IMPLEMENTACIÓN DE UN SISTEMA DE AJUSTE DE MÍNIMOS CUADRADOS EN FPGA

SISTEMAS EMPOTRADOS – MÁSTER SSDE

ABEL ANAYA HERRERA
ALBERTO DEL ÁGUILA GÓMEZ
DIEGO NICOLÁS MACHUCA MEOÑO
VÍCTOR SAN TELESFORO GARCÍA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

ÍNDICE

INTRODUCCIÓN

En esta práctica se ha implementado el algoritmo de los mínimos cuadrados en una FPGA con el objetivo de demostrar las ventajas que tienen los dispositivos hardware reconfigurables a la hora de realizar operaciones lógicas.

La lógica reconfigurable es una rama de las arquitecturas de computadores que se encarga de construir sistemas flexibles, capaces de modificar su hardware mientras trabajan. La idea de tener elementos de procesamiento reconfigurables ha estado presente desde la década de los 60, pero ha sido a partir de finales de los 80, con la aparición de las FPGAs comerciales, cuando su uso ha sufrido un gran despegue.

La principal característica que diferencia a un dispositivo reconfigurable frente a un procesador de propósito general es que nos permite obtener una mayor velocidad de procesamiento en cuanto al número de operaciones por segundo que es capaz de realizar gracias a su alto nivel de paralelismo.

Los circuitos integrados para aplicaciones específicas (o ASIC, por sus siglas en inglés) también nos ofrecen una gran potencia, pero para diseños más pequeños donde no se requiere un gran número de unidades un computador reconfigurable puede llegar a tener un menor coste que un diseño equivalente basado en ASIC. Además, existen aplicaciones que requieren una mayor flexibilidad y un nivel de actualización mayor, que un ASIC no puede satisfacer. En cambio, los computadores reconfigurables, para poder realizar estas operaciones requieren de un mayor consumo de energía.

Las FPGAs son la versión más utilizada y comercial de los dispositivos reconfigurables dentro del ámbito industrial. Contienen grandes matrices bidimensionales regulares de lógica e interconexiones programables de grano fino, que pueden ser configuradas para implementar los circuitos digitales que requiera el usuario. Cada bloque de lógica es una LUT (*Look-Up Table,* por sus siglas en inglés), es decir, un elemento de memoria que almacena el resultado de una función lógica. Estas LUTs se estructuran en *Clusters* que presentan interconexiones programables, como pueden ser los CLBs de Xilinx.

RESUMEN TÉCNICO

Se ha propuesto realizar una implementación en VHDL de un algoritmo de aproximación por mínimos cuadrados con el objetivo de obtener una función matemática que describa el comportamiento de unos determinados datos de entrada.

Este objetivo por limitaciones de tiempo se ha reducido a la obtención de un polinomio de grado 4 que represente lo mejor posible los datos de entrada. Los datos de entrada se proporcionarán al algoritmo en un array fijo codificado en el código desarrollado.

En cuanto a las herramientas utilizadas, se ha elegido software de código abierto tanto para la compilación del código VHDL (GHDL Compiler) como para la visualización de simulaciones (GTKWave) para no depender de licencias de proveedores externos y evitar utilizar herramientas muy pesadas. Para la síntesis del bitcode se usará la herramienta Vivado de Xilinx, indicada por el fabricante de la placa de desarrollo Basys3 para su FPGA Artix-7.

