CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

IMPLEMENTACIÓN DE SISTEMAS DE PROCESAMIENTO DE SEÑALES E IMÁGENES MEDIANTE LA TARJETA DE DESAROLLO ATLYS DE XILINX Y SIMULINK.

"PROTOTIPO"

SUSTENTANTE:

DIEGO ARMANDO HERNÁNDEZ RAMÍREZ

CARRERA:

TECNÓLOGO EN ELECTRÓNICA Y COMUNICACIONES.

ASESOR:

MTRO. JOSÉ MARÍA VALENCIA VELASCO

GUADALAJARA, JAL. A 11 DE JULIO DE 2015.

Hoja de Aprobación.

Uso de Simulink y la Tarjeta de desarrollo Atlys de Xilinx para la implementación de Sistemas de Procesamiento de Señales e imágenes.

Diego Armando Hernández Ramírez.

Asesor técnico: Mtro. José María Valencia Velasco.

Asesor metodológico: Mtro. Cuauhtémoc Rafael Aguilera Galicia.

Resumen

Placeholder, TBD;

Agradecimientos

Índice

Índice de figuras

Índice de cuadros

Parte I

Introducción.

1. Planteamiento del problema.

En la actualidad la electrónica está presente en prácticamente en todos los aspectos de nuestra vida a través de una gran infinidad de dispositivos y sistemas: teléfonos inteligentes, monitores de ritmo cardiaco, cámaras fotográficas, televisores, automóviles, refrigeradores, computadoras, etc.

Todos estos dispositivos realizan de manera interna la manipulación e interpretación de señales eléctricas, que en otras palabras es lo que se conoce como procesamiento de señales. El procesamiento de una señal puede aplicarse, por ejemplo, en el reconocimiento de voz para determinar quién es la persona que habla; para determinar, mediante una imagen, piezas defectuosas en una línea de producción o para la protección de información (encriptación).

El procesamiento de señales involucra la realización de operaciones matemáticas sobre las señales, las cuales son llevadas por sistemas cuya única función es precisamente el llevar a cabo esas operaciones, los procesadores digitales de señales (DSP, por sus siglas en inglés) y los arreglos programables (FPGA, por sus siglas en inglés) son los encargados de ello.

En muchas aplicaciones de procesamiento de señales se requiere una velocidad de procesamiento elevada (por ejemplo procesamiento de video) por lo que, debido al paralelismo de su operación, los FPGA son aptos para ser utilizados en ellas[**?**].

Con el fin de explotar las ventajas que los FPGA poseen y poderlos aplicar de una manera eficaz en el procesamiento de señales es necesario contar con sólidos conocimientos principalmente en metodologías de diseño digital e implementación matemática de algoritmos; estos conocimientos deben adquirirse desde la academia, puesto que es el tiempo ideal en que el futuro ingeniero o arquitecto de sistemas puede ir desarrollando, a través de la experimentación, las habilidades necesarias para crear prototipos en el que se involucre el procesamiento de señales.

Teniendo como objetivo principal el recortar la curva de aprendizaje, las empresas líderes en FPGA como Xilinx, Altera y Synopsys proporcionan plataformas de trabajo que puede interactuar

con Matlab (software especializado que permite la implementación y prueba de algoritmos). De esta forma, el alumno puede poner en práctica de forma ágil y sin complicaciones los conocimientos adquiridos en las áreas de procesamiento digital de señales.

Muchas veces la información que el fabricante proporciona sobre sus plataformas de trabajo es escasa y poco concreta, lo que puede impactar negativamente en el interés del alumno, provocando que los conocimientos y conceptos no queden del todo entendidos.

2. Propósito.

2.1. Objetivo general.

- Describir el proceso de implementación de un sistema de procesamiento de señales e imágenes mediante hardware reconfigurable (FPGA) y la tarjeta de desarrollo Atlys.

2.2. Objetivos específicos.

- Facilitar el diseño e implementación de un sistema de procesamiento de audio en tiempo real, basado en el desarrollo de un algoritmo de eco, así como un sistema de detección de bordes en una imagen basado en el algoritmo Sobel, ambos utilizando bloques de Xilinx System Generator para Simulink.
- Mostrar la conversión de los algoritmos matemáticos básicos que intervienen en el procesamiento de señales, a hardware en FPGA, haciendo uso de la abstracción que proporciona Simulink
- Describir las técnicas de implementación más eficientes para obtener el mayor rendimiento sobre la familia FPGA Spartan 6 utilizada en la tarjeta Atlys
- Diseñar las propiedades intelectuales (IPs) más comunes en el tratamiento de señales tales como bloques de filtros FIR, IIR y convoluciones, utilizando los entornos de programación de MATLAB® y Xilinx®.

- Explicar los diferentes métodos de ejecución del hardware diseñado en Simulink®, sobre la tarjeta Atlys.
- Justificar el uso de MATLAB/Simulink® y Xilinx/ISE® para el diseño e implementación de algoritmos complejos en contraste con el uso tradicional de HDL puro.

Parte II

Antecedentes.

3. Procesamiento Digital de Señales.

3.1. Elementos de un sistema DSP.

El procesamiento digital de señales es una de las tecnologías más vanguardistas que ha marcado varios segmentos tecnológicos como las comunicaciones digitales, ciencias médicas, diseño de radares, reproductores de música de alta fidelidad, por nombrar sólo algunas. Esta área se distingue de todas las demás dentro de las Ciencias de la Computación, por el tipo de datos único que utiliza: **las señales [?].**

El propósito principal de un sistema de procesamiento digital de señales es manipular matemáticamente algún tipo de información tomada del mundo real. Dicha información es naturalmente analógica, es decir, la representación de sus valores y funciones son continuos, por lo que necesita ser previamente digitalizada antes de ser procesada. Algunos ejemplos de señales comúnmente utilizados son voz, audio, video, temperatura y presión.

Generalmente las operaciones que se realizan sobre dichas señales son adiciones, sustracciones, multiplicaciones y divisiones, mismas que se deben ejecutar muy rápidamente con el fin de generar una salida mucho más precisa[**?**]. Esto se hace para cumplir una amplia variedad de objetivos, tales como: mejoras en la visualización de imágenes, reconocimiento y generación de voz, compresión de datos para almacenamiento y transmisión, etc.

El procesamiento digital de señales analógicas puede ser descrito en tres etapas:

- La señal analógica es *digitalizada*, es decir, se *muestrea* y cada muestra es a su vez, *cuantizada* a un número finito de bits. Este proceso es llamado **conversión analógico-digital**.
- Los muestreos digitalizados son procesados por un *Procesador Digital de Señales.*

- Las muestras resultantes de la etapa de procesamiento, se convierten de nuevo a un formato *analógico* mediante alguna técnica de reconstrucción analógica (**conversión digital-analógico**).

Un sistema DSP típico se muestra en la Figura 3.1.

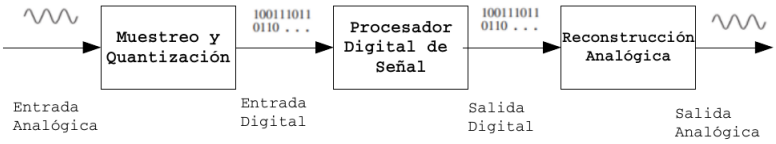

Figura 3.1: Diagrama a bloques de un Sistema de procesamiento Digital de Señales.

Las secciones a continuación describen con más detalle, los teoremas y técnicas fundamentales que intervienen en los sistemas DSP.

3.1.1. Teorema de muestreo.

El *teorema de muestreo* establece que para una representación precisa de una señal $\mathbf{x(t)}$ por sus muestras de tiempo $\mathbf{x(nT)}$, dos condiciones deben cumplirse:

- La señal $\mathbf{x(t)}$ debe ser de banda limitada, es decir, su espectro de frecuencia debe ser limitada para contener las frecuencias hasta cierta frecuencia máxima, digamos $\mathbf{f_{max}}$, y sin frecuencias más allá de eso. Un espectro típico de banda limitada se muestra en la Figura 3.2.
- La frecuencia de muestreo $\mathbf{f_s}$debe ser elegida para ser al menos el doble de la frecuencia máxima $\mathbf{f_s}$, es decir,

$$\mathbf{f_s \leq 2f_{max}} \tag{3.1}$$

- En términos del intervalo del tiempo de muestreo:

$$\mathbf{T \leq \frac{1}{2f_{max}}} \tag{3.2}$$

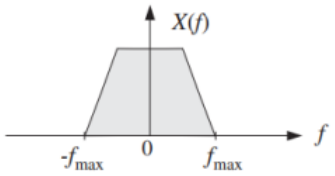

Figura 3.2: Banda de espectro limitado típica [**?**].

La velocidad de muestreo mínima permitida por el teorema de muestreo, que es (4.1), es llamada *Tasa de muestreo de Nyquist.* Para valores arbitrarios de $\mathbf{f_s}$, la magnitud de $\frac{\mathbf{f_s}}{\mathbf{2}}$ es llamada *Frecuencia de Nyquist***.** Esta frecuencia también define las frecuencias de corte de los filtros pasa bajas que se requieren en las operaciones en DSP-.

3.1.2. Cuantización.

El muestreo y la cuantización son requisitos indispensables para cualquier operación de procesamiento digital en señales analógicas. Una señal digital es una secuencia de números[1] en donde cada muestra es representada por un número finito de digitos[2].La cuantización es el proceso de convertir una señal discreta en el tiempo de amplitud continua, a una señal digital, representando cada valor muestreado como un número finito de digitos. La Figura 3.3 muestra la representación gráfica de una señal cuantizada.

