

Universidad Tecnológica Nacional

Facultad Regional Paraná



Medidas Electrónicas II

“SÍNTESIS DIGITAL DIRECTA”

Profesores:

- Ing. Cappelletti Carlos A.
- Ing. Krenz Mónica F.

Alumnos:

- Bazzana Eric Martin
- Cancellieri Luciano
- Castelucci Leandro
- Schenone Leandro
- Sequeira Franco

Fecha de entrega: 06/11/2020

Resumen

En este trabajo realizaremos una búsqueda y análisis de información sobre Síntesis Digital Directa. Estudiaremos que es, su funcionamiento, especificaciones, diagramas de bloques y algunos productos disponibles en el mercado. Además, buscaremos implementar este método en algún microcontrolador para lograr generar distintas ondas periódicas de frecuencia variable.

Palabras clave: DDS - SDD - Arduino Due – Síntesis Digital Directa

Indice

Resumen.....	2
Indice.....	3
SINTESIS DIGITAL DIRECTA - TEORIA	6
¿Qué es?	6
¿Para Qué Sirve?	6
Diagrama de bloques.....	7
Principio de funcionamiento	8
Función de cada bloque	9
Acumulador de Fase	10
Tabla de Salida o Conversor de Fase a Amplitud	14
Conversor digital Analogico	14
Filtro Pasa Bajo.....	15
Desarrollo matemático	20
Fuentes de ruido y espuelas en DDS.....	25
Efecto de la resolución del DAC, error de cuantificación (eDA):	25
Los efectos del sobre muestreo en el rendimiento espurio	28
El efecto del truncamiento en el acumulador de fase (eP):.....	29
No linealidad del DAC:	31
Señales espurias a la salida del DAC:.....	31
Otras fuentes de ruido:	32
Especificaciones.....	35

Descripción de terminales del AD9850:	37
Aplicación del AD9850:	38
Otro modelo existente en el mercado.....	40
DDS Actuales en el Mercado	42
Módulo con AD9850.....	42
Módulo con AD9833.....	43
Módulo con AD9834.....	45
Ventajas y desventajas de los DDS	48
Algunas de las ventajas que poseen los DDS son:.....	48
Algunas desventajas que poseen los DDS son:.....	48
IMPLEMENTACIONES.	49
Generador de Funciones con Arduino Uno en Proteus.	49
Tabla de Memoria:	50
Código del Arduino:.....	52
Salidas Obtenidas en la Simulación:.....	53
DDS con Arduino DUE.	56
Objetivos planteados por el grupo:	56
Elementos Necesarios:	56
Características Arduino Due:	56
Temporizadores Internos y DAC en Arduino DUE:	57
Primeras Pruebas:.....	59

Código 2:.....	61
Código 3:.....	63
Implementación Final:	65
CONCLUSIONES GENERALES.....	77
Bibliografía	79

SINTESIS DIGITAL DIRECTA - TEORIA

¿Qué es?

La Síntesis digital directa (DDS) es una técnica empleada por sintetizadores de frecuencia para producir una forma de onda analógica, generando una señal variable en el tiempo en forma digital y luego realizando una conversión DAC (convertidor digital-analógico). Esto permite la resolución de frecuencia fina en una amplia gama de frecuencias y cambios rápidos entre esas frecuencias.

¿Para Qué Sirve?

La función básica de un sintetizador de frecuencia es generar más de una frecuencia utilizando solo un elemento determinante de frecuencia.

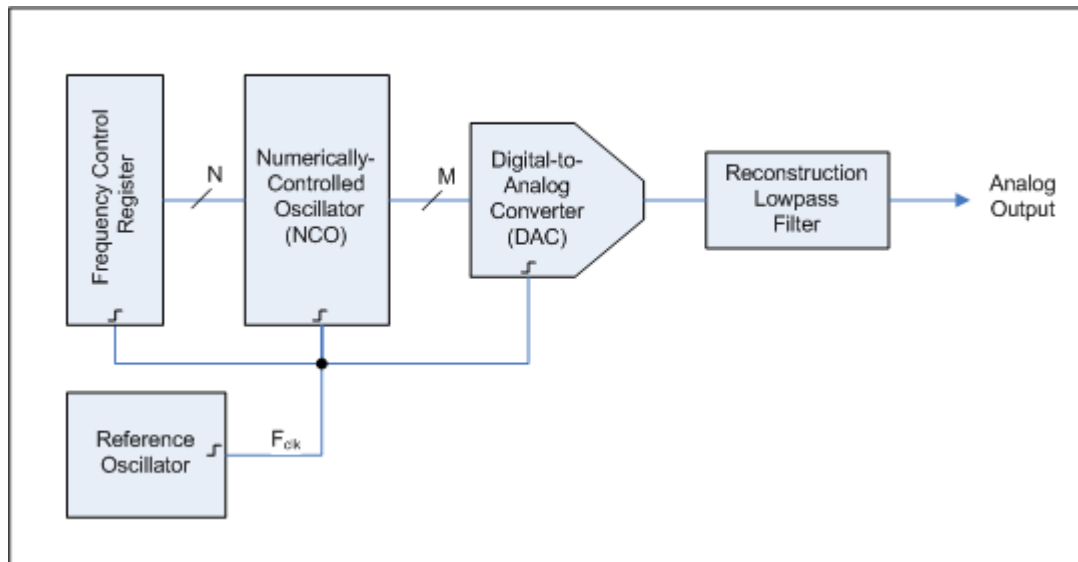
Partiendo de que algunos dispositivos como los excitadores de radiofrecuencia requieren varias frecuencias. Incluso si estas frecuencias no se requieren simultáneamente, cada frecuencia que no sea un múltiplo exacto de otra necesita su propio oscilador. La mayoría de estas aplicaciones multifrecuencia también requieren la capacidad de saltar entre frecuencias de forma rápida y dinámica, lo cual demanda un nivel de control que puede probar ser imposible para un sistema que debe seleccionar rápidamente entre múltiples cristales distintos.

Para ello se suelen utilizar sistemas de síntesis digital directa, debido a que son dispositivos utilizados para crear digitalmente formas de ondas de tipo análogas y frecuencias arbitrarias, tomando como base una sola señal de frecuencia fija.

Las características de los DDS permiten que este tipo de tecnología sea empleado en radares militares y sistemas de comunicaciones principalmente, aunque también incluyen una gran variedad de aplicaciones, como módems de cable, equipos de medición, generadores de formas de onda arbitrarias, estaciones base celulares y estaciones base de bucle local inalámbrico.

Otras aplicaciones que se destacan son equipos automatizados de pruebas para dispositivos electrónicos, hasta sistemas de comunicaciones inalámbricos o satelitales. Otras aplicaciones son sistemas de radar/sonar de arreglo de fase, despliegue de imágenes para la industria médica y sistemas ópticos de telecomunicaciones.

Diagrama de bloques



Podemos diferenciar los bloques en que está constituido un sintetizador digital directo básico en una referencia de frecuencia (un cristal o SAW oscilador), un oscilador controlado numéricamente (NCO) y un convertidor de digital a analógico (DAC) como se muestra en la figura.

El oscilador de referencia proporciona una base de tiempo estable para el sistema y determina la precisión de la frecuencia del DDS. Además le proporciona el reloj a la NCO que produce en su salida una discreta en el tiempo, cuantificado la forma de onda de salida deseada (a menudo un senoide) cuyo período está controlado por la palabra digital contenida en la frecuencia de registro de control. La forma de onda muestreada, digital se convierte a una forma de onda analógica por el DAC. El filtro de reconstrucción de salida rechaza las réplicas espectrales producidos por el de retenedor de orden cero inherente en el proceso de conversión analógico.

Principio de funcionamiento

El funcionamiento de la síntesis digital se basa en un acumulador de fase que genera una serie de estados digitales, cuyo valor aumenta linealmente formando una rampa numérica. Esta señal es periódica y representa la fase instantánea de la forma de onda de salida, que va desde cero a 2π radianes. Esta es la entrada digital a una tabla de búsqueda que convierte la rampa numérica en una onda sinusoidal (Figura 1). Si bien la forma de onda de salida más común de los DDS es la onda sinusoidal, las ondas de rampa, triangulares y cuadradas también se generan fácilmente.

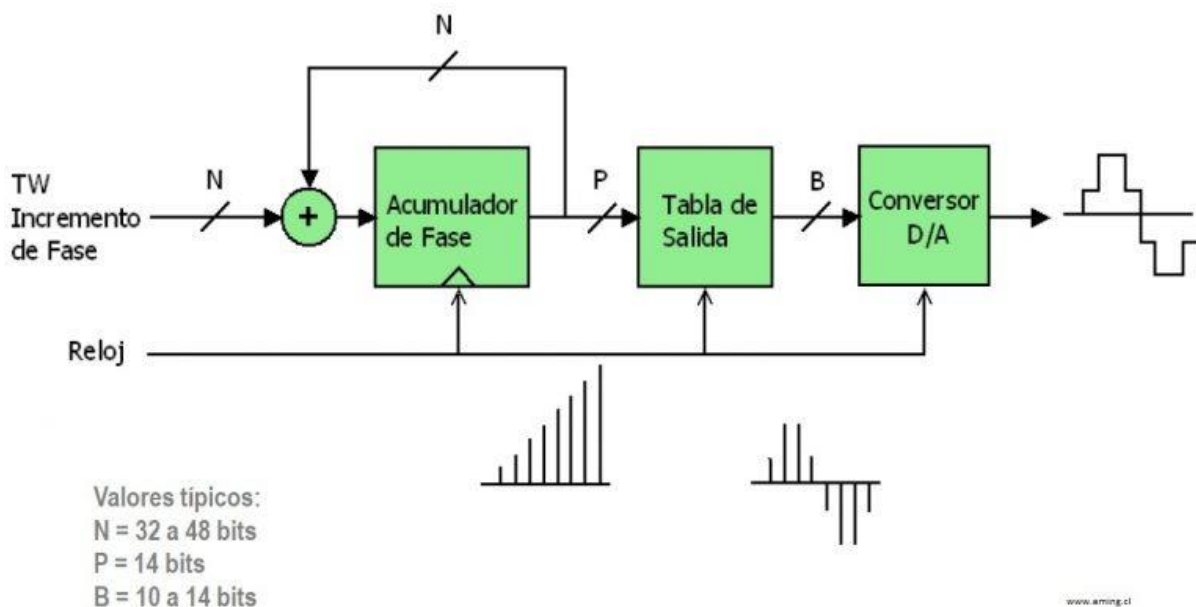
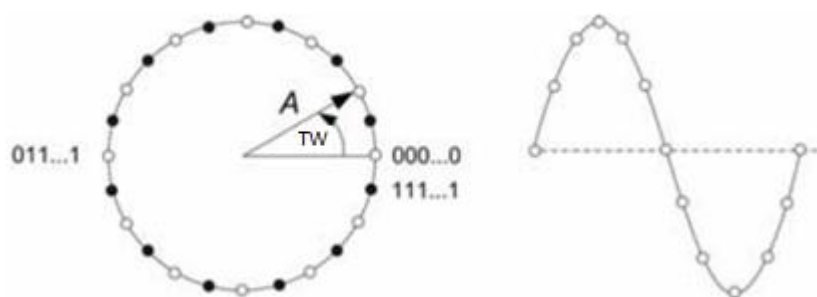


Figura 1. Esquema básico de un oscilador numéricamente controlado (NCO), bloque fundamental de un DDS.

La salida obtenida en la tabla de búsqueda de la conversión fase-amplitud se envía al DAC y se convierte en una forma de onda analógica, que generalmente es sinusoidal. Dado que la entrada al DAC consiste en una serie de valores muestreados, para la salida se necesitan pasos de cuantificación. Estos pasos producen imágenes espectrales en múltiplos de la tasa de muestreo del dominio de la frecuencia que no son deseadas. Un filtro de paso bajo, colocado después del DAC, suprime estas respuestas espectrales no deseadas.

Función de cada bloque

La síntesis digital directa (DDS) se basa en una tabla, la cual típicamente se encuentra alojada en una memoria ROM que guarda una secuencia de valores instantáneos, equidistantes, de la forma de onda deseada (normalmente una senoide), y un reloj de frecuencia fija que establece la cadencia de lectura de dichos valores. La figura muestra cómo se puede generar una senoide mediante un vector que gira describiendo un círculo. Si la longitud del vector es A , la altura de la punta del vector sobre la horizontal es $A \sin \phi$, de modo que cada punto del círculo corresponde a un punto concreto del ciclo de la senoide. Una vuelta completa alrededor del círculo con una velocidad constante genera un ciclo completo de la senoide. La frecuencia de la salida viene determinada por la forma como se lean los valores escritos en la tabla. Si por ejemplo se leen uno a uno, el tiempo necesario para que la salida describa un ciclo completo es N veces mayor que el que se tarda si se leen de N en N . La frecuencia de salida se selecciona digitalmente, a través de N , y conserva la estabilidad de la frecuencia de referencia ya que es ésta la que dicta el instante en el que se hace la lectura.



Un esquema básico de un DDS consta de un acumulador de fase (un registro digital) que determina cuál de las muestras instantáneas almacenadas se va a buscar, un convertidor fase–amplitud que obtiene el valor correspondiente y que está basado en una memoria ROM, un interpolador digital y un convertidor D/A.

Acumulador de Fase

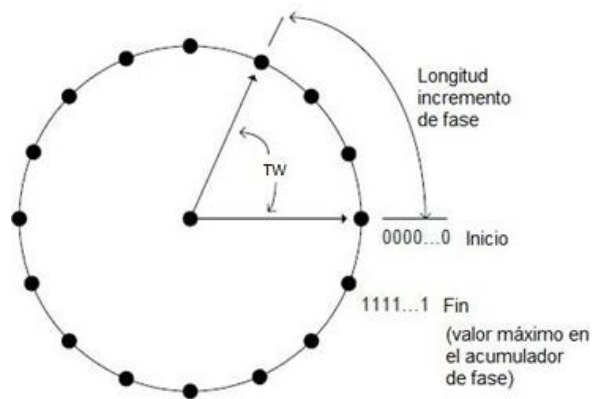
Su funcionamiento para el caso de una senoide es el siguiente: se calcula, periódicamente y en tiempo real, un ángulo de fase linealmente creciente, $TW = 2\pi \cdot f_s \cdot t$, donde f_s es la frecuencia deseada (seleccionada numéricamente en el panel frontal mediante N, que es una palabra digital entre 000...0 y 111...1) y t se mide en periodos T_r de la frecuencia patrón de referencia f_r , es decir, en el instante j, $t = j \times T_r$. De este modo a cada periodo de reloj la salida del acumulador se incrementa en $2\pi f_s$. Luego se buscan en una memoria los valores de $\sin \phi$. Las muestras de salida repiten su valor con una frecuencia que es la de rebasamiento del acumulador. Si éste es de n bits, la frecuencia f_s con que se produce su rebasamiento es:

$$f_s = TW * \frac{f_r}{2^n}$$

Donde f_r y n son fijas. Cuanto mayor sea TW (incremento de fase), antes desborda el acumulador. A la vez, la señal de salida contendrá menos muestras porque sólo se leerán algunos valores de la tabla en cada vuelta alrededor del círculo de fase. Dado que, según el criterio de Nyquist, para generar una senoide hacen falta por lo menos dos muestras en cada ciclo, es necesario un interpolador que calcule los valores intermedios de la salida entre muestras. Si en cambio TW es pequeña, se leen más valores de la tabla en cada ciclo de la salida. La resolución en la frecuencia de salida es $\frac{f_r}{2^n}$. Si por ejemplo $f_r = 50$ MHz con $n = 14$, la resolución es 3052 Hz.

Como vemos en la siguiente figura. Un registro (número) de n bits, llamado Acumulador de fase, con cada ciclo de reloj se incrementa en un cierto valor que denominaremos incremento de fase TW. El acumulador de fase puede tener un valor (expresado en números decimales) entre 0 y 2^{n-1} . Este incremento periódico del valor numérico contenido en el acumulador de fase hará que luego de determinada cantidad de ciclos de reloj alcance su valor máximo 2^{n-1} . Mientras mayor sea el valor de TW, se requerirán

menos ciclos de reloj para llegar al valor máximo del acumulador de fase, el cual una vez alcanzado hace que el valor del acumulador de fase vuelva a reiniciar a la siguiente suma de TW (siguiente ciclo de reloj). Podemos hacer la analogía entre el acumulador de fase y el ángulo de un fasor que gira a incrementos constantes TW, donde 0° es el valor numérico 0 y cuyos 360° equivalen a un valor 2^n .



Si en el bloque denominado “tabla de salida” (figura 1) a cada valor contenido en el acumulador de fase asociamos un determinado valor de amplitud, por ejemplo, una senoide, tenemos un oscilador digital sinusoidal. En la práctica, a cada valor contenido en el acumulador de fase se le considera una dirección de memoria apuntada hacia los datos contenidos en la tabla de salida, y son estos datos los que contienen la amplitud que corresponde al seno (o coseno) del ángulo equivalente al número contenido en el acumulador de fase. Como los valores típicos de n van desde los 32 hasta 48 bits, si asociamos a cada valor posible en el acumulador de fase una determinada amplitud, ocuparemos excesiva memoria, por lo cual en la práctica se consideran sólo los P bits más significativos (usualmente 14 bits en DDS comerciales).

El valor de TW no necesariamente será un divisor exacto de 2^n , por lo cual al completar un ciclo (giro completo del fasor mostrado en la figura 2) podría no empezar en valor 0 el siguiente ciclo, sino en una fracción de TW. En esencia, cuando con la siguiente adición de TW al acumulador de fase sobrepasa la capacidad numérica de este, simplemente se ignora el bit de acarreo (overflow en inglés) y el ciclo se inicia nuevamente desde un valor numérico correspondiente a la fracción de TW que se puede expresar en N bits, contenida en el acumulador de fase.

A continuación, visualizamos un pequeño ejemplo del funcionamiento del acumulador de fase de fase:

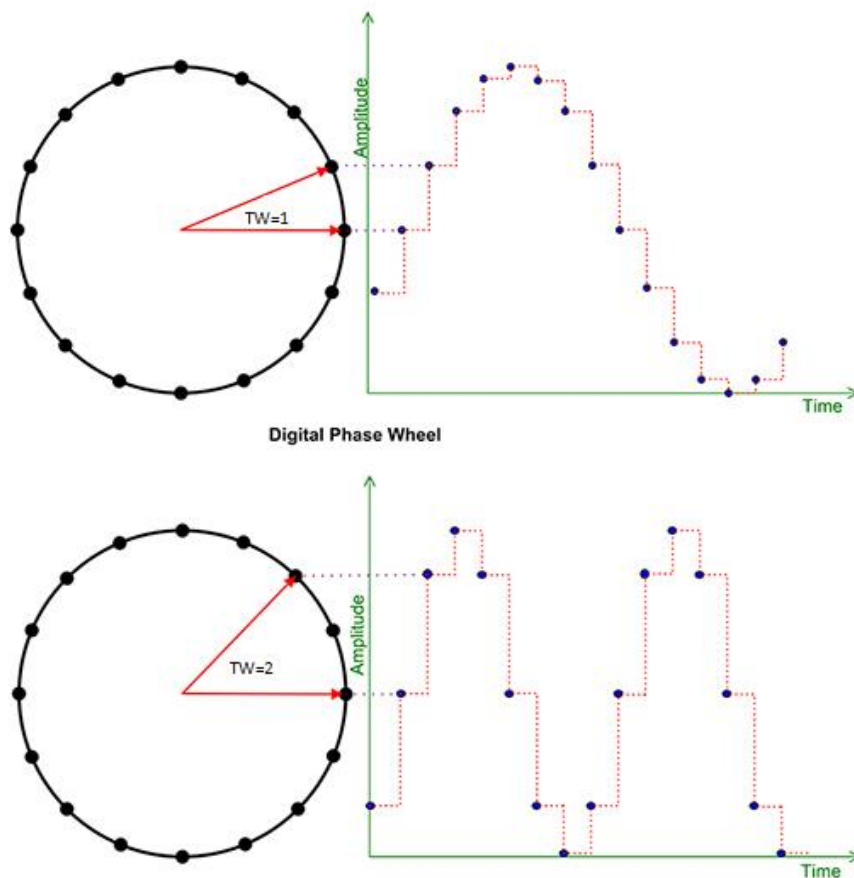


Figura 2: Vista simplificada del funcionamiento de un acumulador de fase de 16 estados que utiliza una rueda de fase para visualizar la forma en que la palabra de sintonía afecta la frecuencia de salida del DDS. (Fuente de la imagen: Digi-Key Electronics).

Como vimos anteriormente, el acumulador de fase es un contador de módulo N , tiene 2^N estados digitales y se incrementan por cada pulso de entrada de reloj del sistema. La magnitud del incremento depende del valor de la palabra de sintonía denominada TW , aplicada a la etapa de sumador del acumulador. La palabra de sintonía, fija la dimensión del incremento del contador. Esto determinará la frecuencia de la forma de onda a la salida.

El acumulador de fase, generalmente cuenta con 24-48 bits; a los 24 bits se observan 224 o 16,777,216 estados. Este número representa la cantidad de valores de fase entre 0 y 2π radianes o el incremento de fase que se puede alcanzar. Para un acumulador de fase de 24 bits, la resolución de fase es de 3.74×10^{-7} radianes. Si se emplea un acumulador de fase más grande, el incremento de fase se vuelve aún más fino.