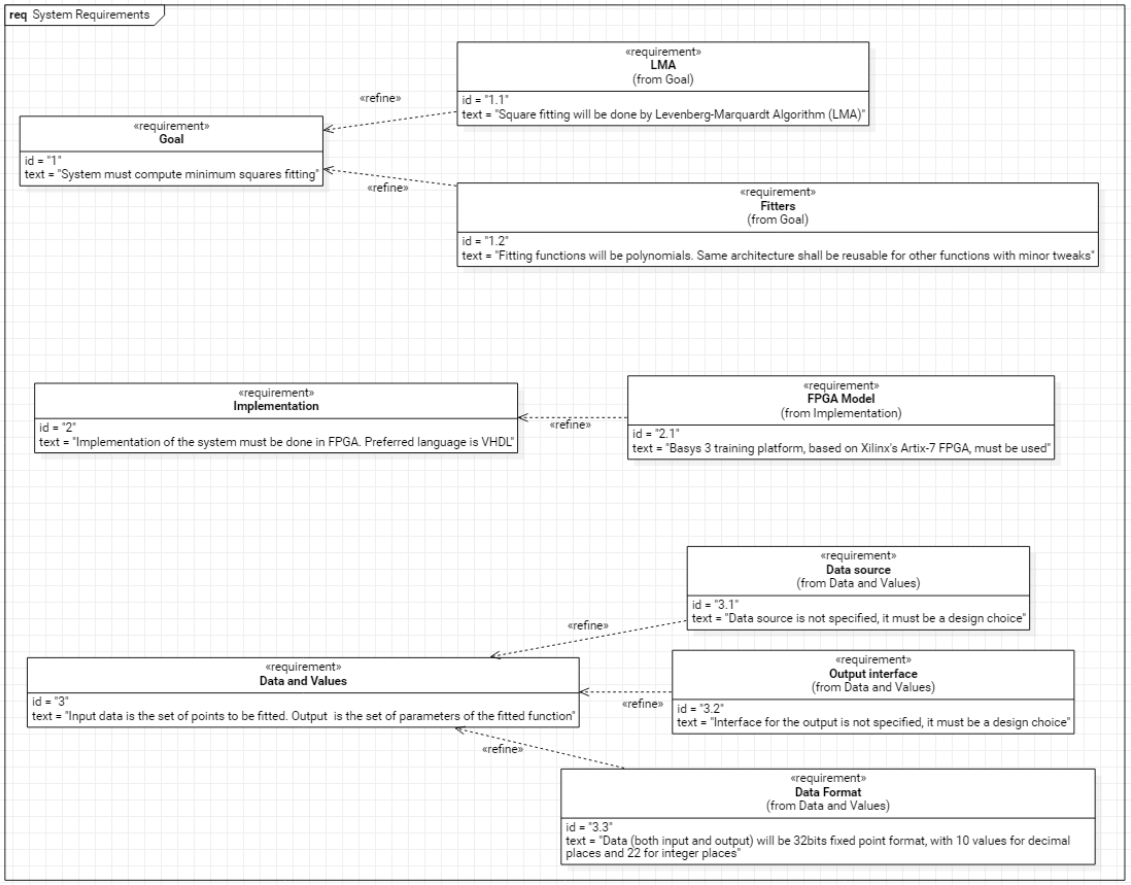
Sobre el algoritmo de aproximación de mínimos cuadrados, se ha decidido utilizar el algoritmo de Levenberg-Marquardt (LMA). Este algoritmo se basa en realizar una serie de cálculos sobre los valores de entrada de forma iterativa, de forma que el resultado de cada iteración mejora la estimación realizada sobre los datos. Este proceso se repite hasta la obtención de un polinomio que proporcione una aproximación aceptable respecto a los datos reales. Otras posibilidades de algoritmos de optimización son el Reflexive-Trust-Region y el Interior-Point. Se ha elegido LMA por su mayor simplicidad, ya que las técnicas de descenso de gradiente y aproximación de Gauss eran más conocidas para el equipo.

En cada iteración del algoritmo, la estimación resultante se compara con el dato real y en caso de ser menor a una cota de error seleccionada como entrada del algoritmo se para la ejecución, proporcionando los coeficientes del polinomio resultante tras la aproximación por mínimos cuadrados.

El algoritmo de Levenberg-Marquardt basa su funcionamiento en la combinación del algoritmo de Gauss-Newton y el descenso del gradiente. Aunque es un algoritmo robusto que consigue encontrar soluciones incluso encontrándose lejos de mínimos locales, al usar el descenso del gradiente tiene el problema de que puede encontrar mínimos locales en lugar de mínimos globales, por lo que se recomienda realizar varias ejecuciones con distintos puntos iniciales para maximizar su rendimiento. Este algoritmo es una mejora respecto al algoritmo de Gauss-Newton que se trata de una versión modificada del algoritmo de optimización de Newton para sistemas no lineales.

