algoritmos complejos como consecuencia directa, requiere más poder computacional.

Velocidad.- Los parámetros para medir la velocidad del procesador de un DSP son millones de instrucciones por segundo (MIPS), millones de operaciones flotantes por segundo (MFLOPS) o por millones de operaciones de Multiacomulación por segundo (MMACS).

Recursos computacionales.- La memoria RAM y ROM, así como temporizadores y contadores de propósito general, hacen del DSP una opción para sistemas embebidos.

Periféricos.- El número y tamaño de periféricos, permiten al DSP la comunicación con procesos externos optimizando o completando un sistema, estos pueden ser memorias externas, convertidores A/D y D/A.

El DSP escogido para llevar a cabo la implementación del filtro adaptativo con redes neuronales fue el TMS320C5515 de Texas Instruments™, sus características de mencionan a continuación.

1.3.3 Procesador Digital de Señales TMS320C5515

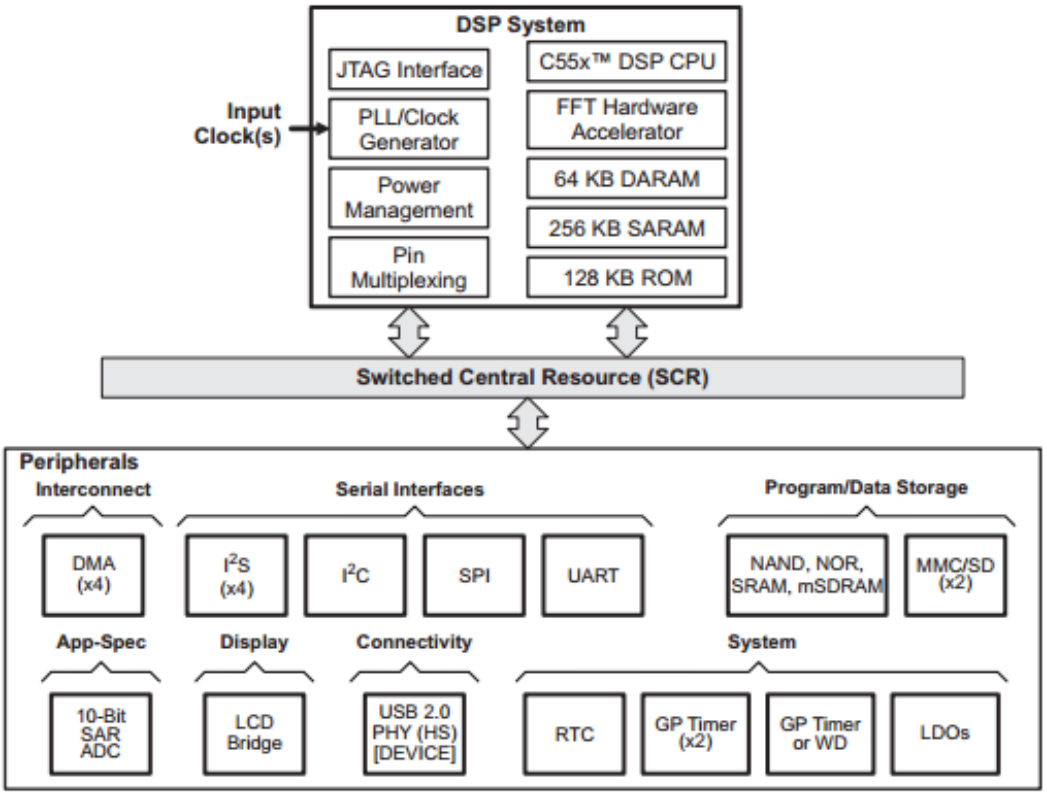


Figura 1.36 Diagrama de Bloques Funcional TMS320C5515

Fuente: Datasheet TMS320C5515.

El dispositivo es un miembro de TMS320C55xx de punto fijo procesador de señal digital (DSP) de la familia de productos TI ™ (TEXAS INSTRUMENTS) y está diseñado para aplicaciones de baja potencia. El DSP de punto fijo se basa en la generación núcleo del procesador CPU TMS320C55xx ™ DSP figura (1.37). La arquitectura C55xx ™ DSP logra un alto rendimiento y bajo consumo de energía mediante una mayor paralelismo y enfoque total en el ahorro de energía. La CPU soporta una estructura de bus interno que se compone de un bus del programa, uno con 32 bits de datos leídos (autobús) y dos con 16 bits de datos leídos (autobuses), dos datos de 16 bits escriben autobuses y autobuses adicionales dedicadas a la actividad periférica y DMA figura (1.36), estos

autobuses proporcionan la capacidad de realizar hasta cuatro datos de 16 bits lee y dos datos de 16 bits, escribe en un solo ciclo.

El dispositivo también incluye cuatro controladores DMA, cada uno con 4 canales, proporcionando el movimiento de datos para los contextos de 16 canales independientes, sin intervención de la CPU. Cada controlador de DMA puede realizar una transferencia de datos de 32 bits por ciclo, en paralelo e independiente de la actividad de la CPU. (Texas Instruments, 2015)

Figura 1.37 DSP TMS320C5515
Fuente: Datasheet TMS320C5515.

1.3.4 Tarjeta de desarrollo TMS320C5515 eZdsp

El USB C5515 eZdsp es una herramienta de evaluación para el procesador de señal digital de Texas Instruments TMS320C5515 (DSP). Es una placa de circuito impreso con base (PCB) USB que permite a los ingenieros y programadores para evaluar ciertas características de la TMS320C5515 DSP. La memoria USB C5515 eZdsp es un 2.85inch x 2.65inch seis capas placa de circuito impreso alimentado por el bus USB del ordenador personal o portátil. Esto significa que no requiere una fuente de alimentación externa. Tiene nueve conectores que proporcionan al usuario acceso a diversas señales en el C5515. La memoria USB C5515 eZdsp tiene cinco diodos emisores de luz (LED) y dos interruptores de botón, Estos están bajo el control de software de aplicación que se ejecuta en el procesador

C5515. También cuenta con seis puntos de prueba para el monitoreo de señales. (Texas Instruments, 2015)

1.3.4.1 Características

- Codec Texas Instruments TLV320AIC3204 estéreo
- Conector Micro SD
- Interfaz USB 2.0 de procesador C5515
- 32MB NOR Flash
- I2C pantalla OLED
- Compatible con Windows XP, Vista y Win7
- Conector de borde de Expansión
- Compatible con Texas Instruments estudio compositor código
- Potencia suministrada por interfaz USB
- USB incorporado JTAG XDS100 emulador

1.3.4.2 Aplicaciones

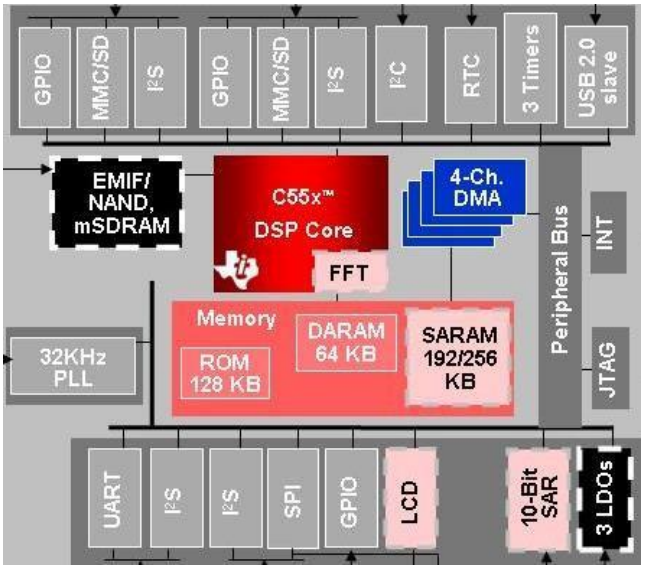


Figura 1.38 Diagrama de Bloques del TMS320C5515

Fuente: Texas Instruments.

Audio, Dispositivos Portátiles, Comunicaciones Inalámbricas figura (1.38).

1.3.5 Entrada y Salida en la tarjeta TMS320C5515 eZdsp

Las aplicaciones típicas con uso de un DSP requieren de una etapa de conversión de la señal, observado en la figura (1.39). Donde la entrada y la salida son señales analógicas pero el procesamiento efectuado de manera digital. Esto se logra mediante funciones ADC (Conversión Analógica Digital) y de forma contraria para la salida mediante la DAC (Conversión Digital Analógica).

Figura 1.39 Sistema analógico/digital, digital/ analógico

Fuente:Autores.

La tarjeta TMS320C5515 eZdsp incluye TLV320AIC3204 es un códec de audio estéreo basado en la tecnología delta-sigma. AIC3204 cubre las operaciones de 8 kHz mono de grabación estéreo a 192 kHz, y contiene configuraciones de canal de entrada programables que cubren las configuraciones de una sola terminal y diferenciales, así como flotante o mezclar las señales de entrada. También incluye un preamplificador de micrófono estéreo controlado digitalmente. Los bloques de procesamiento de señales digitales pueden eliminar el ruido audible que puede ser introducido por acoplamiento mecánico, por ejemplo, zoom óptico de una cámara digital.

El rango de alimentación de tensión analógica para el TLV320AIC3204 es únicamente de 1.5V-1.95V, y para la tensión digital es 1.26V-1.95V. La facilidad de diseño a nivel del sistema LDO (offer very low dropout) integrados que generan el análogo apropiado o la oferta digital a partir de tensiones de entrada

que van desde 1.8V a 3.6V. El dispositivo es compatible con voltajes de E / S digitales en el rango de 1.1V-3.6V.

"El PLL es altamente programable y puede aceptar los relojes de entrada disponibles en el rango de 512 kHz a 50 MHz". (Texas Instruments, 2015)

1.3.6 Entorno de Programación Code Composer Studio™

Code Composer Studio es un software de desarrollo integrado **(IDE)** que apoya a Microcontroladores y procesadores embebidos. Code Composer Studio cuenta con un conjunto de herramientas utilizadas para desarrollar y depurar aplicaciones figura (1.40), incluyendo una optimización del compilador C / C ++, editor de código fuente, entorno de construcción de proyectos, depurador, pro-file, y muchas otras características, Code Composer Studio combina las ventajas del marco de software Eclipse con capacidades de depuración integrados avanzados de TI™ que resulta en un entorno de desarrollo abundante en características de peso para desarrolladores de sistemas integrados.

Varios libre, nodo bloqueado (atado a un PC) y (red) licencias flotantes disponibles.

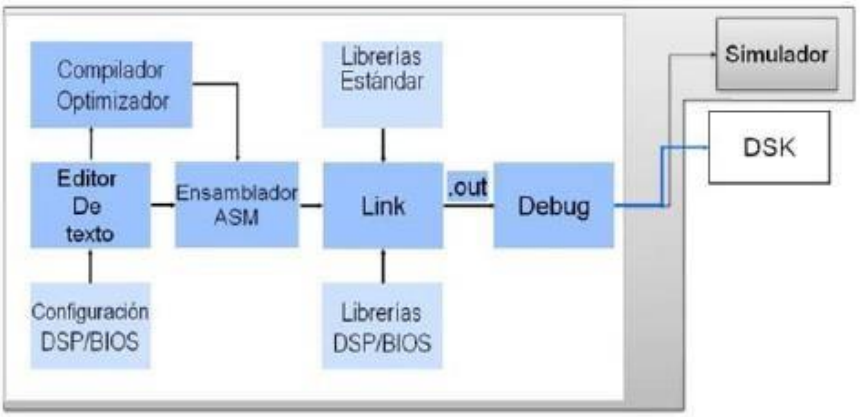


Figura 1.40 Ciclo de un proyecto en CCS

Fuente: Luis Vázquez - IPN.

El archivo fuente, escrito en C, es realizado por un editor de texto incluido en el entorno, también es posible escribir el archivo fuente en ensamblador pero no se optó por esta forma debido a la complejidad de uso y el alto grado de conocimiento que requiere esta arquitectura interna de microprocesador. Los archivos objeto (.obj) y las librerías son enlazados para crear el archivo ejecutable (.out), posteriormente el archivo .out es cargado en el DSP. La creación de un proyecto en CCS se presentará a continuación.

Inicialmente adherir todas las carpetas de soporte que necesita figura (1.41), para ello haga clic en:

- File
  - New
    - Project
      - CCS Project
        - Next

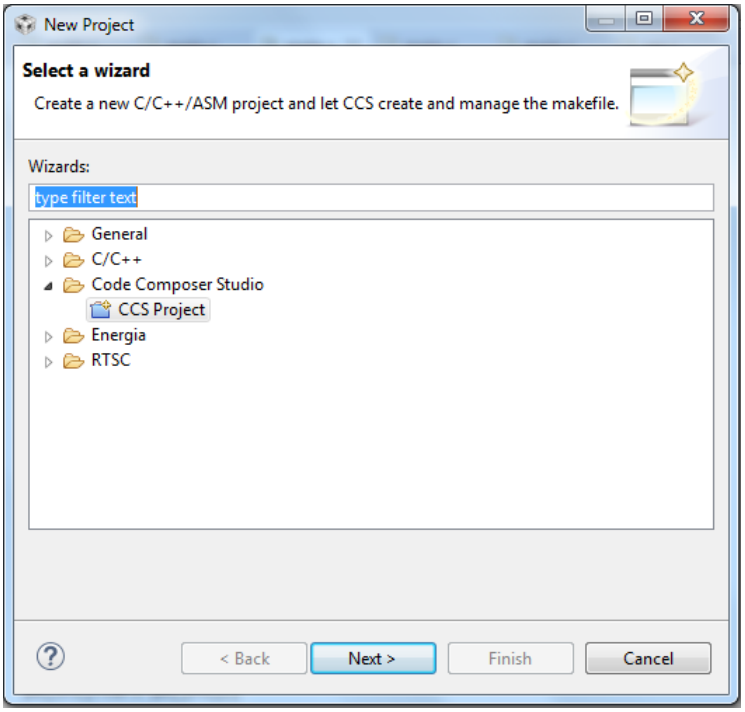


Figura 1.41 Creación de Nuevo Proyecto Code Composer Studio

Fuente: Autores.

En la siguiente ventana figura (1.42) coloque el nombre del proyecto, establezca la familia C5500 y la variante USBSTK5515, para la conexión elija XDS100v2 USB Emulator, finalmente:

- Empty Project (with main.c)
- Finish

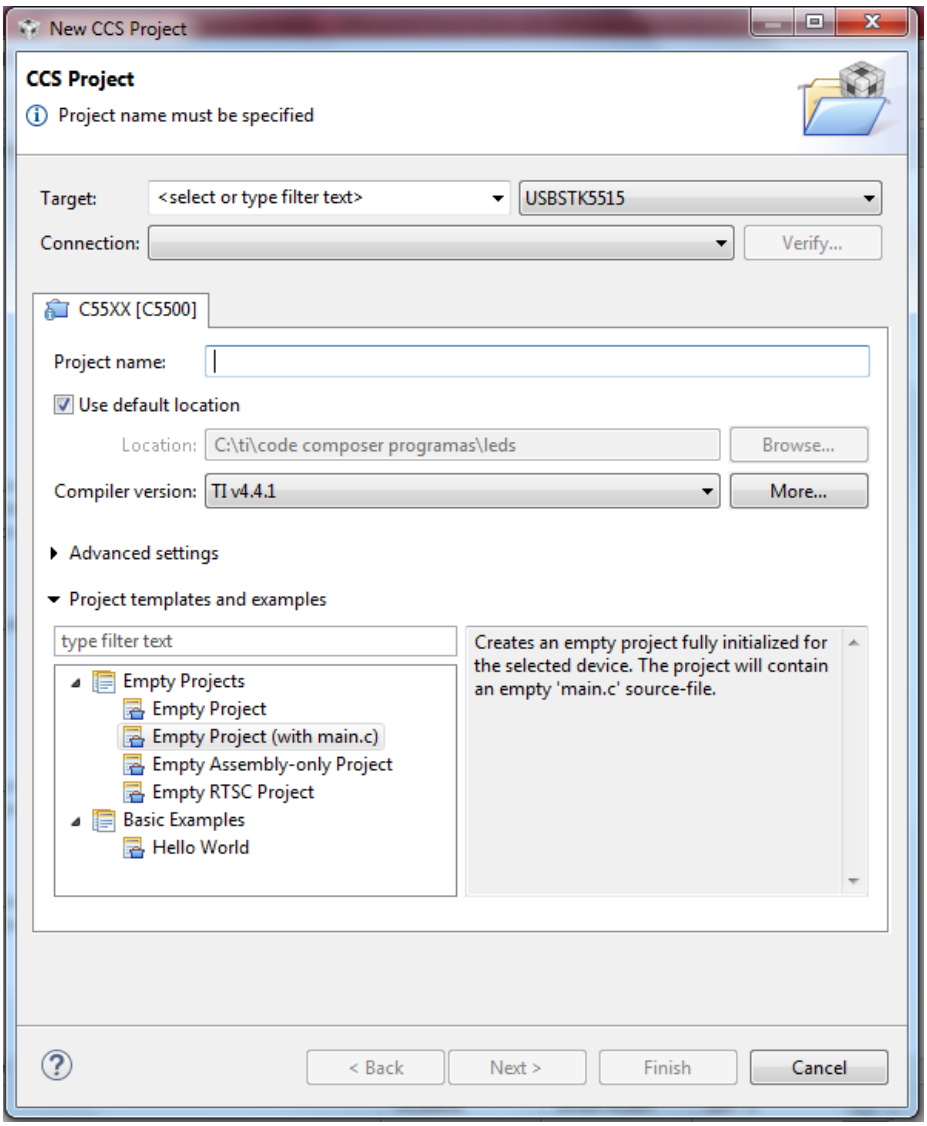


Figura 1.42 Nombre del proyecto

Fuente: Autores.

En la ventana principal haga clic derecho sobre el proyecto y seleccione:

- Properties
  - Resource
    - General

En el comando “Linker Command File”

- Browse
  - C5515 support files
    - lnkx.cmd
      - Open

En el comando “Runtime Support Library” figura (1.43).

- Browse
  - rts55x.lib

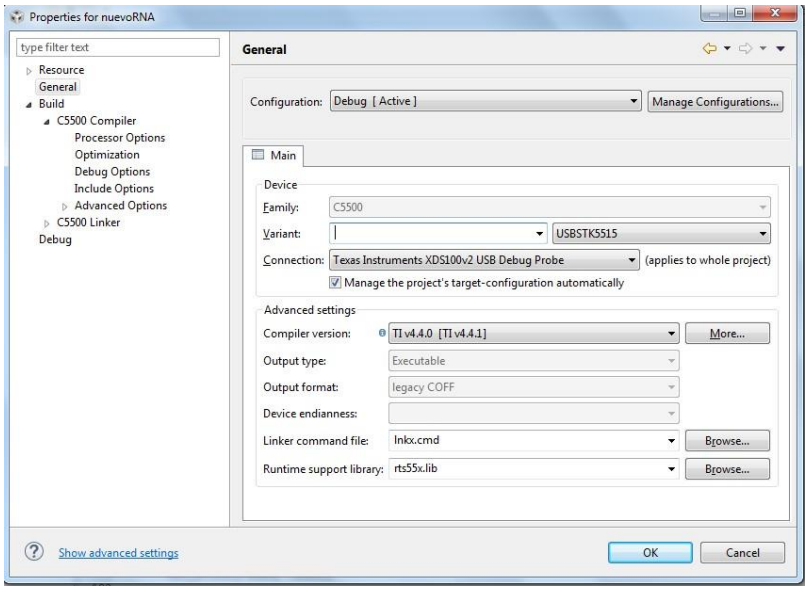


Figura 1.43 Configurar el "lnk.cmd" y "rts55x.lib."

Fuente: Autores.

Nuevamente enla ventana de Properties, hacemos clic en:

- Build
  - C5500 Compiler
    - Include Options

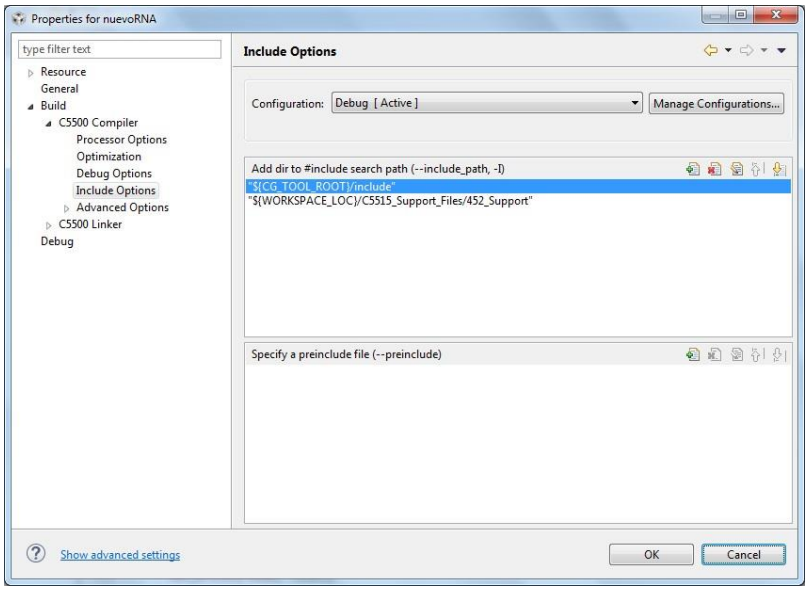


Figura 1.44 Configuramos "Include Options" y selecciona Add

Fuente: Autores.

Seleccione "Add" en la parte superior derecha y CCS solicita un directorio de ti, en el campo coloque $ {WORKSPACE_LOC} \ C5515 Support_Files \ C5515 Incluye" como se aprecia en la figura (1.45).

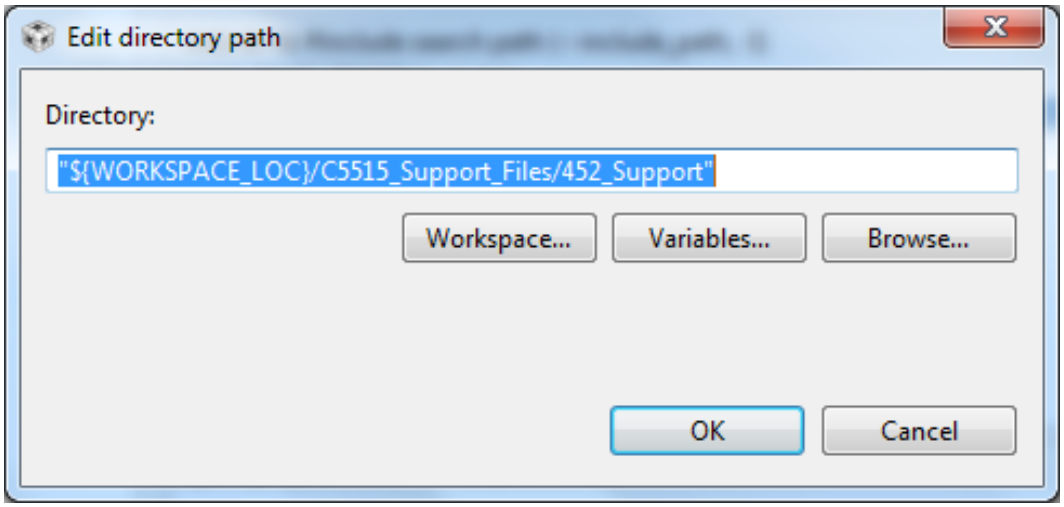


Figura 1.45 Campo directorio

Fuente: Autores.

Ahora que ya le hemos dicho al compilador dónde encontrar nuestros archivos de cabecera, todavía tenemos que decirle dónde encontrar el código para resolver los símbolos declarados en esos archivos de cabecera. Para ello, vaya a:

- Build
  - C5500 Linker
    - File Search Path

En la sección "Include library file or command", haga clic en el botón "Add" como antes. En la ventana “Add file path” que aparece, escriba "${WORKSPACE_LOC} \ C5515 Support_Files \ usbstk5515bsl.lib"

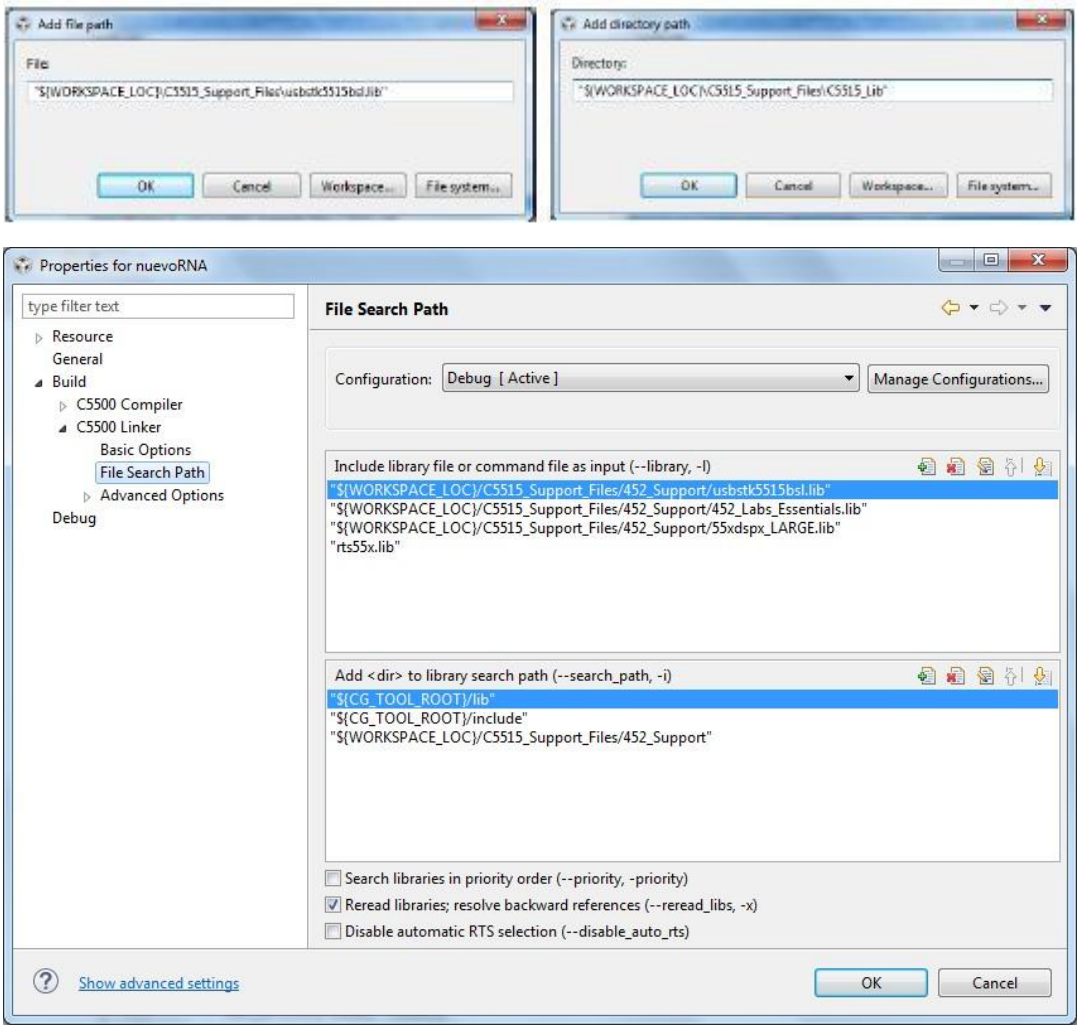


Figura 1.46 Configuración File Search Path

Fuente: Autores.

Seleccione “Add <dir> to library search path”, haga clic en el botón "add" como antes. En la ventana "Add directory path" que aparece, escriba "${WORKSPACE_LOC} \ C5515 Support_Files \ C5515 Lib" véase figura (1.46).

Todavía tenemos que decirle al compilador qué silicon version y el modelo de memoria en el que va a correr. Para ello, seleccione la opción:

- Build
  - C5500 Compiler
    - Processor Options

Vuelva a colocar el C5515 en la revision silicon version con "CPU: 3.3" (que le dice al compilador cual es la revisión del procesador que estamos usando). Asegúrese de que el modelo de memoria que se especifica sea “large” figura (1.47). Luego haga clic en “Ok” en la esquina inferior derecha de la ventana y regresará a la ventana de su proyecto.

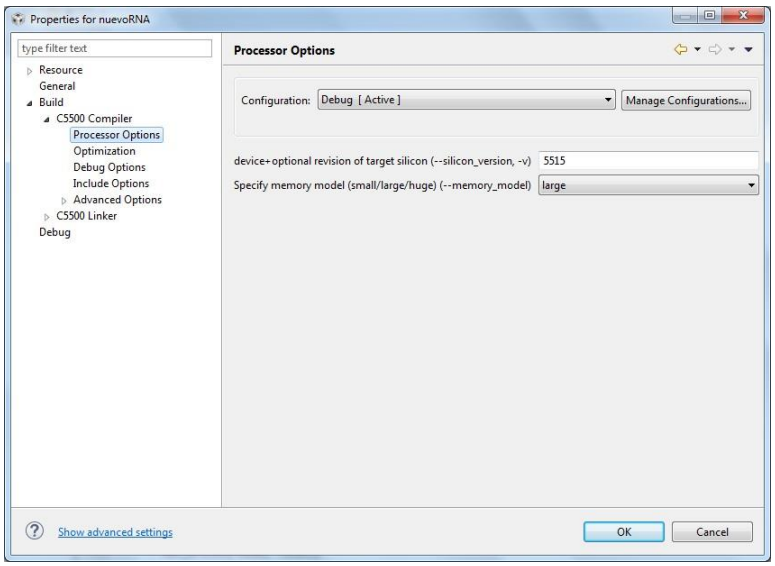


Figura 1.47 Configuración de revision silicon versión y large

Fuente: Autores.