Los estados del acumulador son periódicos y se representan como si estuvieran en un círculo. Los puntos del círculo equivalen a todos los estados de fase del acumulador. En este caso por simplicidad, el acumulador cuenta con 16 estados. Si la palabra de sintonía es igual a uno como en el diagrama superior, el incremento del paso de cada reloj es uno y se seleccionan todos los estados durante el período completo.

La salida analógica de cada estado se proyecta a la derecha de la rueda de fase. Como este es un dispositivo cuantificado, la salida analógica mantiene el estado actual hasta que el reloj adelanta la rueda de fase al siguiente estado. La forma de onda de salida consiste en un solo ciclo de la onda sinusoidal cuantificada que contiene dieciséis valores.

En el diagrama inferior, el valor de la palabra de sintonía se establece en dos. Con esta configuración, se selecciona un estado cada dos en la rueda de fase. La salida analógica ahora consta de dos ciclos, cada uno con ocho amplitudes, lo que da un total de dieciséis estados. Con la palabra de sintonía fija en dos, la frecuencia de salida es ahora el doble del valor obtenido anteriormente.

La frecuencia de salida del DDS se establece a partir del valor de la palabra de sintonía y aumenta proporcionalmente a dicho valor. La tasa de muestreo permanece fija en la frecuencia de reloj del sistema, y el tiempo entre las muestras de salida es constante. Dado que la frecuencia de salida depende del incremento de la palabra de sintonía, a medida que aumenta su valor y hay menos saltos, la frecuencia es mayor. La palabra de sintonía se puede incrementar hasta que haya solo dos muestras por ciclo, llevando la salida del DDS a la frecuencia Nyquist

o a la mitad de la frecuencia de reloj del sistema. En general, por su diseño, el DDS se limita a tener siempre una frecuencia de salida inferior al límite de Nyquist.

Junto con la frecuencia de reloj del sistema, la frecuencia de salida del DDS también depende del valor de la palabra de sintonía y de la longitud del acumulador. Se expresa mediante la siguiente ecuación:

$$f_{OUT} = \frac{TW \cdot f_{CLK}}{2^N}$$

$$\begin{aligned} f_{OUT} &= \text{frecuencia de salida} \\ N &= \text{largo en bits del acumulador de fase} \\ TW &= \text{número de incremento de fase} \\ f_{CLK} &= \text{frecuencia del reloj} \end{aligned}$$

Tabla de Salida o Conversor de Fase a Amplitud

La salida del acumulador de fase, siendo la fase instantánea de la forma de onda de salida, se utiliza para dirigir el convertidor de fase a amplitud. El convertidor de fase a amplitud, genera una palabra digital cuyo valor es la amplitud de la forma de onda sinusoidal para la fase de entrada.

La cantidad de bits utilizados para impulsar el convertidor de fase a amplitud es menor que la utilizada para el acumulador de fase. Este proceso se conoce como “truncamiento de fase” y sirve para reducir el área del molde y el consumo de energía de las etapas digitales que le siguen al acumulador de fase. Si bien causa algunos componentes espectrales de señal falsa llamados ramales de truncamiento, se minimizan con un diseño cuidadoso.

Conversor digital Analógico

Según sea el valor contenido en el acumulador de fase, en cada ciclo de reloj, se envía al conversor digital – analógico el valor correspondiente de la tabla de salida, obteniendo a la salida de este, una senoide cuya frecuencia es función de TW (valor que controlamos externamente), la frecuencia de reloj (fija, definida por el hardware) y N (capacidad del

acumulador de fase, también definido por el hardware). La ecuación que determina la frecuencia de salida es la vista anteriormente.

El conversor digital analógico es el punto más débil de este sistema dado que es el que nos produce un cuello de botella con respecto a la velocidad de conversión.

Filtro Pasa Bajo

Las formas de onda que se muestran en la Figura 2 son ricas en armónicos debido al carácter escalonado, como resultado, se requiere de un Filtro de paso bajo que suele ubicarse luego del ADC para eliminar tanto estos armónicos espectrales, como así también las respuestas de frecuencias de señales falsas generadas por otros procesos dentro del DDS.

Por ejemplo, el espectro de salida del DDS para un dispositivo sincronizado en f_c con una frecuencia de salida inferior a $f_c/2$ se muestra en la Figura 3. El espectro de salida muestra la línea espectral de salida f_{OUT} , junto con las frecuencias de imagen por encima y por debajo de la frecuencia de reloj y todos los armónicos hasta el tercero.

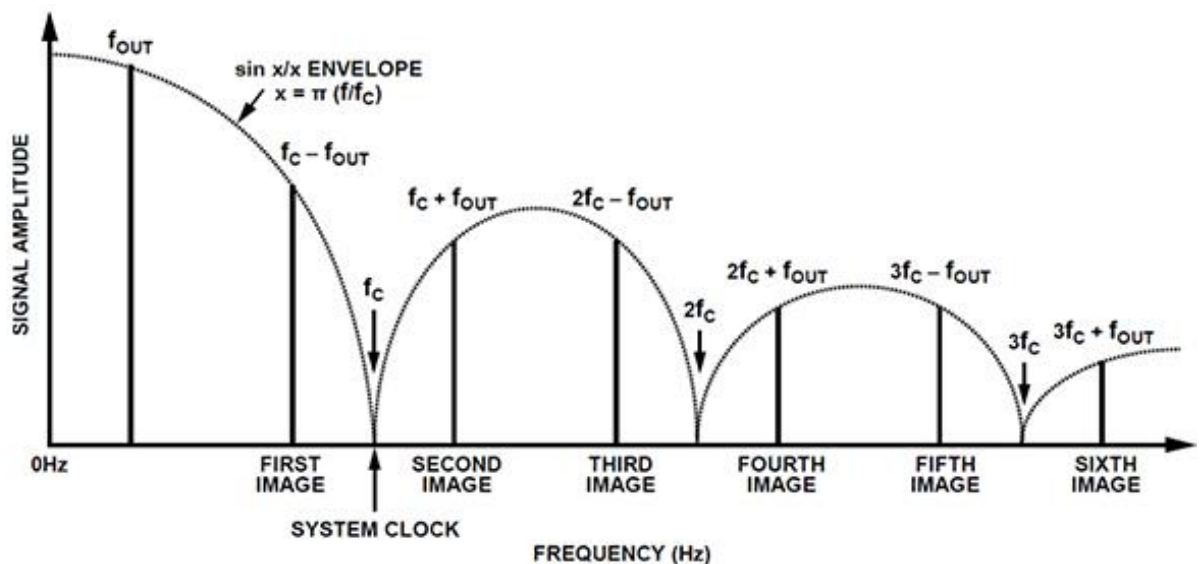
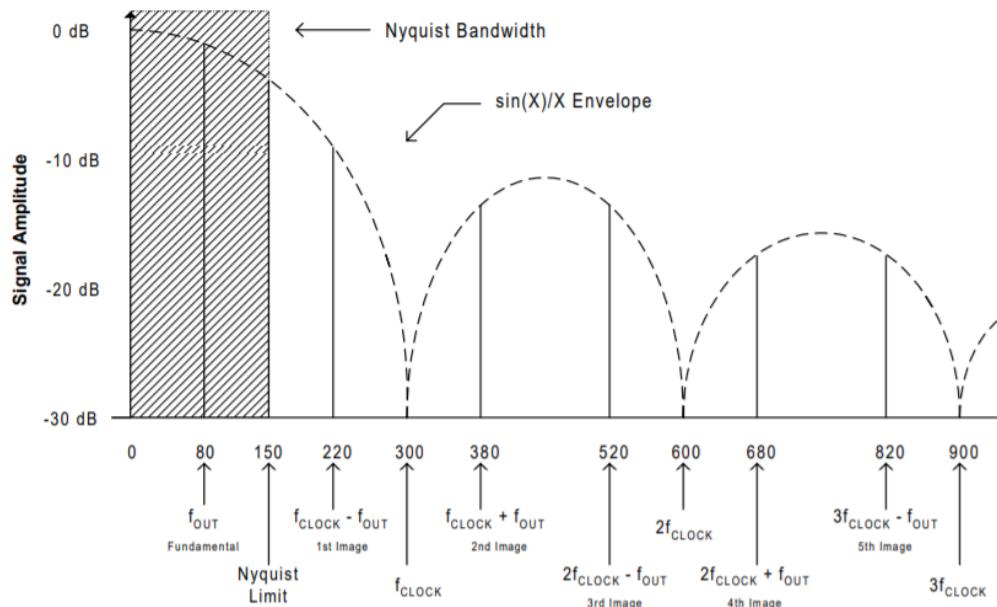


Figura 3: Vista espectral de un DDS con una frecuencia de reloj del sistema de f_c y una frecuencia de salida de f_{OUT} que muestra los componentes de la frecuencia de salida hasta el tercer armónico del reloj. (Fuente de la imagen: Analog Devices)

Es necesario comprender la teoría de muestreo al analizar la salida muestreada de un DDS. En este ejemplo, el reloj de muestreo (f_{CLOCK}) es de 300 MHz y la frecuencia de salida fundamental (f_{OUT}) es 80 MHz.



El Teorema de Nyquist dicta que se requiere un mínimo de dos muestras por ciclo para reconstruir la forma de onda de salida deseada. Las imágenes de las respuestas se crean en el espectro de muestreo de la salida, en $f_{\text{CLOCK}} \pm f_{\text{OUT}}$. La primera respuesta de la imagen ocurre en este ejemplo en $f_{\text{CLOCK}} - f_{\text{OUT}}$ o 220 MHz. Las tercera, cuarta y quinta aparecen a 380 MHz, 520 MHz, 680 MHz y 820 MHz (respectivamente). Se observa que los valores nulos aparecen en múltiplos de la frecuencia de muestreo.

En el caso de que la frecuencia f_{OUT} exceda la frecuencia f_{CLOCK} , la primera respuesta de la imagen aparecerá dentro del ancho de banda de Nyquist (DC - $\frac{1}{2} f_{\text{CLOCK}}$) como una imagen con alias. La imagen alias no se puede filtrar desde la salida con el filtro anti-aliasing tradicional de Nyquist.

En las aplicaciones DDS típicas, se utiliza un filtro de paso bajo para suprimir los efectos de la imagen.

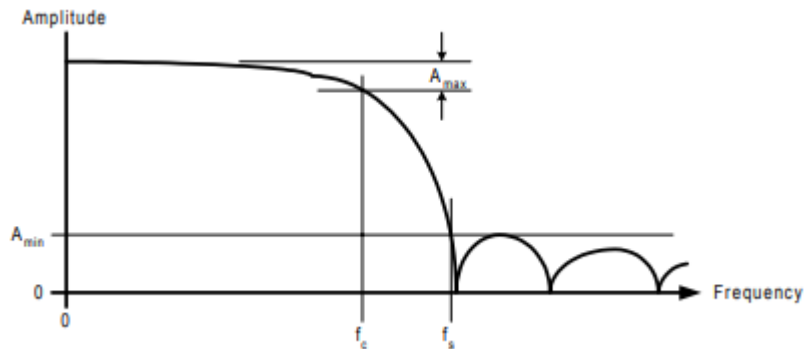
El rango de la frecuencia de salida del DDS es de 0 Hz al límite de Nyquist en $f_c/2$. La formación $\frac{\sin(x)}{x}$ se debe a la señal cuantificada en el dominio del tiempo, como se muestra en la Figura 2. Los ceros de la función $\frac{\sin(x)}{x}$ se producen en la frecuencia de reloj y en todos sus armónicos. Se pueden aplicar correcciones de amplitud para cancelar la formación $\frac{\sin(x)}{x}$ con el fin de mejorar la planeidad de la amplitud en todo el rango de salida.

Se aplica un filtro de paso bajo con un corte pronunciado por encima del rango de frecuencia del DDS para reducir considerablemente la amplitud de las líneas espectrales por encima de Nyquist. Si el rango de frecuencia DDS se extiende a la frecuencia de Nyquist, entonces el filtro requeriría una pendiente de corte pronunciado-infinita para excluir la frecuencia de imagen más baja sobre la frecuencia de reloj, que se superpondría a la frecuencia Nyquist. Esta es una de las razones por las que el rango de frecuencia DDS rara vez se extiende a la frecuencia Nyquist.

Por lo mencionado anteriormente el filtro antialias es un elemento crítico en el diseño de un sistema DDS. Los requisitos que deben imponerse en el diseño del filtro dependen en gran medida de los detalles del sistema DDS.

Antes de analizar los diversos tipos de sistemas DDS, es beneficioso revisar algunos de los tipos de filtros básicos en términos de sus características de dominio de tiempo y dominio de frecuencia.

Los parámetros de filtro típicos de interés son la frecuencia de corte (f_c), la frecuencia de banda de detención (f_s), la atenuación de banda de paso máxima (A_{max}) y la atenuación de banda de detención mínima (A_{min}).



Hay una razón importante para explorar la relación entre los dominios de tiempo y frecuencia con respecto a los filtros. Específicamente, la elección de un tipo de filtro particular depende de si una aplicación requiere un filtro con ciertas características de dominio de tiempo o un filtro con ciertas características de dominio de frecuencia. Uno debe darse cuenta de que existe una compensación entre las características deseables de los dos dominios. A saber, una respuesta de dominio de tiempo suave y una respuesta de dominio de frecuencia aguda. Desafortunadamente, un filtro que exhibe una banda de paso bien definida necesariamente tendrá un timbre y un sobreimpulso en su respuesta al impulso. Del mismo modo, un filtro con una característica de dominio de tiempo uniforme no producirá una transición brusca entre su banda de paso y la banda de parada.

Otro parámetro de filtro importante es el retraso de grupo (que está relacionado con la respuesta en el dominio del tiempo). El retraso de grupo es una medida de la velocidad a la que las señales de diferentes frecuencias se propagan a través del filtro. Generalmente, el retraso del grupo en una frecuencia no es el mismo que en otra frecuencia; es decir, el retraso del grupo generalmente depende de la frecuencia. Esto puede causar un problema cuando un filtro debe transportar un grupo de frecuencias simultáneamente en su banda de paso. Dado que las diferentes frecuencias se propagan a diferentes velocidades, las señales tienden a extenderse entre sí en el tiempo. Lo que se convierte en un problema en las aplicaciones de comunicación

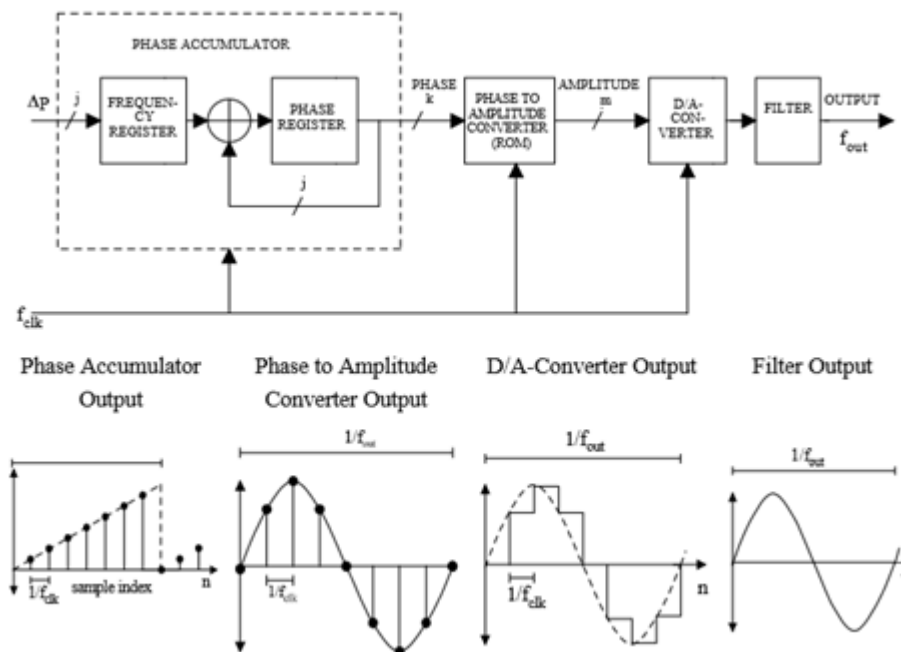
de datos de banda ancha donde es importante que las señales de múltiples frecuencias que se envían a través de un filtro lleguen a la salida del filtro al mismo tiempo.

Existen muchas clases de filtros que existen en la literatura técnica. Sin embargo, para la mayoría de las aplicaciones, el campo se puede reducir a tres familias de filtros básicos. Cada uno está optimizado para una característica particular en el dominio del tiempo o de la frecuencia. Los tres tipos de filtro son las familias de respuestas Chebyshev, Gaussian y Legendre. Las aplicaciones de filtro que requieren características de respuesta de frecuencia bastante nítidas son mejor atendidas por la familia de respuestas Chebyshev.

Sin embargo, se supone que el rizado y el sobreimpulso en el dominio del tiempo no presentan un problema en tales aplicaciones. Por el contrario, las aplicaciones de filtro que requieren características de dominio de tiempo suaves (sobreimpulso y rizado mínimo y retraso de grupo constante) son mejor atendidas por la familia gaussiana de respuestas de filtro. En estas aplicaciones se supone que no se requieren transiciones de respuesta de frecuencia bruscas. Para aquellas aplicaciones que se encuentran entre estos dos extremos, la familia de filtros Legendre es una buena opción.

Desarrollo matemático

El sintetizador digital directo (DDS) se muestra de forma simplificada en la figura.



El DDS tiene los siguientes bloques básicos; un acumulador de fase, un convertidor de fase a amplitud (convencionalmente una ROM sinusoidal), un convertidor digital a analógico y un filtro.

El acumulador de fase consta de un registro de frecuencia de j bits que almacena una palabra de incremento de fase digital seguida de un sumador completo de j bits y un registro de fase. La palabra de incremento de fase ($TW = \Delta P$) de entrada digital se ingresa en el registro de frecuencia. En cada pulso de reloj, estos datos se agregan a los datos previamente contenidos en el registro de fase. La palabra de incremento de fase representa un paso de ángulo de fase que se agrega al valor anterior en cada $1 / f_{clk}$ (frecuencia de clock) segundos para producir un valor digital que aumenta linealmente. El valor de fase se genera utilizando la propiedad de desbordamiento del módulo 2^j de un acumulador de fase de j bits. La tasa de desbordamientos es la frecuencia de salida.

$$f_{out} = \frac{\Delta P f_{clk}}{2^j} \quad \forall \quad f_{out} \leq \frac{f_{clk}}{2},$$

Donde $\Delta P = TW$ es la palabra de incremento de fase, j es el número de bits del acumulador de fase, f_{clk} es la frecuencia del reloj y f_{out} es la frecuencia de salida. La restricción proviene del teorema de muestreo. La palabra de incremento de fase ΔP es un número entero, por lo tanto, la resolución de frecuencia se encuentra configurando $\Delta P = 1$.

$$\Delta f = \frac{f_{clk}}{2^j}.$$

La memoria de solo lectura (ROM) es una tabla de búsqueda sinusoidal, que convierte la información de fase digital en los valores de una onda sinusoidal. En el caso ideal sin cuantificación de fase y amplitud, la secuencia de salida de la tabla viene dada por

$$\sin(2\pi \frac{P(n)}{2^j}),$$

Donde $P(n)$ es un valor de registro de fase (el j -bit) (en el n -ésimo período de reloj). El período numérico de la secuencia de salida del acumulador de fase se define como el valor mínimo de P_e para el cual $P(n) = P(n + P_e)$ para todo n .

El período numérico de la secuencia de salida del acumulador de fase (en ciclos de reloj) es

$$P_e = \frac{2^j}{\text{GCD}(\Delta P, 2^j)},$$

Donde $\text{GCD}(\Delta P, 2^j)$ representa el máximo común divisor de ΔP y 2^j . El período numérico de las muestras de secuencia recuperadas de la ROM sinusoidal tendrá el mismo valor que el período numérico de la secuencia generada por el acumulador de fase.

Por lo tanto, el espectro de la forma de onda de salida del DDS antes de una conversión digital a analógica se caracteriza por un espectro discreto que consiste en puntos P_e . La salida ROM se presenta al conversor D/A, que desarrolla una onda sinusoidal analógica cuantificada.

El espectro de salida del convertidor D / A contiene frecuencias $n f_{clk} \pm f_{out}$, donde $n = 0, 1, \dots$ etc. Las amplitudes de estos componentes son ponderadas por la siguiente función.

$$\sin c\left(\frac{f}{f_{clk}}\right).$$

Este efecto puede corregirse mediante un filtro de sinc inverso (f / f_{clk}). El filtro que está después del convertidor D / A elimina los componentes de muestreo de alta frecuencia y proporciona una salida de onda sinusoidal pura. A medida que el DDS genera frecuencias cercanas a $f_{clk} / 2$, la primera imagen ($f_{clk} - f_{out}$) se vuelve más difícil de filtrar. Esto da como resultado una banda de transición más estrecha para el filtro. La complejidad del filtro está determinada por el ancho de la banda de transición. Por lo tanto, para mantener el filtro simple, la operación DDS está limitada a menos del 40 por ciento de la frecuencia del reloj.