DESARROLLO DE LA SOLUCIÓN

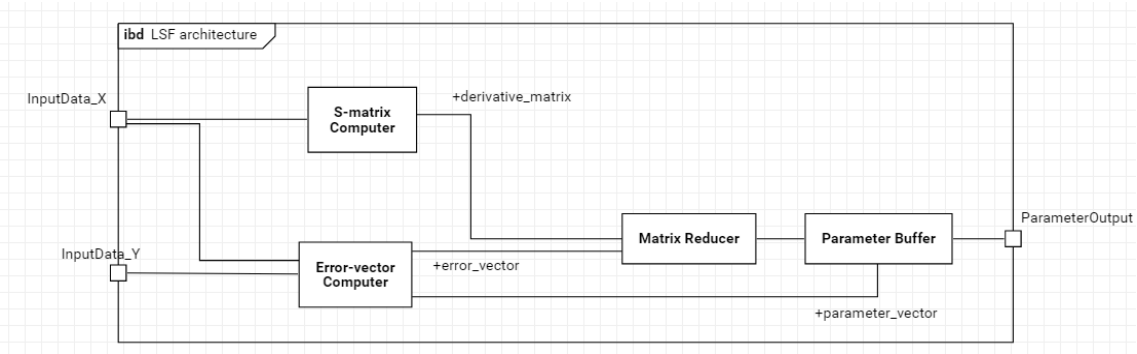
Inicialmente, se han recogido los siguientes requisitos para el diseño de la solución:



Como primera aproximación al problema, se ha decidido implementar el ajuste de un polinomio de grado 4 $P(x) = A + Bx + Cx^2 + Dx^3 + Ex^4$. Esto se debe a que las derivadas parciales respecto a cada parámetro son funciones lineales respecto a éstos, por lo que la implementación del producto $J^T J$ es más sencilla. Sin embargo, la aproximación de polinomios conlleva posibles desbordamientos si no se tienen en cuenta correctamente los tamaños de las señales y registros.

También, para simplificar el problema, el sistema hace los cálculos con valores de tipo entero, que se mapean mediante un escalado de LSB (en este caso 2^-10) a los valores originales.

Una vez establecidas las restricciones de diseño, se ha planteado la siguiente arquitectura mediante separación de funcionalidades y pipelining.



Para controlar la ejecución correcta del hardware, cada bloque genera señales de control que notifican a los bloques siguientes la existencia de nuevos datos y por tanto inician su ejecución.

Los bloques "S-matrix computer" y "Matrix reducer" se desarrollan dentro de la misma entidad. Ésta recibe la variable independiente y calcula el producto $J^T J$, al que más tarde aplica el método de reducción de Gauss-Jordan (GJ) para obtener el incremento de los parámetros en la iteración actual. Si se desarrolla este producto, se observa que puede implementarse como una acumulación de la operación realizada sobre cada elemento del conjunto de entrada. En este caso, el jacobiano del polinomio, y el producto de éste por su transpuesto, es el siguiente:

$$J = \begin{pmatrix} \frac{dx_1}{dA} & \frac{dx_1}{dB} & \frac{dx_1}{dC} & \frac{dx_1}{dD} & \frac{dx_1}{dE} \\ \frac{dx_2}{dA} & \frac{dx_2}{dB} & \frac{dx_2}{dC} & \frac{dx_2}{dD} & \frac{dx_2}{dE} \\ \frac{dx_3}{dA} & \frac{dx_3}{dB} & \frac{dx_3}{dC} & \frac{dx_3}{dD} & \frac{dx_3}{dE} \end{pmatrix} = \begin{pmatrix} x_1^4 & x_1^3 & x_1^2 & x_1 & 1 \\ x_2^4 & x_2^3 & x_2^2 & x_2 & 1 \\ x_3^4 & x_3^3 & x_3^2 & x_3 & 1 \end{pmatrix}$$

$$J^T J = \begin{pmatrix} \Sigma x^8 & \Sigma x^7 & \Sigma x^6 & \Sigma x^5 & \Sigma x^4 \\ \Sigma x^7 & \Sigma x^6 & \Sigma x^5 & \Sigma x^4 & \Sigma x^3 \\ \Sigma x^6 & \Sigma x^5 & \Sigma x^4 & \Sigma x^3 & \Sigma x^2 \\ \Sigma x^5 & \Sigma x^4 & \Sigma x^3 & \Sigma x^2 & \Sigma x \\ \Sigma x^4 & \Sigma x^3 & \Sigma x^2 & \Sigma x & \Sigma 1 \end{pmatrix}$$

Extendido a un número arbitrario de puntos x_n, resulta en el sumatorio de cada punto elevado a una potencia determinada. Esto facilita el diseño del bloque ya que puede recibir información de forma secuencial. Como se comentaba anteriormente, la utilización de polinomios implica que el rango de valores que almacena la matriz es muy grande, por lo que requerirá de tamaños de palabra suficientes para contener dichos resultados. También se pueden aplicar limitaciones de rango de entrada para asegurar el funcionamiento correcto.

Internamente, el bloque debe controlar las tres fases de su ejecución: captura de datos, cálculo de la matriz de acumulación y reducción GJ.