Véase el anexo 5 el desarrollo paso a paso de ejemplos de programación en code composer studio.

CAPITULO II

2. METODOLOGIA

2.1 Tipo de estudio

2.1.1 Descriptivo

El tipo de investigación es descriptiva, puesto que se observa y describe el comportamiento de un filtro adaptativo con redes neuronales utilizando DSP y elimina el ruido en señales de audio en tiempo real, mediante el desarrollo del algoritmo de aprendizaje basado en el tratamiento de la señal de error. Para obtener una respuesta óptima, se somete a un proceso en la cual se ajustan los pesos, disminuyendo significativamente el error.

2.2 Métodos, Técnicas e Instrumentos

2.2.1 Métodos

2.2.1.1 Analítico/ Deductivo

El método empleado es analítico-deductivo en el proyecto debido al análisis particular que se realiza en la señal de ruido en el desarrollo del sistema y en la forma de interactuar entre el DSP y la red neuronal artificial previo a que el sistema desempeñe su función de manera eficiente.

2.3 Técnicas

2.3.1 Observación

La técnica es caracterizada por recolectar información de las diferentes situaciones que se producen en el entorno contribuyendo al desarrollo del proyecto, para el diseño e implementación de un filtro adaptativo con redes neuronales de tipo cancelador de ruido utilizando DSP.

2.3.1 Instrumentos

Los instrumentos necesarios son libros, archivos, páginas web y datasheet.

2.4 Población y muestra

2.4.1 Población

La población consta de los diferentes tipos de ruido y señales que contaminan la señal de audio en tiempo real:
Las fuentes de ruido provienen de autopista y áreas recreativas (audios pregrabados) para comprobar su efectividad en cada área, y también fueron realizadas en tiempo real. Las señales son de tipo triangular, impulso y sinusoidal.

2.4.2 Muestra

La muestra tomada para el análisis del proyecto de tesis será la señal contaminada (suma de la señal deseada más el ruido) y el ruido seleccionado.

2.5 Hipótesis

La implementación de un filtro adaptativo con redes neuronales utilizando DSP eliminará los problemas de ruido en señales de audio.

2.6 Operacionalización de variables

Variables	Dimensiones	Indicadores
Independiente: Diseño e Implementación de un Filtro Adaptativo utilizando Procesador Digital de Señales	Formato de Datos	Punto Flotante
		Punto Fijo
	Velocidad	MIPS
		MFLOPS
		MMACS
	Memoria	RAM
		ROM
	Periféricos	Memoria Externa
		A/D
		D/A
Dependiente: Eliminación del Ruido	Señales	Sinusoidal Triangular Cuadrática
	Audio	Ruido *Pregrabado *Tiempo Real *Voz

Tabla 1 Variable Dependiente e Independiente

Fuente: Autores.

2.7 Procedimientos

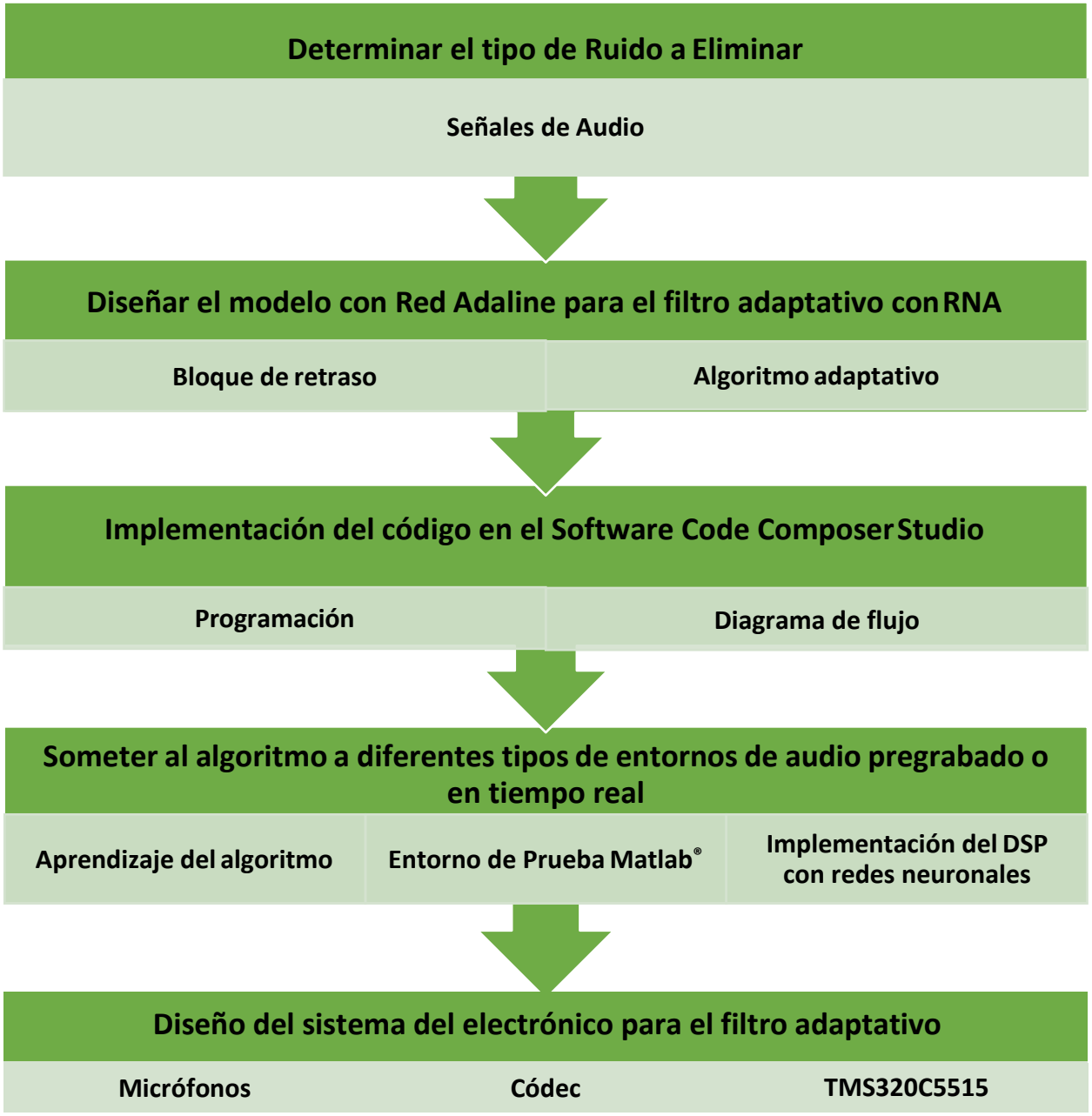


Figura 2.1 Procedimiento de Filtro Adaptativo con RNA.

Fuente: Autores

.

2.8 Procedimiento y análisis

2.8.1 Diseño

2.8.1.1 Modelo con red Adaline para el filtro adaptativo

El filtro adaptativo utilizando una red Adaline requiere de bloques de retraso de señal, los bloques permitirán que los datos de las muestras pasadas y la actual se guarden temporalmente en un vector hasta que se realice el proceso de actualización de pesos. Los bloques de retardo del arreglo neuronal están representados en la siguiente figura (2.2):

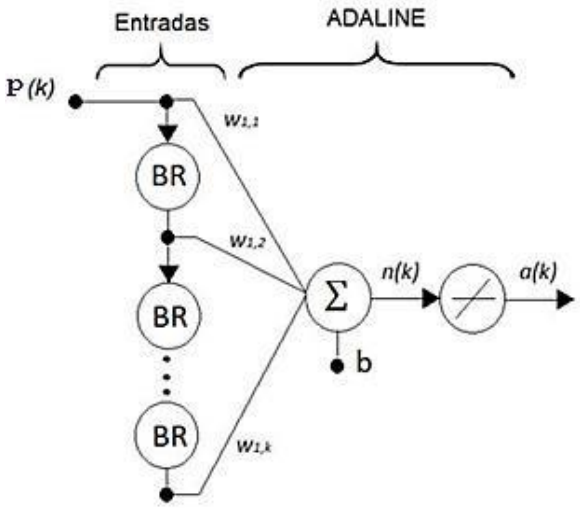


Figura 2.2 Arreglo neuronal
Fuente: Andrés Santiago - IPIITA.

Entonces del arreglo mostrado en la anterior figura obtenemos la ecuación:

$$P=[p(k),p(k-1),p(k-2),p(k-3)] \tag{2.1}$$

Regla de Aprendizaje

Adaline es una red de aprendizaje supervisado que necesita conocer anticipadamente los valores de entrada. A continuación se muestra la forma de los pares de entrada /salida:

$$\{p_1,t_1\},\{p_2,t_2\},\ldots,\{p_Q,t_Q\} \tag{2.2}$$

Donde p_Q es la entrada a la red y t_Q es su correspondiente salida deseada, cuando una entrada p es representada en la red, la salida de la red es comparada con el valor *t* que es el asociado.

El algoritmo LMS se deriva de la regla Windrow-Hoff donde deduce en términos generales que para la actualización de pesos de una red Adaline, la ecuación queda de la siguiente manera:

$$W(k+1) = W(K) + \propto \frac{e(k)p(k)}{|p(k)|^2} \tag{2.3}$$

Donde *k* representa la iteración actual del proceso de actualización, *W(k+1)* es el siguiente valor que se colocará en el vector de pesos y *W(k)* representa el valor actual en el mismo vector. El error actual *e(k)* es la diferencia entre la salida deseada *t(k)* y la salida de la red $a(k)=W^T(k)p(k)$ antes de la actualización:

$$e(k)=t(k)-W^T(k+1)p(k) \tag{2.4}$$

La variación del error en cada iteración es representada por:

$$\Delta e(k)=\Delta(t(k)-W^T(k+1)p(k))=-p^T(k)\times W(k) \tag{2.5}$$

De acuerdo a la ecuación (2.2) la actualización de pesos, considerada del error es:

$$\Delta W(k) = W(k+1) - W(k) = \propto \frac{e(k)p(k)}{|p(k)|^2} \tag{2.6}$$

Las ecuaciones (2.5) y (2.4) combinadas se obtiene:

$$\Delta e(k) = -\propto \frac{e(k)p^{T}(k)p(k)}{|p(k)|^2} = -\propto e(k) \qquad (2.7)$$

De esta manera, el error es reducido por el factor α, mientras los pesos van actualizándose a medida que se colocan valores a la entrada. Si se presenta un nuevo patrón el bucle de actualización comenzará nuevamente, el error disminuye por el factor α y el proceso sigue. Los valores iniciales del vector de pesos son escogidos, en nuestro vector inicial con [0.7 0.3 0.3 -0.3] y se actualizarán hasta que el algoritmo alcance la convergencia.

El factor α lleva el control de la estabilidad y velocidad de convergencia el proceso de entrenamiento como se observa en la ecuación (2.6), si el valor de α es muy bajo, el algoritmo pierde velocidad y tarde demasiado en alcanzar la convergencia, si el valor de α es muy alto el algoritmo pierde estabilidad y se vuelve oscilante alrededor del valor de convergencia. Para señales de entrada independientes en el tiempo, la estabilidad es asegurada para los valores que varían entre:

$$0 < \alpha < 2 \qquad (2.8)$$

Si el valor de α es mayor a 1 es innecesariamente sobre-corregido, por lo tanto un rango de valores convenientes para el ritmo de aprendizaje es:

$$0{,}1 < \alpha < 1 \qquad (2.9)$$

Extendiendo el algoritmo a la actualización de las ganancias se obtiene:

$$b(k+1) = b(k) + \propto e(k) \qquad (2.10)$$

El algoritmo de Widrow-Hoff Delta corrige el error y si las señales de entrada son de similar longitud, cuando ocurre la actualización de pesos y ganancias

tiende a reducir el error cuadrático medio, característica fundamental del algoritmo.

El proceso de actualización de pesos del el algoritmo de Widrow-Hoff Delta está dado por la siguiente ecuación:

$$w(k+1)=w(k)+2\propto e(k)p(k) \tag{2.11}$$

Finalmente se determina que la matriz de pesos W estará definida por la cantidad de bloques de retardo, el subíndice k corresponde a la cantidad de entradas a la neurona, **b** es el factor que polarizará a la neurona y como respuesta de la neurona se obtendrá **a**.

La salida **a** de la neurona es comparada con la señal para obtener el error, el error calculado es quien determinará la adaptabilidad de la neurona. El diagrama de bloques presentado a continuación muestra la idea general del proceso véase la figura (2.3).

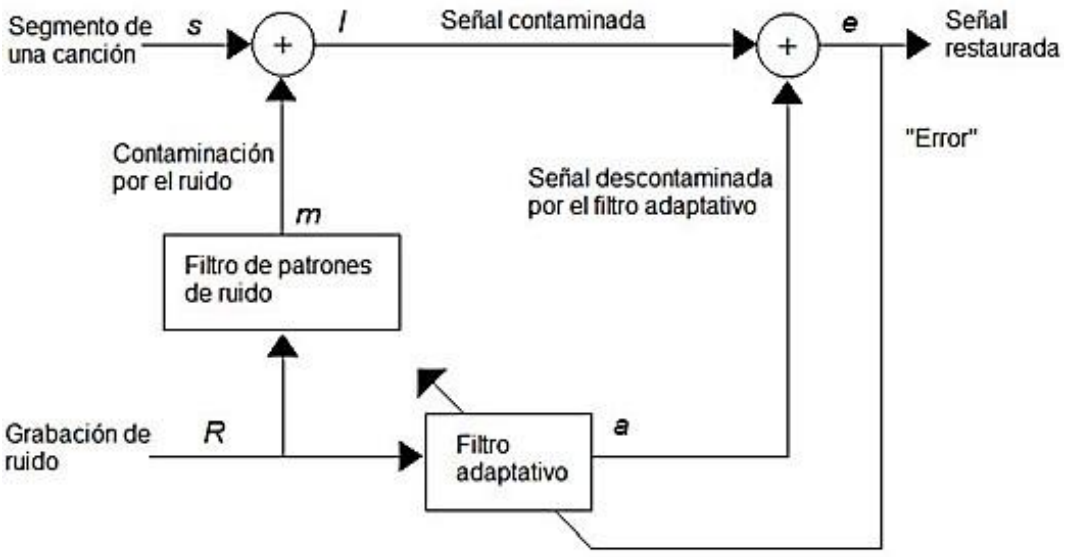


Figura 2.3 Diagrama de bloques de la señal contaminada y el error

Fuente: Filtro Adaptativo Difuso – IPN

2.8.1.2 Esquema Electrónico del filtro adaptativo

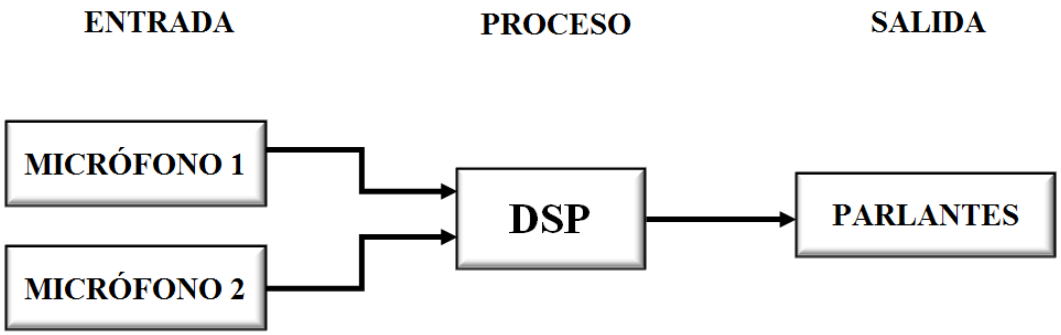


Figura 2.4 Diagrama de Bloques funcionales

Fuente: Autores

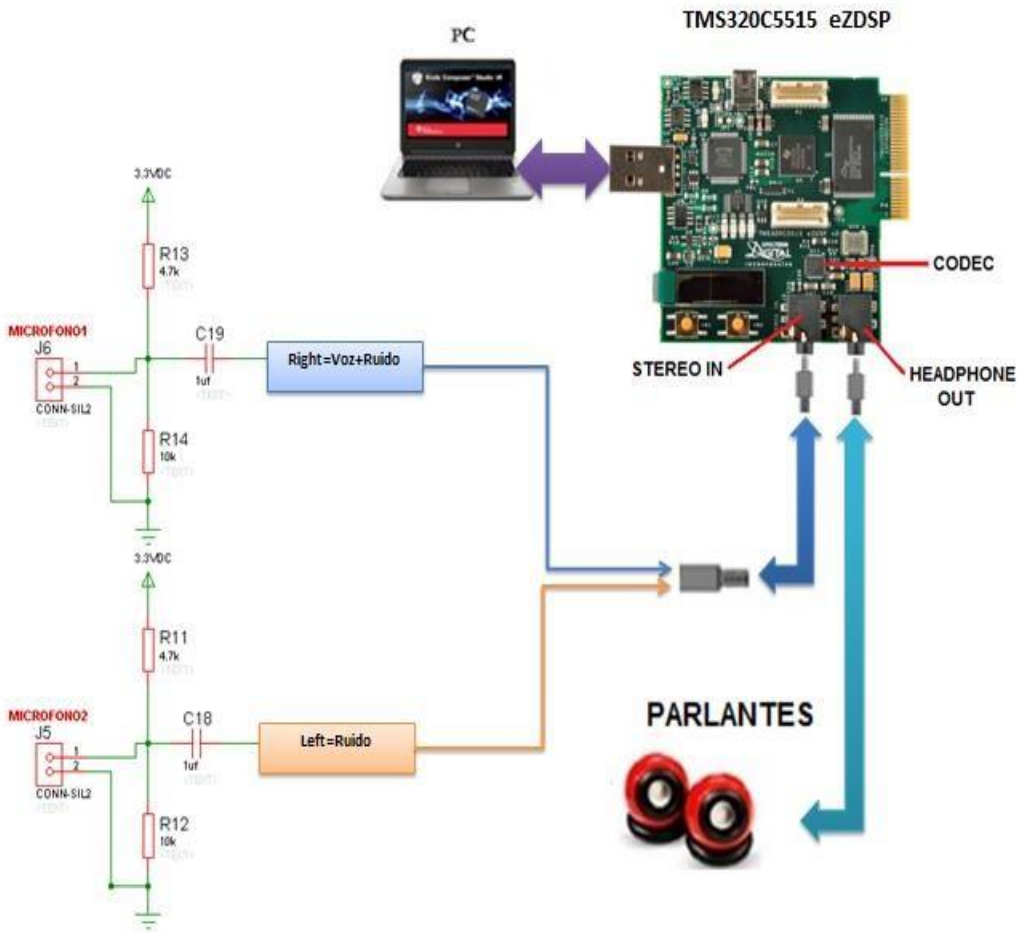


Figura 2.5 Esquema del Circuito del Sistema

Fuente: Autores.

2.8.2 Esquema de Circuito del Micrófono

Para la obtención de la señal contaminada (señal más ruido) y la señal de ruido pura se procede con el diseño de un circuito de dos micrófonos figura (2.6) alimentados por un divisor de voltaje (2.12) para controlar la cantidad de corriente en la entrada al dispositivo y con un capacitor de 1uf para acople del mismo, donde $V_{R,L}$ es la salida R o L del micrófono respectivamante.

$$V_{R,L} = \left(\frac{R14}{R13+R14}\right) 3.3VDC \tag{2.12}$$

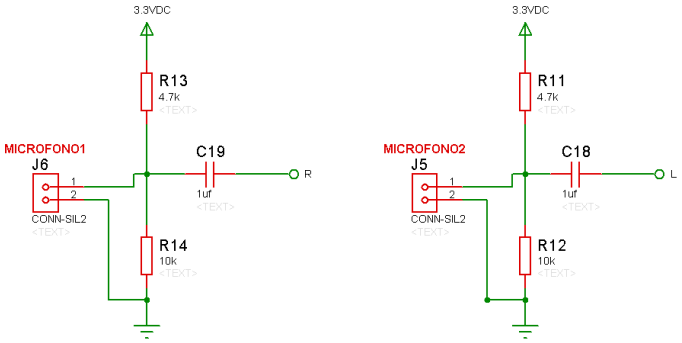


Figura 2.6 Diseño del circuito con dos micrófonos-Software Proteus

Fuente: Autores.

El circuito total de los micrófonos y la simulación en 3D de la placa realizado en el software ARES de PROTEUS, se puede apreciar en las figuras (2.7) y (2.8).

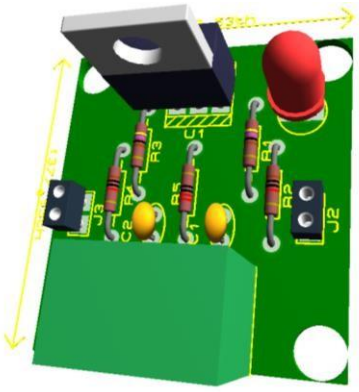


Figura 2.7 Circuito con dos micrófonos en 3D-Software Proteus

Fuente: Autores.

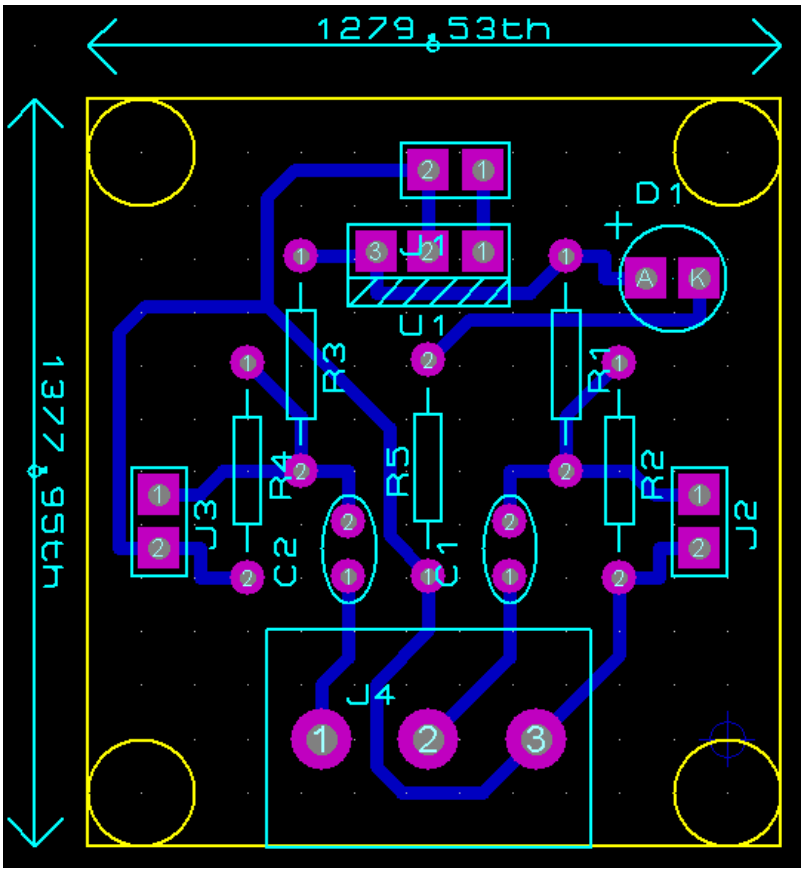


Figura 2.8 Diseño de circuito con dos micrófonos en ARES -Software Proteus

Fuente: Autores.

2.8.3 Descripción del código en el Software Code Composer Studio

El DSP C5515 ha sido programado de tal forma que está orientado a ejecutar diversos procesos para poder realizar funciones como: interfaz I2C, proceso simultáneo de datos y envío de información.

Todos estos procesos realizados de acuerdo a un orden preestablecido favorecen a la optimización del DSP. A continuación se estructura el proceso de operación a seguir, como se muestra en la figura (2.8):

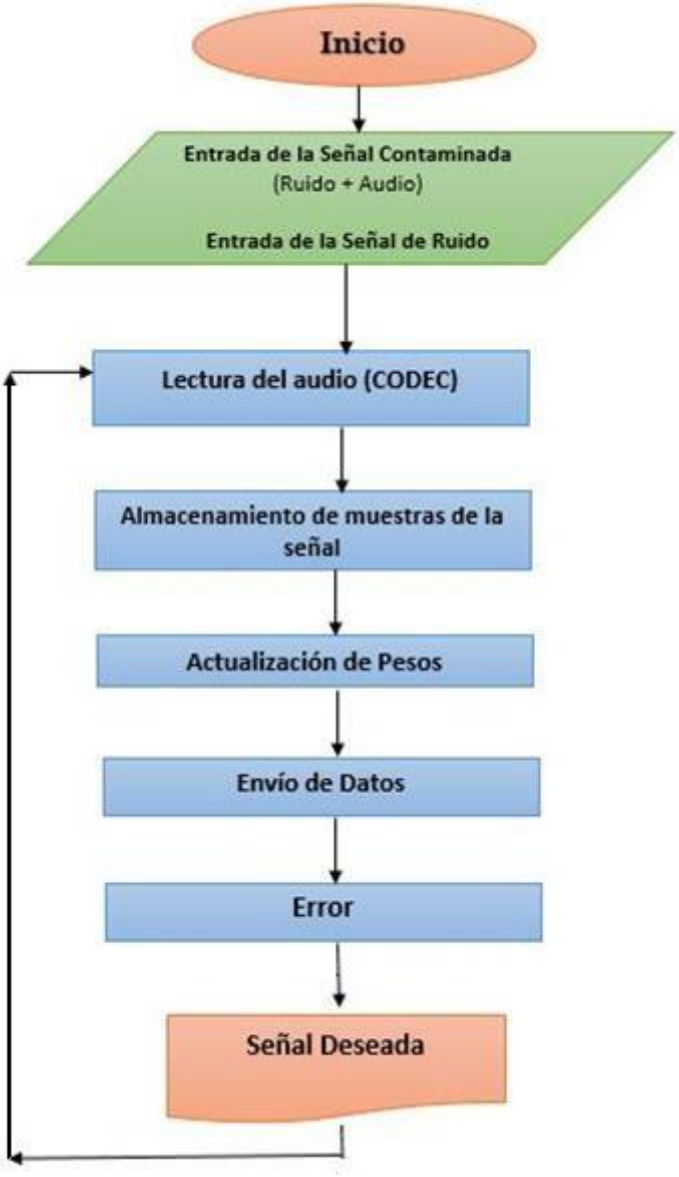


Figura 2.9 Diagrama funcional del proceso ejecutado por el DSP

Fuente: Autores

Inicialización del Programa

El programa inicia a través de la declaración de librerías, directivas, variables y comentarios en el DSP TMS320C5515, definen parámetros iniciales que comprende la puntualización de entradas y salidas.

Directivas para “incluir” otros ficheros (#include) y “definir” constantes y macros (#define).

#include Incluye el fichero, cuyo nombre se indica, para su compilación con el resto del código, como el nombre del fichero va entre <…> busca en el directorio include del sistema (/usr/include).

```
#include <usbstk5515.h>
#include <usbstk5515_i2c.h>
```

usbstk5515.h: Incluye la librería del procesador C5515.
usbstk5515_i2c.h: Incluye la librería del módulo I2C para la comunicación entre el DSP y CODEC.

Se incluyen en ese punto para ser compilados con nuestro fichero fuente.
El código siguiente muestra la dirección de memoria única asignada por el fabricante para TCR0 y TIMCN1_0.

```
#define TCR0      *((ioport volatile Uint16 *)0x1810)
#define TIMCNT1_0 *((ioport volatile Uint16 *)0x1814)
```

La declaración de las variables se realiza fuera del programa principal o función principal dada por la estructura:

```
void main(void){    }
```

Además el **void main(void)** es la función principal del programa que no espera un valor de retorno.

Se incorporan variables de tipo flotante para los pesos **(w0)** y entero de longitud de dos bytes **(Int16)** como **(me0)** y **(ru0)** para la mezcla y el ruido del audio

respectivamente, la declaración según su funcionalidad y condiciones iniciales vienen dados por las siguientes instrucciones:

```
Int16 me0,me1,me2,met,ru0,ru1,ru2,rut;Int16 at, error, YY, mezcla, vari;
Int16 out;
Int16 a0,a1,a2,at,error;
float w0,w1,w2,mult;
w0=0.7;          me0=0;          ru0=0;
w1=0.2;          me1=0;          ru1=0;
w2=0.2;          me2=0;          ru2=0;
```

La inicialización del DSP como de los otros módulos a utilizarse y la definición de los parámetros que comprende cada uno de ellos de manera individual, son representadas las siguientes líneas de programación:

```
USBSTK5515_init();    // USBSTK5515_init(); Inicializa el Procesador.
AIC_init();           // AIC_init(); Inicializa el Audio Codec.
```

Lectura de datos y almacenamiento

El AIC3204 es un códec de audio estéreo, que tiene como característica entrada y salida programable. Soporta la mezcla de señales de entrada analógica y digital. El dispositivo realizar operaciones, desde voz mono 8KHz hasta reproducciones estéreo a 192KHz, también una lectura de 16 bits equivalente a 65536 niveles ADC para un mejor desempeño en el proceso de la señal de audio. Conectando el plug estéreo de entrada a la tarjeta, donde ingresará la señal perturbada (audio y ruido) por la derecha y el ruido puro por la izquierda, las señales ingresan al Códec AIC3204 donde serán procesadas.