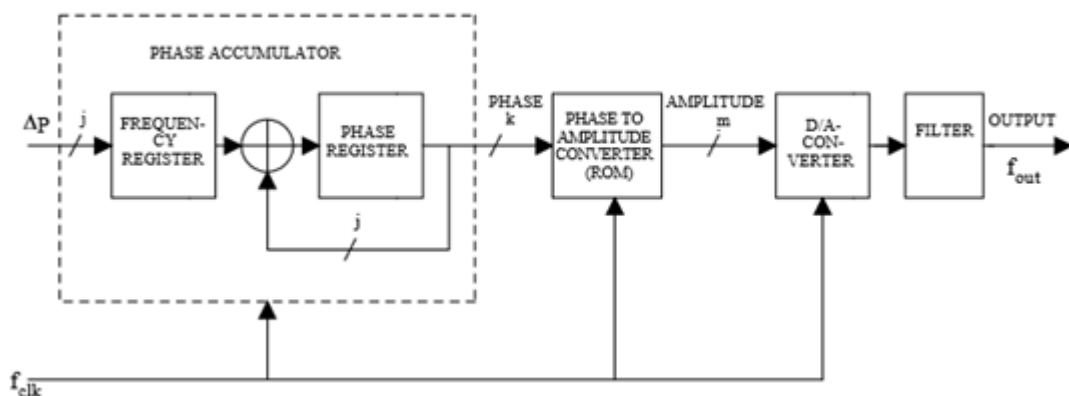


Table 2.1. For an accumulator of 3 bits ($j=3$) controlled with an input of $\Delta P = 3$ and $\Delta P = 2$.

Accumulator output $\Delta P = 3$ and $j = 3$	Carry output	Accumulator output $\Delta P = 2$ and $j = 3$	Carry output
000 (0)	1 Cycle begins	000 (0)	1 Cycle begins
011 (3)	0	010 (2)	0
110 (6)	0	100 (4)	0
001 (1)	1	110 (6)	0
100 (4)	0	000 (0)	1
111 (7)	0	010 (2)	0
010 (2)	1	100 (4)	0
101 (5)	0	110 (6)	0
000 (0)	1	000 (0)	1

La señal de salida de carry del acumulador de fase se utiliza como salida. La frecuencia promedio del DDS se obtiene de:

$$f_{out} = \frac{\Delta P f_{clk}}{2^j}$$

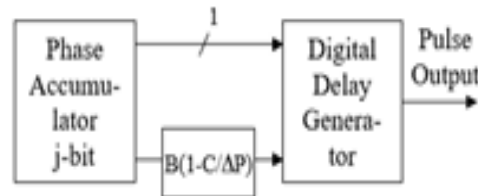
Siempre que ΔP se divide en 2^j , la salida es periódica (columna 3), pero todos los demás casos crean jitter (fluctuación del retardo a la variabilidad temporal durante el envío de señales digitales).

La salida puede cambiar su estado solo a la velocidad del reloj. Si la frecuencia de salida deseada no es un factor (un divisor) de 2^j , se crea un error de fase entre la salida ideal y la real. Este error de fase aumentará (o disminuirá) hasta que alcance un período de reloj completo, momento en el cual vuelve a cero y comienza a acumularse nuevamente (columna 1).

Idealmente, nos gustaría generar una transición cada $8/3 = 2.6667$ ciclos (ver columna 1 en la), pero esto no es posible porque el acumulador de fase puede generar una transición solo en múltiplos enteros del período de reloj. Después de la primera transición, el error es $-1/3$ período de reloj (deberíamos transitar después de 2.6667 relojes, y transitamos después de 3), y después de la segunda es $-2/3$ período de reloj (deberíamos transitar después de 5.33, y lo hacemos después de las 6). Existe una relación clara entre el error y los parámetros ΔP (palabra de incremento de fase) y C (salida del acumulador de fase en el momento de la generación de acarreo). El error es exactamente $-C / \Delta P$.

Al usar un generador de retardo digital, la salida de carry se conecta primero a un circuito lógico que calcula primero la relación $-C / \Delta P$ y retrasa la señal de carry. El retraso negativo debe convertirse en un retraso positivo, que es $1 - C / \Delta P$, $\Delta P > C$ en todas las situaciones (el error de desbordamiento de carry nunca puede ser tan grande ΔP).

Se supone que el tiempo de retardo de toda la línea de retardo cumple exactamente $T_{clk} = 1 / f_{clk}$. Para los componentes de retraso dentro de la línea de retraso hay salidas adicionales B-1 con tiempo de retraso.



$$T_{cv} = \frac{y T_{clk}}{B}, \quad \text{where } y = 1, \dots, B-1,$$

y donde $B = 2^b$ en este caso. El retraso aplicado (yT_{clk} / B) es un múltiplo de los componentes de retraso dentro de la línea de retraso, y el tiempo de retraso positivo es

$$T_{clk} - \frac{C T_{clk}}{\Delta P}.$$

A partir de estas dos ecuaciones, es fácil resolver y (entrada del generador de retardo digital)

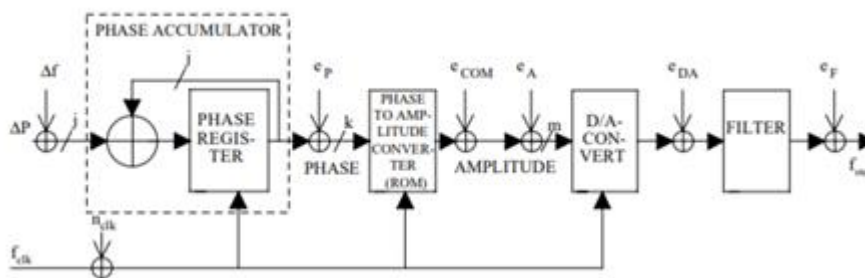
$$y = \left\lfloor B \left(1 - \frac{C}{\Delta P} \right) \right\rfloor,$$

Donde [...] denota truncamiento a valores enteros. La división $C / \Delta P$ requiere mucho hardware.

El generador de retardo también podría implementarse con técnicas analógicas

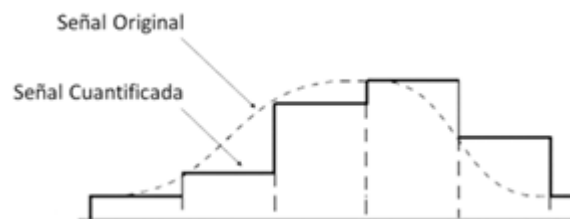
Fuentes de ruido y espuelas en DDS

El modelo del ruido y las espuelas en el DDS tiene seis fuentes. Las fuentes son: el truncamiento del direccionamiento de bits del acumulador de fase la ROM sinusoidal (e_P), distorsión por comprimir la ROM sinusoidal (e_{COM}), la precisión finita de las muestras sinusoidales almacenadas en la ROM (e_A), la conversión digital a analógica (e_{DA}), un filtro posterior (e_F), el ruido de fase de la frecuencia del reloj (n_{clk}) y el error de frecuencia (Δf). La frecuencia error (Δf) provoca un desplazamiento de frecuencia, pero no ruido y espuelas.



Efecto de la resolución del DAC, error de cuantificación (e_{DA}):

La resolución de un DAC se especifica por el número de sus bits de entrada. Por ejemplo, la resolución de un DAC de 10 bits de entrada, se dice que tiene "resolución de 10 bits". El impacto de la resolución del DAC es más fácilmente entendido mediante la visualización de la reconstrucción de una onda sinusoidal como se muestra en la figura.



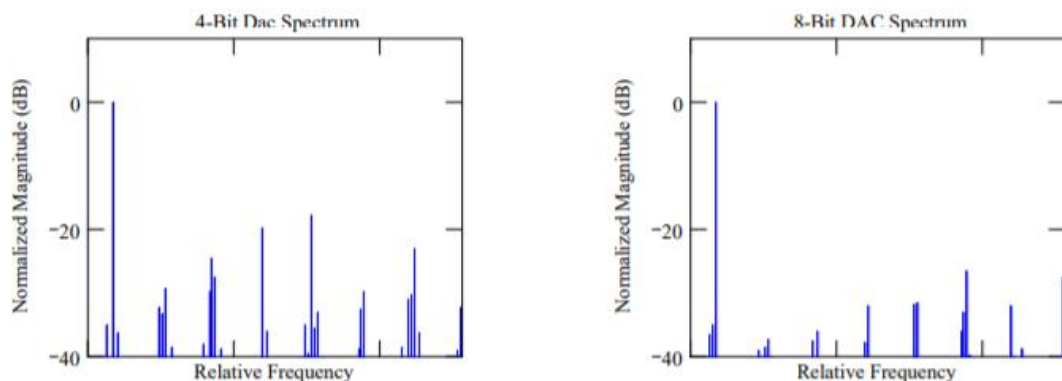
Las líneas verticales son marcadores de tiempo e identifican los instantes en el que la salida del DAC se actualiza a un nuevo valor. Por lo tanto, la distancia horizontal entre las líneas verticales representa el período de la muestra. Se puede observar con absoluta claridad la desviación entre la salida de la señal del DAC y la onda sinusoidal perfecta. La distancia vertical entre las dos trazas en el instante del muestreo es el error introducido por el DAC como

resultado de su resolución finita. Este error es conocido como error de cuantificación y da lugar a un efecto conocido como distorsión de cuantificación.

Para entender la naturaleza de la distorsión de cuantificación, hay que tener en cuenta los cambios abruptos en forma de escalón en la salida del DAC. Estos implican la presencia de componentes de alta frecuencia superpuestos en la frecuencia fundamental. Es en estos componentes de alta frecuencia en donde se presenta la distorsión de cuantificación. En el dominio de la frecuencia, los errores de distorsión de cuantificación son armónicos dentro de la banda de Nyquist y aparecen como señales espurias discretas en el espectro de salida del DAC.

Como la resolución del DAC aumenta, la distorsión de cuantificación disminuye, es decir, el contenido de señales espurias en el espectro de salida del DAC disminuye. Esto tiene sentido, ya que un aumento en la resolución resulta en una disminución en el error de cuantificación. Esto, a su vez, se traduce en menos errores en la reconstrucción de la onda sinusoidal.

Menos error implica menos distorsión; es decir, contenido menos contenido espurio. Esto es gráficamente:



Debemos tener en cuenta que las espuelas asociadas con el DAC de 8 bits son generalmente más bajas que en los del DAC de 4 bits.

De hecho, la relación entre la resolución DAC y la cantidad de distorsión es cuantificable. Si el DAC funciona a su nivel de salida de escala completa:

La potencia de ruido (SQR) viene dada por:

$$\text{SQR} = 1.76 + 6.02B \text{ (dB)}$$

Donde B es el número de bits de resolución DAC.

Por ejemplo, un DAC de 8 bits exhibe un SQR de 49.92dB. Cabe señalar que el SQR

La ecuación solo especifica la potencia de ruido total debido a errores de cuantificación. No proporciona ninguna información sobre la distribución de las espuelas o el nivel máximo de espuelas, solo la combinación poder de todas las espuelas en relación con lo fundamental.

Un segundo punto a considerar es que la ecuación SQR se aplica solo si el DAC opera a escala completa.

A niveles de salida por debajo de la escala completa, la potencia en el fundamental se reduce, pero la cuantización El error permanece constante. El efecto neto es una reducción en SQR; es decir, el ruido de cuantización se vuelve más significativo en relación con lo fundamental. El efecto de operar el DAC en valores menores a la escala máxima es cuantificable y se da como:

$$A = 20 \log (\text{FFS}) \text{ (dB)}$$

*Donde FFS es la fracción de escala completa a la que opera el DAC. Por lo tanto, la ecuación *SQR se convierte en:

$$\text{SQR} = 1.76 + 6.02B + A = 1.76 + 6.02B + 20\log (\text{FFS}) \text{ (dB)}$$

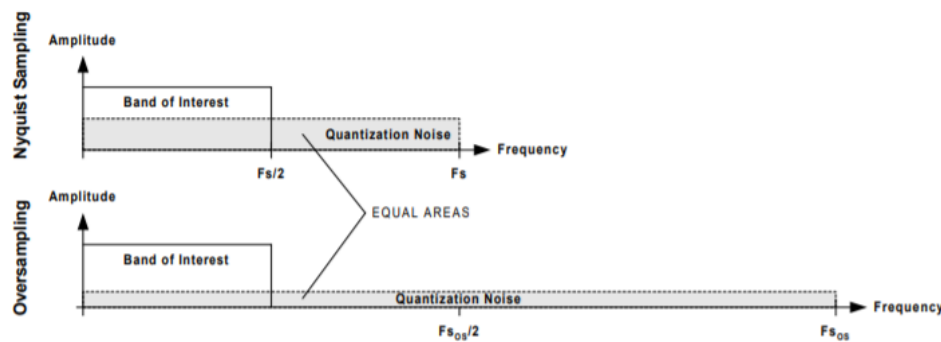
Continuando con el ejemplo anterior, si opera el DAC al 70% de la escala completa ($A = 0.7$), el resultado SQR es 46.82dB (una reducción de 3.1dB del rendimiento original de SQR).

Los efectos del sobre muestreo en el rendimiento espurio

En el sobre muestreo, se da una frecuencia de muestreo superior a la requerida por los criterios de Nyquist.

Recuerde, Nyquist requiere que el ancho de banda de la señal muestreada se limite a la mitad de frecuencia de muestreo. Si el ancho de banda de la señal muestreada está restringido intencionalmente a una fracción de el requisito de Nyquist, entonces la frecuencia de muestreo es superior al requisito de Nyquist y se emplea sobre muestreo.

En la figura se muestra cómo el sobre muestreo mejora la SQR.



La cantidad de potencia de ruido de cuantificación depende de la resolución del DAC. En el caso sobre muestreado, la cantidad total de potencia de ruido de cuantificación es la misma que en el caso de muestra de Nyquist. Dado que la potencia de ruido es la misma en ambos casos (es constante), y el área del rectángulo de ruido es proporcional a la potencia del ruido, luego la altura del ruido del rectángulo en el caso sobre muestreado debe ser menor que el caso muestreado de Nyquist para mantener la misma área. En la banda de interés el área del rectángulo de ruido es menor que para el caso sobre muestreado. Por lo tanto, para una cantidad dada de potencia de señal en la banda de interés, la relación señal / ruido es mayor cuando se emplea sobre muestreo.

El efecto del sobre muestreo es cuantificable y se da como:

$$C = 10\log (F_{sOS} / F_s) \text{ (dB)}$$

Donde F_s es la frecuencia de muestreo de Nyquist y F_{sOS} es la frecuencia de sobre muestreo. El SQR modificado. La ecuación es:

$$\begin{aligned} \text{SQR} &= 1.76 + 6.02B + A + C \\ &= 1.76 + 6.02B + 20\log(F_{FS}) + 10\log(F_{sOS} / F_s) \text{ (dB)} \end{aligned}$$

Volviendo al ejemplo anterior, si operamos el DAC al 70% de la escala completa y sobre muestreo por un factor de 3, el SNR se convierte en 51.59dB. Esto constituye una mejora general de 1.67dB

Por encima del rendimiento original de SQR a gran escala. En este caso, sobre muestreo más que compensado para operar el DAC a solo el 70% de la escala completa.

El efecto del truncamiento en el acumulador de fase (eP):

El truncamiento de fase es un aspecto importante de la arquitectura de los DDS. Considere un DDS de 32 bits, para convertir directamente esos 32 bits de fase a una amplitud correspondiente sería requerir 232 entradas en la LUT. Serían 4 294 967 296 entradas, si cada entrada se almacenará con 8 bits de precisión, entonces se requeriría de 4 gigabytes de memoria en la tabla de búsqueda por lo que es poco práctico implementar un diseño de este tipo. La solución es utilizar una fracción de los bits más significativos de la salida del acumulador para proporcionar la información de fase. Por ejemplo, en un diseño de DDS de 32 bits, sólo los 12 bits superiores se podrían utilizar para la información de fase. Los 20 bits más bajos sería de (1/256) parte de un círculo completo o 1.41 grados (360/256).

En la figura, la resolución del acumulador de fase se identifica por los puntos de la circunferencia externa. Si solo se utilizan los 5 bits más significativos del acumulador para transmitir información de fase, entonces la resolución se convierte en la (1/32) parte del círculo completo o 11,25 grados (360/32) suponiendo que se utiliza un valor de palabra de sintonía de 6, es decir, el acumulador va a contar por incrementos de 6. En la figura se muestran los primeros cuatro ángulos de fase correspondientes a los pasos del conteo (6) del acumulador.



Es necesario tener en cuenta que la primera etapa de la fase (6 conteos en el círculo exterior) se queda corto del primer punto interior marcado. Así, surge una discrepancia entre la fase del acumulador (el círculo exterior) y la fase determinada por la resolución de 5 bits (el círculo interior). Esta incoherencia resulta en un error de fase de $8,46^\circ$ ($6 \times 1,41^\circ$), representada por el arco “E1” en la figura. En la segunda etapa de fase del acumulador (6 conteos más en el círculo exterior) la fase del acumulador debe residir entre el primer y segundo punto de la circunferencia interna. Una vez más, existe una discrepancia entre la fase del acumulador y la fase determinada por los 5 bits de resolución.

El error en este caso es de $5,64^\circ$ ($4 \times 1,41^\circ$) representado por el arco E2 en la figura 2.7. Del mismo modo, en el tercer paso de fase del acumulador el error es de $2,82^\circ$ ($2 \times 1,41^\circ$). Sin embargo, en la 4ta etapa de fase, la fase del acumulador y la fase de resolución de los 5 bits coinciden resultando en que no existe error de fase. Este patrón continúa cada vez que el acumulador realiza los incrementos de 6 conteos en el círculo exterior.

Los errores de fase introducidos por truncar el acumulador darán lugar a errores en la amplitud durante el proceso de conversión de fase a amplitud inherente en el sistema de los DDS. La diferencia entre el valor ideal de amplitud y el truncado presenta un error variable de forma periódica, el cual es independiente de la palabra de sintonización elegida. Puesto que estos errores de amplitud son periódicos en el dominio del tiempo, aparecen como líneas

espectrales en el dominio de la frecuencia, lo que se conoce como frecuencias espurias por truncamiento de fase.

No linealidad del DAC:

Esta es una consecuencia de la incapacidad para diseñar un DAC perfecto. Siempre habrá un error con el nivel de salida esperado asociado al DAC para un código de entrada y el nivel de salida real. Los fabricantes expresan este error como DNL (Differential NonLinearity) e INL (Integral Non-Linearity).

El resultado principal del DNL e INL es que la relación entre la salida esperada del DAC y su salida real no es perfectamente lineal. Esto significa que una señal de entrada se puede transformar a través de algún proceso no lineal antes de aparecer en la salida. Si una onda sinusoidal digital perfecta se introduce en el DAC, el proceso no lineal causa que la salida contenga la onda sinusoidal, más armónicos de la misma. Así, una onda sinusoidal distorsionada se producirá en la salida del DAC. Esta forma de error se conoce como distorsión armónica.

El resultado son señales espurias relacionadas armónicamente en el espectro de salida. En ocasiones se utilizan pues pueden ser útiles si se desea generar frecuencias altas, superiores al reloj de los DDS. Generalmente afectan la calidad de las señales moduladas digitalmente en amplitud.

Señales espurias a la salida del DAC:

Muchos diseños de señales mixtas incluyen uno o más circuitos de reloj de alta frecuencia en el circuito integrado. No es raro que aparezcan estas señales de reloj en la salida del DAC por medio de acoplamiento capacitivo o inductivo. Cualquier acoplamiento de una señal de reloj en la salida del DAC se traducirá en una línea espectral a la frecuencia fundamental del espectro de salida de la señal. Otra posibilidad es que la señal del reloj de referencia se encuentre acoplada al reloj de muestreo del DAC.

Esto provoca que la señal de salida del DAC sea modulada por la señal de reloj, teniendo como consecuencia la existencia de señales espurias simétricas alrededor de la frecuencia de la señal de salida. Las técnicas apropiadas de diseño y fabricación son el único seguro contra esta forma de contaminación.

