Universidad Complutense de Madrid
FACULTAD DE INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

SISTEMAS INFORMÁTICOS

ALGORITMOS DE TRIGGERING PARA DETECCIÓN DE EVENTOS Y SU APLICACIÓN PARA DETECCIÓN DE DUST DEVILS SOBRE FPGAS

Curso 2009/2010

Enrique de Lucas Casamayor

Manuel Javier Miguel García

Dirección:

Daniel Mozos Muñoz

Luis Vázquez Martínez

Resumen

Este trabajo trata sobre el estudio de algoritmos disparadores. Dichos algoritmos tienen múltiples aplicaciones en lo referente a predicción de eventos geológicos y meteorológicos como son terremotos, tornados y erupciones volcánicas entre otros.

Concretamente estudiaremos como la aplicación de dichos algoritmos para detección y predicción de dust devils en Marte. Estos son remolinos de aire que se forman debido a un grave contraste entre la temperatura de la superficie con la de la atmósfera.

Apoyándonos en los datos de la misión espacial Mars Pathfinder, estudiaremos estos curiosos fenómenos que recorren la superficie marciana con gran frecuencia.

Nuestro objetivo final es descubrir si la implementación de los algoritmos disparadores, aplicados a la detección de dust devils, es más eficiente ejecutando un software en un procesador corriente, o por otra parte ejecutándolo en un sistema hardware especifico generado con hardware reconfigurable.

Finalmente mediante comparativas de tiempo, estudiaremos que método se adapta mejor a los algoritmos disparadores.

Abstract

This work is about the study of triggering algorithms. These algorithms have multiple applications in the field of prediction of geological events, as earthquakes, tornados or volcanic eruptions.

Concretely we will study how to apply them in detection and prediction of martian dust devils, those whirlwind are formed for a high contrast between superficial and atmospheric temperature.

We will take support on Mars Pathfinder mission data, and we will study these phenomens which walk the martian ground with a high frequency.

Our final goal is to find out if the implementation of triggering algorithms applied to dust devils detection is more efficient using a common computer or in the other hand using an specific system generated with reconfigurable hardware.

Finally by using times matches we will find out which method is better for triggering algorithms.

Palabras clave: Algoritmos de Triggering, Marte, FPGA, Varianza de presión y temperatura, STA/LTA, Mars Pathfinder, Dust devils, VHDL sintetizable, Terremotos

Índice de contenidos

Autorización

Autorizamos a la Universidad Complutense de Madrid a difundir y a utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o los prototipos desarrollados.

Firmado: ....

Enrique de Lucas Casamayor Manuel Javier Miguel García

Madrid, 2 de Julio de 2010

1. Introducción

Con este trabajo queremos exponer los algoritmos disparadores describiendo su utilidad y aplicaciones. Aunque el objetivo final será centrarnos en las posibilidades que nos ofrecen dichos algoritmos en la exploración espacial, concretamente en los procesos de investigación en el planeta rojo del Sistema Solar, es decir Marte.

Antes de proseguir deberíamos aclarar que los algoritmos disparadores, traducción literal de Triggering Algorithms, también son conocidos como algoritmos detectores de eventos, o simplemente algoritmos detectores, e incluso en algunos casos son también conocidos como detectores de fase, pero estos últimos suelen estar ligados al estudio de la actividad sísmica, por lo tanto no esta tan generalizada esta denominación.

Cabe mencionar, recordando que antes los nombramos como algoritmos detectores, o reconocedores de eventos, que aún no hemos definido de forma concreta lo que es un evento. Dada la flexibilidad de estos algoritmos y sus diversos usos, para cada caso concreto un evento es algo distinto, pero que no obstante es algo que se puede medir. En algunos casos el evento será un seismo, en otros la erupción de un volcán, pero también puede ser algo tan sencillo como la bajada de temperatura en la superficie de Marte.

Obviamente el concepto de evento es decisivo en la descripción de estos algoritmos pero cambia radicalmente según el contexto en el que nos encontremos.

Originalmente los algoritmos detectores se han aplicado desde décadas en la Tierra para realizar estudios sobre distintas actividades geológicas de interés. Un caso de estudio muy importante de estos consiste en la detección y predicción de terremotos y la actividad sismológica. Y es aquí precisamente donde los algoritmos que estamos tratando cobran gran importancia, pues ofrecen múltiples aplicaciones y usos.

Son diversas las aplicaciones de estos algoritmos, más allá de la detección de terremotos, otro caso importante de estudio es la predicción de la actividad volcánica. Otro ejemplo puede ser el análisis de mareas. Podemos afirmar que están presentes en casi todas las areas calientes de estudio de la Geología. En definitiva como decíamos antes cada posible aplicación de los algoritmos detectores está ligada a los eventos relevantes en cada caso.

Pero hoy en día sus aplicaciones han evolucionado, pues aunque aún se utilizan en la Tierra, han dado el gran salto al espacio. Las características únicas de estos algoritmos los hacen muy prácticos para su uso en la carrera espacial.

Así pues, uno de los objetivos de este trabajo será estudiar algunas aplicaciones directas de los algoritmos detectores en las misiones de exploración de Marte. En este momento viene a colación introducir uno de los casos de estudio más interesantes y específicos de los que se dan en Marte, nos referimos al estudio de los Dust Devils.

Estos Dust Devils consisten en pequeños remolinos de viento, aunque no serían comparables a un huracán, pues a diferencia de estos últimos tienen una escasa anchura pero sin embargo son considerablemente altos.

Los Dust Devils ocurren con bastante frecuencia en la Tierra, no obstante nos interesan más los casos marcianos. Se sabe que uno de estos remolinos pasó muy cerca del Mars Pathfinder, el primer vehículo de exploración terrestre que recorrió la superficie marciana.

Debido a las condiciones atmosféricas de Marte, los Dust Devils pueden alcanzar más tamaño y energía que los Terráqueos pudiendo incluso ser una amenaza para los equipos tecnológicos enviados a Marte. Ahora podemos apreciar el peso que tienen los algoritmos disparadores en la detección de eventos en el planeta rojo. El objetivo final de este trabajo ver como se puede construir un detector de estos fenómenos, utilizando los algoritmos disparadores.

Una vez hemos introducido al lector los principios de los algoritmos de triggering así como algunas de las aplicaciones más importantes de los mismos, debemos hablar de la forma en que serán llevados a término dichos algoritmos en este trabajo.

Nuestra idea ha sido trabajar paralelamente entre simulación software de los algoritmos disparadores por un lado, y por la otra la implementación de los mismos con hardware reconfigurable, en concreto veremos como podemos implementarlos en una FPGA.

El hardware reconfigurable nos ofrece grandes posibilidades hoy en día en la Tierra, y aún tanto más pueden darse en el espacio. A continuación estudiaremos por qué es interesante usar hardware reconfigurable en estas misiones.

Por regla general cuando se trata de misiones espaciales el hardware suele encontrarse repetido por completo tres veces. Esta redundancia es necesaria debido a que por muy fuerte que sea el aislamiento siempre es posible que algún pulso de radiación afecte a parte del hardware, dejándolo inutilizable.

Estos sistemas de hardware triplicado poosen un módulo de arbitraje, el cual observa los resultados de cada una de las partes. En caso de que una de las partes esté dando valores diferentes a las otras dos, se procede a aislarlo, pues se está produciendo un mal funcionamiento. Esto provoca que el hardware se pierda y no pueda ser aprovechado de nuevo, pues ha sido rechazado.

Es en este punto donde el hardware reconfigurable puede producir un gran cambio. Pues en este caso, una vez detectado el error de una parte del hardware, procederíamos a reconfigurarlo de nuevo y dejarlo funcionando como al principio, de forma que no hay pérdida de hardware. De hecho si una parte de la FPGA es dañada, aún podría utilizarse evitando situar hardware en dicha zona. Además de esto, si utilizamos hardware reconfigurable, no sería necesario que todo el hardware estuviera cargado en todo momento. Así por ejemplo, el hardware necesario para el calibrado de aterrizaje no sería necesario hasta el momento propio del aterrizaje, de forma que durante la travesía espacial de una sonda, no estaría cargado. De esta forma, seríamos capaces de ahorrar espacio y peso en las misiones espaciales, lo cual significa menor energía para lanzar la sonda y mayores posibilidades de éxito.

Hemos visto que el hardware reconfigurable nos ofrece muchas ventajas, pero debemos plantearnos si es posible que tenga alguna desventaja importante, la cuestión a estudiar es si un sistema sintetizado con hardware reconfigurable para un propósito especifico es más o menos eficiente que un sistema conformado por un hardware tradicional, es decir un procesador común, ejecutando un software con el mismo propósito.

Una parte importante de este trabajo trata de dilucidar esta cuestión en un caso concreto, nos referimos claro está a los algoritmos disparadores. Por un lado compararemos los resultados obtenidos por un simulador software y por otro analizaremos los mismos con hardware reconfigurable. El resultado de estas pruebas ha sido sorprendente como el propio lector podrá observar al final de este trabajo.

Llegados a este punto hemos expresado las motivaciones de este trabajo y la forma en la que va a ser realizado, de forma que podemos comenzar el estudio.

2. Marte

El planeta rojo ya era conocido desde la época clásica. Los romanos llamaban con el nombre de Marte al Dios de la guerra pues su color rojo se le relacionaba con la sangre. Ya desde entonces el hombre siempre se ha fijado en Marte y ha soñado con pisar su superficie.

2.1. Un entorno hóstil

Los avances científicos nos han demostrado que Marte y la Tierra tienen muchas cosas en común, por ejemplo, el día marciano dura 24 horas y 40 minutos aproximadamente. Estos días son conocidos como “soles” (término inglés a pesar de que sol en castellano se refiere al gran astro del sistema solar) y su similitud con la duración del día terrestre hace factible que nos planteemos su colonización. A pesar de esto, no debemos confundirnos y hemos de ser conscientes de que Marte es un lugar hostil y poco hospitalario. El objetivo de esta introducción es resaltar y mostrar al lector que Marte es un entorno difícil para el estudio e investigación científica.

La distancia entre Marte y la Tierra oscila entre 56 y 399 millones de kilómetros. Este dato es muy importante para las misiones espaciales. Sin mencionar que el mejor momento para enviar una misión es cuando los dos planetas están más cerca, esto ocurre aproximadamente cada dos años terrestres, hay que tener en cuenta que se tarda un tiempo en enviar y recibir la información a Marte.

Por ejemplo el rover de la misión Mars Pathfinder era operado por ingenieros en la tierra de forma teledirigida. Dada la distancia que separa a ambos planetas, el tiempo que se tarda desde que se envía la orden hasta que el vehículo la recibe, en el peor de los casos llega a ser de 22 minutos, teniendo en cuenta que las ondas electromagnéticas se propagan a la velocidad de la luz, es decir $3 \cdot 10^8$ m/s. Por lo tanto se debe operar con mucho cuidado y precaución.

Los avances tecnológicos han permitido introducir inteligencia artificial en estos vehículos para dotarlos de más autonomía durante el tiempo de latencia necesario para recibir las órdenes.

El año marciano dura 669 días, casi dos años terrestres, y al igual que en la tierra existen las estaciones. No obstante las variaciones de temperatura son mucho más extremas que las que ocurren en la tierra. Mientras que en verano y sobre el ecuador se pueden llegar a alcanzar unas temperaturas de 20ºC durante el invierno se alcanzan fácilmente temperaturas por debajo de los 80ºC bajo cero.

Con esto nos empezamos a hacer una idea de la hostilidad del entorno de Marte. Estos drásticos cambios de temperatura son debidos a que Marte se encuentra más lejos del Sol que la Tierra y además de que su atmósfera es mucho más tenue que la nuestra y no permite que se produzca sustancialmente el llamado efecto invernadero, solo aumenta unos 5ºC la temperatura, muy inferior al que se produce en la Tierra.

Su presión atmosférica varía entre los 7hPa y los 9hPa de media superficial. Siendo la presión terrestre, de unos 1033hPa, es decir más de 100 veces superior. Además la cantidad de ozono presente en su atmósfera es mil veces inferior a la de la Tierra, una concentración que no permite que su capa de ozono pueda bloquear las radiaciones ultravioletas procedentes del Sol. Además la tenue atmósfera no es capaz de bloquear la constante llegada de meteoritos al planeta rojo cuya superficie está llena de cráteres y agujeros, al igual que pasa con la Luna.

A pesar de la baja presión atmosférica las tormentas de polvo son muy comunes y frecuentes. Estas tormentas que pueden ser causadas por vientos de más de 150km/h, muy superiores a la media de las tormentas terrestres, en muchos casos llegan a tener una escala planetaria que puede durar semanas enteras. Esta capa de polvo puede llegar a imposibilitar la observación del planeta, como ocurrió con las sondas estadounideses Mariner en 1969. No hace falta mencionar que este es un grave inconveniente para los rovers que se pasean por la superficie marciana, puesto que la capa de polvo reduce significativamente la incidencia de la luz solar en la superficie, eso sin tener en cuenta que además el polvo levantado se acumula en los paneles solares reduciendo la captación de energía.

En el siguiente punto hablaremos sobre un tema interesante de estudio, como son los Dust devils, fenómenos tormentosos que se dan en la superficie marciana.

Con esta pequeña introducción sobre algunas de las características más señaladas de Marte, esperamos que se aprecie lo que queríamos decir en un principio y es que Marte es un entorno hostil y complicado.

2.2. Dust devils

Son un fenómeno atmosférico ya conocido desde hace mucho tiempo en la Tierra. Consisten en pequeños remolinos de aire más alargados que los tornados y de menor capacidad destructiva. Pueden tener desde medio metro de anchura y pocos metros de altura hasta unos 10 metros de anchura y casi un kilómetro de altura.

Las condiciones propicias para que estos fenómenos se den son terrenos sin vegetación, como desiertos y también el asfalto, con el cielo despejado o un poco nuboso, de forma que la superficie sea capaz de captar el calor del ambiente, pero con una temperatura atmosférica relativamente fría. Además el viento suele desestabilizar el sistema, evitando que pueda formarse. Estos se forman cuando el aire caliente de la superficie asciende rápidamente atravesando una bolsa de aire más frio y con baja presión sobre él. Si se dan las condiciones adecuadas, el aire empieza a rotar y como el aire asciende con rapidez la columna se va alargando poco a poco hacia el cielo, de manera que se intensifica el momento angular. A su vez un segundo flujo de aire también caliente, baja hacia el vórtice recién formado para sustituir el aire saliente, haciendo más intenso el giro del mismo.

Una vez está formado el dust devil es como una chimenea con forma de embudo, por la cual se mueve el aire caliente, por una parte hacia arriba y por otra girando en círculo. Cuando el aire caliente sube, termina por enfriarse y finalmente

es desplazado por el aire caliente en ascensión, manteniéndose entre el vórtice y la pared giratoria manteniendo el sistema estabilizado. Este además puede mantenerse en movimiento durante más tiempo si se desplazan sobre una superficie caliente. Como el dust devil va absorbiendo el aire caliente circundante, haciendo que este ascienda, al final termina por absorber también el aire frío, desestabilizando el sistema y haciendo que se termine por disipar.

Figura 1. Foto de un dust devil, tomada sobre la superficie de Marte

Fue en los años 70 cuando las sondas Viking detectaron por primera vez estos fenómenos en Marte. Dadas las características geológicas de Marte, los dust devils pueden llegar a alcanzar una anchura y altura incluso cien veces mayor que los detectados en la Tierra. Los dust devils típicos de Marte suelen tener una anchura de un kilómetro y una altura de diez. Lógicamente los más grandes pueden suponer una amenaza real tanto para los dispositivos tecnológicos enviados al planeta rojo como para posibles hábitats humanos.

No obstante también es posible que su paso sea beneficioso para los instrumentos. En 2005, el rover Spirit detectó que un dust devil había pasado sobre él, limpiando el polvo de los paneles solares y haciendo que el nivel de energía aumentase drásticamente.

Otro peligro de estos remolinos es la carga eléctrica que llevan. Al menos en la Tierra se ha podido medir que poseen una carga aproximada de 4000V por metro, siendo capaces de generar campos magnéticos, las investigaciones llevadas a cabo parecen sugerir que es posible que ocurra lo mismo con los dust devils marcianos, lo cual puede producir grandes problemas de comunicación, pues interferirían con el sistema de radio, y además puede provocar un aumento en la adesión de particulas de polvo a trajes y equipamento espaciales. Por supuesto al ser capaces de generar campos electromagnéticos los instrumentos

de medición se ven también afectados tomando medidas erróneas e incluso pueden llegar a dañarse.

Hasta el momento ninguno de los landers robóticos que han llegado a la superficie marciana ha detectado rastros de actividad eléctrica reseñable. Pero también es cierto que estos robots no estaban provistos de sensores eléctricos, por ello la tendencia actual de la investigación marciana pasa por estudiar las características eléctricas que acompañan a estos fenómenos atmosféricos únicos.

Gracias a los medios tecnológicos de que disponemos hoy en día y a que se ha observado mucho tiempo la superficie de Marte, podemos asegurar que los dust devils son un fenómeno muy común en la superficie y por tanto son un elemento a tener en cuenta en la investigación del planeta rojo, pues su acción puede ser negativa, o en algunos casos incluso positiva, para los recursos tecnológicos y posiblemente en un futuro humanos, que sean enviados a Marte. Por ello su estudio y la posibilidad de poder predecirlos es de vital importancia.

Como veremos más adelante, los algoritmos disparadores nos van a permitir crear detectores de dust devils muy eficientes, pues consumen poca energía y no necesitan equipos tecnológicos de última generación. Estos detectores permitirán a su vez capturar suficiente información de cada dust devil que se detecte.

Cuando se dispare un evento de detección la sonda empezará a medir con una mayor frecuencia recogiendo así mayor cantidad de datos de un momento que puede resultar de gran valor científico o ser simplemente un falso positivo, pero en todo caso serán mediciones motivadas por cambios en las magnitudes de medida. El éxito dependerá de la bondad del algoritmo detector.

Lo que se ha hecho hasta ahora (debido a la gran distancia entre la Tierra y marte) es programar con antelación las frecuencias de medida de las sondas por lo que la detección o no de eventos ha dependido, aparte de los conocimientos actuales sobre Marte, de la intuición del programador y la suerte, lo que hace que nos perdamos gran cantidad de eventos de interés científico.

Esto es lo que precisamente se trata de evitar con los algoritmos de disparo ya aumentamos con mucho las posibilidades de conseguir información sobre estos fenómenos.

3. Origen de datos: Misión Mars Pathfinder

En nuestras simulaciones hemos utilizados datos reales recogidos en Marte. En concreto para el estudio hemos utilizado la información procedente de la misión Mars Pathfinder. Esta misión fue la primera exitosa que incluyó un vehículo rover, el Sojourner, y alcanzó Marte en Julio de 1997 tras seis meses de viaje por el espacio.

Los datos obtenidos de esta misión están disponibles para todo aquel que requiera utilizarlos. Posteriormente indicaremos de donde pueden el lector obtener dichos datos.

Como estos datos eran muy importantes para nuestro trabajo optamos por descargarlos directamente, utilizando un spider, esto es, un programa encargado de descargar información de un sitio, o varios sitios webs de forma automática.

En su sección correspondiente el lector podrá ver cómo descargar estos datos. Por convenciencia para nuestro trabajo decidimos convertir dichos datos a archivos con formato xml. Eso se debe principalmente a que los datos que están a disposición del público tienen un formato de archivo de texto plano. Además otro inconveniente es que cada uno de los archivos puede contener información referente a uno o varios soles (días marcianos). Por lo tanto no nos resultaba práctico este formato.

Nuestro spider se encarga de obtener la información de estos archivos y guardarla adecuadamente con el formato XML en varios archivos, uno para cada sol.