Lectura de la señal de entrada y almacenamiento se emplea con el siguiente código.

```
AIC_read2(&right, &left); // lectura del canal right y left de 16 bits
me0=right;     // asignación del valor de right a me0
ru0=left;      // asignación del valor de left a ru0
```

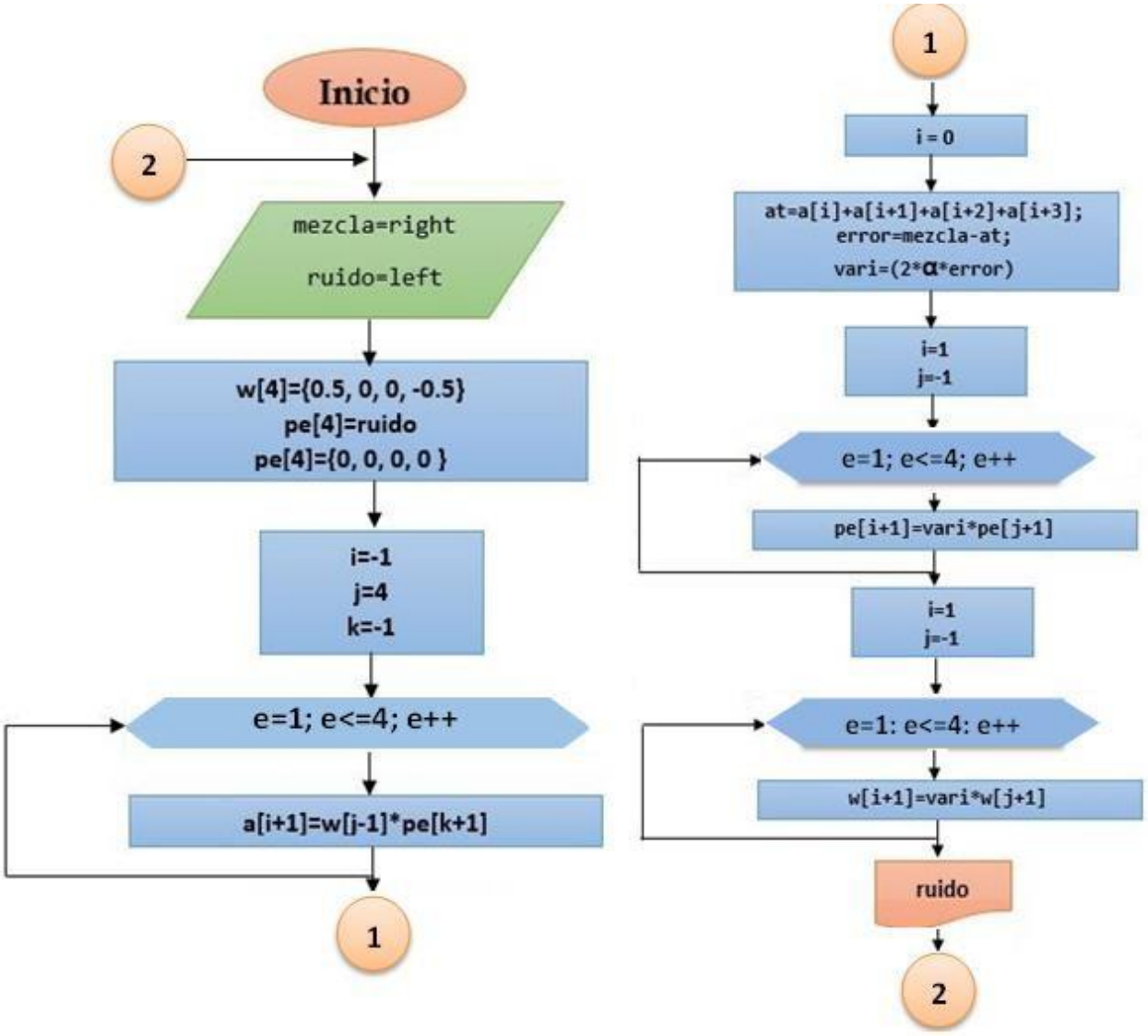


Figura 2.10 Diagrama de flujo del proceso del programa

Fuente: Autores

Salida de Datos

La salida de datos es en tiempo real, obtenemos en el headphone out a través de la conexión de parlantes.

```
AIC_write2 (error, error) // escritura o salida del DAC de 16 bits por el canal right y left.
```

Desarrollo del Programa Final

El programa final en el software Code Composer véase en el Anexo 3.

2.8.4 Pruebas con el Filtro Adaptativo RNA en diferentes tipos de ruido en tiempo real

2.8.4.1 Aprendizaje del algoritmo

Adaline

"En el algoritmo Widrow-Hoff Delta, los valores de los incrementos $\Delta W(k)$ y $\Delta b(k)$ se calcularon en base a las derivadas parciales de la función del error cuadrático medio con respecto a pesos y ganancias respectivamente".(Sotos, 2012) Después de haber evaluado las derivadas parciales el proceso de actualización puede expresarse como:

$$w(k+1)=w(k)+2\propto e(k)p(k) \quad (2.12)$$

$$b(k+1)=b(k)+2\propto e(k) \quad (2.13)$$

Se encuentran en forma independiente $w(k)$ y $b(k)$, la regla empleada de actualización de parámetros usada por Adaline, en su forma matricial el algoritmo de actualización de pesos y ganancia se muestra de la siguiente manera:

$$W(k+1)=W(k)+2\propto e(k)p(k) \quad (2.14)$$

$$b(k+1)=b(k)+2\propto e(k) \quad (2.15)$$

Estas ecuaciones (2.12) y (2.13) representadas en la programación como:

a[i+1]=w[j-1]*pe[k+1]

at=a[i]+a[i+1]+a[i+2]+a[i+3]

error=mezcla-at

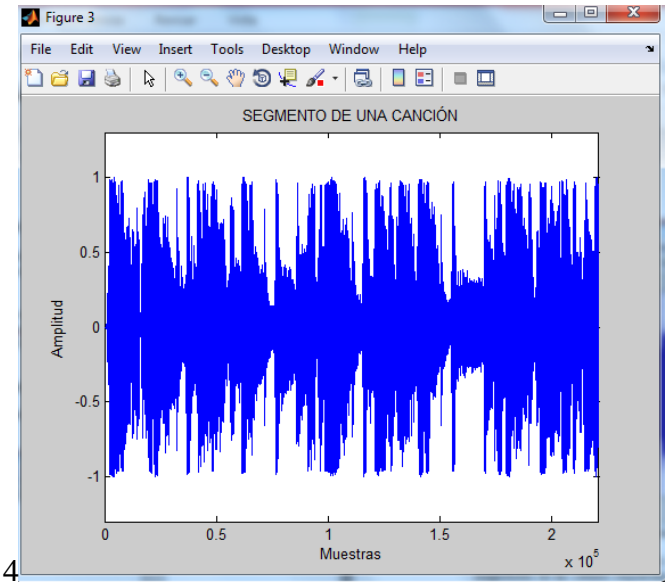
vari=(2*(0.0000000001)*error)

pe[i+1]=vari*pe[j+1]

2.8.4.2 Entorno de prueba Matlab®

Para el análisis del funcionamiento de la red neuronal se programa en el software Matlab® 7.10.0 para la visualización de los gráficos de la señal eliminada correspondientes a la salida.

1) En la entrada se observa la señal de un segmento de una canción, muestreada a 44100 samples/s durante siete segundos, por lo tanto el archivo digital contiene 221184 datos. Obsérvese la figura (2.11).



4

Figura 2.11 Segmento de la canción

Fuente: Autores.

1) Obsérvese la figura (2.12) se encuentra la señal de ruido con el mismo muestreo e igual cantidad de datos.

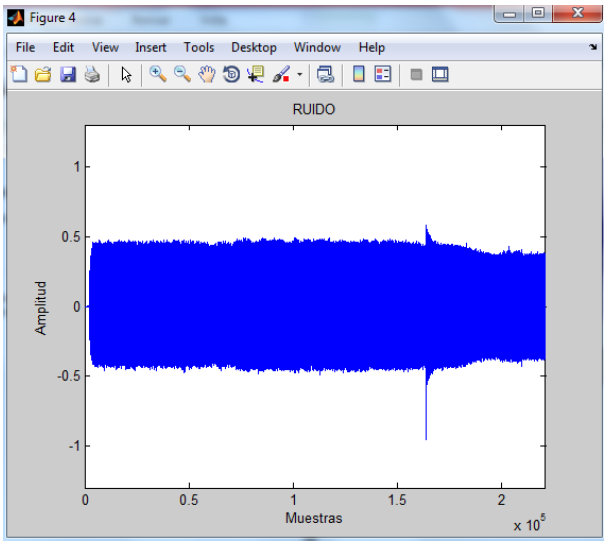


Figura 2.12 Señal de Ruido

Fuente: Autores

2) La señal mezclada es la suma de ambas gráficas (2.11) y (2.12) segmento de canción y ruido respectivamente, será la que ingrese a la red. Obsérvese la figura (2.13)

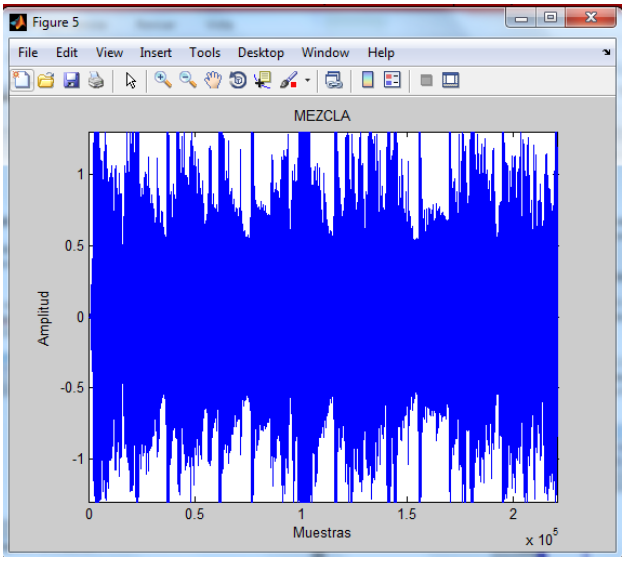


Figura 2.13 Mezcla de ambas señales

Fuente: Autores

Las anteriores figuras muestran la señal de muestreo en el tiempo vs las magnitudes de las señales requeridas.

3) En las siguientes figuras (2.14) y (2.15) el factor de aprendizaje es de 0.001con tres bloques de retraso. Los valores iniciales el vector de peso (W) es de [5; 0; -5] y para b fue 0, cantidades que fueron tomadas de manera aleatoria.

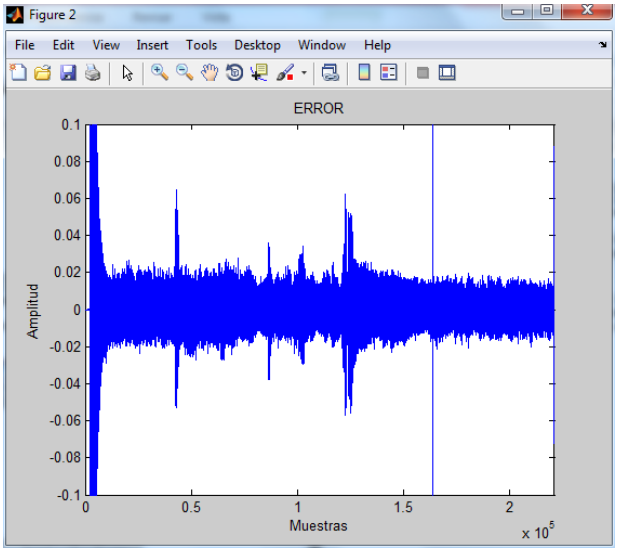


Figura 2.14 Señal eliminada de la red con α=0.001

Fuente: Autores.

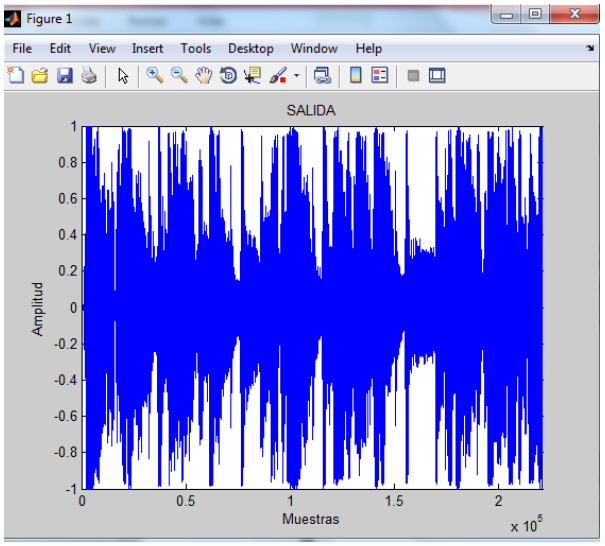


Figura 2.15 Salida de la red con α=0.001

Fuente: Autores.

La utilización de Matlab® es necesaria para evidenciar gráficamente cuanto de error se elimina en la señal deseada como se aprecia en la figura (2.14), el ruido es generado por una señal sinusoidal cuya frecuencia es de 2.5KHz. En el DSP no se puede observar la eliminación del error sin embargo su apreciación es de forma auditiva.

El programa final en el software Matlab véase en el Anexo 4.

2.8.4.3 Implementación en el DSP con redes neuronales

Para demostrar el funcionamiento debemos someter el filtro a diversas pruebas de sonidos, escogiendo los tres tipos frecuentados en estas aplicaciones, y que se describen a continuación:

Caso 1: Pista audio contaminado y el ruido (señal generada)

El audio contaminado se compone de una pista de música a la cual se le inyecta una señal de tipo sinusoidal, cuadrática o triangular, provocada intencionalmente por el generador de señales.

Caso 2: Pista audio contaminado y el ruido (Audio pregrabado)

El audio en esta prueba se compone de una señal de voz pregrabada a la cual se le inyecta una señal de ruido pregrabado.

Caso 3: Audio contaminado y el ruido (audio tiempo real)

El audio en esta prueba se compone de una señal de voz en tiempo real a la cual se le suma un ruido exterior en tiempo real proveniente de un ambiente externo como: salón de clase, autopistas, bares, sitios ruidosos, etc.

2.8.5 Comprobación de Hipótesis

Para la comprobar la hipótesis se utiliza el método estadístico CHI-CUADRADO, que accede a dos grados de posibilidad, alternativa la que se quiere comprobar y nula (rechaza la hipótesis alternativa) para constatar en cumplimiento de la hipótesis que fue:

2.8.5.1 Planteamiento de la hipótesis estadística

Hipótesis nula (Ho): La implementación de un filtro adaptativo con redes neuronales utilizando DSP no significa que eliminará los problemas de ruido en señales de audio.

Hipótesis alternativa (Hl): La implementación de un filtro adaptativo con redes neuronales utilizando DSP eliminará los problemas de ruido en señales de audio.

2.8.5.2 Establecimiento del nivel de significancia.

Las pruebas se realizaron con un 95% de confiabilidad, es decir, se trabajó con un nivel de significancia de α=0.05.

2.8.5.3 Determinación del valor estadístico de prueba.

Si el valor de CHI-CUADRADO es menor o igual que el CHI-CUADRADO critico entonces se acepta la hipótesis nula, caso contrario se la rechaza.

$$X^2 \leq \text{Valor Crítico}$$

Para aceptar o rechazar esta hipótesis se tomaron en cuenta 2 escenarios, un escenario A, que es la medición de datos del filtro cancelador de ruido y un escenario B, que es la medición de datos de un filtro cancelador de ruido con redes neuronales artificiales.

En la tabla (2) se muestran los valores obtenidos en cada uno de los escenarios en los cuales se realizaron las pruebas.

	RUIDO	ERROR	SALIDA	Total
ESCENARIO A	0,50869751	0	0,024017334	0,532714844
ESCENARIO B	0,50869751	-0,00313785	-0,028844572	0,476715088
TOTAL	1,01739502	-0,00313785	-0,004827238	1,009429932

Tabla 2 Variables obtenidas en las pruebas

Fuente: Autores

Para obtener las frecuencias esperadas se multiplica el total de cada columna, por el total de cada fila, y se divide entre el total de cada fila y columna, puede apreciarse en la tabla (3) y la tabla (4) representa valores críticos definidos por este método.

	RUIDO	ERROR	SALIDA
ESCENARIO A	0,536918326	-0,00165596	-0,002547519
ESCENARIO B	0,480476694	-0,00148189	-0,00227972
TOTAL	1,01739502	-0,00313785	-0,004827238

Tabla 3 Frecuencias Esperadas
Fuente: Autores

		PROBABILIDAD			
		0,995	**0,990**	**0,975**	**0,950**
GRADOS DE LIBERTAD	**1**	0,000	0,000	0,001	0,004
	2	0,010	0,020	0,051	0,103
	3	0,720	0,115	0,216	0,352
	4	0,207	0,297	0,484	0,711
	5	0,412	0,554	0,831	1,145
	6	0,676	0,872	1,237	1,635
	7	0,989	1,239	1,690	2,167
	8	1,344	1,646	2,180	2,733
	9	1,735	2,088	2,700	3,325
	10	2,156	2,558	3,247	3,940
	11	2,603	3,053	3,816	4,575
	12	3,074	3,571	4,404	5,226

Tabla 4 Valores Críticos Método Chi-Cuadrado
Fuente: Autores

La tabla (5) muestra los valores obtenidos para nuestra comprobación de hipótesis a través del método CHI-CUADRADO.

# de filas	2
# de columnas	3
X^2=	0,26327114
Grados de Libertad	2
Nivel de significación	0,05
Probabilidad	0,95

VALOR CRITICO	0.103

Tabla 5 Resultados del método estadístico del CHI-CUADRADO
Fuente: Autores

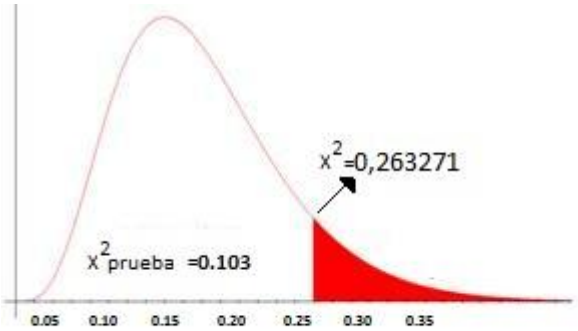


Figura 2.16 Resultado de comparación χ^2_{prueba} con el χ^2_{tabla} método del Chi-Cuadrado
Fuente: Autores

En la figura (2.16) se muestra la prueba chi-cuadrado, requiere la comparación del Valor Crítico (X^2tabla) con X^2(X^2prueba).

De acuerdo con el resultado se obtiene que X^2 es mayor que el valor crítico lo cual lleva a rechazar la hipótesis nula y aceptar la hipótesis alternativa, es decir:
"Hl: La implementación de un filtro adaptativo con redes neuronales utilizando DSP eliminará los problemas de ruido en señales de audio".

CAPITULO III

3. RESULTADOS

Los resultados obtenidos por medio de las pruebas efectuadas en tres ambientes diferentes permiten discriminar el filtro según su rendimiento un ejemplo muy notorio es el de la figura (3.1).

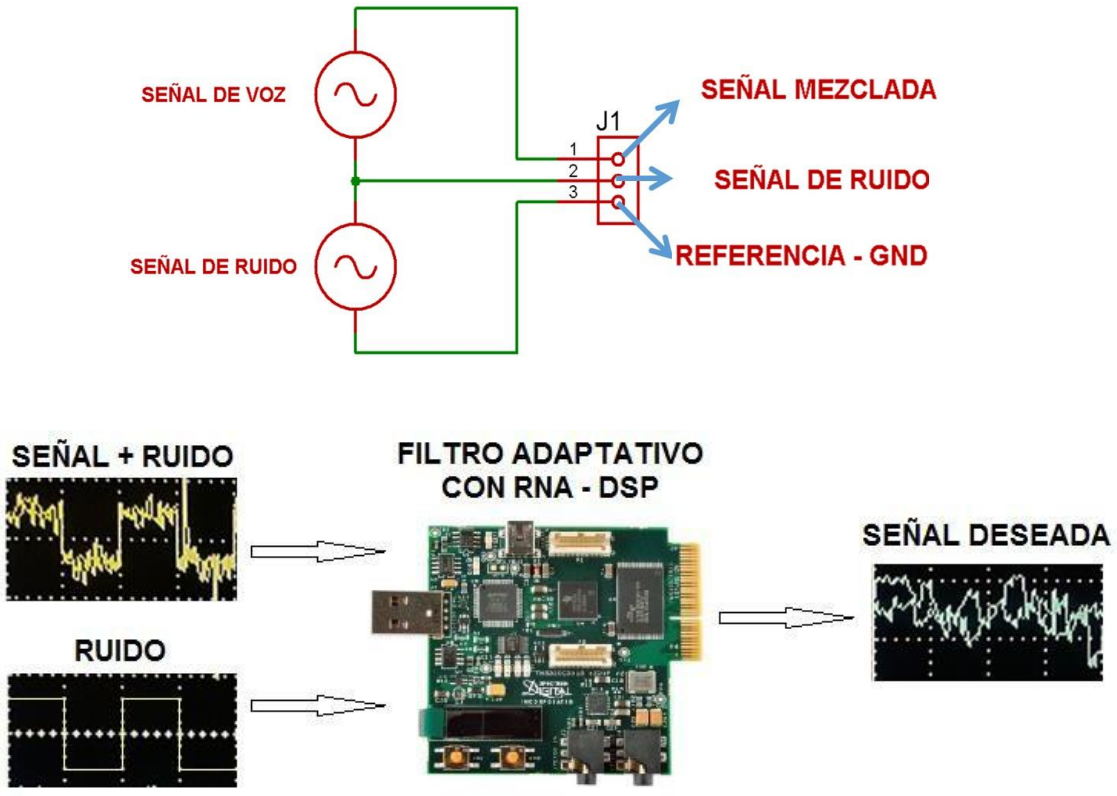


Figura 3.1 Suma de señales de entrada y salida en el DSP
Fuente: Autores.

Caso 1: Pista audio contaminado y el ruido (señal generada)

Para las pruebas se utiliza tres tipos de señales que son: sinusoidal, cuadrada y triangular producidas por un equipo generador de señales, que sumadas a la señal deseada (pista musical), produce un sonido molesto para el usuario, al que le interesa únicamente la señal deseada.

La variación de frecuencia de las señales no tiene importancia ya que el filtro es adaptativo, para la prueba en el osciloscopio se emite una frecuencia a **1KHz** que se encuentra en el rango de frecuencias audibles con una amplitud **1V** pico a pico.
La señal deseada o salida aplicando el filtro adaptativo con redes neuronales se puede apreciar en la figura (3.2), figura (3.3), figura (3.4), y se observa en el channel 2 la salida del filtro adaptativo libre de la señal inyectada (ruido).

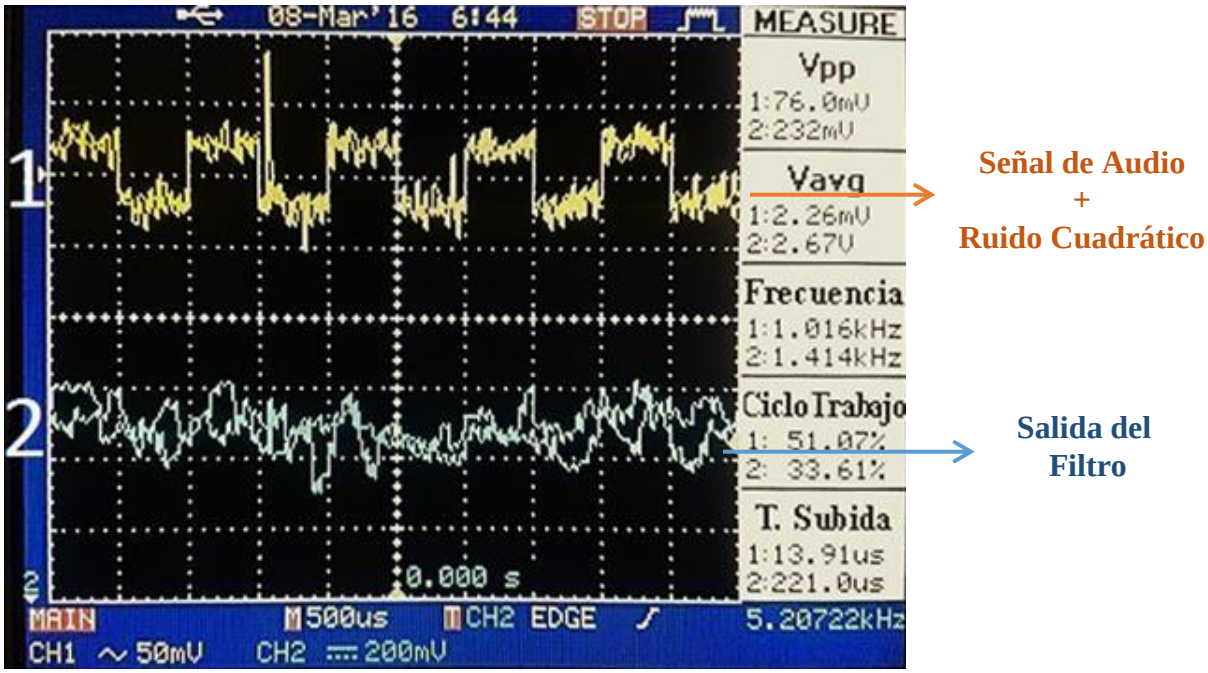


Figura 3.2 Audio sumada a una Señal Cuadrática (CH1) y Salida del filtro (CH2).
Fuente: Autores.

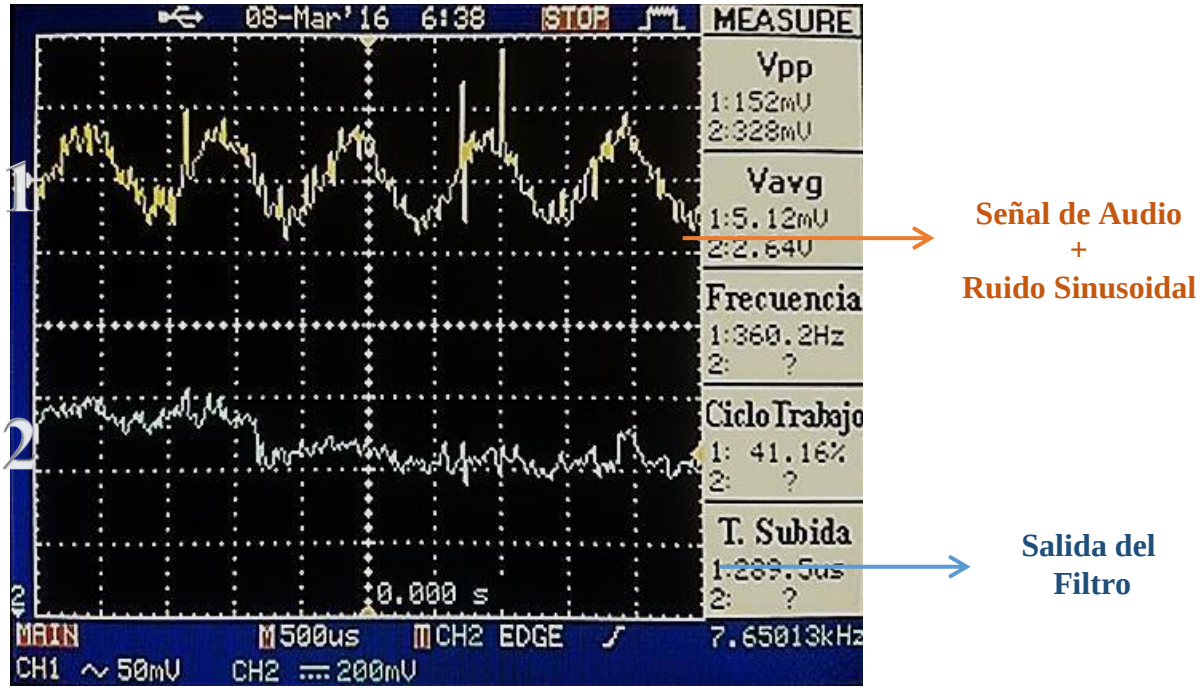


Figura 3.3 Audio Contaminado con una Señal Sinusoidal (CH1) y Salida del filtro (CH2).
Fuente: Autores.

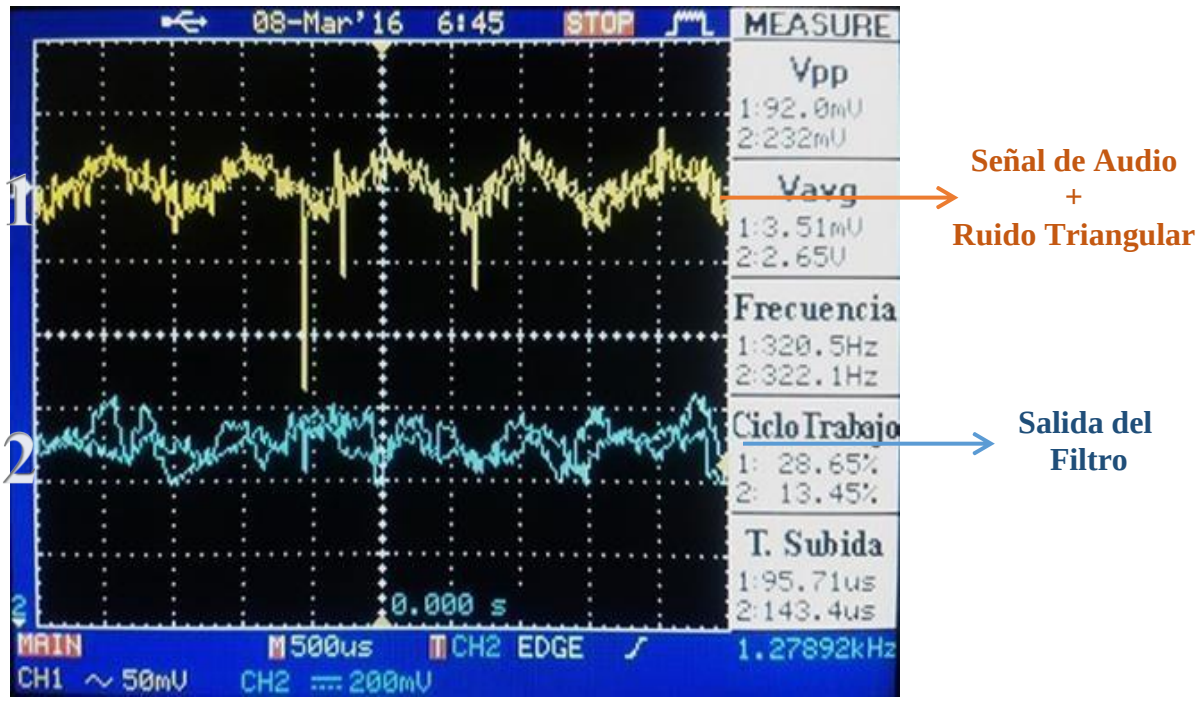


Figura 3.4 Audio Contaminado con una Señal Triangular (CH1) y Salida del filtro (CH2).
Fuente: Autores.