Otras fuentes de ruido:

La pureza espectral máxima alcanzable de una onda sinusoidal sintetizada es en última instancia relacionada con la pureza del reloj del sistema utilizado para manejar el DDS. Esto es debido al hecho de que en un sistema muestreado se espera que el intervalo de tiempo entre las muestras sea constante. Las limitaciones prácticas, sin embargo, hacen que estos intervalos de muestreo perfectamente uniformes sean imposibles de lograr. Siempre hay cierta variabilidad en el tiempo entre muestras que conducen a desviaciones del intervalo de muestreo deseado. Estas variaciones instantáneas que se traducen en corrimientos en la frecuencia de salida y generación de ruido se denominan jitter. Las causas principales del jitter son dos:

Ruido térmico: Denominado en la física como ruido de Johnson, se produce por el movimiento aleatorio de los electrones en los circuitos eléctricos. Cualquier dispositivo posee una resistencia eléctrica que sirve como una fuente de ruido térmico. Puesto que el ruido térmico es movimiento, su espectro de frecuencia es aleatorio e infinito. De hecho, en cualquier ancho de banda dado, la cantidad de potencia de ruido térmico producido por una resistencia dada es constante. Este hecho conduce a una expresión de la tensión de ruido V_r que está dado por la siguiente ecuación.

$$V_r = \sqrt{4kTB}$$

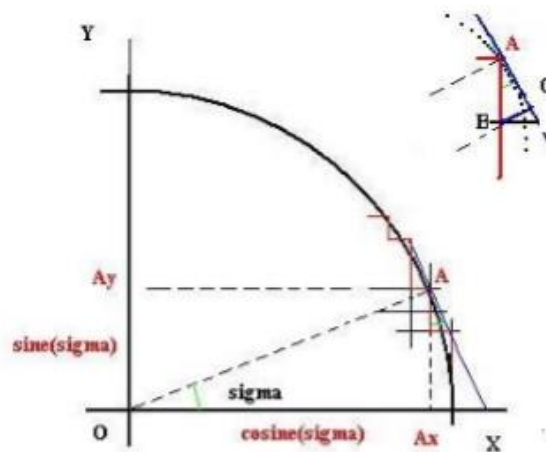
Donde V_r es el valor RMS de la tensión de ruido de Johnson, k es la constante de Boltzmann, T es la temperatura en grados Kelvin, R es la resistencia en ohmios y B es el ancho de banda analizado. La implicación aquí es que independientemente del circuito a utilizar para

generar el reloj del sistema, siempre habrá una cierta cantidad finita de saltos temporales debido al ruido térmico. Por lo tanto, el ruido térmico es el factor limitante cuando se trata de minimizar los saltos temporales. (E. M. Lorca, 2010)

Acoplamiento a fuentes externas: El ruido acoplado puede ser en forma de ruido acoplado localmente causado por la diafonía y/o bucles de tierra dentro o adyacente a la zona inmediata del circuito. También puede ser introducido por fuentes alejadas del circuito. La interferencia que se introduce en el circuito proveniente del medio ambiente circundante se conoce como EMI (interferencia electromagnética). Las fuentes de EMI pueden incluir líneas eléctricas cercanas, radio, transmisores de TV y motores eléctricos, sólo por nombrar unos pocos.

Compensación de la no uniformidad

El uso inmediato de los valores determinados algorítmicamente con sincronización uniforme en la generación de funciones seno y coseno da como resultado distorsiones de fase que deben compensarse para ser aceptables. La solución para la compensación de las distorsiones de fase en la generación de señal es la preservación de la distribución de la uniformidad de las muestras al mismo tiempo que cuando se determinaron los valores. Las coordenadas del punto de aproximación al círculo son valores de funciones de seno y coseno de fase real proporcionales entrelazadas.

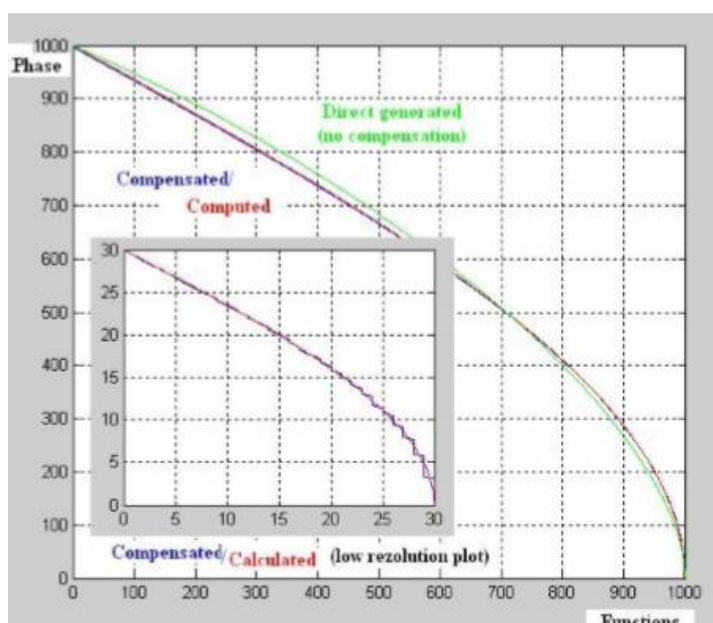


Método de compensación de falta de uniformidad de fase: el retardo de la muestra proporcional a las longitudes de acordes de paso se aproxima por coseno (fase actual).

Una derivación geométrica alternativa del mismo resultado se representa en la figura anterior. El ángulo de fase avanzado en cada paso del algoritmo a lo largo de un eje es proporcional al valor seno o coseno en el avance de fase actual correspondiente como se expresa por las ecuaciones.

$$AC = AB \cos(\sigma)$$

$$\cos(\sigma) = Ax / R \quad (3) \quad AC \sim Ax$$



La simulación MathLab tal como se presenta, demuestra la eficacia del método de compensación de no uniformidad. La simulación muestra que para un generador de señal de amplitud de 10 bits implementado en un FPGA, el método de compensación propuesto es adecuado. Para señales de mayor resolución, se debe desarrollar un método de distribución de temporización de muestra más elaborado.

Especificaciones

El apartado más importante a la hora de elegir o diseñar un DDS es la cantidad de bits que tenga para almacenar datos. La precisión afinable de un CI de síntesis digital directa depende de los registros de frecuencia y de la frecuencia de reloj suministrada. Cuanto más amplios sean los registros, se pueden hacer más partes de la frecuencia de reloj. Por ejemplo, el AD9833 tiene un ancho de bus de 28. Esto significa que puede almacenar 28 bits de datos binarios o dividir una frecuencia de reloj en hasta 268.435.455 maneras en la base 10.

$$11111111111111111111111111111111_2 = 268,435,455_{10}$$

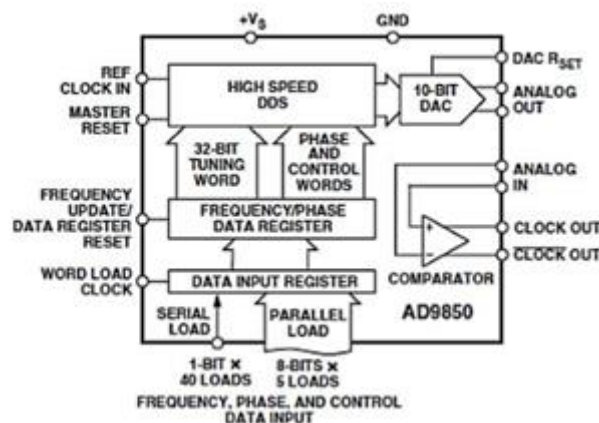
A continuación presentamos un integrado de Síntesis Digital Directa donde se hace un examen de sus especificaciones:

El AD9850 es un dispositivo de alta escala de integración que utiliza las tecnologías más avanzadas en la síntesis de frecuencias acopladas a un comparador y un convertor digital-analógico (DAC) de alto rendimiento y velocidad para formar un completo sintetizador de frecuencias digitalmente programable (véase figura 3.2). Este dispositivo, usando una fuente de reloj precisa, genera a la salida una onda sinusoidal analógica programable en frecuencia y fase, con una excelente pureza espectral.

La señal digital conformada es convertida a analógica mediante un convertor digital-analógico interno de 10 bits de resolución. Un comparador de alta velocidad permite la conversión de la señal sinusoidal a una señal cuadrada con bajo nivel de ruido para aplicaciones

de reloj de alta velocidad. Emplea tecnología avanzada CMOS para proveer gran nivel de funcionalidad y rendimiento con sólo 380mW de disipación de potencia (5V de alimentación).

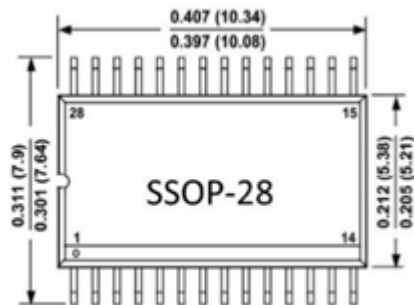
El núcleo DDS rápido e innovador del AD9850 utiliza una palabra de 32 bits para la sintonización de la frecuencia, lo que permite una resolución de sintonía de 0,0291 Hz con una frecuencia de reloj de referencia de 125 MHz. Su arquitectura permite generar frecuencias hasta la mitad de la frecuencia de reloj que se utilice, es decir, hasta 62,5 MHz si el reloj es de 125 MHz y la frecuencia de salida puede ser cambiada digitalmente (asincrónicamente) en un rango de hasta 23 millones de frecuencias nuevas cada segundo. El dispositivo también provee de 5 bits de modulación de fase controlados digitalmente, lo cual permite saltos de fase en la salida que pueden incrementarse en 180°, 90°, 45°, 22.5°, 11.25° o cualquier combinación de estos.



Las palabras de control, de sintonía de frecuencia y modulación de fase pueden ser cargadas al AD9850 de forma paralela o serie. La carga paralela consiste en hacer cinco cargas iterativas de 8 bits (1 byte). El primer byte especifica la modulación de fase, la habilitación del modo power-down y el formato de carga. Los bytes del 2 al 5 comprenden los 32 bits para la sintonización de la frecuencia. La carga serie se realiza mediante un flujo de 40 bits de datos aplicados a un solo terminal.

La frecuencia de reloj de referencia debe ser como mínimo de 1MHz. El dispositivo tiene una circuitería interna que funciona como sensor, de tal manera que, una vez que la frecuencia de referencia excede el umbral, el dispositivo entra en el modo power-down y permanece en él hasta que la frecuencia de entrada supere nuevamente el valor de 1MHz. Este modo previene el paso de una excesiva corriente a través de los registros dinámicos del dispositivo.

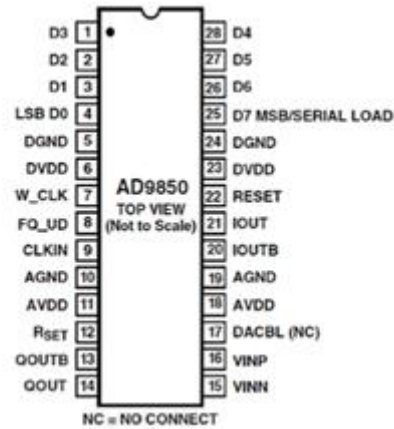
El encapsulado del integrado AD9850 utilizado es del tipo SSOP (Shrink Small Outline Package) de montaje superficial de 28 contactos y tamaño ultra miniatura. La figura 3.3 exponen las dimensiones en milímetros de este tipo de encapsulado.



Una precaución a tener en cuenta es que el AD9850 es sensible a las descargas eléctricas producidas por la estática del cuerpo humano.

Descripción de terminales del AD9850:

En la figura se observa la distribución de terminales del AD9850 y en la tabla 3.1 las descripciones de las funciones de dichos terminales.



Aplicación del AD9850:

En la figura se muestra el diagrama en bloques funcional del AD9850 funcionando como generador de señal de reloj.

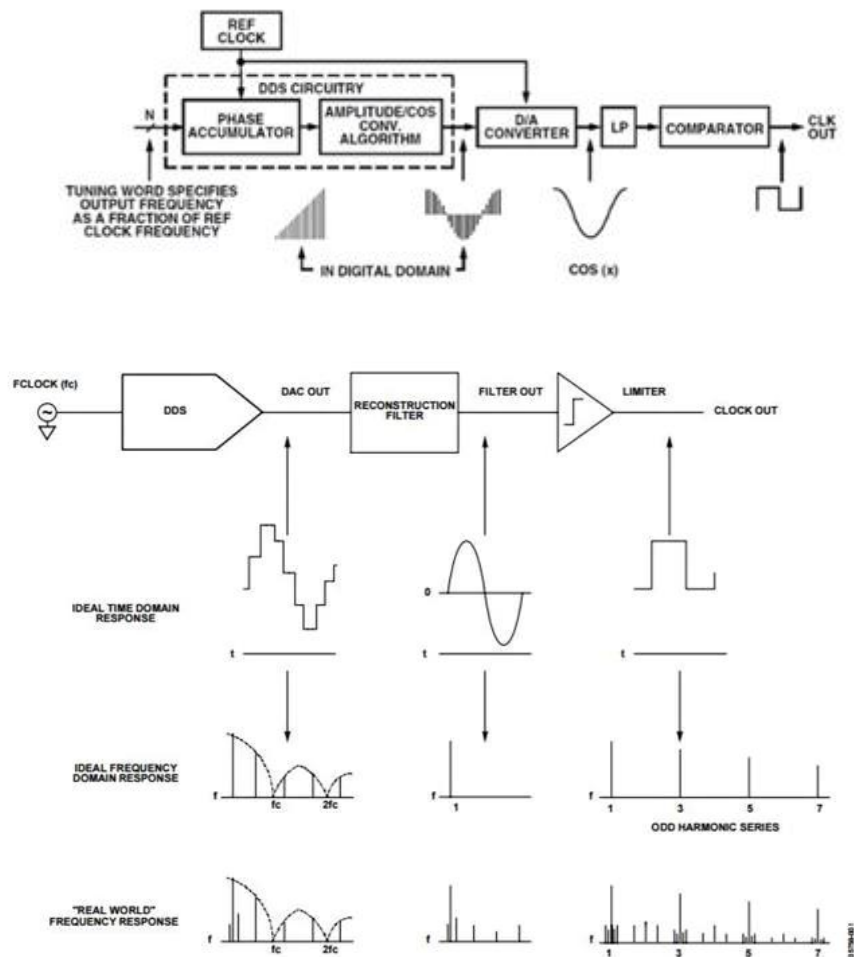


Figure 1. The Process of Generating a Clock from a DDS

El sistema DDS es básicamente un divisor de frecuencia digital donde el incremento de su resolución viene dado por la frecuencia del reloj de referencia dividida por 2^N el número de bits en la palabra de sintonía. El acumulador de fase es un contador que incrementa el valor almacenado por cada pulso de reloj.

La palabra de sintonía de la frecuencia, fija el valor del contador con lo que se determina el tamaño del incremento de fase que es añadido al valor del acumulador de fase en el siguiente pulso de reloj. Entre mayor el incremento añadido sea, más rápido se desbordará el acumulador, lo que resulta en una frecuencia de salida mayor.

La relación entre la frecuencia de salida, el reloj de referencia y la palabra de sintonía del AD9850 viene dada por la ecuación:

$$f_{OUT} = (\Delta F_{ase} \times CLKIN) / 2^{32}$$

Dónde:

ΔF_{ase} es el valor de la palabra de sintonía de 32 bits.

$CLKIN$ es la frecuencia de entrada del reloj de referencia en MHz.

f_{OUT} es la frecuencia de la señal de salida en MHz.

La señal sinusoidal digital de salida del bloque DDS es transformada por el conversor Digital/Analógico de alta velocidad de 10 bits para obtener una señal analógica con bajo ruido.

Otro modelo existente en el mercado

AD9833B RMZ - Sintetizador digital directo

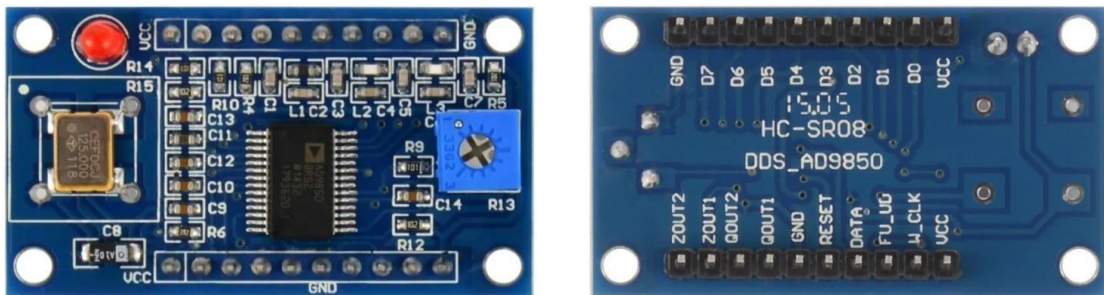
Altura del encapsulado	0.85
Montaje	Montaje superficial
Longitud del encapsulado	3
Ancho del encapsulado	3
Tipo de interfaz digital	Serie (3 hilos, SPI, QSPI, microhilo)
Resolución (bit)	10
Ancho de palabra de ajuste (bit)	28
Cantidad de DAC	1
Cantidad de salidas	1
Frecuencia máxima de entrada (MHz)	25
Relación señal a ruido (dB)	60 (Tipo)
Tipo de salida	voltaje
Voltaje de salida típico (V)	0.65

Corriente de salida típica (mA)	3
Voltaje de suministro operativo mínimo (V)	2.3
Voltaje de suministro operativo típico (V)	2.5 3.3 5
Voltaje de suministro operativo máximo (V)	5.5
Corriente máxima de suministro (mA)	5.5
Soporte de suministro digital	No
Temperatura mínima de funcionamiento (° C)	-40
Temperatura máxima de funcionamiento (° C)	105
Grado de temperatura del proveedor	Industrial
Número de pines	10

DDS Actuales en el Mercado

Módulo con AD9850

Este módulo DDS fabricado por “Analog Devices” se basa en el circuito integrado de AD9850, siendo un DDS de gran tecnología y de alto rendimiento, formando un sintetizador de frecuencias completo y permitiendo de esta manera generar señales senoidales, cuadradas, triangulares de una frecuencia de hasta 40MHz. El módulo es un dispositivo totalmente funcional. Solo se necesita añadir potencia y señales de control para operar este módulo.



Características

- Voltaje operación: +3.3V a +5V DC
- Rango Frecuencia salida: 0-40 MHz
- 4 salidas de señales: 2 Senoidales (Zout1 y Zout2) y 2 Onda Cuadrada (Qout1 y Qout2)
- Ajuste del ciclo de trabajo de la onda cuadrada mediante trimmer
- Palabra de 32 bits para seleccionar la frecuencia
- Interfaz de control simplificada. Carga de datos paralela o serial
- Capacidad de modulación de la fase. Con desplazamientos de fase en incrementos de 180°, 90°, 45°, 22.5°, 11.25° y cualquier combinación de estos.
- Bajo consumo de potencia: 380 mW (reloj de 125 MHz con 5 V). También 155 mW (reloj de 110 MHz con 3.3 V)
- Rango temperatura -40 °C a + 85 °C
- Función de apagado por software

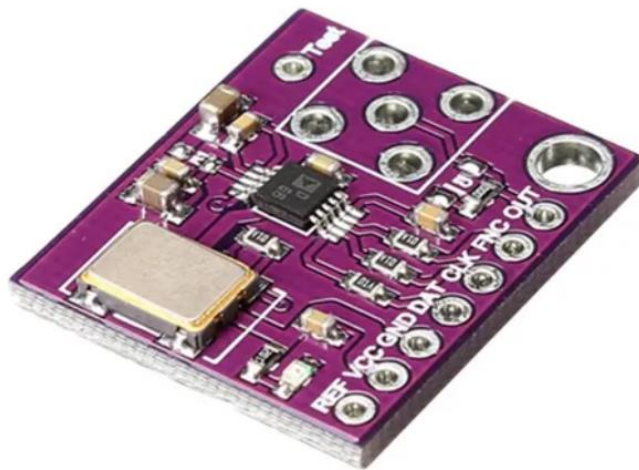
- Dimensiones: 4.51 cm x 2.61 cm

Aplicaciones

- Generador de ondas.
- Circuitos de enganche y recuperación de reloj en comunicaciones digitales.
- Osciladores locales ágiles.
- ADC's controlados digitalmente.

Costo Aproximado:

- \$3000.
-

Módulo con AD9833**Características:*****Especificaciones del Producto:***

- Frecuencia y fase programables digitalmente
- 12,65 mW de consumo de energía a 3 V
- Rango de frecuencia de salida de 0 MHz a 12,5 MHz
- Resolución de 28 bits: 0,1 Hz a reloj de referencia de 25 MHz
- Salidas de onda sinusoidal, triangular y cuadrada
- Fuente de alimentación de 2,3 V a 5,5 V

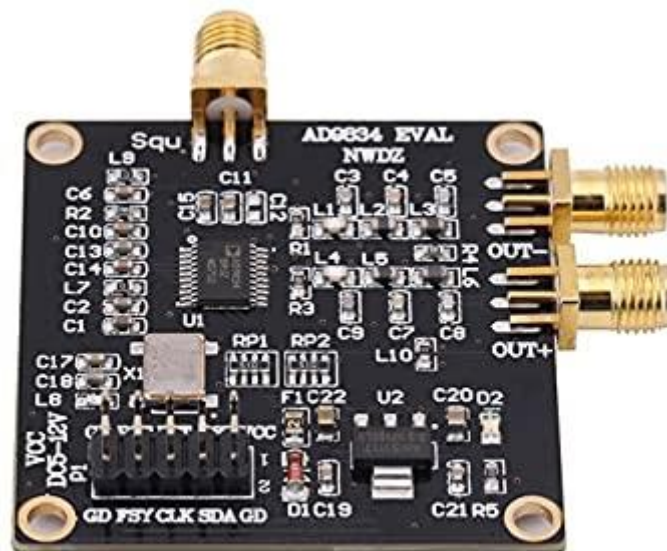
- No se requieren componentes externos
- Interfaz SPI de 3 hilos
- La temperatura de funcionamiento es de -40°C a $+105^{\circ}\text{C}$; las especificaciones típicas son a 25°C

Opción de apagado:

- Tasa de actualización: 25 (máx.)
- VOUT máximo: 0,65 V
- VOUT mínimo: 38mV
- Coeficiente de temperatura VOUT: 200°
- Corriente de entrada: $10\mu\text{A}$
- Alto voltaje de entrada: 1,7-2,8 V
- Entrada de bajo voltaje: 0,5-0,7 V
- Capacitancia de entrada: 3pF
- Tamaño: 17 * 12 mm / 0.66 * 0.47 "

Módulo con AD9834

- Un buen ejemplo de un DDS de baja potencia es Analog Devices' AD9834BRUZ-REEL7.
- Este dispositivo está controlado por una interfaz serial de tres hilos.
- Consume solo 20 milivatios (mW) de un suministro de 3 voltios.
- Puede emitir funciones de onda sinusoidal, de rampa y cuadrada.
- Tiene una frecuencia de reloj máxima de 75 megahertz (MHz), esa frecuencia de reloj significa que puede emitir formas de onda de hasta 37.5 MHz.(segun Nyquist)



- Tres tipos de ondas: este módulo generador de señal DDS es un dispositivo DDS de baja potencia de 75 MHz, capaz de producir salidas senoidales y triangulares de alto rendimiento, y también soporta la generación de onda cuadrada de generación de relojes.
- Función de modulación de fase y frecuencia: este módulo generador de señal AD9834 proporciona una modulación de fase y función de modulación de

frecuencia. cuando el registro de frecuencia es de 28 bits, y la tasa de reloj es de 75 MHz, puede lograr una resolución de 0,28 Hz. Del mismo modo, cuando la tasa de reloj es de 1 MHz, AD9834 puede lograr una resolución de 0.004Hz. Sólo a través de la interfaz de serie para cargar el registro.