Veamos como es el formato de los archivos de la misión Mars Pathfinder con un ejemplo práctico, tiene el siguiente aspecto:

```
     41      46     184.0       4.0      36.0       4.0       2      12.0     156.0   1246749014.000   142.85304
1  9  2  1.626   241.92   249.13   247.40   240.72    6.9305    6.9298   249.07   248.79   248.36   250.05   250.80   249.88  1246749014.000  0
1  9  2  5.520   238.28   247.07   245.72   236.59    6.9307    6.9305   249.07   248.57   247.27   249.40   250.36   249.67  1246749018.000  0
1  9  2  9.413   239.24   247.72   246.62   238.93    6.9317    6.9334   249.18   248.57   247.27   249.73   250.69   248.59  1246749022.000  0
1  9  2 13.307   240.12   248.79   247.98   239.14    6.9280    6.9283   282.64   279.42   279.11   278.56   283.13   282.61  1246749026.000  1
1  9  2 17.200   241.13   249.18   247.44   240.54    6.9287    6.9305   283.17   280.08   280.34   279.44   283.58   279.98  1246749030.000  1
1  9  2 21.094   242.37   247.37   244.93   241.44    6.9302    6.9305   283.35   279.64   279.37   279.09   283.58   283.09  1246749034.000  1
1  9  2 24.987   243.63   247.62   241.89   243.01    6.9295    6.9305   283.35   280.35   280.47   279.39   283.13   282.78  1246749038.000  1
1  9  2 28.880   238.16   248.67   244.39   237.02    6.9295    6.9305   282.86   276.73   275.42   277.63   284.25   283.70  1246749042.000  1
1  9  2 32.774   240.98   247.99   246.03   243.26    6.9282    6.9305   283.79   280.88   281.88   280.85   283.85   283.62  1246749046.000  1
1  9  2 36.667   241.74   247.53   245.87   240.19    6.9255    6.9254   283.70   280.48   280.65   279.88   282.90   282.34  1246749050.000  1
1  9  2 40.561   242.07   247.43   246.19   240.66    6.9270    6.9276   283.93   280.53   280.60   280.27   283.85   283.75  1246749054.000  1
1  9  2 44.454   243.44   245.48   240.64   245.28    6.9262    6.9276   283.79   281.41   282.80   279.97   281.38   282.30  1246749058.000  1
1  9  2 48.347   245.23   247.13   244.57   246.32    6.9267    6.9283   284.32   281.32   282.85   281.42   284.39   284.14  1246749062.000  1
1  9  2 52.241   245.70   249.05   242.05   246.92    6.9267    6.9283   284.19   281.28   282.67   281.02   284.12   284.05  1246749066.000  1
1  9  2 56.134   245.59   250.09   240.75   246.93    6.9257    6.9268   284.28   281.54   282.98   281.38   284.16   284.10  1246749070.000  1
1  9  3  0.028   246.76   250.34   245.88   247.82    6.9252    6.9268   284.50   281.81   283.77   281.86   284.34   284.27  1246749074.000  1
1  9  3  3.921   245.18   251.29   247.55   244.68    6.9240    6.9254   284.54   281.28   281.84   281.42   285.24   284.54  1246749078.000  1
1  9  3  7.815   245.83   251.44   247.15   245.12    6.9243    6.9254   284.54   281.98   283.55   281.99   285.24   284.58  1246749082.000  1
1  9  3 11.708   245.66   247.91   241.00   246.13    6.9240    6.9257   284.63   281.41   282.36   281.16   284.43   284.23  1246749086.000  1
1  9  3 15.601   242.66   249.63   244.69   241.79    6.9258    6.9271   284.63   281.10   281.40   281.24   285.51   284.80  1246749090.000  1
1  9  3 19.495   242.75   247.16   244.90   241.94    6.9243    6.9264   283.53   278.63   277.75   278.73   284.61   284.10  1246749094.000  1
1  9  3 23.388   239.72   247.36   244.42   238.95    6.9250    6.9257   284.46   281.01   280.08   280.10   284.83   284.19  1246749098.000  1
1  9  3 27.282   238.56   248.30   244.36   238.06    6.9255    6.9264   284.59   280.22   278.76   279.09   285.10   284.67  1246749102.000  1
1  9  3 31.175   237.86   248.74   245.87   238.41    6.9238    6.9264   284.37   280.00   278.80   279.48   285.55   284.84  1246749106.000  1
1  9  3 35.069   239.03   248.40   245.23   239.60    6.9248    6.9271   284.85   281.50   281.40   280.94   285.42   284.89  1246749110.000  1
1  9  3 38.962   239.15   248.40   246.08   239.14    6.9243    6.9257   284.68   279.42   277.97   279.17   284.79   284.41  1246749114.000  1
1  9  3 42.855   240.47   248.51   245.14   239.35    6.9250    6.9279   284.90   280.97   279.72   280.01   285.01   284.41  1246749118.000  1
1  9  3 46.749   240.06   247.57   244.98   238.70    6.9226    6.9247   284.81   279.77   278.54   279.53   284.79   284.19  1246749122.000  1
1  9  3 50.642   240.22   249.12   247.36   240.34    6.9228    6.9240   285.12   281.76   281.97   281.69   285.91   285.06  1246749126.000  1
1  9  3 54.536   240.87   250.30   247.87   241.16    6.9226    6.9240   285.12   280.92   280.34   280.85   286.18   285.28  1246749130.000  1
1  9  3 58.429   239.25   250.14   248.08   238.71    6.9208    6.9218   283.97   278.05   277.09   278.87   286.32   285.37  1246749134.000  1
1  9  4  2.323   237.45   249.18   246.27   237.53    6.9236    6.9247   285.16   281.45   281.13   281.20   286.27   285.06  1246749138.000  1
1  9  4  6.216   237.81   248.91   245.58   237.90    6.9223    6.9232   285.08   280.30   278.23   278.95   286.05   285.28  1246749142.000  1
1  9  4 10.109   237.22   249.14   246.86   237.33    6.9233    6.9247   284.59   278.85   277.26   278.95   286.27   285.46  1246749146.000  1
1  9  4 14.003   237.13   249.52   247.12   237.76    6.9236    6.9254   285.03   280.48   279.68   280.19   286.05   285.33  1246749150.000  1
1  9  4 17.896   236.49   247.52   244.94   237.54    6.9221    6.9225   285.21   281.23   280.43   280.80   286.09   285.55  1246749154.000  1
1  9  4 21.790   237.23   248.86   245.61   237.95    6.9227    6.9234   284.68   279.77   278.05   279.17   285.78   285.11  1246749158.000  1
1  9  4 25.683   237.17   248.36   243.72   237.72    6.9229    6.9248   285.39   281.63   280.87   280.89   286.14   285.28  1246749162.000  1
1  9  4 29.577   237.44   247.25   246.87   237.94    6.9197    6.9212   284.85   279.64   277.97   279.00   285.82   285.11  1246749166.000  1
```

La primera fila contiene datos de control, a saber:

- El número de muestras recogidas en la sesión
- Duración de la sesión de recogida de datos
- Intervalo de espera entre una toma y la siguiente
- Datos en relación al estado del sensor de viento

Las siguientes lineas contienen la información que corresponden a los datos tomados en la misma superficie marciana, estos datos son respectivamente:

- Sol, día marciano.
- Hora.
- Minuto.
- Segundo.
- Temperatura recogida por el sensor de calor situado en lo alto del mástil del rover.
- Temperatura del sensor intermedio del mástil.
- Temperatura del sensor en la parte baja del mástil.
- Temperatura del sensor situado bajo el mástil.
- Presión en un rango de 6 a 10 milibares.
- Presión en un rango de 0 a 12 milibares.
- Temperatura del sensor situado en el primer segmento de sensor de viento.
- Temperatura del sensor situado en el segundo segmento del sensor de viento.
- Temperatura del sensor situado en el tercer segmento del sensor de viento.
- Temperatura del sensor situado en el cuarto segmento del sensor de viento.
- Temperatura del sensor situado en el quinto segmento del sensor de viento.
- Temperatura del sensor situado en el sexto segmento del sensor de viento.
- Instante de tiempo de la nave, cuyo valor se cuenta desde que la misión fue iniciada.
- Activación o no del sensor de viento.

Todos los datos referentes a la temperatura están medidos en Kelvin, la unidad internacional de temperatura. En la siguiente figura podemos ver los instrumentos de medición que estaban presentes en la misión.

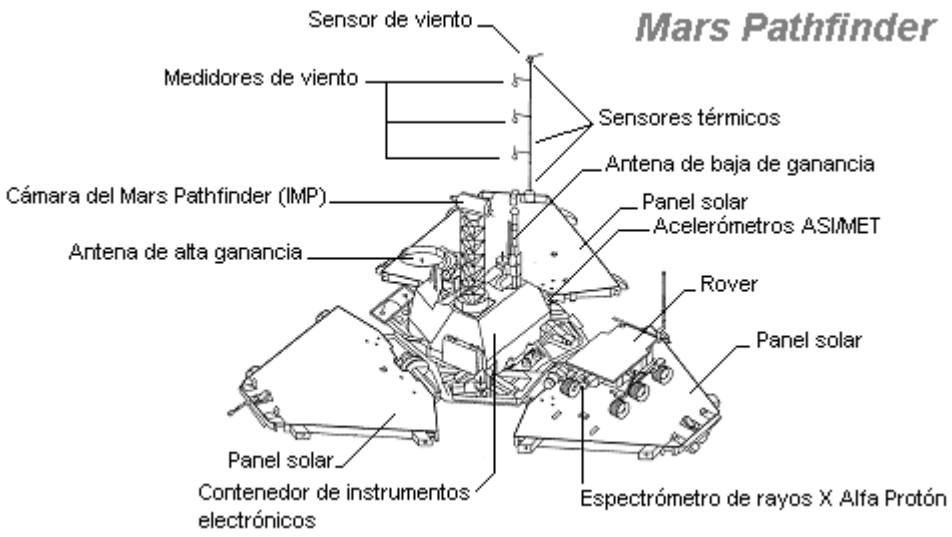


Figura 2. Instrumentos del Mars Pathfinder

Para nuestro trabajo hemos optado por utilizar únicamente la primera columna de las correspondientes a la temperatura. Que se trata de la situada en la parte alta del mástil, pues al estar más alejado de la base sus medidas son más precisas. Debido a que al estar en una posición más alta está más alejada del resto de aparatos, instrumentos y baterías que inevitablemente dada su naturaleza generan calor. Por tanto los sensores más cercanos se verán afectados por dicho calor.

Para la presión hemos utilizado la segunda columna, pues contiene los datos referentes al segundo barómetro, que aunque es un poco menos preciso, mide desde 0 hasta 12 milibares.

Una vez hemos descargado y obtenido los datos, estos en formato XML van a tener el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Listado>
    <Muestra>
        <Dia>7.0</Dia>
        <Hora>0.0</Hora>
        <Minuto>4.0</Minuto>
        <Segundo>33.472</Segundo>
        <PresionGlobal>6.7213</PresionGlobal>
        <PresionSesgada>6.7195</PresionSesgada>
        <TemperaturaAlto>208.25</TemperaturaAlto>
        <TemperaturaMedio>208.9</TemperaturaMedio>
        <TemperaturaBajo>203.55</TemperaturaBajo>
    </Muestra>
    <Muestra>
        <Dia>7.0</Dia>
        <Hora>0.0</Hora>
        <Minuto>4.0</Minuto>
        <Segundo>37.366</Segundo>
        <PresionGlobal>6.7220</PresionGlobal>
        <PresionSesgada>6.7200</PresionSesgada>
        <TemperaturaAlto>207.82</TemperaturaAlto>
        <TemperaturaMedio>207.2</TemperaturaMedio>
        <TemperaturaBajo>201.32</TemperaturaBajo>
```

```
    </Muestra>
    <Muestra>
        <Dia>7.0</Dia>
        <Hora>0.0</Hora>
        <Minuto>4.0</Minuto>
        <Segundo>41.259</Segundo>
        <PresionGlobal>6.7198</PresionGlobal>
        <PresionSesgada>6.7190</PresionSesgada>
        <TemperaturaAlto>207.81</TemperaturaAlto>
        <TemperaturaMedio>206.82</TemperaturaMedio>
        <TemperaturaBajo>201.63</TemperaturaBajo>
    </Muestra>
    …....
</Listado>
```

Para nuestro estudio no consideramos necesario almacenar los datos referentes a los sensores de temperatura situados en los segmentos del sensor de viento. Puesto que la medición de temperatura más fiable es la situada en el sensor en lo alto del mástil como ya dijimos anteriormente.

Cada muestra recogida contiene los valores del día y la hora exactas en que se tomaron, la propiedad Dia se refiere al Sol, día marciano, en que la muestra fue tomada.

La propiedad PresionGlobal contiene la información recogida por el barómetro que mide de 0 a 12 milibares, mientras que PresionSesgada contiene la información del otro barómetro, aquel que mide desde 6 a 10 milibares.

Adjuntamos como anexo el código fuente de la clase Spider, así como la clase Descargador, dichas clases se encuentran en el paquete DatosNasa de nuestro código fuente.

```
/**
 * @author M. Javier Miguel García
 * @author Enrique de Lucas Casamayor
 */
public class Spider
{

   /**
    * Su función es llamar al método descargarTodo
    * @param args
    */
   public static void main(String[] args)
   {
      descargarTodo();
      System.out.println("Datos descargados correctamente");
   }

   /**
```

```
    * Se encarga de ir llamando iterativamente al método descargarPagina,
    * indicándole la página que debe ser descargada en cada momento.
    *
    */
   public static void descargarTodo()
   {
      /**
       * Una URL típica es:
       * http://pds-
atmospheres.nmsu.edu/PDS/data/mpam_0001/surf_rdr/scidata/srAAxxs/srAABBs.tab
       *
       * donde AA representa la carpeta y BB representa el archivo de la misma.
       */

      ListadoMuestrasPF muestras = new ListadoMuestrasPF();
      for (int i = 0; i <15 ; i++)
      {
         for (int j = 0; j < 100; j++)
         {
            String dir = "http://pds-
atmospheres.nmsu.edu/PDS/data/mpam_0001/surf_rdr/scidata/sr";
                                                  //url base
            Integer num = new Integer(i);
            String carp = num.toString();
            while (carp.length() < 2)
            {
               carp = "0" + carp; //insertamos el número de la carpeta
            }
            num = new Integer(j);
            String fichero = num.toString();
            while (fichero.length() < 2)
            {
               fichero = "0" + fichero;
            }
            String extension = "s.tab";
            String url = dir + carp + "xxs/sr" + carp + fichero + extension;
                                          //completamos la url
            System.out.println(url); //mostramos la url en proceso
            ListadoMuestrasPF buffer=new ListadoMuestrasPF();
            Integer ultimoDia=null;
            descargarPagina(url,buffer,ultimoDia);
                     //obtenemos todas las muestras de esa url

            try { //esperamos 10 segundos para seguir bajando páginas
               Thread.sleep(10000);
            } catch (InterruptedException ex) {
```

```
            Logger.getLogger(Spider.class.getName()).log(Level.SEVERE,
                    null, ex);
            }
        }
    }
}


/**
 * Este método se encarga de descargar y extraer los datos de una de las
 * páginas especificadas.
 *
 * @param url Dirección de la url a descargar
 * @param buffer Listado de muestras correspondientes al mismo día
 * @param ultimoDia Indica el día del que se recogió la última muestra
 */
public static void descargarPagina(String url,
        ListadoMuestrasPF buffer, Integer ultimoDia )
{
    ListadoMuestrasPF datos=buffer;
    try
    {
        Descargador d = new Descargador(url);
        String s =d.getCodigoHTML(); //Obtenemos los datos de la página
        String[] lineas= s.split("\n");

        for (int i=1;i<lineas.length;i++) //ignoramos la primera línea pues
                                    //tiene la cabecera.
        {
            ArrayList<String> campos= new ArrayList<String>();
            String[] scampos = lineas[i].split(" ");
            for (int j=0; j<scampos.length; j++)
            {
                if (!scampos[j].equals(""))
                    campos.add(scampos[j]);
            }

            //extraemos uno a uno cada dato, según la columna a la que
            //pertenece
            NumeroDecimal dia=new NumeroDecimal(campos.get(0));
            NumeroDecimal hora=new NumeroDecimal(campos.get(1));
            NumeroDecimal minuto=new NumeroDecimal(campos.get(2));
            NumeroDecimal segundo=new NumeroDecimal(campos.get(3));
            NumeroDecimal temperaturaAlto=new NumeroDecimal(campos.get(4));
            NumeroDecimal temperaturaMedio=new NumeroDecimal(campos.get(5));
            NumeroDecimal temperaturaBajo=new NumeroDecimal(campos.get(6));
```

```
            NumeroDecimal presionSesgada=new NumeroDecimal(campos.get(8));
            NumeroDecimal presionGlobal=new NumeroDecimal(campos.get(9));



           MuestraPF m = new MuestraPF(dia, hora, minuto, segundo,
                                temperaturaAlto, temperaturaMedio,
                                temperaturaBajo, presionSesgada,
                                presionGlobal);


            if (ultimoDia==null) //si es la primera vez que se ejecuta..
            {
               ultimoDia=new Integer(dia.getParteEntera()); //iniciamos el
                                                  //valor
            }
            else
            {
               if (ultimoDia != dia.getParteEntera())
                  //si encontramos un día nuevo, es porque los datos del
                  //día anterior ya han sido recogidos
               {
                  ArrayList<MuestraPF> mues =
                        datos.filtrarPorDia(ultimoDia);
                  ListadoMuestrasPF muestrasDia =
                        new ListadoMuestrasPF(mues);
                  //almacenamos los datos en un archivo xml
                  muestrasDia.guardarXML("sol" + ultimoDia + ".xml");
                  System.out.println("---------------------------------");
                  System.out.println("sol" + ultimoDia + ".xml");
                  System.out.println("---------------------------------");
                  ultimoDia=new Integer(dia.getParteEntera());
                  datos=new ListadoMuestrasPF(); //reiniciamos el buffer
               }
            }
            datos.add(m);
         }
      }
      catch (IOException ex)
      {
         System.out.println("Error al descargar el fichero: " + url);
      }
   }
}
```

Y finalmente la clase Descargador, que es la encargada de obtener los datos del código HTML de una página dada.

```
/**
 * Esta clase se encarga de descargar el HTML de una URL
 * @author M. Javier Miguel García
 * @author Enrique de Lucas Casamayor
 *
 */
public class Descargador
{

  private String codigoHTML;

  /**
   *
   * @return El código HTML descargado de la url
   */
  public String getCodigoHTML()
  {

      return codigoHTML.toString();
  }

  /**
   *
   * @param url URL a descargar
   * @throws IOException Lanza esta excepción en el caso de que no exista
   *                     la página
   */
  public Descargador(String url) throws IOException
  {
      codigoHTML = new String();
      URL u;
      InputStream is = null;
      DataInputStream dis;

      String s;

      u = new URL(url);
      is = u.openStream();  //esta operación es la que lanza la IOException,
                            //suele saltar cuando la URL no es válida
      dis = new DataInputStream(new BufferedInputStream(is));
```