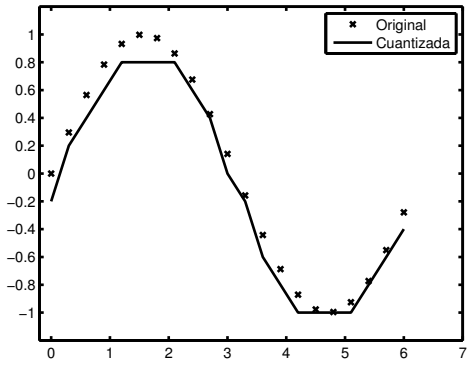

Figura 3.3: Ejemplo de cuantización en una señal senoidal. Los puntos marcados con una «x» representan la señal original, mientras que las coordenadas marcadas con «–» muestran la señal cuantizada .

[1]Esta secuencia es conocida como **Muestra.**

[2]En el diseño de sistemas DSP, se dice que la señal es de **precisión finita** cuando la representación está dada por un número finito de datos.

Este proceso induce un error llamado *error de cuantización,* el cual se debe al cambio en la representación de la señal original, de un valor continuo a un set de valores discretos. En términos matemáticos, la operación de cuantización de las muestras $\mathbf{x(n)}$ se denota como $\mathbf{Q[x(n)]}$. Tomando $\mathbf{x_q(n)}$ como la secuencia de muestras cuantizadas, el sistema completo queda como:

$$\mathbf{x_q(n)} = \mathbf{Q[x(n)]} \tag{3.3}$$

El error de cuantización es representado por la secuencia $\mathbf{e_q(n)}$, como la diferencia entre los valores cuantizados y el valor de la muestra actual:

$$\mathbf{e_q x(n)} = \mathbf{x_q(n)} - \mathbf{x(n)} \tag{3.4}$$

3.2. Sistemas Discretos Lineales e Invariantes en el Tiempo..

La relación entrada/salida de de los sistemas *Lineales e Invariantes en el Tiempo* (LTI, por sus siglas en inglés) está definida por la *convolución* en tiempo discreto de la respuesta del impulso finito aplicado a la entrada del sistema.

Los sistemas LTI pueden ser clasificados en dos tipos, dependiendo de si su respuesta al impulso tiene duración finita o infinita, estos son: *Respuesta al Impulso Finito* (FIR, por sus siglas en inglés) o *Respuesta al Impulso Infinito* (IIR, por sus siglas en inglés).

En términos matemáticos, se dice que un sistema es lineal cuando, por ejemplo, dos señales de entrada $\mathbf{x_1(t)}$ y $\mathbf{x_2(t)}$ tienen salidas $\mathbf{y_1(t)}$ y $\mathbf{y_2(t)}$ respectivamente. Entonces, la salida del sistema al impulso $\alpha_1\mathbf{x_1(t)} + \alpha_2\mathbf{x_2(t)}$ es $\alpha_1\mathbf{y_1(t)} + \alpha_2\mathbf{y_2(t)}$. Es decir, cumplen con las reglas de *homogeneidad* y *superposición.*

También, un sistema es invariante en el tiempo cuando $\mathbf{y(t)}$ es la salida correspondiente a $\mathbf{x(t)}$, entonces para cada τ, $\mathbf{y(t-\tau)}$ es la salida que corresponde a $\mathbf{x(t-\tau)}$. Es decir, si agregamos un retraso a la entrada o salida, el resultado debe ser exactamente el mismo.

Además, los *sistemas LTI* deben ser causales, lo que significa que la salida del sistema no puede anticipar la entrada del mismo, tal que, para todo impulso de entrada $\delta\mathbf{(t)}$, la salida $\mathbf{h(t) = 0}$ mientras $\mathbf{t < 0}$.

Otra característica que debe cumplir, es la de ser un sistema sin memoria. Se dice que un sistema tiene memoria cuando la señal de salida depende de las entradas pasadas y/o futuras. Por ejemplo, la ecuación del cálculo del voltaje de un resistor representa un sistema sin memoria dado que $\mathbf{v(t) = Ri(t)}$, mientras que el voltaje en un capacitor representa un sistema con memoria por su ecuación $\mathbf{v(t) = v(t_0) + \int_{t_0}^{t} i(\tau)d\tau}$.

En términos de hardware, todos los *sistemas LTI* pueden ser implementados a base de sumadores, multiplicadores y unidades de retraso. Estos sistemas son facilmente realizables dado que se basan a partir de un número finito de elementos, como los antes mencionados.

Las siguientes subsecciones resumen conceptos importantes que son utilizados en cualquier aplicación y diseño de sistemas DSP.

3.2.1. Convolución.

La convolución es la forma matemática de combinar dos señales para formar una tercera. Es la técnica más importante en el Procesamiento Digital de Señales. Usando la estrategia de la descomposición del impulso, los sistemas pueden ser descritos por una señal llamada *respuesta al impulso.* La convolución es importante porque relaciona las tres señales de interés: la señal de entrada, la señal de salida y la señal de respuesta al impulso.

Un punto fundamental a comprender de los sistemas DSP, es que estos trabajan descompiniendo la señal de entrada en simples componentes aditivos, cada uno de estos componentes se pasa a través de un sistema linear, y los componentes de salida resultantes son sintetizados, o en otras palabras, sumados. Esta descomposición se puede hacer de dos formas distintas: *Descomposición por impulsos* y *Descomposición por el método de Fourier.* Cuando la Descomposición por impulsos es utilizada, el procedimiento se puede describir matemáticamente utilizando la **convolución.**

Esta operación se basa en dos términos importantes en los sistemas DSP. El primero es la **función delta,** simbolizada por $\delta[\mathbf{n}]$. La función delta es un impulso normalizado, que es, la muestra número cero con un valor asignado de una unidad, mientras que las otras muestras tienen asignadas un valor de cero. Por esta razón, la función delta es llamada **unidad de impulso.**

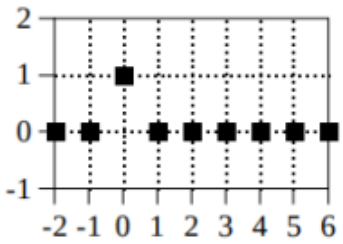

Figura 3.4: Función delta graficada.

Por otra parte, la **respuesta al impulso** es la señal que existe en el sistema cuando una función delta es aplicada a la entrada. Si dos sistemas son diferentes en cualquier manera, ambos tendrán diferentes respuestas al impulso. Comunmente, las respuestas a la entrada y salida son llamadas $\mathbf{x}[\mathbf{n}]$ y $\mathbf{y}[\mathbf{n}]$, la respuesta al impulso es usualmente llamada $\mathbf{h}[\mathbf{n}]$.

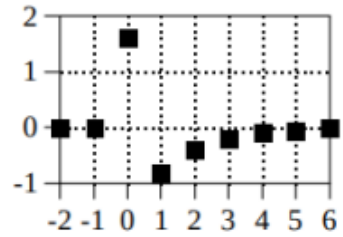

Figura 3.5: Función de respuesta al impulso graficada.

En otras palabras, una señal de entrada $\mathbf{x}[\mathbf{n}]$, entra en un sistema linear con una respuesta al impulso $\mathbf{h}[\mathbf{n}]$, resultando en una señal de salida $\mathbf{y}[\mathbf{n}]$. En forma matemática, la ecuación de convolución de una sola muestra queda como se muestra a continuación:

$$\mathbf{y}[\mathbf{n}] = \mathbf{x}[\mathbf{n}] * \mathbf{y}[\mathbf{n}] \tag{3.5}$$

Tomando la ecuación 4.5 como guía, siendo $\mathbf{x}[\mathbf{n}]$ una señal de N puntos, desplazándose desde 0 a *N-1*, y $\mathbf{h}[\mathbf{n}]$ una señal de M puntos desplazándose de 0 a *M-1*[**?**] , la convolución de ambas señales es una señal *N+M-1*ejecutándose desde 0 a *N+M-2*, esto es:

$$\mathbf{y}[\mathbf{i}] = \sum_{\mathbf{j=0}}^{\mathbf{M-1}} \mathbf{h}[\mathbf{j}]\mathbf{x}[\mathbf{i}-\mathbf{j}] \tag{3.6}$$

La ecuación anterior representa la definición formal de la convolución, la cual también es conocida como la *suma de convolución* o *convolución discreta.*

Un ejemplo de la convolución tomando como vectores de entrada $\overrightarrow{v}_{\mathbf{actual}} = [\mathbf{v_1}, \mathbf{v_2} \ldots \mathbf{v_n}]$ y $\overrightarrow{v}_{\mathbf{anterior}} = [\mathbf{v_n}, \mathbf{v_{n-1}} \ldots \mathbf{v_1}]$ se muestra en la **Figura 3.6.**

En el **Apéndice D** se muestra el script interactivo que se puede ejecutar en el ambiente de Matlab, el cual fue tomado de [**?**] . Este script muestra el resultado de la convolución discreta de dos impulsos $\mathbf{y}[\mathbf{n}] = \delta[\mathbf{n}] * \delta[\mathbf{n} - \mathbf{1}]$, tomando como vector a $\mathbf{n} = [\mathbf{1}, \mathbf{0}, \mathbf{1}]$.

