Proyecto Fin de Master
Curso 2006-2007

GESTOR DE TAREAS PARA HARDWARE DINAMICAMENTE RECONFIGURABLE 2D

Raquel Sánchez Delgado

Dirigido por:
Prof. Hortensia Mecha López
Dpto. de Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

Tabla de contenidos

1. Introducción

El proyecto consiste en una simulación de un gestor para tareas que han de ejecutarse en una FPGA.

Para manejar el espacio libre disponible para ubicar las tareas dentro de la FPGA, se propone una estructura de datos, llamada *Conjunto de Listas de Vértices* (VLS); mediante la cuál se puede conocer la cantidad de espacio libre y la forma que tiene este espacio.

Si no usáramos esta estructura, habría que recorrer toda la FPGA para saber qué posiciones están ocupadas, y posteriormente considerar qué posiciones son capaces de alojar la tarea, con el coste tan elevado que esto requiere en tiempo y espacio. Si se optara, por utilizar rectángulos para representar el espacio libre, como se propone en [BKKR2000], el coste para gestionarlos es bastante elevado.

El *Conjunto de Listas de Vértices*, precisa de otras estructuras auxiliares que son la *Lista de Tareas en Ejecución* y la *Lista de Tareas en Espera*. La *Lista de Tareas en Ejecución*, almacena los datos sobre las tareas que se están ejecutando actualmente en la FPGA. La *Lista de Tareas en Espera*, guarda la información necesaria acerca de las tareas que a su llegada no pudieron ubicarse dentro de la FPGA. Estas tres estructuras de datos son gestionadas por el *Planificador de Tareas*.

El Planificador de Tareas es el encargado de decidir qué hacer cuando una tarea llega o cuando termina su ejecución; para lo cual utiliza las tres estructuras de datos y otros dos módulos, que son el *Selector de Vértices* y el *Actualizador de Listas de Vértices*.

Cuando una tarea nueva llega, el planificador comprueba si hay hueco en la FPGA consultando el *Conjunto de Listas de Vértices*. Si hay hueco entonces invoca al *Selector de Vértices*, para elegir la ubicación de la tarea dentro de la FPGA, inserta la información de la tarea en la *Lista de Tareas en Ejecución* y llama al *Actualizador de Listas de Vértices*. Si por el contrario, al llegar la tarea no hubiera suficiente espacio disponible el planificador meterá la tarea en la *Lista de Tareas en Espera*.

Cuando una tarea finaliza su ejecución, el planificador se encarga de invocar al *Actualizador de Listas de Vértices* y de sacar la tarea de *la Lista de Tareas en Ejecución*.

El *Planificador de Tareas* además se ocupa de comprobar si las tareas que están en la lista de espera se pueden mandar ejecutar, debido a que en ese momento haya espacio disponible. También comprueba si cada una de las tareas de la lista de espera siguen cumpliendo las restricciones temporales, para que en caso de que hubiera hueco se ejecutaran en la FPGA; si alguna de las tareas no cumpliera estas restricciones de tiempo entonces la tarea se descartaría, eliminándose de la *Lista de Tareas en Espera*.

El *Selector de Vértices* se encarga de encontrar los posibles huecos para la colocación de una nueva tarea y de seleccionar uno entre los distintos candidatos; para

lo cuál se dispone de distintas heurísticas, que son First Fit, Adyacencia 2D, Fragmentación, Adyacencia 3D y Look ahead. Este modulo consulta el *Conjunto de Listas de Vértices* para poder elegir el vértice donde se ubicara la tarea durante su ejecución.

El *Actualizador de Listas de Vértices*, es el módulo que se encarga de mantener actualizada el *Conjunto de Listas de Vértices (VLS)* y se utiliza cada vez que una tarea acaba de llegar al gestor o acaba de finalizar su ejecución dentro de la FPGA.

2. Motivaciones

Las FPGAs son dispositivos de hardware reconfigurable, que permiten reconfiguración parcial en tiempo de ejecución y ofrecen muy buenos resultados en términos de rendimiento.

La integración de bloques RAM y elementos de grano grueso en las FPGAs, tales como multiplicadores o procesadores, ha permitido aprovechar mejor el área reconfigurable.

Las FPGAs son normalmente más lentas que su correspondiente ASIC. Sin embargo, tienen varias ventajas sobre los ASIC, de entre las que destacan un menor tiempo de fabricación, capacidad de reprogramación, y menores gastos en el tiempo de desarrollo.

Las FPGAs ofrecen la posibilidad de multitarea HW, ya que los avances en tamaño permiten que quepan varias tareas distintas simultáneamente en la FPGA. Además, las tareas HW pueden entrar y salir de forma autónoma gracias a la capacidad de reconfiguración dinámica y parcial que poseen las FPGAs.

Si se quieren incorporar elementos de hardware reconfigurables para beneficiarse de las ventajas ya comentadas, es necesario extender el sistema operativo para gestionar este tipo de recursos, a lo que se denomina “Gestión de Hardware Reconfigurable”, y se trata más adelante en el punto 3.

La arquitectura interna de las FPGAs es habitualmente 2D, y por tanto también los chips actuales, aún así en la tecnología de FPGA actual prevalece la técnica de reconfiguración 1D (basada en columnas). Ya que el mercado, hasta ahora, había considerado innecesaria la flexibilidad extra de la reconfiguración 2D, como sucedía en la Virtex II. Sin embargo, existen arquitecturas anteriores tales como XC6200 o actuales como la Virtex IV y Virtex V, que sí permiten reconfiguración 2D.

En el presente, aunque existen recursos multitarea en dos dimensiones (2D), la gestión de las FPGA es en una dimensión (1D); así que resultan de interés general las técnicas de gestión 2D.

Uno de los problemas más interesantes de la multitarea HW es la colocación de las tareas, que consiste en decidir dónde localizar el bitmap de una nueva tarea cuando debe ser ejecutada. La mayoría de los investigadores, se ocupan de este problema centrándose en una gestión de recursos homogéneos y de grano fino.

Es necesaria una estructura de datos para mantener la información acerca del área libre disponible. El algoritmo debe elegir el mejor lugar para colocar la tarea recién llegada, en términos de una métrica dada por el usuario y de la forma más eficiente posible. Las estructuras propuestas y más relevantes hasta ahora, entre otras, se han basado en rectángulos máximos expuesta en [BKKS2000], superpuestos o no, modelos de escalera, etc.

La aproximación utilizada en este trabajo, mantiene la información acerca del área libre disponible mediante una estructura de lista de vértices, que representa los límites del área ocupada.

Por otro lado, aún con buenas heurísticas de colocación, la fragmentación del área libre disponible es inevitable cuando las tareas entran y salen de la FPGA. Ya que el estado de la fragmentación es una medida indirecta de la probabilidad de encontrar una localización conveniente para una nueva tarea en la FPGA en un futuro próximo, es necesaria una precisa métrica de fragmentación. En este trabajo, se ha desarrollado una métrica, y se ha utilizado para implementar una heurística de selección para la colocación.

3. Funcionamiento básico del gestor

3.1. Descripción de FPGA

La FPGA la representaremos como un modelo de dos dimensiones (2D) formada por una malla homogénea de W * H bloques básicos, siendo W la anchura o número de columnas y H la altura o número de filas de la FPGA.

Se supone que cada bloque básico está compuesto por un número suficiente de CLBs, como para incluir suficientes elementos de procesamiento, así como de un interfaz estándar de E/S para que haya comunicación e interconexión entre los elementos de la FPGA, de forma similar al presentado en [BCAA2005].

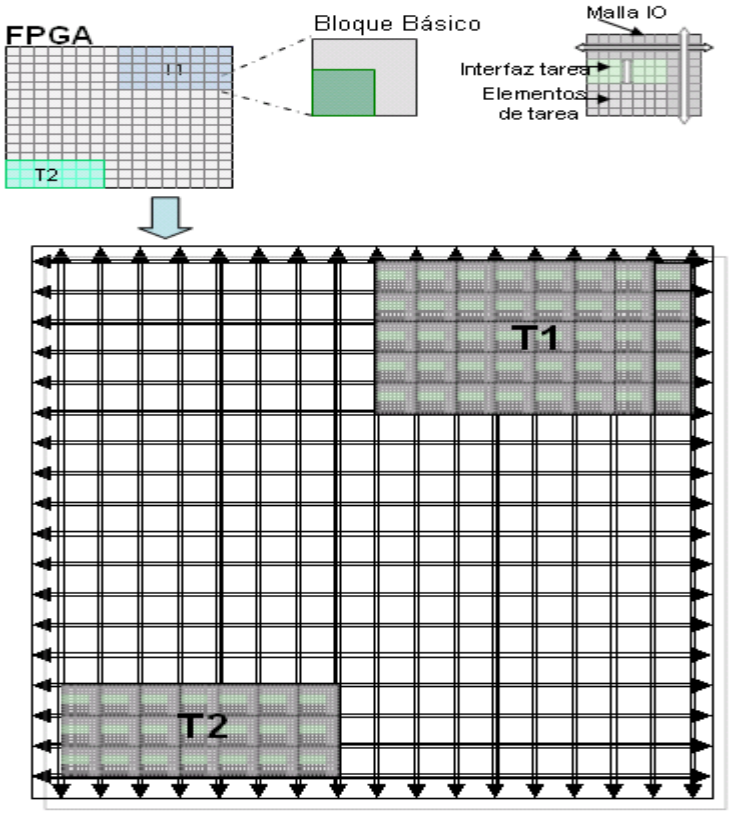


Fig. 1. FPGA, tarea y modelo E/S.

En cuanto al tamaño de las particiones, es variable, ya que aporta mayor flexibilidad, las tareas se ajustan más exactamente a su tamaño y se aprovecha mejor el espacio libre. Al ser el tamaño de las particiones variable, puede haber fragmentación externa (al salir las tareas dejan huecos libres), es necesaria una gestión dinámica y compleja de los recursos.

3.2. Descripción de las tareas

El tamaño mínimo de una tarea se corresponde con el tamaño de un bloque básico; y en general, una tarea puede comprender un número arbitrario de bloques básicos distribuidos en un rectángulo de $w_i * h_i$.

Las tareas se caracterizan por ser independientes entre sí y son reubicables, por poderse insertar en posiciones arbitrarias de fila y columna.

Cada una de las tareas está definida con la siguiente tupla de parámetros:

$$T_i = \{ w_i, h_i, t\_ex_i, t\_arr_i, t\_max_i \}$$

donde:

- w_i, es el ancho de la tarea
- h_i, es la altura de la tarea
- t_ex_i, es el tiempo de ejecución
- t_arr_i, es el tiempo de llegada de la tarea T_i
- t_max_i, es el tiempo máximo permitido para finalizar la ejecución de la tarea.

Todos los tiempos usados son absolutos, excepto t_ex_i que es referido al tiempo de origen.

Para cada tarea, se define el volumen de computación de la tarea como:

$$V_i = w_i * h_i * t\_ex_i$$

Este volumen de computación representa la carga de trabajo que la tarea demanda a la FPGA.

3.3. Conjunto de Listas de Vértices

El *Conjunto de Listas de Vértices* (VLS), es una estructura que sirve para tener información acerca del área libre de la FPGA. En ella, se almacenan los posibles vértices candidatos, de tal forma, que se seleccione uno de ellos para la colocación de una nueva tarea.

Cada componente del *Conjunto de Listas de Vértices* (VLS), es una *Lista de Vértices* (VL), que describe un hueco; entendiendo como hueco, el fragmento independiente de espacio contiguo libre en la FPGA.

A su vez, cada *Lista de Vértices* (VL), está compuesta por una sucesión de vértices que representan los límites del hueco correspondiente, como se muestra en la figura 2.

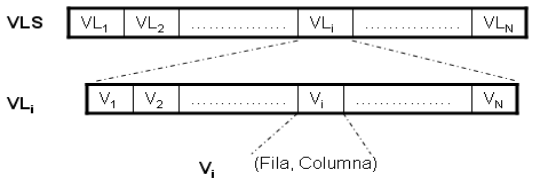


Fig. 2. Estructura de la VLS

Un vértice, se obtiene como la intersección entre los lados de la *Lista de Vértices* (VL) y los lados de las tareas que hay en ejecución dentro de la FPGA.

Los vértices se han representado como una 3-upla de parámetros:

{Fila, Columna, Virtual}

donde Fila y Columna, se corresponden con el número de fila y columna respectivamente, donde está situado el vértice en la FPGA; y Virtual, indica si es virtual o no.

Habitualmente, si el vértice no es virtual, el campo Virtual se omitirá, y sólo aparecerá en el caso de que lo sea. Tanto el parámetro Fila como el de Columna, se encuentran en un rango comprendido entre 0 y el ancho o alto, respectivamente, que tenga la FPGA.

Cada *Lista de Vértices* (VL) que forman el *Conjunto de Listas de Vértices* (VLS), es una lista ordenada de vértices, donde el primer vértice es el menor vértice de toda la lista, entendiendo por menor vértice como aquel que tiene menor fila que todos los demás; y en caso de que haya dos con la misma fila, se elige el que tenga menor número de columna. El primer vértice gráficamente se puede ver como el vértice que más abajo y a la derecha se encuentre. El segundo vértice, se obtiene variando la columna hacia la izquierda con respecto al primero; el tercero, variando la fila con respecto al segundo, y así sucesivamente. Este orden de obtener los vértices, es distinto en el caso de que exista una isla, y por tanto existan vértices virtuales en la lista de vértices.

En la representación gráfica del estado de la FPGA, el origen de las coordenadas se encuentra en la esquina inferior derecha. El eje vertical son las filas de la FPGA y el horizontal las columnas; de este modo, la esquina superior izquierda tendrá como coordenadas (filasFPGA, columnasFPGA).

La figura 3 representa un posible estado de la FPGA, además se puede ver un dibujo del Conjunto de Listas de Vértices que también se detalla a continuación:

[[(0,2),(0,6),(2,6),(2,7),(4,7),(4,8),(6,8),(6,6),(8,6),(8,2),(6,2),(6,0),(2,0),(2,2),(3,2),(3,4), (1,4),(1,2)]]

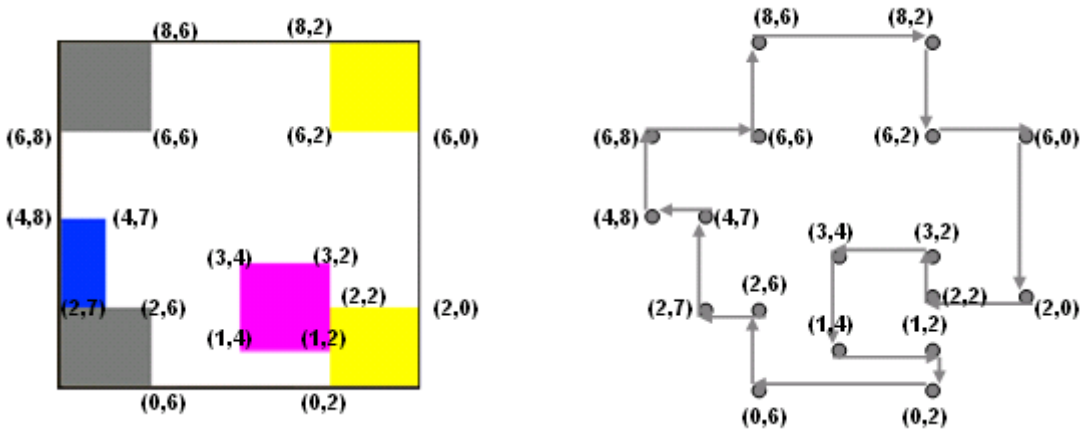


Fig. 3. Muestra de posible estado de FPGA junto con su VLS

En la figura 4, se puede ver un ejemplo de estado de FPGA junto con su VLS, que se caracteriza por tener dos listas de vértices con el vértice repetido (2, 5) en cada una de ellas. El Conjunto de Listas de Vértices se puede ver con más detalle a continuación:

[[(0,5), (0,8), (2,8), (2,5)], [(2,0), (2,5), (4,5), (4,6), (8,6), (8,2), (4,2), (4,0)]]

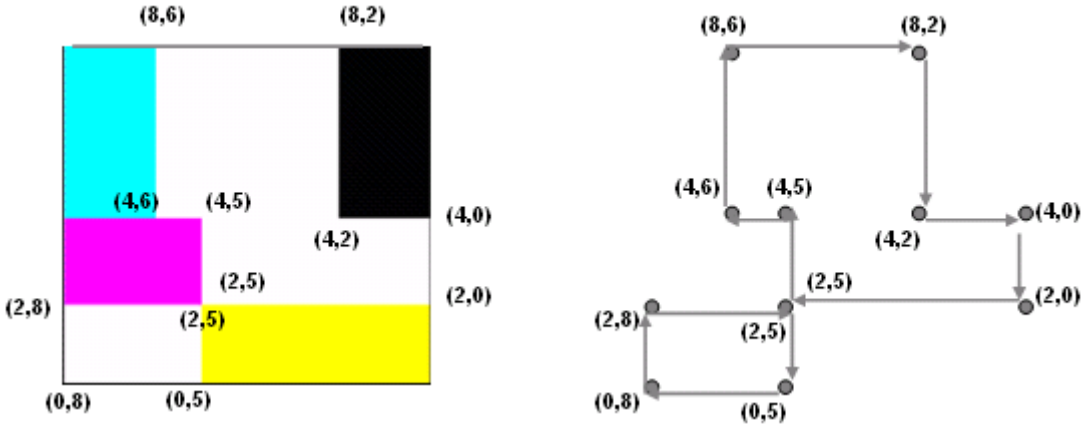


Fig. 4.

En las *Listas de Vértices*, existe un caso especial que son los *vértices virtuales*, que son consecuencia de *islas* dentro de la FPGA. Hay islas, cuando una o varias tareas no están colocadas en los límites de la FPGA, y alrededor sólo hay bloques básicos sin ocupar. Las islas se originan al extraer tareas que ya han finalizado su ejecución.

Como consecuencia de las islas, aparecen los vértices virtuales. Estos vértices no son como los demás vértices, sino que surgen como solución para mantener la información sobre las islas; de forma que se obtenga una única lista de vértices dentro de un hueco, que contiene una isla.

La lista de vértices que comprende una isla está caracterizada porque poseerá como mínimo tres vértices virtuales. El primero, esta situado en un límite de la FPGA o en un bloque básico ocupado por una tarea; el segundo, pertenece a la tarea que ha originado la isla y aparece después de rodear toda la isla; y el tercero, coincide con el primer vértice virtual, y surge cuando se ha rodeado completamente la isla y se va a volver a una lista sin vértices virtuales inmediatos.

A continuación, una posible situación de isla con la que nos podemos encontrar junto con su conjunto de listas de vértices. Los vértices virtuales son (0, 2) y (2, 2).

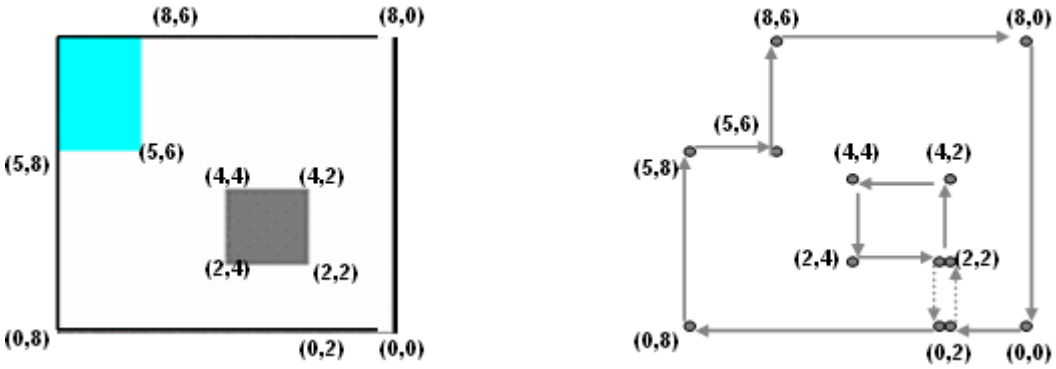


Fig. 5. Isla junto con su VLS

En el *Conjunto de Listas de Vértices*, se guardan todos los vértices que forman el perímetro del área libre de la FPGA. Sin embargo, para la colocación de tareas en la FPGA, no se consideran todos los vértices que haya en la lista de vértices, sino que sólo se consideran unos pocos, a los que se denominan *candidatos*. Un vértice es candidato, si provoca una forma cóncava. Los vértices virtuales no son candidatos.

El *Conjunto de Listas de Vértices* (VLS), hay que actualizarlo al insertar nuevas tareas, o cuando una tarea haya finalizado su ejecución. De esta forma, también se mantiene siempre actualizada la información del área libre que tenemos de la FPGA mediante la VLS.

Una vez que la inserción de una tarea T_N, en el hueco H_i ha sido confirmada, el *Selector de Vértices* devuelve el vértice candidato y los parámetros de la tarea al *Actualizador de Listas de Vértices*, para actualizar la forma del hueco, modificando la correspondiente VL_i.

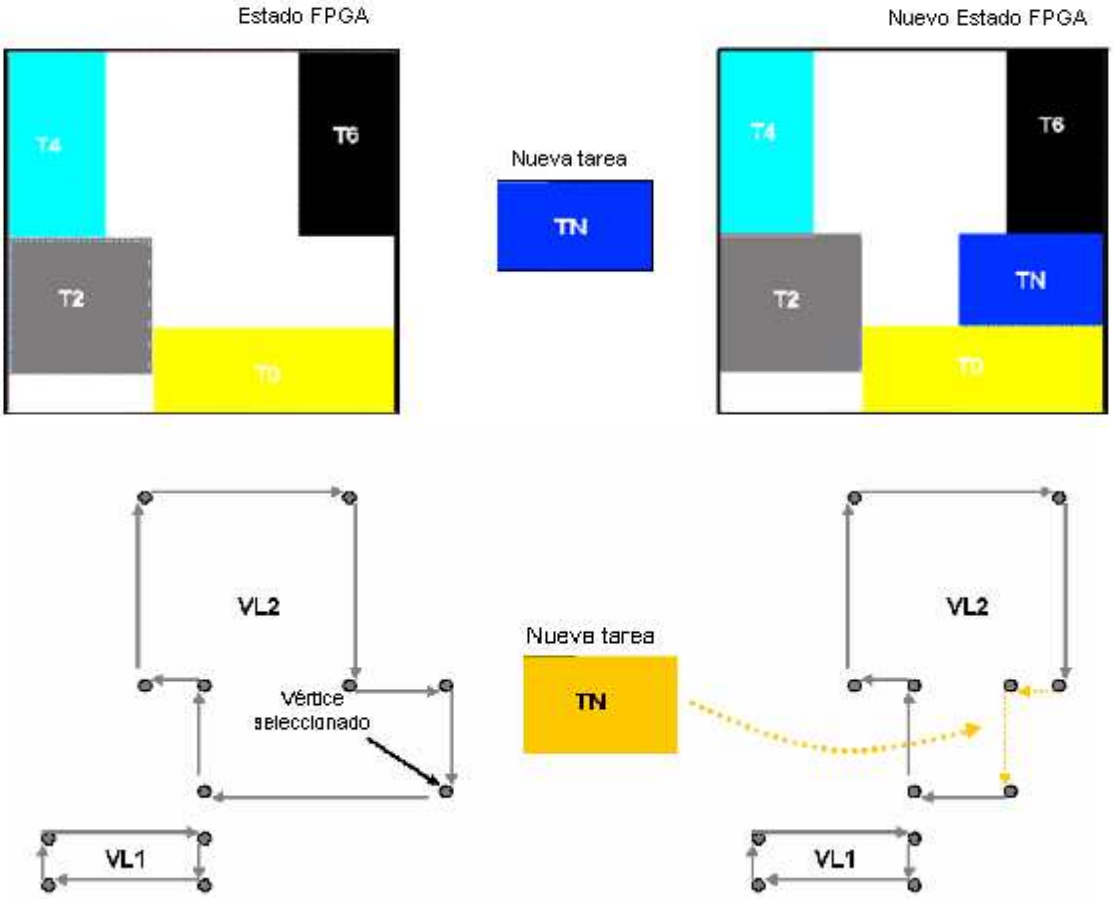


Fig. 6. Inserción de una tarea nueva

Al insertar una nueva tarea en la FPGA, se pueden crear dos huecos, como se muestra en la figura 7, y por tanto, hay que actualizar el *Conjunto de Listas de Vértices,* para que tenga dos *Listas de Vértices* distintas, donde cada una de ellas representa un hueco libre de la FPGA.

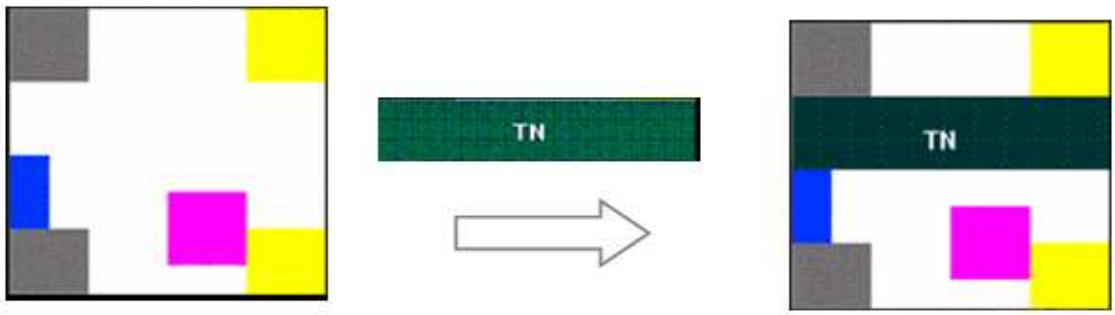


Fig. 7. Al insertar la tarea T_N, surgen dos huecos en la FPGA

Cuando el *Planificador de Tareas* detecta que una tarea finaliza su ejecución, extrae esta tarea de la *Lista de Tareas en Ejecución* y llama al *Actualizador de Listas de Vértices* para actualizar VLS. Varias situaciones pueden darse entonces, como que la tarea sea parte o no del límite del área libre definido por una VLi. Estas situaciones conviene estudiarlas por separado.

La primera situación es que podrían surgir islas, por lo que habría que gestionarlas mediante vértices virtuales.