Caso 2: Pista audio contaminado y el ruido (Audio pregrabado)

El audio pregrabado de la señal deseada (pista musical) y el audio pregrabado considerado como ruido (ruido ambiental), son ingresados al circuito sumador, aplicando el filtro adaptativo con redes neuronales, se obtiene como respuesta en el osciloscopio una señal de entrada y salida similar, debido a que las señales son audios pregrabados y visualmente tienen la misma forma (como una señal de ruido eléctrico), para un determinado tiempo de audio se representa en el Software Matlab® véase la figura (2.12) y figura (2.14) que representa la señal sumada y la señal de salida del filtro respectivamente. Teniendo en cuenta que utilizando el DSP y realizando la prueba mencionada el filtro funciona perfectamente de forma audible.

Caso 3: Audio contaminado y el ruido (audio tiempo real)

El audio de la señal de voz en tiempo real y el audio inyectado en tiempo real es considerado como ruido proveniente de un ambiente externo ya sea un salón de clase, autopistas, bares o sitios ruidosos, son ingresados al circuito sumador, aplicando el filtro adaptativo con redes neuronales, se obtiene como respuesta en el osciloscopio la cancelación total el ruido (ambiente externo), véase la figura (3.5) donde se observa en el channel 2 la salida del filtro adaptativo libre de la señal inyectada (ruido) y el channel 1 la suma de la señal de voz en tiempo real y el ruido.

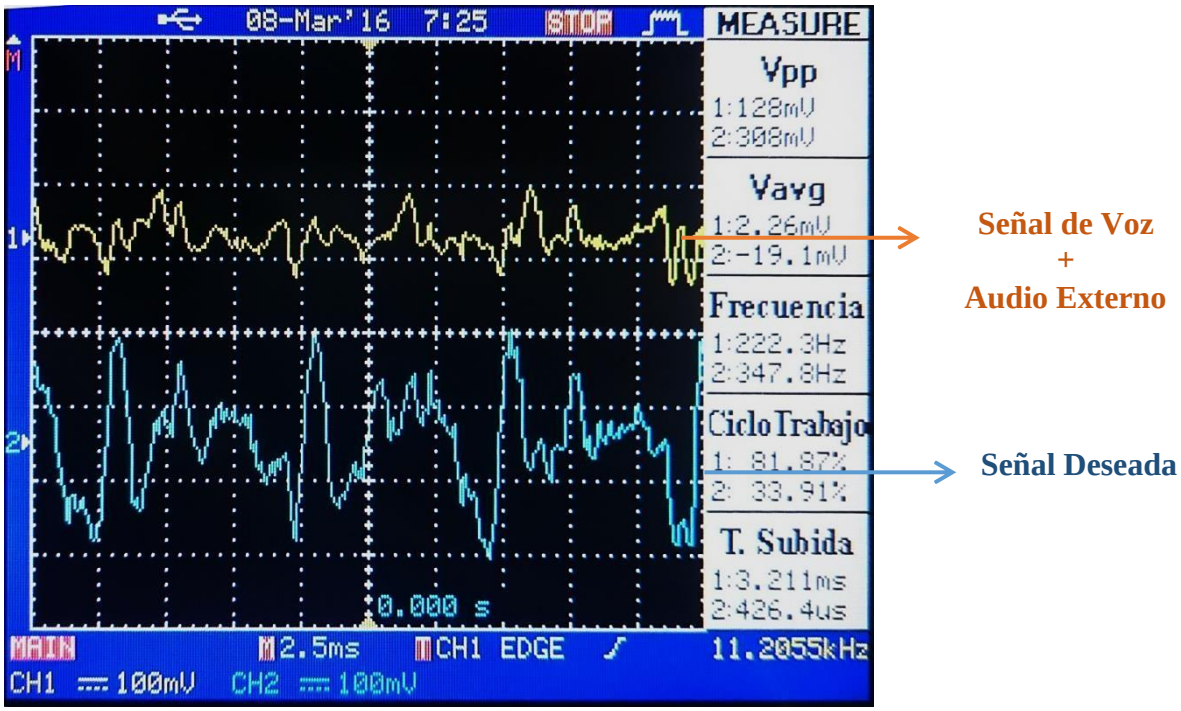


Figura 3.5 *Salida del filtro (CH1) Suma de Señales en tiempo real (CH2).*
Fuente: Autores.

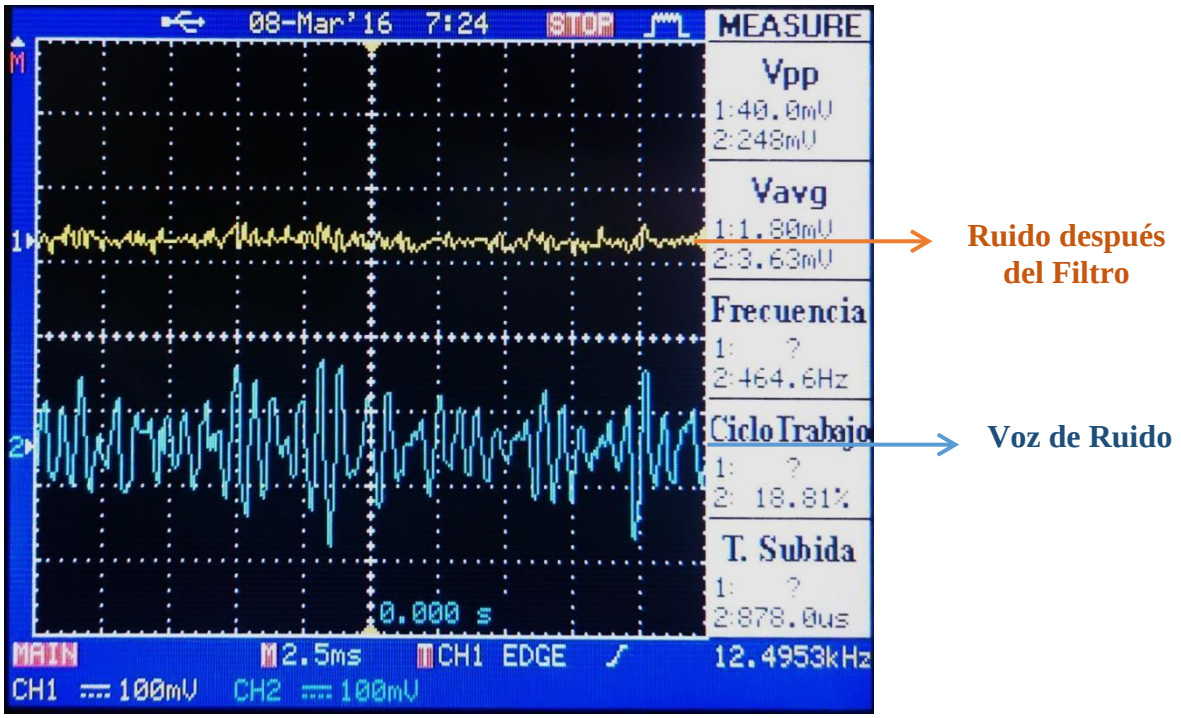


Figura 3.6 *Anulación del ruido (CH1) Ruido Puro de micrófono (CH2).*
Fuente: Autores.

CAPITULO IV

4. DISCUSIÓN

El filtro adaptativo con redes neuronas artificiales es un dispositivo capaz de eliminar el ruido producido por diferentes tipos de ambientes como el de áreas recreativas, autopistas, etc. Usualmente empleado en la telefonía, el propósito general es que el abonado pueda escuchar perfectamente y sin problemas la comunicación, brindando al cliente calidad de servicio.

El sistema consta de un filtro, un algoritmo adaptativo y el método de aprendizaje, que agrupados forman un cancelador de ruido, cuya capacidad es adaptarse a cualquier frecuencia de entrada

Imagínese una vida donde la señal de radio se escuche perfectamente sin interferencias y en la comunicación telefónica se perciba únicamente la voz de la persona que se encuentra al otro lado de línea, pues esto cada vez se va haciendo realidad con la ayuda de los filtros que hoy en día se elaboran gracias al estudio de procesamiento de señales (DSP), con diversas estrategias que permiten en algunos casos intuir su respuesta, aprendizaje de errores, orden de la frecuencia de corte, etc, que caracterizan a un tipo de filtro para diferentes aplicaciones.

La tecnología se ha incrementado en las aplicaciones en el tratamiento de señales aumentando la velocidad de los microcontroladores como en este caso el de un DSP, los filtros se encuentran en todos los dispositivos de comunicación que utilizamos, como por ejemplo el celular, debido a que siempre está expuesto a entornos de ruidos ambientales tornándose incómodos, también la psicofonía, ecocardiografía fetal.Finalmente en las telecomunicaciones la cancelación de ruido es de vital importancia en la transmisión de información.

CAPITULO V

5. Conclusiones y Recomendaciones

5.1 Conclusiones

- El filtro con redes neuronales artificiales cancela significativamente el ruido inyectado por diferentes tipos de fuentes como ambientes externos ya sean salones de clase, autopistas, bares, sitios ruidosos, etc.

- El tiempo de adaptabilidad del filtro con redes neuronales depende de la razón de aprendizaje (α) y a su vez la razón de aprendizaje depende del número de bits de lectura de datos, si la (α) es mayor 0.000000001 la neurona aprende rápido pero el sistema es inestable, es decir los valores de los pesos (w) varían desde los décimos de la parte decimal del número , si la (α) es menor a 0.0000000001 la neurona se demora un poco en aprender pero su filtrado es óptimo es ya que los pesos (w) varían en los diezmilésimos de la parte decimal del número y son valores semejantes a los requeridos para la eliminación del ruido.

- Para un filtro en tiempo real los bloques de retardo afectan directamente a la calidad de audio debido a la velocidad de procesamiento del DSP, ya que al trabajar con más de tres bloques de retardo pierde la calidad en el audio.

- Para la comprobación de los diferentes tipos de pruebas la velocidad de procesamiento del DSP TMS320C5515 eZDSP USB Stick demostró ser capaz de utilizar tres bloques de retraso sin perjudicar la señal.

5.2 Recomendaciones

- El factor de aprendizaje (α) debe ser un valor tan pequeño que pueda modificar a los pesos en la cifra diezmilésima de un número decimal, ya que mientras más alto es, se torna oscilante perdiendo la estabilidad y no logra filtrar el ruido, mientras que si resulta ser muy bajo el proceso es más lento para alcanzar la convergencia.

- Los pesos iniciales (w) son aleatorios, pero si son muy altos la neurona tarda en aprender, coloque valores similares a (0.7, 0.2,-0.5) obtenidas en las pruebas realizadas en el software Matlab.

- Las variables de los pesos (w) deben ser de tipo float, ya que los valores son decimales y se encuentran mayormente entre 0.1- 0.999.

- Para la conexión de los micrófonos de entrada se sugiere alimentarlos por medio de un regulador de voltaje de 3.3 voltios ya que se puede averiar el códec del DSP por sobre picos de voltaje o corriente.

- Para la realizar la suma colocar primero la señal del generador de ondas seguido del audio, ya que si se coloca de manera inversa el desacoplamiento elimina la señal de audio, esto sucede en forma visual, de forma audible el filtro funciona normal.

CAPITULO VI

6. PROPUESTA

6.1 Título de la propuesta

DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO PARA LA CANCELACIÓN DE RUIDO CON REDES NEURONALES UTILIZANDO DSP.

6.2 Introducción

El ruido es una perturbación no deseada que aparece en un sistema de comunicaciones, sin embargo no tiene relación alguna con la señal original, de manera que se debe hacer un tratamiento de filtrado adicional. La búsqueda de técnicas para el procesamiento estadístico de señales con diversos métodos de predicción de señal o patrones dinámicos con la finalidad de mejorar la transmisión de información. Los procesos usados como linealidad, sistema estacionario y estadístico de segundo orden se tornan no suficientes a razón que no todas las señales físicas en tiempo real son generadas por procesos dinámicos que a la vez no son lineales, no estacionarios y no estadístico por lo que resulta no ser óptimo el procesamiento de la señal.

Un método para optimizar el sistema es mediante el uso de un algoritmo adaptativo con adaline usando DSP para la cancelación del ruido en tiempo real.
Los filtros adaptativos con redes neuronales se enfocan a una cancelación del ruido al máximo, ya que este sólo maneja el conjunto señal deseada más ruido, esta opción permite que el algoritmo aprenda de la señal de error, obteniendo a su salida una respuesta más eficiente, manejable y óptima. Adaline es una red de aprendizaje

que supervisada usa el algoritmo de aprendizaje de Widrow-Hoff Delta que reduce el error cuadrático medio, esto ocurre cuando se da el proceso de actualización de pesos y ganancias.

La gran ventaja del sistema radica que todo el proceso se da en tiempo real, la rapidez de convergencia es notable gracias al filtro adaptativo con red adaline que da tratamiento a la señal de ruido, la velocidad de procesamiento del DSP contribuye a la respuesta rápida del sistema su apreciación se da en forma audible, lo que da como respuesta un sistema de mayor eficiencia.

6.3 Objetivos

6.3.1 Objetivo General

- Diseñar e implementar un filtro adaptativo para la cancelación de ruido con redes neuronales utilizando DSP.

6.3.2 Objetivos Específicos

- Investigar los modelos de redes neuronales para la cancelación de ruido en filtros adaptativos.
- Seleccionar el modelo adecuado para el algoritmo.
- Desarrollar el modelo con redes neuronales para el filtro adaptativo.
- Desarrollar el sistema en lazo cerrado del filtro adaptativo para el DSP.
- Verificar el filtro adaptativo sometido a diferentes tipos de ruido en tiempo real.

6.4 Fundamentación Científico-Técnico

El sistema de un filtro adaptativo con redes neuronales utilizando DSP, tiene una significativa representación en la transmisión de información, encaminada al procesamiento de audio, principalmente a la telefonía, circuitos de comunicación, generadores electrónicos de música, limitándose a la cancelación de ruido en tiempo real mediante la capacidad de adaptabilidad aprovechando la velocidad del procesador.

6.5 Descripción de la propuesta

Este proyecto consiste en la cancelación de ruido con filtros adaptativos con redes neuronales, la fuente de información es emitida por el bloque del circuito de los micrófonos mediante la programación se procesa los datos de audio donde se basa en la obtención de muestras pasadas y la actual de la señal de ruido, entra al códec de audio estéreo AIC3204, se realiza el tratamiento de la señal de ruido, la información es almacenada y a través de la actualización de pesos el sistema adapta a la señal de manera que a su salida está libre de ruido, la mejor manera de apreciar este proceso es de forma audible, su cambio es considerable.

El sistema debe ser capaz de adaptarse a muy alta velocidad, debido a que es en tiempo real.

6.6 Diseño organizacional

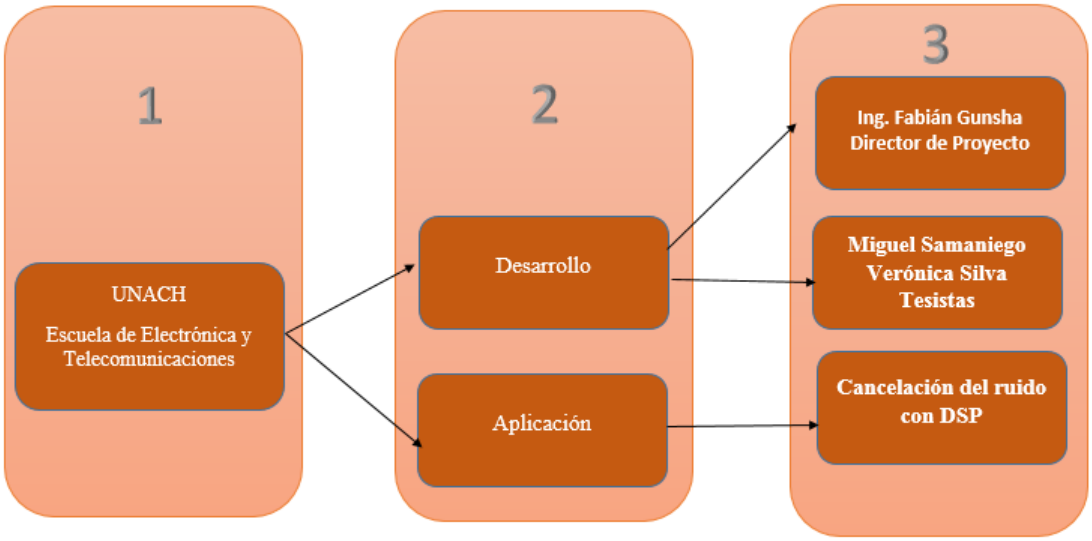


Figura 6.1 Esquema organizacional

Fuente: Autores.

6.7 Monitoreo y Evaluación de la propuesta

La evaluación de la propuesta se realizará a través de pruebas, sometiendo a la señal de audio a diferentes tipos de ruido.

El impacto que produce la implementación de un filtro adaptativo con redes neuronales es de gran beneficio, al lograr la eliminación del ruido en la transmisión de información especialmente en la telefonía, cuando es perjudicada por un algún tipo de ruido.

7. BIBLIOGRAFÍA

[1] F. E. H. Montero, «Cancelación de Ruido a través de Técnicas Neuronales,» Brazil, 1999.

[2] K. S. Lin, «Digital Signal Processing Applications with the TMS320 Family V.1 Prentice Hall and Texas Instruments Digital Signal Processing Series,» de *Digital Signal Processing Applications with the TMS320 Family V.1 Prentice Hall and Texas Instruments Digital Signal Processing Series*, Prentice-Hall, 1987.

[3] J. M. Sotos, «Aplicación de Redes Neuronaes Artificiales en el procesado versátil de señales Electrocardiográficas,» Valencia, 2012.

[4] M. Marcelino, FILTROS DIGITALES, España, 2009-2010.

[5] Santiago A, Anzueto Á, «FILTRO ADAPTABLE USANDO UNA RED NEURONAL DINÁMICA PARA CANCELACIÓN DE RUIDO,» (01 de 09 de 2015), Obtenido de upiita: http://www.boletin.upiita.ipn.mx/index.php/ciencia/629-cyt-numero-50/1166-filtro-adaptable-usando-una-red-neuronal-dinamica-para-cancelacion-de-ruido

[6] H. Mark, NEURONAL NETWORK TOOLBOX TM 7, United States: COPYRIGHT, 2010.

[7] T. INSTRUMENTS, Code Composer Studio™ v6.1, United States, August 2015.

[8] T. INSTRUMENTS, TMS320C5515 DSP System - User's Guide, United States, May 2014.

[9] J. Hernández, FILTROS ADAPTATIVOS, Sevilla, 2008.

[10] D. Matich, Redes Neuronales: Conceptos Básicos y Aplicaciones, Rosario, 2001.

8. ENLACES WEB

[1] http://www2.dis.ulpgc.es/~li-pso/tema5/tema5.htm
[2] http://www.dspace.espol.edu.ec/bitstream/123456789/20043/22/Algoritmos_Adaptativos
[3] http://www.dspace.espol.edu.ec
[4] ftp://ftp.udistrital.edu.co/Documentacion/Electronica/Dsp/capitulo5.PDF
[5] http://gtas.unican.es/files/docencia/TDS/apuntes/tema6wp.pdf
[6] http://www.ti.com/lit/ds/slos602c/slos602c.pdf
[7] http://www.dariolara.com/tda/tds/RNA.pdf
[8] http://ocw.uv.es/ingenieria-y-arquitectura/filtrosdigitales/tema_5_realización_de_ de_sistemas_en_tiempo_discreto.pdf
[9] http://www.thedigitalmap.com/~carlos/p51_94/infav4
[10] http://konocimientointeligenciaartificial.blogspot.com/
[11] http://biologiacelular.com.ar
[12] http://bibing.us.es/proyectos/abreproy/11284/fichero/Volumen+1%252FCap%EDtulo+2

9. ANEXOS

ANEXO 1

DATASHEET

TMS320C5515 eZDSP

USB Stick

1.0 Overview of the C5515 eZdsp USB Stick

The C5515 eZdsp USB Stick is an evaluation tool for the Texas Instruments TMS320C5515 Digital Signal Processor (DSP). This USB bus powered tool allows the user to evaluate the following items:

- The TMS320C5515 processor along with its peripherals
- The TLV320AIC3204 codec
- The Code Composer Studio IDE™ software development tools

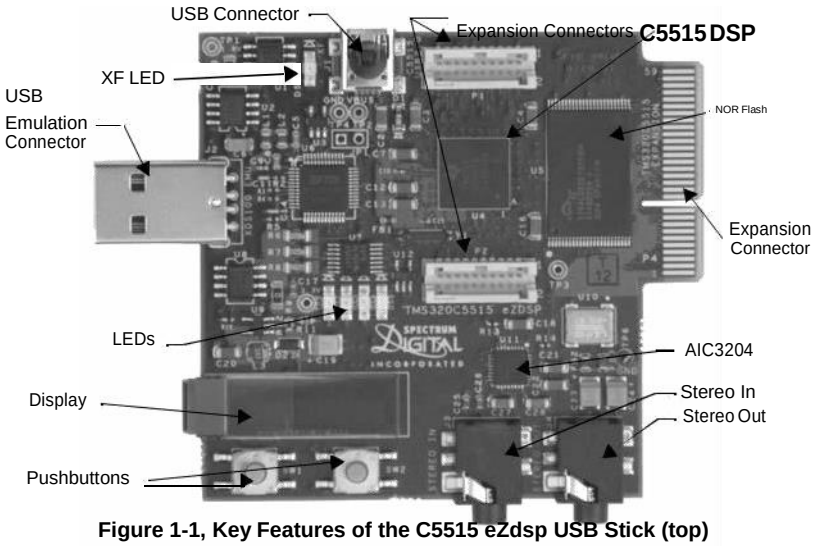


Figure 1-1, Key Features of the C5515 eZdsp USB Stick (top)

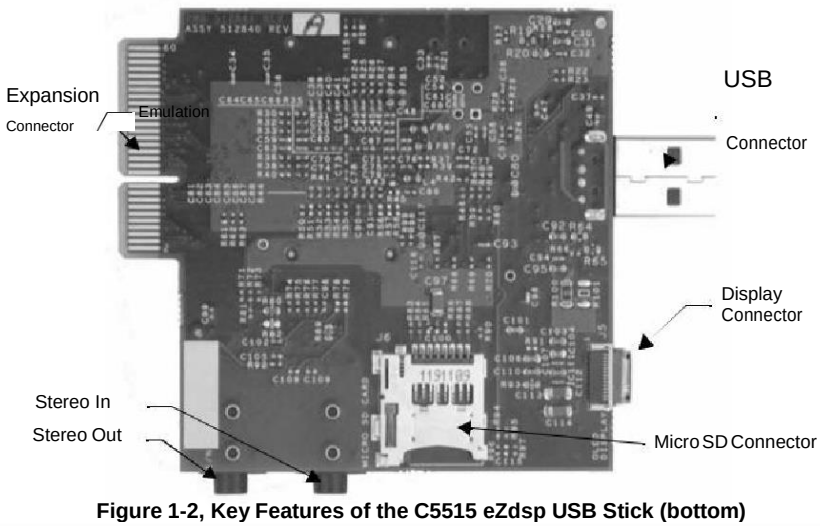


Figure 1-2, Key Features of the C5515 eZdsp USB Stick (bottom)

1.1 Key Features of the C5515 eZdsp USB Stick

The C5515 eZdsp USB Stick has the following features:

- Texas Instrument's TMS320C5515 Digital Signal Processor
- Texas Instruments TLV320AIC3204 Stereo Codec (stereo in, stereo out)
- Micro SD connector
- USB 2.0 interface to C5515 processor
- 32 Mb NOR flash
- I^2C OLED display
- 5 user controlled LEDs
- 2 user readable push button switches
- Embedded USB XDS100 JTAG emulator
- Bluetooth board interface
- Expansion edge connector
- Power provided by USB interface
- Compatible with Texas Instruments Code Composer Studio v4
- USB extension cable

1.2 C5515 eZdsp USB Stick Block Diagram

The block diagram of the C5515 eZdsp USB Stick is shown below.

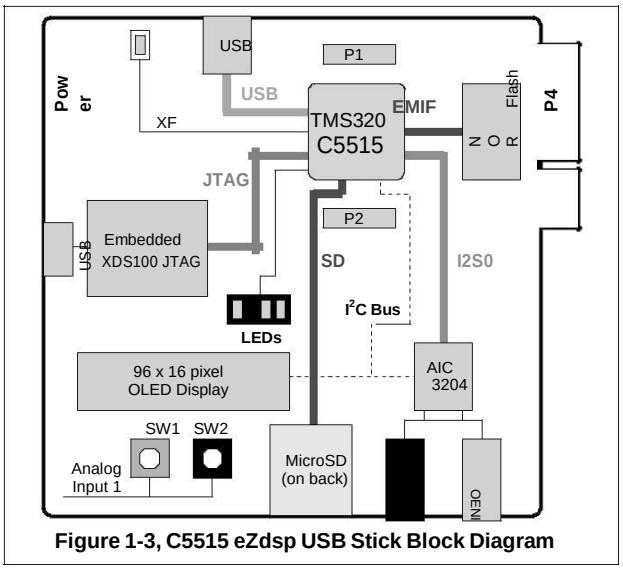


Figure 1-3, C5515 eZdsp USB Stick Block Diagram

1.3 C5515 eZdsp Memory Map

The C5515 eZdsp supports on chip DARAM, and off chip NOR Flash. The addressing for each of these memory blocks is shown in the figure below.

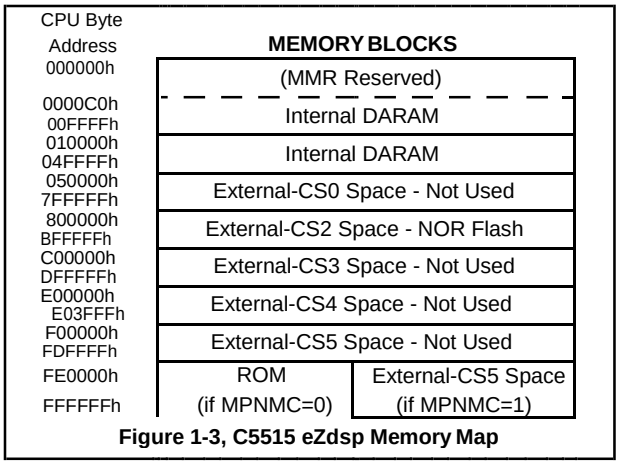


Figure 1-3, C5515 eZdsp Memory Map

Note: MPNMC bit in ST3 Status Register is cleared(0) at RESET so the C5515 will attempt to execute its boot load sequence.

1.4 C5515 eZdsp I^2C Addressing

The C5515 eZdsp has multiple I^2C devices for different purposes. The table below shows the addresses of these devices on the I^2C bus.

Table 1: C5515 eZdsp I^2C Addresses

eZdsp I^2C Device	I^2C Address	Function
TLV320AIC3204	0x18	Audio CODEC
OSD9616GLBBG01	0x3C	OLED Display

2.0 Board Layout

The C5515 eZdsp USB Stick is a 2.85 x 2.65 inch six (6) layer printed circuit board which is powered off the USB bus of personal computer or laptop computer. This means this board does not require an external power supply.

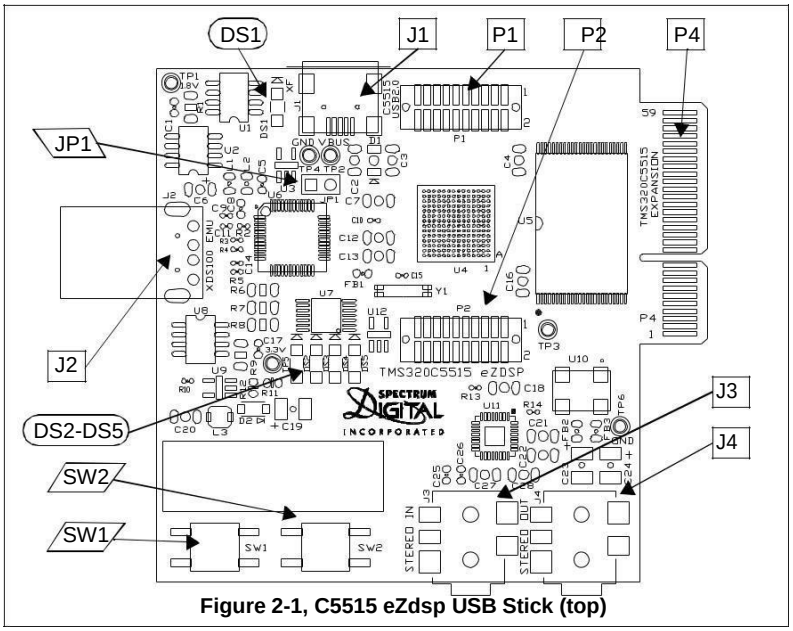


Figure 2-1, C5515 eZdsp USB Stick (top)

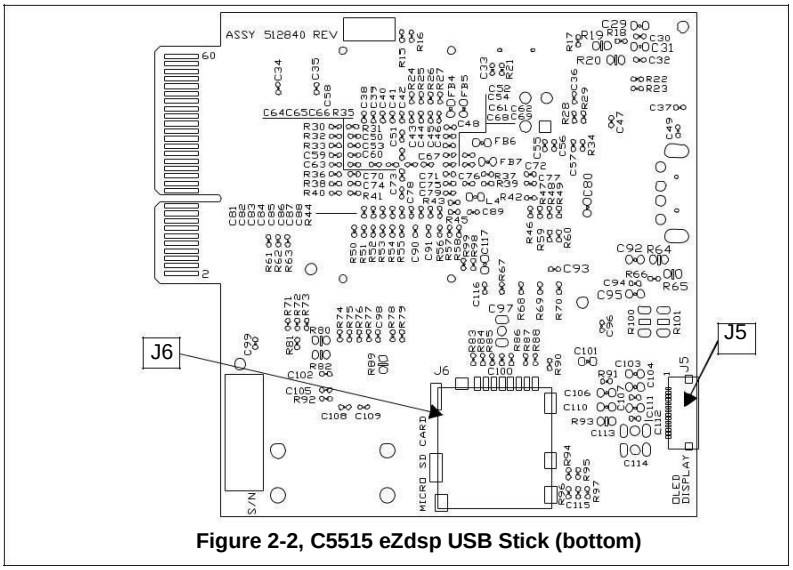


Figure 2-2, C5515 eZdsp USB Stick (bottom)

2.1 Connector Index

The C5515 eZdsp USB Stick has nine (9) connectors which provide the user access to various signals on the C5515 Stick. These connectors are shown in the table below.