La reducción GJ se realiza sobre la matriz extendida que forman $(J^T J \,|J^T (y - P(x))$, y debe implementarse de forma secuencial (el algoritmo completo debe ejecutarse antes de hacer la asignación de salida), por lo que se valorará el uso de variables en lugar de señales ya que estas se actualizan de forma inmediata. Una vez hecha la reducción se devuelven los valores resultantes de la 6a columna, que serán los valores de actualización de los parámetros del polinomio.

Un aspecto que este bloque no implementa, y que es importante dentro del concepto del algoritmo LMA, es la modificación del parámetro lambda o parámetro de aprendizaje. Por decisión de desarrollo el parámetro se ha fijado en 1 para facilitar la implementación de las operaciones en una primera versión. Una vez demostrado el funcionamiento con el parámetro fijo, se implementaría un bloque de control de aprendizaje con métodos de inercia, de forma que conforme avanzan las iteraciones de aproximación a los datos se reduce el impacto de las posibles oscilaciones e inestabilidades.

El bloque "Error vector computer" debe hacer el cálculo del vector $J(y - P(x))$, por lo que necesita las variables dependiente e independiente y los parámetros. El bloque calcula el valor estimado para la variable independiente a partir de la dependiente, así como el error y el jacobiano para dicho punto. Estos valores se acumulan de forma similar a lo mostrado para el bloque anterior, de forma que toma valores secuencialmente y construye la salida esperada.

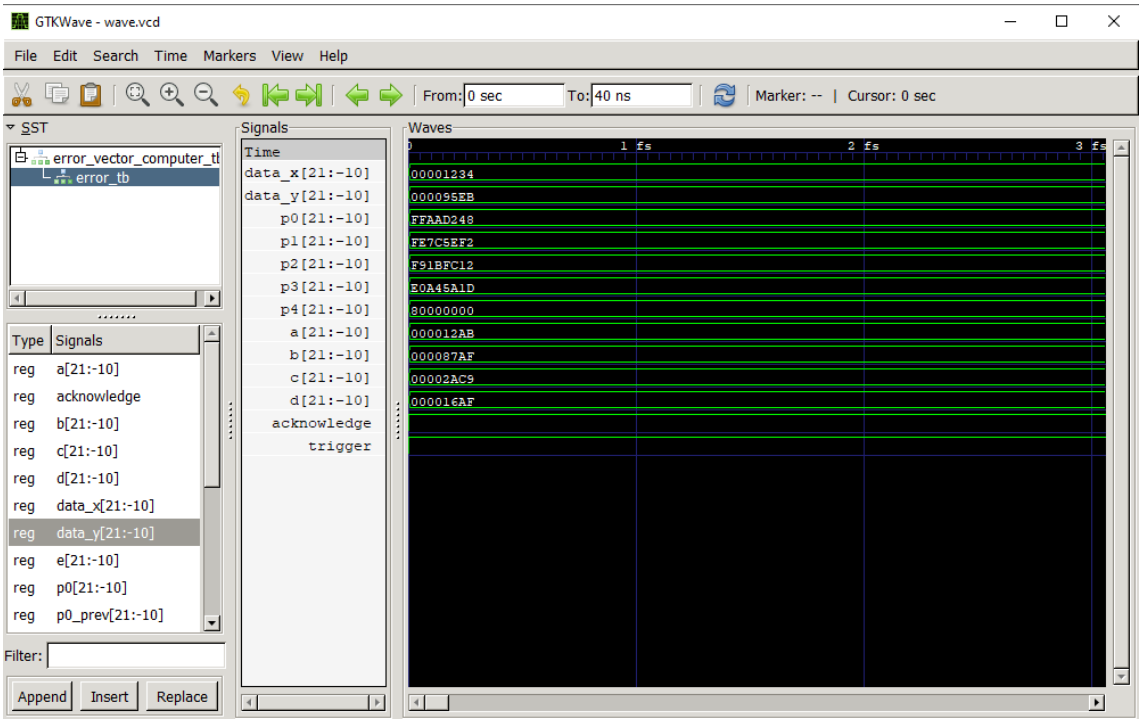
Por último, el bloque "Parameter buffer" actúa como registro de salida y acumulador de los parámetros. También implementa la lógica de decisión sobre si la aproximación es lo suficientemente buena, calculando una métrica sobre el vector de actualizaciones. Dicha métrica equivale al módulo del vector, y permite establecer que si el módulo es menor que cierto umbral, la actualización no es significativa y no debe continuar iterándose.