```
        while ((s = dis.readLine()) != null)
        {
            codigoHTML+= s+"\n";
        }
        is.close();
    }
}
```

Los datos procedentes de la Mars Pathfinder se pueden encontrar en la URL que indica:

http://pds-atmospheres.nmsu.edu/cgi-bin/getdir.pl?volume=mpam_0001&dir=surf_rdr/scidata

Si nos fijamos en la página aparecen 14 carpetas que contienen todas las sesiones de medida. Cuando decimos una sesión de medida queremos decir que la sonda comienza a medir, toma un número variable de muestras y finaliza dicha sesión.

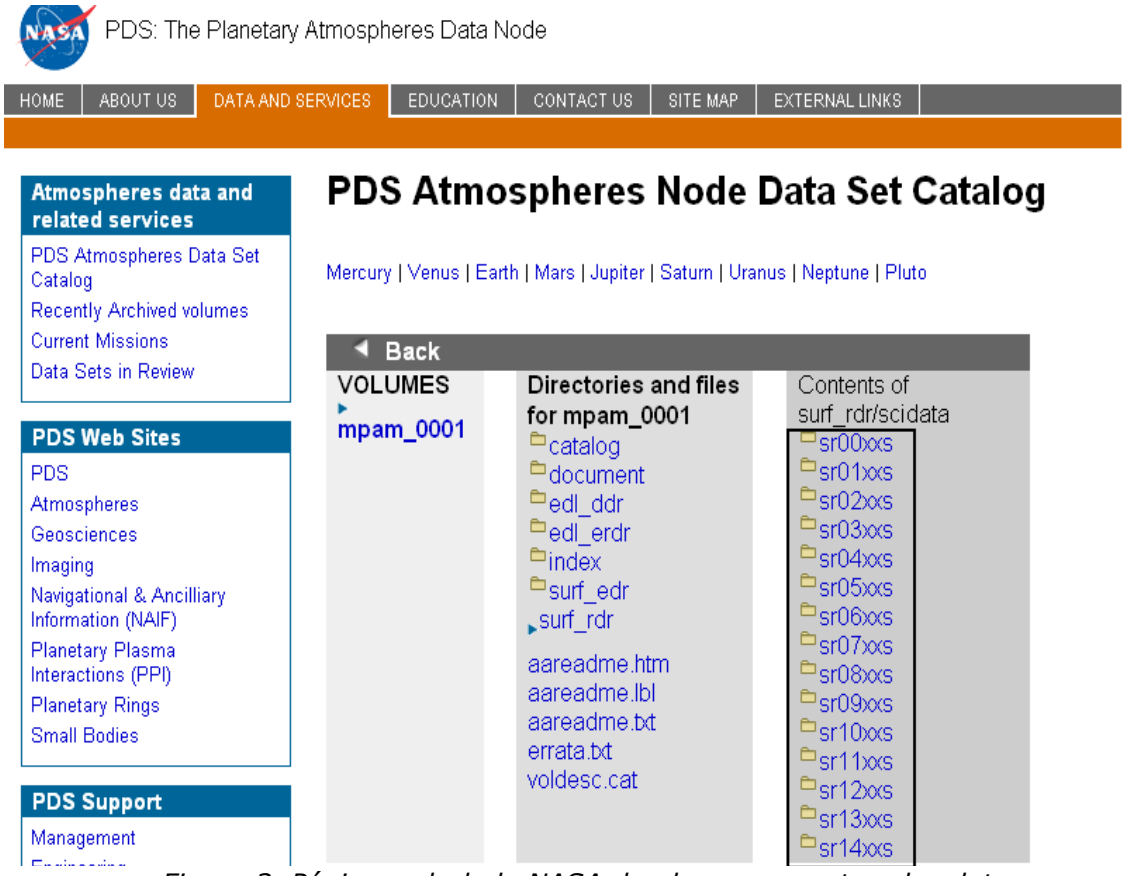


Figura 3. Página web de la NASA donde se encuentran los datos

Todas las carpetas a excepción de la primera y la última contienen cien sesiones de medidas. La primera no llega a cien porque las primeras medidas

fueron tomadas antes de llegar a la superficie de Marte, y la última carpeta contiene las últimas sesiones realizadas, que no llegan a cien.

La cantidad de muestras tomadas en cada una de las sesiones puede variar significativamente, así como el intervalo de medición entre una y otra. Además es posible que un mismo día marciano se realicen varias sesiones.

Como la NASA mantiene los datos tal y como fueron recogidos, su análisis se hace más complicado, por eso decidimos programar un spider que nos permitiese acceder a la información y almacenarla de la forma que queríamos para usarla en nuestro trabajo.

4. Algoritmos disparadores en misiones espaciales

Existen muchos tipos de algoritmos disparadores para la función que necesitamos de ellos, es decir la de predicción de eventos. No obstante hay algunos que funcionan mejor que otros para el contexto de las misiones espaciales. Debemos tener en cuenta de que a pesar de lo que pueda parecer, las misiones espaciales no llevan la capacidad de cálculo superior que cabría esperar. Esto se debe a varias razones:

- Peso y espacio, estos suelen ser muy limitados, debemos tener en cuenta que un procesador muy avanzado también ocupa y pesa mucho. Lo cual no es bueno para la misión.

- Limitaciones energéticas, se deben a que las misiones pueden durar mucho tiempo, meses, e incluso años, de forma que los equipos de cálculo deben consumir la menor cantidad de energía que sea posible.

- Precaución, no podemos permitirnos que una misión falle porque el procesador último modelo que hemos incluido falle. Por ello se tiende a ser conservador y a utilizar equipos que ya han sido exitosos en otras misiones pero que pueden estar tecnológicamente obsoletos.

Debido a que la capacidad de cálculo está tecnológicamente limitada los mejores algoritmos para las misiones espaciales son aquellos que pueden realizar los cálculos de una forma sencilla y eficiente. Es por ello que los algoritmos disparadores que más se suelen utilizar son los del tipo STA/LTA que estudiaremos más adelante, pues a parte de ser muy eficientes en sus resultados, consumen pocos recursos y dan una respuesta rápida.

Además de esto tienen una gran versatilidad y se pueden aplicar a distintos contextos, en nuestro caso los hemos utilizado para construir un detector de dust devils, pero también pueden servir para controlar la actividad volcánica y sísmica entre otros ejemplos.

5. Algoritmos STA/LTA

Los algoritmos conocidos como STA/LTA son los más utilizados entre los algoritmos detectores de eventos. Su sencillez les otorga una gran versatilidad a la hora de ser utilizados, como ya dijimos anteriormente cuando operamos fuera de la Tierra nos encontramos en un entorno hostil y sobre todo, lejos de los equipos de mantenimiento y de las baterías. Por lo tanto cada elemento de la misión debe cumplir su función de la mejor forma posible de manera que logremos evitar todos los obstáculos que nos surjan durante el trascurso de la misma. Llegados a este punto podemos afirmar que los algoritmos STA/LTA, nos permiten ahorrar energía contribuyendo al éxito total de la misión.

5.1. Descripción

Estos algoritmos fueron descritos por primera vez por Lee y Stewart, 1981, su nombre es un acrónimo de su significado en inglés Short Term Average / Long Term Average. Una traducción puede ser media a corto plazo / media a largo plazo. Ciertamente su nombre lo describe bastante bien porque precisamente así se calcula el resultado del algoritmo.

A grandes rasgos la idea se basa en que tenemos dos conjuntos de muestras, pudiendo ser estas muestras valores de temperatura, presión, radiación, etc, dependiendo la magnitud física que estemos midiendo.

De forma que la STA, nos permitirá calcular lo que podríamos definir como el valor actual. Este valor actual, consiste en una aproximación de los últimas muestras procesadas. Pero no obstante pueden variarse estos valores, según las necesidades, el contexto y las aplicaciones en concreto. Posteriormente explicaremos los valores que utilizamos nosotros en nuestras simulaciones. En adelante llamaremos CV al valor actual, por ser estas sus siglas en inglés: Current Value. Este CV, se comparará con una predicción. Esta predicción, PV (Predicted Value), la obtendremos a partir de los valores almacenados por la LTA.

En general siendo N<M:

STA	Aproximación a partir de las últimas **N** muestras, a partir del cual calcular el CV (Current Value).
LTA	Aproximación a partir de las últimas **M** muestras, a partir del cual calcular el PV (Predicted Value).

Por lo tanto todas las muestras recogidas en la STA están contenidas en la LTA, pero no al revés.

Antes de proseguir debemos aclarar que existen muchas posibles variantes para obtener la aproximación, la más común consiste en realizar la media aritmética de los valores que almacenamos, de forma que para calcular la aproximación de la STA calcularemos la media aritmética de las últimas muestras recogidas e igualmente ocurre con la LTA.

No obstante hay otras formas de realizar esta aproximación, una variante sería utilizar la media de los cuadrados, esta nos puede ser útil por ejemplo en situaciones en las cuales tenemos magnitudes positivas y negativas las cuales queremos utilizar simplemente en valor absoluto. Pero como decíamos antes la forma más común es utilizar la media aritmética, pues como ya vimos los algoritmos más prácticos son aquellos que realizan menos cálculos.

Una vez hemos calculado los valores CV y PV, en función de las muestras recogidas por la STA y LTA respectivamente. Sólo tenemos que comparar estos valores y comprobar si se superan los valores límites establecidos (Threshold values):

$$CV / PV > THR_{subida}$$
$$CV / PV < THR_{bajada}$$

Estas condiciones sirven para disparar respectivamente los eventos de subida y bajada. Que representan un cambio significativo y repentino en la mágnitud estudiada.

5.2. Implementación eficiente

Como ya hemos visto los algoritmos STA/LTA son muy sencillos y requieren poca cantidad de cálculo, no obstante debemos tener cuidado para implementarlos de una forma eficiente. Esta explicación es válida para las posibles variaciones que ya mencionamos antes, pues donde nos vamos a centrar sobretodo es en cómo acceder a los datos almacenados de la mejor forma posible. Como todas las variantes deben acceder a dichos datos, a pesar de que luego realicen distintos cálculos con los mismos, si optimizamos este acceso mejoraremos el rendimiento global del algoritmo.

Nuestro objetivo es mejorar al máximo en la medida de lo que nos sea posible el tiempo que se tarda en calcular el resultado, es decir, cuánto tiempo requiere el algoritmo para decidir si se dispara o no, una vez hemos recibido una nueva muestra, o dato de entrada.

Ya sea que usemos los datos para calcular una media, o una varianza con ellos, es obligatorio tener almacenados dichos datos, o muestras recogidas. Por ejemplo, si tenemos un módulo STA, que nos devuelve la media aritmética de las 10 últimas muestras de temperatura, no hace falta decir que debemos tener dichas muestras almacenadas. Esto nos lleva a otro punto importante, la capacidad de almacenamiento. Esta debe ser la mínima necesaria para el correcto funcionamiento del sistema.

Como ya dijimos, en una misión espacial no podemos permitirnos tener recursos que no vamos a utilizar, y la capacidad de almacenamiento es uno de estos recursos. Por supuesto si el algoritmo se ejecutara sobre un ordenador en la Tierra, con una gran cantidad de memoría RAM, sería mucho más sencillo y podríamos intentar hacer más cálculos, pero en nuestro caso debemos ser muy austeros con los recursos.

5.2.1. Uso de un array circular

De esta forma en nuestro trabajo optamos por utilizar una estructura de datos muy popular, el array circular. Esta estructura nos permite almacenar un numero determinado de datos, tenemos que remarcar que debemos conocer el número de datos máximo que vamos a necesitar de antemano.

Por suerte nuestro módulos STA/LTA, tienen unos valores concretos que habremos determinado antes de implementarlos, de forma que son candidatos propicios para utilizar un array circular.

Un array circular consiste, hablando específicamente de software (luego veremos como lo implementamos en hardware) en un conjunto de tamaño finito y conocido de posiciones de memoria consecutivas. De forma que si por ejemplo nuestro módulo LTA recoge las últimas 128 muestras, necesitaremos un hueco en memoría de 128 posiciones seguidas, y sólo necesitaremos esa cantidad de memoria. Naturalmente cualquier ordenador que tengamos en nuestras casas tiene esa cantidad de memoria y mucha más, pero al diseñar el hardware podemos ahorrarnos la necesidad de instalar una memoria RAM de propósito general, pues solo vamos a necesitar esa cantidad de memoria, instalando una memoria RAM especifica dedicada a nuestros propósitos. Así esta memoria, después de una carga inicial de datos, contendrá siempre información útil y necesaria, y no tendrá más memoria de la que necesita aprovechando al 100% los recursos del hardware.

Además del array para almacenar los datos, necesitaremos un registro en el cual almacenar cuál es el primer elemento, y otro para saber cuál es el ultimo. De forma que cuando introduzcamos un nuevo dato, este ocupe la posición de memoría del elemento que lleva más tiempo en el array, manteniendo el orden de entrada que teníamos. La siguiente ilustración muestra como quedaría un array circular de siete posiciones, después de haber insertado de forma consecutiva los números del uno al ocho.

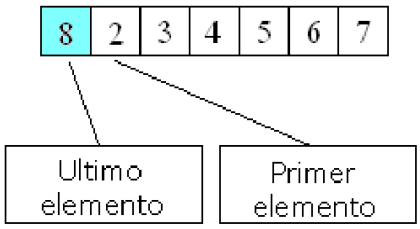


Figura 4. Esquema del array circular.

De esta forma podemos acceder rápidamente a los datos y tenerlos ordenados siempre por el orden de llegada, que es precisamente lo que necesitamos en la implementación de nuestro algoritmo. Pues querremos tener las muestras recogidas ordenadas por su momento llegada. Así pues la forma en que los módulos STA y LTA almacenarán los datos será por medio de un array circular para cada uno.

5.2.2. Cálculo óptimo de la predicción

Con esto hemos resuelto el acceso y almacenamiento eficiente de los datos. Pero aún podemos hacer algunas mejoras. En el caso más sencillo, es decir, cuando calculamos la media aritmética de las muestras recogidas podemos hacer una gran mejora. Cada vez que nos llega un dato, lo guardamos en la memoria, accedemos uno por uno a los datos almacenados para sumarlos y después lo dividimos entre el número correspondiente, es decir la cantidad de muestras para las que hayamos configurado la STA y la LTA.

En lugar de hacerlo de esta forma, pues debemos acceder muchas veces a memoria, lo que vamos a hacer es guardar en un registro la suma acumulada. De esta forma nos ahorramos tener que acceder tantas veces a memoria.

Para mantener actualizado el valor de la suma acumulada, cada vez que nos llega un dato se lo sumamos al registro que guarda la suma, y después le restamos la muestra más que lleva más tiempo en el módulo. Por suerte ya hemos implementado nuestro array circular, como ya vimos antes, de forma que podemos acceder a este valor. Así pues, en lugar de acceder a muchos elementos de la memoria, sumarlos y hacer la división, pasamos a hacer una única suma, una única resta y la división, independientemente del número de muestras que recojan la STA y la LTA.

5.2.3. Costes en tiempo y espacio.

El coste en espacio dependerá linealmente del valor que le demos a la STA y la LTA. Por lo tanto podemos decir que su orden es de O(n) siendo n el valor de la LTA, puesto que este será mayor que el valor de la STA.

El coste de ejecución de procesar una nueva entrada, puesto que solo hacemos una suma, una resta y una división, es constante, es decir O(1).

5.3. Algoritmo de Allen

Este algoritmo está aplicado exclusivamente a la detección de anomalías sísmicas en la Tierra. Se trata de un algoritmo de STA/LTA pero con algunas optimizaciones realizadas por el investigador Rex Allen.

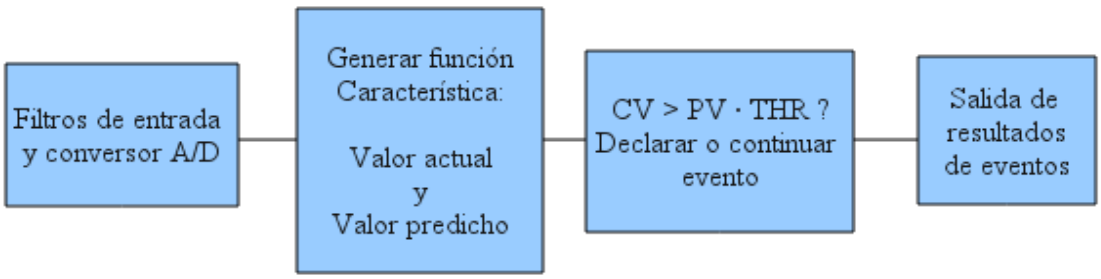


Figura 5. Esquema del flujo de datos en el algoritmo de Allen.

Lo que diferencia al algoritmo de Allen de los algoritmos STA/LTA, es el trabajo previo de procesado de las entradas que realiza. Estos se realizan para el propósito específico de predecir eventos de tipo sismológico.

Los filtros de entrada antialising sirven para limpiar los datos de posibles ruidos, mientras que el conversor Analógico/Digital, permite que los datos puedan ser computados por un sistema informático.

Una vez la muestra está filtrada y convertida a datos digitales, comienza la parte interesante. A dicha muestra se le aplica una función característica.

La función característica nos ayuda a que la señal sea más sensible a posibles variaciones de amplitud y/o frecuencia. Algunas funciones características pueden ser:

CF(i) = |Y(i)| (Valor absoluto**)**

CF(i) = Y(i)2 (Cuadrado de la señal**)**

CF(i) = Y(i)2 + K(Y(i) – Y(i-1))2 (*)

(*)Donde K es una constante cuyo valor se determina para cada estación de medida.

Por la forma que tienen las señales captadas de la actividad sismológica, es interesante procesarlas con la función característica. Sin embargo para nuestro trabajo, las señales que no son más que las medidas de temperatura o presión, por su naturaleza al no ser ondas, no nos resultaba práctica ni necesaria la aplicación de una función característica. Por esa razón no la aplicamos. El resto del esquema de flujo simplemente aplica un algoritmo STA/LTA y analiza los resultados dados.

A pesar de que para nuestro trabajo no hemos utilizado las optimizaciones que este algoritmo presenta, hemos considerado importante presentarlo al lector pues ofrece mejoras que son muy interesantes aunque se aplican en un contexto de estudio distinto del de este trabajo.

6. Simulador software

6.1. Interfaz de usuario, GUI

Para nuestro estudio hemos programado un simulador que nos permita trabajar con los algoritmos de triggering. Como ya vimos anteriormente los datos que vamos a utilizar están en formato XML. El aspecto de la pantalla principal es el siguiente:

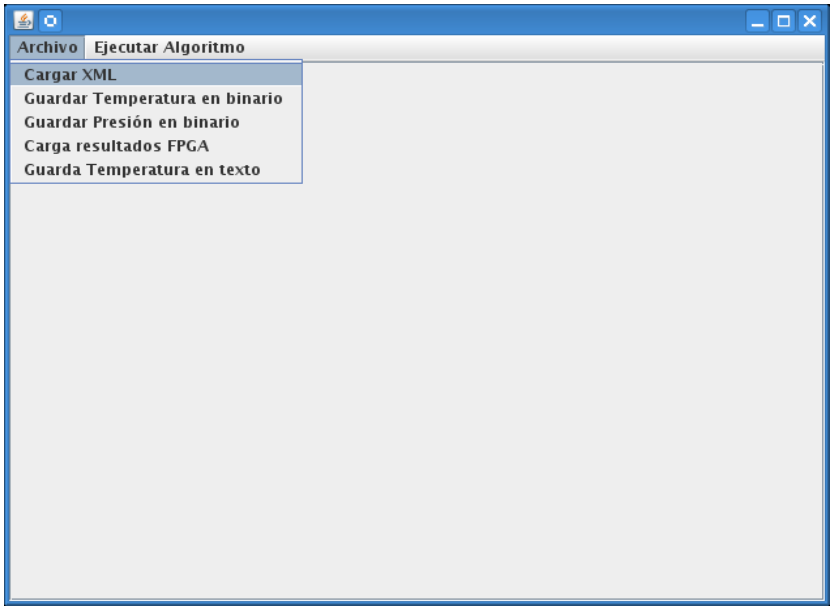


Gráfico. Pantalla principal.

Utilizaremos el menú archivo para seleccionar la opción Cargar XML, que nos permitirá abrir uno de los archivos de muestras:

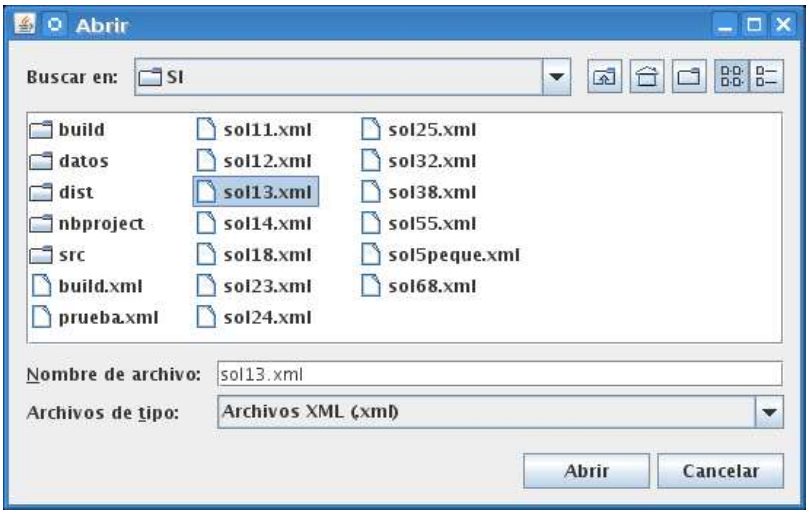
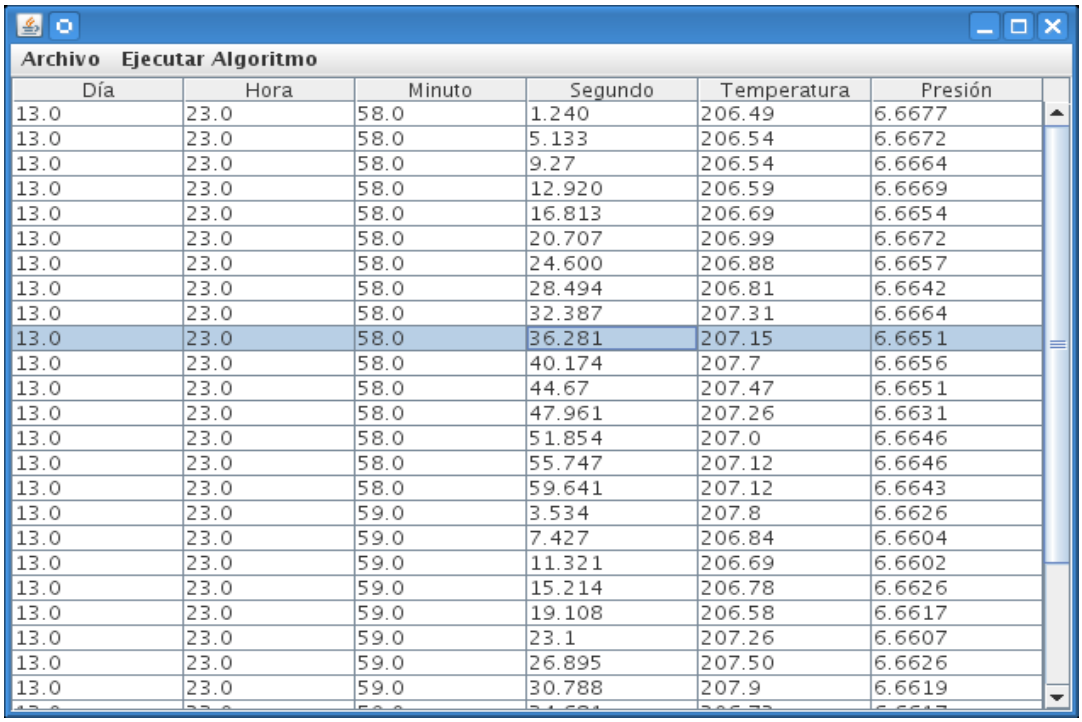


Gráfico. Cargar archivos.

Una vez hayamos pulsado Abrir, en la ventana principal aparecerán las muestras correspondientes a dicho fichero, que tendrá este aspecto:



Día	Hora	Minuto	Segundo	Temperatura	Presión
13.0	23.0	58.0	1.240	206.49	6.6677
13.0	23.0	58.0	5.133	206.54	6.6672
13.0	23.0	58.0	9.27	206.54	6.6664
13.0	23.0	58.0	12.920	206.59	6.6669
13.0	23.0	58.0	16.813	206.69	6.6654
13.0	23.0	58.0	20.707	206.99	6.6672
13.0	23.0	58.0	24.600	206.88	6.6657
13.0	23.0	58.0	28.494	206.81	6.6642
13.0	23.0	58.0	32.387	207.31	6.6664
13.0	23.0	58.0	36.281	207.15	6.6651
13.0	23.0	58.0	40.174	207.7	6.6656
13.0	23.0	58.0	44.67	207.47	6.6651
13.0	23.0	58.0	47.961	207.26	6.6631
13.0	23.0	58.0	51.854	207.0	6.6646
13.0	23.0	58.0	55.747	207.12	6.6646
13.0	23.0	58.0	59.641	207.12	6.6643
13.0	23.0	59.0	3.534	207.8	6.6626
13.0	23.0	59.0	7.427	206.84	6.6604
13.0	23.0	59.0	11.321	206.69	6.6602
13.0	23.0	59.0	15.214	206.78	6.6626
13.0	23.0	59.0	19.108	206.58	6.6617
13.0	23.0	59.0	23.1	207.26	6.6607
13.0	23.0	59.0	26.895	207.50	6.6626
13.0	23.0	59.0	30.788	207.9	6.6619

Gráfico. Contenido del archivo XML cargado.

Ahora ya podemos ver todas las muestras recogidas para el sol13, como podemos observar están perfectamente identificadas, por día, hora, minuto y segundo. Además se puede leer la temperatura y la presión recogidas en la muestra.

Cabe mencionar que la Temperatura corresponde al valor Temperatura-Alto que no es más que el valor del sensor de temperatura situado en la parte alta del mástil. Y la presión, es el valor PresionGlobal que ya mecionamos anteriormente.

Llegados a este punto podemos ejecutar uno de los algoritmos con las muestras que ya han sido cargadas, para ello utilizaremos el menú Ejecutar Algoritmo y seleccionaremos algoritmo STA/LTA Medias:

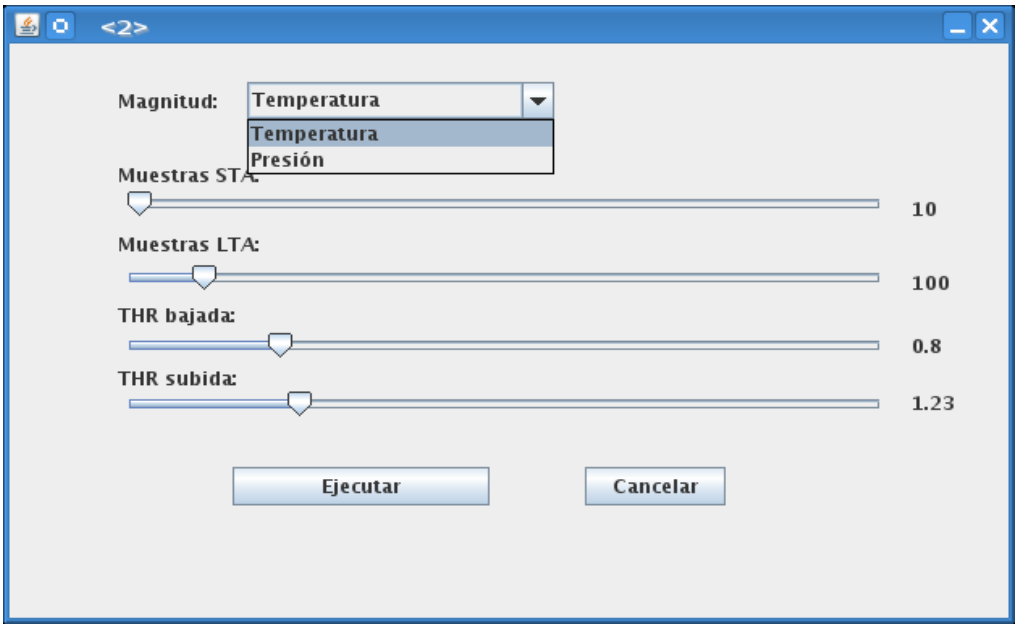


Gráfico. Selección de parámetros

La ventana que nos aparece sirve para configurar los parámetros del algoritmo, por lo pronto debemos elegir qué Magnitud queremos analizar, ya sea la presión o la temperatura. Para hacer una prueba seleccionaremos en este caso la temperatura, y los siguientes valores:

Muestras STA	128
THR bajada	1.03
Muestras STA	128
THR bajada	1.03

En este momento solo debemos pulsar el botón ejecutar para que se ejecute el algoritmo y nos aparezcan los resultados en una gráfica:

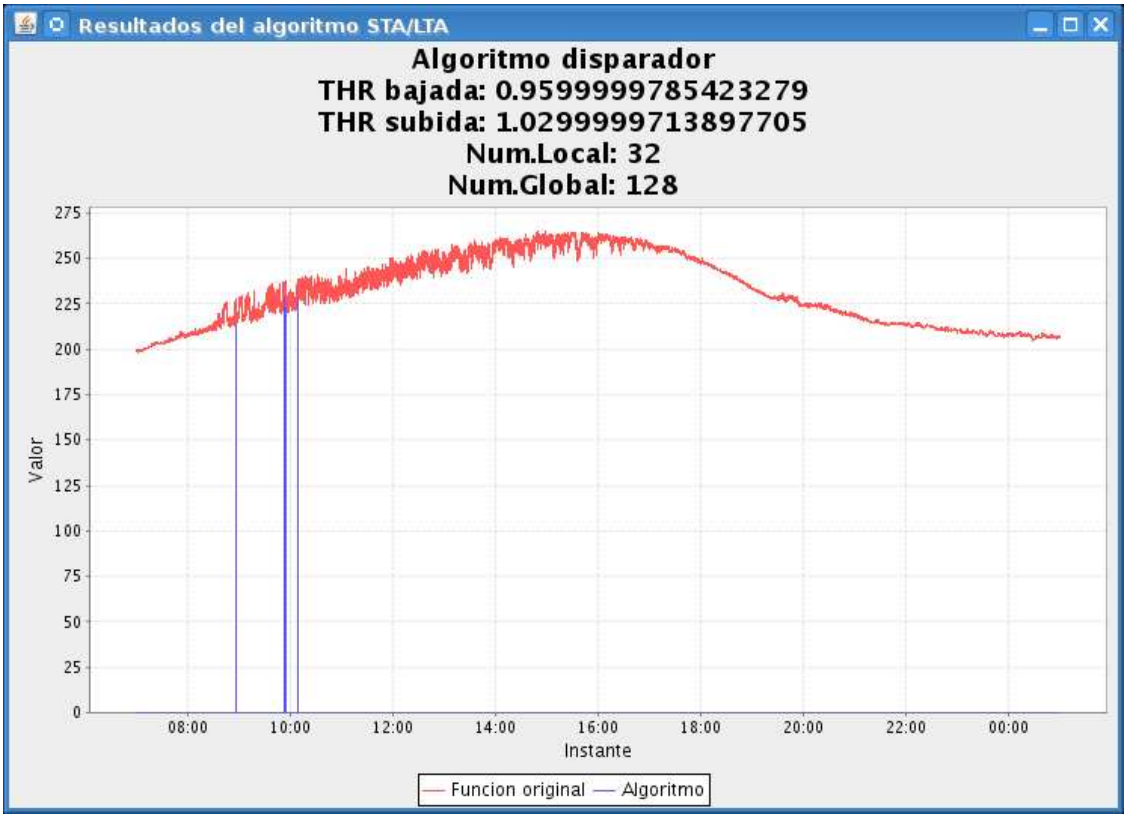


Gráfico. Resultado de la ejecución del algoritmo.

Como podemos apreciar en la imagen, en rojo tenemos la variación de temperatura y en azul tenemos el estado del algoritmo, que valdrá cero cuando no se ha disparado el mismo y el valor de la función en caso de que sea haya disparado.

Podemos hacer zoom, pulsando sobre el gráfico, en caso de que estemos interesados en inspeccionar una parte concreta del mismo. Para ello pulsamos cualquier parte de la grafica, y arrastramos el cuadro azul hacia abajo y hacia la derecha. Haciendo zoom podremos ver mejor donde se ha disparado el algoritmo.

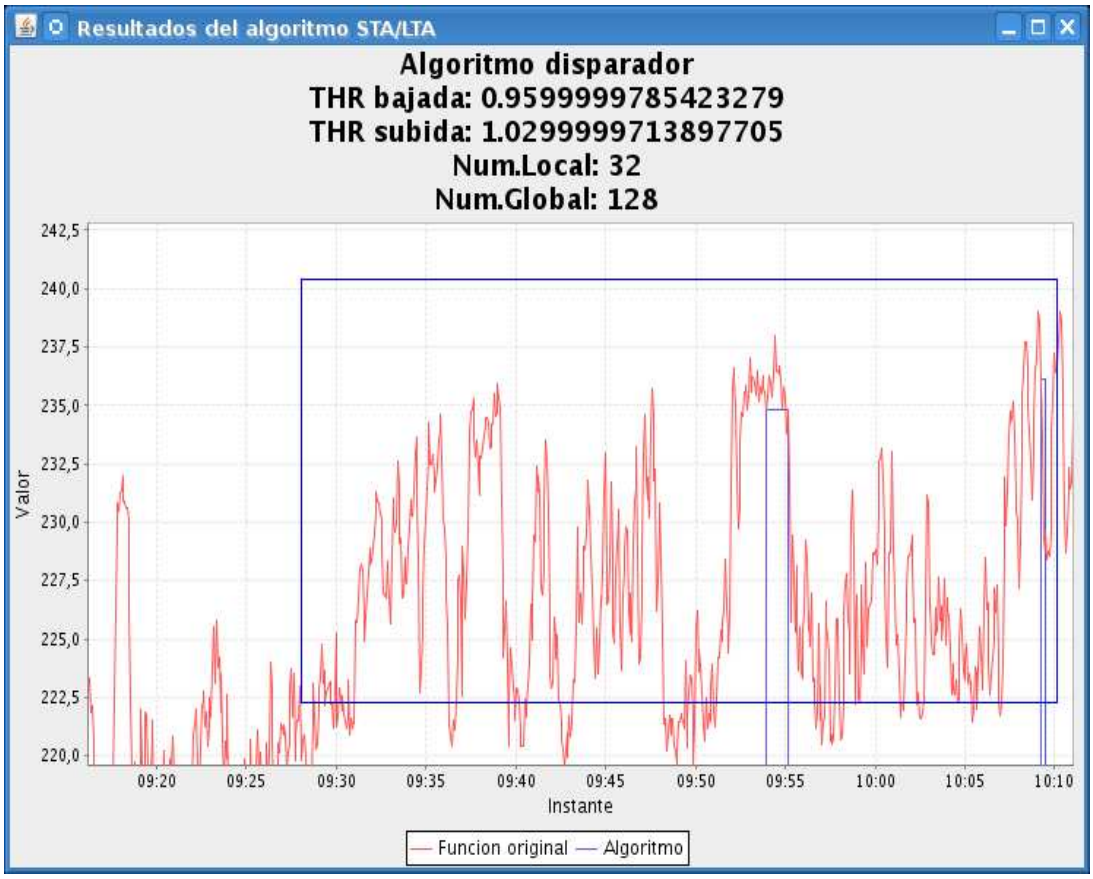


Gráfico. Selección de área en la gráfica.

El recuadro azul de la imagen representa el cuadrado de selección de zoom. También se pueden seleccionar las opciones de escalar el zoom pulsando el botón derecho sobre el gráfico.

Llegados a este punto también nos interesa saber cómo ejecutar el detector de dust devils. Para ello solo tenemos que seguir los mismos pasos que hicimos antes con el algoritmo STA/LTA, es decir, cargar los datos desde un archivo XML. Pero en este caso seleccionaremos en el menú “Ejecutar Algoritmo” la opción de “Detector de dust devils”. Una vez hecho esto, nos aparecerá la siguiente ventana para configurar el detector:

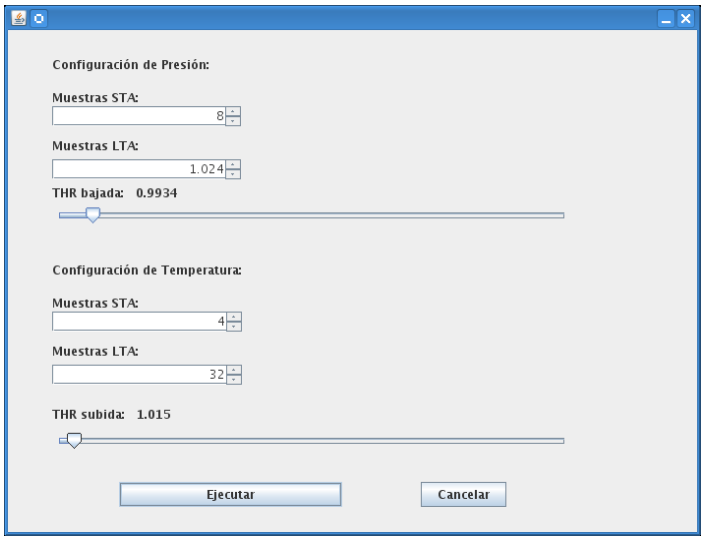


Gráfico. Selección de parámetros

Aquí hay algo distinto del configurador de algoritmos STA/LTA. La diferencia radica en que no se pueden establecer los parámetros de THR de subida de la presión ni el THR de bajada de la temperatura. El motivo de esto es que para detectar los dust devils no necesitamos tener en cuenta estos parámetros, pero esto lo veremos más adelante. Una vez establecidos los parámetros pulsaremos sobre el botón ejecutar y nos aparecera una gráfica similar a la que ya vimos anteriormente, por ejemplo:

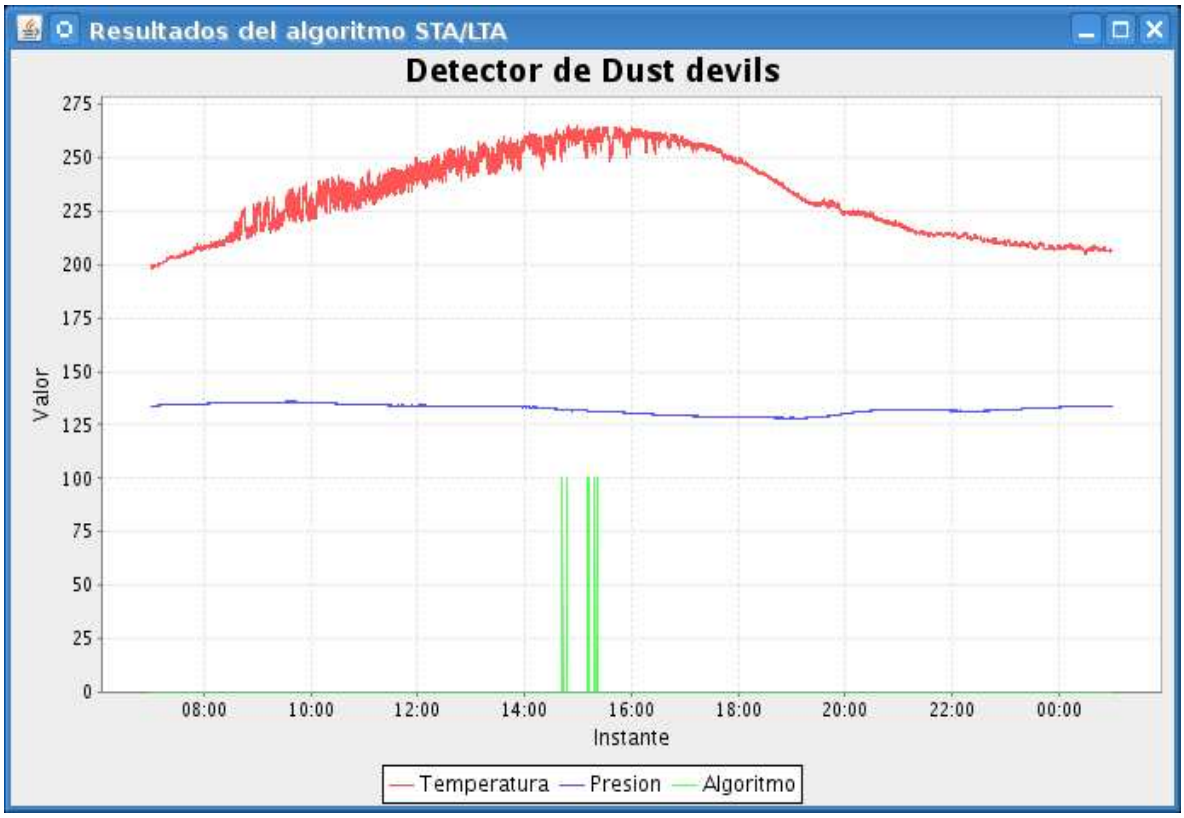


Gráfico. Resultados ejecución.

En rojo tenemos la temperatura de las muestras, correspondientes a cada instante, en azul tenemos la muestras de presión. Vease que las muestras de presión están a escala, con motivo de mejorar su visualización, pues en Marte, la presión atmosférica es muy baja como ya vimos antes.

Finalmente en verde se pueden apreciar los momentos en los que se ha disparado el detector.

Otra opción interesante es la de guardar los datos de las muestras en un archivo binario. Esto es necesario porque la información, el conjunto de muestras, que le enviamos a la FPGA está en ese formato.

Para pasar un conjunto de muestras a un archivo binario simplemente tenemos que pulsar sobre el menú Archivo y seleccionar Guardar Temperatura en binario o Guardar Presión en binario.

6.2. Implementación de algoritmos

Procederemos a mostrar el código fuente de las clases que ejecutan nuestro algoritmo STA/LTA. Antes de entrar a ver el código debemos aclarar un par de conceptos. En este caso hemos utilizado un algoritmo que realiza las predicciones aplicando la media aritmética.

Para evitar confusiones explicaremos aquí lo que entendemos por eventos de subida y eventos de bajada. A diferencia del algoritmo de Allen que solo detecta cuándo se produce un aumento de la actividad sísmica, para nosotros también resulta importante determinar cuando una magnitud está cayendo drásticamente. Concretamente, a la hora de predecir dust devils, debemos detectar que la presión atmosférica ha caído y que la temperatura ha subido.

Por ello nuestro algoritmo STA/LTA debe estar preparado para responder a estos dos tipos de eventos, pues su condición de disparo es distinta. Aclarado este punto, pasemos a ver el código fuente:

```
/**
 * Esta clase es la que realiza los cálculos y procesa las nuevas muestras, es
 * la que ejecuta el algoritmo en sí misma.
 *
 * @author M. Javier Miguel García
 * @author Enrique de Lucas Casamayor
 */

public class AlgoritmoMedias
{
   private GestorMuestras local; //hace la función del módulo STA
   private GestorMuestras global; //hace la función del módulo LTA
   private double thrSubida;
   private double thrBajada;
   private int estado; //indica si salta el disparador


   /**
    * Constructor, crea una instancia de AlgoritmoMedias
    * @param maxMuestrasLocales Número máximo de muestras que recogerá el STA
    * @param maxMuestrasGlobales Número máximo de muestras que recogerá el LTA
    * @param thrSubida Valor límite para considerar un evento de bajada
    * @param thrBajada Valor límite para considerar un evento de subida
    */
   public AlgoritmoMedias(int maxMuestrasLocales, int maxMuestrasGlobales,
         double thrSubida, double thrBajada)
   {
```

```
        local = new GestorMuestras(maxMuestrasLocales);
        global = new GestorMuestras(maxMuestrasGlobales);
        this.thrSubida = thrSubida;
        this.thrBajada = thrBajada;
    }

    /**
     * Procesa un nuevo valor de entrada
     * @param m Muestra a procesar, puede ser de presión, temperatura, etc.
     */
    public void procesarMuestra(Muestra m)
    {
        local.nuevaMuestra(m);
        global.nuevaMuestra(m);

        double division = local.getMedia() / global.getMedia();
            //Realizamos la división entre las medias

        if ( division > thrSubida  || division < thrBajada )
            //Comprobamos si se ha producido algún evento
        {
            if (division > thrSubida) //si corresponde, cambiamos el estado
            {
                estado = 1;
            }
            else
            {
                estado=-1;
            }
        }
        else estado=0;
    }

    /**
     * 0: No ocurre ningún evento
     * 1: Evento de subida
     * -1: Evento de bajada
     * @return Devuelve el estado del algoritmo
     */
    public int getEstado()
    {
        return estado;
    }
}
```

Como habrá podido observar el lector con motivo de hacer el diseño del algoritmo más parecido a la implementación en VHDL, hemos separado la parte lógica que comprueba si el algoritmo debe ser disparado o no, la cual está en la clase que acabamos de ver, de la gestión de las muestras, que pertenece a la clase GestorMuestras.

Esta clase va a simular los módulos STA y LTA que veremos posteriormente en hardware. La función de los mismos es almacenar las muestras recogidas y calcular su media aritmética. Procedemos a exponer el código de dicha clase para mayor claridad:

```
/**
 * Esta clase gestiona y organiza las muestras recogidas
 *
 * @author M. Javier Miguel García
 * @author Enrique de Lucas Casamayor
 */
public class GestorMuestras
{
   private ArrayList<Muestra> muestras;
   private int maxMuestras;
   private double acumulador;

   /**
    * Constructora de la estructura
    *
    * @param max Indica el tamaño máximo del gestor
    */
   public GestorMuestras(int max)
   {
      maxMuestras = max;
      acumulador=0;
      muestras = new ArrayList<Muestra>();
   }

   /**
    * Introduce una nueva muestra y ajusta el valor del acumulador
    * @param m Nueva muestra
    */
   public void nuevaMuestra(Muestra m)
   {
      muestras.add(m);
      acumulador+=m.getValor();
```

```
    if (muestras.size() >= maxMuestras+1) //si hemos llegado al limite..
    {
        acumulador-=muestras.get(0).getValor(); //quitamos el primer elemento
        muestras.remove(0);
    }
}

/**
 *
 * @return Devuelve la media aritmética de las muestras que hay en el sistema
 */
public double getMedia()
{
    if (muestras.size() == 0) return 0;
    return acumulador / muestras.size();
}

 /**
 *
 * @return Devuelve la media de los cuadrados de las muestras recogidas
 */
public double getMediaCuadratica()
{
    double resul=0;
    for (int i = 0; i<muestras.size();i++)
    {
        resul += Math.pow(muestras.get(i).getValor(),2);
    }
    resul /= ((double) muestras.size());
    return resul;
}

/**
 *
 * @return Devuelve la varianza de las muestras que hay en el sistema
 */
public double getVarianza()
{
    double media = this.getMedia();
    double resul=0;
    for (int i = 0; i<muestras.size();i++)
    {
        resul += Math.pow(muestras.get(i).getValor() - media,2);
    }
```

```
        resul /= ((double) muestras.size());
        return resul;
    }
}
```

6.3. Detector de dust devils

Para el detector de dust devils vamos a utilizar dos módulos que ejecuten el algoritmo STA/LTA que vimos antes. Como ya vimos antes los dust devils están asociados a variaciones en la presión y la temperatura, concretamente los parámetros que los caracterizan son una bajada de la presión atmosférica y una subida de la temperatura. Si detectamos que se dan estas condiciones entonces podemos intuir que se está produciendo un evento. A continuación vamos a exponer el código fuente del detector:

```
/**
 *
 * @author M. Javier Miguel García
 * @author Enrique de Lucas Casamayor
 */
public class DetectorDustDevils
{
    private AlgoritmoMedias temperatura;
    private AlgoritmoMedias presion;
    private int estado;

    /**
     * Constructora del detector de dust devils.
     *
     * @param STAtemp Número de muestras locales para la temperatura
     * @param LTAtemp Número de muestras globales para la temperatura
     * @param THRUPtemp Valor límite de subida para la temperatura
     * @param THRDOWNtemp Valor límite de bajada para la temperatura
     * @param STApres Número de muestras locales para la presión
     * @param LTApres Número de muestras globales para la presión
     * @param THRUPpres Valor límite de subida para la presión
     * @param THRDOWNpres Valor límite de bajada para la presión
     */
    public DetectorDustDevils(
                    int STAtemp, int LTAtemp, double THRUPtemp, double THRDOWNtemp,
                     int STApres, int LTApres, double THRUPpres, double THRDOWNpres)
```

```
{
      temperatura = new AlgoritmoMedias(STAtemp,LTAtemp,THRUPtemp,THRDOWNtemp);
         //Iniciamos el detector para la temperatura
      presion = new AlgoritmoMedias(STApres,LTApres,THRUPpres,THRDOWNpres);
         //Y el de la presión
   }

   /**
    * Procesamos una nueva muestra
    * @param m Muestra a procesar
    */
   public void procesarMuestra(MuestraPF m)
   {
      //Obtenemos una muestra de temperatura y otra de presión
      Muestra estatemp=new Muestra(m.getTemperaturaAlto().getValor());
      Muestra estapres=new Muestra(m.getPresionGlobal().getValor());

      //Le pasamos las muestras a los detectores correspondientes
      temperatura.procesarMuestra(estatemp);
      presion.procesarMuestra(estapres);

      /*
       * Comprobamos las condiciones de disparo, es decir que se hayan
       * disparado ambos detectores, si es así cambiamos el estado del
       * detector.
       *
       */
      if (temperatura.getEstado() == 1 && presion.getEstado() == -1)
      {
         estado=1;
      }
      else
      {
         estado=0;
      }
   }

   /**
    * Devuelve 1 si se ha disparado el detector
    * Devuelve 0 si no se ha disparado
    * @return Estado del detector
    */
```