Table 1: C5515 eZdsp USB Stick Connectors

Connector	# Pins	Function	Schematic Page	Board Side
J1	4	C5515 USB	3	Top
J2	6	Emulation USB	13	Top
J3	2	Audio In	10	Top
J4	2	Audio Out	10	Top
J5	14	LCD Interface	11	Top
J6	8	Micro SD Interface	8	Top
P1	20	Bluetooth Board Interface	7	Top
P2	20	Bluetooth Board Interface	7	Top
P4	30 x 2	Expansion	12	Top/Bottom

The following manufacturer and parts numbers can be used to interface to the connectors on the C5515 eZdsp:

Table 2: C5515 eZdsp Mating Connectors

Connector	Manufacturer	Part #
J2	PC or laptop	
J3	CUI Inc	CUI SP-3501, Digi-Key CP-3502-ND
J4	CUI Inc	CUI SP-3501, Digi-Key CP-3502-ND
P4	Samtec	Samtec MEC1-130-02-S-D-A, Digi-Key SAM8118-ND

2.1.1 J1, C5515 USB Connector

The USB connector, J1, is attached the C5515 processor for use by C5515 software applications. The signals on the pins of this connector are shown below.

Table 3: J1, C5515 USB Connector

Pin #	Signal Name
1	USBVDD
2	D-
3	D+
4	ID / NC
5	USBVSS/GND
6-9	GND

2.1.2 J2, XDS100 USB Connector

The USB connector, J2, is used to attach the C5515 eZdsp stick to a personal computer or laptop. The signals on the pins of this connector are shown below.

Table 4: J2, XDS100 USB Connector

Pin #	Signal Name
1	5V_USB
2	D+
3	D-
4	GND
5	Shield Ground
6	Shield Ground

2.1.3 J3, Audio In Connector

The Audio In connector, J3, is used to bring signals into the TLP320AIC3204 codec. The signals on the pins of this connector are shown below.

Table 5: J3, Audio In Connector

Pin #	Signal Name	AIC3204 Pin #
1	GND-AIC	
2	AIC_LINE2L	15
3	AIC_LINE2R	16
4	No connect	
5	No connect	

The figure below shows a typical stereo jack.

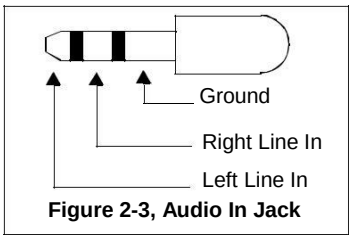


Figure 2-3, Audio In Jack

2.1.4 J4, Audio Out Connector

The Audio Out connector, J4, is used to bring signals from the TLP320AIC3204 codec. The signals on the pins of this connector are shown below.

Table 6: J4, Audio Out Connector

Pin #	Signal Name	AIC3204 Pin #
1	GND-AIC	
2	HEADPHONE_LOUT	25
3	HEADPHONE_ROUT	27
4	No connect	
5	No connect	

The figure below shows a typical audio out jack.

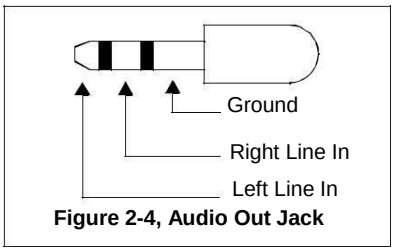


Figure 2-4, Audio Out Jack

2.1.5 J5, LCD Interface

Connector, J5, is used to interface to an LCD character display. The signals on the pins of this connector are shown below.

Table 7: J5, LCD Interface

Pin #	Signal Name
1	C2P
2	C2N
3	C1P
4	C1N
5	VBAT
6	VBREF
7	GND
8	VCC_3V3
9	TARGET_PWR_GOOD
10	I2C_SCL
11	I2C_SDA/
12	IREF
13	VCOMH
14	V13

2.1.6 J6, Micro SD Connector

The Micro SD connector, J6, is used to interface the C5515 processor to a Micro SD card. The signals on the pins of this connector are shown below.

Table 8: J6, Micro SD Connector

Pin #	Signal Name
1	DAT2
2	DAT3
3	CMD
4	VDD
5	CLK
6	VSS
7	DAT0
8	DAT1
9	INSERT
10	INSERT_COM

2.1.7 P1, P2, Bluetooth Board Interface

Connectors P1, and P2 are expansion connectors used to provide an interface to a Bluetooth board. The signals on the pins of these connectors are shown in the tables below.

Table 9: P1, Bluetooth Board Interface

Pin #	Signal Name	Pin #	Signal Name
1	GND	2	SD_DATA0
3	UART_RTS	4	SD_DATA1
5	RTC_CLKOUT	6	SD_DATA2
7	UART_RX	8	SD_DATA3
9	UART_TX	10	GPIO4
11	I2C_SDA	12	GPIO5
13	I2C_SCL	14	I2S2_FS
15	SD_CLK	16	I2S2_CLK
17	SD_CMD	18	I2S2_DX
19	GND	20	I2S2_RX

Table 10: P2, Bluetooth Board Interface

Pin #	Signal Name	Pin #	Signal Name
1	NC	2	GND
3	NC	4	NC
5	NC	6	NC
7	VCC_3V3	8	I2S1_DX
9	VCC_3V3	10	I2S1_RX
11	I2S1_FS	12	NC
13	GPIO12	14	NC
15	GPIO14	16	NC
17	I2S1_CLK	18	UART_CTS
19	GPIO14	20	GPIO13

2.1.8 P4, Expansion Connector

The Expansion connector, P4, is used to bring signals from C5515 DSP out to a connector for user interface. This card edge connector has all of the odd number (1,3,...,59) tabs on the top side of the board and all of the even number tabs (2,4,...,46) on the bottom side of the board. The diagram below shows the position of these tabs.

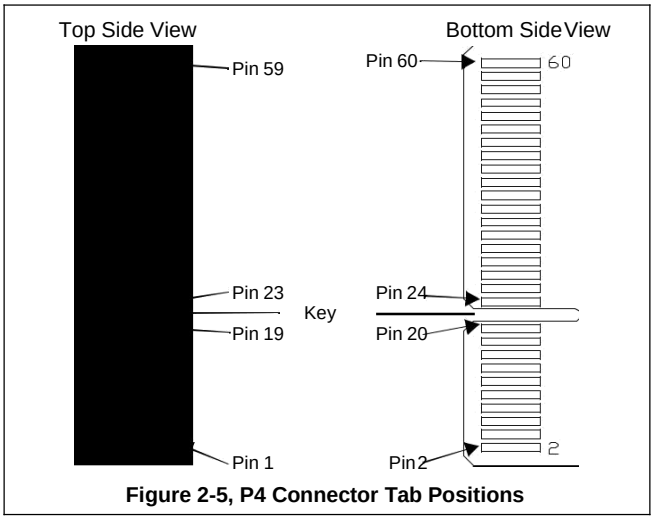


Figure 2-5, P4 Connector Tab Positions

The table below lists the signals that appear on each of the tabs of connector P1. The signals on the pins of this connector are shown below.

Table 11: P4, Expansion Connector

Pin # Top	Signal Name	Pin # Bottom	Signal Name
1	GND	2	GND
3	SPI_CS1	4	GPIO13
5	SPI_CLK	6	GPIO12
7	SPI_TX	8	GPIO14
9	SPI_RX	10	GPIO15
11	GND	12	GND
13	GND	14	GND
15	GND	16	GND
17	I2C_SDA	18	GPIO16
19	I2C_SCL	20	GPIO17
	Key		Key
23	I2S2_CLK	24	SD_DATA3
25	I2S2_RX	26	SD_DATA2
27	I2S2_DX	28	GPIO5
29	I2S2_FS	30	GPIO4
31	GND	32	GND
33	SD_CLK	34	UART_RTS
35	SD_DATA1	36	UART_CTS
37	SD_DATA0	38	UART_RX
39	SD_CMD	40	UART_TX
41	VCC_3V3	42	VCC_3V3
43	VCC_3V3	44	VCC_3V3
45	I2S0_CLK	46	SPI_CS3
47	I2S0_RX	48	VCC_3V3
49	I2S0_DX	50	GPAIN3
51	I2S0_FS	52	GPAIN2
53	SPI_CS2	54	GPAIN1
55	SPI_CS0	56	GPAIN0
57	VCC_3V3	58	VCC_3V3
59	VCC_3V3	60	VCC_3V3

2.2 System LEDs

The C5515 eZdsp USB Stick has 5 Light Emitting Diodes (LED). These LEDs are under the application software control running on the C5515 processor. These LEDs are shown in the table below.

Table 12: System LEDs

LED #	Color	Schematic Page	Signal Name
DS1	Green	2	C5515 XF
DS2	Green	11	GPIO17
DS3	Red	11	GPIO18
DS4	Yellow	11	GPIO15
DS5	Blue	11	GPIO14

2.3 Switches

The C5515 eZdsp USB Stick has two push button switches. These switches can be read by application software running on the C5515 processor. These switches are shown in the table below.

Table 13: Switches

Switch #	Schematic Page	Signal Name/Reading
SW1 only closed	11	GPAIN1, approximately 1.2 volts
SW2 only closed	11	GPAIN1, approximately 0.9 volts
SW1, SW2 closed	11	GPAIN1, approximately 0.72 volts

2.4 Jumpers

The C5515 eZdsp stick has one jumper, JP1. When this jumper is shorted the C5515 eZdsp stick will be powered from the target USB interface, not the embedded USB interface. When this jumper is shorted the embedded USB emulation and debug capability are no longer available. This jumper is shipped in the "**open**" state from the factory.

WARNING !

When the jumper J1 is shorted do NOT plug the connector J2 into a USB port. This will damage the C5515 eZdsp stick.

2.5 Test Points

The C5515 eZdsp USB Stick has six (6) test points for the monitoring of signals. The location of the test points are shown in the figure below.

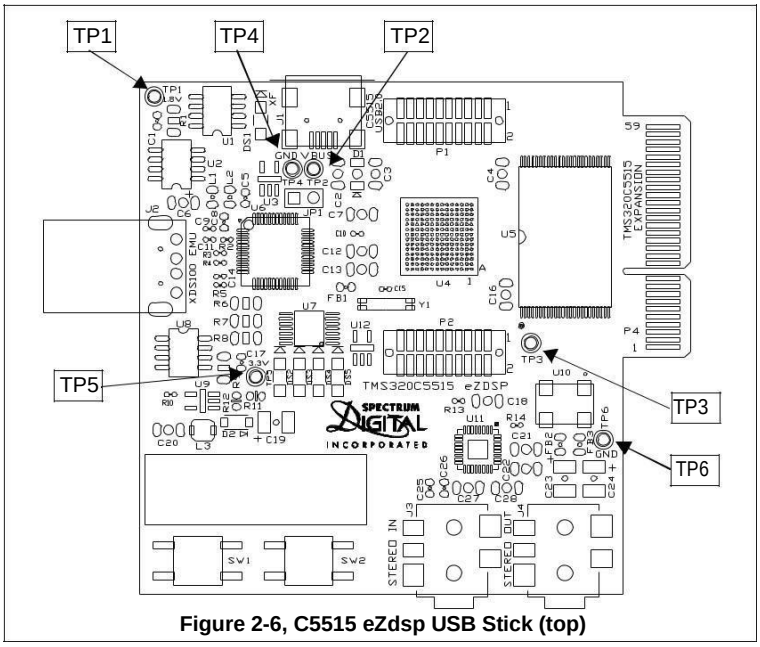


Figure 2-6, C5515 eZdsp USB Stick (top)

The signals on the test points are shown in the table below.

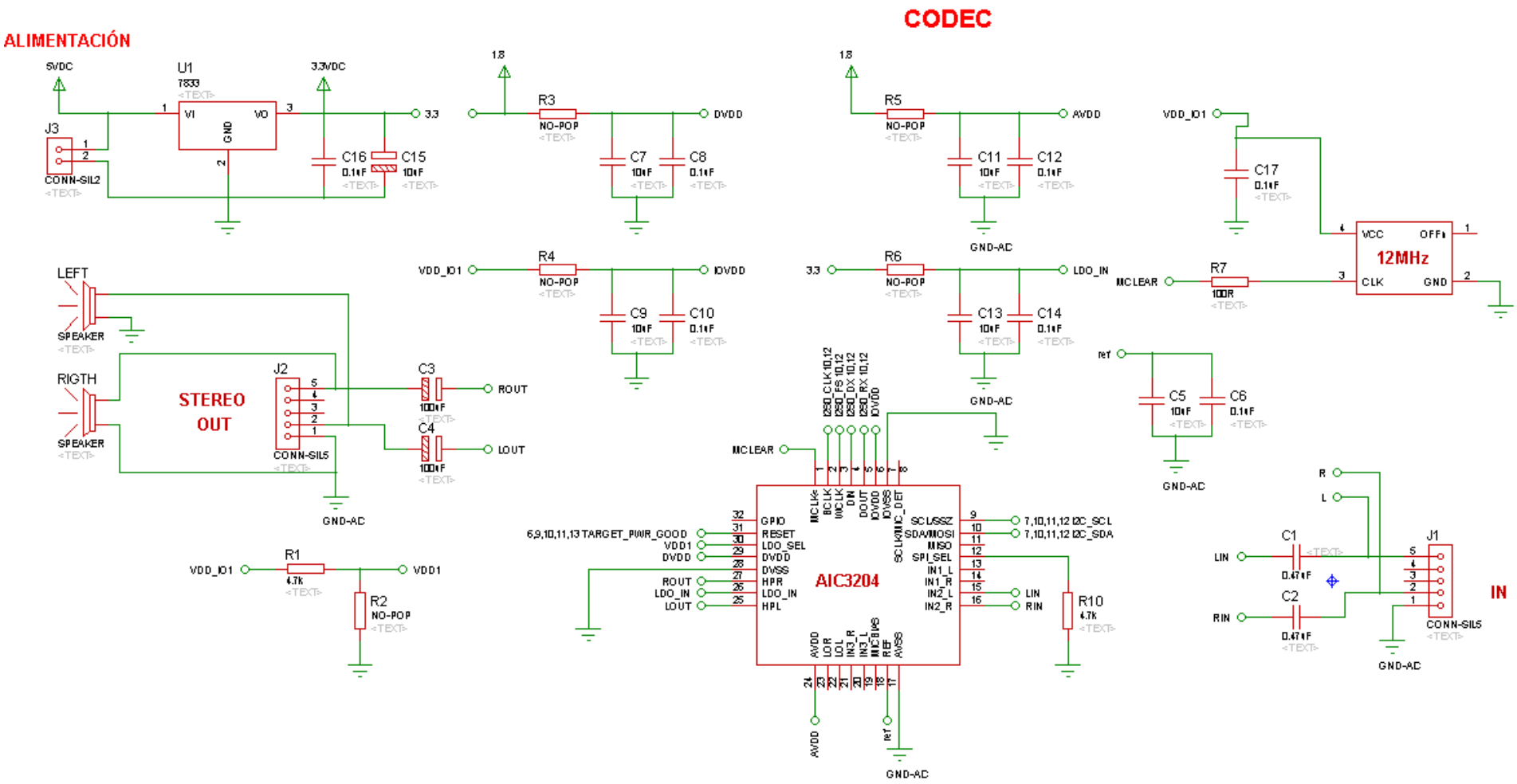
Table 14: Test Points

TP #	Schematic Page	Signal Name
TP1	9	VCC_1V8
TP2	3	VBUS
TP3	3	CLKOUT
TP4	5	GND
TP5	9	VCC_3V3
TP6	10	GND

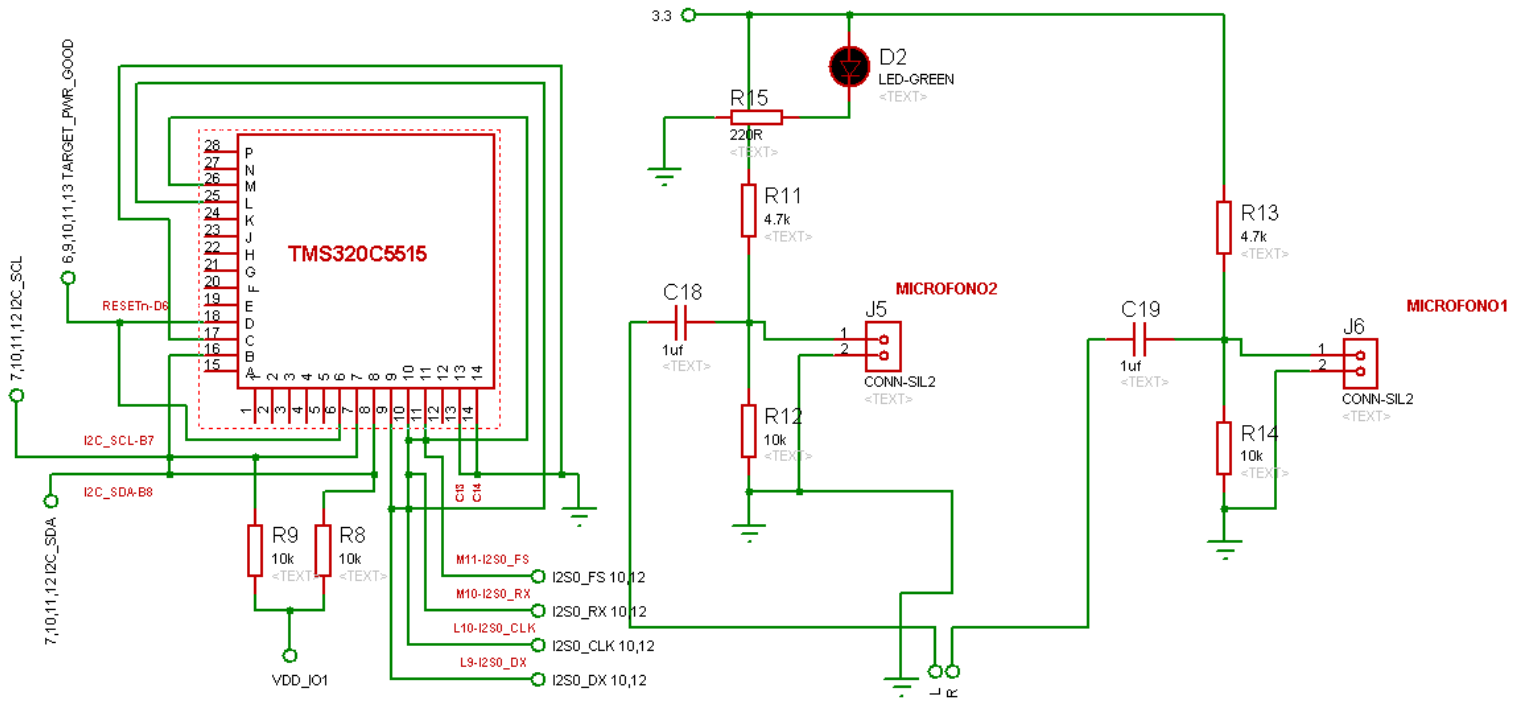
ANEXO 2

DISEÑO DEL CIRCUITO DEL FILTRO ADAPTATIVO CON RNA

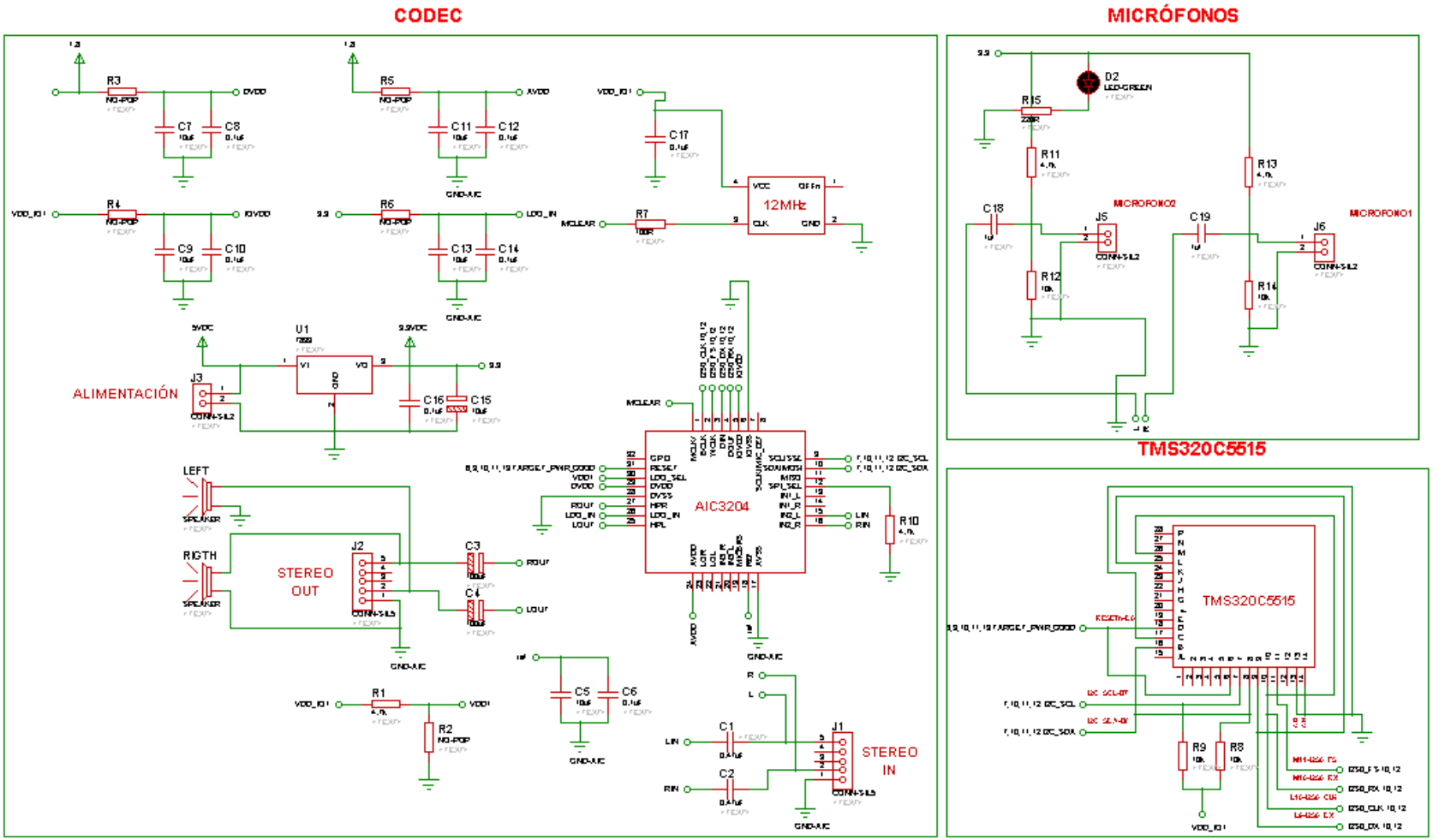
Circuito CODEC



Circuito TMS320C5515 y Micrófonos



Circuito Final del Filtro Adaptativo con RNA



ANEXO 3

DESARROLLO DEL

PROGRAMA FINAL EN CODE

COMPOSER STUDIO

```
/*
 * fir_filter.c
 *
 *      Author: Miguel Samaniego, Verónica Silva
 */

#include <usbstk5515.h>
#include <usbstk5515_i2c.h>
#include <AIC_func.h>
#include <stdio.h>

#define TCR0 *((ioport volatile Uint16 *)0x1810)
#define TIMCNT1_0 *((ioport volatile Uint16 *)0x1814)
#define TIME_START  0x8001
#define TIME_STOP   0x8000
//declaracion de var

Int16 me0,me1,me2,me3,met,ru0,ru1,ru2,ru3,rut;
Int16 out;
Int16 a0,a1,a2,a3,at,error;
float w0,w1,w2,w3,mult;

void main(void){
Int16 right, left;  //AIC inputs
USBSTK5515_init();                                  //Initializing the Processor
AIC_init();                                          //Initializing the Audio Codec
//Priming the PUMP
///////////////aumentado////////
TCR0=TIME_STOP;
TCR0=TIME_START;       //Resets the time register
//////////////////////////////
```

```
w0=0.7;          me0=0;          ru0=0;
w1=0.2;          me1=0;          ru1=0;
w2=0.2;          me2=0;          ru2=0;
w3=-0.2;         me3=0;          ru3=0;

while (1){
AIC_read2(&right, &left);
me3=right;
ru3=left;

AIC_read2(&right, &left);
me2=right;
ru2=left;
AIC_read2(&right, &left);

me1=right;
ru1=left;
AIC_read2(&right, &left);
me0=right;
ru0=left;

a3=w3*ru3;
a2=w2*ru2;
a1=w1*ru1;
a0=w0*ru0;

at=a0+a1+a2+a3;
//printf("%d\n", at );
//printf("%d\n", me0 );
error=me0-at;
//printf("%d\n", error );
```

```
 mult=(2*(0.0000000001)*error);

//mult=mult2;
 //printf("%f\n", mult );
 //mult2=mult;
 //printf("%f\n", mult2 );
 //printf("%d\n", ru3 );
 //printf("%f\n", w3 );
w3=w3+(mult*ru3);
 //printf("%f\n", w3 );
 w2=w2+(mult*ru2);
 w1=w1+(mult*ru1);
 w0=w0+(mult*ru0);
 //printf("%d\n", w3 );
 //printf("%d\n", w2 );
 //printf("%d\n", w1 );
 //printf("%d\n", w0 );
 //error=met-rut;
 //AIC_write2(w3, w3);
 //AIC_write2(at, at);
 //AIC_write2(me0, me0);
 //AIC_write2(ru0, ru0);
 AIC_write2(error, error);
 //AIC_write2(mult, mult);
 }
```

ANEXO 4

DESARROLLO DEL PROGRAMA FINAL EN MATLAB

```
clc; clear all;close all;
Senal=wavread('C:\Users\Usuarios\Documents\MATLAB\filtro con
redes\senal1.wav');
Ruido=wavread('C:\Users\Usuarios\Documents\MATLAB\filtro con
redes\ruido1.wav');
%%lar=1323000;
lar=100000;
%Senal=Senal(1:lar,1);
%Ruido=Ruido(1:lar,1);
Senal1=Senal; Ruido1=Ruido;
Mezcla=Senal+Ruido;
%sound(Mezcla,44100)
% Valores óptimos W=[0.5030;0.9922;-0.4970]; alpha=0.0001;
%W=[0.5;0;0;-0.5];
for i=1:lar
error=Mezcla(i)-Ruido1(i);


    Salida(i,1)=error;
    Error(i,1)=Senal(i,1)-error;
end
%for i=1:lar
 %     if i==1
 %       P=[Ruido(i);0;0;0];
 %     elseif i==2
 %       P=[Ruido(i);Ruido(i-1);0;0];
 %     elseif i==3
 %       P=[Ruido(i);Ruido(i-1);Ruido(i-2);0];
 %     else
 %       P=[Ruido(i);Ruido(i-1);Ruido(i-2);Ruido(i-3)];
 %     end

 %    a=W'*P;
 %    a=(Ruido(i)+Ruido(i-1)+Ruido(i-2)+Ruido(i-3))/4;
 %    error=Mezcla(i)-a;
 %   W=W+(2*(0.001)*error*P);

 %    Salida(i,1)=error;
 %    Error(i,1)=Senal(i,1)-error;
%end

%sound(Salida,44100)

%Resultado=Senal-Salida;
%plot(Salida,'-b');axis([0 lar -1 1]);
title('SALIDA');xlabel('Muestras');ylabel('Amplitud');
%figure;plot(Error,'-b'); axis([0 lar -0.1 0.1]);
title('ERROR');xlabel('Muestras');ylabel('Amplitud');
%figure;plot(Senal,'-b');axis([0 lar -1.3 1.3]);title('SEGMENTO DE UNA
CANCIÓN');xlabel('Muestras');ylabel('Amplitud');
%figure;plot(Ruido,'-b');axis([0 lar -1.3
1.3]);title('RUIDO');xlabel('Muestras');ylabel('Amplitud');
%figure;plot(Mezcla,'-b');axis([0 lar -1.3
1.3]);title('MEZCLA');xlabel('Muestras');ylabel('Amplitud');
```

ANEXO 5

EJEMPLOS DE PROGRAMACIÓN EN CODE COMPOSER STUDIO

EECS 452 Lab 1—Introduction to the C5515 USB Stick

The purposes of this lab are to:

- Provide an introduction to the C5515 USB Stick, the board we will be using throughout this course.
- Provide an introduction to embedded systems and memory-mapped I/O devices.
- Provide a very basic experience with direct digital synthesis (DDS)

In addition, you will gain familiarity with the software development environment and the various forms of documentation and sample code that comes with this platform.