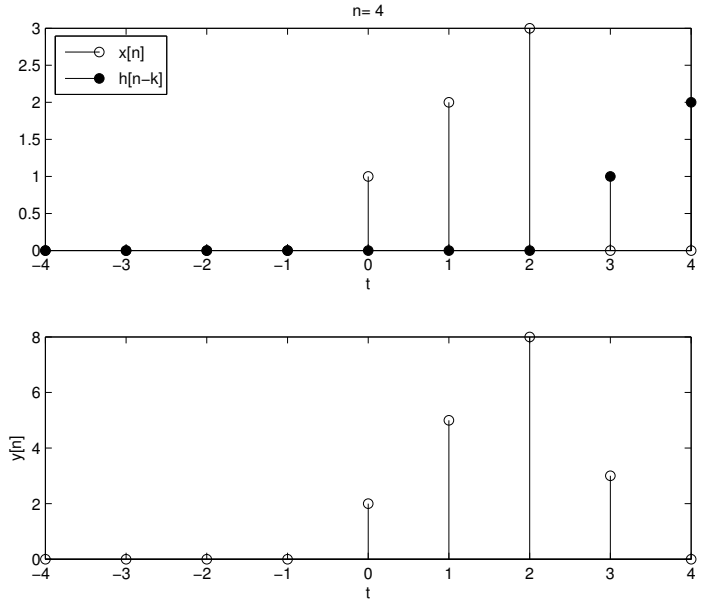

Figura 3.6: Convolución de dos vectores en Matlab.

3.2.2. Filtros FIR.

Un filtro digital de *Respuesta Finita al Impulso* (FIR) es un *sistema LTI* si es definido por un conjunto de coeficientes constantes. La salida de un filtro FIR de órden (o longitud) **L**, a la respuesta de impulso unitario aplicado a la entrada $\mathbf{x}[\mathbf{n}]$, está dada por una versión finita de la ecuación de convolución, como se muestra a continuación:

$$\mathbf{y}[\mathbf{n}] = \mathbf{f}[\mathbf{n}] * \mathbf{x}[\mathbf{n}] = \sum_{\mathbf{k=0}}^{\mathbf{L-1}} \mathbf{f}[\mathbf{k}]\mathbf{x}[\mathbf{n} - \mathbf{k}] \tag{3.7}$$

Para un tren de impulsos en el dominio del tiempo a la entrada $\mathbf{x[n]}$, la ecuación queda como:

$$\mathbf{y[n] = b_0 x(n) + b_1 x(n-1) + ... + b_{M-1} x(n-M+1)} \tag{3.8}$$

$$= \sum_{\mathbf{k=0}}^{\mathbf{M-1}} \mathbf{b_k x(n-k)} \tag{3.9}$$

En otras palabras, la respuesta al impulso consiste sólo de respuesta en los coeficientes, procedida y antecedida por ceros (el filtro producirá una respuesta que irá decayendo a cero y se mantendrá en ese estado, de ahí el nombre característico de este filtro). Matemáticamente se puede expresar esta respuesta al impulso con la siguiente ecuación:

$$\mathbf{h(n)} = \begin{cases} b_n, & 0 \leq n \leq M-1 \\ 0, & otros \end{cases} \tag{3.10}$$

Gráficamente, esta ecuación se puede representar como se muestra en la **Figura 3.7.**

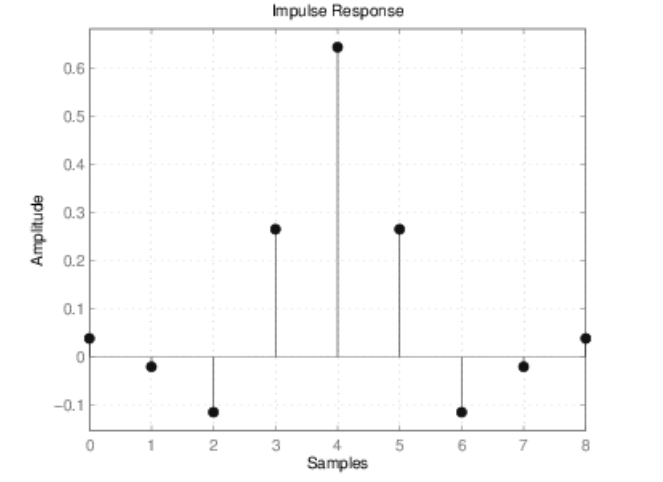

Figura 3.7: Respuesta al impulso de un filtro FIR con $\mathbf{b_0 = 0.9}$.

Este tipo de filtros son más populares en cuanto a implementación, dado que cuentan con características muy útiles, entre las cuales destacan:

- ***Fase Lineal:*** Esta propiedad implica que la fase es una función lineal de la frecuencia, Esto asegura que las señales de todas las frecuencias se retrasan en la misma cantidad de tiempo, eliminando la posibilidad de distorsión de fase.
- ***Estabilidad:*** Para una entrada finita, la salida siempre es finita, además son no recursivos, es decir, no hay una conexión de retroalimentación envuelta en la estructura del filtro.

Existen muchos métodos de implementación de esto filtros, la estructura más básica es conocida como *forma directa,* la cual consta de utilizar la ecuación en diferencia, no recursiva, mostrada en (4.7)[**?**], lo que es equivalente a la sumatoria convolucional. La estructura se muestra en la **Figura 3.8**, el cual representa un filtro con un número de coeficientes $\mathbf{b_{0..3}} + \mathbf{1}$**.**

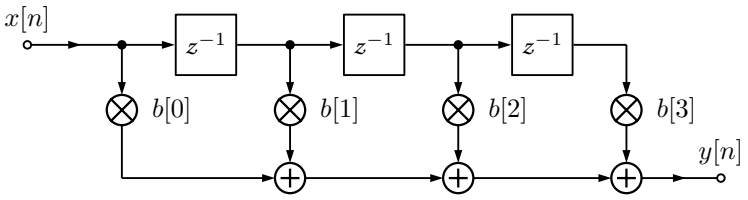

Figura 3.8: Estructura en Forma Directa de un Filtro FIR. Imagen creada usando el paquete TikZ de LaTeX

En términos generales, existen cuatro formas básicas de implementación de este tipo de filtros, las cuales son:

- Método por ventanas (Rectangular, Barlett, Hanning, Hamming, Blackman y Kaiser).
- Muestreo en frecuencia.
- Aproximación de Chebyshev y algoritmo de intercambio de Remez (conocido como método de Rizado Constante).
- Mínimos Cuadrados.

La implementación más popular utilizada en FPGA es la del Método por Ventanas debido a que muchos de los paquetes de software incluidos en las herramientas de diseño de filtros, utilizan este método como el principal, lo cual recorta el tiempo de desarrollo del mismo.

3.2.3. Filtros IIR.

El sistema de Respuesta Infinita al Impulso (*IIR* por sus siglas en inglés) se caracteriza por utilizar las muestras de la señal de salida en instantes anteriores en adición a las muestras presentes más las muestras pasadas de la misma función de salida, es decir, este filtro cuenta con lazos de *retroalimentación* y *anticipación,* por lo que es conocido como un **sistema discreto recursivo**, a diferencia del filtro FIR que se caracteriza por ser **no recursivo**, esto significa que la función de salida del filtro puede continuar indefinidamente aún cuando la entrada tienda a cero y permanezca en cero.

El filtro IIR es un sistema de clase lineal, invariante en el tiempo que se representa en función de una ecuación en diferencia:

$$y[n] = \sum_{k=1}^{N} a_k y[n-k] + \sum_{k=0}^{M} b_k x[n-k] \tag{3.11}$$

La primera suma representa la parte "auto-regresiva" o la parte IIR, y la segunda suma representa el "promedio móvil" o la parte FIR[**?**].

Estos filtros *IIR* tienen una relación con los demás filtros en tiempo continuo. Ambos tienen respuesta al impulso de longitud infinita y son descritos por funciones de transferencia racionales en el dominio de la frecuencia. El método en que estos filtros se diseñan, a partir de estructuras discretas, es usando métodos de aproximación numérica, como los métodos de *Chevysheb*, *Butterworth* o *Bessel*, además de que es posible utilizar un gran número de tablas pre diseñadas, disponibles en los libros afines al tema en cuestión[**?**] . Dado que este tipo de filtros no serán utilizados en el marco metodológico de este documento, sólo se abarcarán las características fundamentales de ellos.

El formato de implementación del filtro IIR utilizando el ***método de Chevysheb*** minimiza la diferencia absoulta entre la respuesta en frecuencia ideal y la actual sobre toda la banda de paso mediante la incorporación de rizo uniforme en la banda de paso. La transición desde la banda de paso hacia la banda de corte es más rápida que en el formato *Butterworth.*

El filtro de ***Butterworth*** representa la mejor aproximación de un filtro ideal pasa bajas en frecuencias analógicas. La respuesta en la banda de paso y la banda de corte es máximamente plana[**?**].

El filtro de ***Bessel*** no se puede implementar digitalmente utilizando muchas las herramientas comerciales de modelado matemático, por esta razón, no es muy común en el campo del diseño DSP. Además, estos filtros requieren un órden más alto, por lo que la utilización de elementos discretos para su implementación se dispara, volviendo al filtro de *Bessel* muy complicado en términos de implementación.

La estructura de un filtro IIR se puede representar mediante la ecuación **(4.11)**, en donde se puede observar que es similar a tener un filtro FIR que comprende los coeficientes de **b** y otro filtro FIR inverso con los coeficientes de **a**, como se muestra en la **Figura** 3.9.