- Excelente diseño: esta placa de generador de señal DDS adopta con una interfaz serie de tres cables para escribir datos, y también es compatible con el procesamiento de señal digital y los estándares de microcontroladores. Las secciones analógicas y digitales son independientes, puedes utilizar diferentes fuentes de alimentación, muy fácil y cómodo de usar.
- Función de suspensión: este módulo generador de señal de onda tiene una función de suspensión de pin de control para apoyar el control del modo de apagado desde el exterior, y ninguna parte del dispositivo puede apagarse para minimizar el consumo de energía, económico para usted.

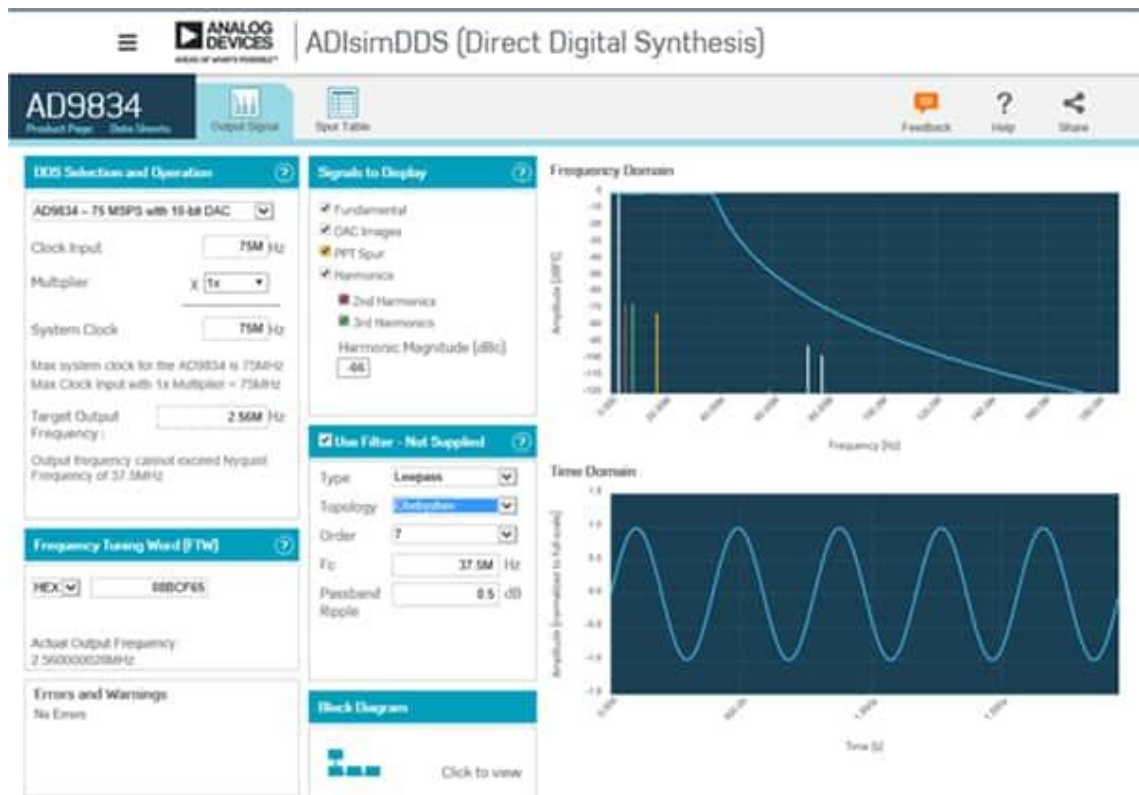
Aplicaciones:

- Se puede utilizar para la excitación de frecuencia/generación de formas de ondas
- Ajuste de fase de frecuencia y modulación
- Sistemas de comunicación RF de baja potencia
- Medición de flujo de aire y líquido
- Aplicaciones de sensores
- Equipos médicos

Herramienta para el AD9834

Con el objetivo de colaborar con el proceso de diseño, es frecuente que los proveedores ofrezcan buenas herramientas de selección para simplificar la tarea. El DDS AD9834 es compatible con Analog Devices' ADIsimDDS, es una herramienta de diseño en línea que

permite a los diseñadores evaluar varias configuraciones, incluidas las frecuencias de salida, las palabras de sintonía y los relojes de referencia .



¿Cómo funciona?

- El programa comienza con la selección de un producto DDS específico, en este caso el AD9834
- El usuario ingresa la frecuencia de reloj del sistema y la frecuencia de salida deseada,
- y el programa calcula la palabra de sintonía para el acumulador de fase.
- Una pantalla de dominio de frecuencia muestra el espectro de la salida DDS, incluida la señal de salida, los armónicos, las imágenes DAC y los armónicos e imágenes del reloj.
- Se puede aplicar un simulador de filtro a la salida DDS para ver los efectos de distintos filtros en el espectro de salida.

Ventajas y desventajas de los DDS

Algunas de las ventajas que poseen los DDS son:

- La frecuencia de la señal sinusoidal se sintoniza digitalmente, típicamente con resoluciones de menos de un 1Hz.
- La fase de la onda sinusoidal es digitalmente ajustable, con sólo un ligero aumento en la complejidad del circuito.
- Debido a que el DDS es digital, la frecuencia y la fase son determinadas numéricamente, no hay errores de la señal por el cambio de temperatura o por el envejecimiento de los componentes.
- El salto de frecuencia a frecuencia, o de fase es extremadamente rápido.
- La interfaz del control digital de la arquitectura de los DDS facilita el medio donde este sistema puede ser controlado remotamente y optimizado con una alta resolución bajo el control del procesador o microcontrolador.
- Control preciso de la frecuencia de salida sin afectar la fase.
- Rápidos cambios arbitrarios tanto como en frecuencia, como en fase.
- Modulación precisa.

Algunas desventajas que poseen los DDS son:

- La frecuencia de salida debe ser menor o igual a la mitad de la fuente del reloj de referencia.
- La amplitud de la señal sinusoidal es fija, esto debido a la naturaleza del DAC interno del DDS. Esta amplitud puede ser modificada con circuitos externos adicionales.
- Debido a que la señal sinusoidal es generada digitalmente por técnicas de muestreo, el usuario debe aceptar una pequeña cantidad de distorsión, esto es que la señal no es 100% pura. Esto se resuelve con un filtro pasa bajos de reconstrucción.

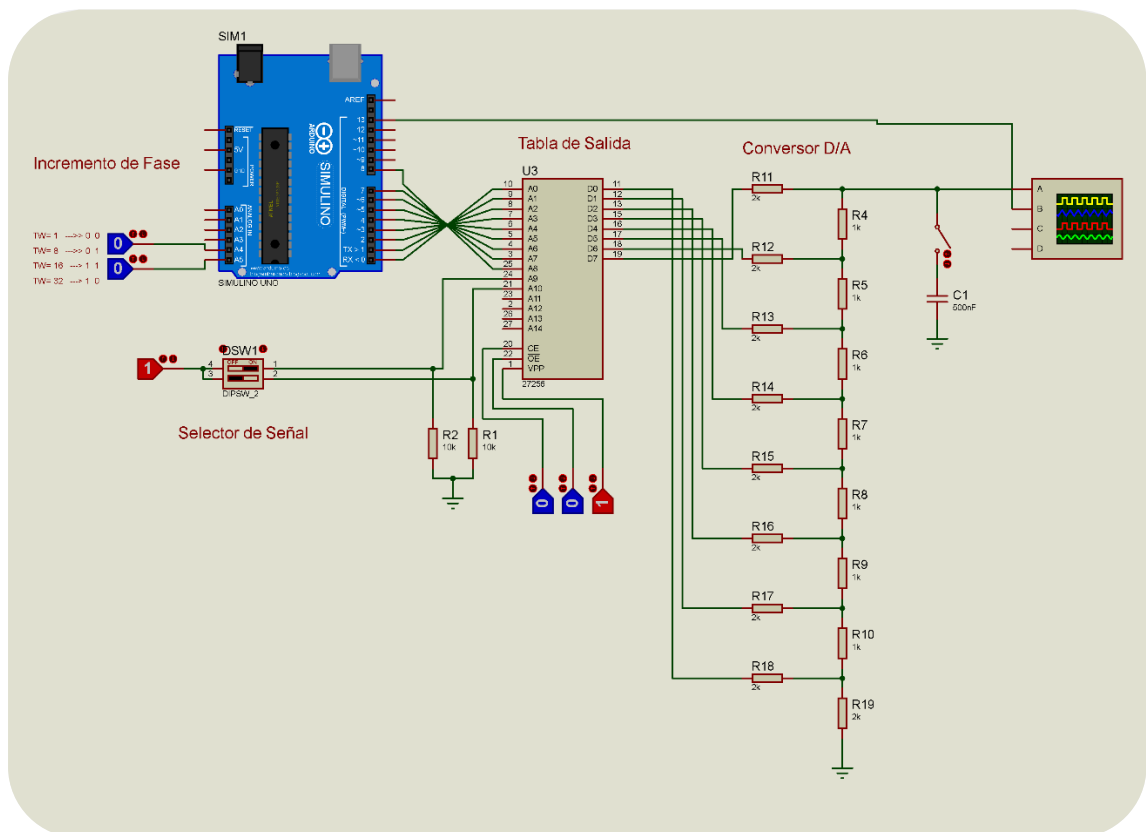
IMPLEMENTACIONES.

En esta sección se adjuntarán tanto las simulaciones implementadas en el Software Proteus con Arduino Uno como un apartado para la implementación realizada con Arduino Due.

En ambos casos la idea fue lograr aplicar los conceptos Teóricos vistos anteriormente e implementarlos en Microcontroladores para lograr obtener ondas periódicas de salidas que varíen su frecuencia al aumentar o disminuir el incrementador de fase.

Generador de Funciones con Arduino Uno en Proteus.

A continuación, se puede observar como a través de los pines A4 y A5 de Arduino se maneja el incremento de fase de nuestro DDS, el micro en este caso brindará en función del incremento seteado y del valor del acumulador una dirección de memoria determinada para ingresar a la memoria donde se cargaron previamente valores de ondas senoidales, cuadradas, dientes de sierra y triangular.



El valor de salida de la tabla para por un conversor digital analógico (en este caso se utilizó un R2R), luego por un filtro pasa bajo y por último leemos la salida atreves de un osciloscopio.

Si bien podían guardarse los valores de las tablas directamente en la memoria EEPROM del Arduino, en este primer caso de Implementación nos pareció mas visible ver el método de Síntesis Digital Directa utilizando una memoria externa en el simulador, pero no quita la posibilidad de utilizar dicha posibilidad más adelante para ahorrar componentes.

Tabla de Memoria:

A continuación, se adjuntan las tablas de valores cargadas en la memoria, para esto se utilizó el Software HxE para generar el archivo “.bin” que ingresamos finalmente en la memoria:

[illegible]

[illegible]

```

A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 9F 9E 9D 9C 9B 9A 99 98 97 96 95 94 93 92 91 90 8F
8E 8D 8C 8B 8A 89 88 87 86 85 84 83 82 81 80 80 7F 7E 7D 7C 7B 7A 79 78 77 76 75 74
73 72 71 70 6F 6E 6D 6C 6B 6A 69 68 67 66 65 64 63 62 61 60 5F 5E 5D 5C 5B 5A 59 58
57 56 55 54 53 52 51 50 4F 4E 4D 4C 4B 4A 49 48 47 46 45 44 43 42 41 40 3F 3E 3D 3C
3B 3A 39 38 37 36 35 34 33 32 31 30 2F 2E 2D 2C 2B 2A 29 28 27 26 25 24 23 22 21 20
1F 1E 1D 1C 1B 1A 19 18 17 16 15 14 13 12 11 10 0F 0E 0D 0C 0B 0A 09 08 07 06 05 04
03 02 01 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19
1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35
36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51
52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D
6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80 80 81 82 83 84 85 86 87 88 89
8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4
A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD
BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6
D7 D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF
F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28
29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44
45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77 78 79 7A 7B 7C
7D 7E 7F 80 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97
98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0 B1
B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA
CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0 E1 E2
E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC
FD FE FF

```

Código del Arduino:

```

#include <avr/io.h>
#include <avr/interrupt.h>
#include <TimerOne.h>
unsigned int salida=0;
int selector1=0, selector0=0;
boolean h=true;
int y=0;
int n=1;
void setup()
{
  Timer1.initialize(400);          // Dispara cada 1250 Hz
  Timer1.attachInterrupt(FaseSDD); // Activa la interrupcion y la asocia a ISR_Filtro
  analogReference(DEFAULT);
  pinMode(A5,INPUT_PULLUP);
  pinMode(A4,INPUT_PULLUP);
  for(int i=0;i<=13;i++){
    pinMode(i, OUTPUT);
  }
}
void FaseSDD(){
  h=!h;
  digitalWrite(13,h);
  selector0=digitalRead(A4);
  selector1=digitalRead(A5);
}

```

```

if(selector0==0 && selector1==0)
n=1;
if(selector0==0 && selector1==1)
n=8;
if(selector0==1 && selector1==1)
n=16;
if(selector0==1 && selector1==0)
n=32;

y=y+n;

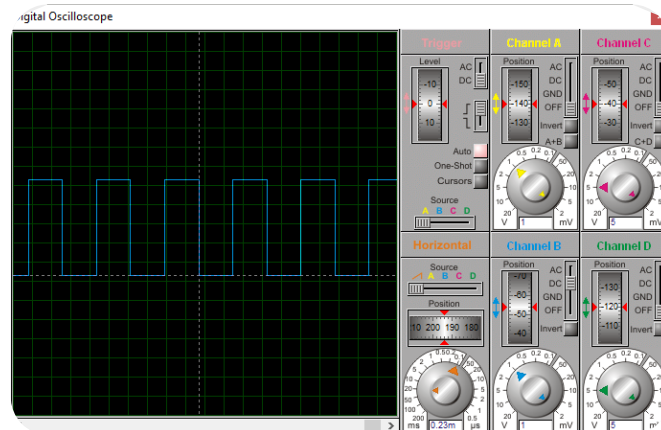
if(y<512)
{
//Convertimos y escribimos en la salida.
salida=(unsigned int)y;
digitalWrite(0,salida & 1);
digitalWrite(1,salida >> 1 & 1);
digitalWrite(2,salida >> 2 & 1);
digitalWrite(3,salida >> 3 & 1);
digitalWrite(4,salida >> 4 & 1);
digitalWrite(5,salida >> 5 & 1);
digitalWrite(6,salida >> 6 & 1);
digitalWrite(7,salida >> 7 & 1);
digitalWrite(8,salida >> 8 & 1);
}
else
y=0;
}
void loop()
{
}

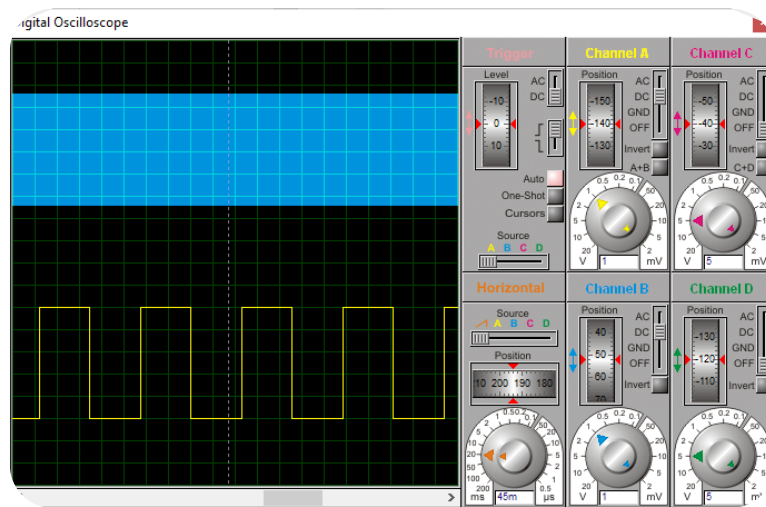
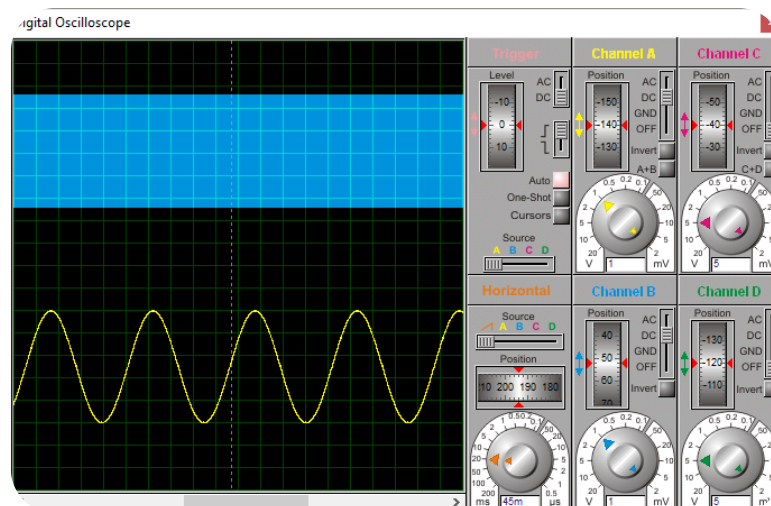
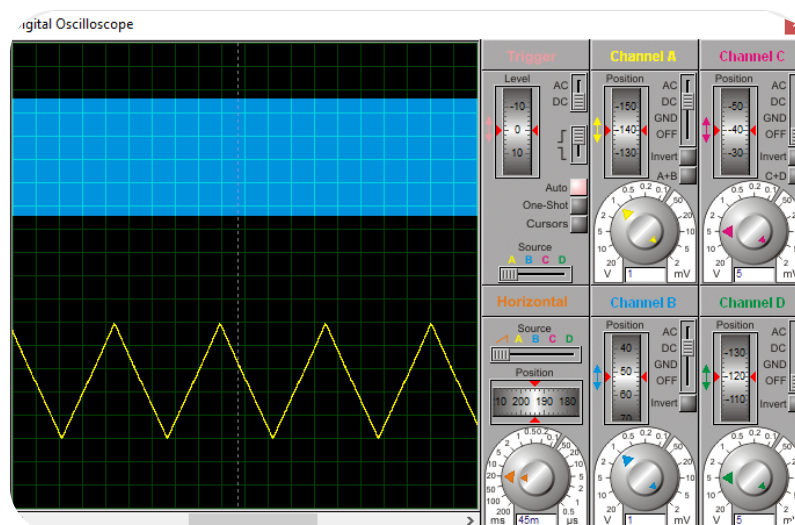
```

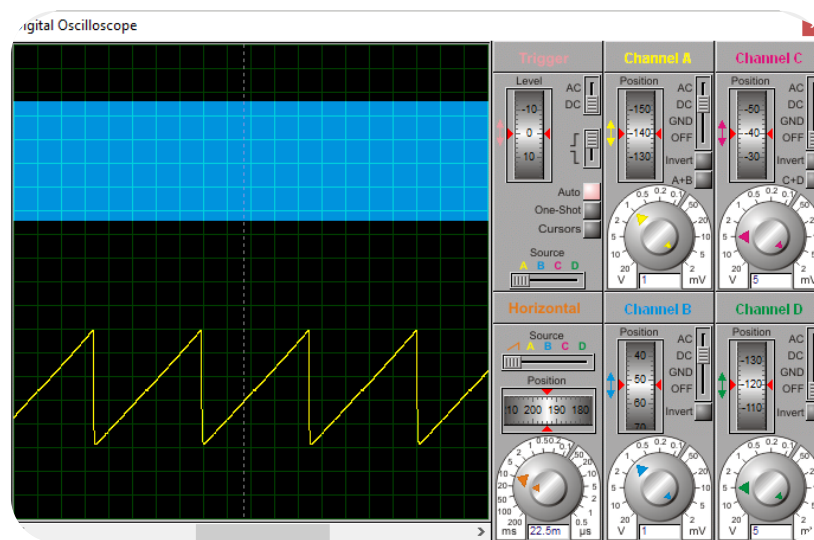
Salidas Obtenidas en la Simulación:

En este caso veremos en azul la frecuencia de clock y en amarillo las frecuencias de salida para un paso TW=1.