Figure 1: The C5515 eZDSP USB Stick

1. Background Material

There is quite a bit of material you'll want to understand before you start working on this lab. Some of that material was covered in previous courses, but some of it you'll be expected to pick up on your own.

Programming

All labs this semester will rely upon you having a fairly strong understanding of C programming. In EECS 280 and Engineering 101 you gained a fairly strong background in C++ programming. The differences between C and C++ are largely trivial. The two biggest, for purposes of this class, are that:

- There are no classes, nor are their function members in structures.
- Rather than using streams ("cout" and "cin") to do input and output we use "printf" and "scanf".

A brief tutorial on "C for C++ programmers" can be found on the 452 lab webpage under "General references"

Overview of Memory-mapped input/output

When using an embedded system, you generally need some way to move data into, and out of, the processor. How does the computer do that? In 280 the answer was simple; you just call a function/class (ofstream's *open* function to open a file was the most likely scheme). But that begs the question—how does the function do it? How do does the processor talk to the disk and other I/O devices?

The answer is fairly straightforward. Recall that your program and its data all live in memory. Each instruction, variable and array element are stored in a specific memory location. You can think of the computer's memory as a big array in which the program and its data reside. When you declare a variable, a memory location is reserved for that variable. Pointers are just the index into that big array that is memory. So when you do a *new* in C++, the pointer you get back from new is just the index into that big array where the operating system has given you space to work.

It turns out certain elements of that big memory array aren't really memory, they are instead dedicated I/O devices. What that means is that some memory locations are special and reads and writes to them might have side effects. A simple example would be to have memory location 0xDEADBEEF[1] mapped to an LED. It might be configured so that if the value in that memory location was non-zero the LED would turn on. One could have a different memory location where a switch's value could be found (say depending on the switches setting the memory location might be a 0 or a 1). Thus, your program could find the state of that switch simply by reading that memory location.

Significantly more I/O devices can be addressed with Memory Mapped Input/Output (MMIO). In fact all I/O devices; including disk drives, monitors and keyboards; communicate with the processor via MMIO. Wikipedia has a Memory Mapped I/O overview which you might find provides a helpful background.

Details of Memory-mapped input/output on the C5515

In most machines Memory Mapped I/O is simply a part of the overall memory map; but on the C5515 (and most TI DSPs) there is a separate memory space dedicated to the use of I/O devices. This is signaled by the keyword ***ioport***. This tells the processor that the address associate with this variable is mapped to the I/O address space, instead of the Data address space. Let's look at an example of a pointer to I/O memory:

```
ioport int * RandMemLoc;                  //This declaration is "wrong" because it
RandMemLoc = (ioport int *) 0x12;         //lacks the volatile keyword!
```

The first line of this example declares and sets a variable named **RandMemLoc** which is pointer to an integer. It is then set to point to the memory location 0x12, in the ioport memory space. Here's an analogy which might help understand what the keyword **ioport** is doing here. If the address 0x12 is a page number then ioport is a title of a book.

There is also another keyword that must be used with MMIO, and that is the keyword ***volatile***. That keyword tells the system that it must load and store all the way to memory with each and every use[2]. Here is an example of how to set a variable to volatile. We'll do both the declaration and the assignment of value in one line this time.

```
volatile ioport int * RandMemLoc  = (ioport int *) 0x12;
```

[1] (Recall that "0x" indicates that the number that follows is written in hexadecimal, so 0xDEADBEEF would be 3735928559 base 10).

[2] Normally the *computer* will cache data and the *complier* will store commonly-used data in its register file. The volatile keyword announces the fact that the data could be read or written by something other than the program and therefore every read and write to the data needs to actually go all the way to memory (it can't stop in the register file or cache).

Handing in your work

In the pre-lab there will be a number of questions (labeled Q1, Q2, etc.). You are to create a single document (likely in Word) that answers each of the questions. It should be clear what question you're answering and the answers should be in order. The pre-lab is to be done individually, and *each student will turn in their own answers for the pre-lab.* Pre-labs are due at the beginning of the lab.

Make a copy of your pre-lab, you will need it for the in-lab part.

The lab and post-lab will also have a number of questions. The post-labs will tend to be longer questions that can be done outside of lab. Each team will turn in one set of answers for this part. The lab section also has a number of "sign-off" points where the GSI will need to sign-off that you've done a step correctly. The GSI may provide you with a blank sheet or you may need to create your own. You'll turn in this sign-off sheet along with the questions and any other requested information.

Post-labs are due at the beginning of the lab section on the following week.

2. Pre-Lab

The pre-lab questions in this class are to be done individually unless otherwise noted. Each student will turn in their own pre-lab.

In general demo boards are shipped with sample code that exercises the basic functionality of the board. The C5515 USB Stick is no exception. The problem with relying on such sample code is that A) sometimes (usually?) it's written poorly or in an unclear way and B) it can't cover everything. So you end up needing to fall back on the actual documentation which itself is sometimes less than clear.

For this pre-lab you are asked to write a function that "simply" allows you to turn on and off the four LEDs on our board. As is common in embedded systems, the problem is finding the right information in the documentation. In this pre-lab we'll walk you through finding the right information and writing the needed code from scratch. Below we have provided most of two functions which configure the LEDs and allow an individual LED to be turned on and off as well as a main which uses them. The function **My_LED_init()** configures the processor pins connected to the LEDs to be outputs. The function **toggle_LED(int index)** takes a number (0, 1, 2 or 3) and lights the corresponding LED. We've left a number of blanks for you to fill in.

The first step in locating the information we need to code the LED on/off function is to find how the LEDs are connected to the processor.

Our goal is to write code to turn on/off the LEDs. We know that the LEDs are connected to the MMIO. The first step is to find the documentation that tells us this mapping. Normal you would go to the manufacturer or distributor for this information, but you can find it on the EECS452 website. Look for the document named USBSTK5515 Technical Reference Revision A. You want to find which GPIO pins (General purpose I/O) are connected to the LEDs.

Q1. Which GPIO pin is connected to each of the four LEDs? (Red, Blue, Yellow and Green). Use the schematic (Apedix-12) not the table as there appears to be a typo in the table. In general, it is a good policy to trust the schematic above other references.

Once you find which GPIO pins are connected to the LEDs you have to find out how to use the GPIO pins. You can get this information in the TMS320C5515 General-Purpose Input/Output User's Guide. This user's guide ca be found on the EECS452 website (C5515_gpio_guide.pdf.) You will need to read sections 2.4, 3.1 and 3.3 at a minimum. You would be well-served to read all of chapters 2 and 3 to get a comprehensive understanding of the GPIO system on the C5515 processor.

Important note: *the LEDs are active low: to turn them on we need the GPIO pin to be 0.*

Q2. What are the names and memory locations of the registers you'll need to use to set the GPIO pins associated with the LEDs to be outputs?

Q3. For *each* of the four LEDs indicate which bit number of which register you'll need to write to in order to set that LED's GPIO pin to be an output.

Q4. Assuming the GPIO pins have been setup as outputs, indicate the register names, their addresses and which bits you'll need to write to what value in order to turn on the LEDs.

Now that we've figured out what values we need to write to which memory locations, we need to figure out how to write a single bit to a given memory location. Before answering the next question you may want to search a bit around the internet on the topic of "bitwise manipulation."

Q5. Answer the following questions. Assume all values are 16-bit unsigned where bit 15 is the most-significant bit and bit 0 the least significant.

a. What is the value of the number where bit 4 is a 1 and all other bits are zero? Provide your answer in both decimal and hex.
b. What is the value of the number (1<<4)? Provide your answer in both decimal and hex.
c. If x=0x2222, what is the hex value of x after the line **x|=(1<<4)**?
d. Write C code which sets bit 7 and 8 of the variable x.
e. Write C code which clears bit 1 of the variable x using the **&=** operator.
f. Write C code which toggles bit 13 of the variable x using the **^** operator.

Q6. Provide the code below with the blanks correctly filled in.

```
/*
 * main.c
 *      Author: GSI
 */

#include <usbstk5515.h>
#include <stdio.h>

//Addresses of the MMIO for the GPIO out registers: 1,2
#define LED_OUT1 *((ioport volatile Uint16*)  blank a  )
#define LED_OUT2 *((ioport volatile Uint16*)  blank b  )
//Addresses of the MMIO for the GPIO direction registers: 1,2
#define LED_DIR1 *((ioport volatile Uint16*)  blank c  )
#define LED_DIR2 *((ioport volatile Uint16*)  blank d  )


//Toggles LED specified by index Range 0 to 3
void toggle_LED(int index)
{
       if(index == 3)  //Blue
              LED_OUT1 = LED_OUT1 ^ (Uint16)(1<<(  blank e  ));
       else if(index == 2)  //Yellow(ish)
              LED_OUT1 = LED_OUT1 ^ (Uint16)(1<<(  blank f  ));
       else if(index == 1)  //Red
              LED_OUT2 = LED_OUT2 ^ (Uint16)(1<<(  blank g  ));
       else if(index == 0)  //Green
              LED_OUT2 = LED_OUT2 ^ (Uint16)(1<<(  blank h  ));
}

//Makes the GPIO associated with the LEDs the correct direction; turns them off
void My_LED_init()
{
       LED_DIR1 |=  blank i  ;
       LED_DIR2 |=  blank j  ;

       LED_OUT1 |=  blank k  ;  //Set LEDs 0, 1 to off
       LED_OUT2 |=  blank l  ;  //Set LEDs 2, 3 to off
}

void main(void)
```

```
{
        Uint16 value;
        USBSTK5515_init(); //Initializing the Processor
        My_LED_init();
        while(1)
        {
                printf("Which LED shall we toggle(0, 1, 2, or 3)?\n");
                scanf("%d",&value);
                toggle_LED(value);
        }
}
```

Now we'll change gears and get you started on some other parts of the lab: drawing on the LCD and generating discrete sine waves.

Q7. Consider the zigzag.c code found on the lab website (or CTOOLS.) It draws a zigzag pattern on the screen. *Draw*, freehand, what you expect this code to generate on the LCD. We are particularly concerned with how many zigzags you expect to see. You may find it helpful to read the document SSD1306 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller (you can find it with a web search for "SSD1306.pdf"). Hint: ignore the OSD9616_send function for now, focus on the state of the **top** and **bottom** arrays. Page 25 of the document will also be helpful. Be aware that this document refers to a LCD with 8 pages but our LCD has only 2 pages.

Q8. Say you have a 50-entry sine table (that is the table[x] has the value for $\sin(2\pi*x/50)$). Say you output each value for 1ms, then after you do the last value (table[49] in this case) you loop around and output table[0] again. Obviously the sine waves will be a bit choppy…

a. What is the frequency of the sine wave you'd generate?
b. What if you instead held each table entry for two samples in a row (2ms)? What would the frequency be?
c. What if you instead drove every other table entry for 1ms (so only table entries 0, 2, 4, etc.)? What would the frequency be?
d. Describe what you'd need to do to generate a 1Hz frequency sine wave.
e. Describe what you'd need to do to generate a 15Hz frequency sine wave.

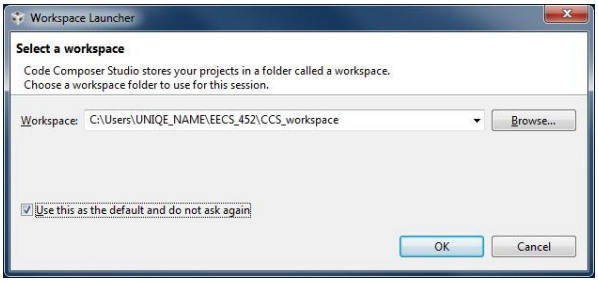
3. In-Lab

This lab has four parts. Also notice there is a "When things go wrong" section at the end.

- The first part is getting familiar with our tool (Code Composer Studio version 5) and learning how to set up a project from scratch using it. This part is a bit frustrating but you shouldn't need to do this very often (we'll generally just start a new project by copying a correctly set-up old one, but sometimes you'll need to do this from scratch).
- The second part will provide an introduction to using MMIO on the C5515. You'll insert the code you had in the pre-lab, fill in the blanks, and get the lights blinking.
- The third part will give you experience with more complex MMIO as you work with the LCD on the board.
- In the final part you will use the LCD to display sine waves you digitally create as well as signals from the function generator.

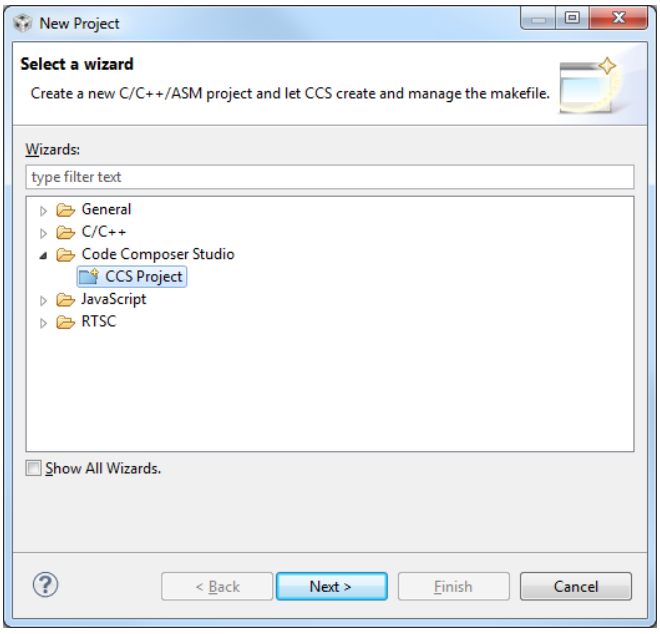
Part 1: Getting started with Code Composer Studio

To start Code Composer Studio (CCS) version 5 double click on the grey Rubik's cube icon on your desktop. Because this is the first time you are opening CCS, you will be prompted to choose a workspace. It is important that you chose a path that CCS sees as "local" to the machine. CCS will not follow symbolic links at compile time. If on a CAEN computer, place your workspace in the C:\ drive. Put your workspace somewhere easy to remember. For example, C:\Users\[uniqename]\EECS_452\CCS_workspace. Check the box so CCS doesn't bother you about it again. Click "OK." CCS will create the directory for you if it doesn't already exist.

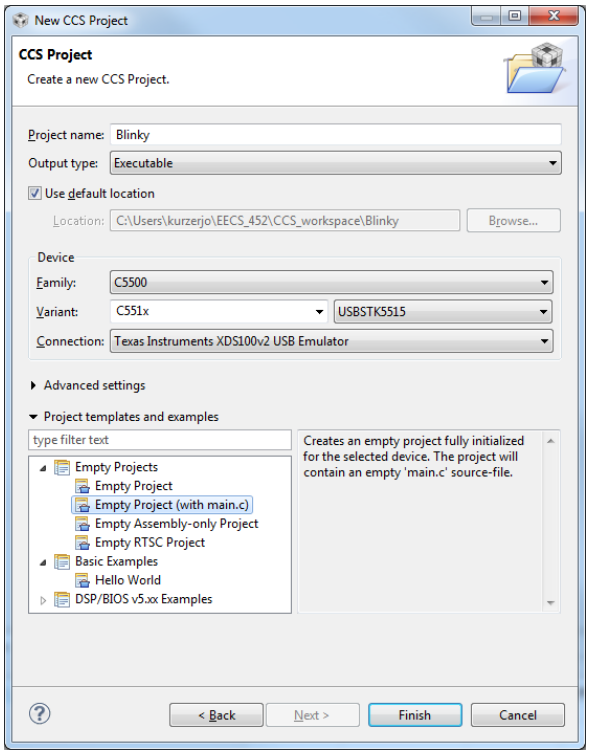


Next, add the support files you will need for the rest of the semester. On the course website download C5515_Support_Files.zip. Unzip this directory an place it in the workspace you just created.

Now click on File->New-> Project. You should get the image to the right. Under 'Code Composer Studio' select 'CCS Project' and click 'Next.'



In the next window, you should get something similar to the figure to the right. We will name this project **Blinky**. Double check the Location of your project is under your account. You should ALWAYS check your workspace before working on any project. You can do a Switch Workspace under "File" in CCS.



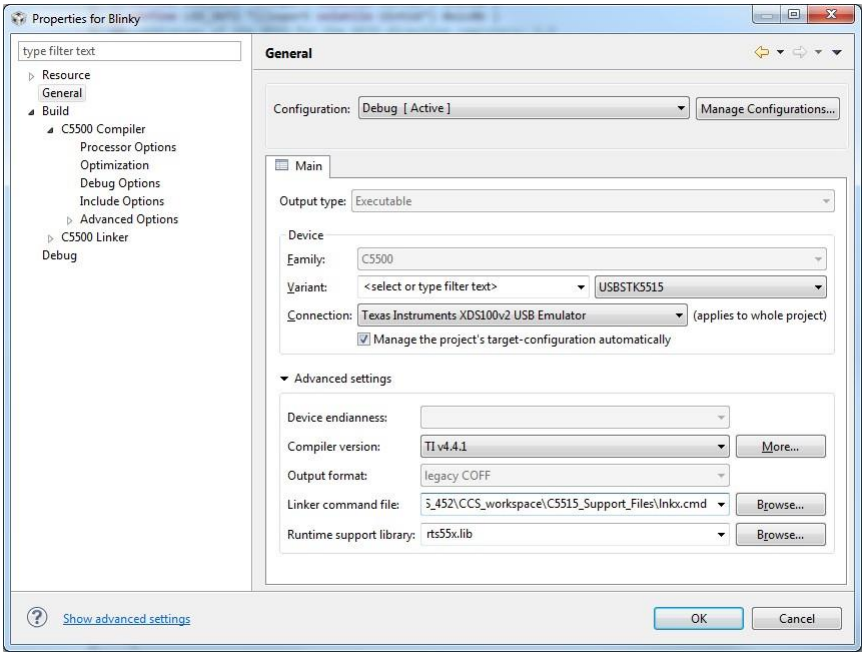
In the device section, set the C5500 family, and the USBSTK5515 variant. For 'Connection', chose the 'Texas Instruments XDS100v2 USB Emulator.' Finaly chose to create and 'Empty Project (with main.c) and click 'Finish.'

You should see that your workspace is populated with a project named Blinky in the 'Project Explorer' window. You may need to close the 'TI Resource Explorer' to see this.

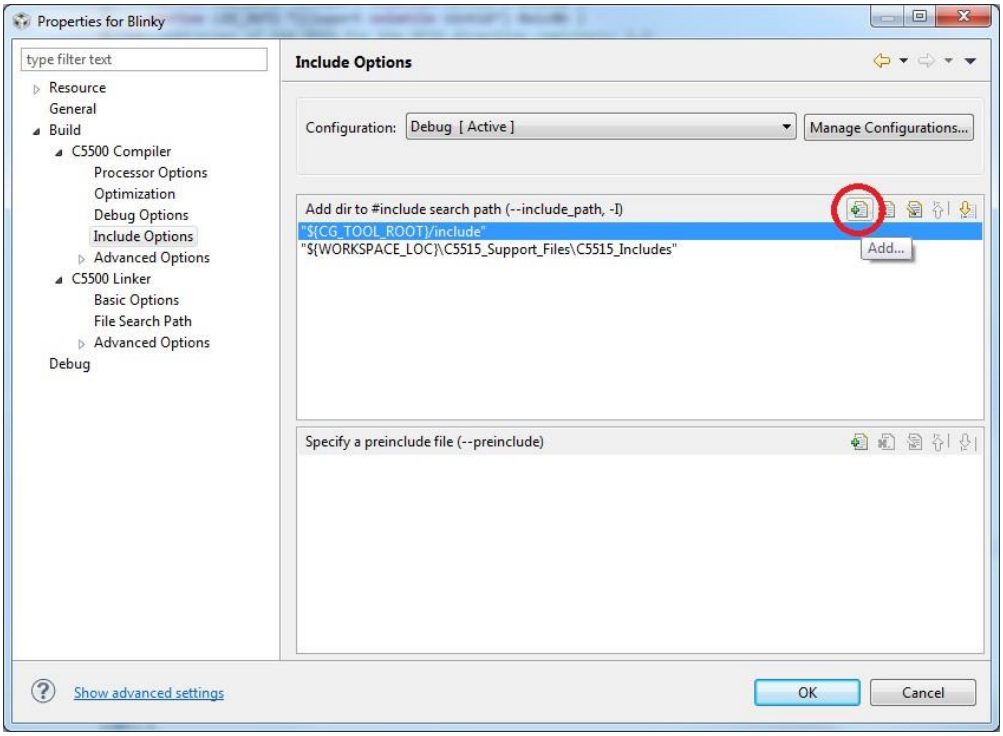
Double click on main.c under Blinky, and you will see an empty main function that the project wizard has created for you. Copy the code from the pre-lab (complete with blanks) into main.c. Replace the blanks as per your answers to the pre-lab. The pre-lab code can be found on the course website inside the Lab1_Files directory named main.c. Save this file with "File->Save" or "CTRL-S" on the keyboard.

Now we have the code, but we still have to tell the compiler and linker where to look for source files and libraries that we need. Right click on the project ("Blinky") and select "Properties". The window below will appear. Look under the "General" settings.

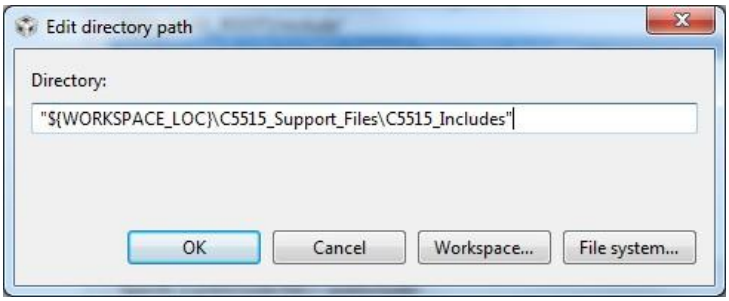
Under "Linker Command File" click on "Browse…" and navigate to your workspace. Inside the C5515 support files select "lnkx.cmd" and click "Open." Under "Runtime Support Library" select "**rts55x.lib**."



Next go to “Build” → “C5500 Compiler” → “Include Options” and you should see the following window:



Select the “Add” icon in the upper right of the window, (circled red in the above image). CCS will ask you for a directory. In the field enter "${WORKSPACE_LOC}\C5515_Support_Files\C5515_Includes" **INCLUDING QUOTES**.



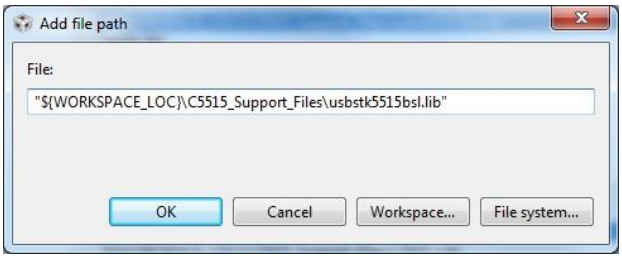
Note: ${WORKSPACE_LOC} is a path variable that is specific to your CCS setup. You can see this and other defined path variables in the properties window. “Resource” → “Linked Resources” in the “Path Variables tab.”

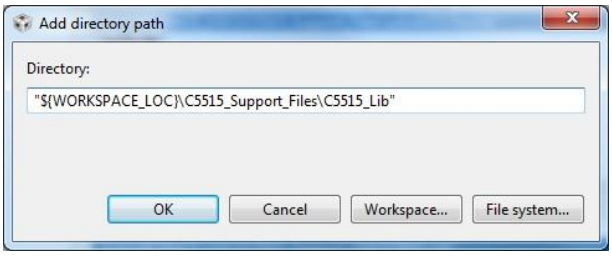
It is worth noting that you ***can*** also have the compiler search your projects for files that you have made. To do this click on the add path icon again. But this time you’d select “Workspace…” to get the window seen below. ***Because we aren’t using our own header file at this time, you will skip this step, but you will to do it in the future if you have any of your own header files.***

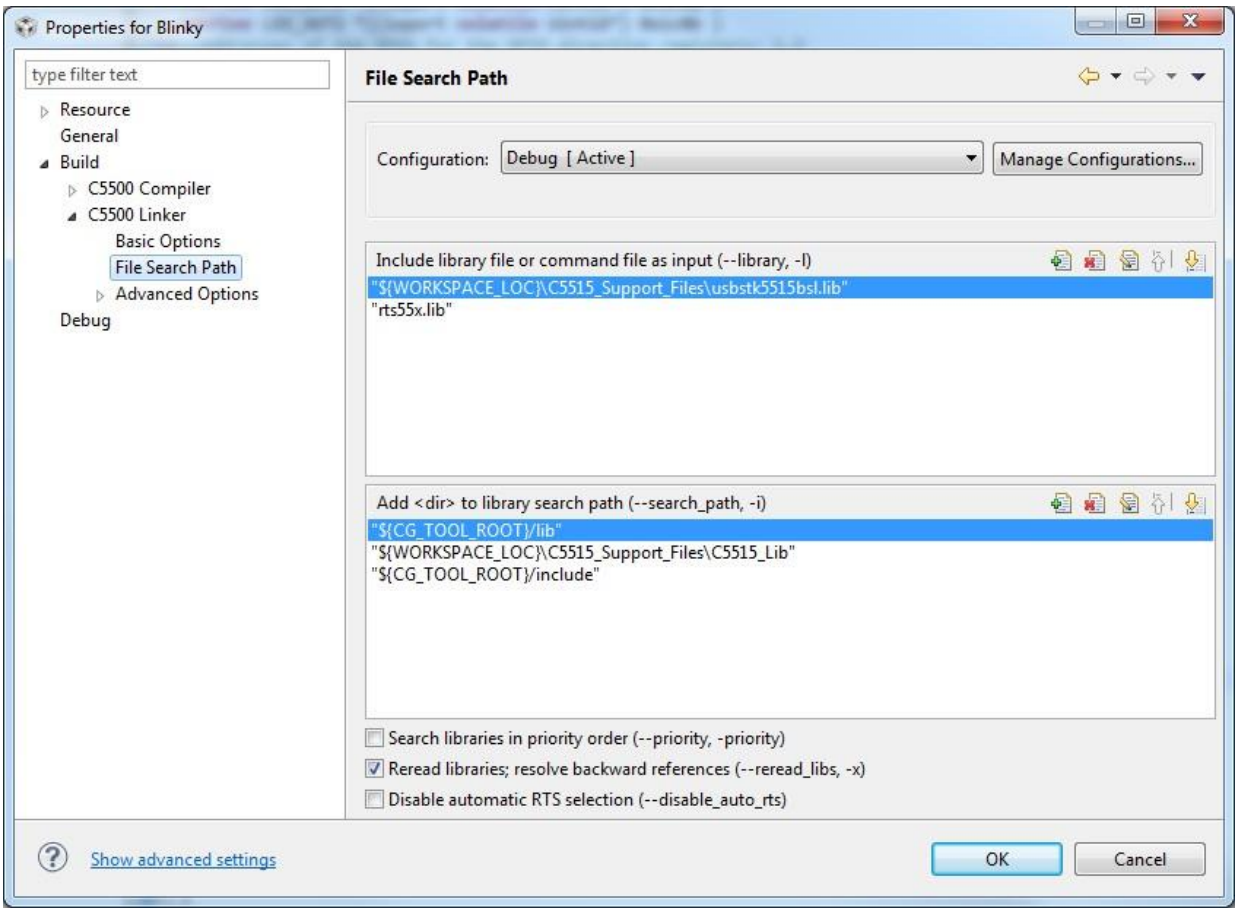
Now that we have told the compiler where to find our header files, we still need to tell it where to find the code to resolve the symbols declared in those header files. To do this, go to “Build” → “C5500 Linker” → “File Search Path.”

In the “Include library file or command” section, click the “add” button as before. In the “Add file path” window that pops up, enter "${WORKSPACE_LOC}\C5515_Support_Files\usbstk5515bsl.lib" (again, include quotes.)

In the “Add <dir> to library search path” section, clickthe “add” button as before. In the “Add directory path” window that pops up, enter "${WORKSPACE_LOC}\C5515_Support_Files\C5515_Lib" (again, include quotes.)