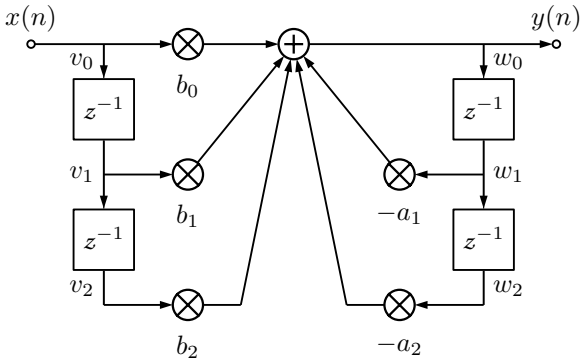

Figura 3.9: Estructura en Forma Directa de un Filtro IIR. Imagen creada usando el paquete TikZ de LaTeX

4. Plataformas FPGA.

4.1. Descripción general.

La Matriz de Compuertas Programables en Campo (FPGA) es un circuito integrado reconfigurable que puede ser utilizado para diseñar circuitos digitales. La configuración de la FPGA es normalmente especificada usando lenguajes de descripción de hardware como SystemVerilog o VHDL y después es traducida, mediante herramientas de síntesis, a un formato binario en el cual se encuentra la información de ruteo e interconexiones necesarias para que el dispositivo ejecute las funciones lógicas para el cual fue diseñado. Esta propiedad de ser reconfigurable y poder ejecutar múltiples tareas tan complejas o sencillas, de forma paralela, ofrece una significante ventaja en muchas aplicaciones como por ejemplo en el diseño de circuitos integrados donde a diferencia del prototipado con ASIC, en donde los diseñadores no tienen la flexibilidad de hacer modificaciones

al prototipo después de que el chip ha sido manufacturado, en el FPGA es posible y muy común el modificar algunas partes del circuito después de que el proyecto ha sido concluido.

La arquitectura de una FPGA se basa en Bloques de Matrices Configurables (CLBs, por sus siglas en Inglés), las cuales proporcionan la lógica programable y una jerarquía de interconexiones reconfigurables para interconectar los CLBs entre sí. Además de estos componentes básicos, las FPGA actuales contienen bloques de memoria internos, controladores para interfaces externas de alta velocidad como memorias DDR y bloques físicos de PCI Express, así como bloques optimizados para operaciones de DSP, microprocesadores físicos y algunas funciones especiales más, que dependen del enfoque o familia de la FPGA, todo esto en el mismo silicio.

La tendencia reciente en la tecnología FPGA es trabajar con arquitecturas de hardware en alto nivel de abstracción, agregarle bloques DSP, procesadores embebidos y transductores de alta velocidad para formar un Sistema Programable en Chip completo (SoPC). Además, las FPGA toman ventaja del paralelismo natural del hardware, ya que exceden el poder computacional de los Procesadores Digitales de Señales, rompiendo el paradigma de la ejecución secuencial y lograr un mayor rendimiento.

Una de las aplicaciones principales de las FPGA es poder ejecutar y modificar arquitecturas digitales múltiples veces, hasta que se ha cumplido el objetivo del prototipo que se estableció al principio, sin ser necesario recurrir a los costosos procesos de fabricación de Circuitos Integrados personalizados. Gracias a esto, se pueden implementar diseños de manera incremental e incluso hacer cambios iterativamente en cuestión de horas en lugar de semanas. También, debido a la creciente oferta de herramientas de diseño en alto nivel, se ha decrementado la curva de aprendizaje y con frecuencia, estas herramientas incluyen valiosas Propiedades Intelectuales (IP) para control y procesamiento de señales avanzadas.

Existe una numerosa cantidad de fabricantes pero sólo dos tipos de FPGAs: Reprogramables (basadas en SRAM o Flash) y Programables una sola vez (OTP). Las FPGAs basadas en SRAM necesitan una memoria de configuración y no retienen los datos cuando son desconectadas de la fuente de alimentación. Las que son basadas en Flash, no necesitan una memoria externa para almacenar la configuración y la pueden mantener aún cuando el dispositivo no está energizado. Anteriormente, las FPGAs basadas en Flash tenían la característica de ser OTP, pero hoy en día existen dispositivos basados en esta tecnología que pueden ser reprogramados tales como las MAX 10 de Altera.

En la próxima sección se cubrirá a detalle, la arquitectura de la familia Spartan6 de Xilinx, ya que es esta la que se encuentra en el kit de desarrollo Atlys de Digilent.

4.2. Estructura general de las FPGA.

Como se mencionó anteriormente, las FPGA modernas ofrecen una serie de componentes que son de gran utilidad al momento de diseñar sistemas digitales. Estos básicamente son:

- Bloques Lógicos Configurables (CLB) para poder implementar funciones lógicas así como registros.
- Memoria en Chip (On-chip memory) que provee almacenamiento de datos dentro del FPGA, generalmente es reducido dado que el área de construcción de memoria en el silicio, tiende a ocupar gran parte de este.
- Propiedades Intelectuales físicas, tales como controladores Ethernet MAC, Transductores, Multiplicadores optimizados, bloques DSP, Procesadores, Controladores de memoria externa DDR, PCIe endpoint físico, etc.
- Recursos de manejo de reloj que generen las frecuencias necesarias para controlar dispositivos como los antes mencionados y que además, puedan ser distribuidos dentro de la FPGA. Esto es muy importante al momento de diseñar sistemas con un alto índice de transferencia de datos.
- Bloques de entrada y salida que comuniquen a la FPGA al mundo exterior.
- Recursos de ruteo para proveer la interconectividad de los Bloques Lógicos Configurables internos y las Propiedades Intelectuales.

La **Figura 4.1** muestra la arquitectura típica de una FPGA con los bloques de construcción básicos. Es importante mencionar que algunos elementos como las Block RAM, bloques DSP, controladores de memoria, etcétera, que se muestran en la imagen, son construidos sobre el mismo silicio, sin quitarle espacio a los elementos lógicos. También cabe mencionar que, las Tablas de Búsqueda, mejor conocidas como Look Up Tables (LUT) que hay dentro de los bloques lógicos son usadas para crear funciones de lógica combinacional, pero también pueden ser configuradas como memorias RAM o

registros de desplazamiento. Esta es una forma muy eficiente de inferir dichos registros sin tener que usar los elementos de almacenamiento.

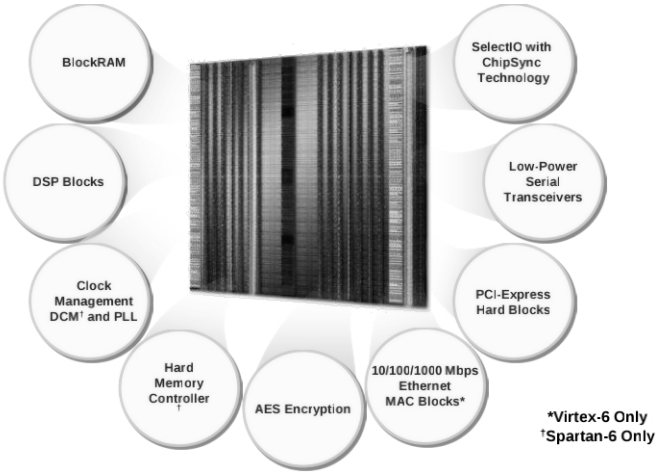

Figura 4.1: Arquitectura general de las FPGA [**?**].

4.3. Arquitectura de la familia de FPGAs Xilinx Spartan-6.

La familia Spartan-6 está fuertemente enfocada a proveer capacidades de integración de sistemas con el menor costo posible para aplicaciones de alto volumen, es decir, es una línea de dispositivos que tienen bloques de comunicación de alta velocidad como PCI Express, controladores de memoria externa DDR3 y Ethernet, entre otros. Además de una vasta densidad de elementos lógicos y registros disponibles que van desde 3,840 hasta 147,443 celdas lógicas, dependiendo del dispositivo seleccionado por el diseñador. Consume la mitad de la potencia comparado con la familia anterior de FPGAs Spartan 3, gracias a que están construidas con una avanzada tecnología de 45nm. Esta línea de FPGAs son muy populares ya que son el balance óptimo entre costo, potencia y rendimiento[**?**].

La innovación más notable en estas FPGA es la re estructuración de la arquitectura interna para implementar LUTs de 6 entradas y doble registro de salida en cada LUT, esto significa que una sola LUT puede implementar funciones lógicas de $2^6 = 64$bits, como por ejemplo, una RAM de 64 bits o un registro de desplazamiento de 32 bits. Anteriormente, la arquitectura se basaba en LUTs de 4 entradas, como se muestra a continuación.

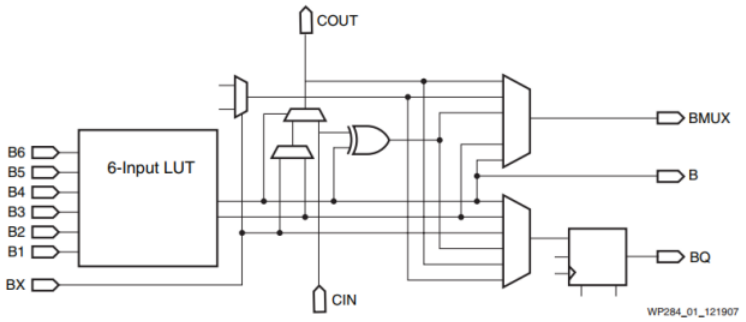

Figura 4.2: LUT de seis entradas [?].

Además, la familia Spartan-6 incluye bloques de memoria RAM (BRAM) de 18Kb, una optimización de dispositivos DSP48A1 los cuales sirven para ejecutar cálculos complejos de manera paralela, controladores físicos de memoria SDRAM, bloques de manejo de reloj internos mejorados para poder generar las frecuencias necesarias para controladores de alta velocidad, así como opciones de configuración y seguridad de IP más avanzados.