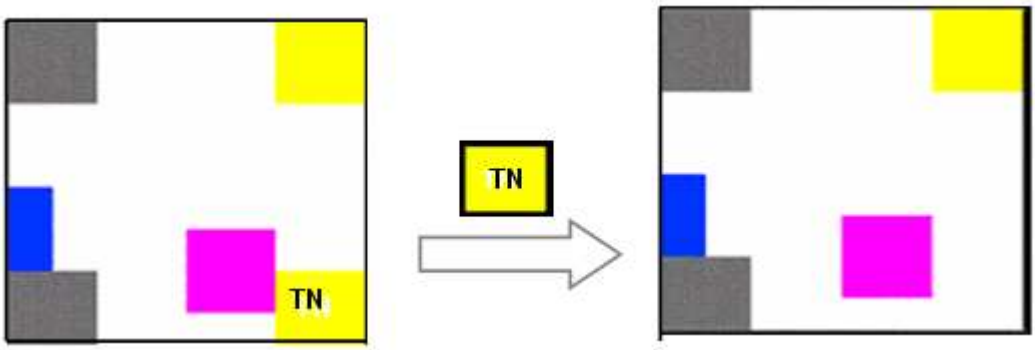


Fig. 8. Aparición de una isla al finalizar T_N

La segunda, consiste en la fusión de dos huecos existentes, lo que provoca que la *Lista de Vértices* contenga un hueco menos, y el que persiste, contenga los dos anteriores. Tanto en estas dos situaciones extraordinarias, como en otras más usuales, es necesario actualizar la VLS, para que siga reflejando la situación del área libre de la FPGA.

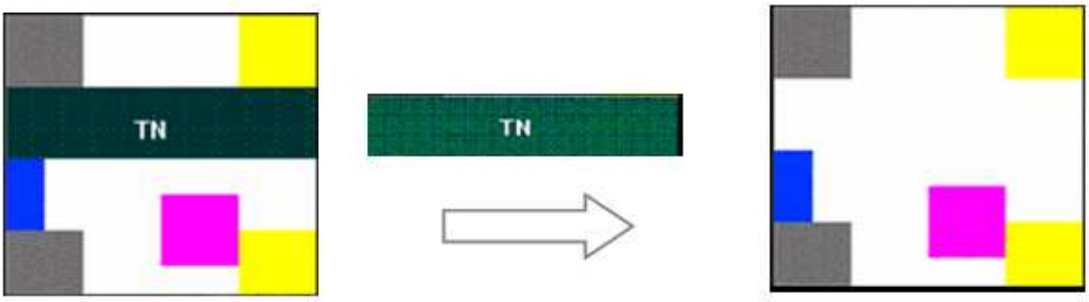


Fig. 9. Fusión de dos huecos al finalizar T_N

Cuando el *Actualizador de VL* detecta que una tarea que finaliza su ejecución, no es adyacente a ningún hueco existente, simplemente crea un nuevo hueco, añadiendo la nueva VL a la VLS, como se muestra en la figura 10.

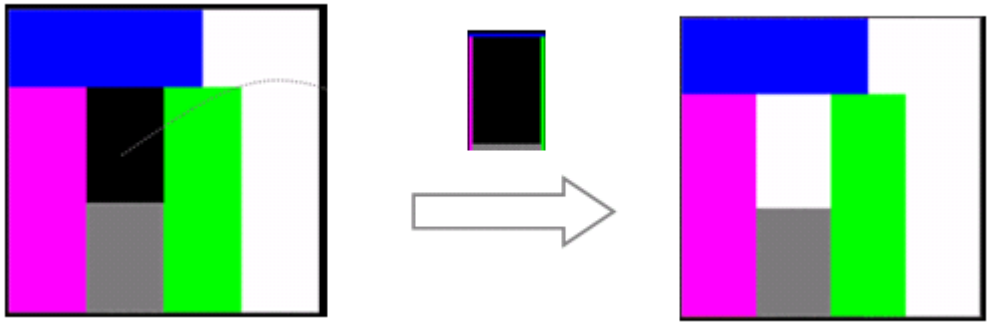


Fig. 10. Extracción de tarea que no está en los límites

Si el *Actualizador de VLS* encuentra un punto adyacente entre cualquier lado de un hueco y cualquier lado del perímetro del área ocupada por la tarea, entonces se añade la nueva área libre al hueco, actualizando la correspondiente VL de VLS. Un ejemplo de esta situación es la figura 11.

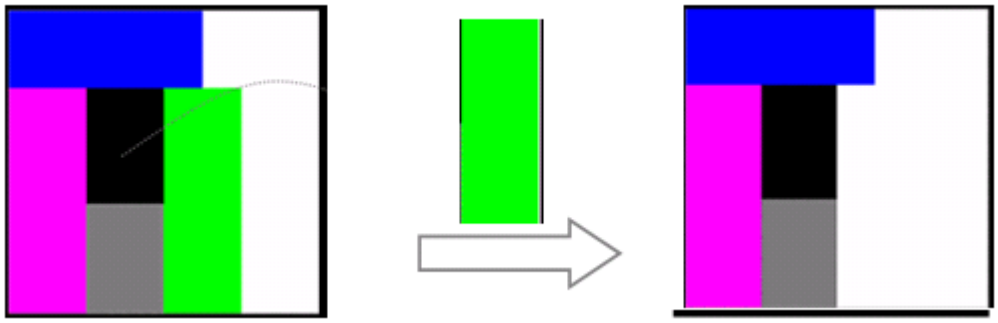


Fig. 11. Extracción de tarea que está en los límites

La arquitectura del Gestor HW se sustenta principalmente sobre tres estructuras de datos que son el *Conjunto de Listas de Vértices*, la *Lista de Tareas en Ejecución* y la *Lista de Tareas en Espera*.

3.4 Lista de Tareas en Ejecución

La *Lista de Tareas en Ejecución* es una lista formada por las tareas que actualmente están en ejecución dentro de la FPGA.

Esta lista es gestionada por el *Gestor HW*, de forma que cuando una tarea finaliza su ejecución, la tarea se elimina de la lista y se saca de la FPGA. Y cuando una tarea es insertada en la FPGA, se coloca al final de la *Lista de Tareas en Ejecución*.

Mediante esta estructura, se tiene información acerca de las tareas que hay en la FPGA, sin necesidad de tener que consultar la misma o la VLS. El Planificador para saber cuando una tarea finaliza su ejecución, tendrá que consultar esta lista, siendo un método más eficiente que el consultar la FPGA o la VLS.

3.5 Lista de Tareas en Espera

La *Lista de Tareas en Espera*, es una estructura de datos donde las tareas entrantes se almacenan cuando no hay espacio suficiente en la FPGA para su inserción inmediata.

Esta lista está ordenada por el t_timeout de las tareas, de forma que las tareas en espera que tienen menor t_timeout (que tienen más próximo el que no se cumplan las restricciones de tiempo necesarias para su ejecución), son las que se encuentran al principio de la lista.

Es consultada por el *Planificador de Tareas* para comprobar si puede colocar alguna de sus tareas, porque haya un nuevo espacio libre, y para verificar si alguna tarea ha alcanzado su t_timeout, en cuyo caso hay que descartarla. El Planificador también usa la *Lista de Tareas en Espera* para insertar en ella las tareas que no tenían una colocación posible en el instante en que llegaron.

Con esta estructura, se puede considerar en un futuro próximo tareas para las que en el instante de llegada no había hueco para ubicarlas. De esta forma, no se descartan recién llegadas si no se pueden colocar, sino que se guardarán en la lista y se intentarán colocar hasta que se cumpla su t_timeout. Obteniendo así un menor número de tareas rechazadas por el Gestor HW.

4. Arquitectura del Gestor

El gestor HW consta de los siguientes módulos:

- Planificador de tareas
- Selector de vértices (Vertex Selector)
- Actualizador de la Lista de Vértices (Vertex List Updater)

Estos módulos hacen uso de las siguientes estructuras de datos:

- Conjunto de Lista de Vértices (VLS), que describe todo el espacio libre de la FPGA.
- Lista de Tareas en Ejecución
- Lista de Tareas en Espera, donde las tareas entrantes se almacenan cuando no hay espacio suficiente en la FPGA para su inserción inmediata, y son ordenadas en orden creciente de t_timeout.

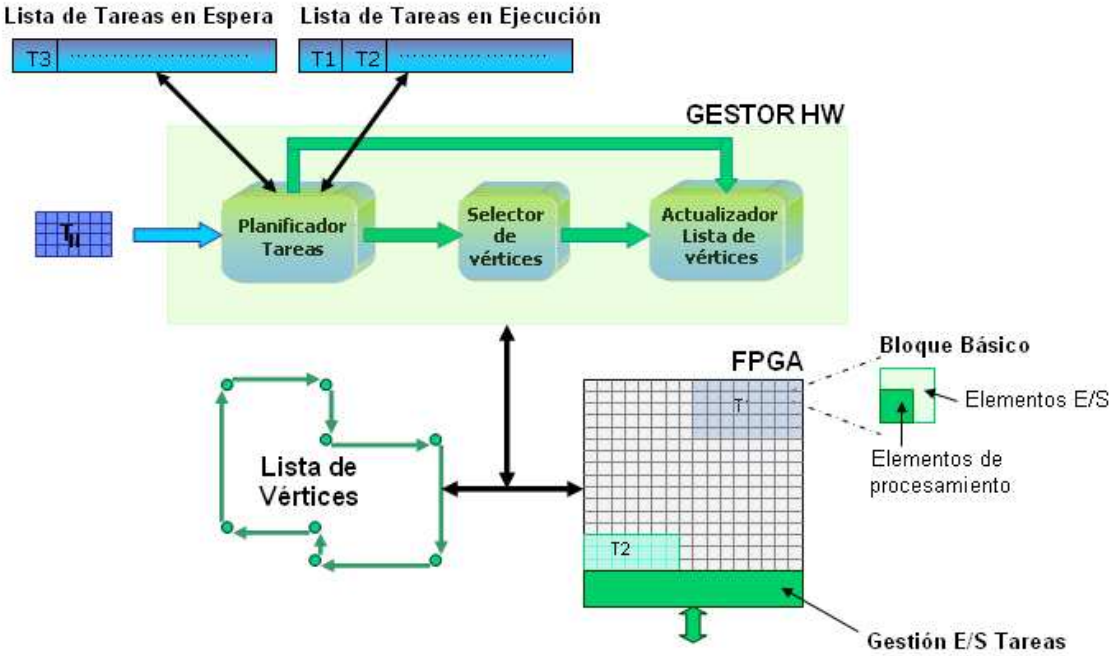


Fig. 12. Estructura del Gestor HW

Cuando una tarea T_N, que acaba de llegar o de la *Lista de Tareas en Espera*, es considerada, el *Planificador de Tareas* llama al *Selector de Vértices* para comprobar si hay alguna posición candidata donde se podría ubicar la tarea. La posición elegida está representada por un vértice, que es elegido de entre los candidatos de acuerdo con la heurística seleccionada. Una vez que se sabe en qué vértice colocar la tarea, la tarea se inserta en la FPGA, la VLS es actualizada por el *Actualizador de Listas de Vértices* y se inserta la nueva tarea en la *Lista de Tareas en Ejecución*. Si la tarea que acabamos de colocar en la FPGA, pertenecía a la *Lista de Tareas en Espera*, entonces habrá que eliminarla de la misma.

Si no se ha podido encontrar un candidato en ninguna VLi, donde la inserción de la tarea sea posible, es decir, si no hay hueco para la tarea T_N, entonces el *Planificador de Tareas* la almacena temporalmente en la *Lista de Tareas en Espera*. Esta lista está

ordenada en función del t_timeout (descrito en detalle en la siguiente sección 5.1.) de cada tarea.

Cuando una tarea termina su ejecución, y debe salir de la FPGA, la VLS es actualizada por el *Actualizador de Lista de Vértices*. Si el área liberada es adyacente a algún hueco existente, se actualiza la VLi correspondiente; y en caso contrario, se crea una nueva VL para describir este nuevo hueco.

Cada vez que una tarea finaliza, hay un incremento en el espacio libre disponible, y el *Planificador de Tareas* intenta insertar el máximo número de tareas de la *Lista de Tareas en Espera* como sea posible. Por tanto, cada vez que se extrae una tarea después de finalizar su ejecución, el *Planificador de Tareas* intenta primero colocar las tareas que están más cerca de llegar a su t_timeout de la *Lista de Tareas en Espera*.

Para actualizar la lista de vértices este módulo hace uso de la *Lista de Vértices* anterior, y de la *Lista de Tareas en Ejecución*.

4.1. Planificador de tareas

RESTRICCIONES TEMPORALES

En la planificación de las tareas, hay que tener en cuenta restricciones de espacio (que no haya hueco en la FPGA) y de tiempo, que son las que se explican en este punto.

Desde que una tarea llega al *Gestor*, en el instante *t_arr*, hasta que realmente se puede iniciar su ejecución (*t_start*), transcurre un intervalo de tiempo; el cuál se indica en la figura 13, en naranja y con el nombre de T1. Este intervalo de tiempo, se puede deber a restricciones de espacio dentro de la FPGA o a la estrategia que selecciona los vértices de ubicación, como por ejemplo sucede con Look-ahead. En este intervalo de tiempo, en el que la tarea ya ha llegado pero no se ha comenzado a ejecutar, la tarea se almacena en la *Lista de Tareas en Espera*; y permanecerá en esta lista hasta que o se cumpla su *time_out*, en cuyo caso hay que desechar la tarea, o se pueda ejecutar la tarea, ya sea porque hay hueco o por decisiones de la estrategia de selección de vértices.

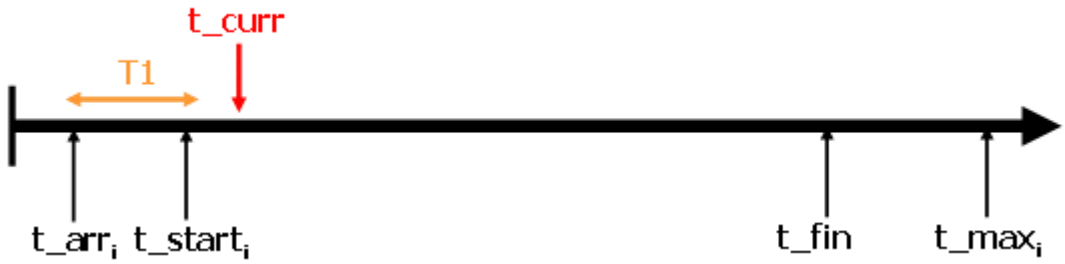


Fig. 13.

En la figura también se muestra *t_curr*, instante actual de la ejecución del *Gestor, t_fin*, instante en que finaliza la ejecución de la tarea, y por último, *t_max*, que es el tiempo máximo permitido para finalizar la ejecución de la tarea, que explicamos a continuación.

El tiempo necesario para cargar la configuración de la tarea es *t_conf*$_i$, que es proporcional a su tamaño. Este tiempo se puede ver, en el diagrama de la figura 15 en color verde.

$$t\_conf_i = k * w_i * h_i$$

Siendo w_i * h_i el tamaño de la tarea T_i. k es un factor de proporcionalidad, que depende del tamaño del bloque básico, las características del interfaz de configuración y la tecnología de la FPGA.

El tiempo en que una tarea finaliza su ejecución es *t_fin*, y se calcula como la suma del instante en que inicia su ejecución, más el tiempo que se tarda en configurar la FPGA para esa tarea, mas el tiempo de ejecución de la tarea.

$$t\_fin_i = t\_start_i + t\_conf_i + t\_ex_i$$

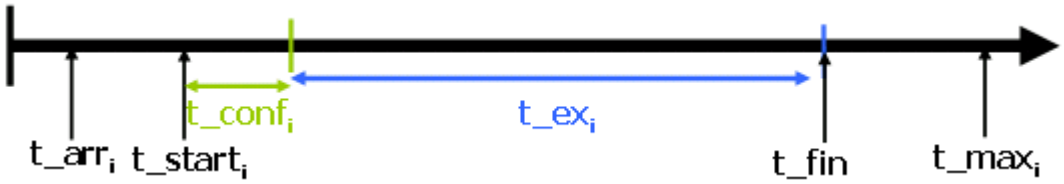


Fig. 14. Muestra de t_fin en un diagrama de tiempo.

El tiempo *t_rem*, es el tiempo que va a permanecer una tarea en la FPGA. *t_rem* se puede ver en la siguiente figura en amarillo, y se calcula según:

$$t\_rem_i = t\_start_i + t\_conf_i + t\_ex_i - t\_curr_i$$

donde *t_start*$_i$ es el instante de tiempo en el que comienza a ejecutarse la tarea T_i, y *t_curr*$_i$ es el instante de tiempo en que actualmente se encuentra la ejecución de todo el Gestor HW.

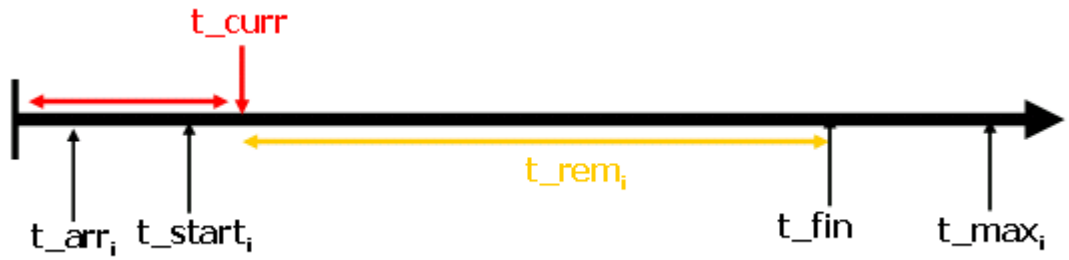


Fig. 15. t_rem de una tarea

El límite de tiempo en el que una tarea puede empezar a ejecutarse lo marca su *t_timeout*; puesto que una vez que una tarea supera su *t_timeout* ésta debe descartarse. La tarea que ha alcanzado su *time_out* ya no puede ejecutarse, por no cumplir las restricciones de tiempo impuestas. En la figura 16, se puede ver el *time_out* de una tarea en color morado.

$$t\_timeout_i = t\_max_i - t\_conf_i - t\_ex_i$$

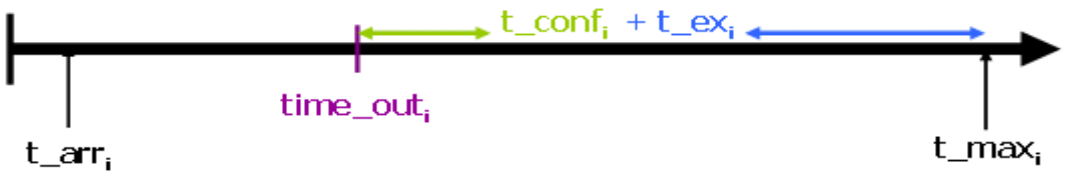


Fig. 16. time_out de una tarea.

PASOS DEL PLANIFICADOR DE TAREAS

El *Planificador de Tareas* sigue una serie de determinados pasos que se detallan en este apartado y se pueden ver en la figura 17.

Si termina una tarea, entonces la extrae de la *Lista de Tareas en Ejecución*, la saca de la FPGA y actualiza la VLS.

Si no ha terminado ninguna tarea, entonces pasa a gestionar la *Lista de Tareas en Espera*. Comprueba si alguna tarea ha alcanzado su t_timeout, en cuyo caso la descarta y la elimina de la lista. Si ninguna tarea ha alcanzado su t_timeout entonces verifica si alguna de las tareas de la lista se puede colocar en la FPGA, si se puede entonces la inserta en la FPGA y en la *Lista de Tareas en Ejecución*, la elimina de la lista de espera y actualiza la VLS.

Si no ha podido colocar ninguna tarea de la lista de espera, entonces comprueba si ha llegado alguna tarea nueva y aún no se ha ubicado.

Si ha llegado una tarea nueva, entonces si hay suficiente hueco en la FPGA entonces la coloca en la FPGA.

Si hay que insertar una tarea, tanto para el caso en que sea nueva como en el que procede de la Lista de Espera, se siguen las pautas de asignarle un tiempo de comienzo, se selecciona un vértice de entre los posibles candidatos, y por ultimo, se inserta la tarea en la lista de ejecución y se actualiza la VLS.

En la figura 17 se muestran, en general, los pasos que da el Planificador durante sus iteraciones.

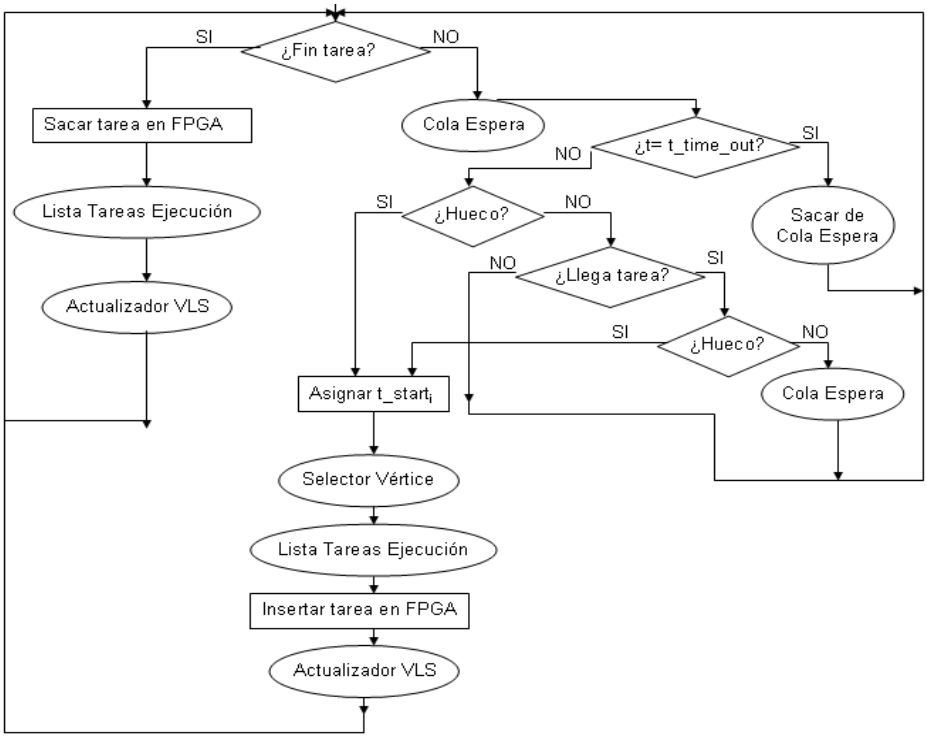


Fig. 17. Esquema de los pasos que sigue el Planificador de Tareas

4.2. Selector de vértices

El *Selector de Vértices,* surge como una solución al problema de seleccionar un vértice determinado, de entre todos los posibles candidatos, para ubicar una tarea dentro de la FPGA.

El *Selector de Vértices* para sus funciones consulta la VLS, a partir de la que obtiene una lista de candidatos. La lista de candidatos la forman vértices en los cuales haya hueco para colocar la tarea.

El *Selector de Vértices,* destaca un vértice candidato (de la lista de candidatos) del resto de acuerdo con una determinada función de coste seleccionada (explicada en el apartado 4).

Las funciones de coste que se han tratado en este proyecto, son First Fit y Best Fit.
First Fit (FF) es una aproximación muy sencilla, en la que se hace una búsqueda de un vértice candidato dentro de la VLS en el que coja la tarea. Se elige el primer vértice candidato en el que quepa, y se para de buscar, detallada en el punto 5.1.

El resto de heurísticas son de tipo Best Fit, que se caracterizan porque se recorre cada *Lista de Vértices* (VL) completamente, y se obtiene un valor para cada vértice candidato de acuerdo con la función de coste seleccionada. Finalmente, la tarea se colocará en el mejor de todos los vértices candidatos, es decir el que mejor valor tenga para la función de coste.

Las heurísticas Best Fit a su vez las podemos clasificar en dos tipos. El primer tipo es aquel en el sólo se tienen en cuenta consideraciones espaciales (2D) y a el pertenecen Adyacencia 2D y Fragmentación. En el segundo tipo también se tienen en cuenta consideraciones temporales (3D) y a el pertenecen Adyacencia 3D y Look ahead.

En la heurísticas 3D las tareas podrían ser consideradas como cajas 3D, en vez de rectángulos, como muestra la figura 18. Entonces, la función de coste basada en adyacencia 2D (explicada más adelante en el punto 5.2.) es fácilmente extensible a adyacencia 3D (comentada en el apartado 5.4.). Sin embargo, la aproximación 3D de la función de coste basada en fragmentación (apartado 5.3.) es difícil, porque debería de ser computada para demasiadas situaciones futuras. Por lo comentado, se optó por llevar la Adyacencia 3D, un paso mas allá en el siguiente instante de tiempo (cuando finalice la siguiente tarea), y a esta función de coste, en este caso, se la denomina Look-ahead (punto 5.5).

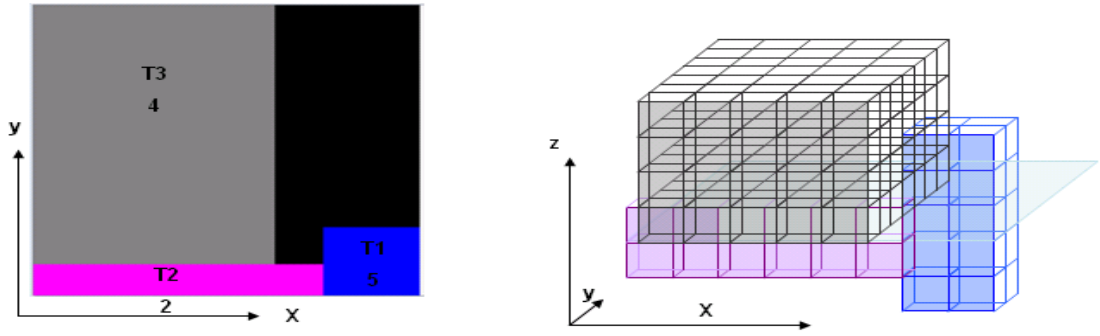


Fig. 18. Representación del mismo estado de FPGA en 2D y 3D

5. Heurísticas

El *Selector de Vértices* es el que realiza la elección del mejor vértice para ubicar la tarea; para hacer esta elección se basa en heurísticas.