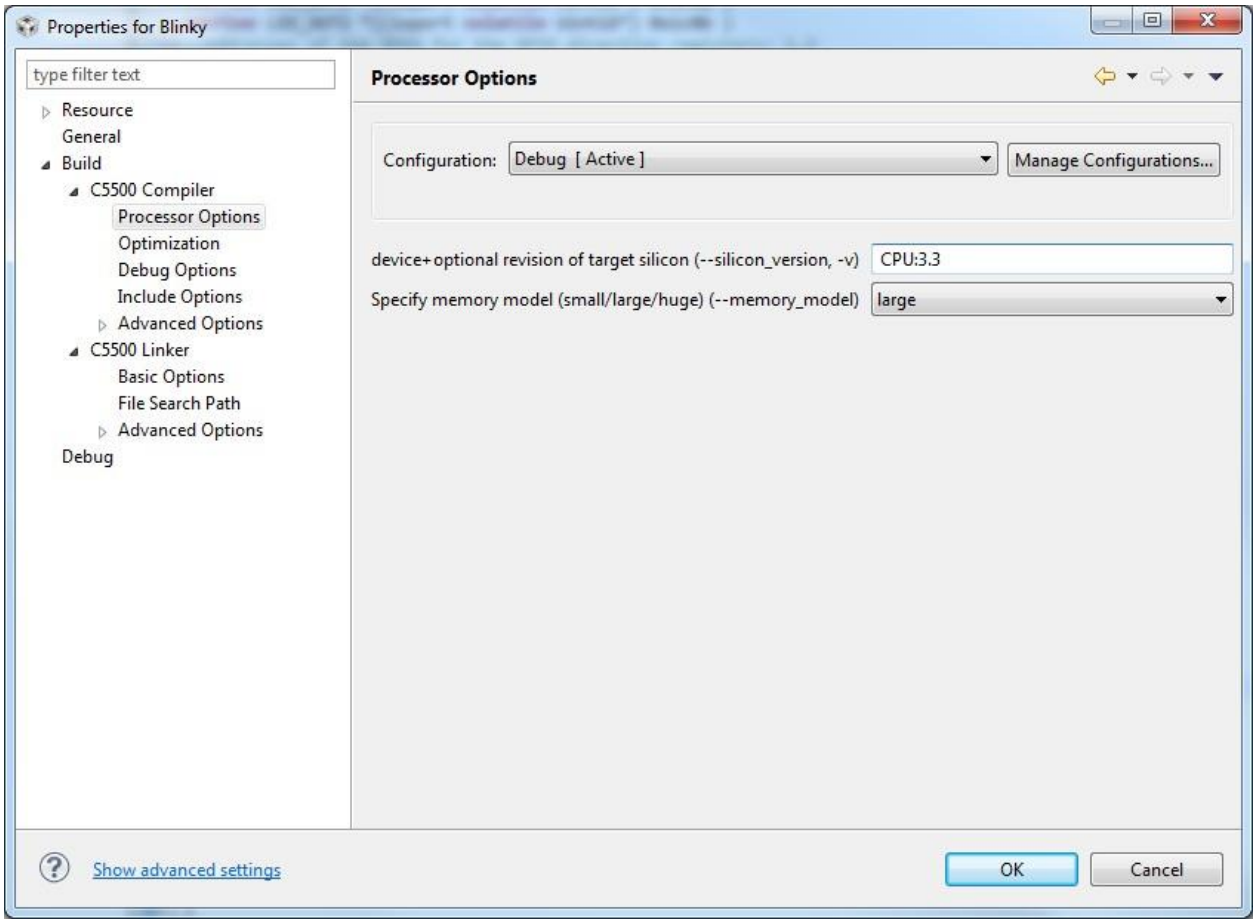






We still need to tell the compiler which silicon version and memory model we will be running. To do this, select the "Build" → "C5500 Compiler" → "Processor Options".

Replace the 5515 in the revision of target silicon with "CPU:3.3" (which tells the compiler which revision of the processor we are using). Make sure the memory model is specified to large. Once you've done that, click OK in the bottom right hand corner of the window and you will return to your project window.



Part 2: MMIO and the LEDs

We finally have CCS setup correctly. What you need to do now is build the project. It is likely that you've got errors of various sorts (syntax, logical, other) in your code. The rest of these directions assume your code is working correctly. If it's not, you are going to have to figure out what's wrong. It could be that you did something wrong in part 1, it could be you've introduced a syntax error or it could simply be that your pre-lab answers are wrong. This is going to take a fair bit of time. *Some debugging suggestions are found below, so take the time to read ahead a bit before you do anything.*

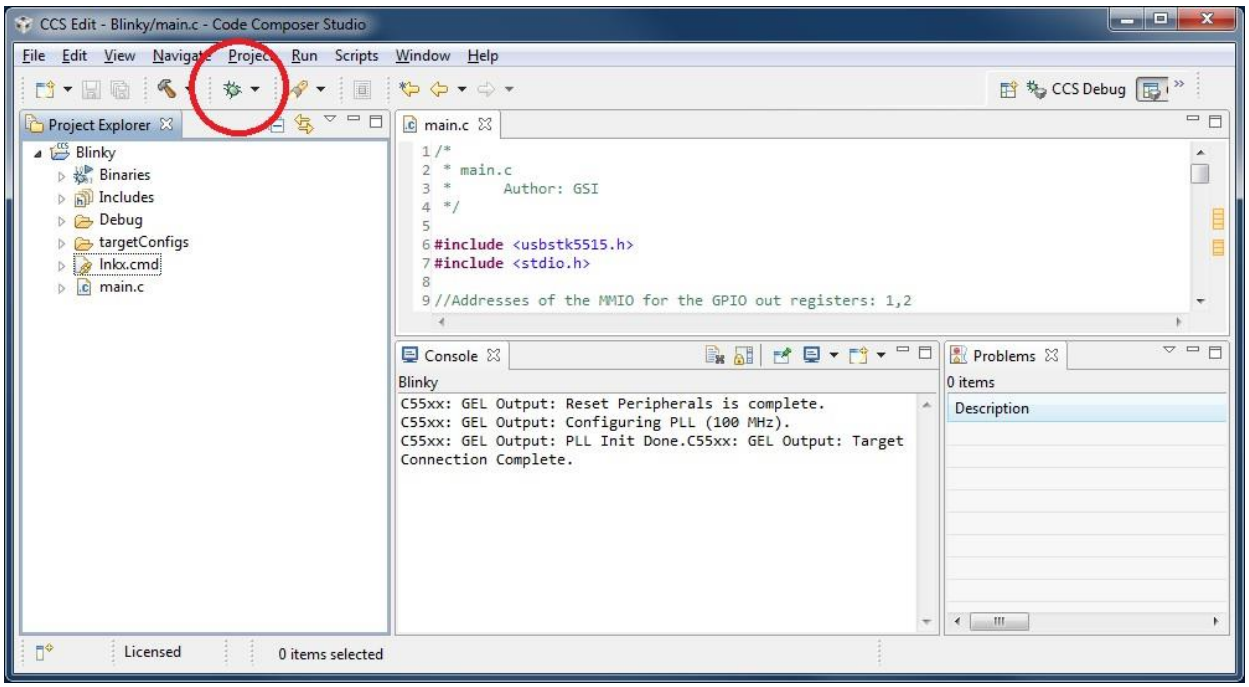
We've now got things set-up, but still need to build the project. This can be done by right clicking the project and selecting the "Build Project" option or by going to the Project tab and selecting "Build

Project" from there. Building readies the project to be loaded on to board by generating the necessary binaries that the C5515 will run. You'll need to fix any errors that pop up here.

If you get stuck at this step consider commenting out all but the following few lines. This should compile and run just fine (though it doesn't do anything other than prompt you for a number). If it doesn't, you've likely done something wrong in Part 1. Also, notice the section at the end of the In-lab called "When things go wrong".

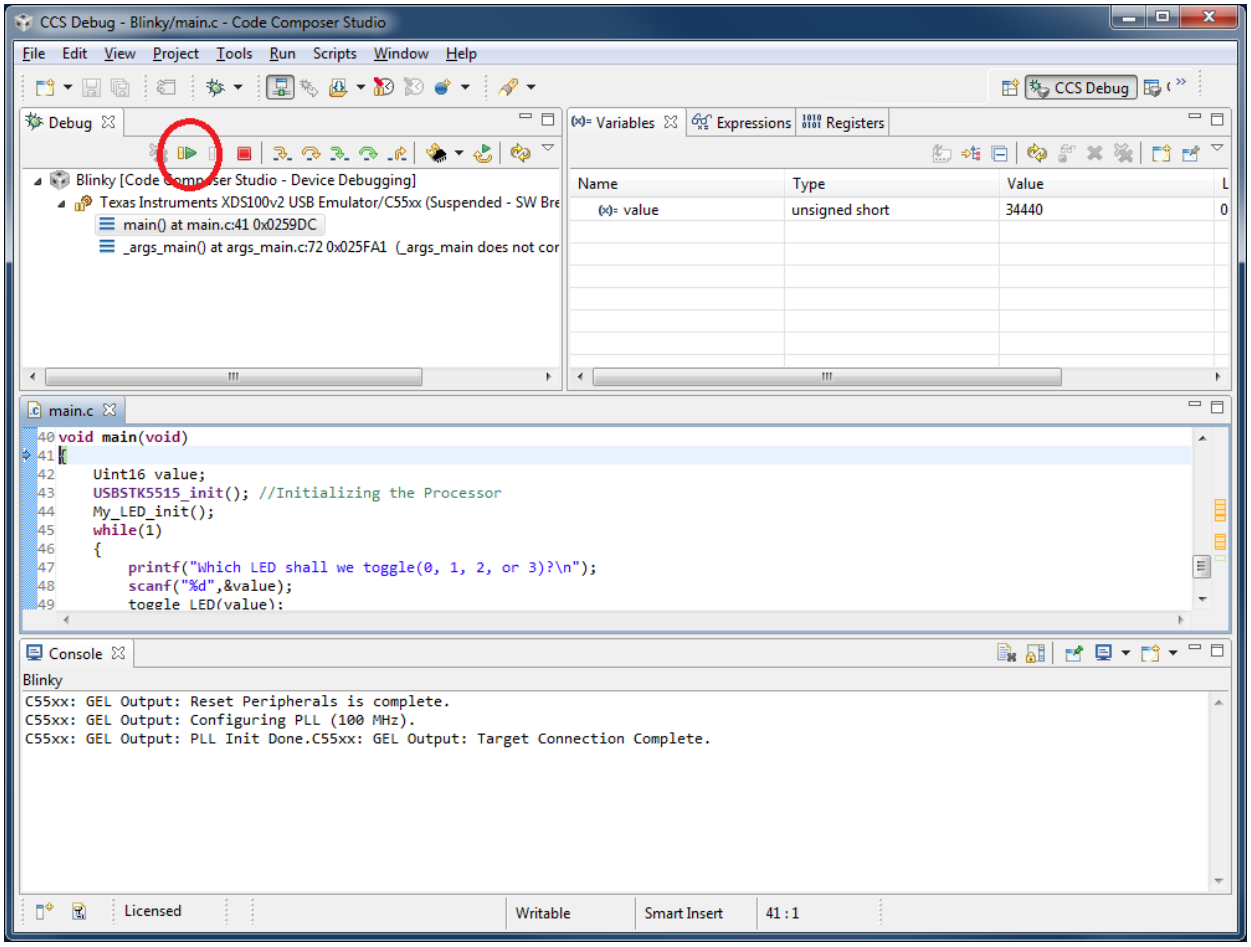
```
#include <usbstk5515.h>
#include <stdio.h>

void main(void)
{
	Uint16 value;
	USBSTK5515_init(); //Initializing the Processor
	while(1)
	{
		printf("Which LED shall we toggle(0, 1, 2, or 3)?\n");
		scanf("%d",&value);
	}
}
```

Now we need to launch the program and load it on to the C5515. First be sure the eZDSP stick is connected to your computer with a USB cable. Then select the "Debug" icon which we circled in red in the following figure. If you get an error message at this point, ask for help from a lab instructor[3].



[3] It may be the case you need to get a first debug configuration set up. This largely involves finding the ".ccxml" file appropriate to your device. Right click on the project folder and chose Open Target Configuration. From there you are looking for the Texas Instruments XDS100v2 USB Emulator as the connection type and the USBSTK5515 as the device.

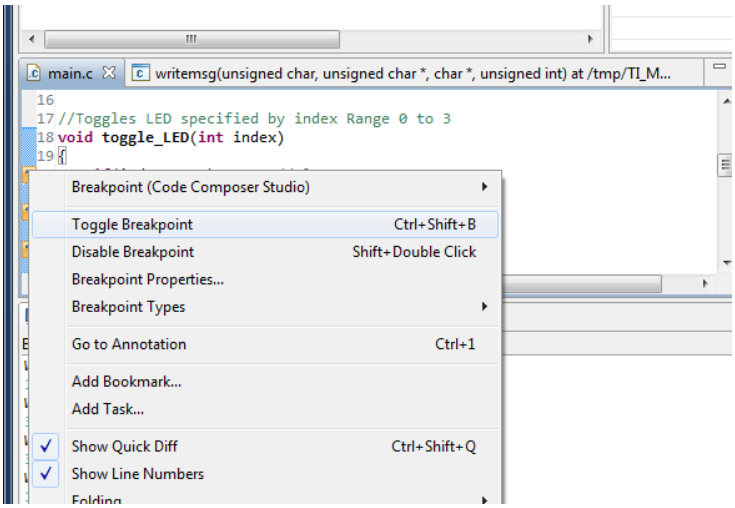
If all goes well you should be changed to the "CCS Debug" perspective as shown below.



This debug perspective allows you to easily run the program you just loaded. To run the program, click the green arrow toward the top-left side of the screen (circled in red in the figure above). The console should prompt you to enter which LED to toggle. If you don't see your console, make it visible "View" → "Console". Click in the "console" and enter 0, 1, 2 or 3 with the keyboard. The LEDs should toggle when you send the command through the console.

Of course, when things to go wrong simply running the program doesn't provide enough feedback for you to understand *what* is going wrong. This is what breakpoints are for. Breakpoints pause a running program so that you can use the debug perspective to see all the values of variables and the values stored in particular places in memory.

To set a breakpoint you will want to pause the program. Click the "Suspend" icon next to the green arrow icon (double yellow bars) to pause the program. Now let's set a breakpoint for the first line of the function **toggle_LED** so we can see the default values of the local variables and memory. To do this go to the window displaying you code and right click on the line you wish to have the breakpoint. Make sure you right click on the actual line number and not inside the code. Select "Toggle Breakpoint". Do this again for the last line of the function.



To view the values stored in memory click on the tab labeled "Memory Browser" (if no such tab exists you can bring it up via "View→ Memory Browser" from the menu). In the dropdown select "IO". In the field to the right of that dropdown, type in the address of LED_OUT1 (don't forget the 0x before the value because this tells the program the address is in hex instead of decimal.)

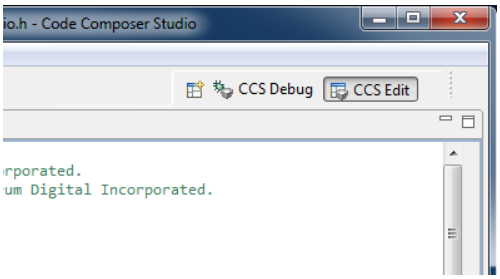
Now click on the green arrow again to run the program. It won't stop until you function is called so toggle LED 3 from the console window and wait for CCS to pause the program (it should not take long.) Set the "IO" memory location to 0x1c0a. Then press the green arrow again. You should see that all of these values have changed. If you find yourself juggling windows, you may find it helpful to "detach" the memory browser and/or console window so you don't need to swap back and forth between them (right click on the tab and select "Detached".) Or you can drag the various tabs around the CCS main window to rearrange things however you like.

Q1. What values do you see at memory locations 0x1c0a as you toggle LED 3 on and off? Does toggling LED1 on and off cause that memory location to change? Why or why not?

Q2. What about memory location 0x1c0b? What values do you see there as you toggle LED 3 and off? Does that value change? Why or why not?

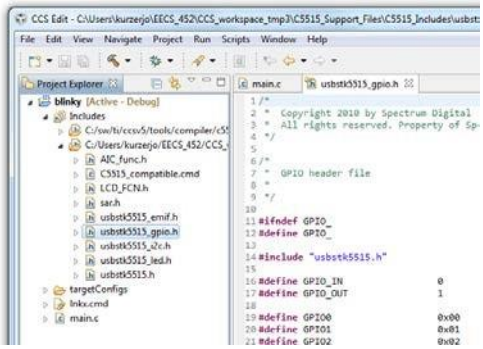
Digital Spectrum (the company that made the board) has provided several functions for us to use so we don't always have to manually manipulate the MMIO.

To find these functions first terminate our run by clicking the red square next to the suspend button. This should automatically switch you back to the "CCS Edit" perspective. You can verify which perspective you are in by looking at the icon labeled "CCS Edit"[4] in the upper right of the screen.



[4] This will sometimes be hidden, it's in the tab on the upper-right, you may need to click on a ">>" icon.

Now in the Blinky project folder you see a folder labeled "Includes." Expand this section to show all the folders that hold your header files. Expand the folder "C5515_Includes". You will see all the header files in that folder. Now double click on the file named usbstk5515_gpio.h. This will open it up in the code viewing window. Notice the functions that are prototyped at the end of this header file. Search for the source code for these functions (either on the Internet or, ideally, on your computer) and figure out what they do.



Q3. Rewrite the code for the "**if(index == 3)"** case of **toggle_LED** using these functions rather than the way we did it. What would you say an advantage of using the pointers was rather than this library code? What is an advantage of using the library code rather than the pointers?

G1. Have your GSI test your working code.

Part 3: More MMIO using the LCD.

There are plenty of peripherals on the board in addition to the LEDs. In this section we will look at how to communicate to the small liquid crystal display (LCD). We will be using functions that were given to us to communicate to the LCD. There is a document which covers the LCD in detail that can be found with the rest of the lab documentation. It is the ***SSD1306 128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller*** (never mind that our LCD is 96 x 16, that's the controller they used).

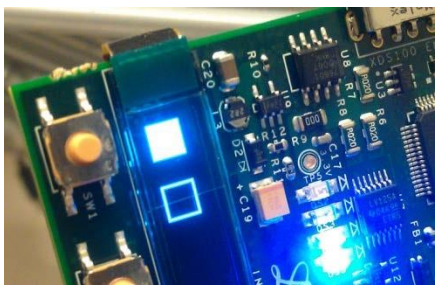
Copy your blinky project in the project explorer. Rename the copy "Square_Maker". Replace the main.c code with zigzag.c (found on the Ctools). The supplied code will produce a zig-zag line on the LCD. You will need to add one new file to the project: LCD_FCN.c, which is also available on the course website. Get this code up and running.

G2. Demonstrate the working zigzag code to your GSI.

Q4. The C5515 has a built-in LCD controller. The folks who designed the C5515 eZDSP Stick chose not to use that built-in controller and instead used an external I2C controller.

a. Explain, in a few sentences, what I2C is.
b. Look at the **OSD9616_send** function. What does it do?
c. In main.c what do you think the purpose of the "top" and "bottom" arrays are? Try to clear (turn off) the entire display. Try to make the entire display turn on.
d. In the prelab you were asked to draw the zigzag figure by hand. Does your answer match what was displayed? What's different if anything?
e. The display is only 16x96 even though the code might lead you to believe it is 16x128. Which parts of top and bottom aren't being displayed?
f. After the printf statement there are a bunch of **OSD9616_send** function calls. What do you suspect their purpose is?

G3. Modify this sample code and make two squares appear anywhere on LCD. Both squares are to be 16x16. One should be with filled in and the other just an outline.



They should not be touching. Something like the image to the right. Demonstrate this working code to your GSI.

Part 4: Generating and working with basic waveforms

Now that you have a basic understanding of how the LCD works, let's work with real signals and try to display something a bit more interesting. For this portion of the lab you will import a pre-made project. This project takes keyboard input and uses that to decide what to display on the LCD. You can display either a 1 KHz sine wave, or you can display an external input. If you push both buttons, you'll be again prompted to choose what to display.

In the Lab1 files from the course website, there is an archived project named "Audio_Project.zip". To import this project go to "File->Import…". Expand the "Code Composer Studio" selection and select "Existing CCS Eclipse Projects" and click "Next." Choose the "Select an archive file" option and "Browse" to the Audio_Project.zip file. Click Finish. Now build and run Audio_project.

You should get a prompt asking you what you want to display. Choose "0" and you'll get a simple sin wave. Your other options are to see the input from the left or right stereo input. The code grabs its data once you make a selection and then displays it. **It does *not* continually update, the data is just sampled once and thrown in a buffer.** Next, use the function generator to generate a 1.5 KHz sign wave, centered at ground and with a Vpp of 1.4V. Display this output on our oscilloscope. (You will want the function generator's output load set to Hi-Z by pressing "Utility" → "Output Setup" → "High Z" → "DONE".)

Connect the function generator to the "stereo in" of the board using either the left (white) or right (red) channel. Ask your GSI for help if needed.

Q5. At what voltage levels does our converter start saturating? (Do not drive more than 4V peak-to-peak to the board please; it can likely handle it, but…). Don't worry about being overly precise (within 0.2V will be fine).

Now modify the provided code so that if "0" is selected, you are again prompted to choose between displaying the following frequencies: 500Hz, 750Hz, 1kHz, 2kHz, and 3.5kHz. Your code must use the same sine table. This means either repeating or skipping entries in the sine table when creating the array to be displayed. Spend a few minutes thinking about how to efficiently implement this code without using division or mod. While efficiency isn't really important here as this operation isn't done continuously, it's good practice. Bit shifting is the key here.

G4. Implement these changes in main.c under "**if (mode == 0)**" code. Show your results and code to the GSI. Your code should be reasonably efficient.

WHEN THINGS GO WRONG:

Here is a list of fairly common issues you might encounter.

- If there is a compilation error regarding unresolved symbols it may mean that your include files or paths are wrong.

- If an error occurs during linking then your file search path in the Linker options doesn't include a folder with that symbol in it.
- If there is an error about the support library not supporting model 3, check that you set the memory model and silicon revision correctly.
- If your program fails while running check your console. If you see a warning or error message saying one or more sections fall into non-writable memory, then go into your Linker Options->Runtime Environment and switch it from ROM to RAM. You may still get the warning but it probably won't impede you program's performance.
- If you get a linker error stating the maximum space for local variables is exceeded, then move your declarations of your arrays outside the function. This will make them global variables and cause the compiler to store them in a different place.
- If you get a runtime error, restart code composer.
- If you can't get DEBUG working, right click on the project folder and chose "Open Target Configuration". From there you are looking for the XDS100v2 USB Emulator as the connection type and the USBSTK5515 as the device.

4. Post-Lab

Q1. Go to the C5515_Lib folder and examine the function in usbstk5515_led.c which initializes the ULEDs. Copy that function and describe what is happening line-by-line for that function. You will need to look at the #defines in usbstk5515.h and usbstk5515_led.h file to truly understand what's going on there.

Q2. Go into the AIC_func.c file and provide a line-by-line description of the **AIC_write2** function.

Each group should hand-in the following material, neatly stapled:

- Your sign-off sheet. It should be on the front and include each partner's name and unique name.
- A typed set of answers to the questions from the in-lab and post-lab. If figures are required, neat, hand-drawn, figures are acceptable.
- The portion of the code you modified to generate the different frequencies in part 4. Don't provide the whole file please; just the section you changed.

EECS 452 Lab 2—Basic DSP Using theC5515 eZDSP Stick

The purposes of this lab are to:

- Provide an platform for using direct digital synthesis (DDS)
- Provide an experience of basic FIR design and implementation for a DSP.
- Learn how to measure CPU utilization and how to judge if you are meeting "real time" constraints.
- Reinforce the introduction to embedded systems and memory-mapped I/O devices from the previous lab

1. Lab Overview and Background Material

In the first part of this lab, you will explore a better way to implement DDS. The way we generated waveforms in the previous lab was slow and memory expensive. In this lab you will implement an algorithm which is faster, more memory efficient, and allows for dynamic frequency scaling.

In the second part of this lab, you will be designing and implementing an FIR filter. For the pre-lab, you will use Matlab's fdatool to find the coefficients for this filter. We will use this tool because finding more than five or ten coefficients for an FIR filter by hand would be very time consuming, but we will be making a sixty tap filter (filter with sixty coefficients). Understanding how to use fdatool to design filters is the easy part, you will still need to implement the filter which is no easy task; people get paid to do nothing but that. You will explore the various means of doing an array-based implementation of an FIR filter, and understand the trade-offs associated with them.

2. Pre-lab

The pre-lab will prepare you for both the DDS and filter design parts. In this pre-lab you will:

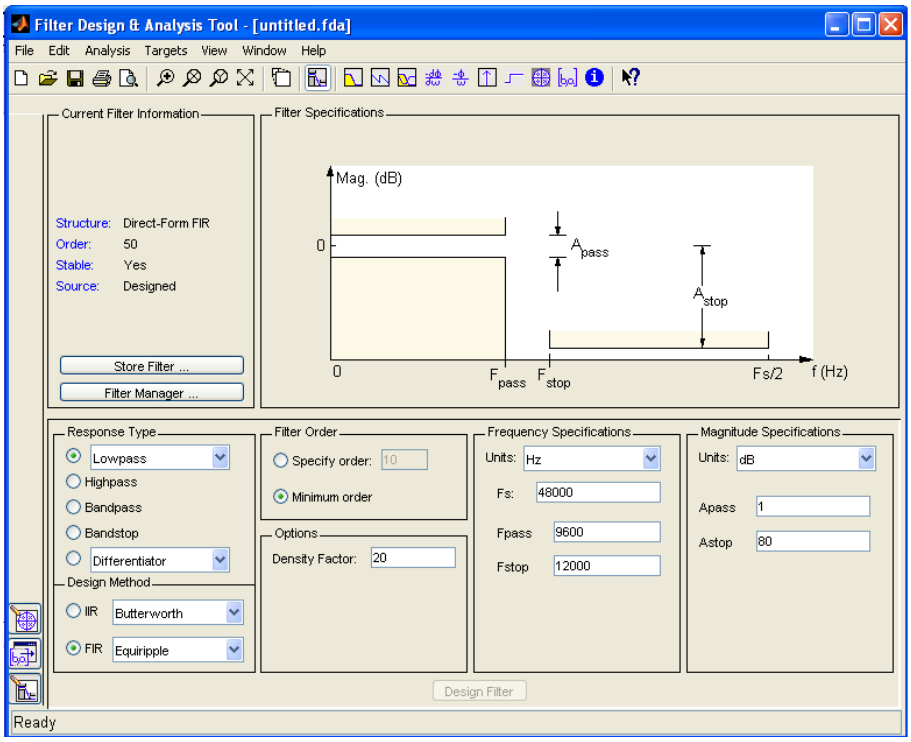
- Get the coefficients for an order 60 FIR filter using "fdatool" in Matlab.
- Read and understand DDS code.
- Examine and designing a *very simple* anti-imaging filter for use with both the FIR filter and the DDS outputs.

FIR filter

We are going to use Matlab to generate an FIR filter because it has the tools to make a filter with hundreds of taps in seconds, where doing by hand might take hours. One issue with our documenting this process is that Matlab literally changes once or twice a year. This means that the "look and feel" of the tools you are using may be different than shown below. It also means that different locations may be running different versions.

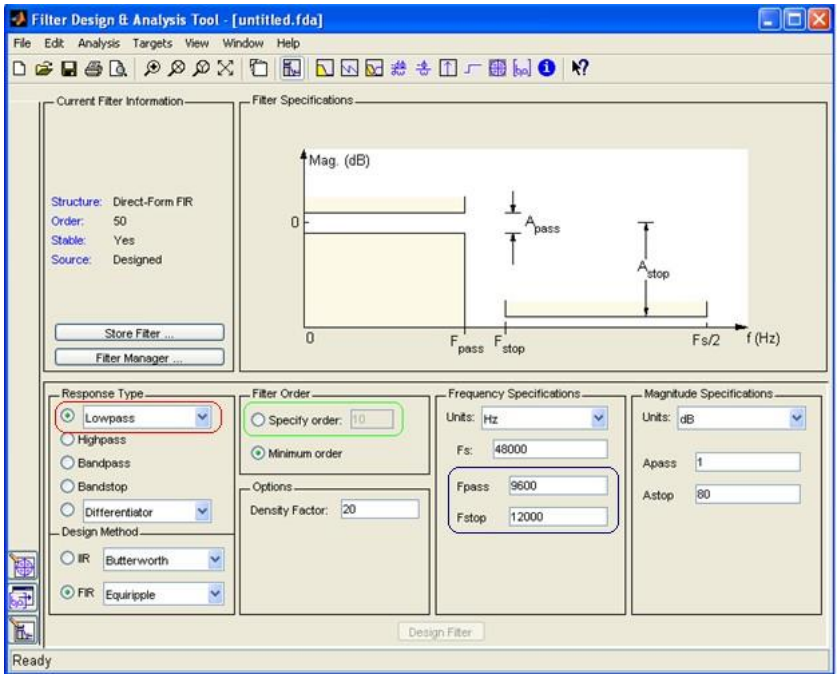
With these caveats in mind, let's start the task of designing an FIR filter in Matlab. Open up Matlab on your computer. Type the command "fdatool" into the command window. You should get the following window on your screen.

From here, you can design an FIR or IIR filter via an interactive GUI. We are going to arbitrarily make a low-pass filter first. Go to the left hand side of the window and select "lowpass" in the "Response Type" section, which we have circled it in red below. Make sure the bubble is green and the text field reads "Lowpass".

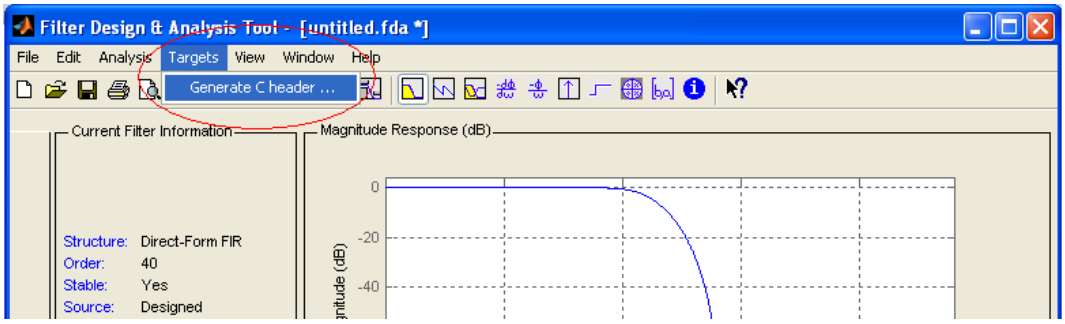


Now you *could* let Matlab decide how large to make your filter by selecting the "Minimum order" option in the Filter Order section of the window, just right of the Response Type window. However we will specify the order. **To do this fill in the bubble next to Specify order in the Filter Order section we circled green below, and type 60 into the field next to it**.