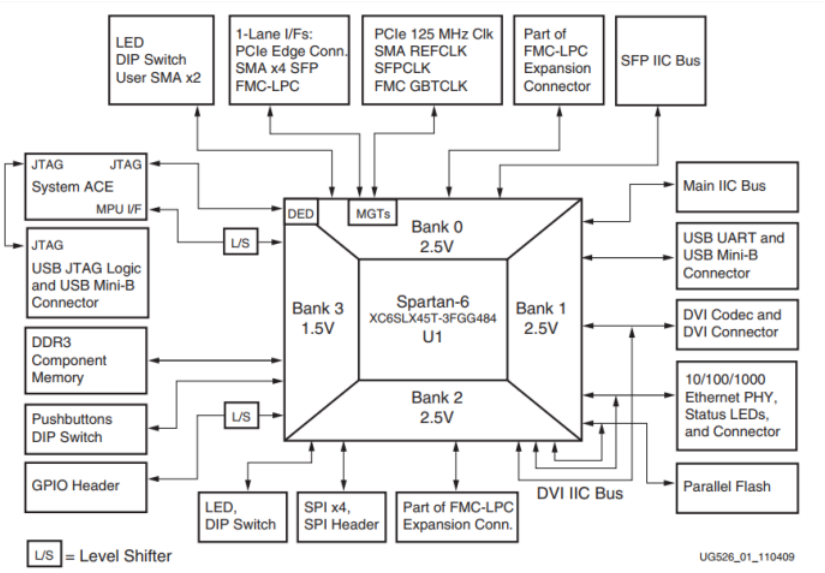

Figura 4.3: Diagrama a bloques de una FPGA Spartan-6 [?].

Debido a la construcción en 45nm, se han podido incorporar una mayor cantidad de CLBs en esta familia de FPGAs. Los CLBs son los recursos lógicos principales necesarios para implementar circuitos secuenciales y combinatorios. Cada elemento CLB es conectado a una matriz de switches programables para acceder a otra matriz de ruteo como se muestra en la Figura 4. Cada elemento

CLB contiene un par de SLICEs. Estos dos SLICEs no tienen una conexión directa entre si. Cada SLICE tiene un bloque de acarreo en cadena (carry chain).

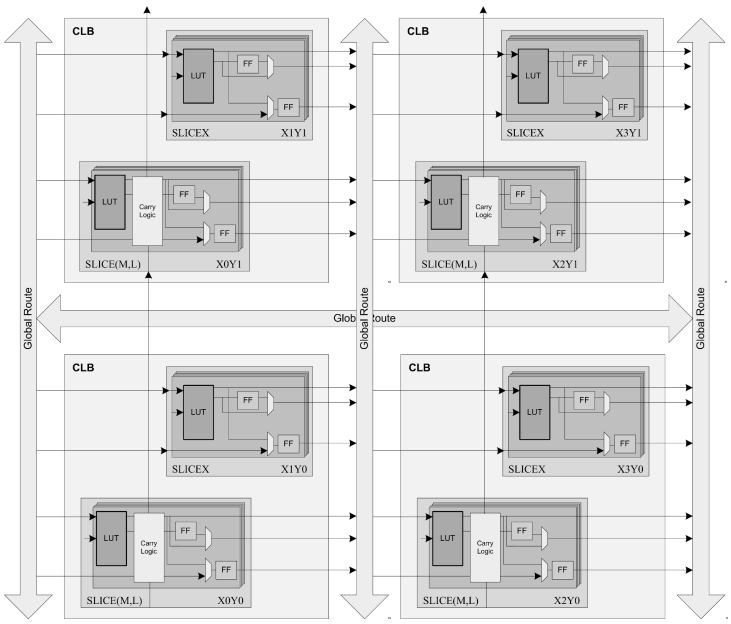

Figura 4.4: Bloque Lógico Configurable.

Cada SLICE contiene cuatro LUTs, cuatro elementos de almacenamiento (flip flop), un amplio número de multiplexores y un bloque de acarreo lógico. Esos elementos son usados por todos los SLICEs para proveer las funciones lógicas, aritméticas y algunos tipos de memoria ROM. Adicionalmente, algunos SLICEs pueden implementar dos funciones adicionales: almacenar datos al adoptar la función de RAM distribuida y desplazar datos adoptando la función de registro de desplazamiento de 32 bits. Son llamados SLICEM (por Memoria), los comunes son llamados SLICEL (Por Lógico).

La Figura describe con detalle la arquitectura de cada SLICE en un CLB. Los multiplexores antes mencionados sirven para proveer la conectividad entre los recursos lógicos que rodean a los CLBs, mientras que la red de elementos de acarreo en cadena dentro de los CLBs, hacen la función de ruteo para generar sumadores lógicos más eficientes.

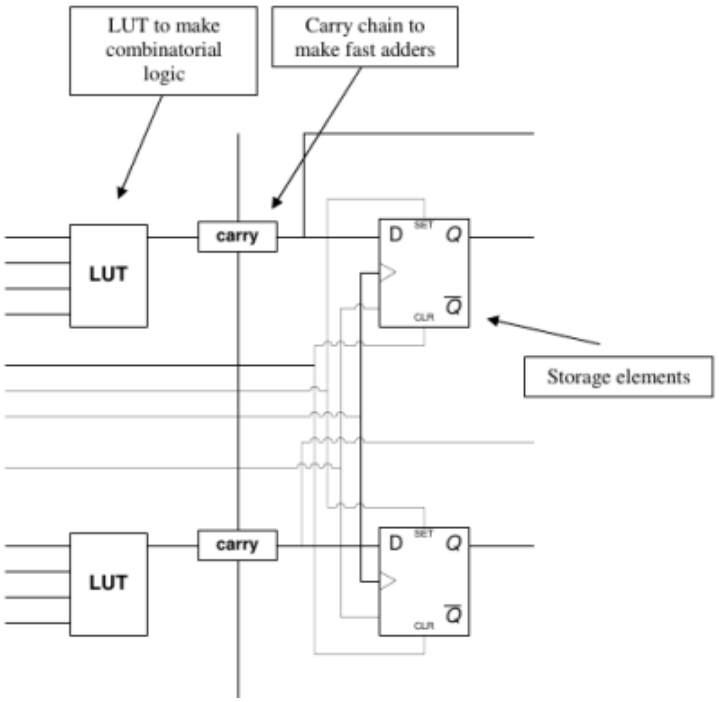

Figura 4.5: Estructura detallada de un SLICE [**?**].

Los dispositivos Spartan-6 cuentan con un gran número de memorias BRAM de 18Kb, las cuales están construidas por dos memorias controladas independientemente de 9Kb cada una. Estas memorias son colocadas en columnas, el número total de ellas depende del tamaño del dispositivo Spartan-6. Las memorias BRAM se pueden utilizar en cascada para habilitar implementaciones de mayor tamaño de Kilo bits, con un pequeño impacto en el timing. La Figura 6 muestra una BRAM en cascada con dos distintos puertos de lectura y escritura.

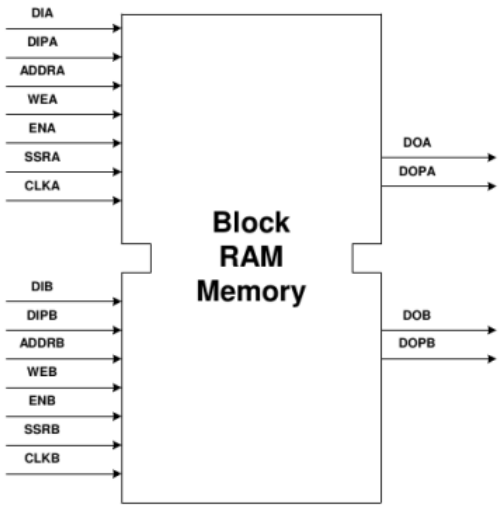

Figura 4.6: Block RAM de doble puerto [**?**].

4.4. Kit de desarollo Atlys de Digilent.

El sistema de desarrollo Atlys fabricado por Digilent bajo el programa Xilinx University Program es la plataforma seleccionada para cumplir con los objetivos de esta investigación. El kit Atlys es una plataforma avanzada para el desarollo de sistemas digitales. Está basada en un dispositivo FPGA Xilinx Spartan-6 LX45, la familia LX está optimizada para implementaciones lógicas de alto rendimiento[**?**].

Los periféricos que se incluyen son de gama alta, tales como controladores Gigabit Ethernet, entrada y salida de video HDMI, memoria RAM DDR2 de 128 Megabytes de almacenamiento en un bus de 16 bits, además de puertos de USB host y entrada y salida de audio a través de un códec AC97. Este kit es compatible con toda la suite de diseño de Xilinx ISE, por lo que se pueden implementar una gran variedad de diseños para diferentes áreas de investigación con un costo relativamente bajo.

Las principales características técnicas de esta plataforma se muestran a continuación:

- FPGA Spartan-6 LX45-3 con 6,822 slices, cada una contiene 4 LUTS de 6 entradas y 8 flip-flops, con una frecuencia máxima de ejecución a 500MHz.[3]

[3]Para una mayor comprensión de estas catacterísitcas técnicas, referirse a la sección ***"4.3 Arquitectura de la familia de FPGAs Xilinx Spartan-6"*** de este documento.

- 58 DSP slices (elementos lógicos optimizados para operaciones de DSP).
- Oscilador CMOS a 100 MHz como reloj base para PLL/DCM.
- Códec de audio AC97 con líneas de entrada/salida de audio y ariculares.
- Dos entradas y dos salidas de video HDMI.

Entre algunas otras. En la **Figura 4.7** se puede observar una imagen de esta plataforma.