DEMOSTRACIÓN DEL FUNCIONAMIENTO

Con el objetivo de demostrar la capacidad del sistema para obtener un polinomio de grado 4 a partir de una serie de datos de entrada se ha decidido utilizar simulaciones VHDL. Estas simulaciones estimulan el código VHDL desarrollado para el sistema y muestran la evolución del sistema junto con sus valores a lo largo del tiempo.

Utilizando este método al final de la simulación pueden observarse cuáles son los coeficientes del polinomio resultante al aplicar este algoritmo.

Se han realizado simulaciones para cada uno de los bloques lógicos que componen al sistema para poder analizar su comportamiento y facilitar la depuración del código. A continuación se muestra la simulación del componente "Error vector computer", encargado de calcular el error producido en cada aproximación para ser suministrado al "Matrix reducer".



Simulación del componente "Error vector computer" durante una iteración

Dado que no se ha podido probar el código desarrollado en hardware real, no existen evidencias ni resultados en este ámbito. No obstante, podrían verse los resultados representando los coeficientes en una pantalla LED o transmitiendo la información por puerto serie.

Para verificar que los resultados son coherentes, se ha desarrollado un script en Python que implementa el algoritmo. De esta forma contamos con una referencia que permitiría comprobar la validez del sistema.

A modo de primera aproximación, este mismo código se ha utilizado como referencia para implementar el procedimiento de construcción de la matriz del sistema, así como el de la reducción GJ. Esta decisión se ha tomado por criterios de eficiencia en cuanto al tiempo de desarrollo, pero plantea la seria cuestión de la eficiencia de la solución al basarse en los principios del desarrollo software y no en los del desarrollo de descripción hardware.

CONCLUSIONES

Desafortunadamente, no hemos sido capaces de probar en el dispositivo que el sistema planteado es válido. En este punto, nos basamos en los resultados producidos por el script que han servido de base para el desarrollo de la solución en VHDL para estimar que la arquitectura propuesta es válida, aunque los detalles de implementación pueden jugar un papel crítico en el éxito o fracaso de la misma en la fase de síntesis. Entre otros detalles pueden contarse:

- Incompatibilidades de síntesis de algunos paquetes: aunque este detalle se ha tenido en cuenta desde el principio, cabe la posibilidad de que algún paquete que ofrezca funcionalidades críticas del sistema (por ejemplo si se hubieran utilizado paquetes de aritmética de coma flotante) pueda no ser sintetizable, por lo que habría que rehacer gran parte de la implementación.
- Desbordamientos e inestabilidad numérica: al ser un problema poco restringido (sin límites explícitos en la cantidad de datos o en el rango de los mismos, por ejemplo) es fácil caer en fallos de implementación que, en casos extremos, pudieran dar lugar a soluciones inestables o a desbordamientos. Ante esto sólo cabe redefinir las restricciones de diseño del sistema, en cuyo caso la falta de experiencia del equipo en este ámbito es un factor en contra, pues puede darse el caso de que las nuevas condiciones impuestas hagan que el sistema no se adecúe a los escenarios esperados.

Otro componente a tener en cuenta es la decisión de la arquitectura. Debido a la poca experiencia, la arquitectura utilizada se ha extraído de un dominio más conocido para el equipo como es el desarrollo software: la complejidad del problema, sumada a la relativa novedad que supone trabajar con lógica reconfigurable y la limitación en cuanto al tiempo que se ha podido dedicar a este proyecto, ha obligado a optar por soluciones más directas que permitan enfocar la atención en la implementación antes que en el diseño de la arquitectura. Sin embargo, explorar otras arquitecturas como el uso de coprocesadores o unidades de coma flotante podría haber sido beneficioso.

Sin embargo, y a pesar de no haber conseguido el objetivo, hemos podido validar algunos componentes de forma aislada satisfactoriamente, lo cual es un indicio de la validez del sistema de forma global. También pone de manifiesto que el proceso de análisis del problema, la planificación del diseño y las estrategias usadas para definir la solución son válidas. En conjunto, el resultado es un punto de partida que, aunque admite mejoras, verifica la viabilidad de la propuesta.

BIBLIOGRAFÍA

- *Department of Civil and Environmental Engineering, Duke University, & Gavin, H. P. (2020, September). The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems. https://people.duke.edu/~hpgavin/ce281/lm.pdf*
- *Roth, C. H., & John, L. K. (2008). Digital Systems Design Using VHDL (2nd ed.). Thomson Learning.*