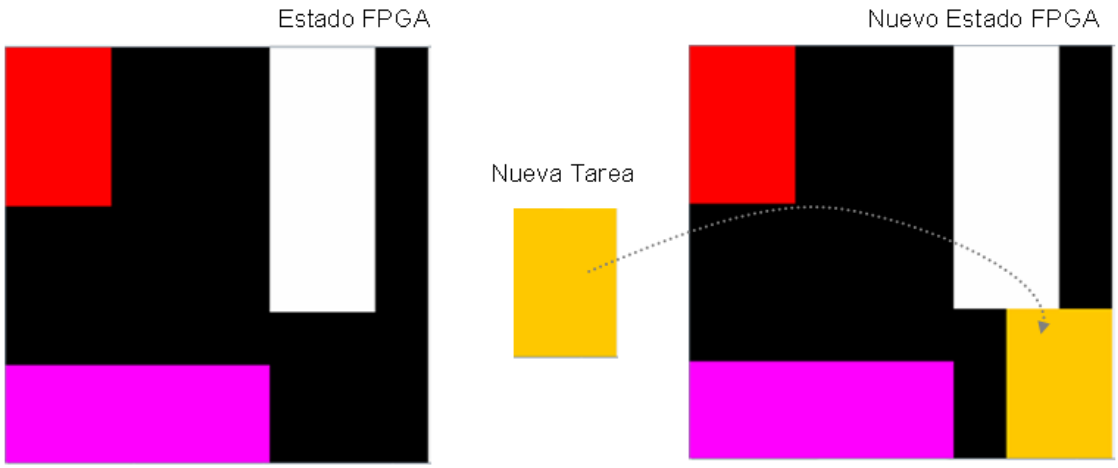
Para elegir la mejor ubicación entre las candidatas de la VLS se proponen varias heurísticas. Además, también se propone una heurística Look-ahead, que retrasa la ubicación hasta el siguiente evento conocido (punto 4.5.).

Las heurísticas que se han tratado en este proyecto se pueden clasificar como First Fit y Best Fit; dentro de las Best Fit hay otros dos tipos, el que solo tiene en cuenta consideraciones espaciales (Adyacencia 2D y Fragmentación) y el que además tiene en cuenta consideraciones temporales (Adyacencia 3D y Look ahead).

5.1. First Fit

FF es una aproximación muy sencilla, en la que se hace una búsqueda de un vértice candidato dentro de la VLS en el que quepa la tarea. Se elige el primer vértice candidato en el que se pueda colocar y se para de buscar.

La figura 19 muestra un ejemplo simple para ilustrar esta función de coste. El estado de la FPGA, es mostrado en la parte de arriba, con tres tareas ejecutándose en ese momento; además en la parte superior también se puede ver la correspondiente VL. Cuando una nueva tarea de 3*2 bloques llega, esta aproximación busca entre las posibles posiciones candidatas. Usando el criterio FF la tarea se ubicaría en el primer candidato, que además es el primero en el que se puede colocar, y finaliza la búsqueda.



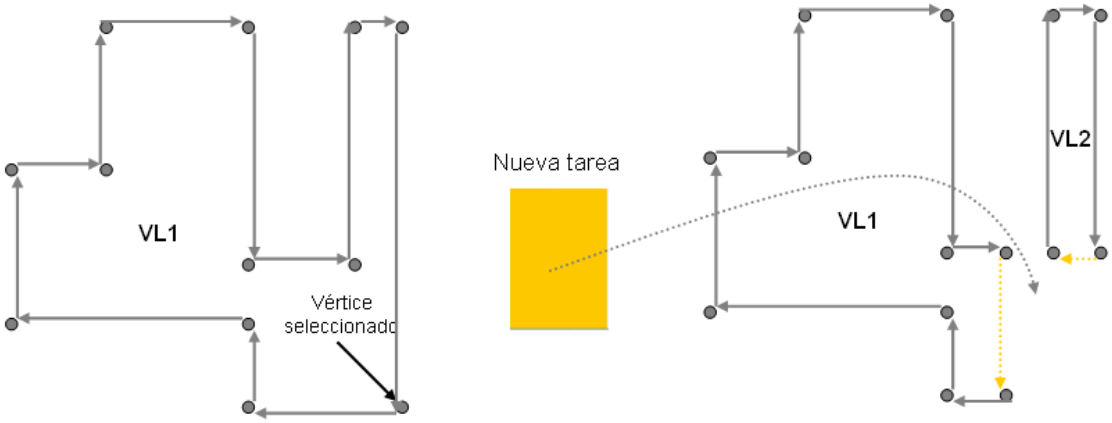


Fig. 19. Inserción de tarea con FF

Como se puede deducir del ejemplo mostrado en la figura anterior, con FF la ubicación de la tarea no es nada buena, puesto que la selección del primer vértice ha dado lugar a que haya dos huecos en la FPGA, desaprovechándose así espacio; lo cual da lugar a proponer otras heurísticas.

5.2. Adyacencia 2D

Adyacencia 2D, es una heurística Best-Fit. Esta función de coste recorre cada VL completamente y obtiene un valor para cada vértice candidato, de acuerdo con su valor de adyacencia. Finalmente la tarea se colocará en el mejor de todos los vértices candidatos, es decir, el que mayor valor de adyacencia tenga.

Es una forma de obtener una función de coste sencilla. La idea es insertar la tarea en el vértice donde se alcance el mayor contacto entre los bordes de tareas y los limites del área libre, definida por los diferentes lados de cada VL.

Esta adyacencia 2D es computada para cada vértice, en términos de unidades de longitud de bloques básicos como:

$$Ady\text{-}2D_j = \Sigma_h \, (VL\_lado_h \cap T_N\_limite)$$

donde T_N es la tarea a insertar, $Ady\text{-}2D_j$ es el valor de adyacencia 2D para el vértice V_j y VL_h es la VL que estamos considerando en ese momento.

La figura 20 se muestra un ejemplo simple de un posible estado de FPGA. Cuando, con este estado, llega una nueva tarea de 2*2 bloques, esta aproximación calcula el valor de adyacencia para cada posible vértice candidato, como se puede ver en la figura 21. Usando el criterio de adyacencia, la tarea se ubicaría en el séptimo u octavo candidato, donde el valor de adyacencia es 6.

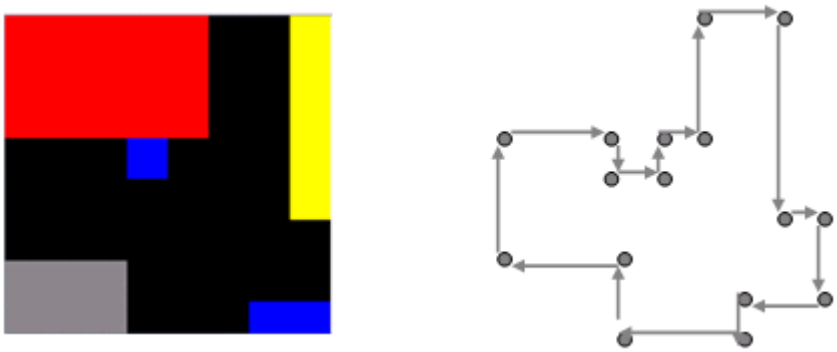


Fig. 20.

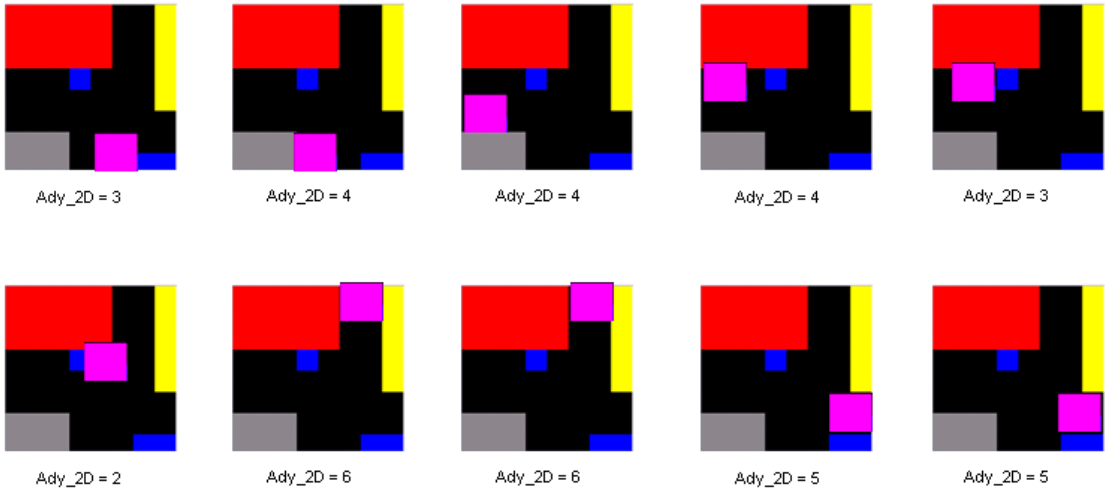


Fig. 21. Valores de adyacencia 2D

5.3. Fragmentación

La heurística basada en fragmentación, es una heurística Best-Fit, así que recorre cada VL completamente y obtiene un valor para cada vértice candidato, de acuerdo con su medida de fragmentación. Finalmente, la tarea se colocará en el mejor de todos los vértices candidatos, es decir el que menor medida de fragmentación tenga.

La métrica para estimar el estado de fragmentación de la FPGA, se basa en el número de huecos y su forma. Esta métrica es muy útil, porque el estado de fragmentación es una medida indirecta de la probabilidad de encontrar una localización posible para una nueva tarea en la FPGA en un futuro próximo.

Se puede tener una estimación del estado de fragmentación de la FPGA usando una métrica de fragmentación, similar a la propuesta en [15], donde el nivel de fragmentación de un área libre para un estado de FPGA dado se estima como sigue:

$$F = 1 - \prod_i [\, (4/V_i)^n * (A_i / A_{F\_FPGA})]$$

Donde el término entre corchetes representa la idoneidad de un hueco dado H_i con área A_i y número de vértices del hueco V_i:

- $(4/V_i)^n$ representa la idoneidad de la forma del hueco i para acomodar tareas rectangulares. Obsérvese que cualquier hueco con cuatro vértices tiene la mejor forma. Para la mayoría de experimentos se emplea n = 1, pero se

podría usar mayor o menor valor de n si se quiere penalizar más o menos la ocurrencia de huecos con formas complejas y así dificultar su uso.

- (A_i / A_{F_FPGA}) representa el área relativo normalizado del hueco. A_{F_FPGA} es el número de bloques básicos libres de la FPGA, y A_i es el número de bloques básicos libres de ese hueco. Nótese que si toda el área libre pertenece al mismo hueco entonces este término es igual a uno.

Como se puede ver, cuantos más huecos separados de área sin ocupar tenga la FPGA menos capacidad global tendrá la FPGA de colocar tareas. El valor de F crece rápidamente con el número de huecos.

La figura 22 muestra diferentes situaciones con los valores de fragmentación dados por esta métrica.

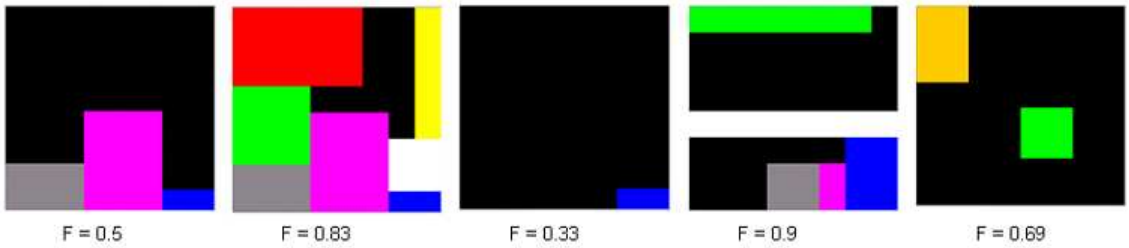


Fig.22. Diferentes situaciones de FPGA y valores de fragmentación.

La heurística basada en fragmentación consiste en elegir la métrica que se acaba de detallar como función de coste para selección de vértices. Esta heurística estima la fragmentación producida en el área libre de la FPGA para cada posible vértice candidato, y finalmente inserta la tarea en el vértice donde menor nivel de fragmentación se produzca. Así, se tiene una forma del área libre de FPGA más adecuada para acomodar futuras tareas, evitando huecos con formas complejas y la proliferación del número de huecos.

En la figura 23, se muestra un posible estado de ocupación de la FPGA. Cuando una nueva tarea de 2*2 bloques básicos llega, el nivel de fragmentación es calculado para todos las posibles localizaciones. El segundo vértice candidato provoca la menor fragmentación del área libre resultante, como se puede ver en la figura 24, con un valor de 0.67 y se elige como vértice para ubicar la tarea recién llegada.

Fig. 23.

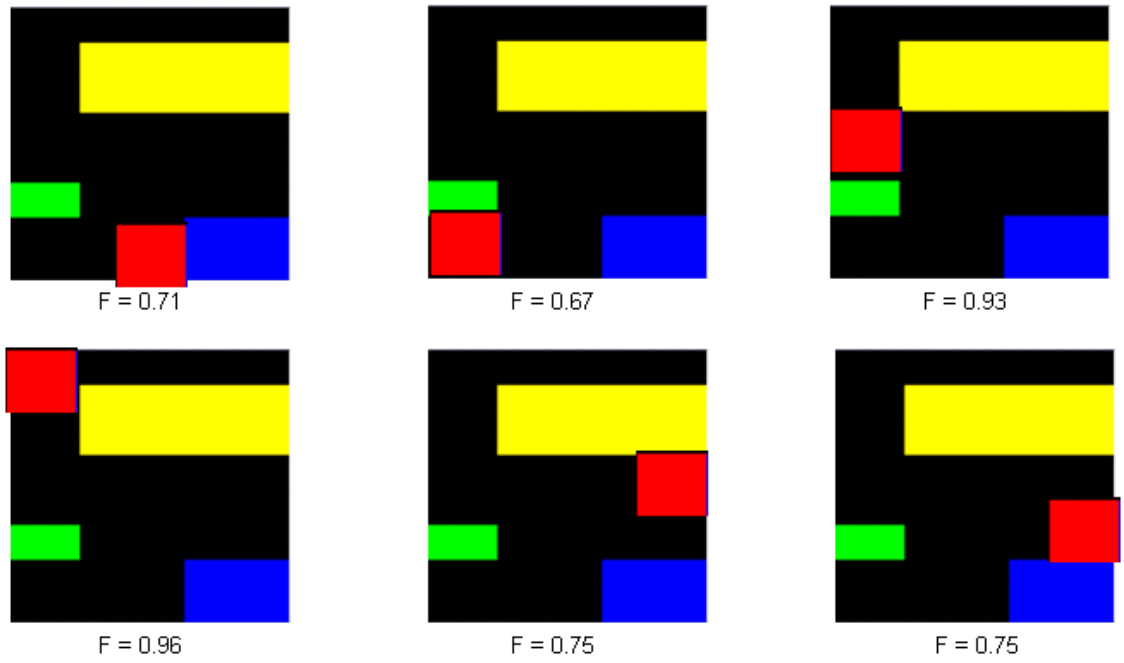


Fig. 24.

5.4. Adyacencia 3D

Esta heurística es una mejora de la heurística de Adyacencia 2D descrita en la sección 5.2. Considerando la FPGA como una malla y las tareas como cajas más pequeñas, con el tiempo como eje vertical, el algoritmo intenta apilar la caja correspondiente a la nueva tarea tan cerca como sea posible del resto de las cajas que ya hay en la FPGA o de los límites de la FPGA. Esta heurística es similar a una que podríamos usar en la vida real para colocar cajas reales dentro de una malla real.

Para tener el tiempo en cuenta, es necesario almacenar el nuevo valor para cada lado de la lista de vértices: el valor t_rem de la tarea del lado al que pertenece, un tipo de tiempo de vida del lado. Para simplificar los cálculos, si un lado pertenece a varias tareas contiguas, la que tenga menor t_rem es elegida como tiempo de vida para el lado completo.

La adyacencia 3D es computada para cada vértice j como la longitud del límite de una nueva tarea en contacto con cada lado h adyacente de VL, multiplicado por la adyacencia temporal entre la nueva tarea y el lado de VL. Esta adyacencia temporal es el mínimo entre el tiempo de vida del lado y el t_ex de la tarea que acaba de llegar:

$$Ady\text{-}3Dj = \Sigma_h \, (VL\_lado_h \cap T_N\_limite) * min \, (t\_rem\_lado_h, \, t\_ex_N)$$

Esta aproximación se puede ver en la figura 25, cuando T_N con t_ex = 5 y mostrada en la figura de color gris, es colocada cerca de las tareas que se están ejecutando, la adyacencia 3D es 45 (5*4 + 5*5), mientras que el valor de adyacencia 2D para la misma FPGA y distribución de tareas es de 9 (4+5). Las tareas que se están ejecutando son en azul {2, 2, 4,1, 10} y en rosa {6, 1, 3, 2, 10}, y t_curr = 3 (instante en el que llega T_N).

Debe observarse que para el perímetro de la FPGA, el tiempo de vida del lado es considerado como infinito, así t_ex_N es utilizado. Entonces, cuando sea posible en el

instante actual, la tarea se colocará preferiblemente cerca de los bordes de la FPGA, o cerca de otras tareas de larga vida.

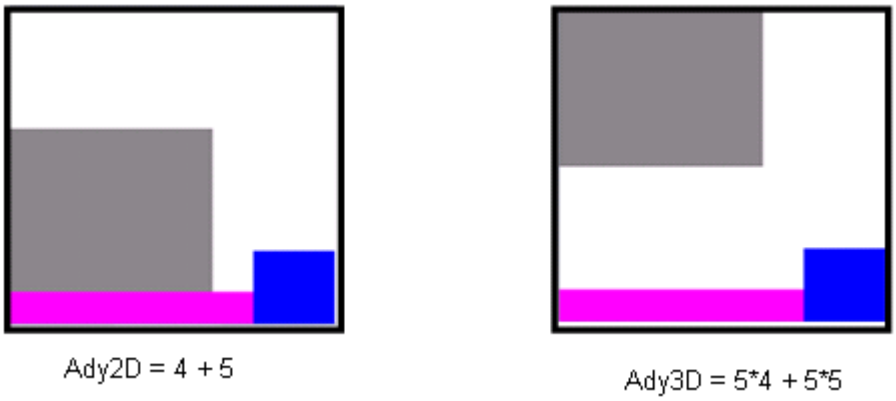


Fig. 25. Ejemplos de adyacencia 2D y adyacencia 3D

La figura 26 muestra un ejemplo usando una FPGA 8*8 con tres tareas ejecutándose, t_rem tiene un valor para todas de 1. Cuando una tarea T_N llega, con tamaño de 2*2 celdas básicas y un valor de t_ex de 2, el Gestor HW prueba a planificarla en el instante actual t_curr. Los valores de adyacencia 3D son calculados para todos los vértices candidatos como se muestra en la figura 34. Entonces la nueva tarea, será insertada en el segundo vértice candidato, donde el valor de adyacencia 3D es el máximo.

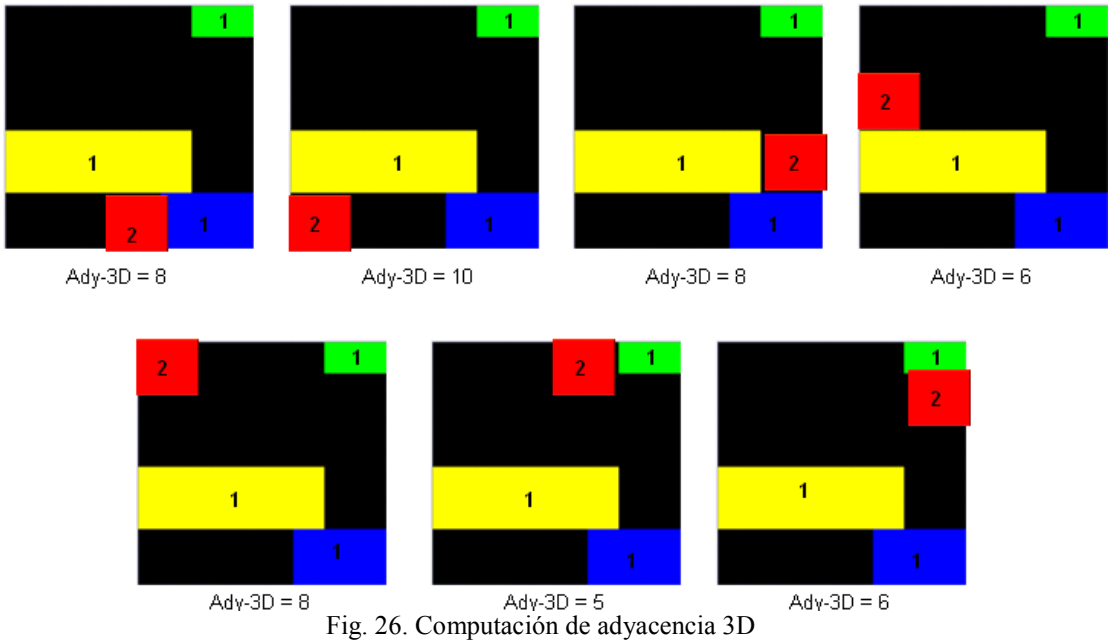


Fig. 26. Computación de adyacencia 3D

5.5. Look-ahead

Esta heurística utiliza el valor de adyacencia 3D, computado como se ha descrito anteriormente. En esta aproximación el valor es calculado para todos los posibles candidatos, en el instante actual t_curr y en el próximo evento de tiempo conocido (cuando ocurra el próximo fin de una tarea). La idea es que se podría obtener un valor de adyacencia 3D mejor para una localización sólo disponible en un próximo futuro.

A veces no es posible utilizar esta función de coste, si la nueva tarea tiene que ser planificada antes de que cualquier tarea pueda salir de la FPGA, debido a su valor t_max.

En la figura 27, se muestran cuatro tareas en ejecución con las siguientes características: en azul {2,2,4,1,10}, en rosa {2,2,1,1,10}, en gris {2,2,2,1,10} y en verde {2,2,4,1,10}. En el instante t = 1, llega la tarea en color negro con los siguientes parámetros {5, 2, 6, 1, 15}. La tarea espera hasta el siguiente evento (t=2), en el que ha salido la tarea en color rosa y se obtiene mejor valor de adyacencia 3D que cuando llego.

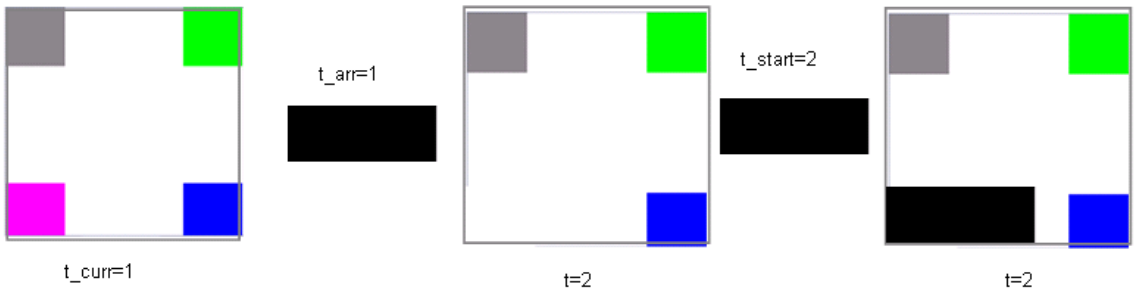


Fig. 27. Ejemplo Look ahead

En la figura 28, se muestran los valores de adyacencia 3D al lado de cada uno de los vértices candidatos en el instante t=1 y t=2.

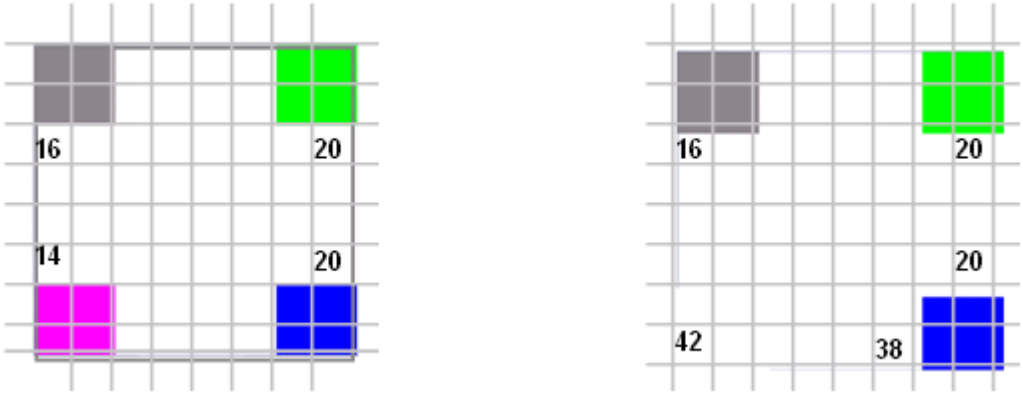


Fig. 28. Valores de adyacencia 3D en el instante actual y en el siguiente evento.

5.6. Comparación

En este apartado, vamos a explicar gráficamente las diferencias que existen entre las estrategias de selección de vértices (heurísticas).

Partiendo del siguiente fichero de entrada de tareas:
2 2 4 1 10
6 1 3 2 10
5 4 5 3 20

Hacemos cinco ejecuciones del mismo variando la heurística de selección. Si tomamos First Fit, en el instante de ejecución podremos ver el siguiente estado de FPGA.

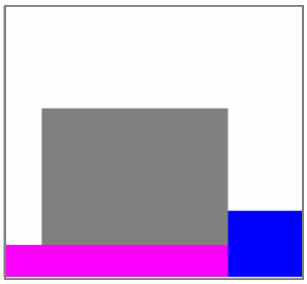


Fig.29.

Si elegimos la heurística basada en fragmentación la situación será la reflejada en la figura 30 y si tomamos adyacencia 2D hay que observar la figura 31.

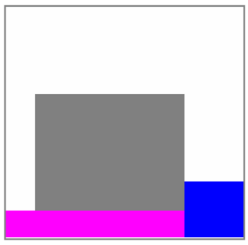


Fig.30.

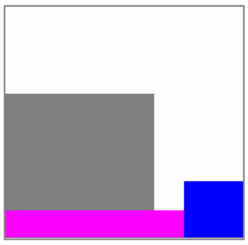


Fig.31

Si el usuario escogiera cualquiera de las heurísticas 3D (Adyacencia 3D y Look ahead), vería el estado de la FPGA de las figuras 32 y 33.

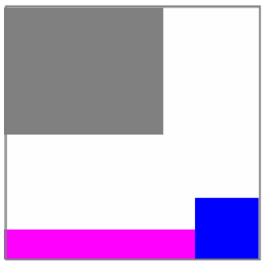


Fig.32.

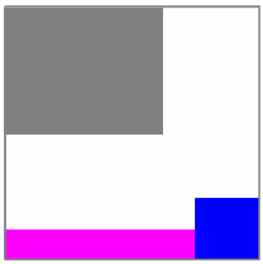


Fig.33.

Analizando más a fondo las figuras de las ejecuciones se puede ver que, en este caso no hay diferencias entre First Fit y fragmentación, puesto que ambas provocan el mismo número de vértices en la nueva VLS y no originan ningún hueco nuevo.