```
    public int getEstado()
    {
        return estado;
    }
}
```

6.4. Tiempos de ejecucción

Para comprobar cual es el tiempo medio de ejecución de los algoritmos hemos procedido a ejecutarlos sobre un ordenador con las siguientes características:

Procesador	Intel Q8200 Core2 Quad, con una frecuencia de 2,33GHz
Memoria RAM	4GB DDR3 a una frecuencia de 1333MHz
Tarjeta gráfica	Nvida Geforce 9800GTX+ 512MB de RAM DDR3
Placa base	Asus P5QC
Sistema operativo	Ubuntu 10.04LTS

Algoritmo STA/LTA de medias

Parámetro	Valor
Muestras STA	32
Muestras LTA	128
THR de bajada	1.021
THR de subida	0.798

Nº de muestras	Tiempo de ejecución (milisegundos)	Muestras / Segundo
3348	69	48521.739130434784
33	7	4714.285714285715
31	5	6200.0
897	21	42714.28571428572
798	20	39900.0
16637	180	92427.77777777778
13865	129	107480.62015503876
13951	149	93630.87248322148
13912	128	108687.5
12793	141	90730.4964539007

En total se procesaron: **76265**
La media ponderada de muestras por segundo es de: **94931.00704**

Algoritmo STA/LTA de medias para Dust Devils

Parámetro	Presión	Temperatura
Muestras STA	8	4
Muestras LTA	1024	32
THR de bajada	0.994	-
THR de subida	-	1.015

Nº de muestras	Tiempo de ejecución (milisegundos)	Muestras / Segundo
842	23	36608.69565217391
33	5	6600.0
31	5	6200.0
897	29	30931.03448275862
798	26	30692.30769230769
16637	159	104635.22012578616
13865	139	99748.20143884892
13951	148	94263.51351351352
13912	151	92132.45033112583
12793	169	75698.22485207101

En total se procesaron: **73759 muestras**
La media ponderada de muestras por segundo fue de: **91819.59363**

Por la forma del algoritmo, sabemos que se ha ejecutado en el mismo hilo, por lo tanto, a pesar de que el ordenador posee 4 núcleos, solo se ha ejecutado en uno de ellos. Hay que mencionar que no se pierde tiempo en operaciones de entrada y salida pues cuando se comienza a procesar el algoritmo los datos ya están cargados en la memoria principal. No obstante debemos advertir de que los cálculos realizados por el algoritmo son en punto flotante, mientras que en hardware nuestro sistema está optimizado para trabajar en coma fija.

7. Sistema Hardware

7.1. Introducción

Para la implementación en hardware se ha usado el lenguaje de descripción de hardware VHDL orientando el diseño a una fpga Virtex II pro Xc2vp30, la cual tiene todos los elementos que necesitamos.

Nos hemos decidido por realizar un diseño íntegro en VHDL en vez de usar el procesador powerPC ya que buscamos minimizar el número de ciclos y el consumo, pudiendo reducir ambos con un diseño específico.

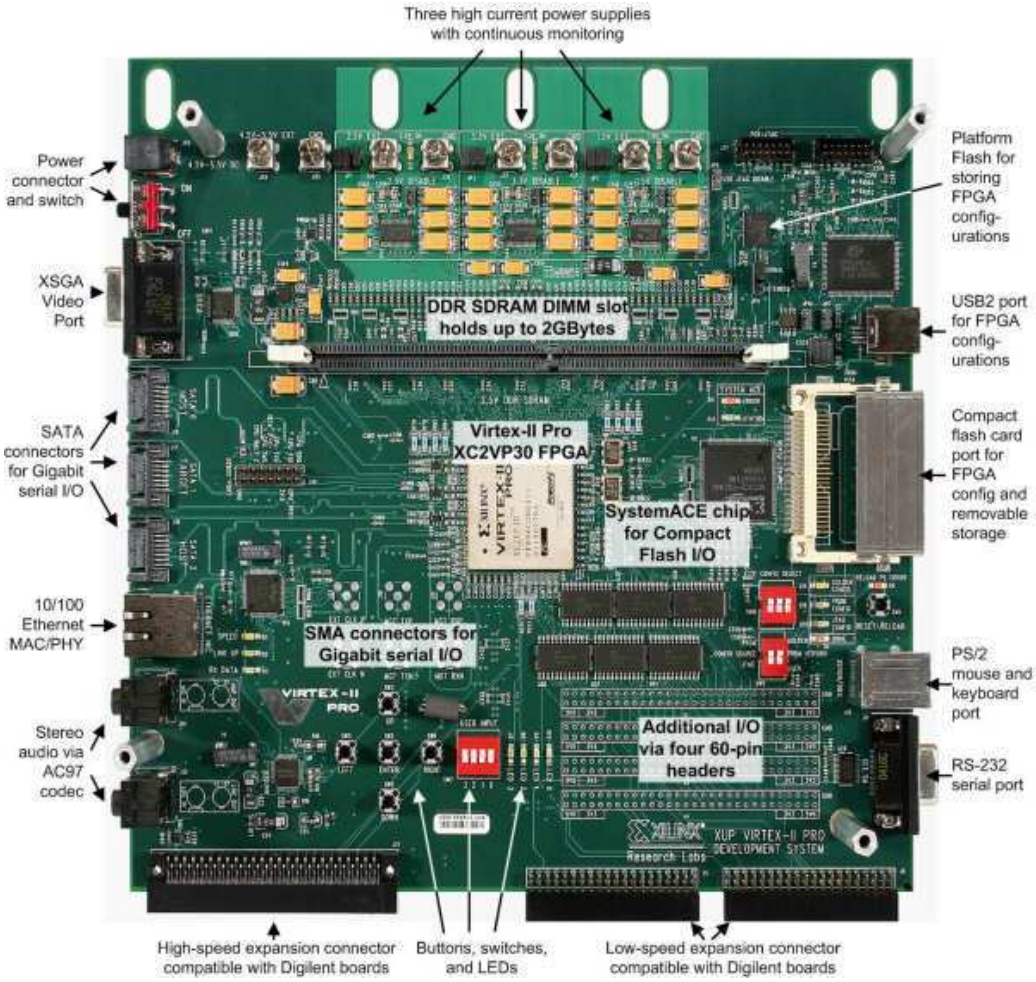


Figura 6. FPGA Virtex II Pro XC2VP30.

Se ha procedido a diseñar en primer lugar una estrategia de comunicación con la FPGA que nos permitiera enviar largas series de datos y recoger los resultados en tiempo real. Esta fase ha llevado un tiempo representativo del tiempo de implementación del hardware. Seguidamente se han implementado en vhdl los algoritmos probados en la aplicación software y se ha comprobado su corrección.

7.2. Comunicación. Formato de trama

La comunicación se ha llevado a cabo mediante el puerto serie usando el protocolo RS232 de 8 bits de datos a 9600 baudios de velocidad. La parte de comunicación software la ha proporcionado el programa de licencia gratuita RealTerm y la parte hardware ha sido implementada en vhdl. Si bien el punto 7 detalla el sistema hardware hemos creído adecuado incluir en el toda la comunicación con la FPGA, incluida la parte software.

El por qué de esta estrategia de comunicación está en que lo importante es reducir el tiempo de ejecución de nuestros algoritmos para un dato y no tener un sistema que ejecuta un conjunto de datos en el mínimo tiempo posible, ya que el sistema realmente va a recibir datos de los sensores con una frecuencia de 1Hz en el mejor de los casos. Por ello una simulación de la comunicación entre sensores y algoritmo como la presentada resulta suficiente.

Protocolo RS232

La siguiente figura muestra el envío de un carácter usando el protocolo RS-232 de 8 bits de datos.

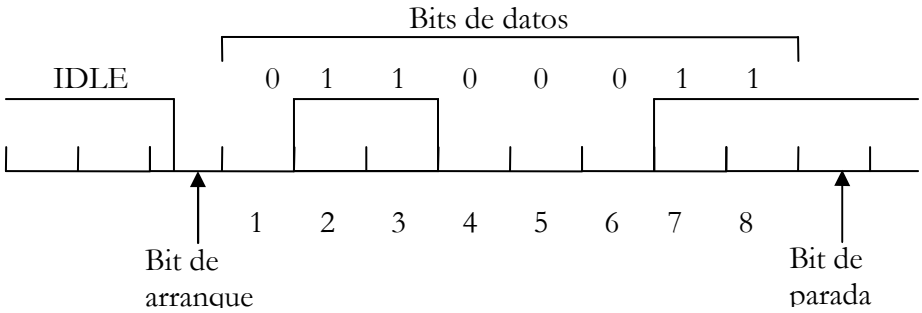


Figura 7. Protocolo RS-232 de 8 bits de datos.

La línea de transmisión se mantiene a uno mientras no se envían datos, esto es, está de estado de parada. Cuando se quiere transmitir se pone a cero la señal durante el tiempo denominado “bit de arranque” y se inicia la transmisión de los datos. Se conviene un bit final denominado “bit de parada”. Seguidamente la línea entra nuevamente en estado de parada o comienza una nueva transmisión.

7.2.1. Comunicación software

Para la parte de comunicación del pc hemos usado el software de libre distribución RealTerm y lo hemos configurado consecuentemente para que envíe y reciba datos a/de la FPGA.

RealTerm

Configuración del puerto serie:

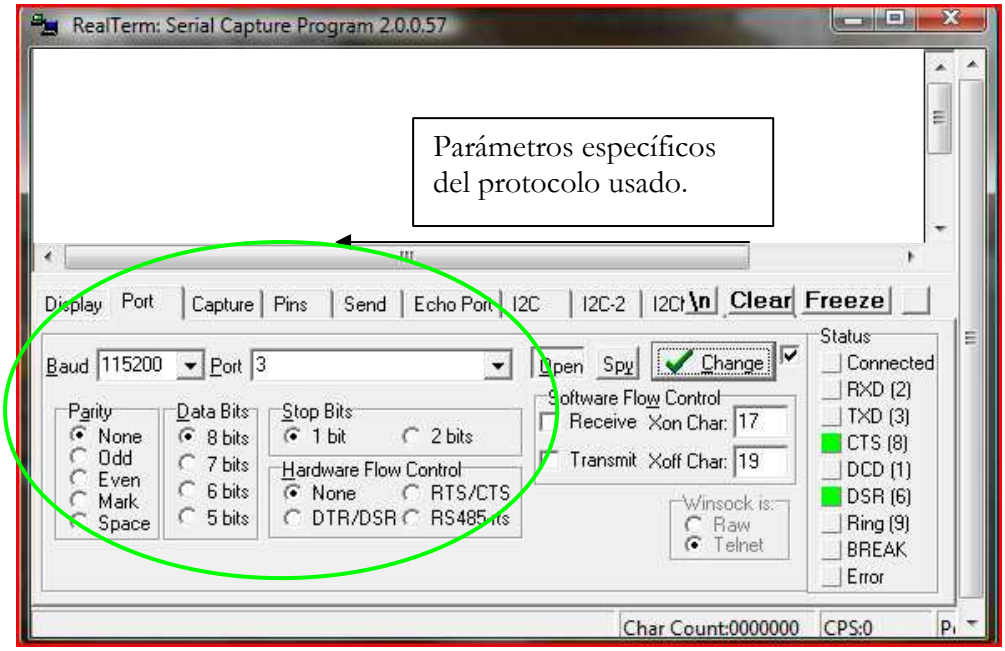


Figura 8. Configuración puerto serie.

Envío de datos:

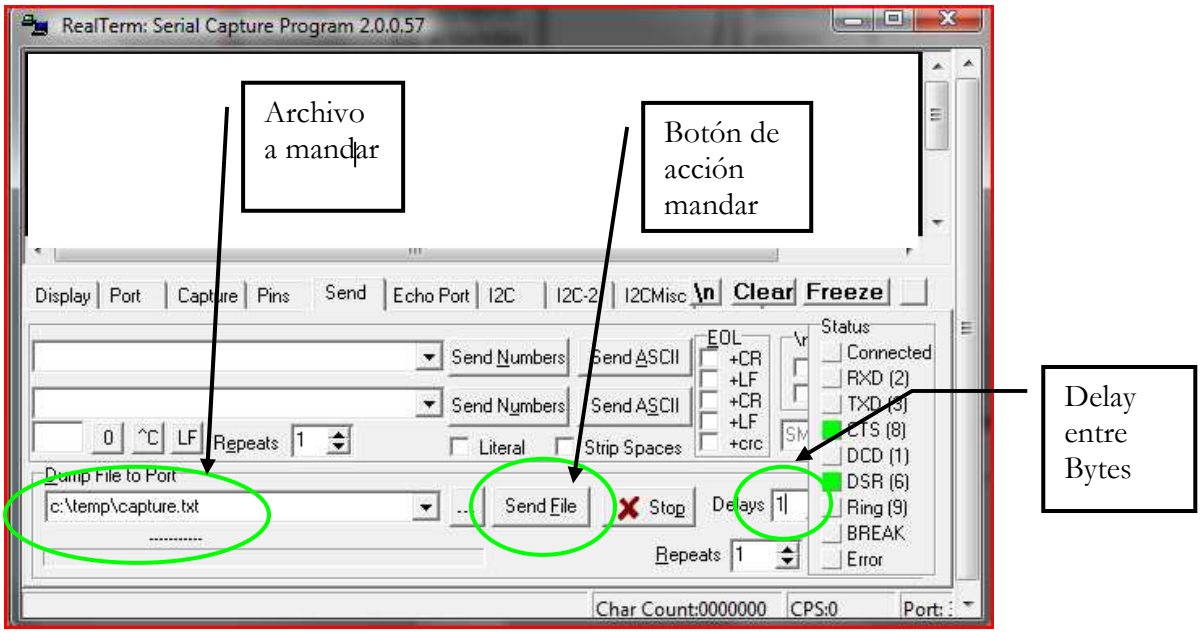


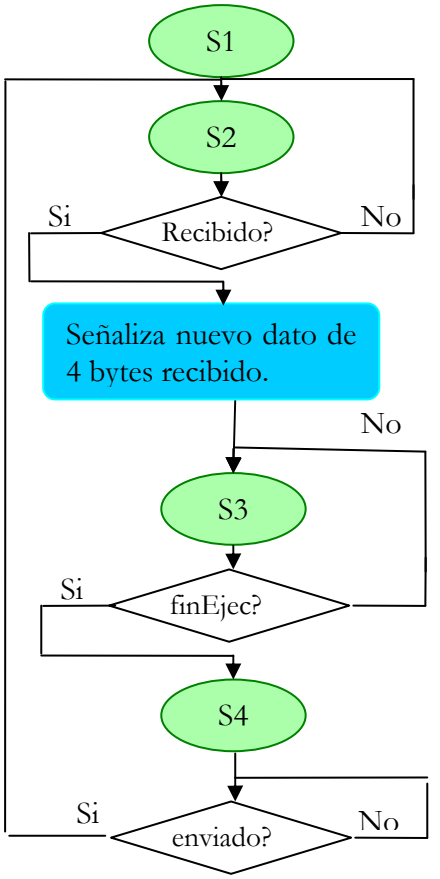
Figura 9. Configuración envío de datos.

7.2.2. Comunicación hardware

Para los datos de ejecución del proyecto se ha convenido que con 32 bits sería suficiente para enviar las mediciones y recibir a su vez información de ejecución del sistema. Por lo tanto cada 4 comunicaciones del protocolo RS-232 (cada 4 bytes) tenemos un dato completo enviado o recibido (datos de 32 bits). Además cabe decir que usamos comunicación full-duplex (enviamos y transmitimos datos simultáneamente).

Diagrama de flujo comunicación FPGA

Estado	Cometido
S1	Reset
S2	Recibe dato
S3	Espera fin ejecución
S4	Envía dato



El sistema pasa del estado de reset a un estado de recepción de datos. Cada 4 bytes recibidos provee el dato al modulo que ejecuta el algoritmo y lo señaliza. En ese momento el sistema entra en un estado de espera hasta que finaliza la ejecución del algoritmo. Una vez se le comunica al sistema que la ejecución ha terminado y que dispone de la respuesta, se procede a enviar los datos de ejecución, en este caso también de tamaño 4 bytes.

7.2.3. Formato de trama

Tenemos tramas de datos diferentes para envío y recepción de datos desde el pc.

Envío

Consiste en el dato procedente de la nueva medición y que será tratado en el sistema. Este dato será un dato de tipo entero en formato binario.

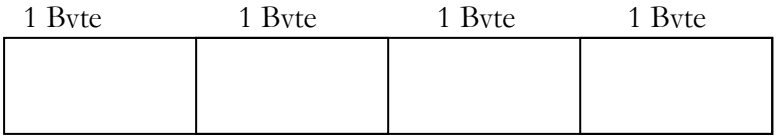


Figura 10. Trama datos enviada.

Recepción

Recibimos en primer lugar el resultado del algoritmo para el último dato recibido por la FPGA, esto es, si se ha producido o no un disparo. Además recibimos el valor del cociente usado para la condición de disparo y el número de muestra que ha producido este resultado.

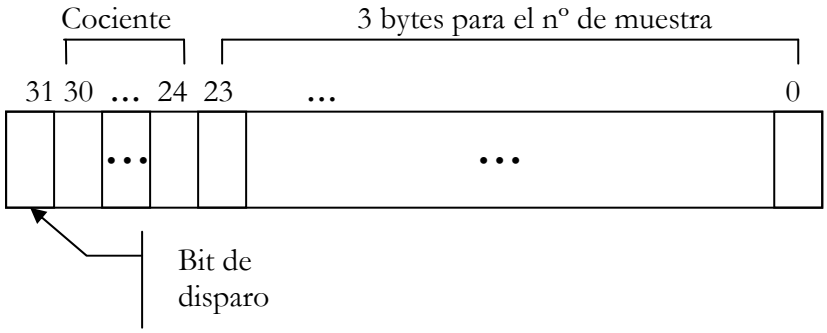


Figura 11. Trama de datos recibidos.

7.3. Algoritmo STA/LTA (medias)

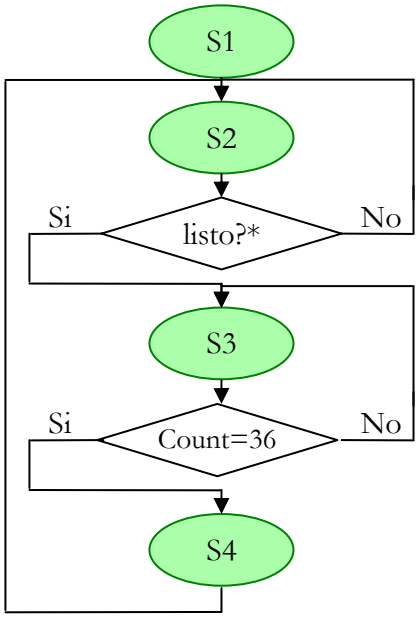
La evolución del algoritmo completo viene determinada por el diagrama de flujo del componente *Bloque principal* así como de los componentes *ModuloSTA* y *ModuloLTA* además de la señal que produce el software de comunicación que señaliza cuando se dispone de un nuevo dato a procesar.

- El componente Bloque principal pasa de un estado de reset a un estado de espera.
- La señal de nuevo dato a procesar se pasa directamente a los módulos ModuloLTA y ModuloSTA. Se mantiene el estado actual hasta que se recibe la notificación de que ambos módulos han acabado su ejecución para el nuevo dato recibido.