Frecuencia de Clock:



Onda Cuadrada:**Onda Senoidal:****Onda Triangular:**

Onda Diente de Sierra:

DDS con Arduino DUE.

Objetivos planteados por el grupo:

Visto que muchas veces para probar diferentes circuitos como filtros, amplificadores, etc en nuestra carrera a falta de elementos propios hemos tenido que recurrir ineludiblemente al laboratorio, nuestro objetivo es lograr realizar un generador de funciones que integre los conceptos de la Síntesis Digital de Frecuencia permitiendo variar fácilmente su forma de onda y su frecuencia y que además sea fácilmente utilizable abarcando la menor cantidad de dispositivos para su implementación, pero dejando abierta la puerta a posibles agregados externos como displays, pulsadores, etc.

Elementos Necesarios:

- Placa de Desarrollo Arduino Due.
- PC con Arduino IDE.
- Cable USB a MICRO USB.
- 1 resistencia de 10[k Ω] a la salida del DAC.

Características Arduino Due:

- Microcontrolador: AT91SAM3X8E.
- Voltaje de operación: 3.3V.
- Voltaje recomendado de entrada (pin Vin): 7-12V.
- Pines de entrada y salida digitales: 54 pines I/O, de los cuales 12 proveen salida PWM.
- Pines de entrada análogos: 12.
- Corriente de salida total en los pines I/O: 130mA.
- Corriente DC máxima en el pin de 3.3V: 800mA.
- Corriente DC máxima en el pin de 5V: 800mA.
- *Memoria Flash: 512 KB toda disponible para aplicaciones del usuario.*
- *SRAM: 96 KB (en dos bancos de: 64KB y 32KB).*
- *Velocidad de reloj: 84 MHz.*
- *Pines de salida análogos: 2.*

Si bien es importante conocer todas las características del microcontrolador a utilizar, las ultimas 4 son para tener muy en cuenta a la hora de lo que se plantea implementar resaltando en especial el ultimo ya que dispondremos de 2 DAC en la misma placa por los cuales podremos sacar nuestras señales sin necesidad de un DAC externo. Cabe resaltar que el

Arduino Mega también posee estas salidas y posee también memoria EPROM en la cual se podrían guardar las tablas de las diferentes funciones.

Temporizadores Internos y DAC en Arduino DUE:

Agregamos este apartado para tener una idea general de los temporizadores disponibles en Arduino Due ya que los utilizaremos para obtener las frecuencias deseadas.

La CPU SAM3X8E tiene 3 contadores de temporizador (TC) llamados TC0, TC1, TC2. Cada TC incluye tres canales idénticos de 32 bits. Cada canal se puede programar de forma independiente para realizar una amplia gama de funciones que incluyen medición de frecuencia, recuento de eventos, medición de intervalo, generación de pulsos, temporización de retardo y modulación de ancho de pulso (PWM). Cada canal tiene tres entradas de reloj *externas*, cinco entradas de reloj *interno* y dos señales de entrada / salida multipropósito que pueden ser configuradas por el usuario. Cada canal impulsa una señal de interrupción interna que se puede programar para generar interrupciones del procesador. El TC integra la lógica del decodificador en cuadratura conectado frente a los 3 temporizadores e impulsado por entradas TIOA0, TIOB0 y TIOA1. Cuando está habilitado, el decodificador en cuadratura realiza el filtrado de línea de entrada y la decodificación de señales en cuadratura. El bloque TC tiene dos registros globales que actúan sobre los tres canales TC. El Registro de control de bloques permite que los tres canales se inicien simultáneamente con la misma instrucción. El Registro de modo de bloque define las entradas de reloj externo para cada canal, lo que les permite encadenarse. Los relojes se asignan a los contadores de temporizador de la siguiente manera:

- **TIMER_CLOCK1** - MCK / 2
- **TIMER_CLOCK2** - MCK / 8
- **TIMER_CLOCK3** - MCK / 32
- **TIMER_CLOCK4** - MCK / 128
- **TIMER_CLOCK5** - SLCK

MCK es el reloj maestro (84 MHz) y SLCK es el reloj lento (32 kHz). Cabe señalar que es posible seleccionar el reloj lento como reloj maestro, en cuyo caso la entrada TIMER_CLOCK5 es equivalente al reloj maestro. Los TC se pueden encadenar utilizando TIOA0, TIOA1, TIOA2 como una entrada de reloj externo para los TC posteriores, lo que permite una mayor división de la frecuencia de reloj.

En nuestro caso un contador se incrementa con cada tic del temporizador 2 (TIMER_CLOCK2 - MCK/8). Cuando el contador alcanza un umbral, se llama a la interrupción del programa y el ángulo de fase se incrementa en un cierto paso de fase. El ángulo de fase se asigna a un valor digital a través de una tabla de valor de fase y dicho valor se escribe a la salida del DAC.

Teniendo en cuenta esto último la frecuencia de salida mínima de la señal periódica se calcula como:

$$Frec. minima = \frac{\frac{frecuencia\ del\ Timer\ seleccionado}{contador + 1}}{Cantidad\ de\ muestras\ en\ la\ Tabla} = \frac{\frac{f_clock}{c + 1}}{n}$$

La “ f_clock ” se establece en una frecuencia alta ($f_clock = 84\text{ MHz} / 8$) para permitir la resolución de tiempo más alta posible de la señal de salida. Los valores más altos de n aumentan la resolución temporal. Luego “ c ” y “ n ” juntos definen la frecuencia de la señal.

Para determinar la frecuencia de salida máxima a una determinada resolución (número de muestras n), no solo debemos considerar los valores anteriores, sino también la latencia de procesamiento para alimentar el DAC con valores digitales a convertir y la latencia del DAC para convertir un digital a un valor analógico. La latencia DAC del microcontrolador SAM3X8E es de 25 períodos de reloj DAC, donde el reloj DAC corre a la mitad de la velocidad del reloj maestro, es decir, 42 MHz. El DAC se alimenta simultáneamente a través de un búfer FIFO mientras que el DAC convierte el siguiente valor. Siempre que se pueda proporcionar un valor al DAC a una frecuencia de al menos $42\text{ MHz} / 25 = 1,68\text{ MHz}$ --- lo cual es factible para

Due corriendo a 84 MHz, la latencia de procesamiento no se convertirá en el cuello de botella. Por ejemplo, podemos generar una señal de 1 kHz con 1024 muestras ya que:

$$1000 \text{ Hz} * 1024 = 1,024 \text{ MHz} < 1,68 \text{ MHz}$$

Sin embargo, para generar una señal de 2 kHz, tenemos que disminuir la resolución a 512 muestras.

Primeras Pruebas:

Comenzamos la implementación probando el siguiente código obtenido del [Foro de Arduino](#) en donde se logran salidas de 4 ondas distintas en frecuencias variables entre 1[Hz] y 170[Hz] a pasos de 1[Hz]. En los códigos adjuntados a continuación vemos que en una primera instancia se define una función que contiene las tablas de cada onda con n=120 y luego en el programa principal a través de la lectura de un potenciómetro en su pin ADC y 2 pulsadores se seleccionan la onda y la frecuencia deseada.

```
#ifndef _Waveforms_h_
#define _Waveforms_h_

#define maxWaveform 4
#define maxSamplesNum 120

static int waveformsTable[maxWaveform][maxSamplesNum] = {
    // Sin wave
    {
        0x7ff, 0x86a, 0x8d5, 0x93f, 0x9a9, 0xa11, 0xa78, 0xadd, 0xb40, 0xba1,
        0xbff, 0xc5a, 0xcb2, 0xd08, 0xd59, 0xda7, 0xdf1, 0xe36, 0xe77, 0xeb4,
        0xeec, 0xf1f, 0xf4d, 0xf77, 0xf9a, 0xfb9, 0xfd2, 0xfe5, 0xff3, 0xffc,
        0xfff, 0xffc, 0xff3, 0xfe5, 0xfd2, 0xfb9, 0xf9a, 0xf77, 0xf4d, 0xf1f,
        0xeec, 0xeb4, 0xe77, 0xe36, 0xdf1, 0xda7, 0xd59, 0xd08, 0xcb2, 0xc5a,
        0xbff, 0xba1, 0xb40, 0xadd, 0xa78, 0xa11, 0x9a9, 0x93f, 0x8d5, 0x86a,
        0x7ff, 0x794, 0x729, 0x6bf, 0x655, 0x5ed, 0x586, 0x521, 0x4be, 0x45d,
        0x3ff, 0x3a4, 0x34c, 0x2f6, 0x2a5, 0x257, 0x20d, 0x1c8, 0x187, 0x14a,
        0x112, 0xdf, 0xb1, 0x87, 0x64, 0x45, 0x2c, 0x19, 0xb, 0x2,
        0x0, 0x2, 0xb, 0x19, 0x2c, 0x45, 0x64, 0x87, 0xb1, 0xdf,
        0x112, 0x14a, 0x187, 0x1c8, 0x20d, 0x257, 0x2a5, 0x2f6, 0x34c, 0x3a4,
        0x3ff, 0x45d, 0x4be, 0x521, 0x586, 0x5ed, 0x655, 0x6bf, 0x729, 0x794
    }
    ,

    // Triangular wave
    {
        0x44, 0x88, 0xcc, 0x110, 0x154, 0x198, 0x1dc, 0x220, 0x264, 0x2a8,
        0x2ec, 0x330, 0x374, 0x3b8, 0x3fc, 0x440, 0x484, 0x4c8, 0x50c, 0x550,
        0x594, 0x5d8, 0x61c, 0x660, 0x6a4, 0x6e8, 0x72c, 0x770, 0x7b4, 0x7f8,
        0x83c, 0x880, 0x8c4, 0x908, 0x94c, 0x990, 0x9d4, 0xa18, 0xa5c, 0xaa0,
        0xae4, 0xb28, 0xb6c, 0xbb0, 0xbf4, 0xc38, 0xc7c, 0xcc0, 0xd04, 0xd48,
        0xd8c, 0xdd0, 0xe14, 0xe58, 0xe9c, 0xee0, 0xf24, 0xf68, 0xfac, 0xff0,
        0xfac, 0xf68, 0xf24, 0xee0, 0xe9c, 0xe58, 0xe14, 0xdd0, 0xd8c, 0xd48,
        0xd04, 0xcc0, 0xc7c, 0xc38, 0xbf4, 0xbb0, 0xb6c, 0xb28, 0xae4, 0xaa0,
        0xa5c, 0xa18, 0x9d4, 0x990, 0x94c, 0x908, 0x8c4, 0x880, 0x83c, 0x7f8,
        0x7b4, 0x770, 0x72c, 0x6e8, 0x6a4, 0x660, 0x61c, 0x5d8, 0x594, 0x550,
        0x50c, 0x4c8, 0x484, 0x440, 0x3fc, 0x3b8, 0x374, 0x330, 0x2ec, 0x2a8,
        0x264, 0x220, 0x1dc, 0x198, 0x154, 0x110, 0xcc, 0x88, 0x44, 0x0
    }
}
```

```
,
// Sawtooth wave
{
  0x22, 0x44, 0x66, 0x88, 0xaa, 0xcc, 0xee, 0x110, 0x132, 0x154,
  0x176, 0x198, 0x1ba, 0x1dc, 0x1fe, 0x220, 0x242, 0x264, 0x286, 0x2a8,
  0x2ca, 0x2ec, 0x30e, 0x330, 0x352, 0x374, 0x396, 0x3b8, 0x3da, 0x3fc,
  0x41e, 0x440, 0x462, 0x484, 0x4a6, 0x4c8, 0x4ea, 0x50c, 0x52e, 0x550,
  0x572, 0x594, 0x5b6, 0x5d8, 0x5fa, 0x61c, 0x63e, 0x660, 0x682, 0x6a4,
  0x6c6, 0x6e8, 0x70a, 0x72c, 0x74e, 0x770, 0x792, 0x7b4, 0x7d6, 0x7f8,
  0x81a, 0x83c, 0x85e, 0x880, 0x8a2, 0x8c4, 0x8e6, 0x908, 0x92a, 0x94c,
  0x96e, 0x990, 0x9b2, 0x9d4, 0x9f6, 0xa18, 0xa3a, 0xa5c, 0xa7e, 0xaa0,
  0xac2, 0xae4, 0xb06, 0xb28, 0xb4a, 0xb6c, 0xb8e, 0xbb0, 0xbd2, 0xbf4,
  0xc16, 0xc38, 0xc5a, 0xc7c, 0xc9e, 0xcc0, 0xce2, 0xd04, 0xd26, 0xd48,
  0xd6a, 0xd8c, 0xdae, 0xdd0, 0xdf2, 0xe14, 0xe36, 0xe58, 0xe7a, 0xe9c,
  0xeb6, 0xed0, 0xef2, 0xf04, 0xf26, 0xf48, 0xf6a, 0xf8c, 0xfae, 0xfc0, 0xfe0
}
,
// Square wave
{
  0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
  0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
  0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
  0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
  0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff, 0xfff,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
}

};
```

#endif

*****Código Principal*****

```
/*
Simple Waveform generator with Arduino Due

* connect two push buttons to the digital pins 2 and 3
  with a 10 kilohm pulldown resistor to choose the waveform
  to send to the DAC0 and DAC1 channels
* connect a 10 kilohm potentiometer to A0 to control the
  signal frequency

*/

#include "Waveforms.h"

#define oneHzSample 1000000/maxSamplesNum // sample for the 1Hz signal expressed in microseconds

const int button0 = 2, button1 = 3;
volatile int wave0 = 0, wave1 = 0;

int i = 0;
int sample;

void setup() {
  analogWriteResolution(12); // set the analog output resolution to 12 bit (4096 levels)
  analogReadResolution(12); // set the analog input resolution to 12 bit

  attachInterrupt(button0, wave0Select, RISING); // Interrupt attached to the button connected to pin 2
  attachInterrupt(button1, wave1Select, RISING); // Interrupt attached to the button connected to pin 3
}

void loop() {
  // Read the the potentiometer and map the value between the maximum and the minimum sample available
  // 1 Hz is the minimum freq for the complete wave
  // 170 Hz is the maximum freq for the complete wave. Measured considering the loop and the analogRead() time
  sample = map(analogRead(A0), 0, 4095, 0, oneHzSample);
```

```

sample = constrain(sample, 0, oneHzSample);

analogWrite(DAC0, waveformsTable[wave0][i]); // write the selected waveform on DAC0
analogWrite(DAC1, waveformsTable[wave1][i]); // write the selected waveform on DAC1

i++;
if(i == maxSamplesNum) // Reset the counter to repeat the wave
  i = 0;

delayMicroseconds(sample); // Hold the sample value for the sample time
}

// function hooked to the interrupt on digital pin 2
void wave0Select() {
  wave0++;
  if(wave0 == 4)
    wave0 = 0;
}

// function hooked to the interrupt on digital pin 3
void wave1Select() {
  wave1++;
  if(wave1 == 4)
    wave1 = 0;
}

```

Conclusiones Respecto al código:

Nos fue muy útil como para comenzar a ver formas posibles de implementación pero visto las frecuencias que puede llegar a manejar el micro y que nuestra idea era utilizar el método de SDD (en este caso utiliza tablas con muestras pero la frecuencia de salida la varía con el comando `delayMicroseconds()` y no aumentando o disminuyendo un contador de fase) decidimos seguir investigando formas de obtener mayores frecuencias de salida y de eliminar el potenciómetro y los pulsadores externos para adaptarnos a nuestros objetivos.

Código 2:

En este caso se busco la manera de eliminar los elementos externos y para esto se procedió a utilizar el monitor serie del IDE de Arduino. En el código el micro lee en buffer serie y la función *DatosPorSerie()* aumenta o disminuye la frecuencia de salida en función a lo que se le solicite desde la pc.

```

/* PROGRAMA QUE GENERA UNA SEÑAL SENOIDAL CON 51 MUESTRAS*/

//***** Variables Globales *****/
volatile int frec=100;
String comando;

//***** Tabla Función Senoidal *****/
int seno[ ] =
{
  2048,2304,2557,2802,3035,3252,3450,3626,
  3777,3901,3996,4060,4092,4092,4060,3996,

```

```

3901,3777,3626,3450,3252,3035,2802,2557,
2305,2048,1791,1539,1294,1061,844,646,
470,319,195,100,36,4,4,36,100,195,319,
470,646,844,1061,1294,1539,1791,2048
};

//***** Setup *****
void setup() {
  Serial.begin (9600);
  analogWriteResolution(12); // Configura resolución de 12 bits
}
//***** Loop *****
void loop()
{
  for(int i = 0; i<50;i++) // Inicia lazo para enviar los 51
  {
    if(seno[i]>4095)
      seno[i]=4095; // Valor máximo 4095

    analogWrite(DAC1, seno[i]); // Envio de datos a DAC1
    delayMicroseconds(frec); // Ajuste de Frecuencia
  }
  if (Serial.available() > 0) //si hay algo en el buzz
    DatosPorSerie();
}

//***** Leemos los comandos Recibidos y ajustamos la frecuencia *****
void DatosPorSerie()
{
  comando = Serial.readString();
  if (comando == "--")
  {
    frec=frec+15;
    if(frec>255) //si es mayor a la maxima
      frec=255;

    Serial.println("Frecuencia Disminuida");
    Serial.print("val=");
    Serial.println(frec);
  }

  if (comando == "+")
  {
    frec=frec-15;
    if(frec<1) // si es menor a la frec minima
      frec=1;

    Serial.println("Frecuencia Aumentada");
    Serial.print("val=");
    Serial.println(frec);
  }
  if (comando == "max")
  {
    frec=1;
    if(frec<1) // si es menor a la frec minima
      frec=1;

    Serial.println("Maxima Frecuencia");
    Serial.print("val=");
    Serial.println(frec);
  }
  if (comando == "min")
  {
    frec=255;
    if(frec<1) // si es menor a la frec minima
      frec=1;

    Serial.println("Minima Frecuencia");
    Serial.print("val=");
    Serial.println(frec);
  }
}

```

Conclusiones del código:

Con este código logramos obtener una señal senoidal de frecuencia variable sin la necesidad de elementos externos al micro, de todos modos, seguimos sin utilizar el acumulador de fase para variar la frecuencia y observamos que para las diferentes pruebas a realizar deberíamos tener una forma sencilla de variar la cantidad de muestras de nuestra tabla (que en este caso era una tabla fija con 51 valores).

Código 3:

En esta etapa, cambiamos la tabla fija de 51 valores por la función *createSinTable()* que en función del valor definido en *WAVE_SAMPLES* nos permite variar fácilmente la cantidad de valores que deseemos posea nuestra tabla al cargar el código en el micro.

Además para lograr mayor velocidad de conversión Digital-Analógica, en el Setup escribimos el valor 0 de salida en el DAC y luego en el loop utilizamos *dacc_write_conversion_data()* para sacar los valores deseados en función de la tabla.

```

/* PROGRAMA QUE GENERA UNA SEÑAL SENOIDAL CON "WAVE_SAMPLES" MUESTRAS*/

// ***** Variables Globales *****
#define WAVE_SAMPLES 100 //Definimos cuantos valores tendrá nuestra tabla
volatile int frec=100;
String comando;
//***** Genero Tabla de Valores Senoidales *****
uint16_t seno[WAVE_SAMPLES];
void createSineTable()
{
  for(uint32_t nIndex = 0;nIndex < WAVE_SAMPLES;nIndex++)
  {
    // Normalizada a rango de 12 bits 0-4095
    seno[nIndex] = (uint16_t) (((1+sin(((2.0*PI)/WAVE_SAMPLES)*nIndex))*4095.0)/2);
    Serial.println(seno[nIndex]);
  }
}
// ***** Setup *****
void setup()
{
  Serial.begin (9600);
  createSineTable();
  analogWriteResolution(12); // Configura resolución de 12 bits
  analogWrite(DAC1,0); // Para lograr mayor velocidad activo el DAC1 en el setup
}
// ***** Loop *****
void loop()
{
  for(int i = 0; i<WAVE_SAMPLES;i++) // Inicia lazo para enviar los 51
  {
    if(seno[i]>4095)
      seno[i]=4095; // Valor máximo 4095
    uint32_t ulOutput = seno[i]; // obtener la muestra actual
    dacc_write_conversion_data(DACC_INTERFACE, ulOutput); // escribimos el valor de la tabla en el dac
    delayMicroseconds(frec); // Ajuste de Frecuencia
  }
}

```

```
if (Serial.available() > 0) //si hay algo en el buzz
  DatosPorSerie();
}

//***** Leemos los comandos Recibidos y ajustamos la frecuencia *****
void DatosPorSerie()
{
  comando = Serial.readString();
  if (comando == "--")
  {
    frec=frec+15;
    if(frec>255) //si es mayor a la maxima
      frec=255;

    Serial.println("Frecuencia Disminuida");
    Serial.print("val=");
    Serial.println(frec);
  }

  if (comando == "+")
  {
    frec=frec-15;
    if(frec<1) // si es menor a la frec minima
      frec=1;

    Serial.println("Frecuencia Aumentada");
    Serial.print("val=");
    Serial.println(frec);
  }

  if (comando == "max")
  {
    frec=1;
    if(frec<1) // si es menor a la frec mínima
      frec=1;

    Serial.println("Máxima Frecuencia");
    Serial.print("val=");
    Serial.println(frec);
  }

  if (comando == "min")
  {
    frec=255;
    if(frec<1) // si es menor a la frec mínima
      frec=1;

    Serial.println("Mínima Frecuencia");
    Serial.print("val=");
    Serial.println(frec);
  }
}
```

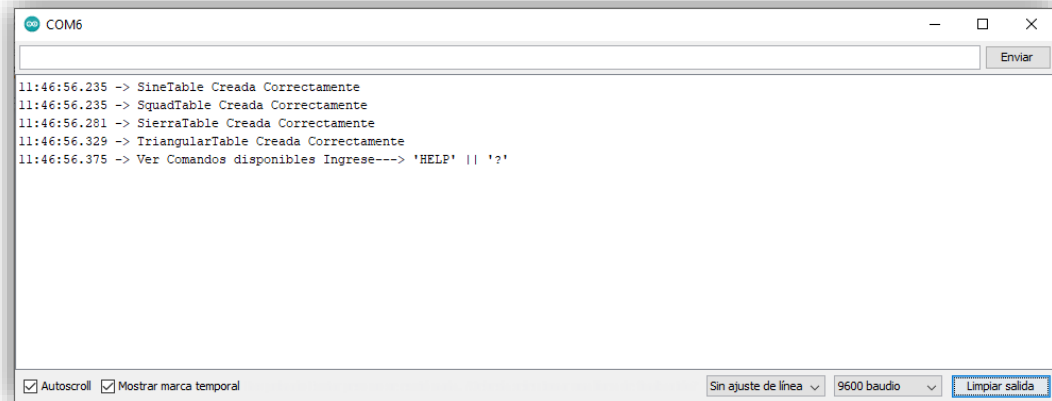
Conclusión del código:

Quedando conformes con el código y con la onda de salida obtenida pasamos a ver la mejor manera de agregar mas formas de onda y de dejar de utilizar la función *delayMicroseconds()* para variar la frecuencia.