En la figura 31, se puede ver como la heurística de adyacencia 2D ha colocado la tarea pegada al límite inferior izquierdo, donde tiene un valor máximo de adyacencia 2D. Con respecto a las figuras 29 y 30, se puede ver como da la sensación de que la tarea en color gris parece que estorba menos en la figura 31 que en la 29 y la 30.

Las heurísticas 3D han colocado ambas la tarea en la esquina superior derecha, donde no va a depender de la ejecución de otras tareas para seguir conservando su adyacencia.

Con respecto a FF y a fragmentación, como sucedía con adyacencia 2D, las heurísticas 3D originan un mejor resultado; y con respecto a adyacencia 2D también mejoran el resultado, debido a que cuando termine la ejecución de la tarea en color rosa en adyacencia 2D se habrá perdido gran parte del contacto con otras tareas, mientras que en las basadas en 3D en el siguiente evento la adyacencia se seguirá conservando.

Si ejecutamos el fichero de tareas que se muestra en la figura 34 con cada una de las heurísticas disponibles podremos observar los estados de la FPGA que se detallan en la figura 35 (FF, fragmentación y Adyacencia 2D), en la figura 36 (heurística basada en Adyacencia 3D) y en la figura 37 (Look ahead).

Fig. 35. Fichero de tareas de entrada.

En la siguientes figuras (35, 36 y 37) se puede observar el estado de la FPGA en la ejecución del fichero de la figura anterior eligiendo distintas heurísticas, desde que comienza a ejecutarse a última tarea en llegar hasta que finaliza su ejecución la ultima.

En la figura 35, se puede observar que aunque a priori la colocación de las tareas, usando las heurísticas de FF y basadas en 2D, pueda parecer buena, a lo largo de su ejecución el hueco que va resultando en cada instante tiene una forma que no facilitaría la inserción de nuevas tareas.

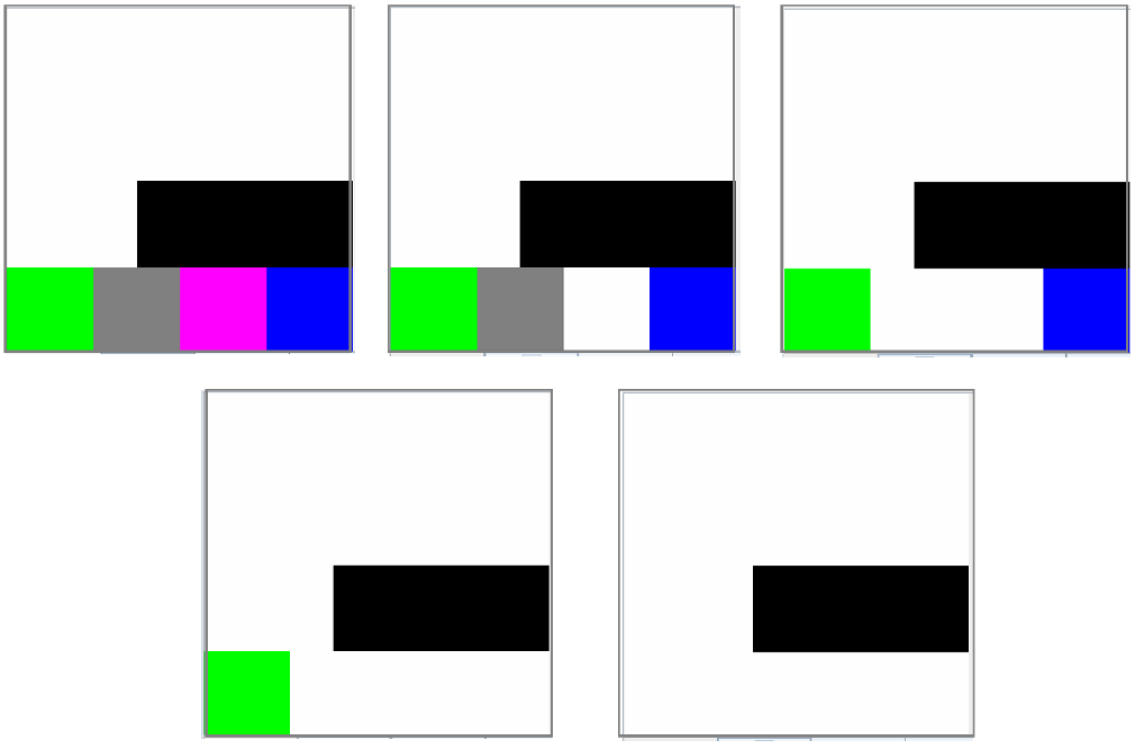


Fig. 35. Ejecución del fichero mostrado en la figura 34, escogiendo como heurística FF, fragmentación y Adyacencia 2D.

En la figura 36, se puede ver que en este ejemplo, la heurística basada en Adyacencia 3D ocasiona desde el principio de la ejecución, una forma de hueco que no es demasiado propicia para la inserción de nuevas tareas.

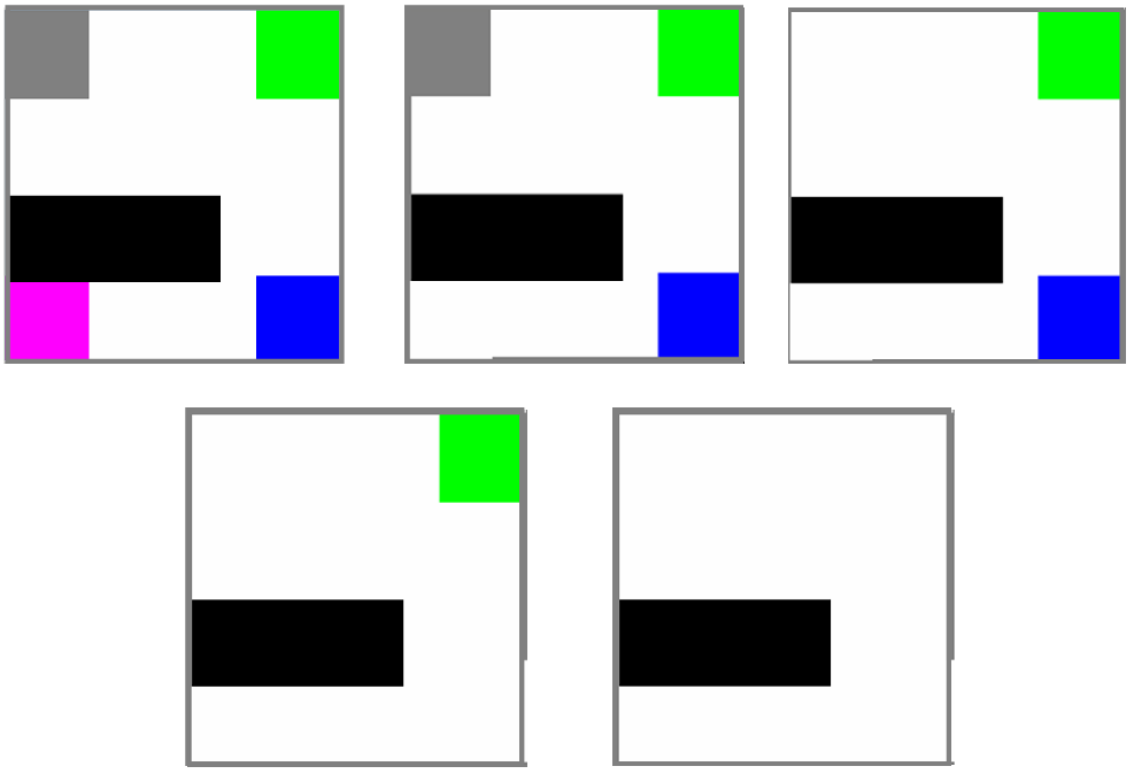


Fig. 36. Ejecución del fichero mostrado en la figura 34, escogiendo como Adyacencia 3D.

En la figura 37, se puede observar que con la heurística basada en Look ahead el hueco de la FPGA durante la ejecución es bastante bueno para meter tareas nuevas de distintos tamaños.

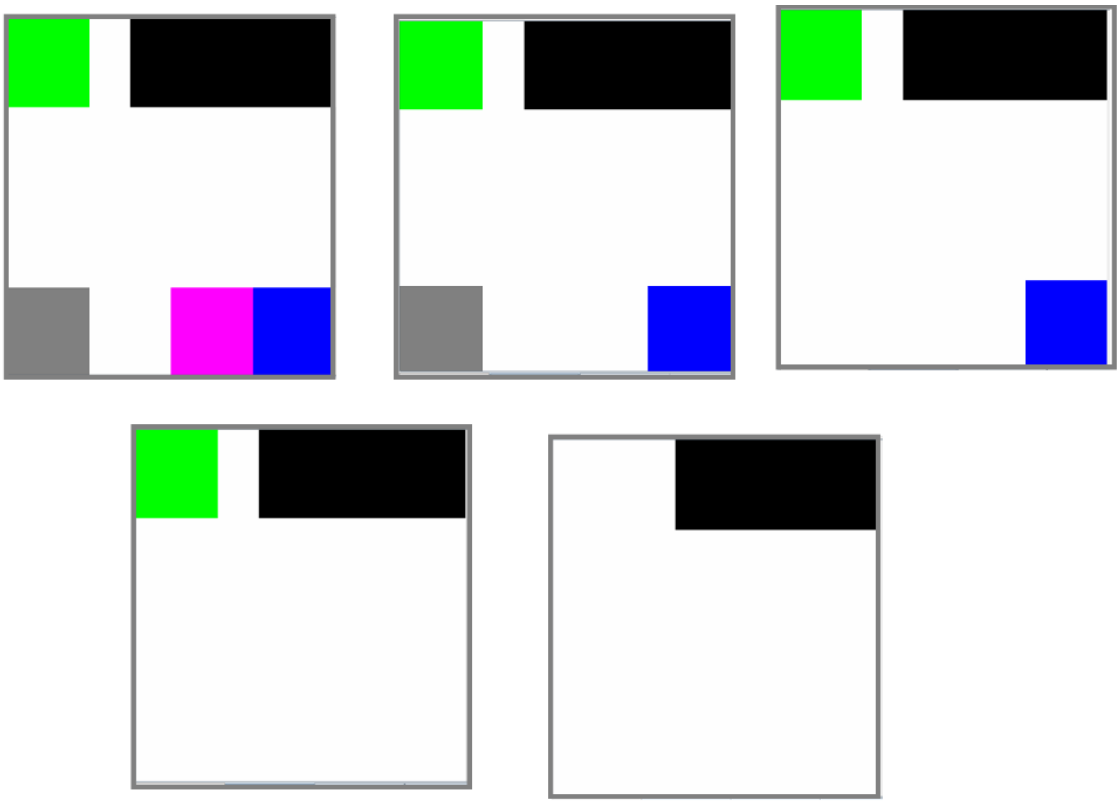


Fig. 37. Ejecución del fichero mostrado en la figura 34, escogiendo como Look ahead.

6. Resultados de pruebas

Los experimentos realizados han tenido lugar con … ficheros de entrada, los cuales han sido ejecutados con cada una de las heurísticas.

Las tareas que se han utilizado para las pruebas tienen un tamaño w*h, siendo w y h la anchura y altura respectivamente de la tarea. Tanto w como h pertenecen a un rango que oscila entre 1 y 100 bloques.

Las características que se han tenido en cuenta para las pruebas han sido:

- Numero tareas ejecutadas
- Numero tareas rechazadas
- % ocupación
- Volumen rechazado.

Para calcular la ocupación he hecho uso de la siguiente fórmula:

$$\%\ ocupación = \Sigma_i(h_i * w_i) / (t\_ex\_total * area_{FPGA})$$

Donde i pertenece al rango de número de tareas ejecutadas. w_i y h_i son respectivamente la anchura y la altura de la tarea T_i. t_ex_total es el instante en que terminan todas las tareas de ejecutarse y no queda ninguna por tratar y $area_{FPGA}$ es el tamaño de la FPGA ($filas_{FPGA}$ * $columnas_{FPGA}$).

Para calcular el volumen rechazado durante una ejecución utilizo:

$$Volumen\_rechazado = \Sigma_i((h_i * w_i) * t\_ex_i)$$

Donde h_i y w_i son la altura y la anchura de la tarea T_i, respectivamente. Y t_ex_i es el tiempo de ejecución de la tarea T_i.

En la figura 38 se puede ver una grafica comparativa entre las heurísticas, en cuanto a tareas ejecutadas se refiere. Se puede observar que las que mejores resultados ofrecen son Adyacencia 2D, Adyacencia 3D y Look ahead.

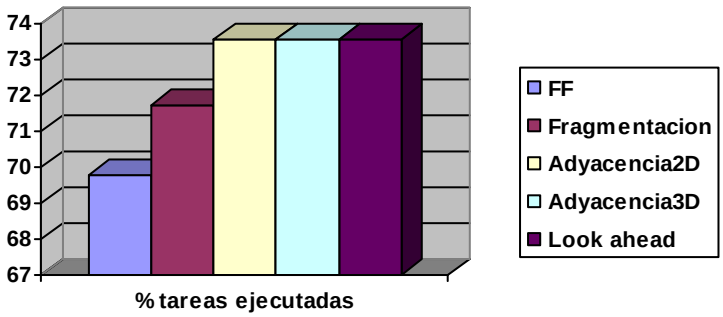


Fig. 38

En la siguiente figura se ofrece una grafica comparativa de la cantidad de tareas rechazadas por cada una de las heurísticas. De este dibujo se desprende igual conclusión que de a grafica anterior que es que las que peores resultados dan son FF y fragmentación.

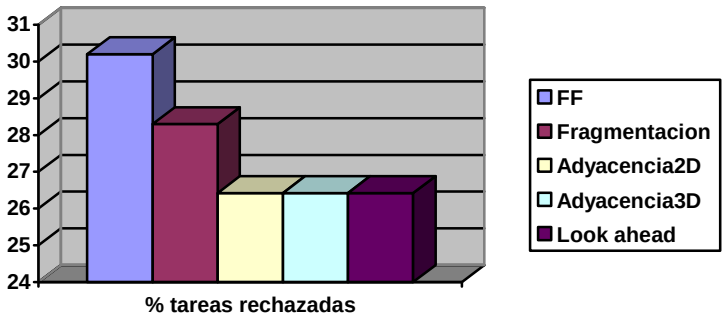


Fig. 39

En la figura 40, se muestra la comparación del % de ocupación durante toda la ejecución, y se puede ver que la que mayor valor de ocupación obtiene es fragmentación.

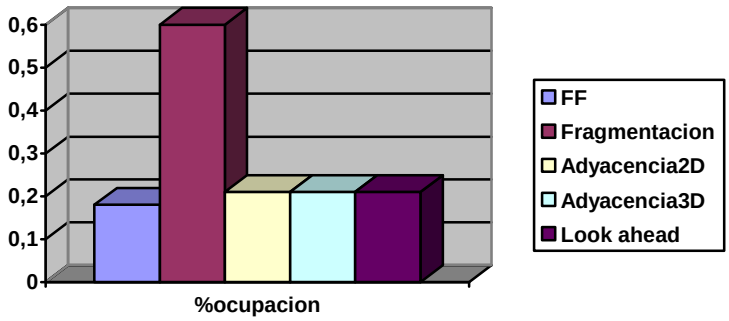


Fig 40.

En la ultima figura se puede ver el volumen de las tareas rechazadas por cada una de las heurísticas, siendo notablemente mejor el resultado obtenido por Look ahead.

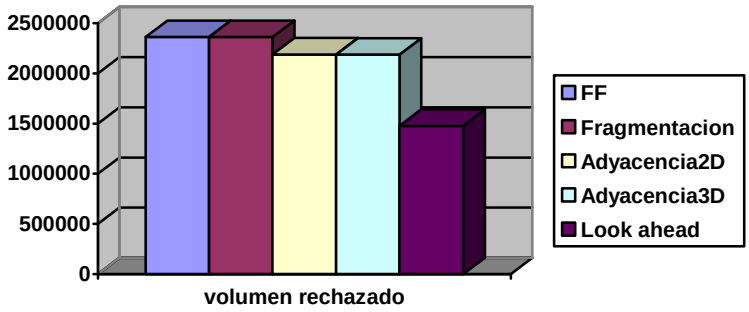


Fig. 41

7. Conclusiones

Se ha presentado una nueva solución para el problema de colocación en el HW multitarea, con un Gestor de Área que usa un nuevo enfoque a la inserción de tareas, basado en una estructura de listas de vértices. Las localizaciones factibles para inserción de tareas se buscan en esta estructura, considerando vértices candidatos con diferentes heurísticas para la inserción.

Esta estructura del Conjunto de Listas de Vértices contiene un reducido de localizaciones candidatas, y es más fácil explorar y mantener que otras estructuras previamente propuestas.

Se han desarrollado varias heurísticas para seleccionar uno de los vértices para ubicar la tarea. La mejor de las heurísticas de selección de vértices es Look ahead, debido a que se beneficia de las ventajas de la basada en Adyacencia 3D y además tiene en cuenta el futuro próximo.

Como trabajo futuro nos planteamos la integración del Gestor con un módulo de defragmentacion, de tal forma que el gestor invoque subrutinas de defragmentacion, y la mejora de la heurística de fragmentación, puesto que la empleada en este trabajo no da tan buenos resultados como se esperaba.

8. Referencias

[MCSG2000] McMillan, S. Guccione, "Partial Run-Time Reconfiguration Using JRTR", FPL 2000, pp. 352- 360. Springer-Verlag, Berlin, August 2000.

[BCAA2005] Bobda, C., Ahmadinia, A., "Dynamic interconnection of reconfigurable modules on reconfigurable devices", IEEE Design &test of Computers, 2005, 22 (5), pp 443-451.

[BKKS2000] Bazargan, K., Kastner, R., Sarrafzadeh, M., "Fast Template Placement for Reconfigurable Computing Systems", IEEE Design and Test of Computers, 2000, 17(1), pp. 68–83.

[TSMM2004] Tabero, J., Septién, J., Mecha, H., Mozos, D., "A Low Fragmentation Heuristic for Task Placement in 2D RTR HW Management", Proc. Int. Conf. On Field-Programmable Logic and Application, Lecture Notes in Computer Science, vol 323. Springer-Verlag, 2004, pp. 241-250.

9. Indice de figuras

APÉNDICES

A.FPGA

FPGA es el acrónimo de Field Progammable Gate Array (matriz de puertas programable).

Las FPGAs pertenecen al conjunto de hardware reconfigurable, es decir, el dispositivo se recibe como un circuito fabricado cuya funcionalidad podremos programar. Existen otros dispositivos electrónicos digitales programables de muy alta densidad como son PLA, PROM, PAL, CPLD…

Los elementos básicos que forman parte de una FPGA son:

- CLB, bloque configurable lógico (configurable logic block).
- IOB, bloque de entrada/ salida (input /output block)
- Bloques de interconexión

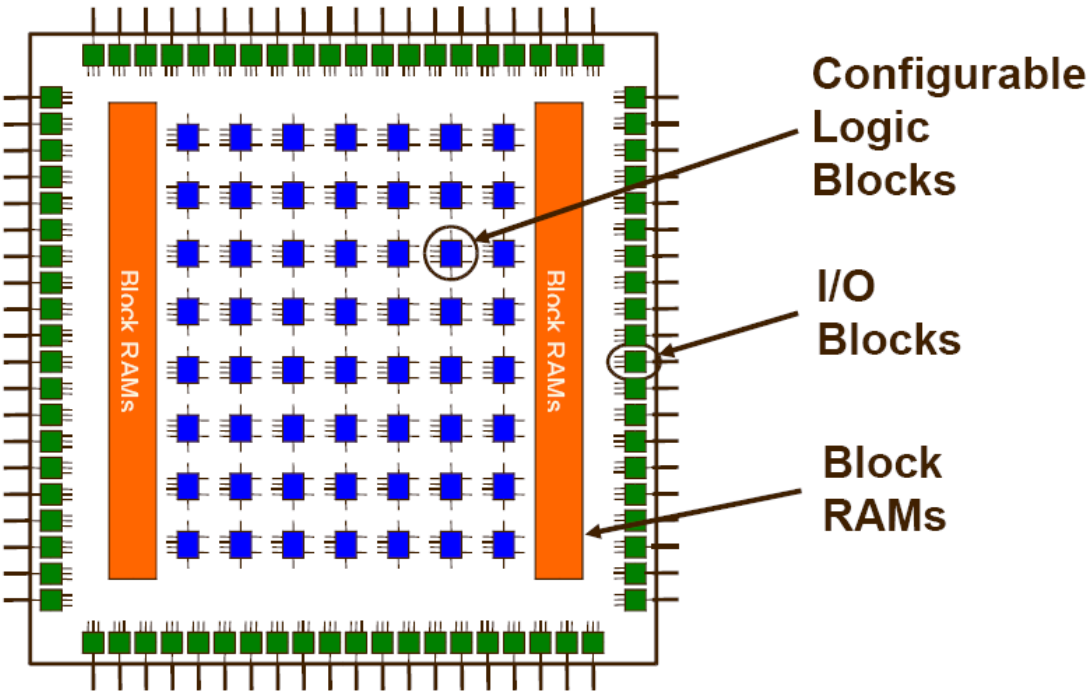


Figura 42: Estructura interna de una FPGA [3]

Internamente una FPGA es una serie de pequeños dispositivos lógicos o CLBs dispuestos sobre el silicio. Estos dispositivos lógicos se organizan como una array de celdas establecido en filas y columnas.

Los CLBs contienen en su interior elementos hardware programable que permiten que su funcionalidad sea elevada. También es habitual que contengan dispositivos de memoria.

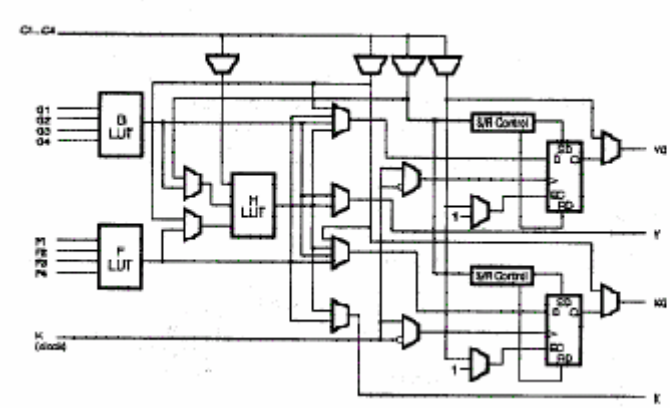


Figura 43: Estructura de un CLB

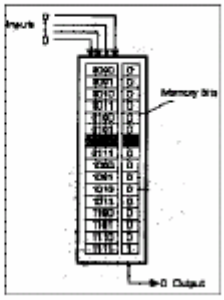


Figura 44: Estructura de una LUT

Para comunicarnos con el exterior existen elementos, también configurables colocados perimetralmente cerca de cada una de las patillas del chip, denominados IOB, estas celdas de entrada/salida tienen características programables para poder definir la forma de trabajo (entrada, salida, entrada/salida…) y están colocadas rodeando la matriz de CLBs.

La forma de conectar CLBs entre sí o con otras partes de la FPGA (entre las que están los IOBs) es a través de una estructura de interconexión. Ésta está formada por un gran número de bloques de interconexión que también pueden programarse y que pueden poseer distintas velocidades.

Esta estructura de interconexión está formada por un canal de rutado vertical y otro horizontal, a los que se pueden conectar cualquier CLB próxima mediante un punto de interconexión. Ambos canales de rutado se unen a través de una matriz de conmutación PSM (programmable switch matrix).

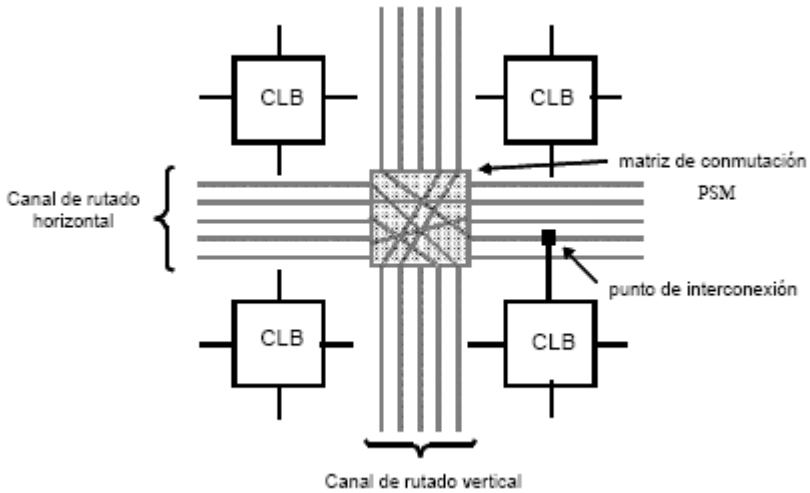


Figura 45: Estructura de interconexión de FPGA

Además de estos elementos también hay elementos de control o configuración, que sirven para controlar los multiplexores, implementar LUTs (Look Up Table) y configurar el interconexionado; y en con frecuencia, también se dispone de memorias de configuración regularmente distribuidas y conectadas en scan-path.

Aparte de esta estructura, que es la básica, cada fabricante añade sus propias ideas, por ejemplo hay algunos que tienen varios planos con filas y columnas de CLBs; el diseño físico y la fabricación es independiente del diseño particular de cada fabricante.

La funcionalidad y conexión es programable, así que los diseños no se fabrican sino que se programan. El diseñador cuenta con la ayuda de herramientas de programación; cada fabricante suele tener las suyas, aunque usan lenguajes de programación comunes, como pueden ser VHDL, Verilog, ABEL, etc.

B. Manual de usuario

Para poder ejecutar este Gestor HW en un equipo, éste debe tener instalado la máquina virtual de Java J2SDK 1.4.2_05; y no precisa de más instalaciones.

El archivo a ejecutar se llama "GestorHW.bat" y únicamente hay que hacer doble clic sobre este archivo; tras lo cual nos encontramos con la pantalla que se muestra en la figura 46.