- Se pasa entonces al último paso de ejecución del algoritmo Allen Medias en la que se calcula el cociente entre los resultados de ModuloSTA y ModuloLTA y se decide si se notifica un evento o no. Se multiplica por cien el resultado de ModuloSTA para obtener un valor con precisión de dos cifras en la división a realizar. Además se le comunica al componente de comunicación que se ha acabado la ejecución del algoritmo mediante la señal finEjecucion y se le pasan los datos sobre la ejecución.

7.3.1. Diagrama de flujo

Estado	Cometido
S1	Reset
S2	Espera ejecución STA y LTA
S3	Calcula cociente
S4	Señaliza fin de ejecución



7.3.2. Diagramas de bloques

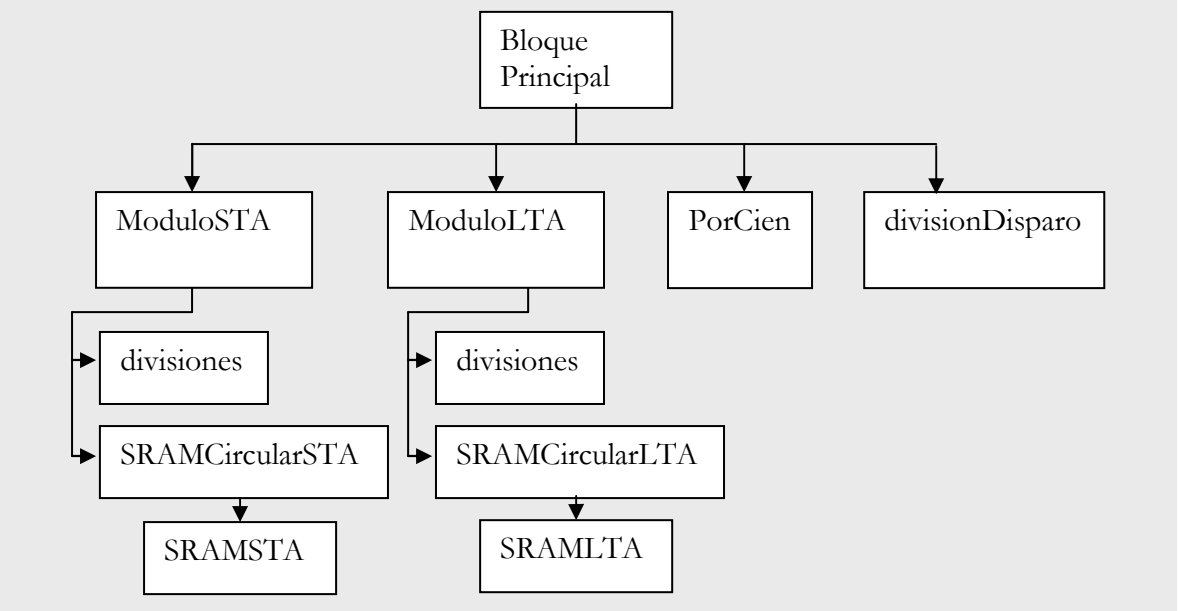
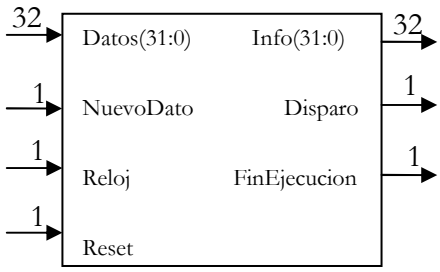


Figura 12. Diagrama de bloques del algoritmo.

7.3.2.1. Bloque STA/LTA (medias)

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
Info	Salida	Información de ejecución	32
FinEjecución	Salida	Señaliza fin de ejecución	1
Disparo	Salida	Resultado de la ejecución	1



Interconexionado

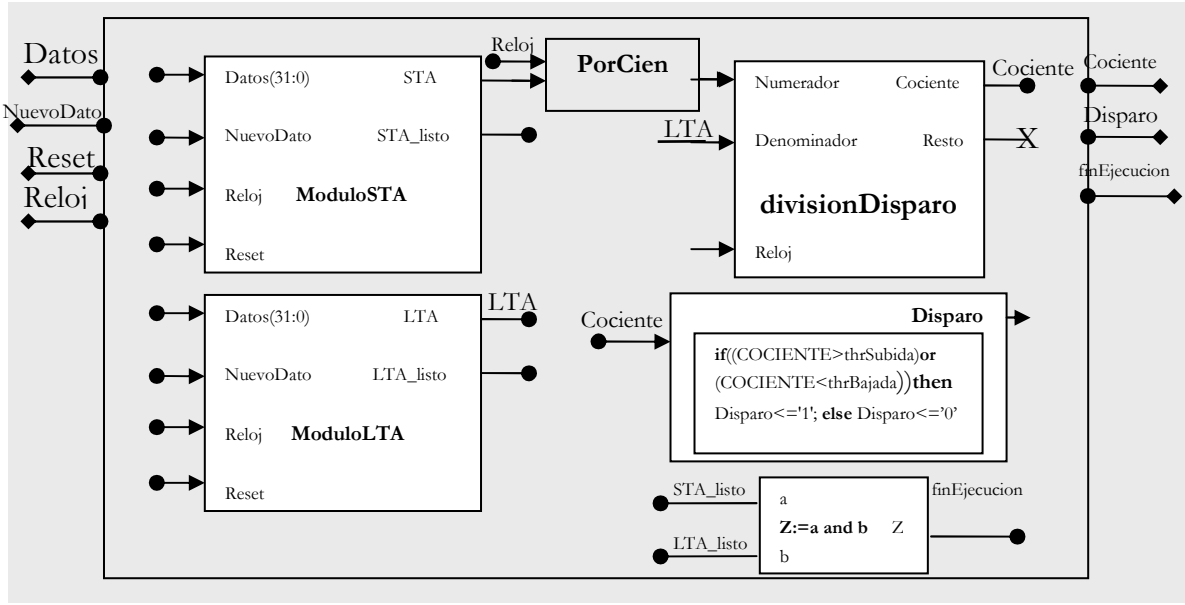


Figura 13. Interconexión de los bloques.

7.3.2.2. Bloques divisionDisparo y divisiones

Dado que los datos que vamos a manejar están normalizados y tenemos un número igual de decimales una vez tratados hemos decidido usar coma fija para las divisiones. Se ha usado el *IP divider generador* de Xilinx para obtener un divisor de complemento a dos, en coma fija y con un ancho de dividendo, divisor y resto de 32 bits.

La latencia de un divisor de estas carácterísticas es de M+4 ciclos, siendo M el número de bits del dividendo. Por lo tanto este módulo tendrá una latencia de 36 ciclos.

7.3.2.3. Bloque ModuloSTA

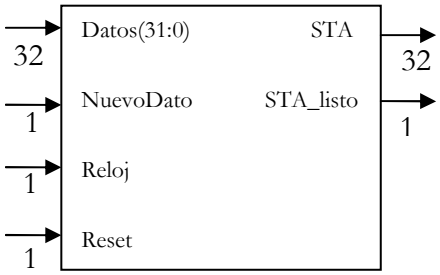
Este bloque hace la media de los datos que se le proporcionan por la entrada. En cada ciclo de ejecución del diagrama de flujo se añade un dato, se calcula la media con ese nuevo dato y se devuelve el valor indicándolo con la señal STA_listo.

Para hacer el cálculo de la media se mantiene un registro *Suma* que contiene la suma de todos los valores almacenados en la ram del bloque además del número de datos almacenados en dicha ram (*Ocupacion*). Una vez se ha añadido un nuevo dato se hace la división entre *Suma* y *Ocupacion* obteniendo así la media aritmética.

Mostramos las entradas y salidas del bloque, el interconexionado interno y el diagrama de flujo correspondientes.

Entradas y salidas

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
STA	Salida	Media valores almacenados en la ram circular	32
STA_listo	Salida	Señaliza fin de proceso del nuevo dato	1



Interconexionado

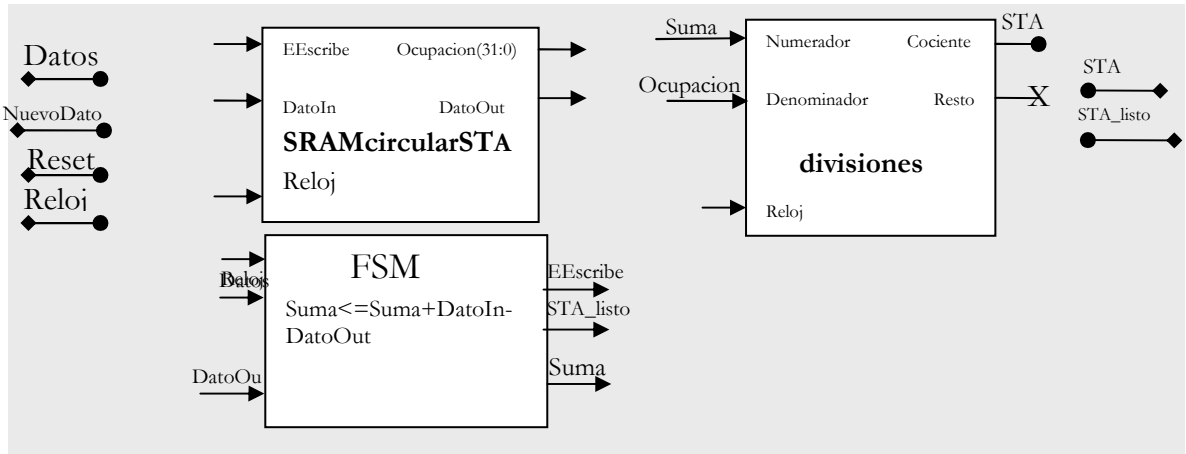


Figura 14. Interconexión de los bloques.

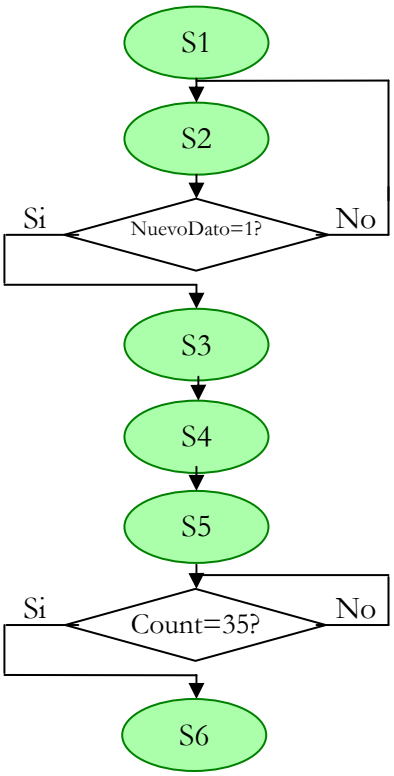


Diagrama de flujo

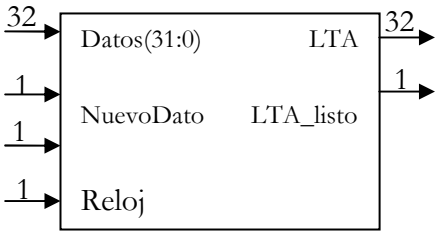
Estado	Cometido
S1	Reset
S2	Espera de nuevo dato
S3	Lectura dato y recálculo de Suma
S4	Grabado del dato en ram. Actualización Ocupación
S5	Cálculo de la media.
S6	Señala fin de procesado del nuevo dato.

El sistema pasa del estado de reset (S1) al estado de espera de nuevos datos (S2) y se mantiene en dicho estado hasta que se le notifica la disponibilidad de un nuevo dato a añadir (NuevoDato=1). Seguidamente se recalcula el valor de *Suma* y *Ocupacion* (S3 y S4). Ahora se hace la división entre esos valores y después de 35 ciclos obtenemos la media (S5) y señalizamos el fin del proceso (S6).

7.3.2.4. Bloque ModuloLTA

Su funcionamiento es idéntico al bloque ModuloSTA, solo se diferencian en el tamaño de la memoria ram circular que contienen.

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
LTA	Salida	Media valores almacenados en la ram circular	32
LTA_listo	Salida	Señaliza fin de proceso del nuevo dato	1



Interconexionado

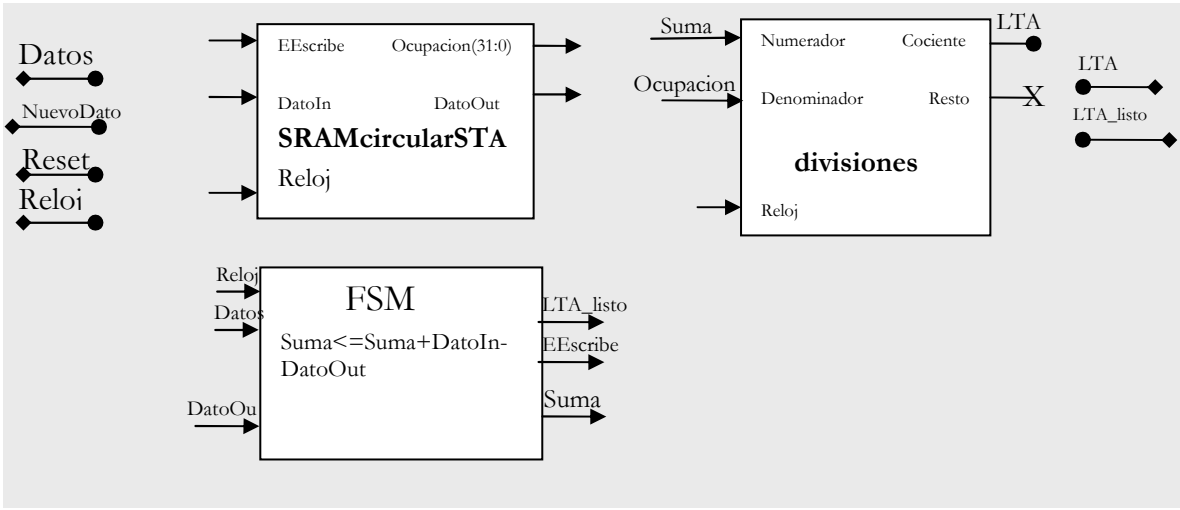
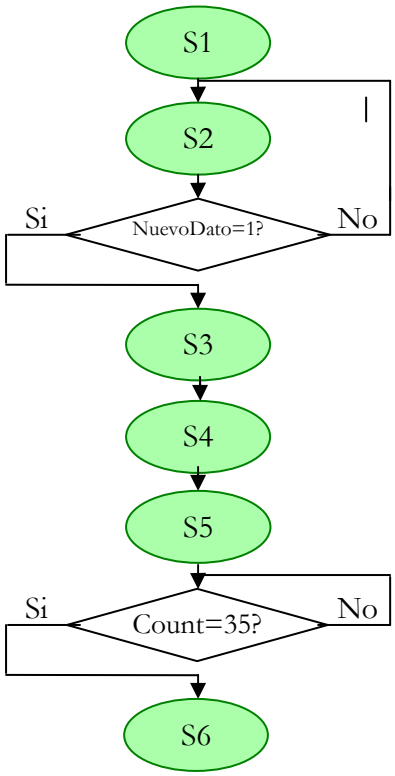


Figura 15. Interconexión de los bloques.

Diagrama de flujo



Estado	Cometido
S1	Reset
S2	Espera de nuevo dato
S3	Lectura dato y recálculo de Suma
S4	Grabado del dato nuevo. Actualización Ocupación
S5	Cálculo de la media.
S6	Señala fin de procesado del nuevo dato.

7.3.2.5. Bloques SRAMcircular STA y LTA

Implementan acceso circular sobre un bloque ramSTA y ramLTA respectivamente.

Señal	Tipo	Objetivo	Bits
Ocupacion	Entrada	Datos ocupados en la ram	32(*)
DatoIn	Entrada	Entrada de datos	32
EEscribe	Entrada	Habilita escritura	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
DatoOut	Salida	Valor apuntado por la señal Ocupacion	32

(*)Bastaría con x bits, siendo (2^x) el tamaño de la ram. ISE optimizará los bits no usados.

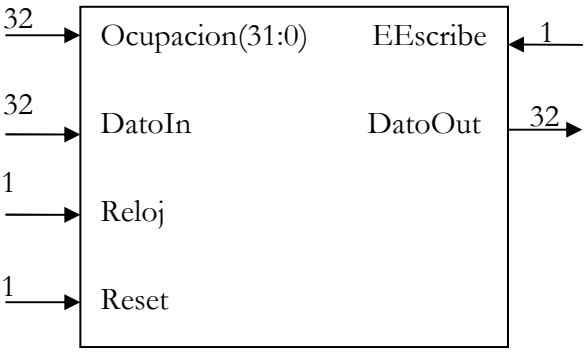
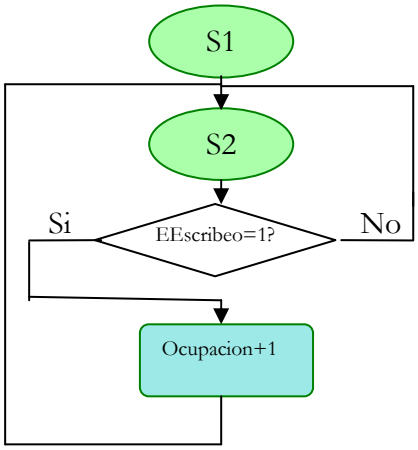


Diagrama de flujo

Estado	Cometido
S1	Reset
S2	Ejecución



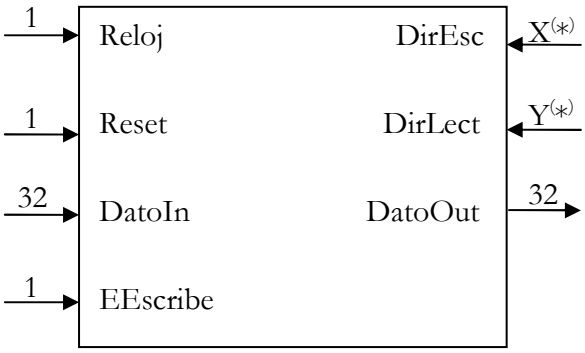
Se mantienen los puertos de escritura y lectura de la ram apuntados por un puntero (*Puntero*) que empieza apuntando a la posición 0. De esta manera cada vez que escribamos un nuevo dato tenemos en la salida el valor que va a ser sobrescrito para usarlo si es necesario. Una vez actualizado el valor avanzamos el puntero. Si el puntero ha sobrepasado el tamaño de la ram, este vuelve a tomar el valor 0.

7.3.2.6. Bloque SRAM

Implementa una ram de acceso síncrono con un puerto de lectura (*DatoOut*) y otro de escritura (*DatoIn*) independientes. La lectura siempre está habilitada y para habilitar la escritura tenemos la señal *EEscribe*.

Se ha elegido usar ram de acceso síncrono en vez de arrays de vhdl para que ISE infiera que queremos usar las blockram que tiene la fpga.

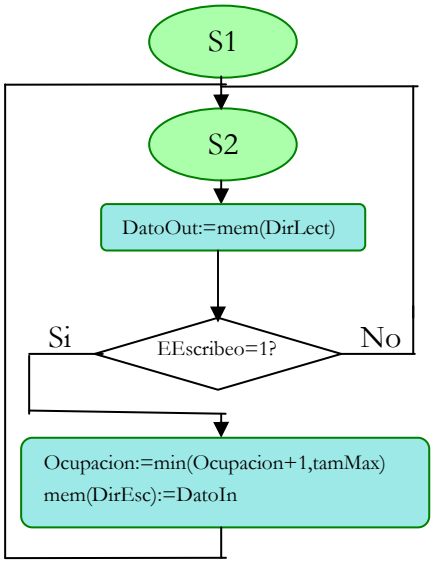
Señal	Tipo	Objetivo	Bits
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
DatoIn	Entrada	Entrada de datos	32
EEscribe	Entrada	Habilita escritura	1
DirEsc	Entrada	Dirección de escritura de DatoIn	
DirLect	Entrada	Posición de lectura para mostrar en DatoOut	
DatoOut	Salida	Valor apuntado por la señal DirLect	32



(*)X e Y son ambas del mismo tamaño, pero dicho tamaño será distinto en la ram de ModuloSTA y la de ModuloLTA. Siempre será mayor en ModuloLTA.

Diagrama de flujo

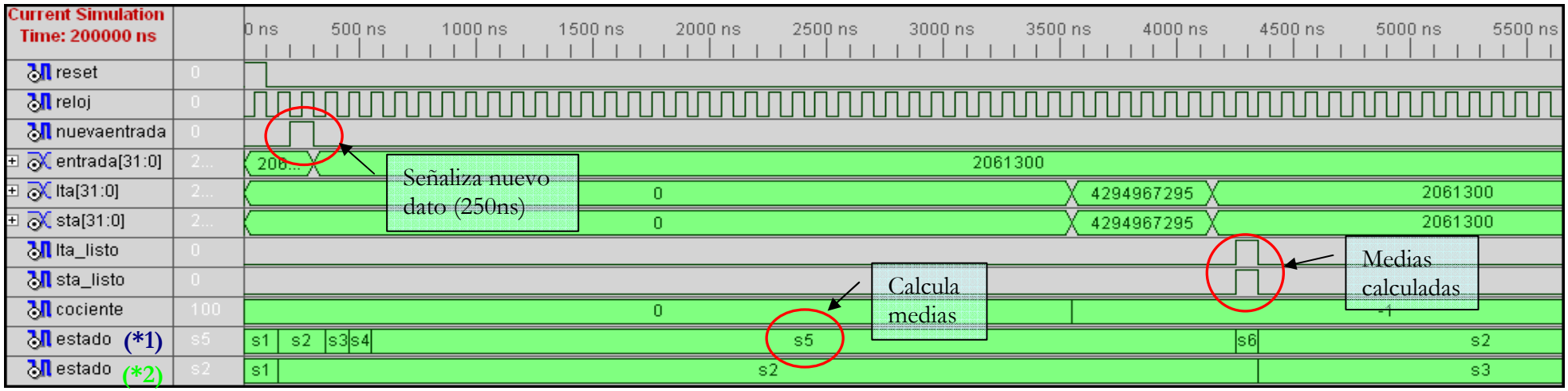
Estado	Cometido
S1	Reset
S2	Ejecución



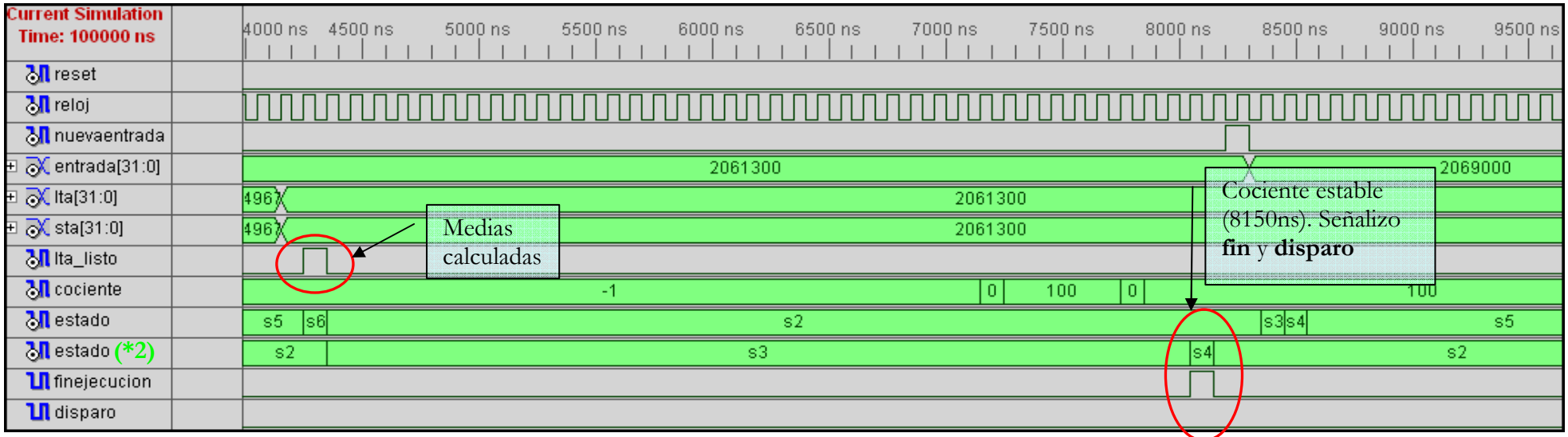
7.3.3. Ciclos de ejecución

La ejecución del sistema se produce en dos pasos principales:

- Cálculo de las medias STA y LTA de manera paralela para cada nuevo dato:



- Cálculo del cociente STA/LTA y evaluación de la condición de salto:



(*1)Estado para los módulos STA y LTA. **(*2)** Estado del bloque principal

En total el número de ciclos de ejecución del sistema desde que se recibe la señal de *nuevaentrada* hasta el siguiente ciclo al que se produce la señal *finejecucion* es de:

$$(T_{fin}-T_{inicio})/T_{ciclo}$$
(8150 ns-250 ns)/100 ns = 79 ciclos

El número de ciclos del algoritmo se debe en su mayoría a los ciclos de latencia de sendas divisiones en los módulos de medias y el módulo principal (al calcular el cociente).

Por lo tanto si pudiéramos eliminar una de estas divisiones con desplazamientos a la derecha tendríamos una disminución significativa en el número de ciclos.

La división STA/LTA no puede ser eliminada ya que LTA no será potencia de dos en general. Sin embargo si el número de muestras almacenadas en los módulos de medias fueran potencia de dos si podríamos eliminar la división para calcular las medias. Esta alternativa se desarrolla más adelante.

7.3.4. Ocupación en la FPGA

A continuación presentamos las estadísticas de implementación del algoritmo en la FPGA para un número de muestras de 64 y 128 para STA y LTA respectivamente.