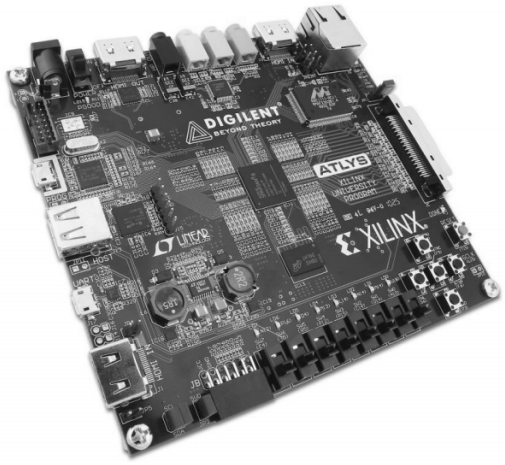

Figura 4.7: Kit de desarrollo Atlys Spartan-6 de Digilent, obtenida del sitio web del fabricante[**?**].

5. Plataformas de Software para el diseño de sistemas DSP.

5.1. Introducción a MATLAB y Simulink.

5.1.1. MATLAB.

El nombre de esta herramienta proviene de recortar y unir las palabras **MAT**rix **LAB**oratory (Laboratorio de Matrices), lo cual nos da una referencia de la metodología que utiliza. MATLAB es un sistema interactivo cuyo elemento básico son matrices las cuales, no requieren ser dimensionadas. Está enfocado a resolver problemas técnicos computacionales a través de un ambiente completo donde los problemas y soluciones son expresados en una notación matemática familiar[**?**]. Además, la integración de herramientas para la adquisición de datos externos tales como sonidos e imágenes,

así como el análisis a través de aplicaciones de visualización como gráficas, hacen de este lenguaje uno de los mas atractivos en el área de las ciencias aplicadas.

Lo que hace de MATLAB una herramienta muy popular en el ámbito académico e industrial, es la rapidez con la que se pueden implementar y resolver algoritmos de mediana a gran complejidad. Esto se debe en gran medida a la facilidad que brinda el intérprete interactivo para crear scripts conocidos como Archivos-M (M-files, por sus siglas en Inglés) que ayudan a describir sistemas con eficacia, así como el gran número de funciones prediseñadas incluidas en los complementos conocidos como «cajas de herramienta» o Toolboxes, los cuales contienen colecciones de funciones para un gran rango de disciplinas tales como procesamiento de señales, sistemas de control, redes neuronales, lógica difusa, por mencionar algunas. Todo esto ayuda a que el usuario implemente sus modelos matemáticos en cuestión de horas y no días, comparado con lenguajes compilados no interactivos como C o Fortran.

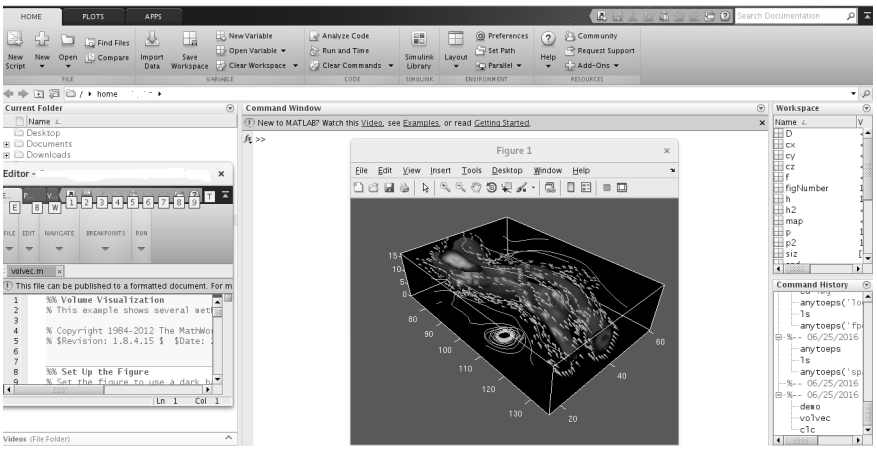

Figura 5.1: Interfaz Gráfica del Usuario de MATLAB.

MATLAB es considerado un lenguaje de programación de alto nivel, debido a la gran abstracción de datos con los que trabaja. Algunas propiedades importantes de esta herramienta son:

- El tipo de datos por defecto es una matriz de doble precisión, lo que significa que se pueden representar números desde 0 a $1.7977e + 308$.
- Es un lenguaje orientado a objetos, lo cual resulta en una mejora en el manejo de la complejidad de aplicaciones y estructuras de datos.

- Los algoritmos diseñados en este paquete de software, pueden ser convertidos a código en lenguaje C, HDL y/o PLC, para ser ejecutados en dispositivos embebidos.
- Matlab en conjunto con simulink, tienen soporte para la herramienta de diseño DSP de Xilinx, lo cual convierte a este ambiente en uno de los más atractivos en este ámbito.

5.1.2. Simulink.

MATLAB tiene un toolbox adicional llamado Simulink que proporciona el modelado, la simulación y el análisis de sistemas dinámicos, dentro de un entorno gráfico. Este sofwtare permite el diseño modular y jerárjico, ayudando al usuario en el manejo y creación de sistemas complejos que son conceptualmente simplificados.

Debido a este amplio uso, la capacidad de diseñar y verificar las implementaciones de hardware desde el mismo ambiente de Simulink, se traduce en una mayor ventaja para el prototipado rápido de nuevas teorías y diseños. Añadiendo las múltiples capas de software y librerias que habilitan la comunicación del ambiente gráfico de Simulink con el mundo exterior, proceso conocido como simulación de *hardware-in-the-loop* (co simulación con módulos de adquisición de datos en hardware, que estimulan entradas y/o envian resultados al modelo de Simulink), proporciona un enorme beneficio adicional en el diseño y verificación del algoritmo sin arriesgar pérdidas de hardware o errores de implementación[**?**].

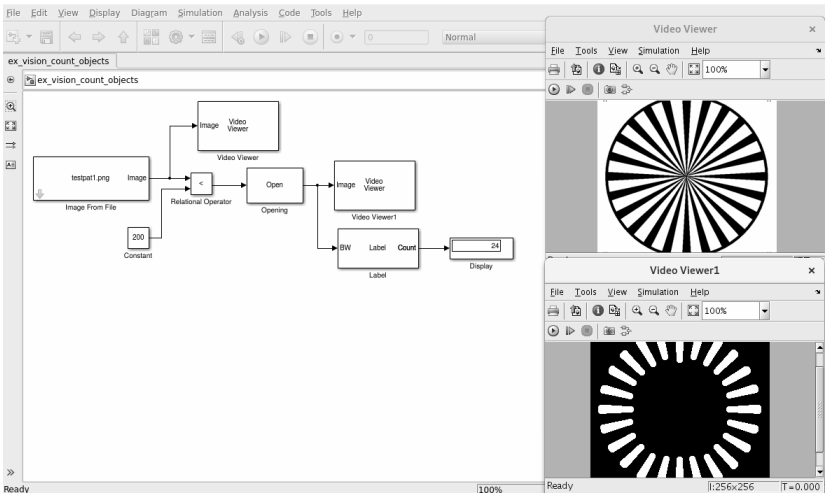

Figura 5.2: Entorno de modelado gráfico Simulink de MATLAB. En este ejemplo se puede observar, como se lleva a cabo la ejecución de un algoritmo de detección de elementos en una imagen, utilizando solamente 8 bloques del *Image Processing Toolbox* de Simulink.

Sin embargo, el comportamiento de la simulación matemática de Simulink en comparación con la implementación en el hardware, no es exacto. Simulink permite que calculos de punto flotante complejos sean completados en un solo paso "virtual", mediante el proceso de hacer más lenta la simulación para permitir cálculos precisos,

En comparación, la implementación en hardware con FPGA requiere un paso de tiempo fijo pre-definido que funciona en tiempo real, mientras que las tasas de muestreo de información se pueden ajustar en diferentes puntos dentro del hardware para permitir la ejecución asíncrona. Cada uno de estos procesos se ejecutan de manera coherente con un paso de tiempo consistente y una longitud de palabra fija. La conversión entre estas dos formas de ejecución, es decir, la simulación del algoritmo en Simulink y la ejecución del mismo en elementos de hardware, debe hacerse de una manera sistemática en donde se tenga en cuenta estas diferencias de derivación[**?**].

5.2. Xilinx System Generator for DSP.

5.2.1. Descripción general.

El toolbox incluido en la suite de Xilinx ISE, llamado *System Generator for DSP* o en su contracción, *Sysgen*, está pensado para resolver el problema de la interfáz entre el mundo de la simulación

matemática de Simulink, y el mundo tangible del hardware en FPGA.

El desarrollo de algoritmos DSP implica transformar las muestras de entrada provenientes de la etapa de conversión A/D, utilizando numerosas operaciones matemáticas complejas sobre dichas señales como la convolución o la aplicación de alguna transformación como la ***Transformada Rápida de Fourier (FFT)*** y además, es necesario tener el equipo de laboratorio requerido para para poder visualizar los espectros de las señales a través de todo el diseño[**?**]. Todo este proceso se vuelve una labor muy intensa cuando se utilizan técnicas tradicionales de RTL en un FPGA.

System Generator es una herramienta de modelado a nivel sistema que facilita el diseño de hardware en FPGA[4]. Esta herramienta extiende la funcionalidad de Simulnk de muchas maneras, con el objetivo de proporcionar un entorno de modelado bastante adecuado para el desarrollo de hardware. Este toolbox proporciona abstracciones de alto nivel que se compilan automáticamente en un FPGA, facilitando la implementación de algoritmos complejos en donde intervengan, por ejemplo, hasta varios bloques de hardware complejos en diferentes dominios de reloj. También proporciona acceso a recursos o macros primitivas subyacentes del FPGA a través de abstracciones de bajo nivel, lo que permite la construcción de diseños altamente eficientes[**?**].