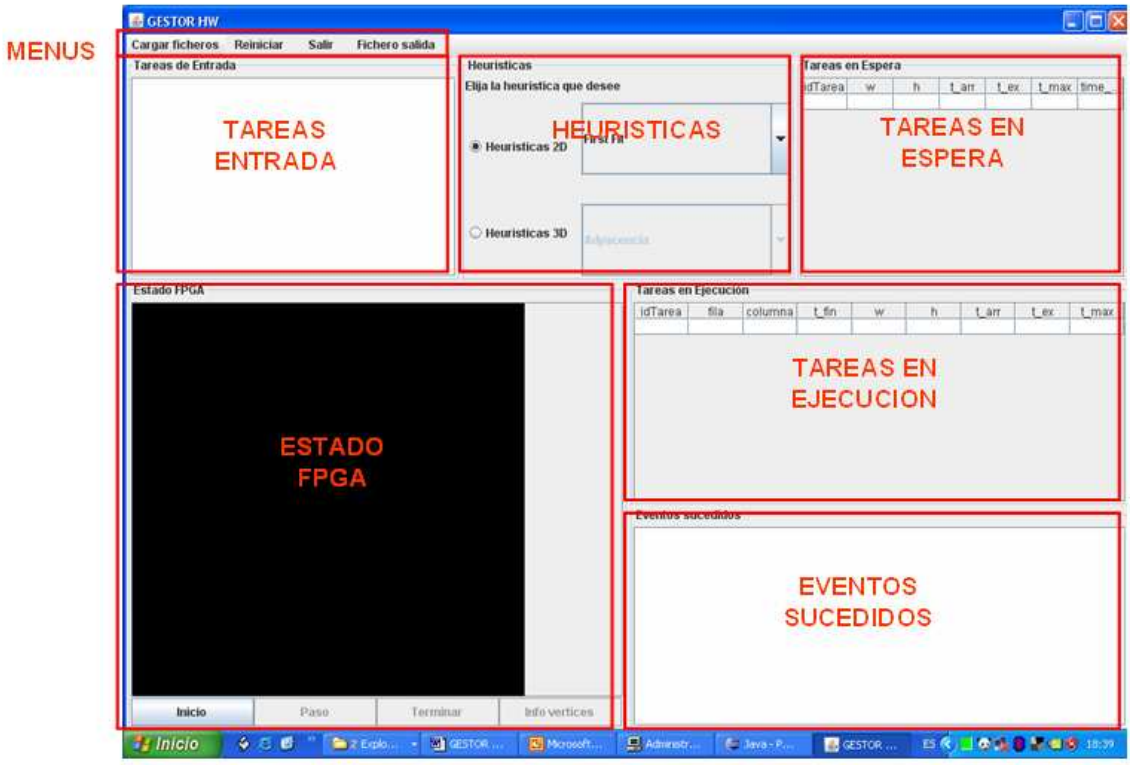


Fig. 46. Apariencia del Gestor HW.

Los elementos que se pueden observar son los siguientes:

a) Zona de menús. Situada en la parte superior a la izquierda.
b) Tareas de Entrada. Se encuentra en la esquina superior izquierda.
c) Heurísticas. Está en la zona superior central.
d) Tareas en Espera. Se muestra en la esquina superior derecha.
e) Estado de la FPGA. Situada en la esquina inferior izquierda.
f) Tareas en ejecución. Se encuentra en la zona central a la derecha.
g) Panel de eventos sucedidos. Se muestra en la esquina inferior derecha.

a) ZONA DE MENUS

En la parte superior de la ventana del Gestor, se dispone de cuatro botones que son: Cargar Ficheros, Reiniciar, Salir y Fichero Salida; cada uno de ellos con una determinada funcionalidad.

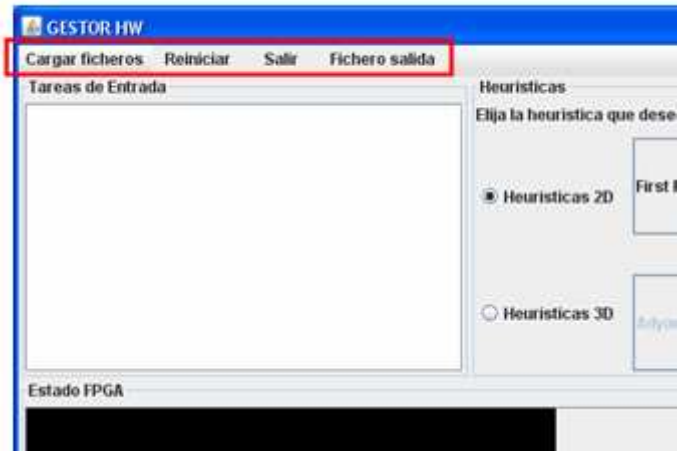


Fig.47

CARGAR FICHEROS

Al pulsar sobre "Cargar Ficheros", dispondremos de dos opciones: Ficheros por defecto y Elegir ficheros, como se puede observar en la figura 48.

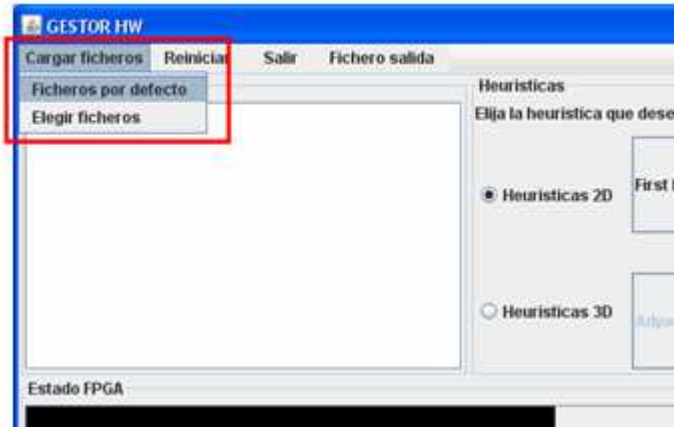


Fig. 48.

Con la primera opción, "Ficheros por defecto", la ejecución del Gestor se realiza tomando como datos de entrada y de configuración, los que se encuentran en los archivos FicheroEntradaTareas.txt y FicheroEntradaConfiguracion.txt, respectivamente. Los datos de salida obtenidos, se escriben en el archivo FicheroSalida.txt.

Estos archivos se encuentran en la ruta: "C:\Gestor\Pruebas\FicherosDefecto" y pueden ser cambiados sin ningún problema, para modificar la información de entrada por defecto.

Si se tomara como elección "Elegir ficheros", se puede decidir de qué ficheros se toman los datos de entrada, de configuración y/o en qué archivo se volcará la información de salida. Mediante la ventana que se muestra en la figura 49.

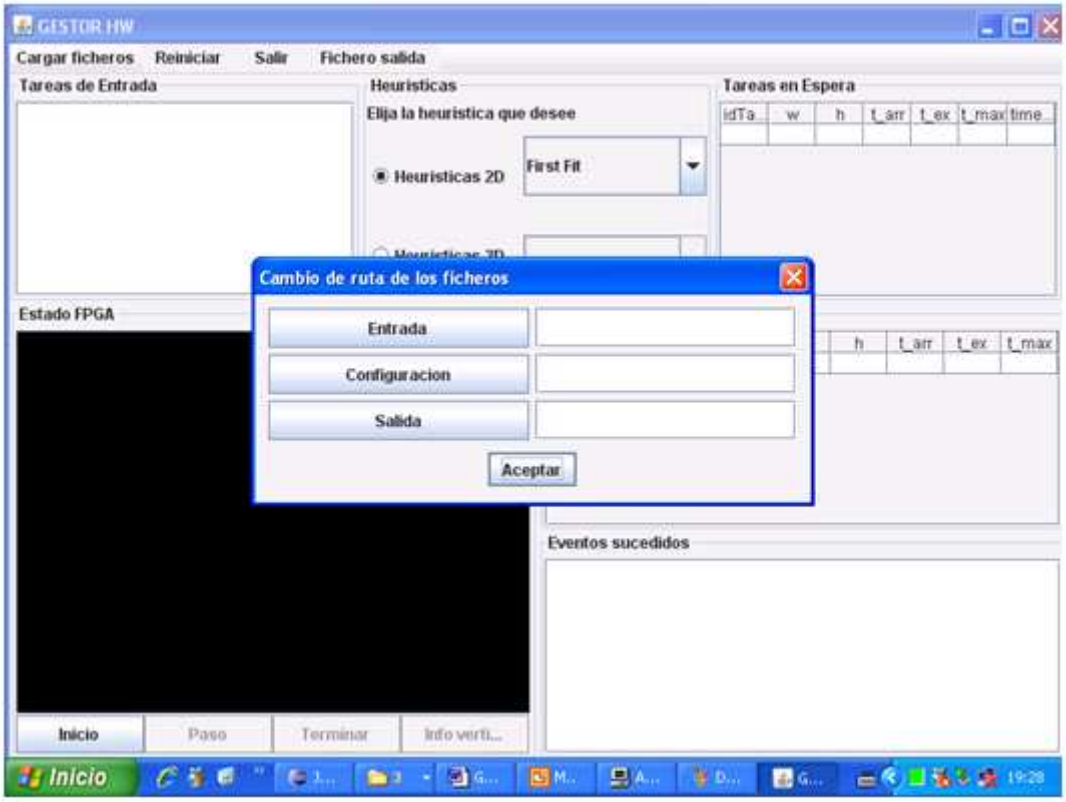


Fig. 49.

Hay varias opciones para elegir los ficheros; se puede cambiar la fuente de cualquiera de los ficheros, de todos a la vez o de ninguno (dejando los cuadros de texto en blanco y pulsando el botón aceptar). Una vez que hayamos decidido de qué fichero queremos cambiar la fuente, o bien se puede escribir la ruta directamente dentro del cuadro de texto, o bien se ofrece la posibilidad de recorrer el árbol de directorios del sistema para seleccionarlo, partiendo de “Mis documentos”, tal y como se puede ver en la figura 50.

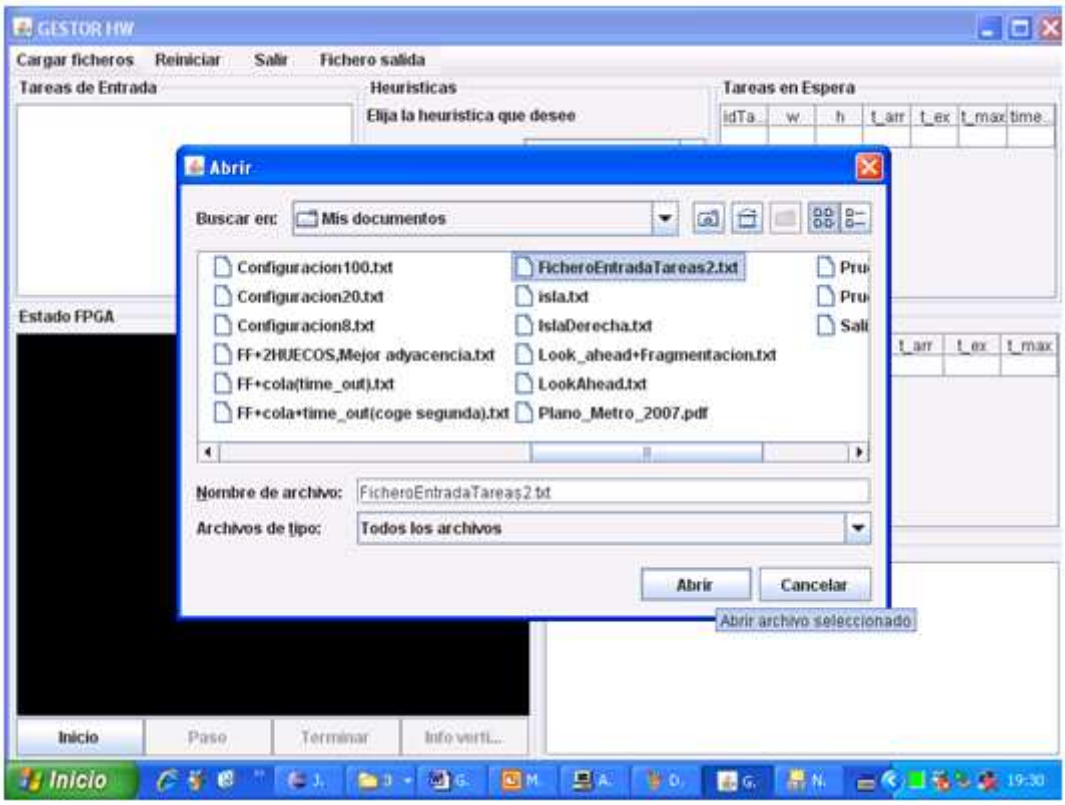


Fig. 50.

Cabe destacar que una vez que se haya tomado la decisión de qué ficheros va a utilizar el Gestor, durante la ejecución del programa no se pueden cambiar los mismos, hasta que se inicie otra nueva ejecución del Gestor. También hay que decir que en el proceso de elegir un fichero del árbol de directorios, la ventana que nos ofrece esa posibilidad, la que se muestra en la figura 50, puede tardar en aparecer unos segundos.

REINICIAR

El botón de "Reiniciar", que se muestra en la figura 51, se utiliza para realizar varias ejecuciones del Gestor de manera encadenada, sin tener que volver a lanzar el ejecutable .bat de nuevo.

Fig. 51.

SALIR

Mediante el botón "Salir", que se muestra en la figura 52, sirve para salir del Gestor (de igual forma que también se hace con el botón de cerrar de la propia ventana).

Fig. 52.

FICHERO SALIDA

Para escribir el fichero de salida, con la información obtenida durante la ejecución del Gestor, se utiliza el botón "Fichero salida", el cuál se destaca en la figura 53.

Fig. 53.

En el fichero de salida hay información acerca de:

- Tareas de entrada. Las características de las tareas que hay en el fichero de entrada
- Heurística. La función de coste que ha escogido el usuario.
- Tiempo total de ejecución. Instante en el que se ha finalizado la última tarea que estaba ejecutándose en la FPGA.
- Información de tareas ejecutadas. Se indica el número de tareas ejecutadas y el promedio de FPGA ocupada durante toda la ejecución.
- Información de tareas rechazadas. Se indica el número de tareas rechazadas, el volumen de tareas rechazadas e información acerca de las tareas rechazadas, tales como los parámetros que caracterizan la tarea rechazada.

b) TAREAS DE ENTRADA

El panel donde se muestran las tareas de entrada que va a tomar el Gestor, se encuentra situado en la esquina superior izquierda.

Las tareas que el Gestor toma como entrada proceden del "C:\Gestor\Pruebas\ FicherosDefecto\FicheroEntradaTareas.txt", en el caso de que se haya tomado la decisión "Ficheros por defecto", o del fichero que se haya elegido anteriormente.

Dentro del panel se pueden ver las características de las tareas de entrada que son:

- ID Tarea: identificador de la tarea
- h: altura de la tarea
- w: anchura de la tarea
- t_arr: tiempo de llegada
- t_ex: tiempo de ejecución
- t_max: tiempo máximo.

Fig. 54.

Al inicio de la ejecución del Gestor, cuando no se ha dado aún ningún paso, en el panel de Tareas de Entrada se muestran todas las tareas, tal y como se leen del fichero de entrada.

El número de identificador de la tarea se asigna en función de la colocación que tenga la tarea dentro del fichero de entrada. En la figura 55., se muestran las tareas que hay en el fichero de entrada y que están reflejadas en el panel de Tareas de Entrada de la figura 54.

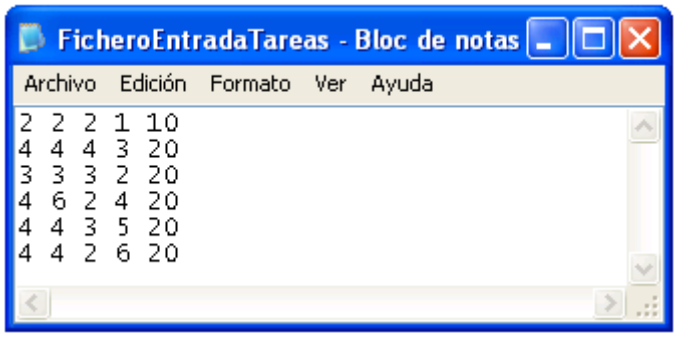


Fig. 55

Las tareas se muestran dentro del panel ordenadas por su tiempo t_arr, y según van transcurriendo los instantes de ejecución y se cumple su t_arr, van desapareciendo del panel de Tareas de Entada.

c) HEURISTICAS

La función de coste para la selección de vértices, que utilizaremos para la ejecución del Gestor, se puede elegir desde el panel de heurísticas, el cual se encuentra en la zona superior central.

Las heurísticas que se pueden seleccionar son de dos tipos: 2D y 3D; siendo las 3D una extensión de las 2D, incluyendo como tercera coordenada el tiempo.

Para seleccionar cualquiera de ellas, primero hay que pulsar sobre el check box que se corresponda con el tipo de función de coste, y a continuación, seleccionar la heurística que se quiera mediante el combo box.

Las funciones de coste 2D son First Fit (FF), basada en adyacencia 2D (Adyacencia 2D) y basada en fragmentación (Fragmentación), como se puede ver en la figura 56.

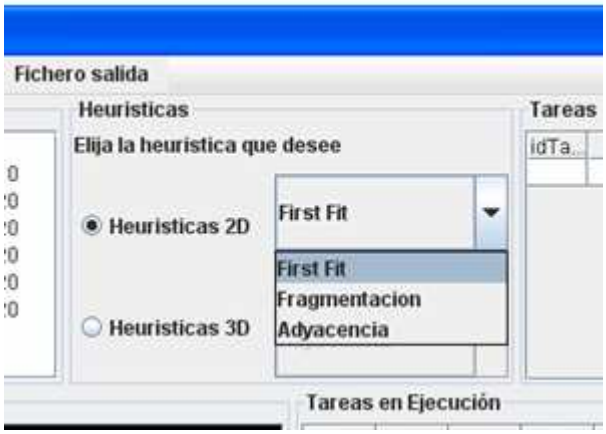


Fig.56.

Las funciones de coste 3D son la basada en adyacencia 3D (Adyacencia 3D) y basada en look ahead (Look-ahead), como se puede ver en la figura 57.

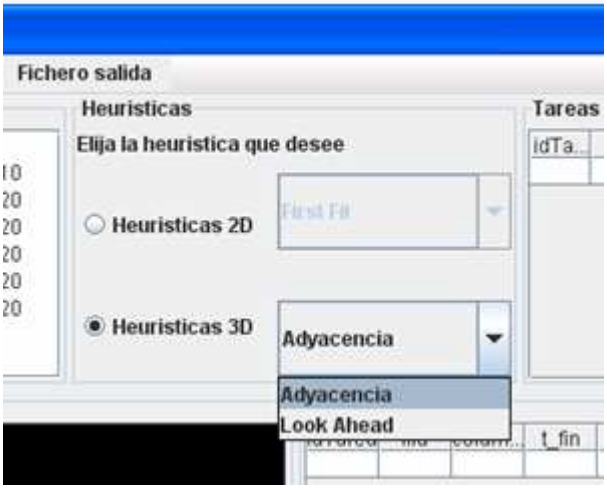


Fig. 57.

Hay que subrayar, que la heurística con la que se quiere ejecutar el Gestor hay que seleccionarla antes de comenzar la propia ejecución del mismo (antes de pulsar el botón "Inicio"). Y que además, durante la ejecución los cambios que se hagan de heurística no serán efectivos hasta que se inicie otra nueva ejecución, ya sea por el nuevo lanzamiento de la aplicación o por la utilización del botón "Reiniciar".

La heurística que se toma por defecto es FF.

d) TAREAS EN ESPERA

Las tareas que entran en la Lista de Tareas en Espera, ya sea porque no haya hueco o por decisión de la estrategia Look-ahead, se pueden ver en el panel de Tareas en Espera, situado en la esquina superior derecha.

Las tareas en espera se muestran en forma de tabla, ordenadas en función de su time_out. Cada una de las columnas de la tabla, donde se muestran las tareas en espera,

se corresponde con los parámetros que caracterizan a cada una de las tareas, así que las columnas que se pueden ver son:

- idTarea: identificador de la tarea que se la ha asignado
- w: anchura
- h: altura
- t_arr: tiempo de llegada
- t_ex: tiempo de ejecución
- t_max: tiempo máximo
- time_out: time out calculado.

En la figura 58, se pueden ver tres tareas dentro de la Lista de Tareas en Espera, ordenadas por time_out.

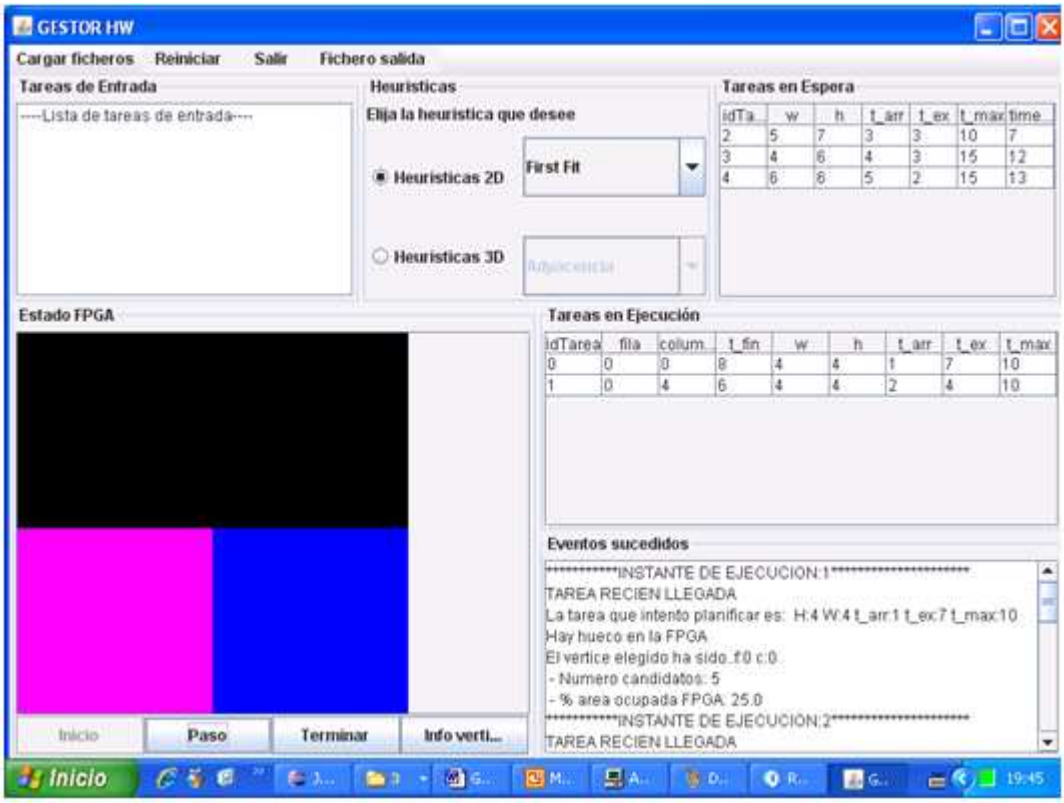


Fig. 58.

e) ESTADO DE LA FPGA

El estado de ocupación de la FPGA en cada instante de la ejecución lo podemos ver a través del panel de Estado de la FPGA, que se encuentra en la esquina inferior izquierda.

Dentro de este panel, podemos hacernos una idea de las tareas que se están ejecutando en ese preciso instante de tiempo, así como del espacio libre que hay en ese momento en la FPGA.

Las tareas que están en ejecución en ese instante se verán en colores (que se han asignado automáticamente en función del identificador que se le dió a la tarea); y los bloques sin ocupar de la FPGA, se verán en negro, como se puede observar en la figura 59.

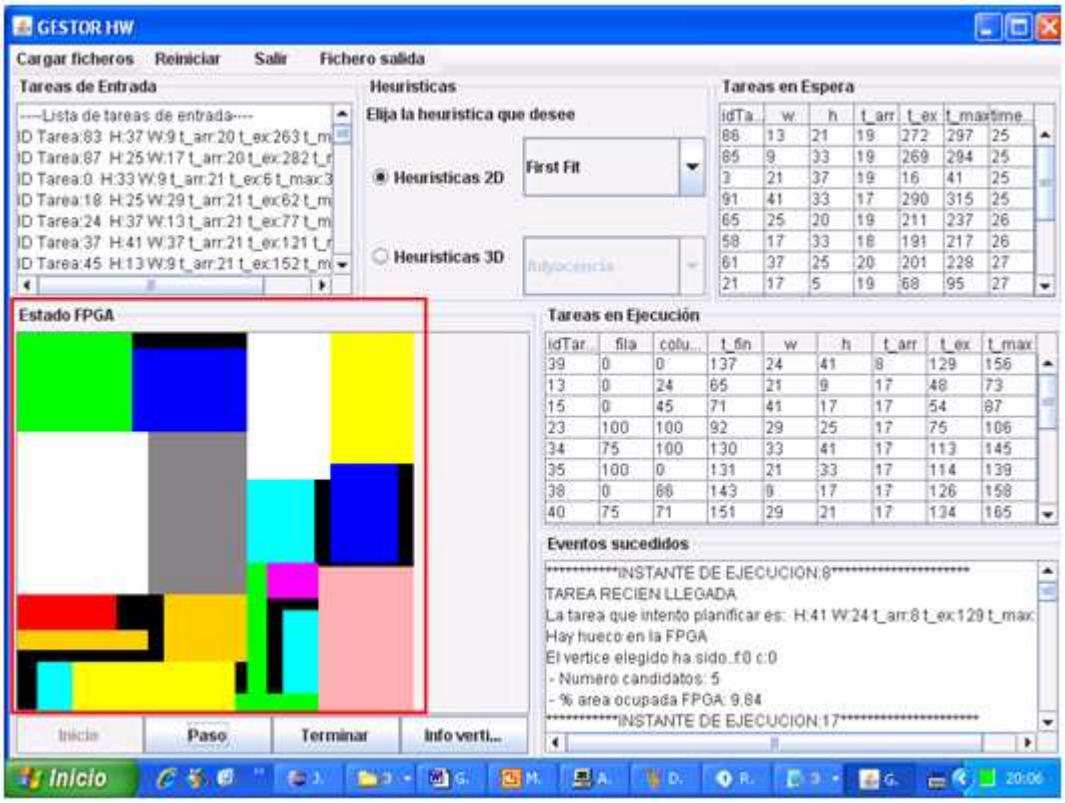


Fig. 59.

En la parte inferior del panel de Estado de la FPGA se dispone de cuatro botones, que son Inicio, Paso, Terminar e Info Vértices.

INICIO

Al pulsar "Inicio", es cuando realmente se comienza la ejecución de las tareas. También, se realizan otras operaciones tales como, ajustar el panel de estado de la FPGA a los parámetros de configuración que se hayan indicado, o como tomar la heurística que el usuario haya escogido en el panel de heurísticas.

Antes de pulsar "Inicio", los demás botones de este panel se encuentran deshabilitados; de esta forma, es imprescindible pulsar Inicio para comenzar la ejecución, y así se ajustarán determinados parámetros que son necesarios mediante él. Una vez que se haya pulsado "Inicio", se habilitarán el resto de los botones de este panel, para continuar con la ejecución.