Implementación Final:

En primer lugar, como nos habíamos propuesto en un principio, logramos controlar la salida del Arduino a través de comandos serie, en este caso utilizando el monitor serie del IDE de Arduino.

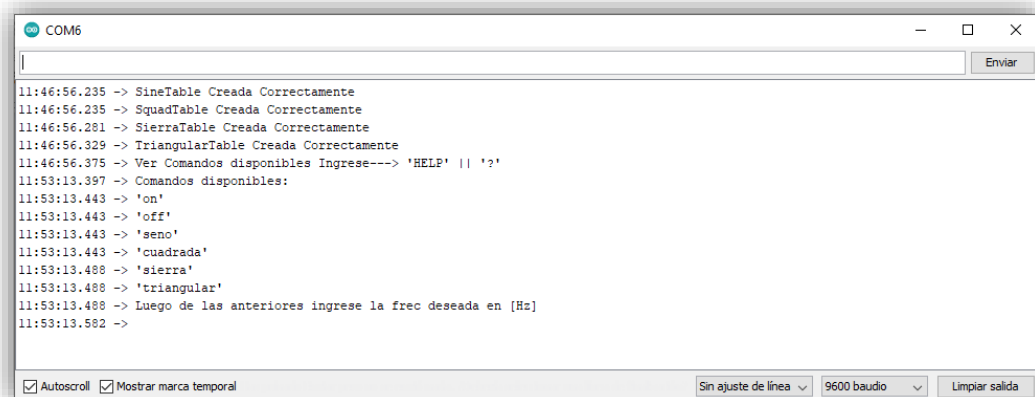
Al encender el micro obtenemos una salida como la siguiente:



```
COM6
11:46:56.235 -> SineTable Creada Correctamente
11:46:56.235 -> SquadTable Creada Correctamente
11:46:56.281 -> SierraTable Creada Correctamente
11:46:56.329 -> TriangularTable Creada Correctamente
11:46:56.375 -> Ver Comandos disponibles Ingrese---> 'HELP' || '?'

Autoscroll Mostrar marca temporal Sin ajuste de línea 9600 baudio Limpiar salida
```

En este vemos que en un principio nos muestra que las tabla con las 4 funciones principales fueron creadas correctamente, esto debido a que son creadas al prender el micro con la cantidad de muestras que definamos al cargar el código. Además, observamos que si uno no conoce los comandos disponibles el micro nos ofrece la ayuda correspondiente siendo estos los siguientes:



```
COM6
11:46:56.235 -> SineTable Creada Correctamente
11:46:56.235 -> SquadTable Creada Correctamente
11:46:56.281 -> SierraTable Creada Correctamente
11:46:56.329 -> TriangularTable Creada Correctamente
11:46:56.375 -> Ver Comandos disponibles Ingrese---> 'HELP' || '?'
11:53:13.397 -> Comandos disponibles:
11:53:13.443 -> 'on'
11:53:13.443 -> 'off'
11:53:13.443 -> 'seno'
11:53:13.443 -> 'cuadrada'
11:53:13.488 -> 'sierra'
11:53:13.488 -> 'triangular'
11:53:13.488 -> Luego de las anteriores ingrese la frec deseada en [Hz]
11:53:13.582 ->
```

Observamos que nos da la opción de prender o apagar la salida de la onda seleccionada en el DAC, de elegir el tipo de onda periódica deseada y por último de elegir la frecuencia deseada.

En el código todo esta parte la manejamos con la función *DatosPorSerie()* y para lograr la frecuencia de salida ingresada utilizamos la siguiente lógica:

- En primer lugar, leemos el valor de frecuencia deseado y lo pasamos de un string a un entero.
- Luego utilizamos la siguiente relación para convertir esa frecuencia ingresada en un valor de incremento de fase que será función de la cantidad de valores que posean nuestras tablas y de el tiempo que tarde nuestro micro en recorrer esa tabla.

$$\text{Incremento de fase} = \frac{\text{Frecuencia} - \text{Deseada} * \text{Cantidad} - \text{de} - \text{Muestras} - \text{por} - \text{Ciclo}}{\text{FREC_CLOCK_DUE}}$$

- Una vez calculado este incremento, ese valor será el que se suma al acumulador de fase en la interrupción que llame el micro (*TC4_Handler()*) y por ende tomará más o menos valores de la tabla dándonos a la salida un valor mayor(menos muestras) o menor de frecuencia(más muestras).
- Respecto a la frecuencia del timer se seleccionó una tal que nos de 39,1 [kHz] y esto se logró en el Setup a través de los siguientes comandos:

```
TC_Configure(/* clock */TC1,/* channel */1, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC |
TC_CMR_TCCLKS_TIMER_CLOCK2);
TC_SetRC(TC1, 1, 275); // sets < > 39090 hz interrupt rate
TC_Start(TC1, 1); // Iniciamos la cuenta
```

En estas 3 líneas vemos que seleccionamos el *Timer_Clock2* y que llamará a la interrupción cuando el acumulador llegue a 275 esto nos queda:

$$Frec_{clock} = \frac{86 * 10^6}{275} = 39090[Hz]$$

Luego en el código definimos que nuestras tablas (que gracias a la resolución de 12 bit del DUE pueden tener hasta $2^{12} = 4095$ muestras) tengan 3909 muestras. Esto para obtener una frecuencia mínima y un paso de 10[Hz] entre cada frecuencia de salida ya que:

$$F_{minima} = \frac{Frec_{clock}}{Muestras_{porCiclo}} = \frac{39090[Hz]}{3909} = 10[Hz]$$

Si quisiéramos una resolución de frecuencia distinta bastaría con manipular alguno de los siguientes valores:

- **Muestras_Por_Ciclo:** Tener en cuenta que solo puede aumentar hasta 4095 y que al disminuirla tendremos que considerar el límite de nyquist, además si la aumentamos, aumentamos la resolución en cuanto al paso que podemos tomar en la tabla pero tardamos más en recorrer un ciclo y por ende nuestra resolución en frecuencia disminuye.
- **Frec_Clock:** Podemos elegir un clock diferente (ya visto anteriormente) o simplemente variar el valor del acumulador (en nuestro caso es 275), si aumentamos este valor, disminuirá la frecuencia con la que se llama a la interrupción que manda los valores de la tabla a DAC, es decir, disminuiríamos Frec_Clock y por ende lograremos obtener una frecuencia mínima menor con la misma cantidad de muestras, aunque también disminuiríamos la frecuencia máxima a la cual podremos llegar.

A continuación, se adjunta el código completo:

```
// ***** Variables Globales *****
String comando; // almacena el comando ingresado al buffer serie
volatile int FuncSelect=0; // maneja que onda se desea a la salida
boolean ON=false; // prende o apaga la salida de la señal
uint32_t ulPhaseAccumulator = 0; // el acumulador de fase apunta a la muestra actual en nuestra tabla de ondas
volatile uint32_t IncrementadorFase = 1; // el incremento de fase controla la velocidad a la que nos movemos por la tabla de ondas
// valores más altos = frecuencias más altas
// Frecuencia con la cual se llamará a la interrupción del micro.
#define FREC_CLOCK_DUE 39090.0
// Definimos la cantidad de muestras en las Tablas de nuestras Ondas. Además el Paso mínimo entre frecuencias=(MUESTRAS_POR_CICLO /
FREC_CLOCK_DUE)
#define MUESTRAS_POR_CICLO 3909

/* Consideraciones extras: int es de 32 bits, en la mayoría de los casos es mejor usar uint32_t pero para arreglos grandes es mejor usar
tipos de datos más pequeños si es posible, aquí estamos almacenando muestras de 12 bits en entradas de 16 bits */
// ----- Definimos los vectores que guardaran las muestras de nuestras Tablas -----
uint16_t nSineTable[MUESTRAS_POR_CICLO];
uint16_t nSquadTable[MUESTRAS_POR_CICLO];
uint16_t nTriangularTable[MUESTRAS_POR_CICLO];
uint16_t nSierraTable[MUESTRAS_POR_CICLO];

// ***** Funciones que Calculan y Crean los valores a almacenar en nuestras Tablas en función de "MUESTRAS_POR_CICLO" *****
// ----- Funcion SENO -----
void createSineTable()
{
    for(uint32_t nIndex = 0; nIndex < MUESTRAS_POR_CICLO; nIndex++)
    {
        // Normalizada a rango de 12 bits 0-4095
        nSineTable[nIndex] = (uint16_t) (((1+sin(((2.0*PI)/MUESTRAS_POR_CICLO)*nIndex))*4095.0)/2);
        //Serial.println(nSineTable[nIndex]);
    }
    Serial.println("SineTable Creada Correctamente ");
}
// ----- Funcion CUADRADA -----
void createSquadTable()
{
    for(uint32_t nIndex = 0; nIndex < (MUESTRAS_POR_CICLO/2); nIndex++)
    {
        // Normalizada a rango de 12 bits 0-4095
        nSquadTable[nIndex] = (uint16_t) (4095);
        //Serial.println(nSquadTable[nIndex]);
    }
    for(uint32_t nIndex = MUESTRAS_POR_CICLO/2; nIndex < MUESTRAS_POR_CICLO; nIndex++)
    {
        // Normalizada a rango de 12 bits 0-4095
        nSquadTable[nIndex] = (uint16_t) (0);
        //Serial.println(nSquadTable[nIndex]);
    }
    Serial.println("SquadTable Creada Correctamente ");
}
// ----- Funcion DIENTE DE SIERRA -----
void createSierraTable()
{
    int paso= 4095/MUESTRAS_POR_CICLO;
    int acum=0;
    for(uint32_t nIndex = 0; nIndex < MUESTRAS_POR_CICLO; nIndex++)
    {
        nSierraTable[nIndex] = (uint16_t) acum;
        acum+=paso;
        //Serial.println(nSierraTable[nIndex]);
    }
    Serial.println("SierraTable Creada Correctamente ");
}
// ----- Funcion TRIANGULAR -----
```

```

void createTriangularTable()
{
    int paso= 4095/(MUESTRAS_POR_CICLO/2);
    int acum=0;
    for(uint32_t nIndex = 0; nIndex < (MUESTRAS_POR_CICLO/2); nIndex++)
    {
        nTriangularTable[nIndex] = (uint16_t) acum;
        acum+=paso;
        //Serial.println(nTriangularTable[nIndex]);
    }
    for(uint32_t nIndex = (MUESTRAS_POR_CICLO/2); nIndex < MUESTRAS_POR_CICLO; nIndex++)
    {
        acum = acum-paso;
        nTriangularTable[nIndex] = (uint16_t) acum;
        //Serial.println(nTriangularTable[nIndex]);
    }
    Serial.println("TriangularTable Creada Correctamente ");
}

// ***** SETUP *****
void setup()
{
    Serial.begin (9600);
    createSineTable();
    createSquadTable();
    createSierraTable();
    createTriangularTable();
    // enciende el reloj temporizador en el controlador de administración de energía
    pmc_set_writeprotect(false); // Permite la reprogramación
    pmc_enable_periph_clk(ID_TC4); // Activar el periférico
    //Configuramos el Timer interno del Due:
    TC_Configure(*clock */TC1,*/ channel */1, TC_CMR_WAVE | TC_CMR_WAVSEL_UP_RC | TC_CMR_TCCLKS_TIMER_CLOCK2);
    TC_SetRC(TC1, 1, 275); // sets <= 39,1 Khz interrupt rate
    TC_Start(TC1, 1); // Iniciamos la cuenta
    // habilita las interrupciones del temporizador
    TC1->TC_CHANNEL[1].TC_IER=TC_IER_CPCS; // Habilitamos las interrupciones ligadas al registro C
    TC1->TC_CHANNEL[1].TC_IDR=~TC_IER_CPCS; // Y quitamos el flag de deshabilitar interrupciones
    //Habilita la interrupción en el controlador de interrupciones vectoriales anidado
    //TC4_IRQn donde 4 es el número de temporizador * canales de temporizador (3) + el número de canal (= (1 * 3) + 1) para el temporizador1 canal
    NVIC_EnableIRQ(TC4_IRQn);

    analogWriteResolution(12); // Configura resolución de 12 bits
    analogWrite(DAC0,0); // habilitamos el DAC para optimizar tiempos y configuramos la resolucion de este
    Serial.println("Ver Comandos disponibles Ingrese--> 'HELP' || '?'");
}

// ***** Loop --> leemos si hay algun comando nuevo ingresado
*****
void loop()
{
    if (Serial.available() > 0) //si hay algo en el buzz
        DatosPorSerie();
}
//***** Interrupcion Llamada por nuestro Timer
*****
void TC4_Handler()
{
    uint32_t ultSalida;
    // Necesitamos obtener el estado para borrarlo y permitir que la interrupción se dispare nuevamente
    TC_GetStatus(TC1, 1); // Reseteamos la interrupción; de no hacerse no se volverá a llamar.
    ulPhaseAccumulator += IncrementadorFase; //Apuntamos a la siguiente muestra en nuestra tabla.
    if(ulPhaseAccumulator > MUESTRAS_POR_CICLO-1) // si el acumulador de fase se desborda -> hemos pasado por un ciclo y lo volvemos a 0
        ulPhaseAccumulator=0;
    if(ON)
    {
        switch(FuncSelect)
        {
            case(0):
                ultSalida = nSineTable[ulPhaseAccumulator]; // obtener la muestra actual
                dacc_write_conversion_data(DACC_INTERFACE, ultSalida); // escribimos el valor de la tabla en el dac
                break;
            case(1):
                ultSalida = nSquadTable[ulPhaseAccumulator]; // obtener la muestra actual
                dacc_write_conversion_data(DACC_INTERFACE, ultSalida); // escribimos el valor de la tabla en el dac
                break;
            case(2):
                ultSalida = nSierraTable[ulPhaseAccumulator]; // obtener la muestra actual
                dacc_write_conversion_data(DACC_INTERFACE, ultSalida); // escribimos el valor de la tabla en el dac
                break;
            case(3):
                ultSalida = nTriangularTable[ulPhaseAccumulator]; // obtener la muestra actual
                dacc_write_conversion_data(DACC_INTERFACE, ultSalida); // escribimos el valor de la tabla en el dac
                break;
        }
    }
}
//***** En esta funcion se encuentra la logica que controla la entrada de comandos via serie *****
void DatosPorSerie()
{
    String comando= Serial.readString();
    if (comando == "help" || comando == "?")
    {
        Serial.println("Comandos disponibles:");
        Serial.println("on");
        Serial.println("off");
        Serial.println("seno");
        Serial.println("cuadrada");
        Serial.println("sierra");
        Serial.println("triangular");
        Serial.println("Luego de las anteriores ingrese la frec deseada en [Hz]");
        Serial.println(" ");
    }
}

```

```

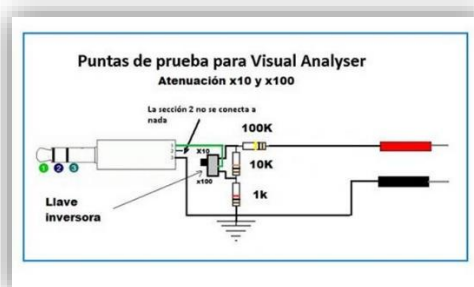
    }
    else if (comando == "on")
    {
        ON=true;
        Serial.println("SISTEMA ON");
    }
    else if (comando == "off")
    {
        ON=false;
        Serial.println("SISTEMA OFF");
    }
    else if (comando == "seno")
    {
        FuncSelect=0;
        Serial.println("Funcion SENO seleccionada");
    }
    else if (comando == "cuadrada")
    {
        FuncSelect=1;
        Serial.println("Funcion CUADRADA seleccionada");
    }
    else if (comando == "sierra")
    {
        FuncSelect=2;
        Serial.println("Funcion DIENTE SIERRA seleccionada");
    }
    else if (comando == "triangular")
    {
        FuncSelect=3;
        Serial.println("Funcion TRIANGULAR seleccionada");
    }
    else if (comando != "cuadrada" || comando != "sierra" || comando != "triangular" || comando != "seno" || comando != "?" || comando != "on" || comando !=
"off")
    {
        IncrementadorFase =(comando.toInt() *MUESTRAS_POR_CICLO)/FREC_CLOCK_DUE; // En funcion de la frec deseada calculamos el incremento de
fase necesario
        float frecSalida=(FREC_CLOCK_DUE*IncrementadorFase)/MUESTRAS_POR_CICLO; // Mostramos via serial la frec de salida.
        Serial.print("Frecuencia Salida=");
        Serial.println(frecSalida);
        Serial.print("Paso=");
        Serial.println(IncrementadorFase);
    }
}

```

Pruebas Realizadas:

Debido a la situación actual de Pandemia si bien no se pudo tener acceso a un osciloscopio, se utilizó la placa de sonido de la PC con y el software Visual_Analyzer para lograr darnos una idea de la salida que estábamos obteniendo al implementar los diferentes códigos en el Arduino DUE.

Se implementó además para mayor seguridad en la placa de audio la siguiente punta de prueba



Limitantes de utilizar Visual_Analyzer:

Debido a los filtros de continua de la placa de audio y de su Ancho de Banda (30 Hz hasta 20KHz dependiendo la placa de audio) sabemos de ante mano que podremos visualizar bien frecuencias de señales senoidales comprendidas entre esas frecuencias, pero para señales como la cuadrada o la diente de sierra que tienen armónicos mayores no las podremos visualizar de la mejor manera aunque si nos dará una salida aproximada pero con falta de armónicos.