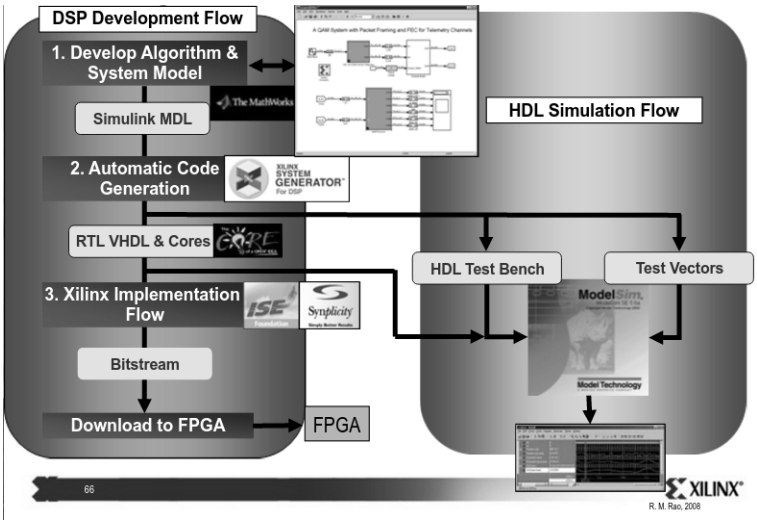

Figura 5.3: Flujo de diseño con System Generator de Xilinx, imagen obtenida del documento " *Wireless communications: from systems to silicon*", del autor Raghu Rao. Wireless Systems Group, Xilinx Inc, 2008.

[4]Se recomienda leer la documentación de Xilinx: *"System Generator for DSP, User Guide (UG640)"* para tener una mayor referencia a todos los conceptos técnicos que intervienen en esta metodología de diseño y que no son tocados por este documento de investigación.

*Syst*em Generator añade librerías o *blocksets* a la interfáz de Simulink, estos se pueden conectar a los demás toolboxes que vienen ya incluidos en la suite de Matlab para generar modelos funcionales de sistemas dinámicos. Estos bloques proveen abstracciones de funciones matemáticas, lógicas, utilización de memorias y funciones DSP que pueden ser facilmente utilizados en el diseño de sistemas complejos y sofisticados.

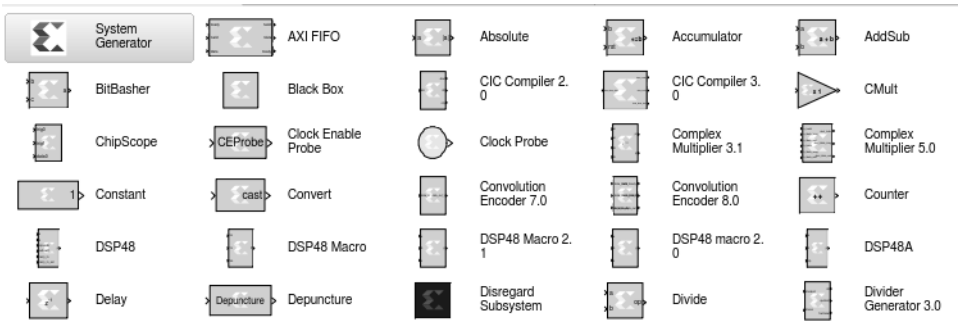

Figura 5.4: Ejemplo de blocksets incluidos en el toolbox de Sysgen, en su versión ISE 14.7.

Una de las características más importantes de este toolbox es que, permite implementar algoritmos haciendo uso de varios ingredientes. Por ejemplo, se pueden integrar bloques de código en *Verilog* o *VHDL,* así como funciones pre diseñadas de matlab (scripts .m) dentro de las conexiones de Simulink, sin afectar la ejecución de simulación y más importante, garantizando la síntesis de todos estos elementos en el FPGA. Además, este *blockset* trabaja con ***precisión de bits*** (*bit-accurate*) y ***precisión de ciclo*** (*cycle-accurate).* La ***precisión de bits*** produce valores en Simulink que coinciden con los valores producidos en el hardware; la ***presición de ciclo*** produce los valores correspondientes en el tiempo correspondiente.

5.2.2. Co-Diseño de Hardware/Software en el ambiente de Xilinx System Generator for DSP.

Es posible que el flujo de diseño con esta herramienta de alto nivel no sea tan claro a primera instancia, más para aquellos que tienen experiencia trabajando con los paquetes de software de Matlab y Xilinx por separado. Existen varias formas en las que *Sysgen* puede se sintetizado en el kit Atlys, algunos flujos representan ciertas ventajas sobre los demás, o simplemente son mejores formas

de lograr el objetivo deseado de una manera más rápida. Los flujos de implementación soportados por este toolbox son:

- **Exploración de Algoritmos mediante simulación *Hardware-in-the-Loop:*** Este flujo está pensado para llevar a cabo una simulación en tiempo real de un prototipado o incluso de un análisis realista del desempeño de un sistema bajo estudio. El bloque de Simulink es sintetizado con una capa extra de hardware que le brinda la capacidad al FPGA de enviar y recibir estímulos directamente al ambiente de simulación, mostrando así una respuesta tangible y cercana a la realidad que el hardware podría brindar. En este modo, la información se puede enviar a través del puerto JTAG o ethernet, dependiendo de la velocidad de tranmisión requerida.

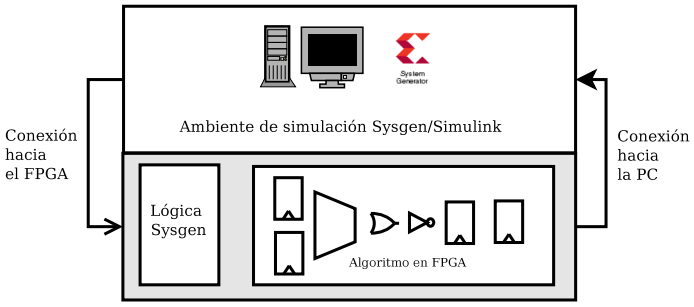

Figura 5.5: Diagrama a bloques del flujo de implementación mediante *Hardware-in-the-loop.*

- **Diseño de una IP como parte de un prototipo más grande:** Se puede empaquetar el sistema previamente modelado y simulado en Sysgen/Simulink para después ser instanciado como parte de un diseño más grande. *Sysgen* tiene bastante limitaciones a la hora de trabajar con interfaces que necesitan estrictos requerimientos de *timing*. Por ejemplo, para trabajar con interfaces ADC/DAC que generan información a grandes frecuencias de reloj, es mejor utilizar este flujo y combinar el modelo de Sysgen con las demás partes necesarias del prototipo, en un solo proyecto. Sysgen puede crear un empaquetado en Verilog o VHDL para sintetizar y/o simular un modelo de forma independiente.

- **Implementación de un diseño completo:** La suite de Xilinx contiene herramientas para distintos enfoques, por ejemplo, síntesis lógica con lenguajes de descripción de hardware, sistemas embebidos basados en la familia de procesadores *MicroBlaze/PicoBlaze*, catálogos de IPs para una gran variedad de sistemas ingenieriles y diseño de sistemas en lenguajes de alto nivel. Sysgen puede trabajar con todos y cada uno de esos enfoques, para lograr un diseño mucho más estable en un menor tiempo.

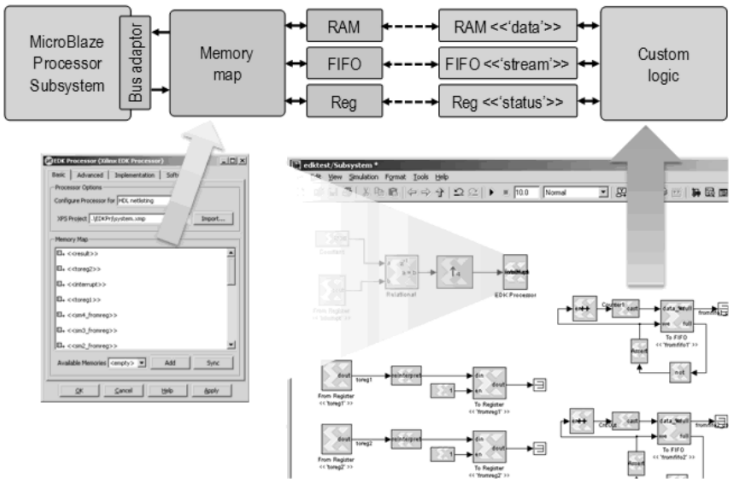

Figura 5.6: Ejemplo de implementación de un diseño completo utilizando el toolbox de Sysgen en Simulink. Imagen tomada de la guía de usuario "*System Generator for DSP User Guide, UG640".* Se puede observar como el procesador Microblaze, puede transmitir y recibir impulsos del modelo en Sysgen.

5.3. Introducción a Xilinx ISE y PlanAhead.

5.3.1. Reseña de Xilinx ISE.

El sistema de Xilinx ISE es un entorno de diseño integrado que consiste en un conjunto de programas para crear (o capturar), simular e implementar diseños digitales en un dispositivo FPGA o CPLD de Xilinx únicamente. Todas las herramientas contenidas en este entorno utilizan una interfáz gráfica de usuario que permiten que todos los programas se ejecuten desde la barra de herramientas[**?**].

Este entorno gráfico mostrado en la Figura 5.7 es conocido como *Project Navigator,* desde esta interfáz se controlan todos los aspectos del diseño, desde la creación de un proyecto y la selección de la familia de FPGA que se va a utilizar, hasta la edición de archivos de descripción de hardware, diseño a base de captura de esquemático y verificación del diseño, sin salir de la interfáz.