PASO

Mediante este botón podemos ver la ejecución del Gestor en cada instante significativo, es decir cuando sucede algo, ya sea que una tarea acaba de llegar, ha terminado su ejecución…

En la figura 60. se puede ver un ejemplo de ejecución del Gestor, con un determinado estado de ocupación de la FPGA. En el instante que se representa en la figura, hay una tarea en ejecución, y una tarea que aún tiene que llegar (que se puede ver en el panel de Tareas de Entrada).

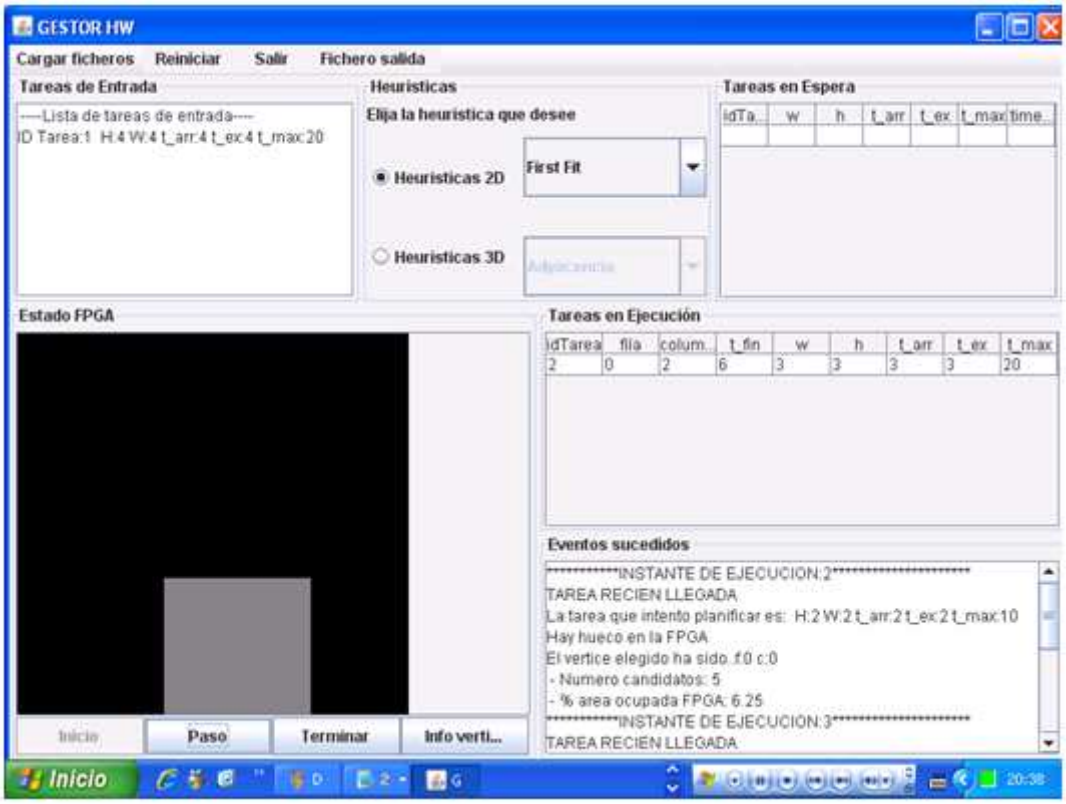


Fig. 60.

Tras pulsar el botón de "Paso", se ha realizado un paso más en la ejecución del Gestor, y el estado de la FPGA se puede ver en la siguiente figura, donde hay una nueva tarea en ejecución, representada de color rosa.

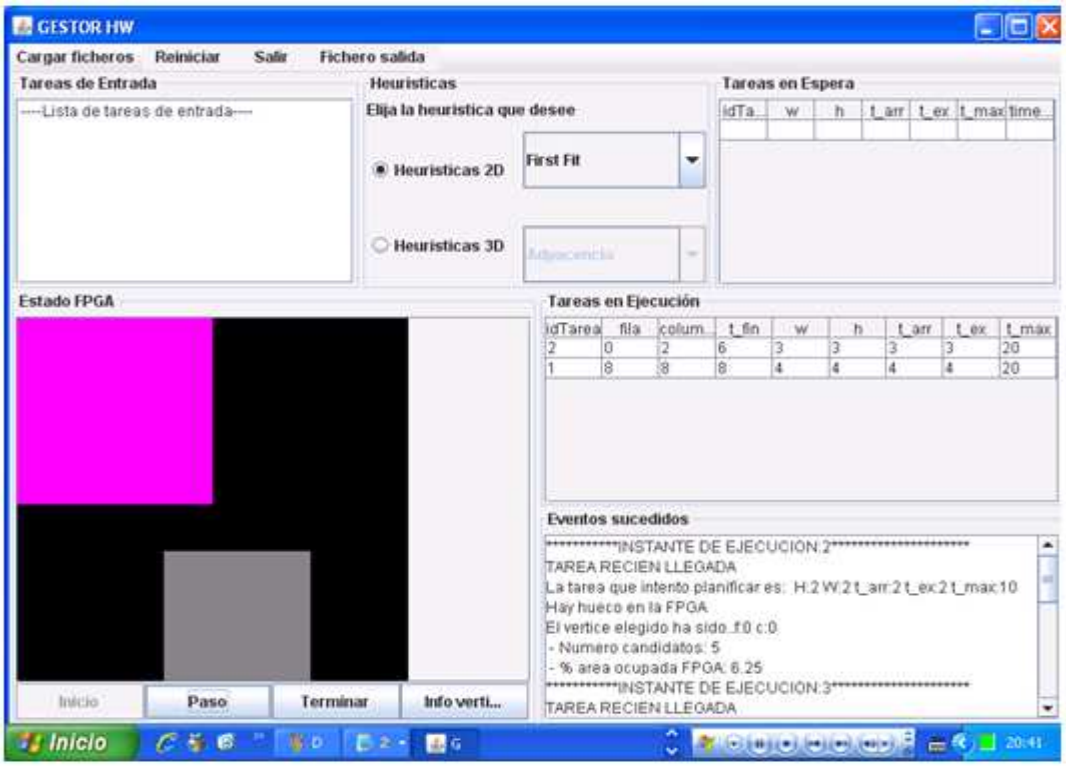


Fig. 61.

Tras pulsar “Paso” se deshabilitará el botón de “Inicio”, puesto que ya no tiene sentido el pulsarlo hasta una nueva ejecución. El botón “Inicio” continuará deshabilitado hasta que se ejecute de nuevo la aplicación o se decida reiniciar.

TERMINAR

El tercer botón, de los que están en el panel de Estado de la FPGA, es “Terminar”, que se usa para realizar la ejecución completa del Gestor, sin tener la posibilidad de observar el estado de la FPGA o de las estructuras auxiliares en cada instante de la ejecución.

Siempre que queremos “Reiniciar” (aunque la ejecución haya sido paso a paso), es recomendable pulsar el botón “Terminar” para que el reinicio sea correcto; si no se hace de este modo, puede que el funcionamiento del Gestor no sea el adecuado.

Hay que destacar que una vez se haya pulsado “Terminar”, el botón “Paso” se deshabilitará puesto que la ejecución ya se ha finalizado.

INFO VERTICES

El botón situado más a la derecha de los que hay en el panel del Estado de la FPGA, es el botón “Info vértices.

Mediante este botón se puede ver la información acerca de la VLS. En la figura 62, se muestra un posible estado de FPGA, durante una ejecución del Gestor.

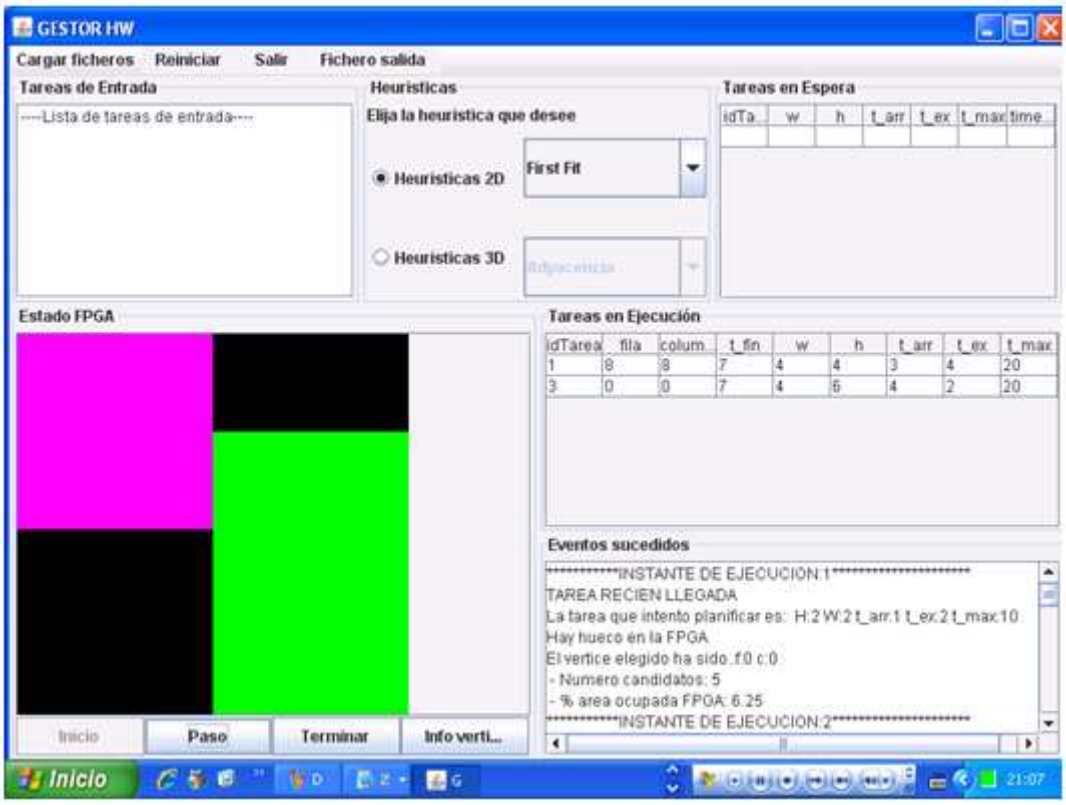


Fig. 62.

Si pulsamos el botón “Info vértices”, podremos ver la información acerca del Conjunto de Lista de Vértices (VLS) que representa los huecos de la FPGA, en ese instante. En esta figura, se puede ver como la VLS esta formada por dos listas de vértices (VL) y cada uno de los vértices que forma cada lista de vértices, que en este caso son cuatro para cada VL.

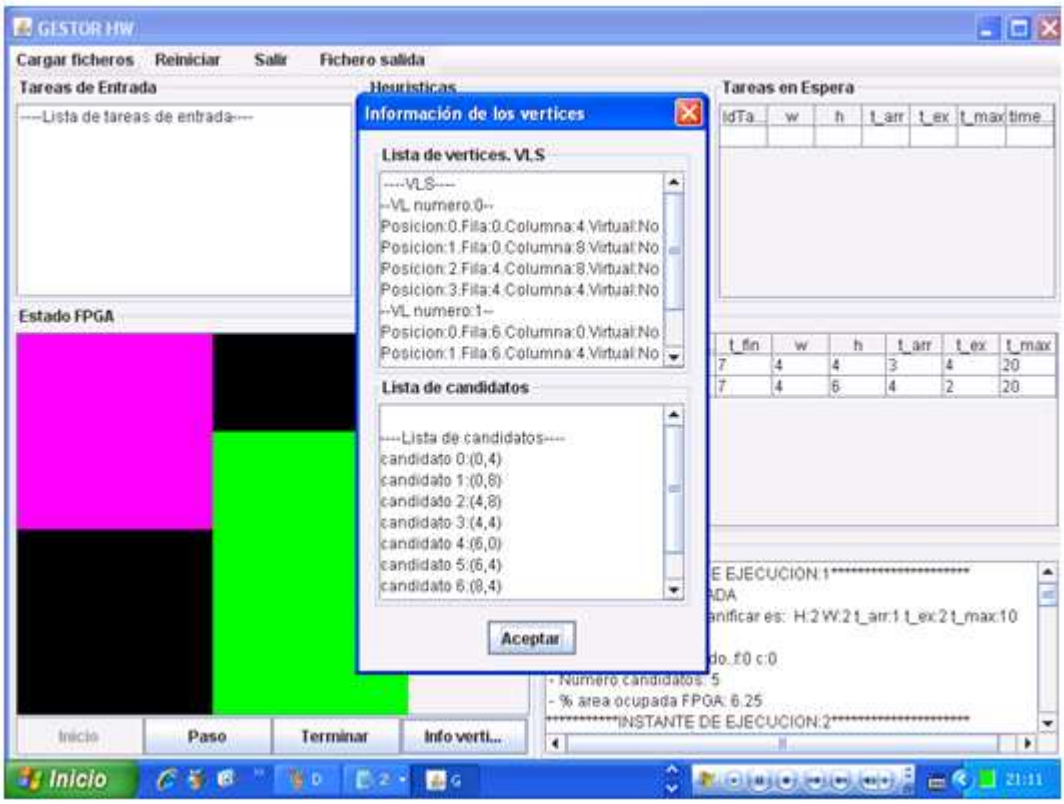


Fig. 63.

En la figura anterior, se puede ver como la ventana que surge al pulsar “Info vértices”, esta dividida en dos. En la mitad superior, se muestra la información relacionada con el Conjunto de Listas de Vértices (VLS). En la mitad inferior, se puede observar la lista de vértices candidatos actual a considerar, que se obtiene del Conjunto de Listas de Vértices (VLS). Es esta lista de vértices candidatos, la que se va a tener en cuenta para la selección de vértices, para la ubicación de nuevas tareas dentro de la FPGA.

Para salir de esta ventana de información, o volver a la ventana principal del Gestor, hay que pulsar el botón “Aceptar”.

f) TAREAS EN EJECUCION

Las tareas que en ejecución, se pueden ver en el panel de Tareas en Ejecución, situado en la zona central a la derecha.

Las tareas en ejecución se muestran en forma de tabla, ordenadas de acuerdo con su t_arr. Cada una de las columnas de la tabla, donde se muestran las tareas en ejecución, se corresponde con los parámetros de cada una de las tareas y de las características de la ejecución de esa tarea. Las columnas que se pueden ver son:

- idTarea: identificador de la tarea que se la ha asignado

- fila: es la fila del vértice candidato que se escogió para la ubicación de la tarea
- columna: es la columna del vértice candidato que se eligió para la ubicación de la tarea
- t_fin: instante en que finaliza la ejecución de la tarea
- w: anchura de la tarea
- h: altura de la tarea
- t_arr: instante de tiempo en que llegó la tarea
- t_ex: tiempo de ejecución de la tarea
- t_max: instante máximo en el que ha tenido que finalizar la ejecución de la tarea

En la figura 64, se pueden ver dos tareas dentro de la Lista de Tareas en Ejecución, ordenadas por t_arr.

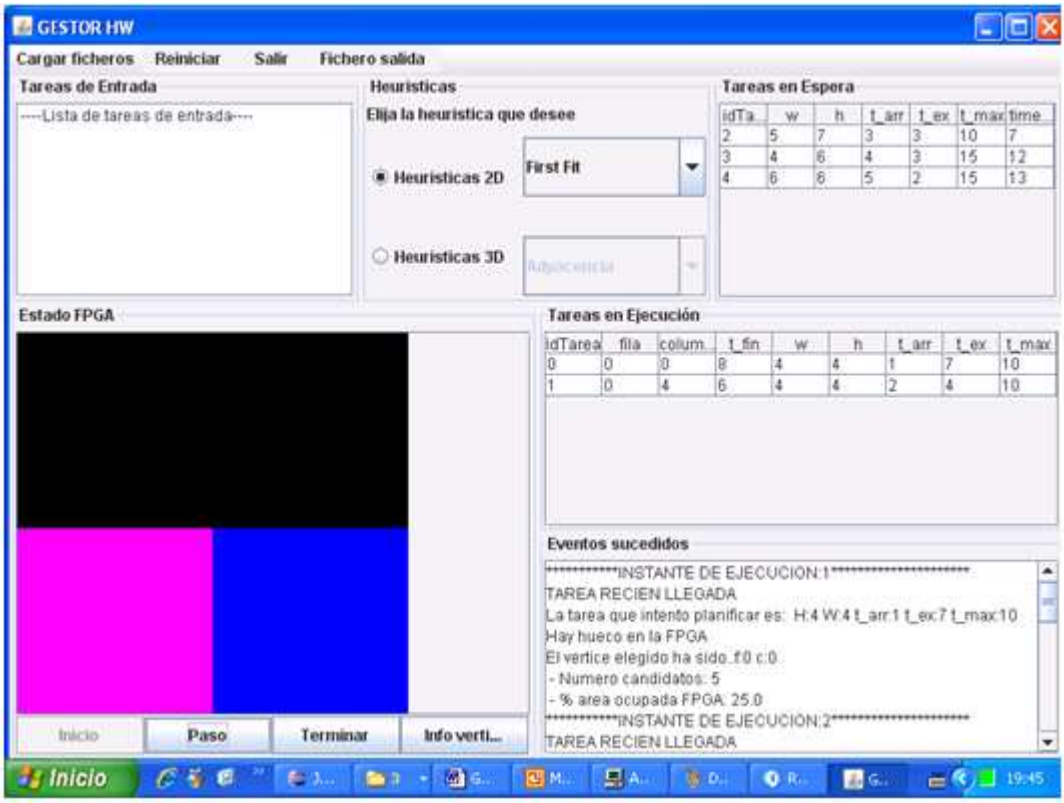


Fig. 64.

g) PANEL DE EVENTOS SUCEDIDOS

El panel donde se muestran los eventos sucedidos durante la ejecución, se encuentra situado en la esquina inferior derecha, y es el panel de Eventos sucedidos.

Dentro del panel se pueden los eventos ocurridos en cada uno de los instantes significativos de la ejecución. Los eventos que se reflejan son los siguientes:

- Número de candidatos en cada instante de la ejecución
- Tarea recién llegada. Se muestran las características de la tarea, si hay o no hueco en la FPGA para esa tarea. En el caso de que haya hueco, qué vértice se elige para la ubicación.
- Tareas finalizadas. Se indica cuál es el identificador de la tarea que ha terminado su ejecución.

- Operaciones relacionadas con la Lista de Tareas en Espera. Las operaciones son comprobar que la FPGA no tiene hueco para una tarea en la Lista de Tareas en Espera, comenzar a ejecutar una tarea de esta lista y comprobar si se ha alcanzado el time_out de una tarea de la lista de espera. En el caso, de que se pueda ejecutar una tarea de la lista de espera dentro de la FPGA, además se indica la posición que ocupaba dentro de la lista y el vértice donde se coloca. Si se ha alcanzado el time_out de una tarea, también se muestra en el panel de eventos sucedidos, los parámetros que caracterizan la tarea, su time_out y la posición que ocupaba dentro de la lista de espera.

En la figura 65, se muestra un ejemplo de Panel de Eventos Sucedidos durante una ejecución del Gestor; y en la figura 66, se muestra en detalle lo que esta escrito en el panel de eventos.

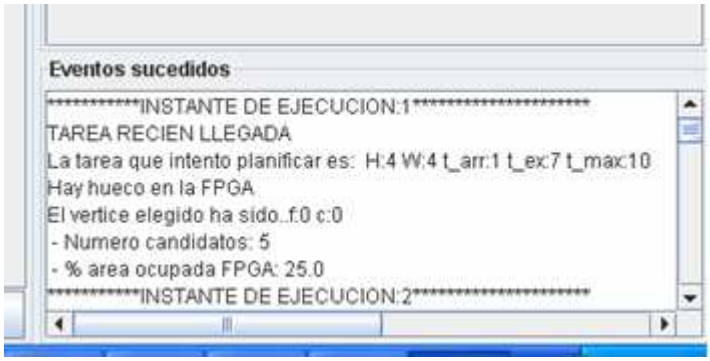


Fig. 65.

***********INSTANTE DE EJECUCION:1*********************
TAREA RECIEN LLEGADA
La tarea que intento planificar es: H:4 W:4 t_arr:1 t_ex:7 t_max:10
Hay hueco en la FPGA
El vertice elegido ha sido..f:0 c:0
- Numero candidatos: 5
- % area ocupada FPGA: 25.0
***********INSTANTE DE EJECUCION:2*********************
TAREA RECIEN LLEGADA
La tarea que intento planificar es: H:4 W:4 t_arr:2 t_ex:4 t_max:10
Hay hueco en la FPGA
El vertice elegido ha sido..f:0 c:4
- Numero candidatos: 4
- % area ocupada FPGA: 50.0
***********INSTANTE DE EJECUCION:3*********************
TAREA RECIEN LLEGADA
La tarea que intento planificar es: H:7 W:5 t_arr:3 t_ex:3 t_max:10
No hay hueco en la FPGA y meto la tarea en la lista de espera
- Numero candidatos: 4
- % area ocupada FPGA: 50.0
***********INSTANTE DE EJECUCION:4*********************
TAREA RECIEN LLEGADA
La tarea que intento planificar es: H:6 W:4 t_arr:4 t_ex:3 t_max:15
No hay hueco en la FPGA y meto la tarea en la lista de espera
- Numero candidatos: 4
- % area ocupada FPGA: 50.0
***********INSTANTE DE EJECUCION:5*********************
TAREA RECIEN LLEGADA
La tarea que intento planificar es: H:6 W:6 t_arr:5 t_ex:2 t_max:15
No hay hueco en la FPGA y meto la tarea en la lista de espera
- Numero candidatos: 4
- % area ocupada FPGA: 50.0
***********INSTANTE DE EJECUCION:6*********************

TAREA FINALIZADA
Ha terminado la tarea con id:1
- Numero candidatos: 5
- % area ocupada FPGA: 25.0
***********INSTANTE DE EJECUCION:6*********************
LISTA DE TAREAS EN ESPERA
He intentado meter una tarea de la lista de espera pero no entra
LISTA DE TAREAS EN ESPERA
Coloco la tarea que ocupa la posicion 1 en la lista de tareas en espera
El vertice elegido ha sido...f:0 c:4
- Numero candidatos: 5
- % area ocupada FPGA: 62.5
***********INSTANTE DE EJECUCION:8*********************
TAREA FINALIZADA
Ha terminado la tarea con id:0
- Numero candidatos: 5
- % area ocupada FPGA: 37.5
***********INSTANTE DE EJECUCION:8*********************
LISTA DE TAREAS EN ESPERA
Se ha alcanzado el time_out de una tarea de la lista de espera
La tarea a desestimar tiene las siguientes carcateristicas: Posicion.0 ID Tarea:2 H:7 W:5 t_arr:3 t_ex:3 t_max:10 t_time_out:7
- Numero candidatos: 5
- % area ocupada FPGA: 37.5
***********INSTANTE DE EJECUCION:9*********************
TAREA FINALIZADA
Ha terminado la tarea con id:3
- Numero candidatos: 4
- % area ocupada FPGA: 0.0
***********INSTANTE DE EJECUCION:9*********************
LISTA DE TAREAS EN ESPERA
Coloco la tarea que ocupa la posicion 0 en la lista de tareas en espera
El vertice elegido ha sido...f:0 c:0
- Numero candidatos: 5
- % area ocupada FPGA: 56.25
***********INSTANTE DE EJECUCION:11*********************
TAREA FINALIZADA
Ha terminado la tarea con id:4
- Numero candidatos: 4
- % area ocupada FPGA: 0.0

Fig. 66.

C. Ejemplo de ejecución

En este apartado, se muestra una ejecución paso a paso mediante capturas de pantalla.

Tras hacer doble clic en Gestor.bat nos encontraremos con la pantalla de la siguiente figura.

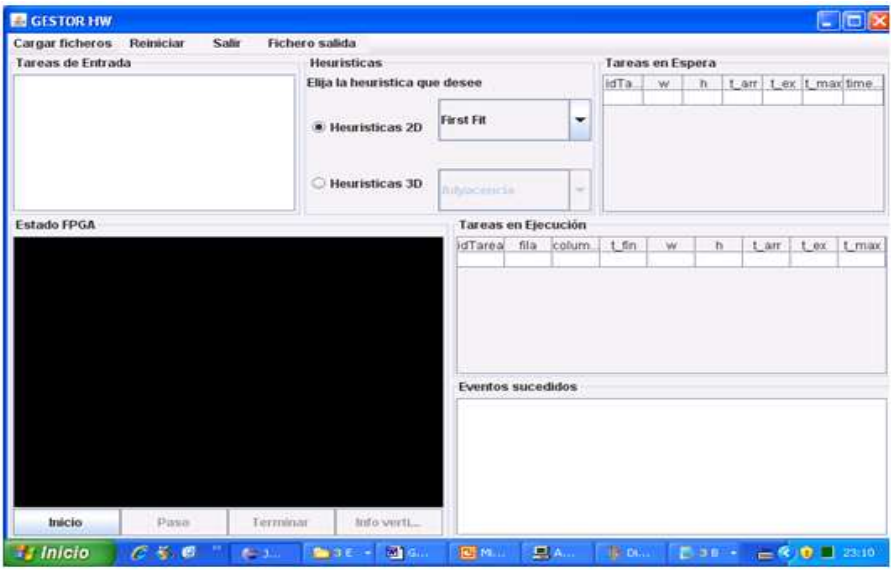


Fig. 67

Para ejecutar el gestor con los ficheros por defecto, hay que primero pulsar sobre "Cargar ficheros", como se muestra en la figura 68., y escoger la opción de "Ficheros por defecto", tal y como se muestra en la figura 69.

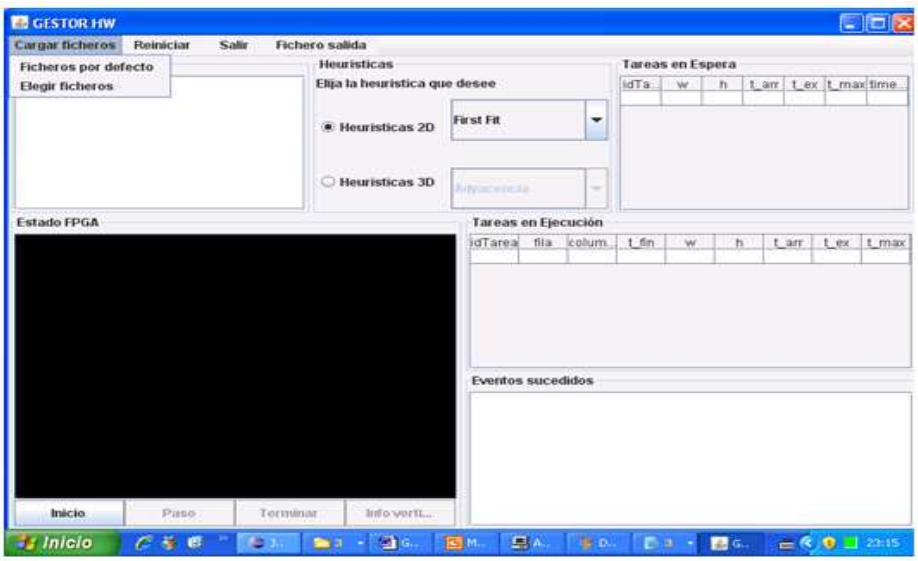


Fig. 68.