```
=========================================================
*                      Final Report                     *
=========================================================
Final Results
RTL Top Level Output File Name     : Allen.ngr
Top Level Output File Name         : Allen
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO
Design Statistics
# IOs                              : 101
Cell Usage :
# BELS                             : 12519
#      GND                         : 5
#      INV                         : 101
#      LUT1                        : 113
#      LUT2                        : 564
```

```
#      LUT3                 : 3131
#      LUT4                 : 432
#      MULT_AND              : 99
#      MUXCY                : 4055
#      MUXF5                : 3
#      VCC                  : 4
#      XORCY                : 4012
# FlipFlops/Latches         : 7427
#      FD                   : 7147
#      FDC                  : 133
#      FDCE                 : 141
#      FDE                  : 6
# RAMS                      : 2
#      RAMB16_S36_S36       : 2
# Shift Registers           : 291
#      SRL16                : 192
#      SRL16E               : 12
#      SRLC16               : 87
# Clock Buffers             : 2
#      BUFG                 : 1
#      BUFGP                : 1
# IO Buffers                : 68
#      IBUF                 : 34
#      OBUF                 : 34
=========================================================================
                        Device utilization summary:
                           ---------------------------
Selected Device : 2vp30ff896-7

 Number of Slices:              3981  out of  13696    29%
 Number of Slice Flip Flops:    7395  out of  27392    26%
 Number of 4 input LUTs:        4632  out of  27392    16%
    Number used as logic:       4341
    Number used as Shift registers:  291
 Number of IOs:                 101
 Number of bonded IOBs:           69  out of    556    12%
    IOB Flip Flops:             32
 Number of BRAMs:                 2  out of    136     1%
 Number of GCLKs:                 2  out of     16    12%
```

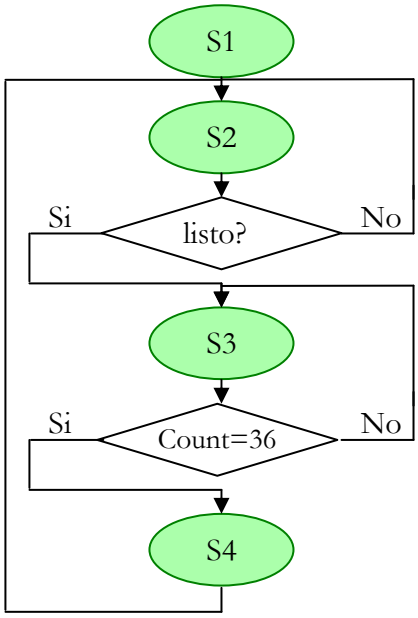
7.4. Optimización Algoritmo STA/LTA (medias)

Como hemos dicho en el apartado anterior si ceñimos el número de muestras almacenadas en los módulos *ModuloSTA* y *ModuloLTA* a una potencia de dos podemos eliminar la división para calcular la media sustituyéndola por un desplazamiento a la derecha. Dicho desplazamiento será de x posiciones, siendo $x=\log_2(nº\ muestras)$.

El funcionamiento del algoritmo es similar al del apartado anterior siendo la única diferencia de tipo estructural.

7.4.1. Diagrama de flujo

Estado	Cometido
S1	Reset
S2	Espera ejecución STA y LTA
S3	Calcula cociente
S4	Señaliza fin de ejecución



7.4.2. Diagramas de bloques

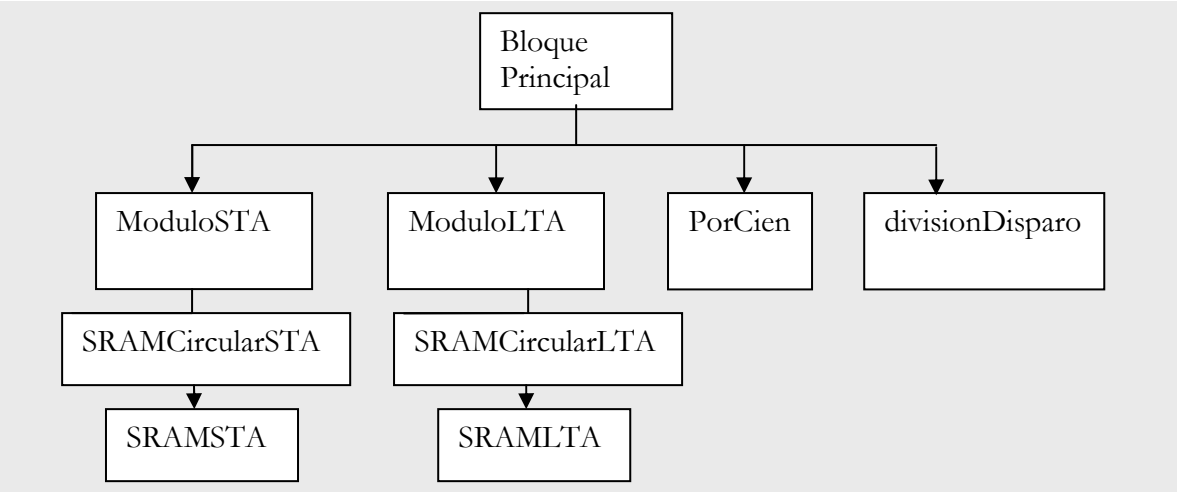


Figura 16. Diagrama de bloques del algoritmo.

Los únicos bloques que presentan variaciones significativas en su funcionamiento respecto al algoritmo anterior son los bloques que calculan las medias por lo que son los únicos explicados.

7.4.2.1. Bloques ModuloSTA y ModuloLTA

En *DesplazamientoDerecha* solo se produce desplazamiento una vez que se ha llenado la SRAMcircular. Será necesario un tiempo de espera para empezar a recibir resultados que consistirá en lo que tarden en llenarse los módulos de muestras. Para ello se usa Ocupacion y solo desde el momento que está llena la ram empezamos a calcular las medias con cada nuevo dato recibido.

Interconexionado

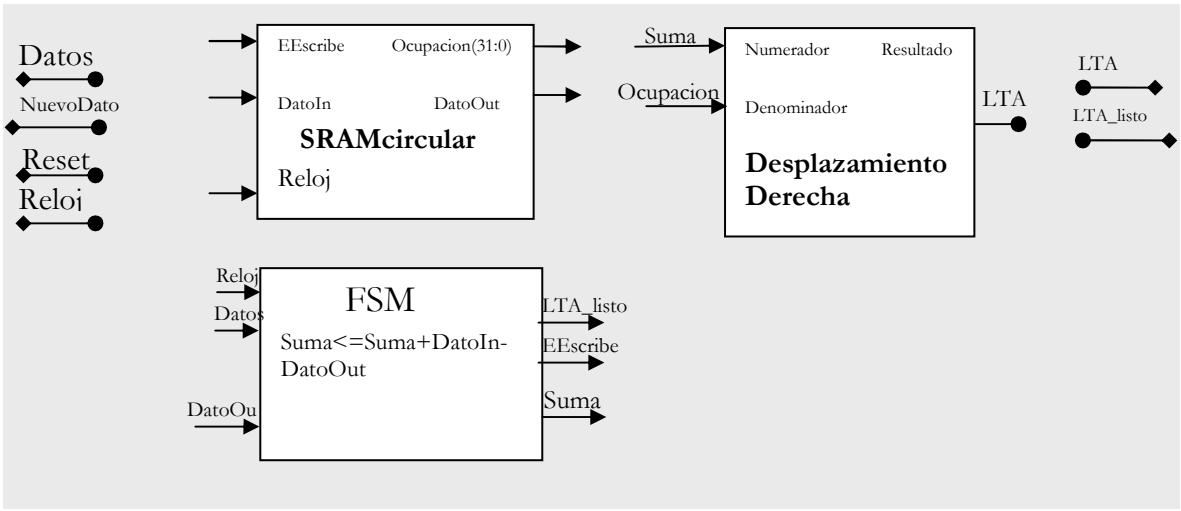
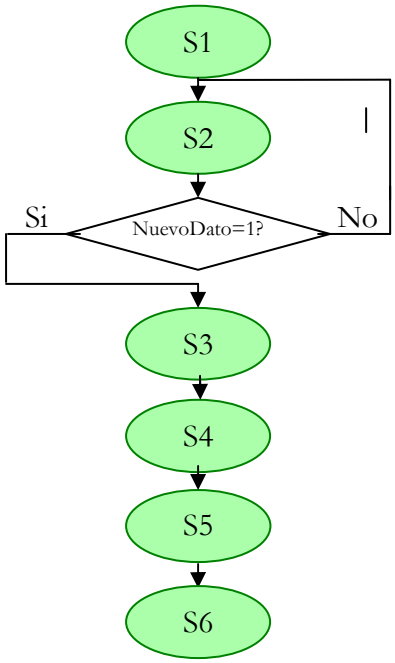


Figura 17. Interconexión de los bloques.

Diagrama de flujo



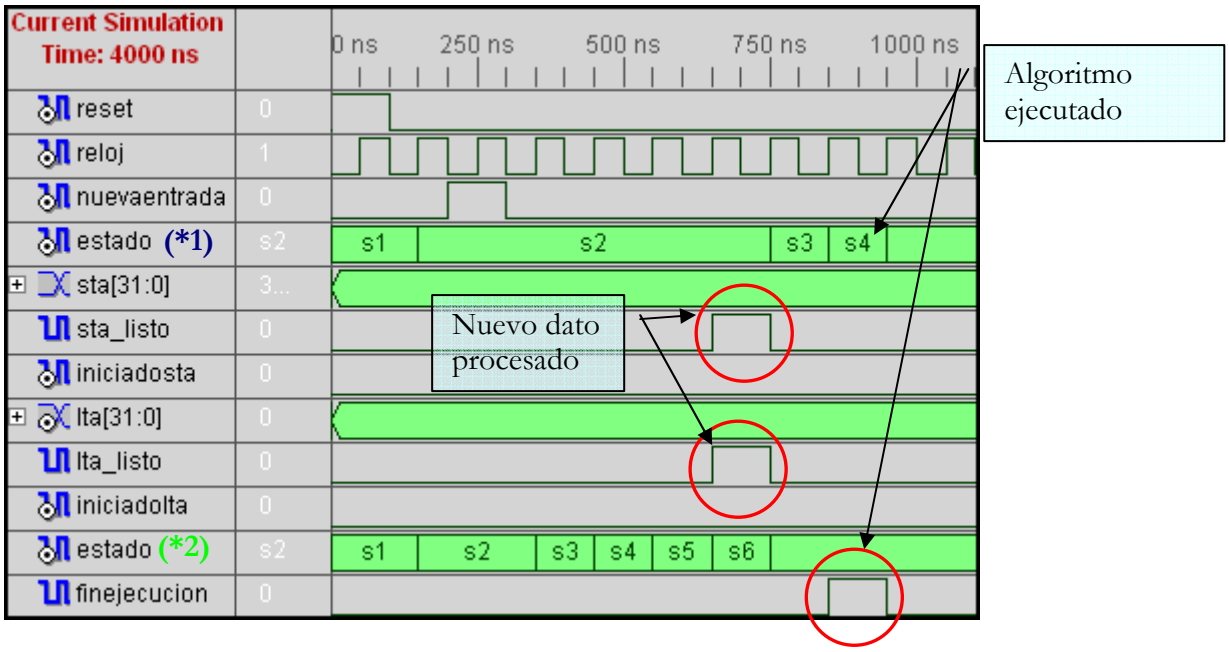
Estado	Cometido
S1	Reset
S2	Espera de nuevo dato
S3	Lectura dato y recálculo de Suma
S4	Grabado del dato nuevo. Actualización Ocupación
S5	Cálculo de la media (Desplazamiento).
S6	Señala fin de procesado del nuevo dato.

7.4.3. Ciclos de ejecución

Este algoritmo requiere un tiempo de inicio para empezar a devolver resultados. Esto se debe a que no comenzamos a calcular el cociente entre STA y LTA hasta que no tenemos la primera de las medias de cada tipo, y esto se produce cuando se ha mandado a procesar el número de muestras que puede almacenar el módulo ModuloLTA, que siempre tendrá mayor capacidad. En este tiempo de inicio el tiempo de ejecución del algoritmo desde que recibimos notificación de nuevo dato es:

$$(T_{fin}-T_{inicio})/T_{ciclo}$$

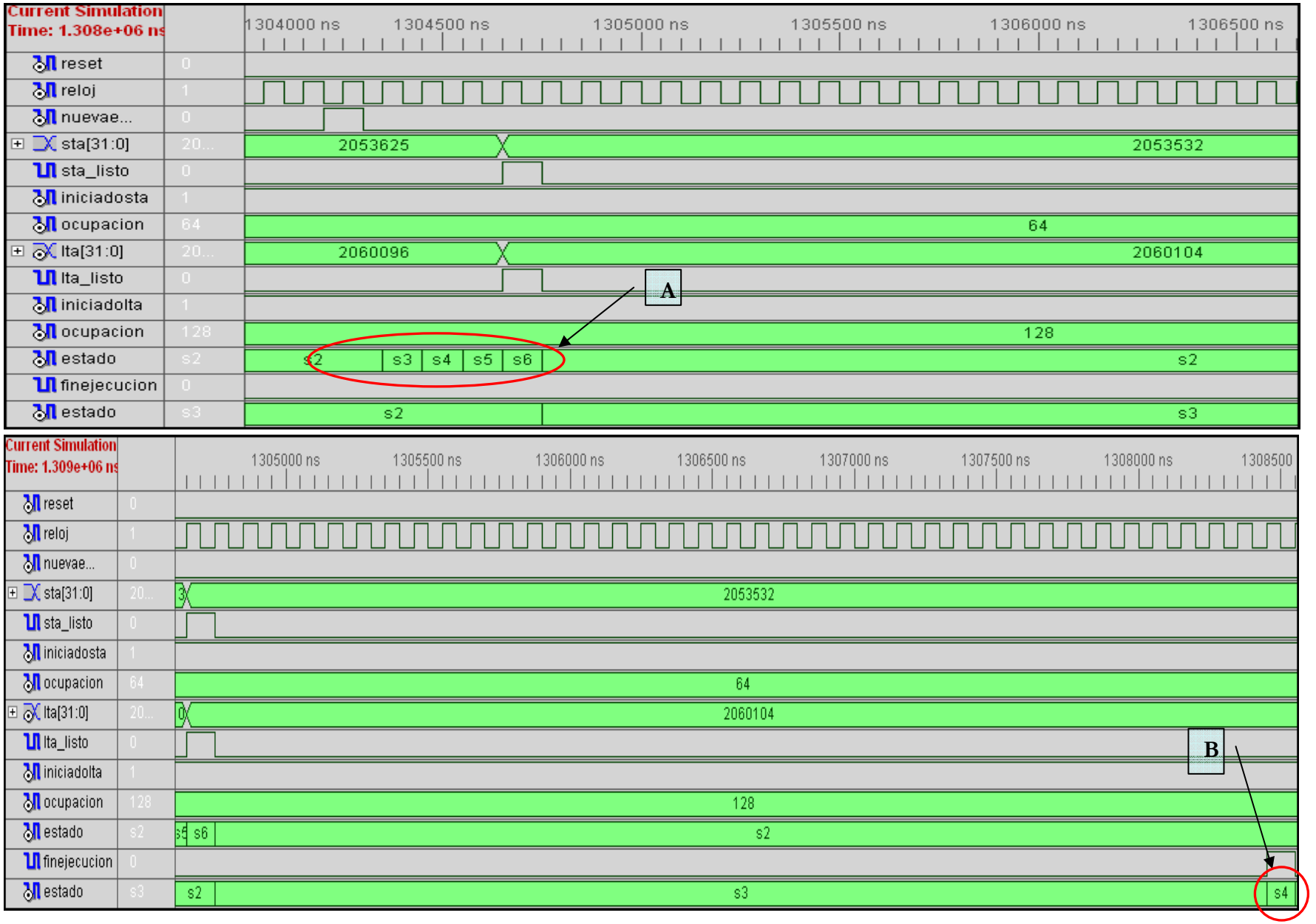
(950 ns-250 ns)/100 ns = 7 ciclos



(*1)Estado para los módulos STA y LTA. **(*2)** Estado del bloque principal

Ahora contamos, aparte de con *lta_listo* y *sta_listo*, con *iniciadolta* e *iniciadosta* que indican cuando podemos empezar a devolver los resultados de la división entre *STA* y *LTA*.

Una vez que se han llenado los módulos que calculan la media estarán activas su señales de *iniciado*. Observamos como el tiempo de cálculo de las medias es de 6 ciclos (**A**) y el tiempo de cálculo del cociente (**B**) es igual al algoritmo sin optimizar (37 ciclos).


Current Simulation Time: 1.308e+06 ns
1304000 ns
1304500 ns
1305000 ns
1305500 ns
1306000 ns
1306500 ns
reset
reloj
nuevae...
sta[31:0]
sta_listo
iniciadosta
ocupacion
lta[31:0]
lta_listo
iniciadolta
ocupacion
estado
finejecucion
estado
2053625
2053532
2060096
2060104
64
128
s2
s3
s4
s5
s6
A
Current Simulation Time: 1.309e+06 ns
1305000 ns
1305500 ns
1306000 ns
1306500 ns
1307000 ns
1307500 ns
1308000 ns
1308500
B

El número de ciclos que tarda en ejecutarse el algoritmo optimizado es:

$$(T_{fin}-T_{inicio})/T_{ciclo}$$

(1028600 ns - 1024300 ns)/100 ns = 43 ciclos

7.4.4. Ocupación en la FPGA

A continuación presentamos las estadísticas de implementación del algoritmo en la FPGA para un número de muestras de 64 y 128 para STA y LTA respectivamente.

```
=========================================================
*                      Final Report                     *
=========================================================
Final Results
RTL Top Level Output File Name     : Allen.ngr
Top Level Output File Name         : Allen
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 101

Cell Usage :
# BELS                             : 4876
#      GND                         : 3
#      INV                         : 41
#      LUT1                        : 109
#      LUT2                        : 254
#      LUT3                        : 1078
#      LUT4                        : 241
#      MULT_AND                    : 33
#      MUXCY                       : 1587
#      VCC                         : 2
#      XORCY                       : 1528
# FlipFlops/Latches                : 2675
#      FD                          : 2407
#      FDC                         : 41
#      FDCE                        : 225
```

```
#      FDE                  : 2
# RAMS                      : 2
#      RAMB16_S36_S36       : 2
# Shift Registers           : 97
#      SRL16                : 64
#      SRL16E               : 4
#      SRLC16               : 29
# Clock Buffers             : 2
#      BUFGP                : 2
# IO Buffers                : 67
#      IBUF                 : 33
#      OBUF                 : 34
=========================================================
                    Device utilization summary:
                    ---------------------------