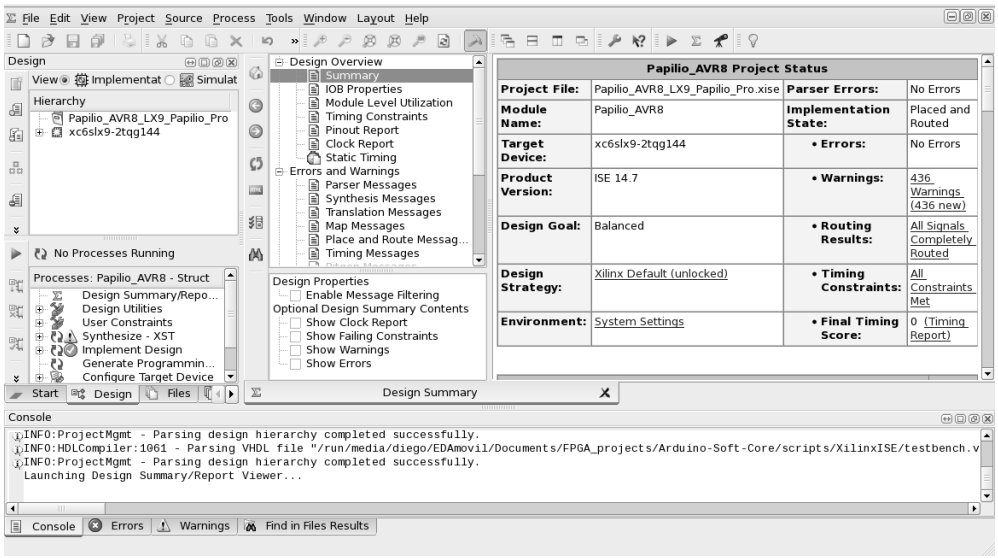

Figura 5.7: Interfáz principal del *ISE Project Navigator*.

Xilinx ISE guía al diseñador en el flujo de construcción en FPGA utilizando el flujo conocido como "*push button*", esto significa que todo el proceso está definido, automatizado y sólo basta presionar un botón para que la herramienta haga el trabajo de llevar la especificación del diseño de RTL al *bitstream*, mediante la ejecución de algunos procesos en segundo plano, los cuales se enlistan en la **Figura** 5.8 y se describen a continuación.

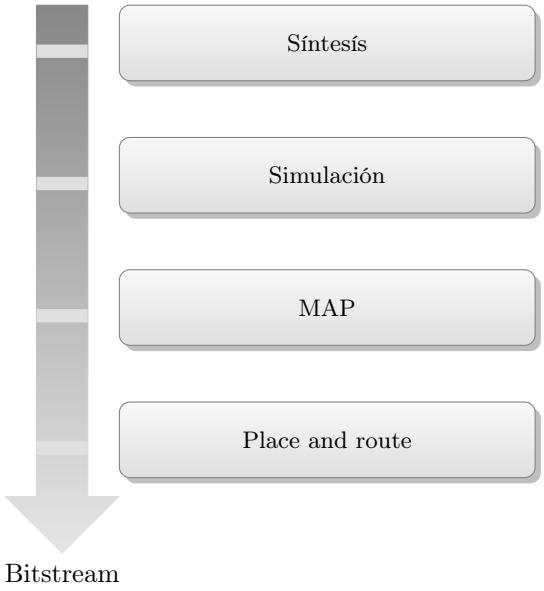

Figura 5.8: Diagrama a flujo del proceso de compilación en FPGA, elaborado usando el paquete Ti*k*Z de LaTeX.

- **Síntesis:** Este es el proceso de convertir un diseño descrito en Lenguaje de Descripción de Hardware a un netlist con un formato propietario conocido como *Archivo Netlist con Información de restriciones físicas* (**NGC**, por sus siglas en Inglés), ejecutando la herramienta llamada *Tecnología de Síntests de Xilinx* (**XST,** por sus siglas en Inglés).

- **Simulación:** XIlinx ISE cuenta con un simulador integrado llamado **ISE Simulator** o **ISim**, el cual soporta Verilog 2001 y VHDL 93, además del uso de ambos lenguajes para diseño y simulación. También ofrece capacidades de visualización y rastreo/análisis de formas de onda tanto analógicas[5] como digitales y depuración de fuentes HDL, entre muchas otras características[6].

- **Map:** En este paso se translada la salida generada en el proceso de síntesis, a primitivos de la FPGA especificada en el proyecto. Al finalizar, el ejecutable llamado igual que el proceso (**Map**) genera un netlist más, pero en un formato conocido como *Descripción Nativa del Circuito* (**NCD** por sus siglas en Inglés). Es importante recalcar que en este proceso es donde se llevan a cabo las optimizaciónes de *timing closure*[7] *y* área, las cuales están relacionadas

[5]La visualización de formas de onda analógicas requiere de una licencia adicional.

[6]Para obtener una lista más completa de las características de **ISim**, puede visitar el sitio web https://www.xilinx.com/products/design-tools/isim.html

[7]*Timing closure* es el proceso mediante el cual se modifica un diseño FPGA para cumplir con los requisitos de tiempo. Por ejemplo, la frecuencia máxima a la que el prototipo puede trabajar, o la generación de frecuencias correctas para protocolos de baja velocidad como RS232 o SPI.

con el desempeño final del diseño[**?**].

- **Place and Route (PAR):** Este es el proceso de interconectar los elementos primitivos, a partir de la salida dada por el proceso de Map, para así poder finalizar la implementación. Este es el paso más importante y el que consume un mayor tiempo de ejecución, debido a que la herramienta de Xilinx encargada de hacer la *colocación* de los primitivos en los CLBs ejecutará diferentes algoritmos para obtener el resultado más óptimo, de otra forma el *enrutamiento* será casi imposible de lograr. En este paso, se obtiene el porcentaje exacto de recursos utilizados en la FPGA y la información para generar el *bitstream.*

Project Navigator se encarga de ejecutar el flujo de implementación descrito, así como generar reportes para el análisis de resultados y manejar la configuración del kit Atlys, para descargar el *bitstream* generado.

5.3.2. PlanAhead.

Hablando en términos más profesionales, el flujo de implementación en FPGA descrito en la subsección anterior se divide en dos procesos globales: *Front-End,* que básicamente abarca la síntesis y simulación, y *Back-end* que abarca Map *y* PAR.

Generalmente, *PlanAhead*[8] es utilizado por usuarios con más experiencia en el campo de diseño con herramientas de Xilinx, puesto que este ambiente ofrece mejoras para la optimización en el *Back-end* que en el *Project Navigator* no se exponen con la misma facilidad.

Sin embargo, *PlanAhead* provee plugins bastante útiles para el *Front-End*, por ejemplo, usa un potente motor para el análisis de los archivos HDL en el diseño, también puede trabajar con netlist previamente sintetizados y resultados de implementación, además de poder crear diferentes versiones de implementación, con lo que se pueden explorar multiples óptimizaciones para un mismo proyecto.

En este documento se utilizará *PlanAhead* sobre *Project Navigator* por las ventajas ya mencionadas, además de que, Xilinx descontinuó la suite ISE para evolucionar a Vivado desde el año 2013. Vivado es una herramienta más poderosa pero no compatible con familias inferiores a la serie 7.

[8]Se recomienda leer la guía **UG673: "Quick Front-to-Back Overview Tutorial"** para mayor detalle en el uso de PlanAhead.

PlanAhead es muy similar a la interfáz de Vivado, por lo que los diseños podrán migrarse facilmente sin necesidad de tener que familiarizarse de nuevo con otra herramienta.

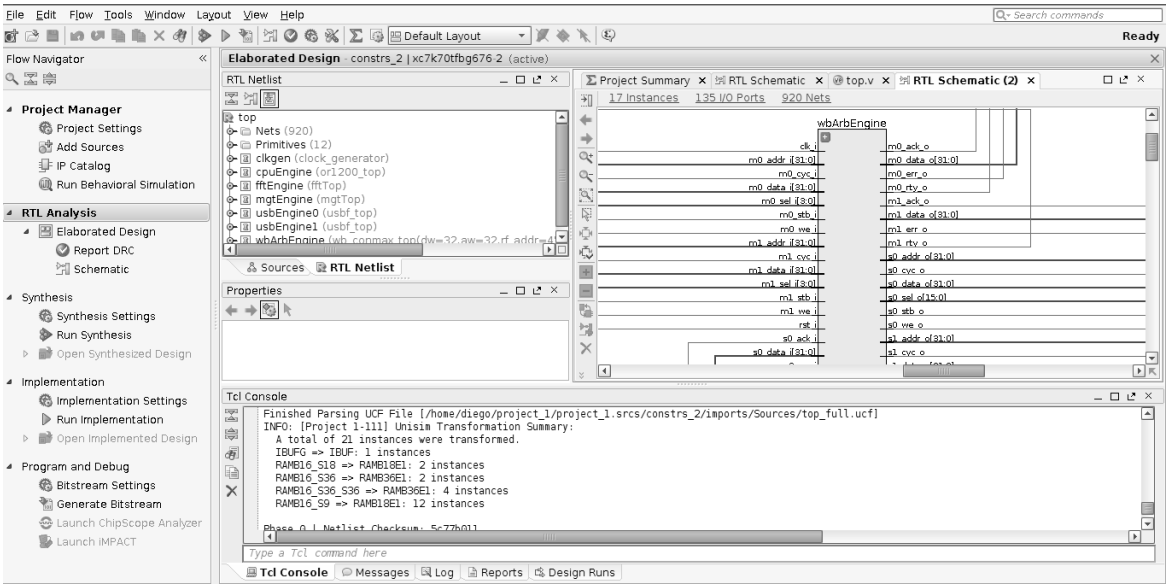

Figura 5.9: Interfáz gráfica de PlanAhead, imagen obtenida del paquete Xilinx ISE.