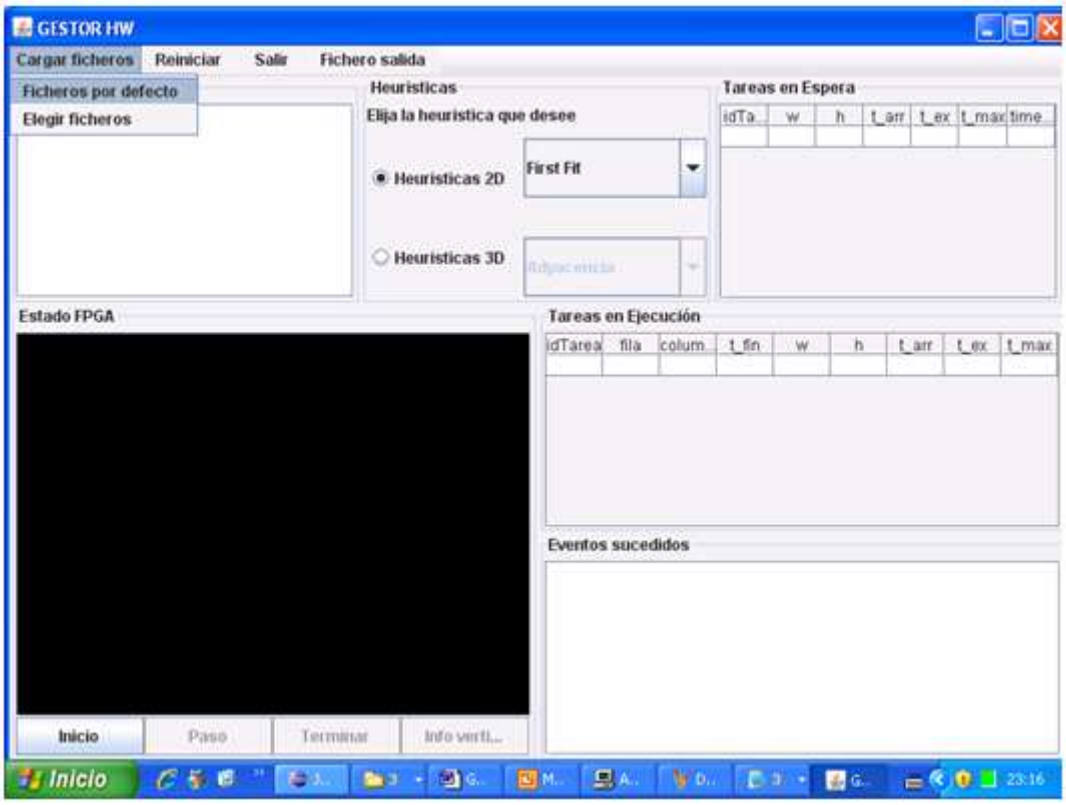


Fig. 69.

Tras haber cargado los ficheros de entrada, habría que seleccionar la heurística que se desee, pero como se pretende una ejecución con FF, entonces no es necesario seleccionar la heurística del panel de heurísticas. Los ficheros de entrada de tareas y de configuración se pueden ver en la siguiente figura.

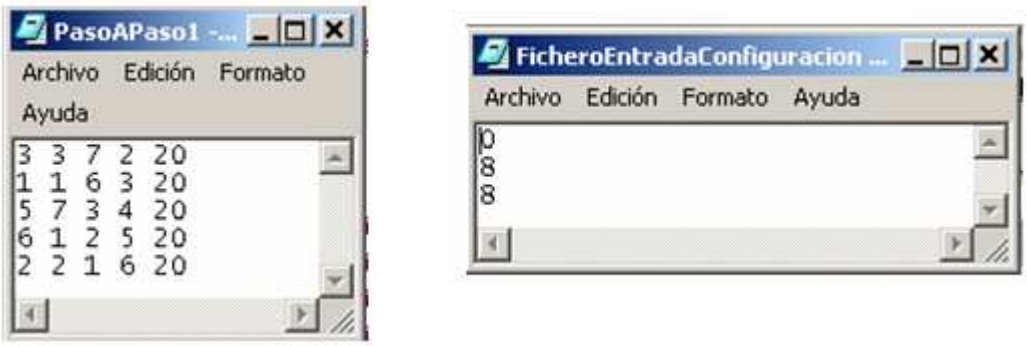


Fig. 70.

Para iniciar la ejecución, hay que pulsar sobre “Inicio”, y se habilitarán el resto de los botones del panel de estado de FPGA, como se puede ver en la figura 71.

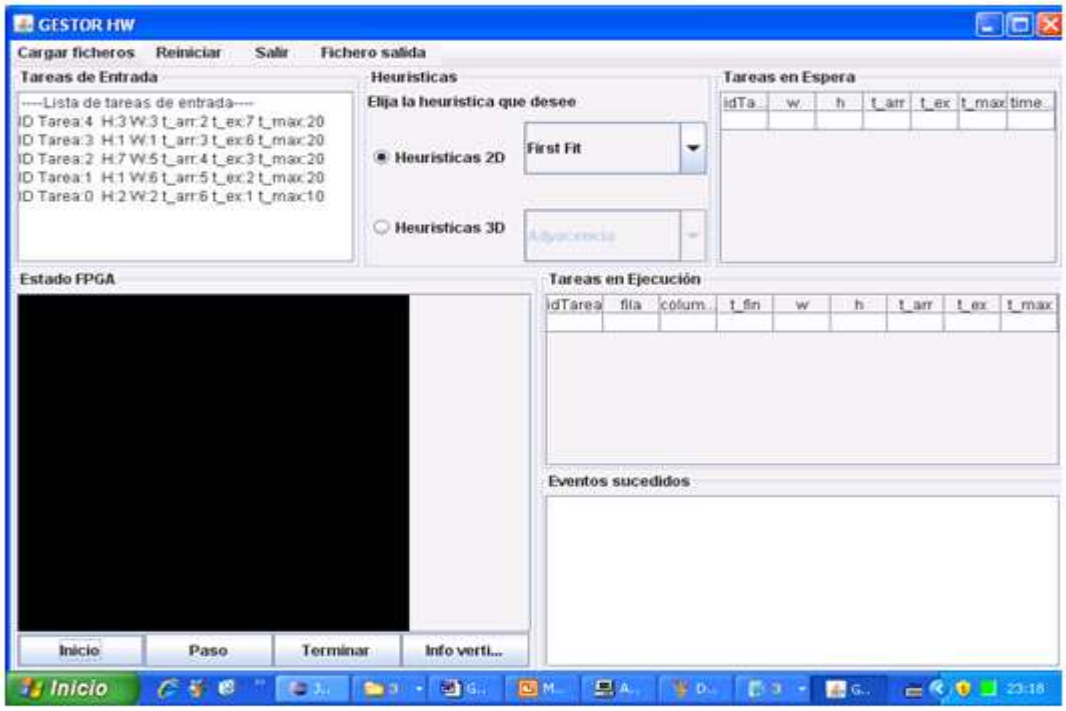


Fig. 71.

Como se desea una ejecución paso a paso, a continuación se pulsa el botón de "Paso" tantas veces como sea necesario hasta que no quede ninguna tarea por llegar, ni ejecutándose en la FPGA o en la lista de espera. La secuencia de capturas de pantalla, que se obtiene se muestra en las figuras que siguen a continuación.

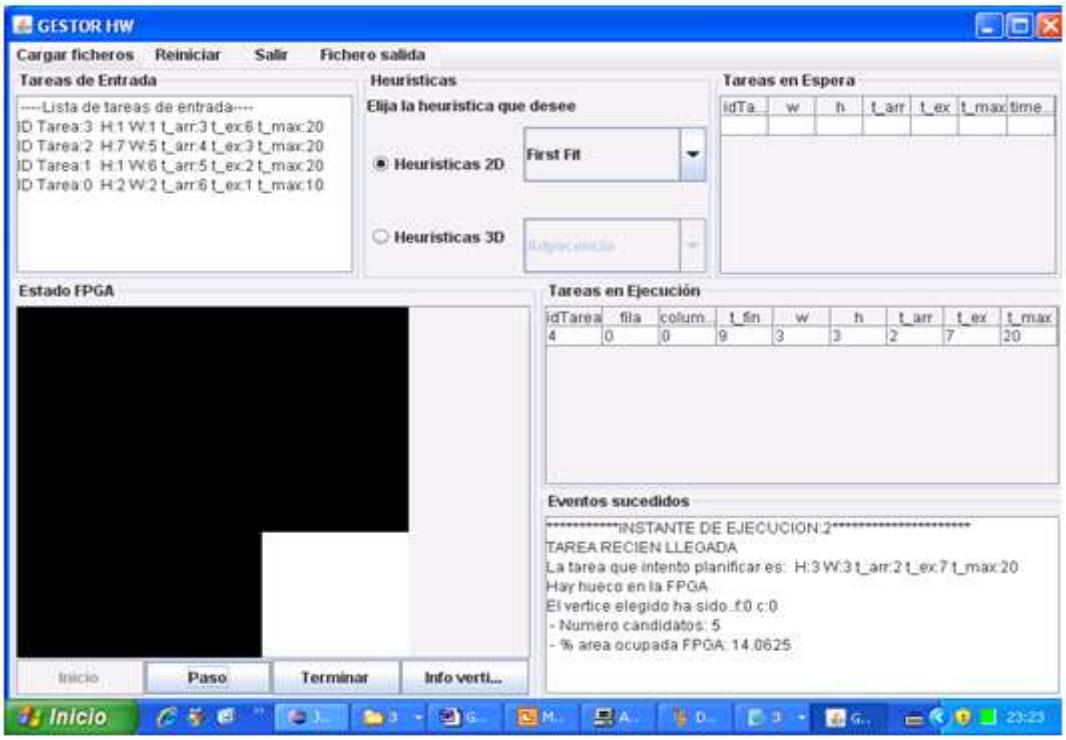


Fig. 72. Comienza a ejecutarse la primera tarea.

En la figura 73, fijarse en que la segunda tarea que ha comenzado a ejecutarse, se ha añadido a la lista de tareas en ejecución.

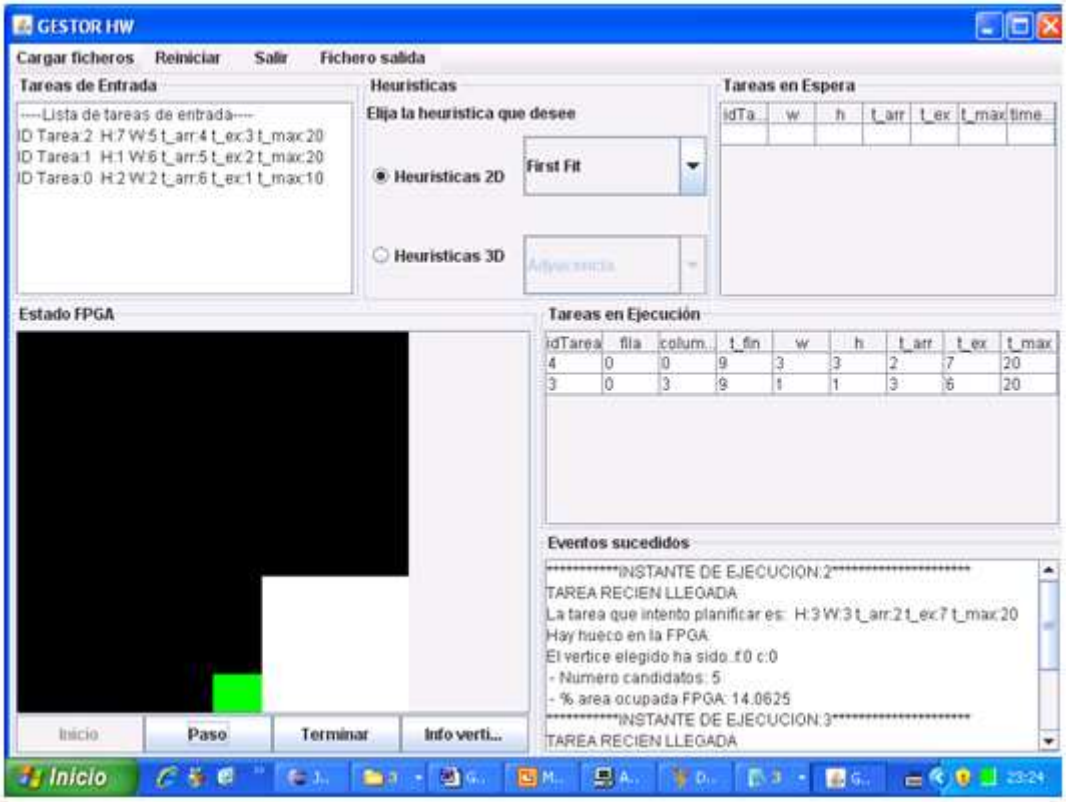


Fig. 73.

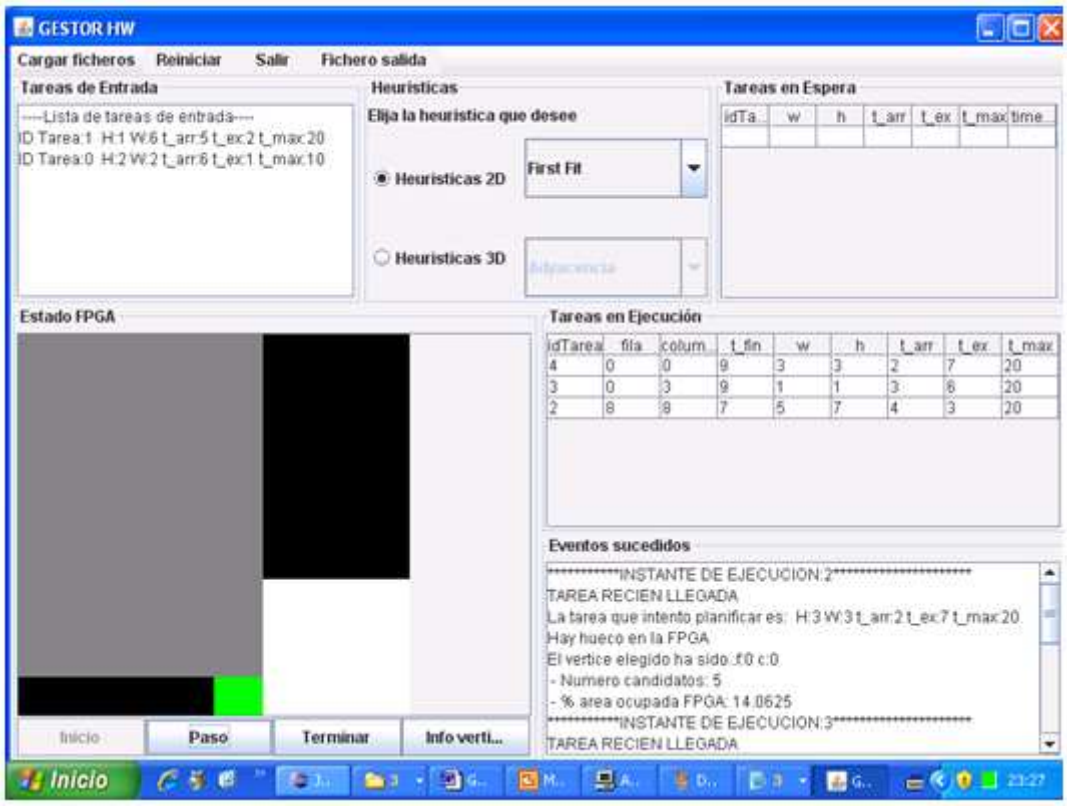


Fig.74. Comienza a ejecutarse la tercera tarea.

Si en el estado que se plantea en la figura anterior número 74, pulsamos el botón "Info vértices" podremos ver la información acerca de la VLS que representa la FPGA y

la lista de candidatos, lo cual se muestra en la siguiente figura. Como se puede observar, hay dos listas de vértices (VL), formando el conjunto de listas de vértices (VLS), lo cuál se debe a que hay dos huecos vacíos dentro de la FPGA.

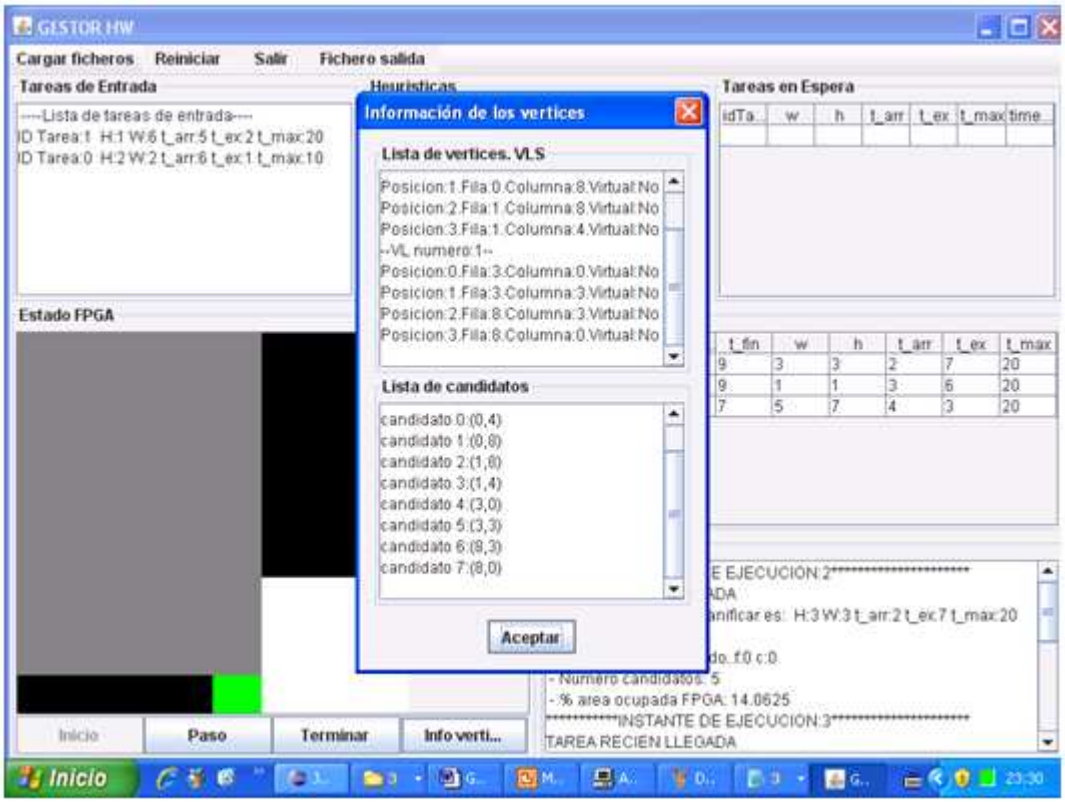


Fig. 75. VLS y lista de candidatos correspondientes al instante de ejecución representado en la figura 74

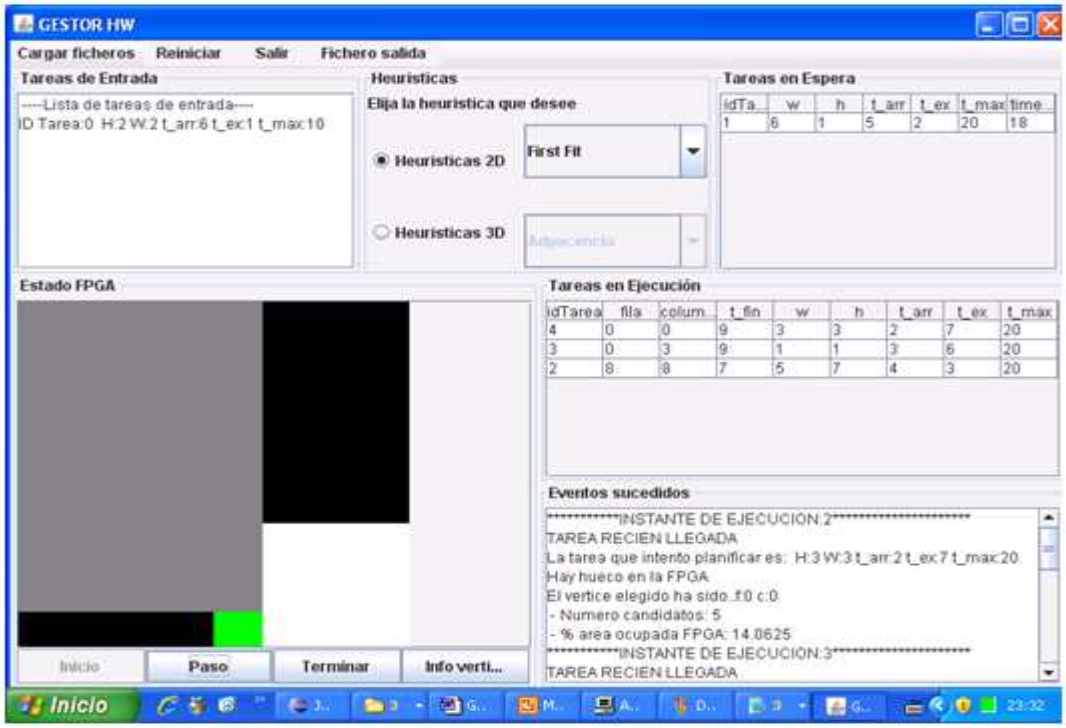


Fig. 76. La cuarta tarea se inserta en la lista de tareas en espera, puesto que no hay hueco en la FPGA.

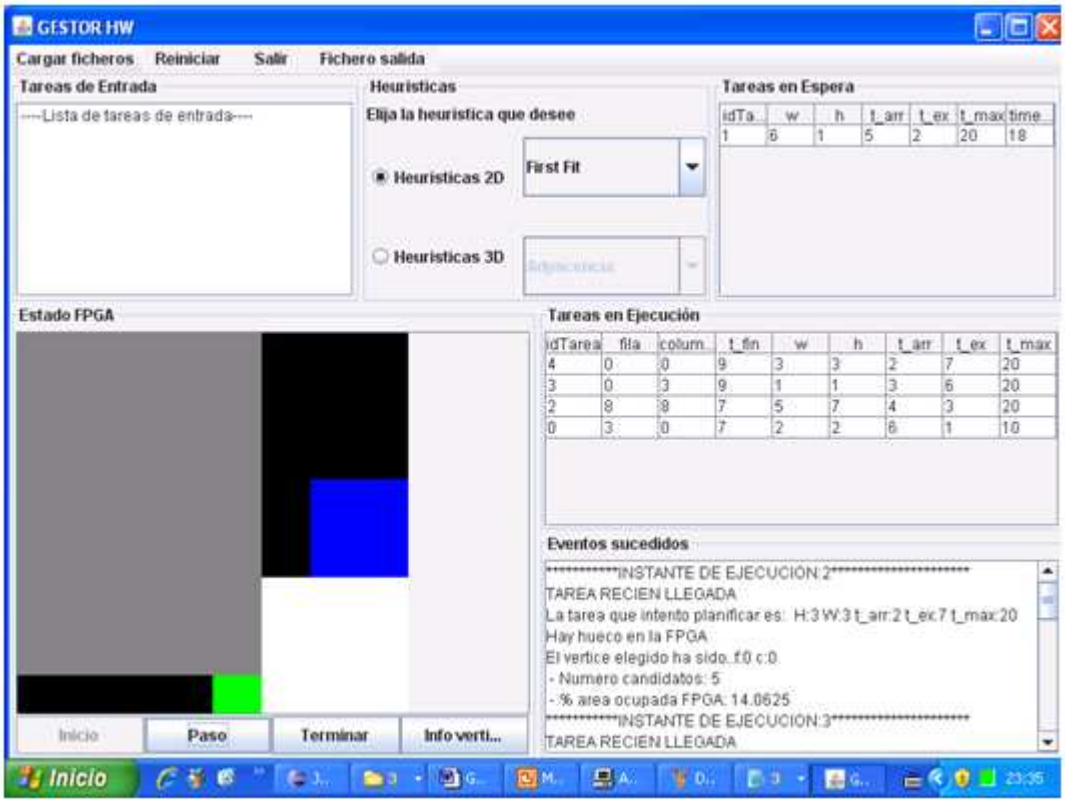


Fig. 77. La quinta tarea en llegar comienza su ejecución.