Selected Device : 2vp30ff896-6

 Number of Slices:              1511  out of  13696    11%
 Number of Slice Flip Flops:    2675  out of  27392     9%
 Number of 4 input LUTs:        1820  out of  27392     6%
    Number used as logic:       1723
    Number used as Shift registers:  97
 Number of IOs:                 101
 Number of bonded IOBs:           69  out of    556    12%
 Number of BRAMs:                  2  out of    136     1%
 Number of GCLKs:                  2  out of     16    12%
```

Como podemos observar en la siguiente tabla el ratio de utilización de la FPGA ha descendido notablemente con respecto a la implementación que realiza divisiones en vez de desplazamientos. Esto se debe a la eliminación de sendos divisores en los módulos ModuloSTA y ModuloLTA que han sido sustituidos por la operación lógica de desplazamiento a la derecha.

Con este diseño mejoramos además de la velocidad de ejecución del algoritmo el espacio que ocupa el diseño en la FPGA y por consiguiente disminuimos el consumo final.

	Sin Optimizar (%)	**Optimizado (%)**
Number of Slices	29	11
Number of Slice Flip Flops	26	9
Number of 4 input LUTs:	16	6

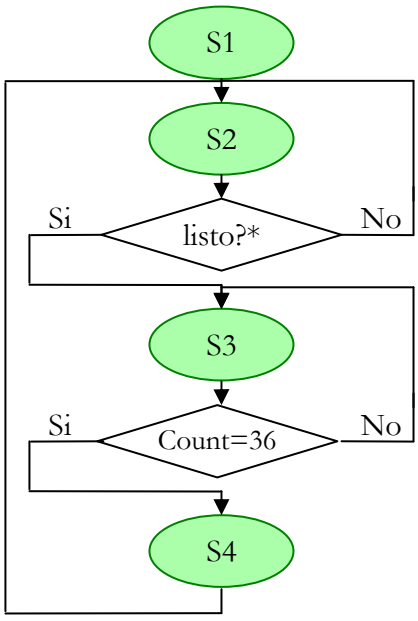
7.5. Algoritmo STA/LTA (medias cuadráticas)

Ese algoritmo es una variación sobre el anterior en el que en vez de calcular las medias se calculan las medias cuadráticas. Se diferencia del algoritmo anterior en los componentes *ModuloLTA* y *ModuloSTA* tienen un multiplicador para calcular el cuadrado del nuevo dato, y es el resultado de dicha multiplicación lo que se usa para los cálculos de las medias. La ejecución del resto del algoritmo es similar al caso anterior.

- El componente Bloque principal pasa de un estado de reset a un estado de espera.
- La señal de nuevo dato a procesar se pasa directamente a los módulos ModuloLTA y ModuloSTA que calculan el cuadrado del nuevo dato y seguidamente la nueva media. Se mantiene el estado actual hasta que se recibe la notificación de que ambos módulos han acabado su ejecución para el nuevo dato recibido.
- Se pasa entonces al último paso de ejecución del algoritmo en la que se calcula el cociente STA/LTA y se decide si se notifica un evento o no. Además se le comunica al componente de comunicación que se ha acabado la ejecución del algoritmo y se le pasan datos sobre la ejecución.

7.5.1. Diagrama de flujo

Estado	Cometido
S1	Reset
S2	Espera ejecución STA y LTA
S3	Calcula cociente
S4	Señaliza fin de ejecución



7.5.2. Diagramas de bloques

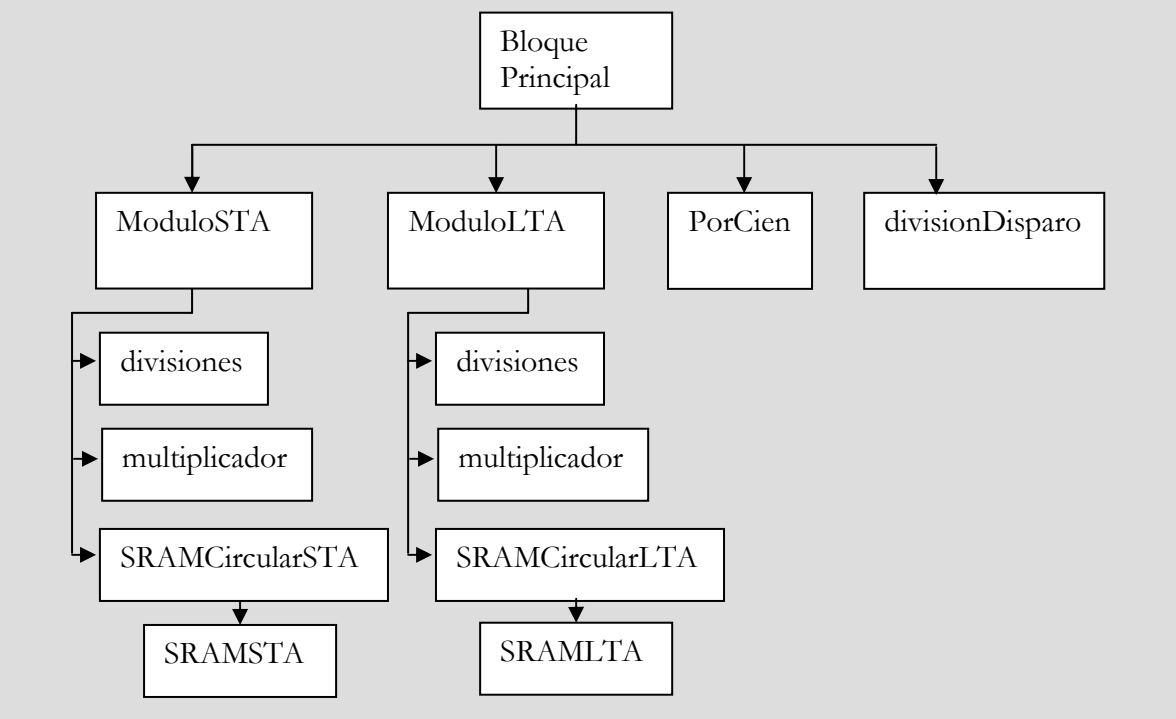


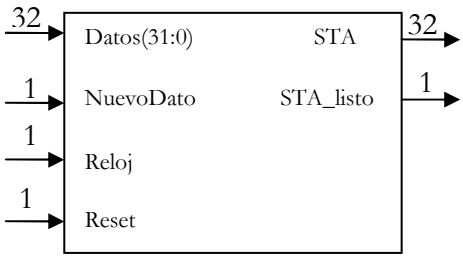
Figura 18. Diagrama de bloques del algoritmo.

Los únicos bloques que se ven modificados son *ModuloSTA* y *ModuloLTA* por lo que son los únicos que detallamos a continuación. Ahora cada uno dispondrá de un multiplicador que obtendrá el cuadrado de los datos de entrada y será ese cuadrado lo que se use para hacer la media. El resto de bloques no varían.

7.5.2.1. Bloque ModuloSTA

Añadimos el cálculo del cuadrado del dato de entrada. Una vez calculado el dato seguimos con la ejecución de manera idéntica al algoritmo anterior.

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
STA	Salida	Media valores almacenados en la ram circular	32
STA_listo	Salida	Señaliza fin de proceso del nuevo dato	1



Interconexionado

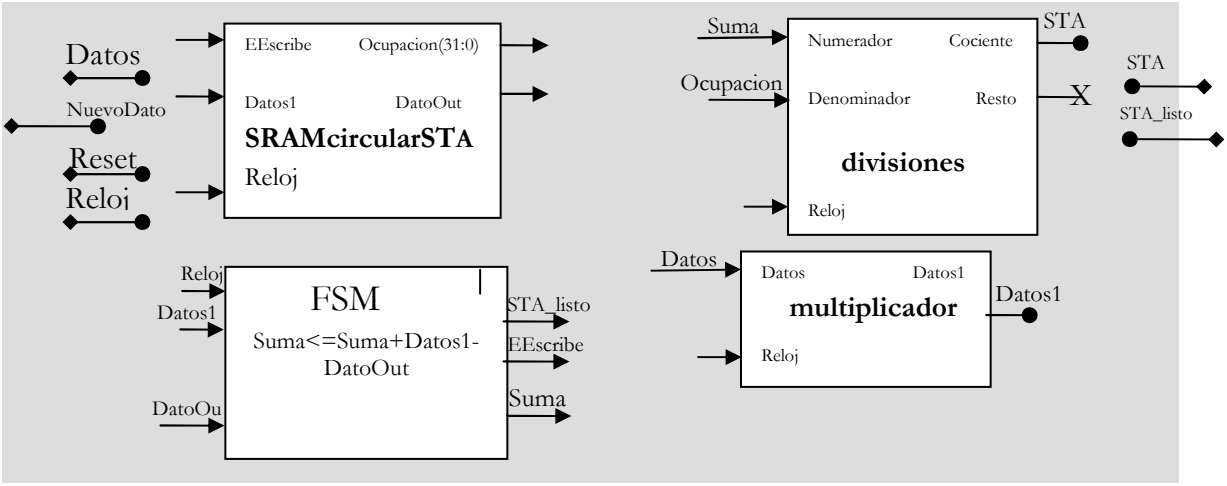
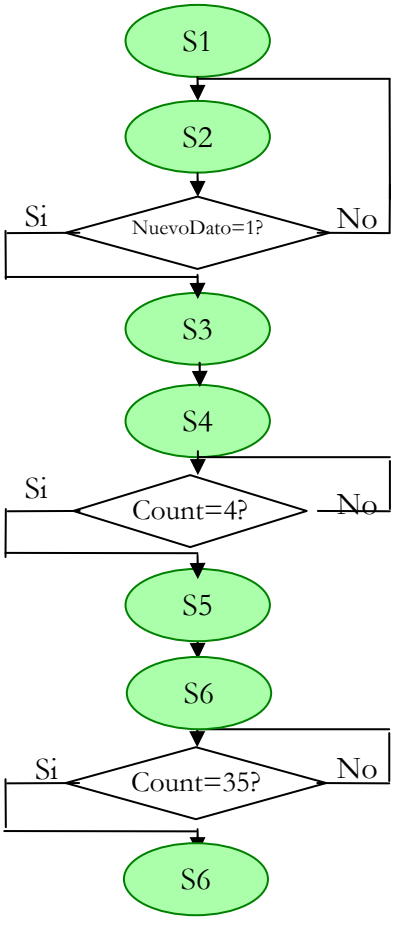


Figura 19. Interconexión de los bloques.

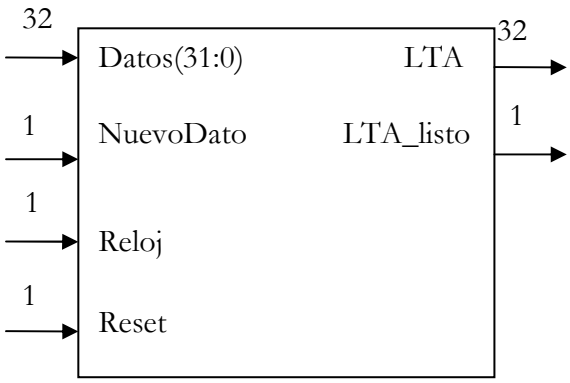
Diagrama de flujo

Estado	Cometido
S1	Reset
S2	Espera de nuevo dato
S3	Lectura dato
S4	Cálculo del cuadrado y recálculo de Suma
S5	Grabado del cuadrado. Actualización ocupación
S6	Cálculo de la media.
S7	Señala fin de procesado del nuevo dato.



7.5.2.2. Bloque ModuloLTA

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
LTA	Salida	Media valores almacenados en la ram circular	32
LTA_listo	Salida	Señaliza fin de proceso del nuevo dato	1



Interconexionado

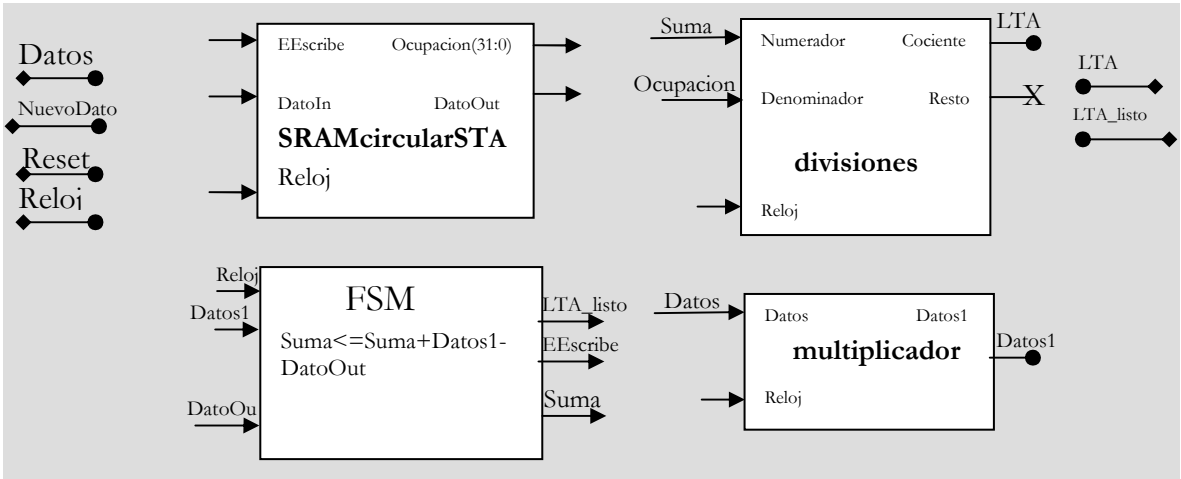
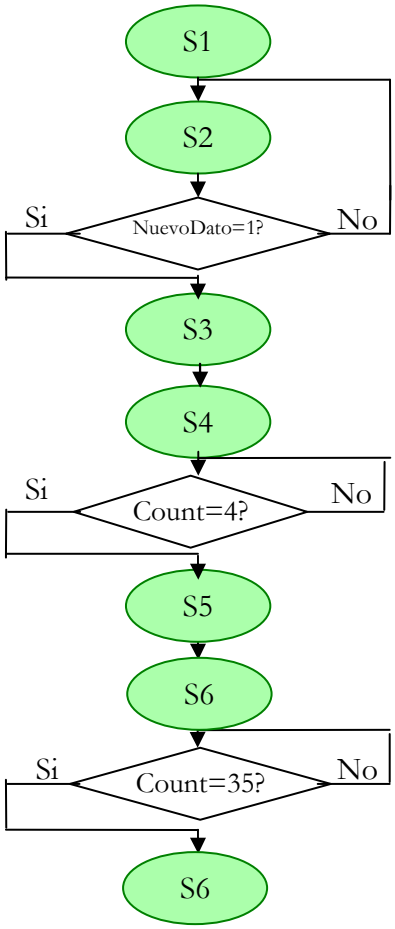


Figura 20. Interconexión de los bloques.

Diagrama de flujo



Estado	Cometido
S1	Reset
S2	Espera de nuevo dato
S3	Lectura dato
S4	Cálculo del cuadrado y recálculo de Suma
S5	Grabado del cuadrado. Actualización ocupación
S6	Cálculo de la media.
S7	Señala fin de procesado del nuevo dato.

7.5.3. Ciclos de ejecución y Ocupación en la FPGA

El número de ciclos de ejecución este algoritmo será superior (al realizar una multiplicación extra) al número de ciclos del algoritmo de medias.

(Nºciclos de ejecución)/dato= 79+(Nºciclos Multiplicador)+1 ciclos

79+5+1=85 ciclos

En cuanto a la ocupación tenemos dos multiplicadores de números de 32 bits en cada módulo de medias cuadráticas por lo que el tamaño del diseño aumenta significativamente respecto al algoritmo que usa medias. Más concretamente, cada multiplicador ocupa 1088 *Luts* de la fpga.

Teniendo en cuenta lo anterior y que con esta variación de medias cuadráticas no se han obtenido en software resultados más significativos que con el algoritmo de medias hemos desestimado implementarlo en hardware.

7.6. Algoritmo Dust Devils

Este algoritmo se ha implementado haciendo uso de dos bloques similares al algoritmo STA/LTA de medias, un bloque para presión y otro para temperatura. La única diferencia respecto al algoritmo general viene dada por la condición de disparo.

Parámetro	Presión	Temperatura
Muestras STA	8	4
Muestras LTA	1024	32
THR de bajada	0.994	-
THR de subida	-	1.015

Se ha implementado un algoritmo con los parámetros de la tabla anterior. La condición es que salte simultáneamente un evento de bajada en la presión y uno de subida en la temperatura. Eventos de subida en presión o bajada en temperatura no indican la presencia de dust devils, por lo que se omiten.

7.6.1. Diagramas de bloques

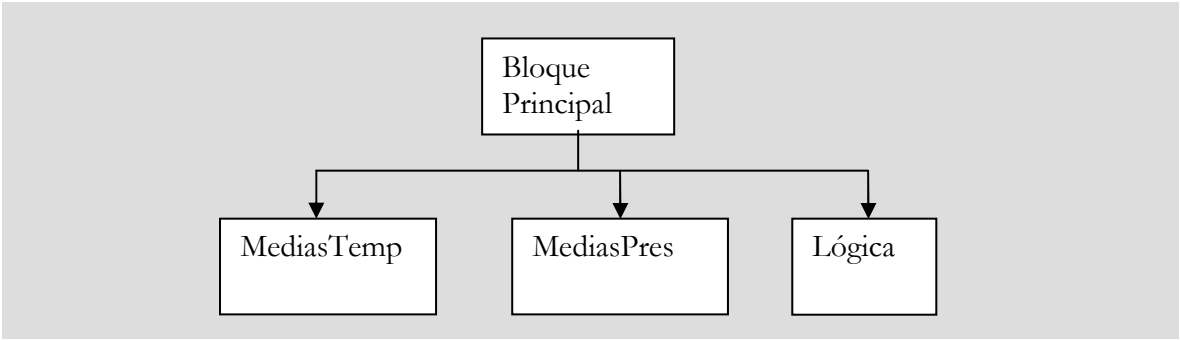
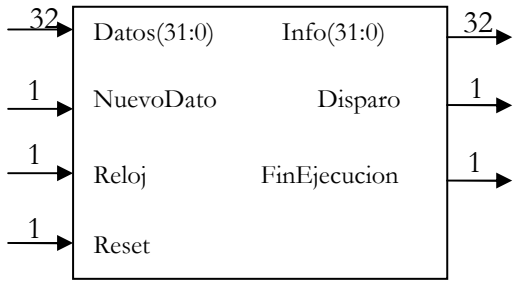


Figura 21. Diagrama de bloques del algoritmo.

Se conectan en paralelo dos bloques del algoritmo de medias y se van sirviendo los datos a cada uno alternativamente según llegan. Será necesaria una nueva estrategia de comunicación. En este caso los grupos de 32 bits impares serán datos de temperatura y los pares de presión. El bloque lógica se encarga de crear las señales necesarias y de seleccionar la ruta de datos correcta en cada caso.

7.6.1.1. Bloque Principal

Señal	Tipo	Objetivo	Bits
Datos	Entrada	Entrada de datos	32
NuevoDato	Entrada	Señaliza un nuevo dato	1
Reloj	Entrada	Reloj del sistema	1
Reset	Entrada	Reset del sistema	1
Info	Salida	Información de ejecución	32
FinEjecución	Salida	Señaliza fin de ejecución	1
Disparo	Salida	Resultado de la ejecución	1



Interconexionado

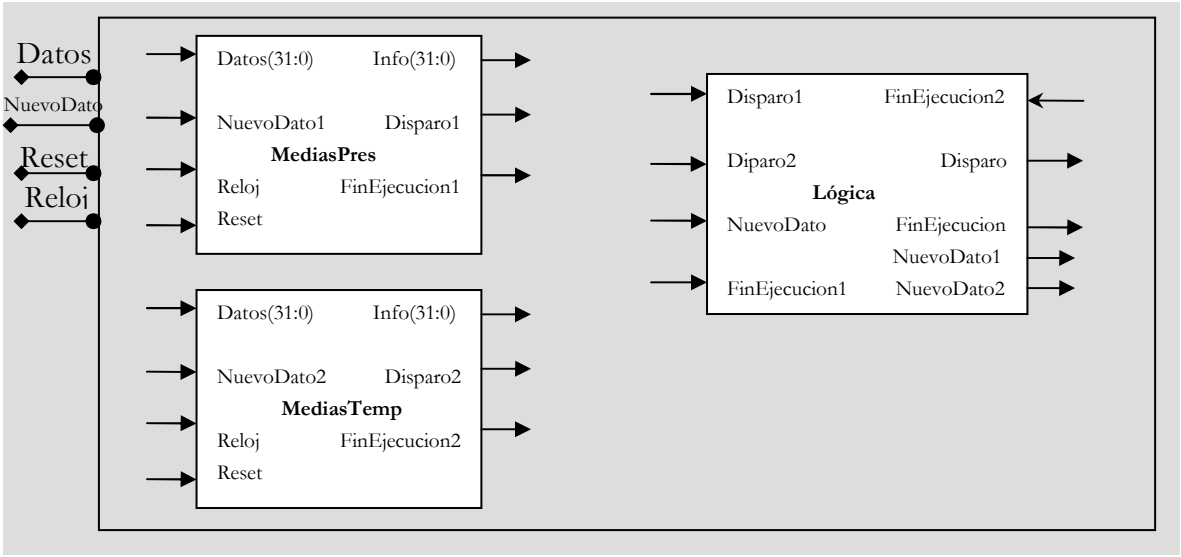


Figura 22. Interconexión de los bloques.

7.6.1.2. Bloque MediasPres

La única diferencia con respecto al algoritmo STA/LTA medias viene dada por la condición de disparo. En este caso la condición de disparo será:

```
si ((STA/LTA) < 0.994) entonces
        Disparo:=1;
fsi
```

7.6.1.3. Bloque MediasTemp

La única diferencia con respecto al algoritmo STA/LTA medias viene dada por la condición de disparo. En este caso la condición de disparo será:

```
si ((STA/LTA) > 1.015) entonces
        Disparo:=1;
fsi
```

7.6.2. Ciclos de ejecución y Ocupación en la FPGA

Al componerse el diseño únicamente de dos bloques del algoritmo de medias en paralelo tenemos el mismo número de ciclos de ejecución que tiene el módulo individualmente.

La ocupación es algo inferior al doble que el caso estudiado del algoritmo STA/LTA medias debido a que se han utilizado otros parámetros para el número de muestras. Obtenemos el siguiente resultado:

```
=========================================================
*                    Final Report                       *
=========================================================

Final Results
RTL Top Level Output File Name     : envoltorio.ngr
Top Level Output File Name         : envoltorio
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 5

Cell Usage :
# BELS                             : 23141
#      GND                         : 9
#      INV                         : 186
#      LUT1                        : 84
#      LUT2                        : 1018
#      LUT3                        : 6158
#      LUT4                        : 520
#      MULT_AND                    : 198
#      MUXCY                       : 7480
#      VCC                         : 6
#      XORCY                       : 7482
# FlipFlops/Latches                : 14310
#      FD                          : 14298
#      FDE                         : 12
```

```
# Shift Registers                 : 582
#      SRL16                      : 384
#      SRL16E                     : 24
#      SRLC16                     : 174
# Clock Buffers                   : 1
#      BUFGP                      : 1


=========================================================================

                            Device utilization summary:
                            ---------------------------

Selected Device : 2vp30ff896-6

 Number of Slices:                     7720  out of  13696    56%
 Number of Slice Flip Flops:          14310  out of  27392    52%
 Number of 4 input LUTs:               8548  out of  27392    31%
    Number used as logic:              7966
    Number used as Shift registers:     582
 Number of IOs:                         5
 Number of bonded IOBs:                 1  out of    556     0%
 Number of GCLKs:                       1  out of     16     6%
```

8. Conclusiones

Vamos a estudiar los resultados de nuestro detector de dust devils y seguidamente procederemos a explicar las ventajas de nuestra implementación hardware.

8.1. Comprobación de los resultados del detector de dust devils

Estudiaremos los resultados que ofrece nuestro detector de eventos. Para ello vamos a usar los datos de la misión Mars Pathfinder. Durante esta misión se detectaron varios dust devils que están bien documentandos por lo que es un conjunto de datos idóneo. Buscamos unos valores óptimos para nuestro detector.

Prueba 1

Pruebas realizadas con los siguientes valores:

Parámetro	Presión	Temperatura
Muestras STA	8	4
Muestras LTA	1024	32
THR de bajada	0.9975	-
THR de subida	-	1.01

Sol	Hora local	Detectado
25	1310	Sí
25	1353	Sí
34	952	No
34	1132	Sí
34	1138	Sí
38	1232	Sí
39	1131	No
39	1347	Sí
49	1102	Sí
52	1203	No

55	1419	Sí
60	1009	No
62	1231	No
62	1234	Sí
62	1406	Sí
68	1142	No
68	1329	Sí
69	1254	Sí
70	1425	Sí

Tabla. Dust devils detectados en cada sol.

Prueba 2

Pruebas realizadas con los siguientes valores:

Parámetro	Presión	Temperatura
Muestras STA	4	4
Muestras LTA	1024	32
THR de bajada	0.998	-
THR de subida	-	1.01

Sol	Hora local	Detectado
25	1310	Sí
25	1353	Sí
34	952	No
34	1132	Sí
34	1138	Sí
38	1232	Sí
39	1131	Sí
39	1347	Sí
49	1102	Sí
52	1203	No

55	1419	Sí
60	1009	No
62	1231	Sí
62	1234	Sí
62	1406	Sí
68	1142	No
68	1329	Sí
69	1254	Sí
70	1425	Sí

Tabla. Dust devils detectados en cada sol.

Podemos observar que al mejorar la sensibilidad de las variaciones de presión, aumentamos la cantidad de dust devils detectados, aunque también aumenta la cantidad de detecciones de dust devils no indicados en el artículo de Renno [1]. Estas detecciones pueden ser falsos positivos o tratarse de dust devils no detectados por el método de Renno.

	Nº de Dust devils detectados
Prueba1	13 (de 19)
Prueba2	15 (de 19)

Tabla. Comparativa dust devils detectados

En una situación real el detector de eventos será usado para aumentar la frecuencia de medida y envío de datos a la Tierra proporcionando autonomía al sistema y poder obtener así datos más precisos sobre los eventos detectados (en nuestro caso dust devils), sean relevantes o no.

8.2. Comparativa de tiempos hardware/software

Es importante mencionar el contexto en el cual se ha realizado esta comparativa. Como ya mencionamos anteriormente, el ordenador sobre el que se han ejecutado las pruebas funciona a 2,33GHz.

El reloj utilizado de la FPGA tiene una frecuencia de reloj de 100MHz, es decir tiene una frecuencia casi 24 veces inferior. Aún así se obtienen unos resultados más que significativamente mejores.

A continuación mostramos el número de muestras por segundo que puede ejecutar el algoritmo en software y en hardware y compararemos la ganancia del hardware respecto al software.

Algoritmo STA/LTA medias

	Software	Hardware(*)	Hardware optimizado(*)
Muestras por segundo	94931.0070	1327311.9241	2383127.2727
Ganancia	1.0000	13.9819	25.1037

Algoritmo detector de dust devils

	Software	Hardware(*)	Hardware optimizado(*)
Muestras por segundo	91819.5936	1327311.9241	2383127.2727
Ganancia	1.0000	14.4562	25.9543

(*)Resultados obtenidos suponiendo que se dispone de un nuevo dato cada vez que se acaba de procesar el actual, tal como sucede en la versión software del algoritmo.

Observamos como la ganancia del hardware diseñado específicamente respecto al algoritmo en software es superior a 13 en ambos casos del algoritmo hardware normal, y superior a 25 en el caso del algoritmo optimizado. Con estos resultados y teniendo en cuenta la diferencia tan significativa entre la frecuencia de reloj del ordenador y la FPGA podemos concluir que el uso de hardware reconfigurable para este tipo de algoritmos proporciona unos buenos resultados, mejores que el software ejecutándose en un ordenador de prestaciones superiores a la FPGA utilizada.

9. Bibliografía

[1] Martian and terrestrial dust devils: Test of a scaling theory using Pathfinder data.
Nilton O. Renno et al, 2000

[2] Characterization of the Martian Surface Layer
Germán Martínez, Francisco Valero and Luis Vázquez, 199
Universidad Complutense de Madrid, Madrid, Spain

[3] Automatic phase pickers: their present use and future prospects
Rex Allen, 1982

[4] Multi-component autoregressive techniques for the analysis of seisgrams
M. Leonard, B.L.N. Kennet, 1999

[5] Les algorithmes de détection automatique d'ondes sismiques
Olivier Cuenot

[6] La gran aventura de la exploración de Marte
Luis Vázquez Martínez, 2008

[7] PDS: The planetary Atmospheres Data Node. Pathfinder Data
NASA, 1996

[8] Introducción a la programación en VHDL
Marcos Sánchez-Élez, Mª Carmen Molina
Universidad Complutense de Madrid, Spain

[9] VHDL sintetizable
Marcos Sánchez-Élez, David Atienza Alonso
Universidad Complutense de Madrid, Spain