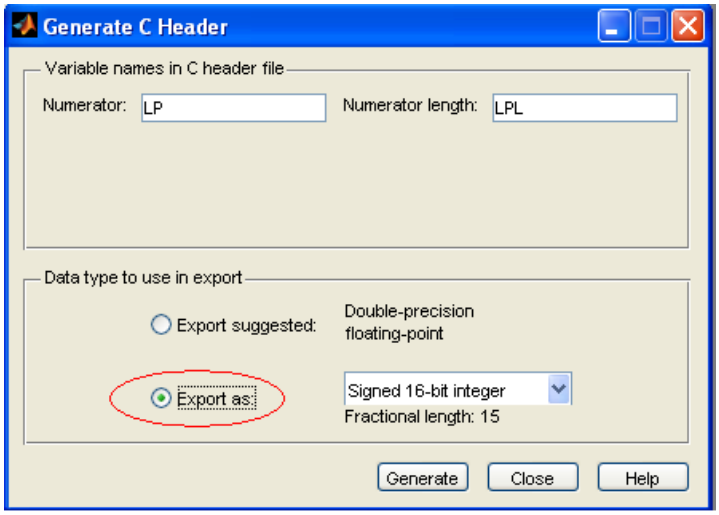
Now we have to choose the transition band of this filter. To do this we will manipulate the values in the "Frequency Specifications" section. We will leave the "Fs" field alone because we will be sampling at 48kHz, which is the default. But we will change what Fpass and Fstop are. **Change Fpass to 9000 and Fstop to 12000 (circled in blue).** Press the "Design Filter" button on the bottom of the window and Matlab will calculate the filter coefficients.



Now that Matlab has generated our FIR filter, we need to export the coefficients so we can use them in a C program. Because it is so common to want to implement filters in C, Matlab has a handy mechanism for generating the output as a C-header file. Select "Targets" and then "Generate C header …" as circled in red below.[1]



You should then get the following window. Please change the "Numerator:" field to "LP" and the "Numerator length:" field to "LPL". These are just the variable names Matlab will use. Later in the lab we will be adding more filters to our processor, so we've chosen these names for the values associated with the low pass (LP) filter. You now need to export these coefficients as sixteen-bit signed integers. To do this, select the "Export as:" option at the bottom on the window. Then click the down arrow and select Signed sixteen bit integer.



Now click the "Generate" button to complete the process. You will be asked where you want to put your new file. You may want to make a new folder on your desktop to accommodate these header files. Rename the file to low_pass.h and click Save.

The code that Matlab generates needs to be modified a bit. **You need to delete the #include near the top of the file**. Also change the "**const int LPL = 61**;" into a "**#define LPL 61**" and the "**const int_16**" next to LP on the next line into "**Int16**". Save your changes. Do not add a semicolon after definitions. They will throw compiler errors.

Q1. Print the file you generated (and modified) as described above.

Q2. The default structure of the FIR filter is "Direct-Form FIR"

a. How is this different from the Transposed-Form?
b. Draw the Direct-Form and Transposed-Form FIR. What do you think the advantage of each structure is?

[1] With this scheme your header file will likely contain a warning about rounding issues for the coefficients. For the most part, the rounding issues aren't going to be relevant for FIR filters or even most IIR filters. But they can cause instability in IIR filters that have poles near or on the unit circle as the rounding could push them outside of the unit circle.

Q3. Answer the following questions assuming you have the FIR filter you just created using Matlab.

a. What is the expected *group delay?*

b. What is the expected *phase shift* in a 1 KHz input?

c. What is the lowest frequency where you would expect to see a 180 degree phase shift?

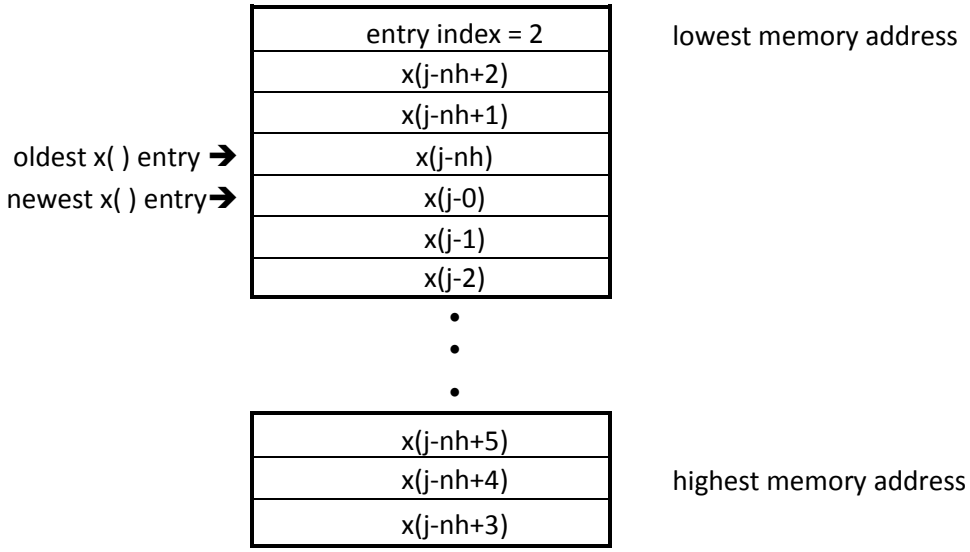
TI's DSPLIB FIR filter

Now we'll take a look at the FIR filter supplied by TI. It is a highly-optimized assembly language implementation. You can read about it in the C55x DSPLIB documentation. It is document spru422j (which can be found with a web search). The documentation starts on page 4-46.

Q4. In your own words, explain the role of each of the arguments to **fir()**.

Figure 4-16 in the C55x DSPLIB documentation describing the dbuffer of the **fir()** function is wrong. Make sure you understand why it is wrong. The corrected one is shown below.

Q5. Redraw the figure assuming index= 4.



Q6. Let's suppose we have a buffer `in` to store audio samples, and this buffer has 256 entries. One way we could index the buffer is to use if statements and reset the index to 0 if it exceeds 255. How could we do this using bit-masking instead? Right one or two lines of code to accomplish this.

DDS

Q7. Read the in-lab section associated with DDS. Consider the figure found in that section with the caption "DDS block diagram". If the register were 16 bits but only the top 6 bits were used to index the sine table, what would be the output frequency of the generated sine wave if f_s were 48 KHz and FTV=2? Show your work.

Q8. If the sine table we are using doesn't have 64 entries and instead has only 48 entries, we can't use all the possible values of the register. *Given that*, what would be the output frequency of the generated sine wave if f_s were 48 KHz and FTV=2? Again, show your work.

Analog filter and other background

The AIC3204 doesn't come with an analog filter because it outputs information meant for stereos or headphones which act as their own low-pass filters. But we are going to need to make our own low-pass filter for use as an anti-imaging (or reconstruction) filter.[2]

Q9. Design a first-order passive low-pass filter. Assume you have only a 10k Ohm resistor and you want the "3dB-down" frequency to be 30 kHz. Assume you have any capacitor value you need.

a. Draw a picture of the circuit and label the values of the components.[3]
b. If you were to use a 10 kHz unit-amplitude sine wave as an input, what would you expect the magnitude of the output to be?
c. As above but for a 30 kHz sine wave? A 100 kHz sine wave?

Q10. Consider a stereo tip-ring-sleeve (TRS) connector. Draw a diagram showing what each part of the connector is generally used for. Cite your source(s).

[2] http://en.wikipedia.org/wiki/Reconstruction_filter provides an overview/reminder of what this is.

[3] http://en.wikipedia.org/wiki/RC_time_constant and http://www.electronics-tutorials.ws/filter/filter_2.html may prove useful if you've forgotten how to do this kind of a thing.

3. In Lab:

As noted above, this lab has two parts, the DDS material and the FIR filter implementation. In addition, you will need to build a simple analog low-pass filter.

Using Starting_Point.zip

Unlike in lab 1, we will have you start by copying a "starting point" Code Composer Studio project from the course website. If you set things up correctly in lab 1, you should find a file named "Starting_point.zip" in your workspace inside the C5515_Support_Files. This should correctly set your paths and take care of all the other set-up stuff done in part 1 of the in lab portion of lab 1. If you encounter problems with this starting point you can instead redo the work you did in lab 1 or copy your "blinky" project and start from there. All three mechanisms can be made to work for you, but we recommend using the provided starting point.

Find the "Starting_Point.zip" file (do not unzip) in the C5515_support_files. In CCS go to "File→Import…" select "Code Composer Studio" → "Existing CCS Eclipse Projects" and click "Next >". Now next to the "Select archive file" option browse for the Starting_Point.zip file. Click "Finish."

Rename the project to whatever it is you want it to be called and start coding from there.

Part 1: Analog Filter and other background

Build the analog filter you designed in the pre-lab. Obviously you'll not be able get an exact value for your capacitor, so find the closest value you can. You may want to have your GSI check your circuit.

Q1. What value capacitor did you choose?

Q2. Input unit-amplitude 10, 30, and 100 KHz sine waves. For each input, list the output amplitude. How well does this match with your pre-lab answers?

Part 2: Direct Digital Synthesis (DDS)

Recall that in lab 1 you wrote code which generated sinusoids of certain frequencies. Now we are going to do true DDS (Direct Digital Synthesis). The idea is expressed with the following graphic.

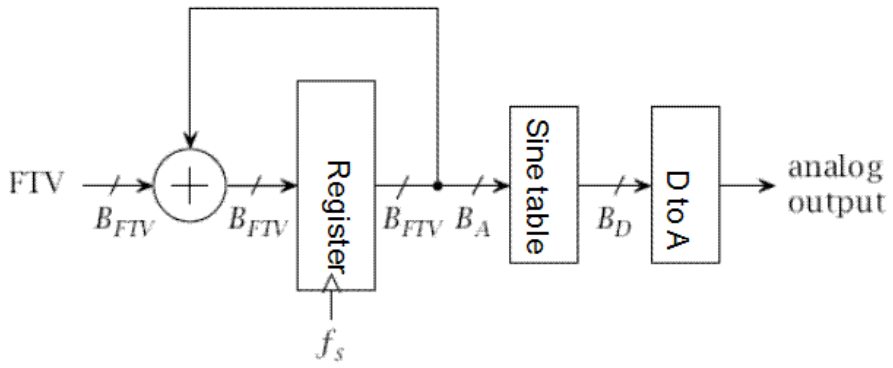


Figure 1: DDS block diagram

Using DDS to create a sinusoid consists of three steps. First a frequency tuning value (FTV) is selected. This is determined by the desired frequency of the sinusoid and will be held constant until we want to change frequency. Second, on each rising edge of the clock (which has a frequency of f_s) the

accumulator register is incremented by the FTV value. Finally, the top "A" bits of the register are used to index a sine table and the value from that table is provided to the digital to analog converter.

Your job is to get this code to output an arbitrary sinusoid by only changing the " blank " and by adding code where it says "//add your code here" (though feel free to add additional variable declarations as needed). You should only need to add a few lines of code.

Q3. If f_s=48 Khz and we are using the six most-significant digits of a 16-bit number to index our 64-entry table, what frequency sine wave would we expect to get if FTV=1? If FTV=1024? Explain your answers.

Connect the "stereo output" of the eZDSP to the oscilloscope and complete the following code (Do not copy from below. The code can be found on the lab2files on the course website).

```
/*
 * dds_true.c
 *
 *     Author: GSI
 */

#include <usbstk5515.h>
#include <usbstk5515_i2c.h>
#include <AIC_func.h>
#include <sar.h>
#include <stdio.h>
#include <math.h>

#define TABLE_SIZE  64
#define fs 48000.0

Int16 sinetable[TABLE_SIZE] = {
              0,3212,6393,9512,12539,15446,18204,20787,
              23170,25329,27245,28898,30273,31356,32137,32609,
              32767,32609,32137,31356,30273,28898,27245,25329,
              23170,20787,18204,15446,12539,9512,6393,3212,
              0,-3212,-6393,-9512,-12539,-15446,-18204,-20787,
              -23170,-25329,-27245,-28898,-30273,-31356,-32137,-32609,
              -32767,-32609,-32137,-31356,-30273,-28898,-27245,-25329,
              -23170,-20787,-18204,-15446,-12539,-9512,-6393,-3212
};

void main(void)
{
       Uint16 FTV;
       Uint16 counter,temp, out;
       float freq;
       USBSTK5515_init();
       printf("What frequency do you want?\n");
       scanf("%f", &freq);

       FTV =   blank   ;

       printf("FTV value corresponding to %6.2lf HZ should be %d\n",freq, FTV);
       AIC_init();
       counter = 0;
       while(1)
       {
```

```
                // add your code here
                out = sinetable[temp];
                AIC_write2(out, out);
        }
}
```

You should notice that your signal is a bit "bumpy" and if you generate the signal at a fairly low frequency (say 10 Hz), the steps are really obvious. Insert the hardware low-pass filter you've built between the stereo output and the oscilloscope.

Q4. What impact, if any, does the filter have on smoothing out the signal?

More DDS: Dynamic arbitrary frequency

Now let's do this for dynamic arbitrary frequency. We will ask for this dynamic change by pushing the buttons on the board. To do this you are going to:

- Copy sar.c from the C5515_Lib->C_code folder into your project. You can do that by right-clicking on the project in the Project Explorer and selecting "Add Files…".
- Add the following code to your dds_true.c in the while(1) block.
- Change the blank entries just as you did before.
- Declare "key" as a Uint16.
- Add the line "`Init_SAR();`" right after the "`AIC_init();`" line.

```
//Check the SAR for FTV changes
key = Get_Key_Human();
if(key == SW1)
{
     temp=FTV<<1;
     FTV=temp>0x1000?0x1000:temp;
     printf("Frequency=: %6.2lf HZ\n",  blank  );

}
else if(key == SW2)
{
     temp=FTV>>1;
     FTV=temp==0?1:temp;
     printf("Frequency=: %6.2lf HZ\n",  blank  );

}
```

G1. Demonstrate that your code works and you can scale the frequency of the output.

G2. Say that the signal you are generating above is named f(x). Modify your code so that you are outputting f(x) +g(x) where g(x) is a 1 KHz sine wave. Scale the output so that there is no overflow or clipping and demonstrate your code. Be careful with parenthesis.

Part 3: FIR filter

Re-import and rename the starting point project located in the C5515 support files. Call this Project "FIR_filter".

Hook up your USBSTK5515's "stereo in" to the function generator. Be sure that the function generator's output is going into the red cable in the stereo adapter cable; also be sure the function generator has its output set to "High Z" (it's under "utility" then "output setup" on the function generator). And be sure to hook up the red cable of the stereo out of the USBSTK to the oscilloscope if it isn't already because we will verify the functionality of our filter with them. Now after that is completed add fir_filter.c (found in the lab2files on ctools) to your project. *Add to project the file that houses the FIR low-pass filter you made in the pre-lab and rename that file to be "low_pass.h".*

```
/*
 * fir_filter.c
 *
 *      Author: GSI
 */

#include <usbstk5515.h>
#include <usbstk5515_i2c.h>
#include <AIC_func.h>
#include <stdio.h>
#include "low_pass.h"

#define ASIZE         61

#define TCR0          *((ioport volatile Uint16 *)0x1810)
#define TIMCNT1_0     *((ioport volatile Uint16 *)0x1814)
#define TIME_START  0x8001
#define TIME_STOP   0x8000

Int16 in [ASIZE];
Uint16 delta_time;

Int16 FIR(Uint16 i)
{
      Int32 sum;
      Uint16 j;
      Uint32 index;
      sum=0;

      //The actual filter work
      for(j=0; j<LPL; j++)
      {
            if(i>=j)
                  index = i - j;
            else
                  index = ASIZE + i - j;
            sum += (Int32)in[index] * (Int32)LP[j];
      }
      sum = sum + 0x00004000;       // So we round rather than truncate.
      return (Int16) (sum >> 15); // Conversion from 32 Q30 to 16 Q15.
}

void main(void)
```

```
{
	Uint16 i;
	Uint16 start_time;
	Uint16 end_time;
	Int16 right, left; //AIC inputs
	Int16 out;

	USBSTK5515_init();	//Initializing the Processor
	AIC_init();		//Initializing the Audio Codec

	//Priming the PUMP
	for(i = 0; i < ASIZE; i++)
	{
		AIC_read2(&right, &left);
		in[i] = right;
	}
	while(1)
	{
		if(i>=ASIZE) i=0;
		AIC_read2(&right, &left);
		in[i] = right;
		out = FIR(i);

		//POSTFILTER:
		AIC_write2(out, out);
		i++;
	}
}
```

The above code uses a circular buffer of 61 entries to house the samples of the incoming waveform. The variable 'i' houses the current location of the most recent sample. This is passed to a function that runs the low pass filter you designed in the pre-lab. Then it returns the sum. The function checks for wrap around on every iteration of the for-loop and compensates when necessary. After the filter there is a rounding estimate and a conversion from 32 bit Q30 to 16 bit Q15.

If you build and launch this project now your project will produce a result that is different than what you designed. Confirm this on your own.

G3. Get the un-optimized code running. Does your filter even work? Show your GSI how your filter behaves at a number of frequencies.

The problem is that your filter is running too slowly and it can't keep up with the real-time demands of the 48 KHz sample rate. So we need to optimize our code a bit. Right click your project and select Properties. Right-click on the project and go to "Properties". Go to the "Build" and then "C5500 Compiler". Go to "Optimization" and change the "Optimization level" to 3 with the drop-down box.

This will optimize your code for speed and will let the filter meet its time constraints[4]. Now build and launch this code. Confirm that the filter behaves as expected.
Note: The output may be inverted.

Q5. While we've written the code for you, you will need to write similar code later. As such, let's examine a few different parts of this program and try to understand them.
   a. Consider the FIR function. What is the array "in"?
   b. In the FIR function, what role does "**sum = sum + 0x00004000;**" play in rounding?
   c. In the while(1) loop, what role does "i" have?
   d. What "pump" are we "priming" with the first for loop in the main? Explain what (if anything) would happen if we removed that code.

Q6. In the pre-lab, you were asked to predict the group delay. What group delay do you actually see? How did you go about measuring it?

One reason that the group delay is not what we expect is that our AIC3204 (the audio chip) is an extremely complex device. It has a very complex set of digital filters itself (including a 20 and 25 tap FIR filter, 5 biquad blocks (capable of implementing a fairly complex IIR filter) plus three different types of decimation filters. And all that's just for the ADC, the DAC has similar features (use page 45 of the AIC manual as a starting point if you want to read more). The net effect is that the AIC3204 has a lot of delay associated with it.

G4. Modify the filter so that one channel of the output is just putting the input right back out while the other is the output of the filter. This should let us see the delay associated with *our* filter.
   a. Use the function generator to generate a signal where you are sure you are seeing the group delay.
   b. Find and display the lowest frequency of the sin wave where the phase shift is 180 degrees (not aliased to that…)
   c. Recall that for our type of filter, the group delay is constant. Can you think of another way to measure group delay when it is not dependent on frequency?

Q7. What was the group delay? What was the lowest frequency at which the phase shift was 180 degrees?

Our design is nice and all but an order 60 filter (61 taps) isn't very impressive: we often want filters with 100 or more taps. But we need to know how big we can make it without running into CPU limits. So let's figure out how to measure that. What we'll do is poll a register that counts clock ticks before and after each new input is filtered.

You may want to save the code you used above under a different name before making any changes (perhaps copy the project or just the fir_filter.c as you see fit.) Once you've saved what you want, replace your **while(1)** block with the following code:

```
TCR0 = TIME_STOP;
TCR0 = TIME_START;//Resets the time register
while(1)
{
```

[4] Do not go to the "Advanced Optimizations" section in the "Advanced Options". Those optimizations do not do anything because those flags don't seem to be recognized by the compiler.

```
            if(i>=ASIZE) i=0;

            AIC_read2(&right, &left);
            in[i] = right;
            //Measuring the time of the filter chosen
            start_time = TIMCNT1_0;
            out = FIR(i);

            end_time = TIMCNT1_0;
            delta_time = (start_time-end_time) << 1;
            //Take care of the prescalar factor
            //POSTFILTER:
            AIC_write2(out, out);
            i++;
    }
```

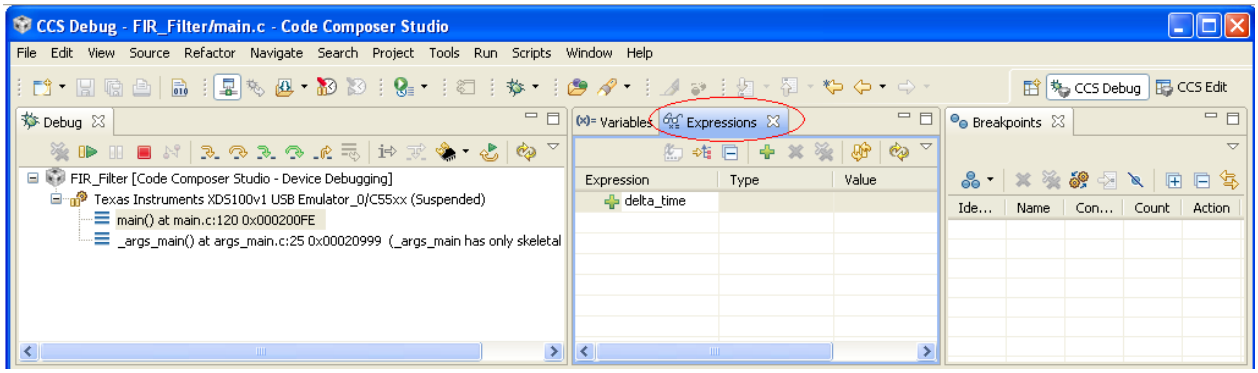
Here we are using a built-in counter which counts down and has a prescalar[5] of two. That means that the start time has a higher value than the end time and that every one count of the counter is two clock ticks.

Q8. Answer the following questions on filter timing.

a. If we have a 100MHz processor and we sample at about 50kHz (makes the math easier) what is the maximum number of cycles that our filter can use for each input before we need to worry about not being able to meet our real-time constraints?

b. How many cycles does our current filter take? To measure this, click the expressions tab, which is circled red in the following graphic and type into the text field delta_time. Launch the program then pause it to read delta_time. You can also toggle a break point at the beginning or end of a loop you're interested in so you're not randomly guessing where you've stopped the code.

c. Given the above, how large do you think we could make the filter before we started to run out of CPU time? (Hint: Try to half/double the length of the filter and see whether you half/double the CPU cycles, roughly. Then you have to consider the cycles taken by AIC_read2/AIC_write2.)



That doesn't seem very useful. What if we did indexing a bit differently? How else could we index into the array of samples? We could use the mod function. Let's try it and see if it helps. Replace the current if else statements in the "for loop" of our low pass filter with:

[5] A prescalar divides the clock before counting. In this case it divides the system clock's frequency by 2. See http://en.wikipedia.org/wiki/Prescaler

```
index = ASIZE + i - j;
index = index % ASIZE;
```

Q9. Now build and launch this code and measure the latency of filtering each sample. How does this compare to what we measured above? Why is that?

How can we do better than our original code? One answer would be to use a technique called bit masking. To set up our system to bit masking, we first must make the sample buffer size the next power-of-two greater than our filter's width. In this case 61 would round up to 64. Make the following changes:

- Change ASIZE to be defined as 64
- Make a new global variable called "**mask"** of type "**const Int32"** and set it to be **ASIZE–1**.
- Now replace our indexing code with the following.

```
index = ASIZE + i - j;
index = index & mask;
```

Q10. Now build and launch this code and measure the latency of filtering each sample. How does this compare with our base measurement?

Why is this faster than both of our previous indexing protocols? To answer this we would need to look at the assembly code generated from our C code. Looking at the assembly allows us to see what assembly instructions our code becomes when it is compiled. This is useful because it allows us to know exactly the processor is doing with our code and data. Unfortunately, for those of you without EECS 370 reading the assembly code is unlikely to be useful, and even for those of you who have taken EECS 370, learning a new assembly language isn't trivial. That said, we want to expose you to the idea of assembly code.

While in the Debug perspective, with the code loaded on the board, select "View->Disassembly". It will make a new window for you to view the assembly language of your code. Find the assembly code associated with the current indexing scheme.

Q11. Pick 3 of the assembly commands you see in the FIR filter function (such as MOV and SUB) and explain what they are doing. Also indicate how many cycles each of those instructions take. You will need to use the C5515 Mnemonic ISR reference (SWPU067E).

Ideally we'd be able to look at the assembly and figure out what's going on. But doing so requires a pretty solid background in assembly language. You should be well aware of this option as there will be times that looking at the assembly code will be useful. But for now, let's just think a bit about the various schemes.

Q12. What are the advantages of using the if/else over the bit masking? What are the limitations of bit masking?

Q13. Using the fastest indexing protocol we have, about how many taps can we have in our filter?

G5. Redo the filter using as many taps as you can. Must go back to Matlab and generate a filter of the correct order. Upload this filter to the C5515 and run the program.

This still isn't very impressive. It turns out that if we were to write this code in assembly and optimize it, we could do a lot better. Thankfully, TI provides a DSP library that does just this for many common types of filters and we will try the FIR filter function in the next section.

Q14. What is the CPU latency per sample with your new filter?

G6. Rewrite the code of your order 60 FIR filter using **Transposed-Form FIR**. Have your GSI check both the code and the functionality.

Part 4: TI's FIR filter

Import the Starting_point project and rename it to TI_FIR. Be sure optimization is set to 3.

Q15. Look at the libraries that are included in the project (right click on project in project window, go to "Properties". Then select "Build→ C5500 Linker"). Look under the "File Search Path". What library do you think is associated with TI's DSPlib?

Use Matlab to generate a low pass filter that has a Fpass of 2000 and Fstop of 3000 with Fs=48000. Instead of specifying an order, choose the "Minimum order" option. Generate a C header file and modify it as needed so you can use the coefficients with the **fir()** function. Name the coefficients file "low_pass1.h". Make sure your header file declarations are consistent with the variables in TI_FIR.c. Make sure the header file is located in one of CCS's search paths.

You will find the code below on the course website in the lab2_files named "TI_FIR.c". Note that the header for the DSPlib functions is "Dsplib.h". Your **main()** should be as follows…

```
#include <usbstk5515.h>
#include <stdio.h>
#include <Dsplib.h>
#include <AIC_func.h>
#include "low_pass1.h"

void main()
{
	Int16 x[1];
	Int16 dbuffer[LPL+2]={0};
	Int16 r[1];
	Int16  left;
	USBSTK5515_init();  //Initializing the Processor
	AIC_init();  //Initializing the Audio Codec

	while(1)
	{
		AIC_read2(x, &left);
		fir(	x,		// input
			LP,		// coef
			r,		// output
			dbuffer,	// Z-1 blocks and more
			1,		// number to process
```

```
                                LPL          // number of parameters
                        );
                AIC_write2(r[0],r[0]);
        }
}
```

Connect things so that the function generator's output is being filtered by the C5515 then displayed on the oscilloscope. Have both the filter input and output displayed on the oscilloscope. Run the program. Instrument your code so that you can display how many cycles the function takes (similar to how we did it in part 3.)

Q16.

a. How many taps are there in your FIR filter?
b. How many cycles did the **fir()** function take to process a single input?

Do the same thing but try Fstop=2200. Now fdatool should design a 500+ order filter.

Q17.

a. How many taps are there in your FIR filter?
b. How many cycles did the **fir()** function take to process a single input?
c. Assuming there is some constant overhead associated with calling the fir() function and that runtime is otherwise linear, how many taps could you process in real-time? Show your work.
d. How does this compare to the code you wrote in the previous part?

G7. Demonstrate your 500+ tap filter to the GSI and show how you've instrumented the code to get the runtime information.

Be sure the oscilloscope is set up to view both the input and output of your 500+ tap filter. Set the function generator to sweep from 500Hz to 3.5kHz. Observe the amplitude of the output (relative to input) as a function of frequency.

G8. Demonstrate the sweeping frequency response to your GSI

Q18.

a. Sketch (roughly) the frequency response of your 500+ tap filter in the swept range.
b. How does your sketch compare to the frequency response that fdatool gave you when you designed the filter?

Note: in fdatool, right click on the Y-axis (where it says Magnitude) and chose the linear version (not dB). You can also zoom the X-axis to the same swept range for comparison.

4. Post Lab:

Q1. One hardware construct that is commonly added to DSPs, but not to "standard" processors, is "circular buffers". See http://www.dspguide.com/ch28/2.htm for example.

a. Explain why they would be helpful here.

b. The C5515 supports circular buffering. Look at TI's manual named "spru371f.pdf" and read section 6.11. Explain, in your own words, how they work.

Q2. Consider the following FIR implementation and compare it to the one in fir_filter.c.

a. What's the advantage of using this one? (Hint: consider the case where you need to use multiple different FIR filters at the same time.)

b. This solution would still make it hard to do a low-pass 100-tap filter and a 50-tap high-pass filter. Why is that and how could you fix the problem?

```
Int16 FIR2(Int16* inBuf, const Int16* taps, Uint16 i)
{
	Int32 sum;
	Uint16 j, index;
	sum=0;

	//The actual filter work
	for(j=0; j<LPL; j++)
	{
		if(i>=j)
			index = i - j;
		else
			index = ASIZE + i - j;
		sum += (Int32)inBuf[index] * (Int32)taps[j];
	}
	sum = sum + 0x00004000;	// So we round rather than truncate.
	return (Int16) (sum >> 15);  // Conversion from 32 Q30 to 16 Q15.
}
```

Hand-in list:

Each group should hand-in the following material, neatly stapled:

- Your sign-off sheet. It should be on the front and include each partner's name and unique name.
- A typed set of answers to the questions from the in-lab and post-lab. If figures are required, neat, hand-drawn, figures are acceptable.
- A printout of your code from G2 and G6. If you modified any header files, be sure to include them also.