Comprobación en Matlab de las Funciones con las cuales se realizan las tablas de las diferentes ondas:

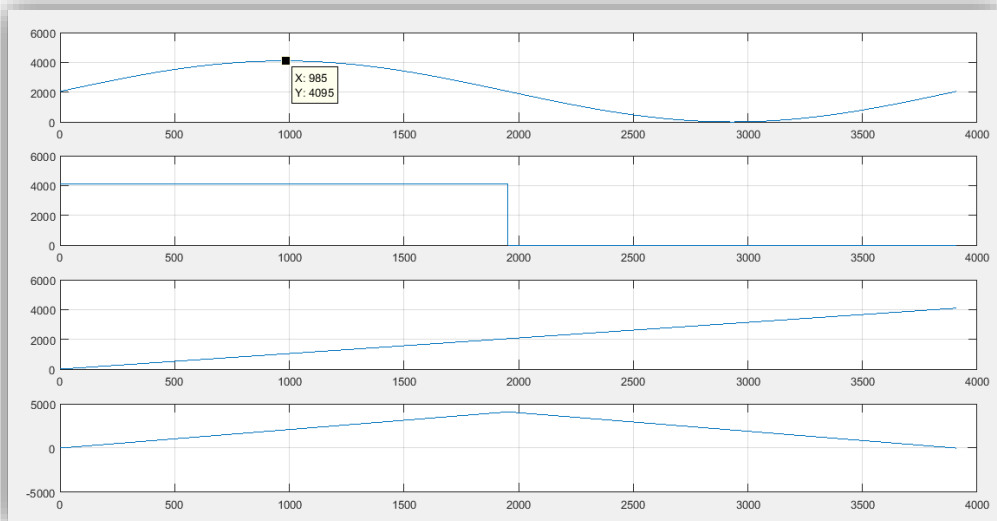
Para verificar que los valores de las ondas que cargamos cuando se inicia el micro no posean errores y se encuentren en el rango del DAC (0 y 4095) graficamos las mismas funciones en Matlab con n=3909.

```
close all
clear all
clc
commandwindow

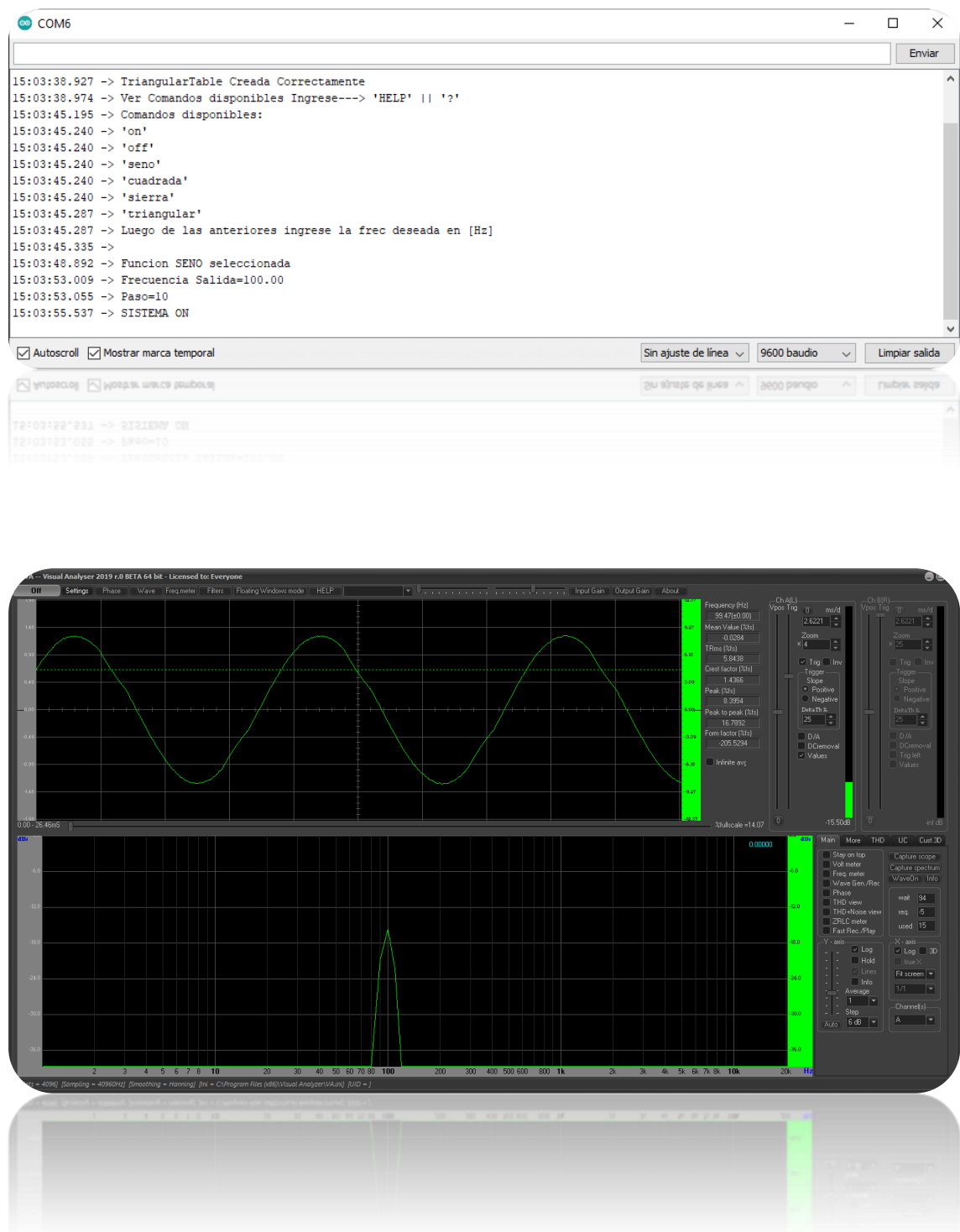
MUESTRAS_POR_CICLO=3909;
% ***** Senoidal *****
for nIndex=1 : MUESTRAS_POR_CICLO
    %Normalizada a rango de 12 bits 0-4095
    nSineTable(nIndex)=((1+sin((2.0*
pi)/MUESTRAS_POR_CICLO)*nIndex))*4095.0)/2);
end
% Grafico
subplot(4,1,1);
plot(nSineTable);
grid on;
% ***** Cuadrada *****
for nIndex = 1: MUESTRAS_POR_CICLO
    nSquadTable(nIndex) = 4095;
end
for nIndex = 1954 : MUESTRAS_POR_CICLO
    nSquadTable(nIndex)= 0;
end
subplot(4,1,2);
plot(nSquadTable);
grid on;
% ***** Diente Sierra *****
paso= 4095/MUESTRAS_POR_CICLO;
acum=0;
for nIndex = 1: MUESTRAS_POR_CICLO
    nSierraTable(nIndex) =acum;
    acum =acum+paso;
end
subplot(4,1,3);
plot(nSierraTable);
grid on;

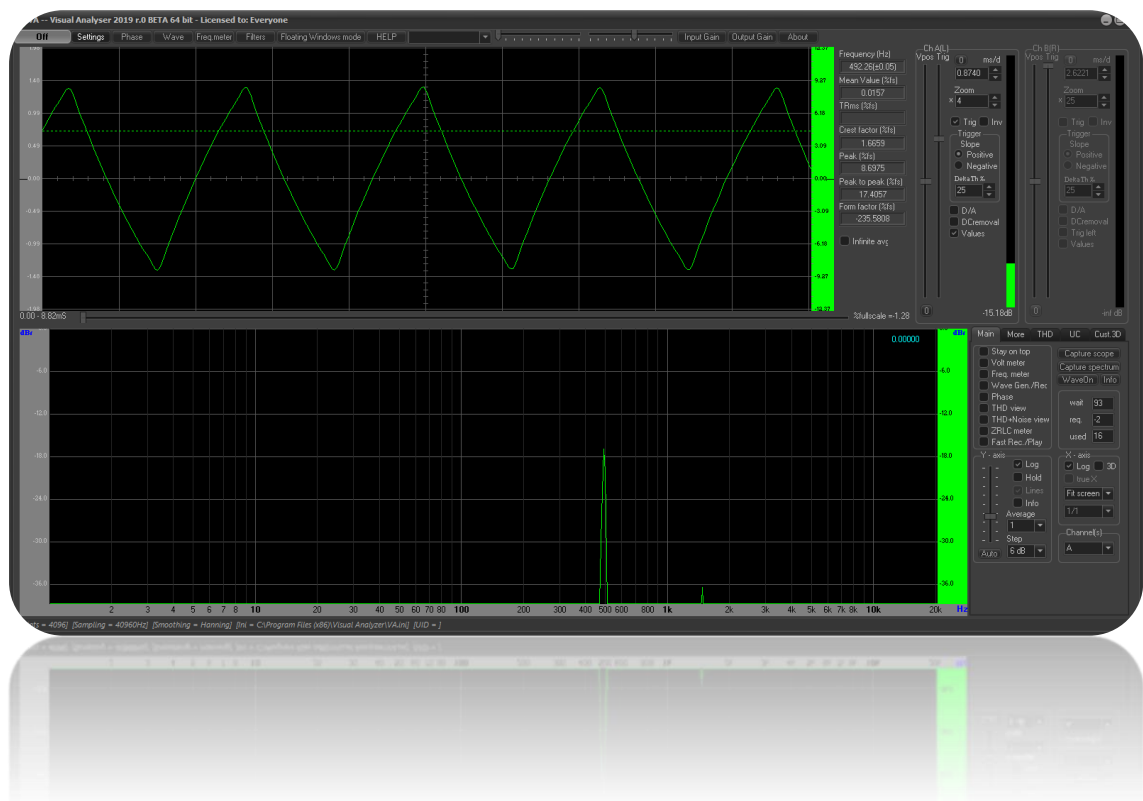
% ***** Triangular *****
paso= 4095/(MUESTRAS_POR_CICLO/2);
acum=0;
for nIndex = 1: 1953
    nTriangularTable(nIndex) = acum;
```

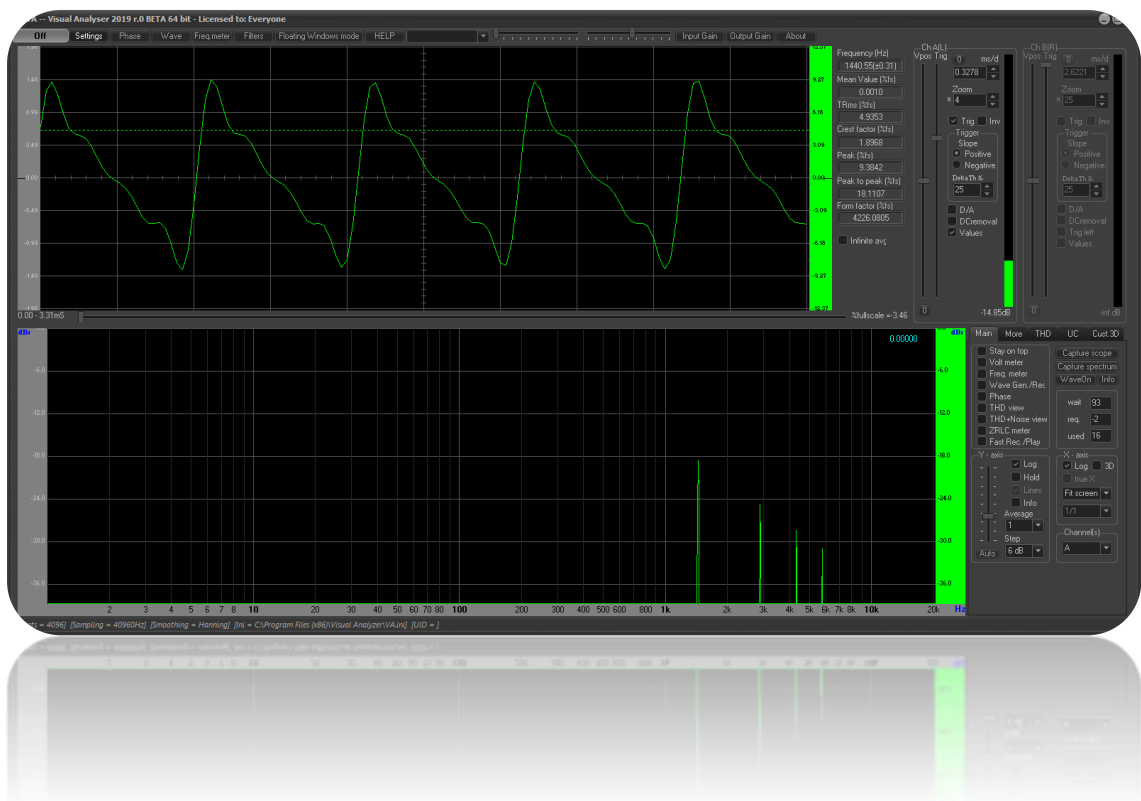
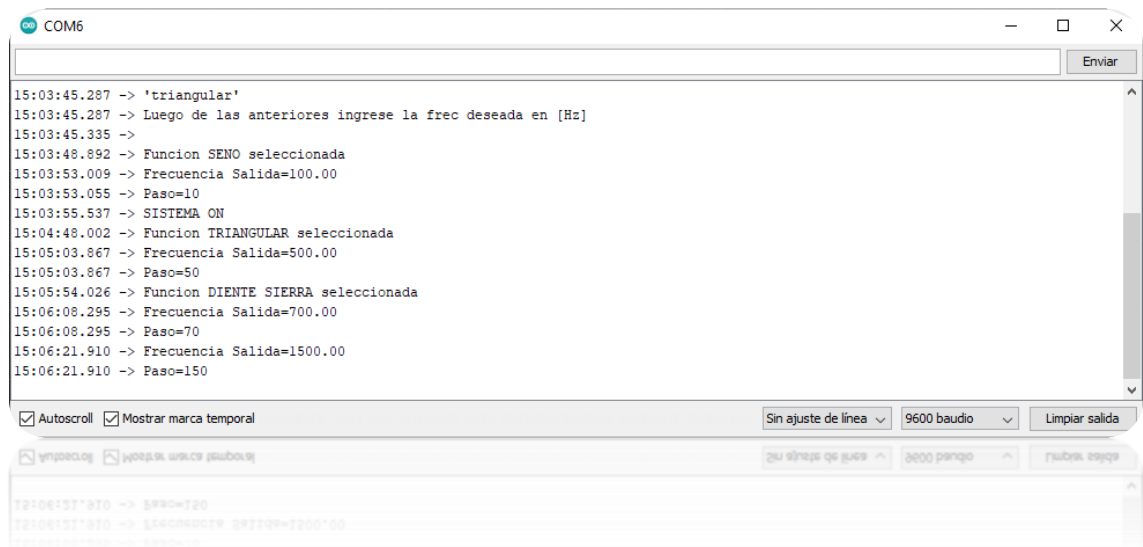
```
    acum= acum+paso;  
end  
for nIndex = 1954: MUESTRAS_POR_CICLO  
    acum = acum-paso;  
    nTriangularTable(nIndex) = acum;  
end  
subplot(4,1,4);  
plot(nTriangularTable);  
grid on;
```

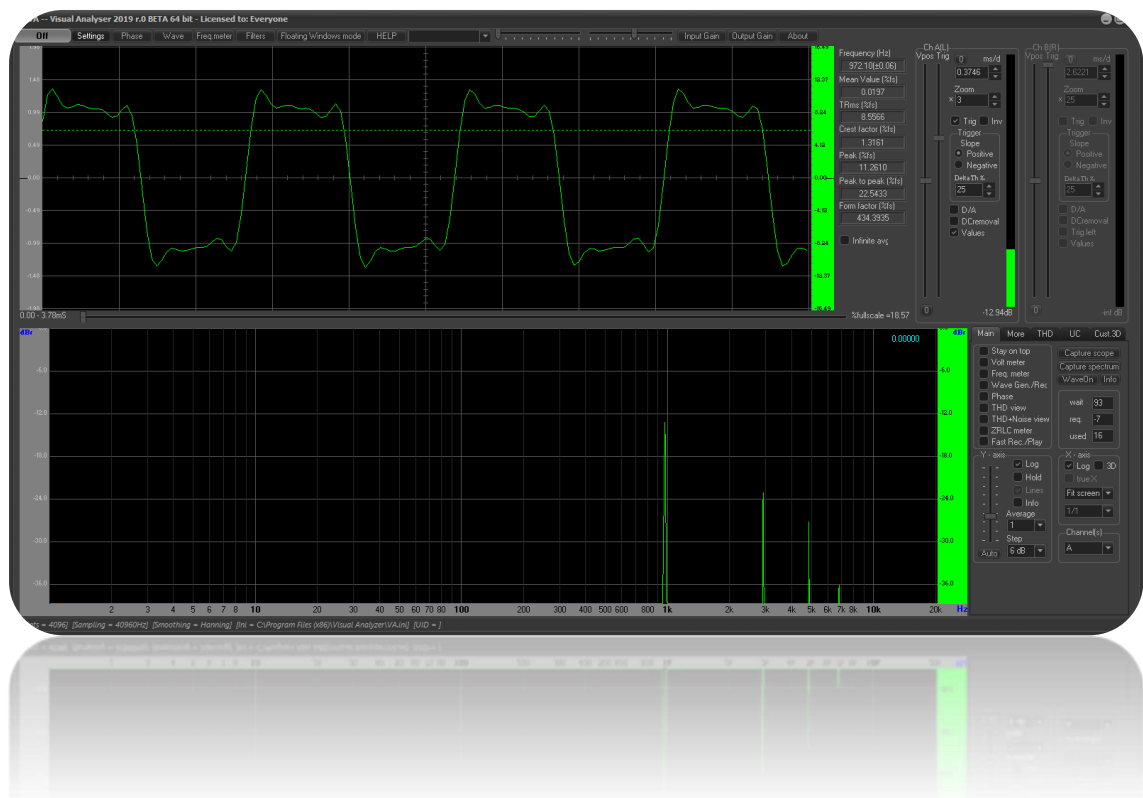
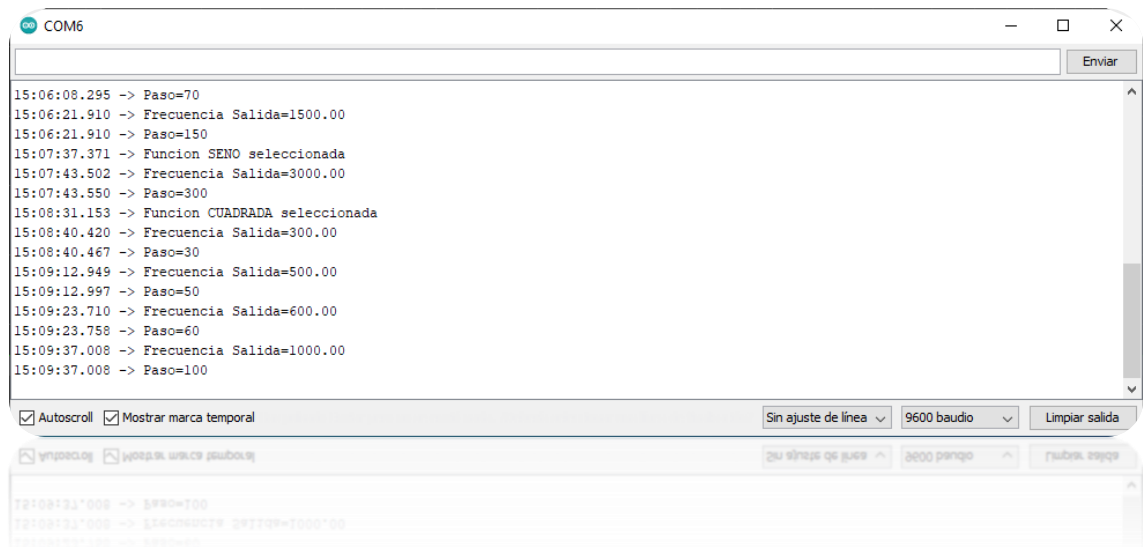


Imágenes Visualizadas en Visual Analyzer:









Conclusiones del código:

- Logramos demostrar fielmente con tan solo un micro y un pc el método de SDD obteniendo 4 ondas se salida de frecuencia variable (entre los 10 Hz y los 7kHz a pasos de 10 Hz). Además, haciendo uso de la teoría, de los valores de frecuencia de clock del micro y de la cantidad de muestras logramos calcular el paso para sacar la frecuencia deseada que se ingrese al micro via serie en tiempo real.
- Al no contar con un instrumento de medición de precisión(osciloscopio) debido al contexto de pandemia nos quedará pendiente un ajuste mas fino de la frecuencia de salida pero el código queda planteado para que al modificar el acumulador sea fácil de calibrar y de esta manera obtener una salida de mayor precisión.

CONCLUSIONES GENERALES

Los componentes estrictamente digitales pueden operar notablemente rápido y con precisión en comparación con una solución de síntesis analógica, a menudo limitada por la precisión del reloj de entrada en lugar de las limitaciones impuestas por el acumulador o memoria. Sin embargo, el DAC introduce problemas como la degradación de señal y un cuello de botella de velocidad de señal que se requiere para producir una forma de onda útil. Por lo tanto, el DAC generalmente es el componente de limitación de rendimiento dentro de un DDS completo.

- Los generadores de señales con tecnología DDS pueden producir formas de onda periódicas en muchas frecuencias con una precisión de frecuencia extrema. Esto se debe al acceso único a la memoria y al mecanismo de sincronización.
- La tecnología DDS se implementa con tres bloques de hardware de nivel superior: el reloj de muestra, el acumulador de fase y la tabla de búsqueda.
- El reloj de muestra crea la palabra de sintonización de frecuencia, actualiza el valor del acumulador de fase y controla la velocidad de salida del DAC.
- El acumulador de fase toma la palabra de sintonización de frecuencia como entrada y proporciona la dirección de memoria digital de la siguiente muestra que se emitirá en la tabla de búsqueda.
- La tabla de búsqueda almacena las formas de onda periódicas como muestras digitales. La tabla de búsqueda toma la dirección de memoria del acumulador de fase y proporciona la muestra de forma de onda digital en esa dirección de memoria al DAC.
- Los generadores de señal con tecnología DDS deben usarse para aplicaciones que requieren una generación de frecuencia precisa o agilidad de frecuencia.

- Las aplicaciones que requieren formas de onda extremadamente grandes, complejas y definidas por el usuario pueden ser mejor atendidas por generadores de formas de onda arbitrarias en lugar de generadores de funciones arbitrarias con tecnología DDS.

Bibliografía

[¿Qué es la síntesis digital directa?](#)

<https://www.arrow.com/es-mx/research-and-events/articles/dds-vs-das>

<https://eming.cl/tag/sintesis-digital-directa/>

<https://www.digikey.com/es/articles/the-basics-of-direct-digital-synthesizers-ddss>

<http://tux.iar.unlp.edu.ar/desarrollo003.htm>

<http://ko7m.blogspot.com/2015/01/arduino-due-timers-part-1.html>

<https://github.com/spellboundfx/sfx-arduino/blob/master/funkGenie/funkGenie.ino>

<https://forum.arduino.cc/index.php?topic=592839.0>

<http://rcarduino.blogspot.com/2012/12/arduino-due-dds-part-1-sinewaves-and.html>

<https://github.com/duerrfk/Due-DDS>

<https://espaciodecesar.com/tag/puntas-de-prueba/>