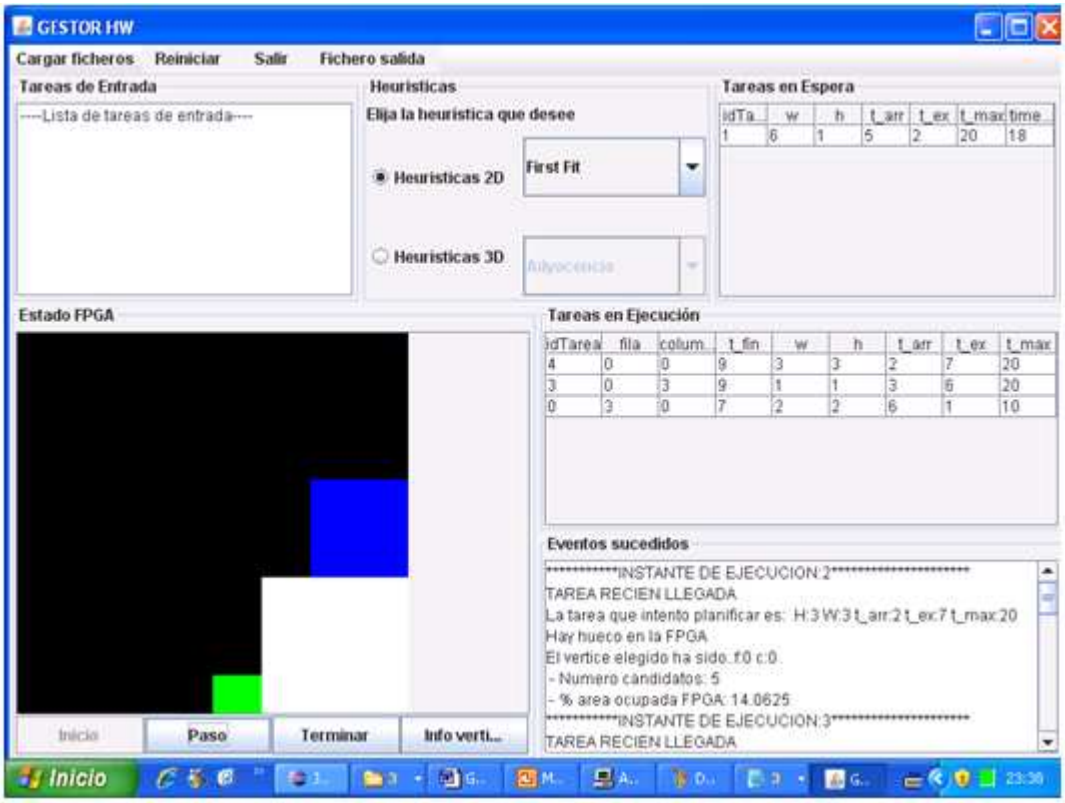


Fig. 78. Fin de la ejecución de la tarea con idTarea = 2

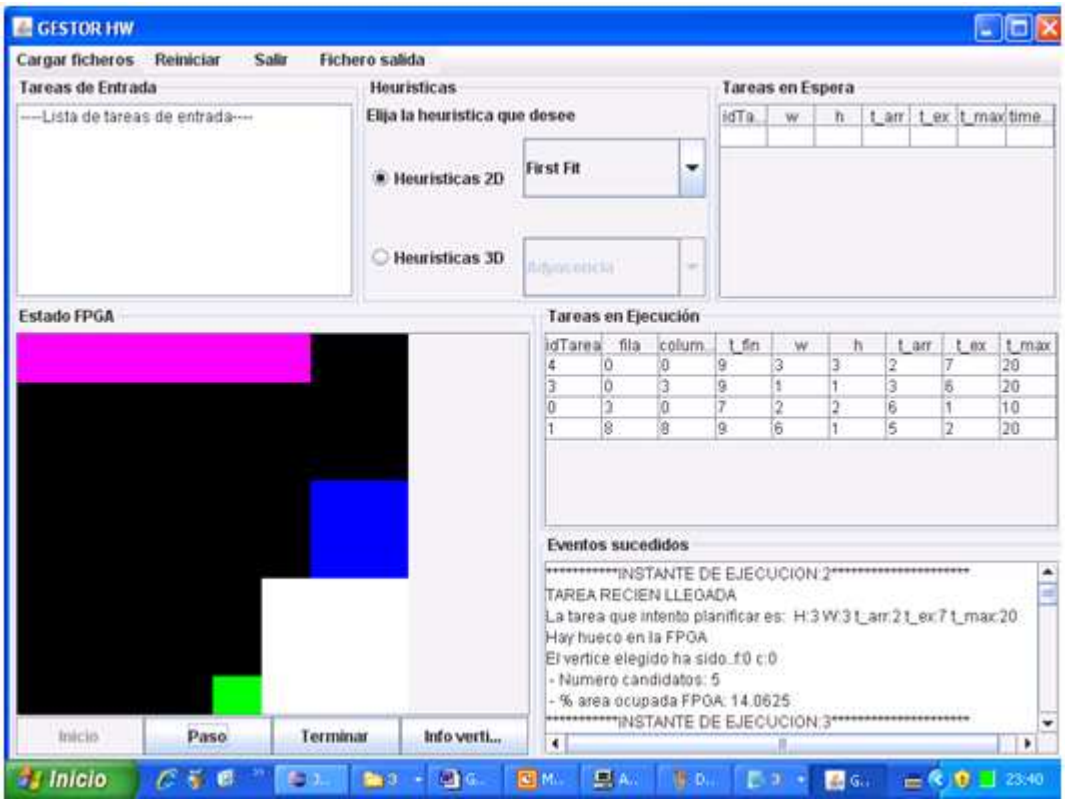


Fig. 79. La tarea con idTarea = 1, sale de la lista de tareas en espera y comienza su ejecución.

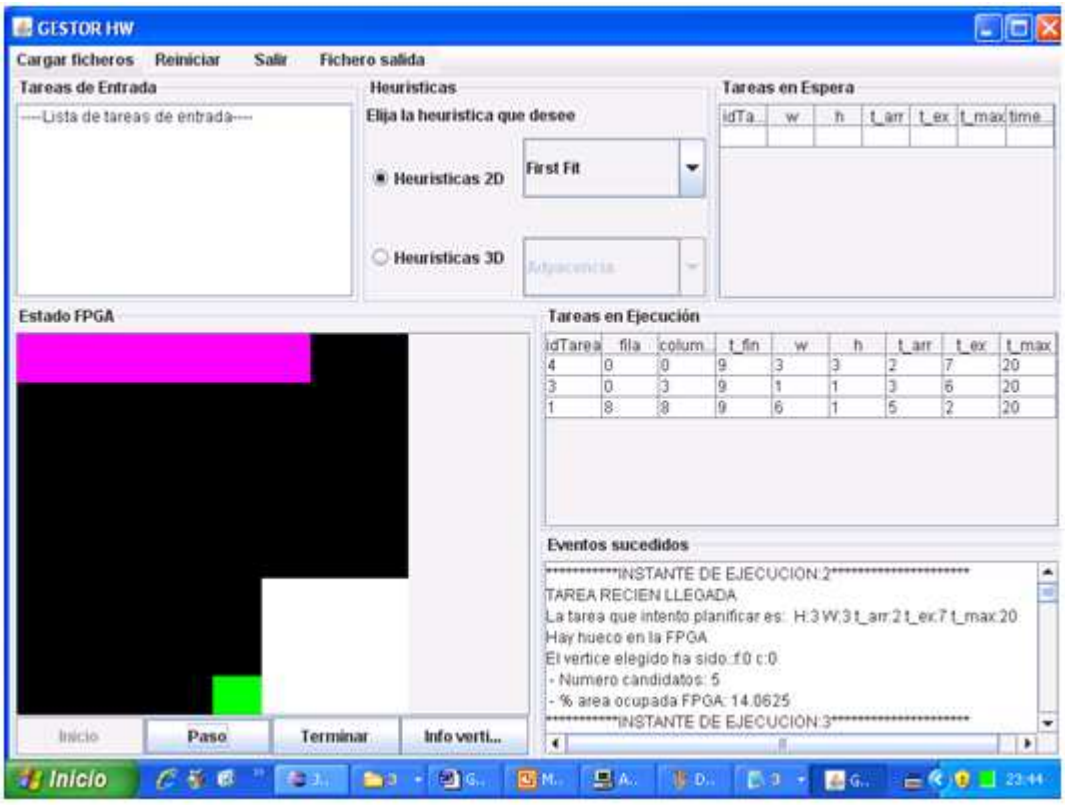


Fig. 80. Finaliza su ejecución la tarea con idTarea = 0

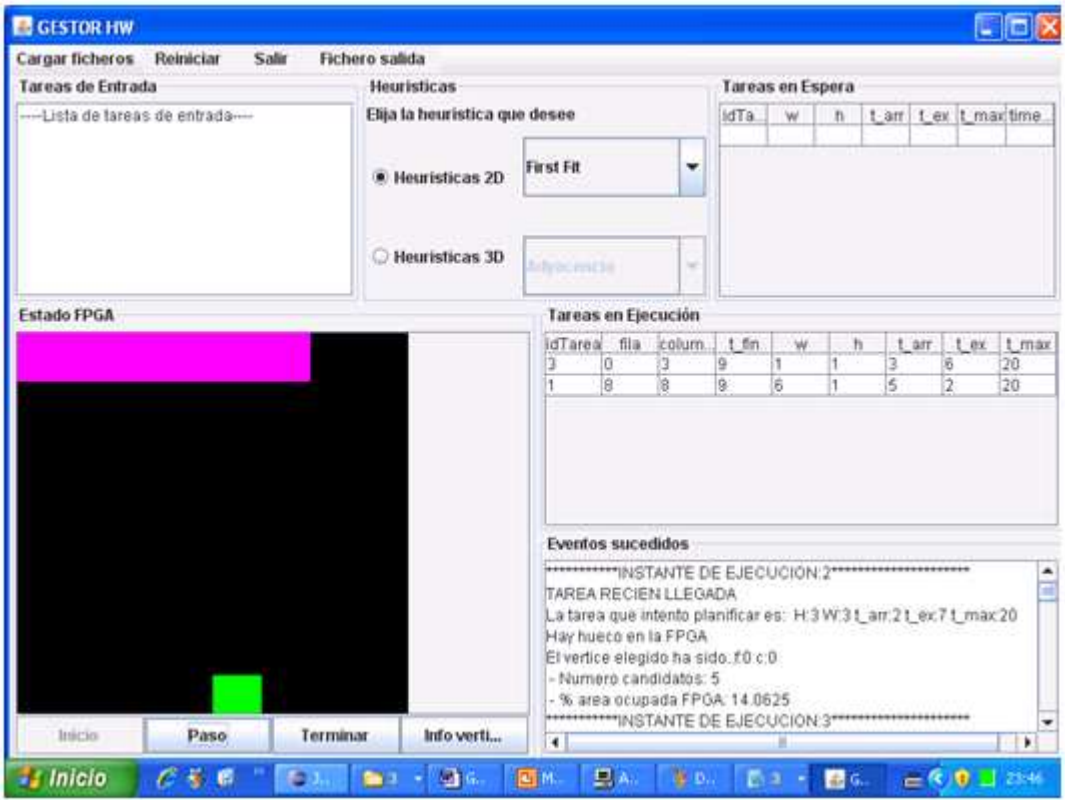


Fig. 81. La tarea con idTarea = 4 termina su ejecución.

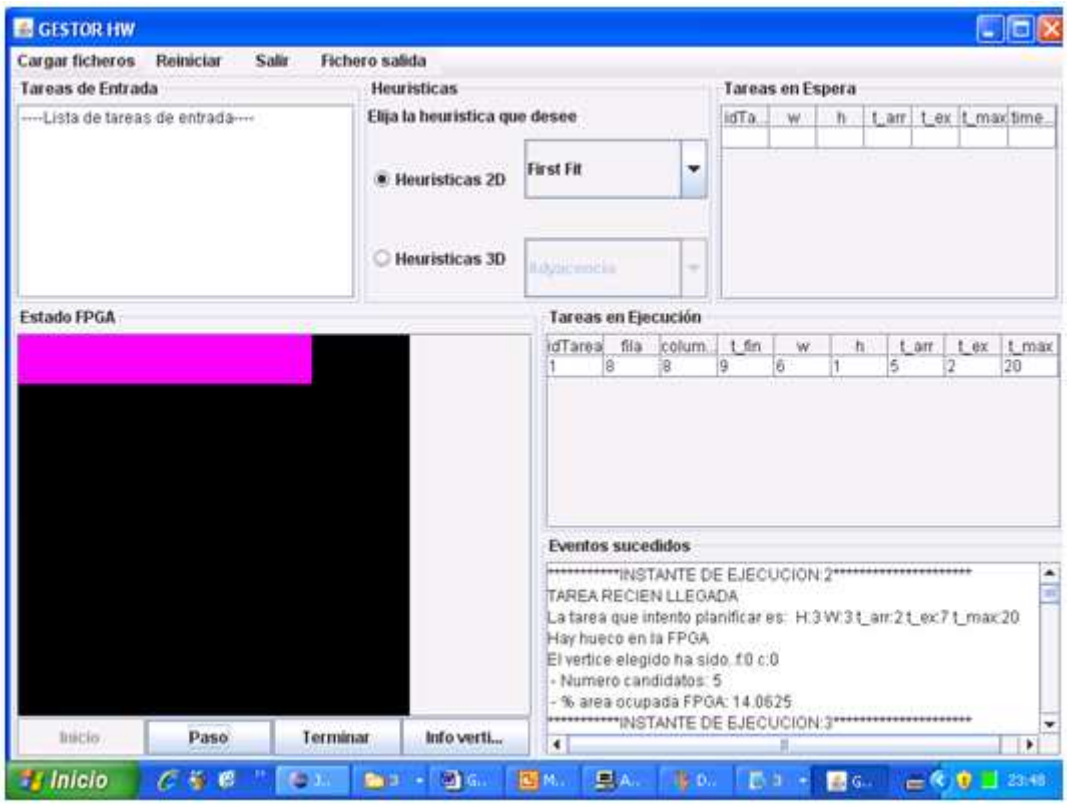


Fig. 82. Finaliza la ejecución la tarea con idTarea = 3.

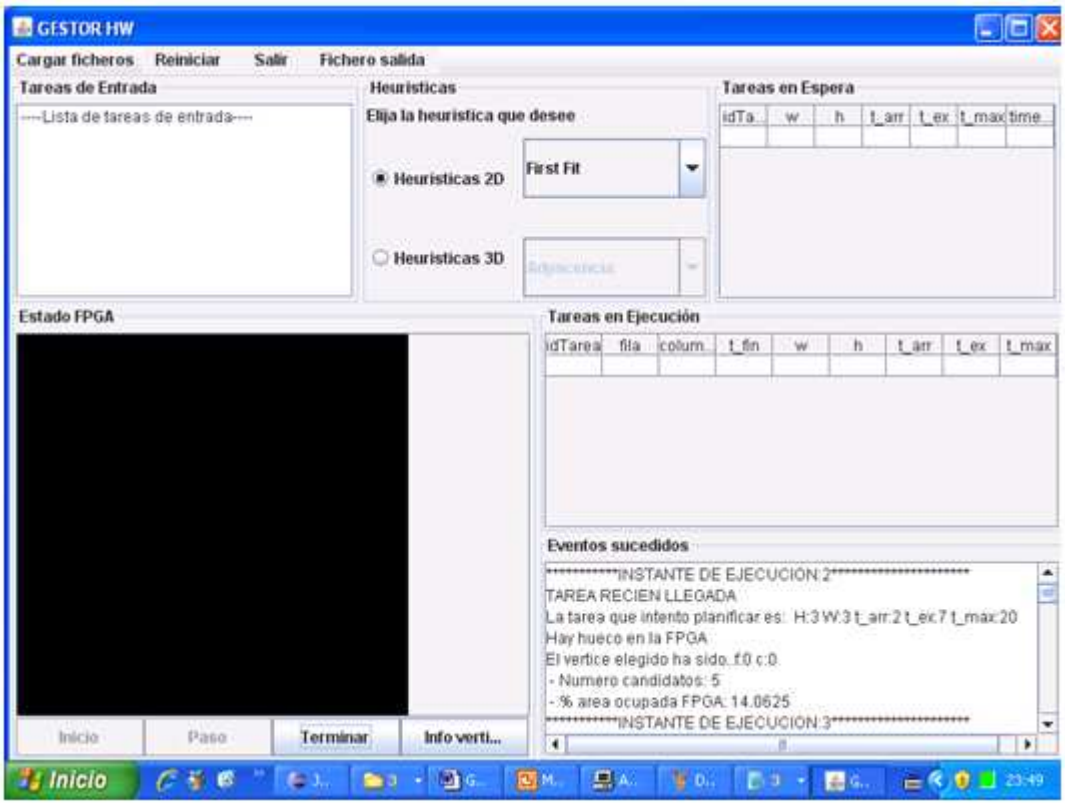


Fig. 83. Finaliza la ejecución de la tarea con idTarea = 1.

En la situación representada de la figura anterior numero 83, se puede ver que no quedan tareas por llegar, ni por terminar de ejecutarse ni tampoco en la lista de tareas en espera, es decir la ejecución del gestor ha terminado. Por tanto, a continuación pulsamos sobre el botón "Terminar", y veremos una pantalla como la que se muestra a continuación.

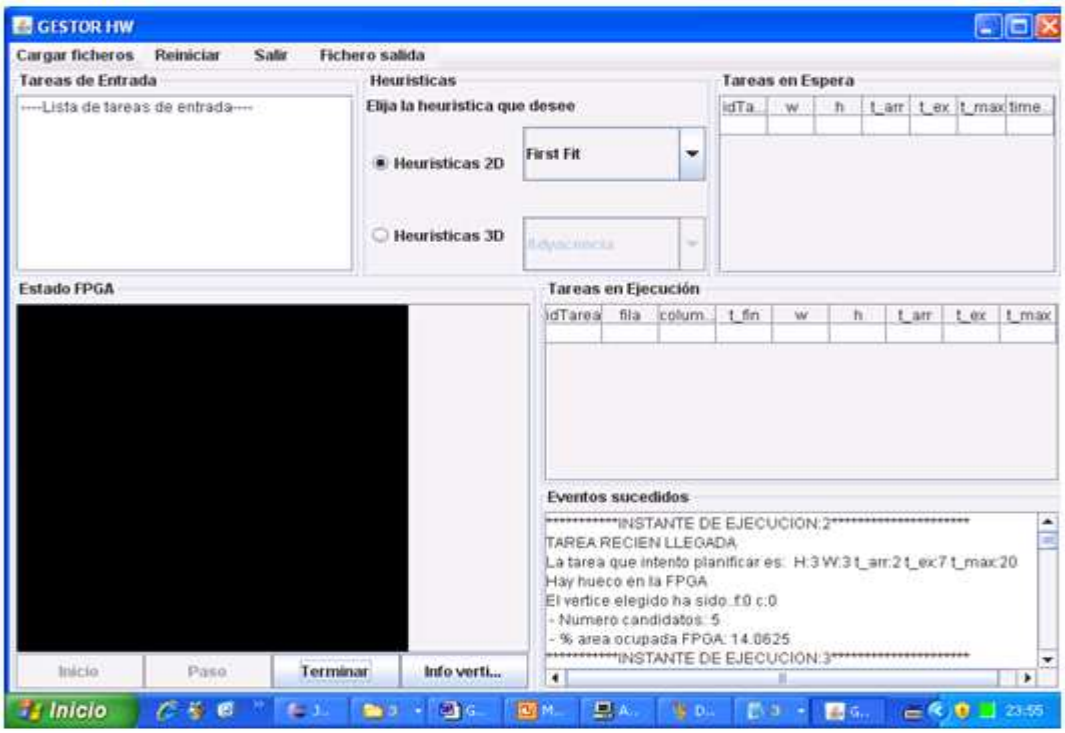


Fig. 84. Tras pulsar el botón Terminar.

Puesto que se quiere un fichero de salida donde quede reflejada la ejecución, habrá que pulsar sobre el botón “Fichero salida”, y se escribirá el fichero “FicheroSalida.txt” con la ruta por defecto determinada, que aparece en la siguiente figura.

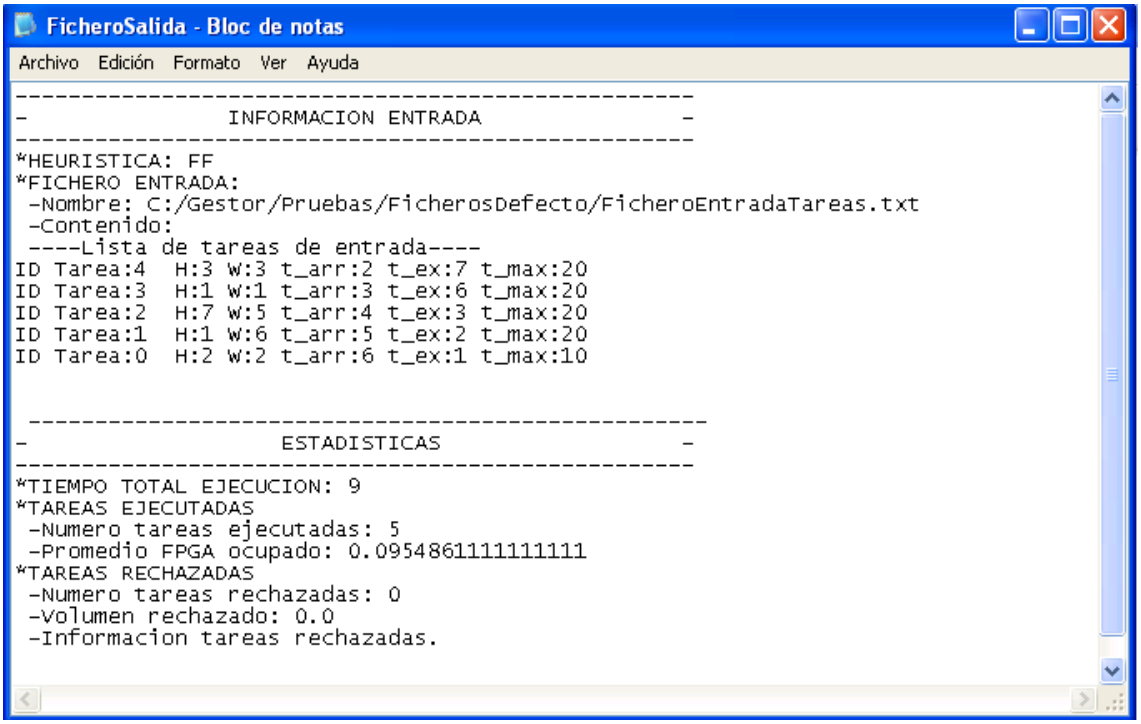


Fig. 85. Fichero de salida de la ejecución mostrada en las figuras anteriores.

A continuación, se quiere ejecutar el Gestor con otro fichero de entrada; así que hay que dar a “Reiniciar” y se verá una pantalla como la que se muestra en la siguiente figura.

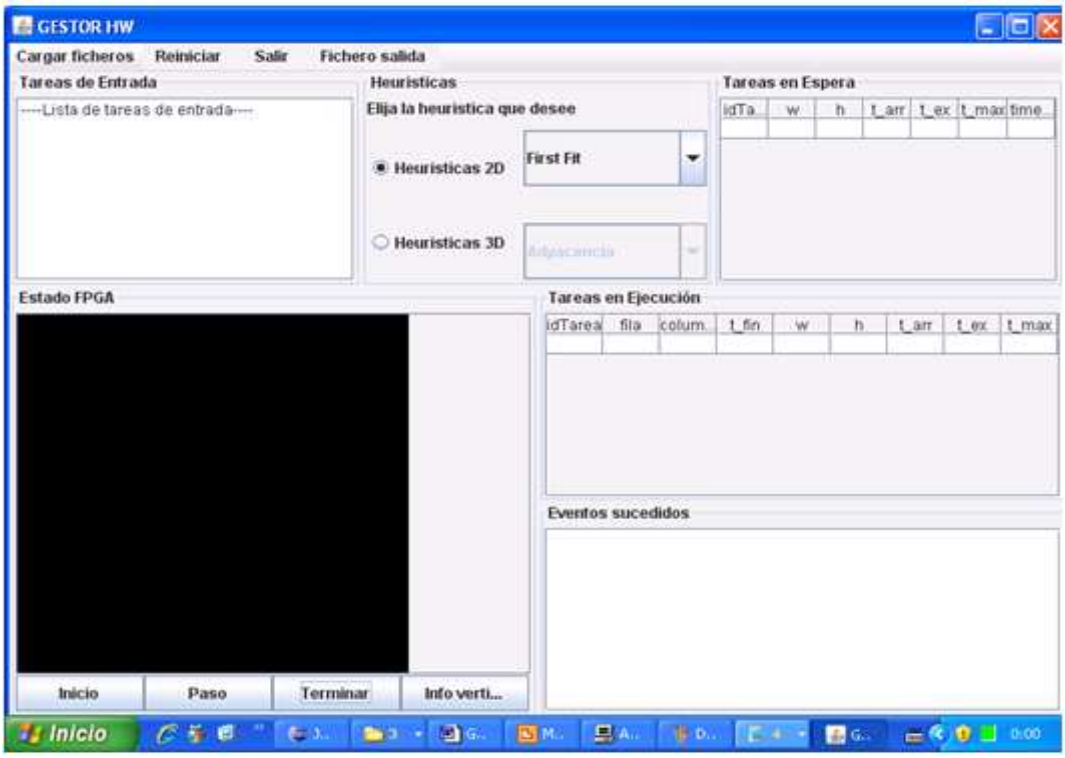


Fig. 86. Pantalla tras reiniciar

Para la siguiente ejecución el fichero de entrada de tareas lo voy a elegir de una determinada ruta, así que pulsare sobre "Cargar ficheros" y después sobre "Elegir ficheros" (tal y como se indica en la figura 86), y se ve una ventana como la que se muestra en el dibujo número 87.

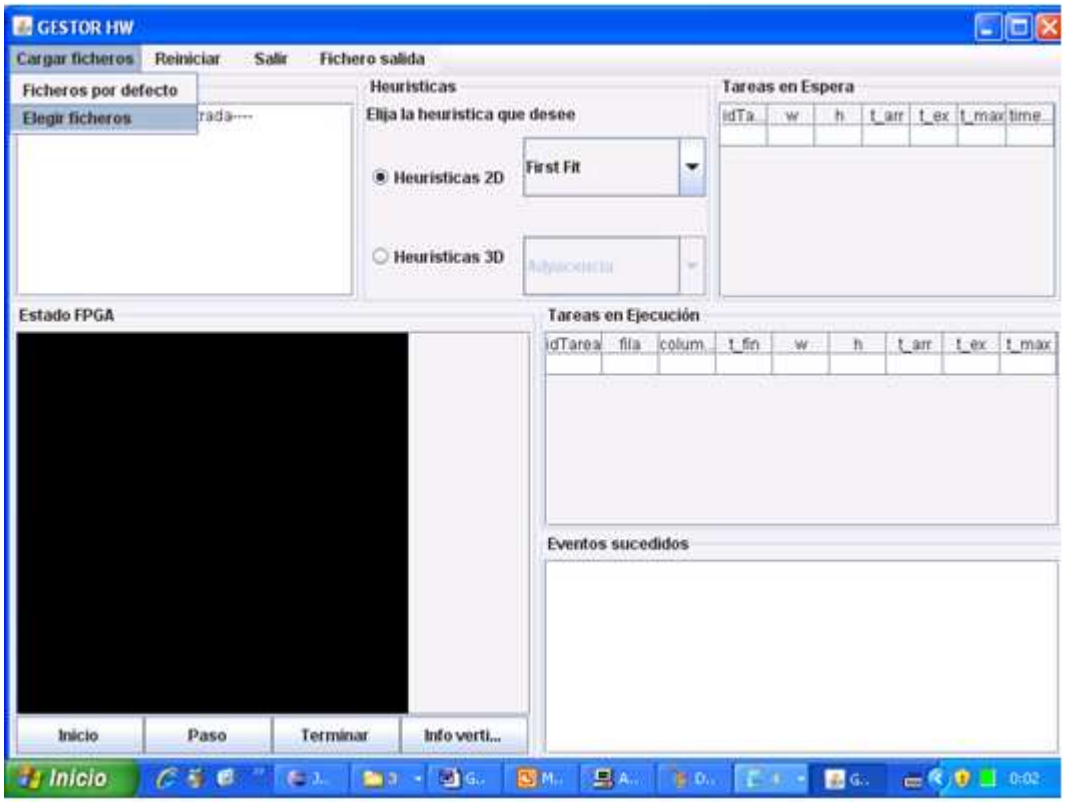


Fig. 87. Cargar ficheros y después Elegir ficheros

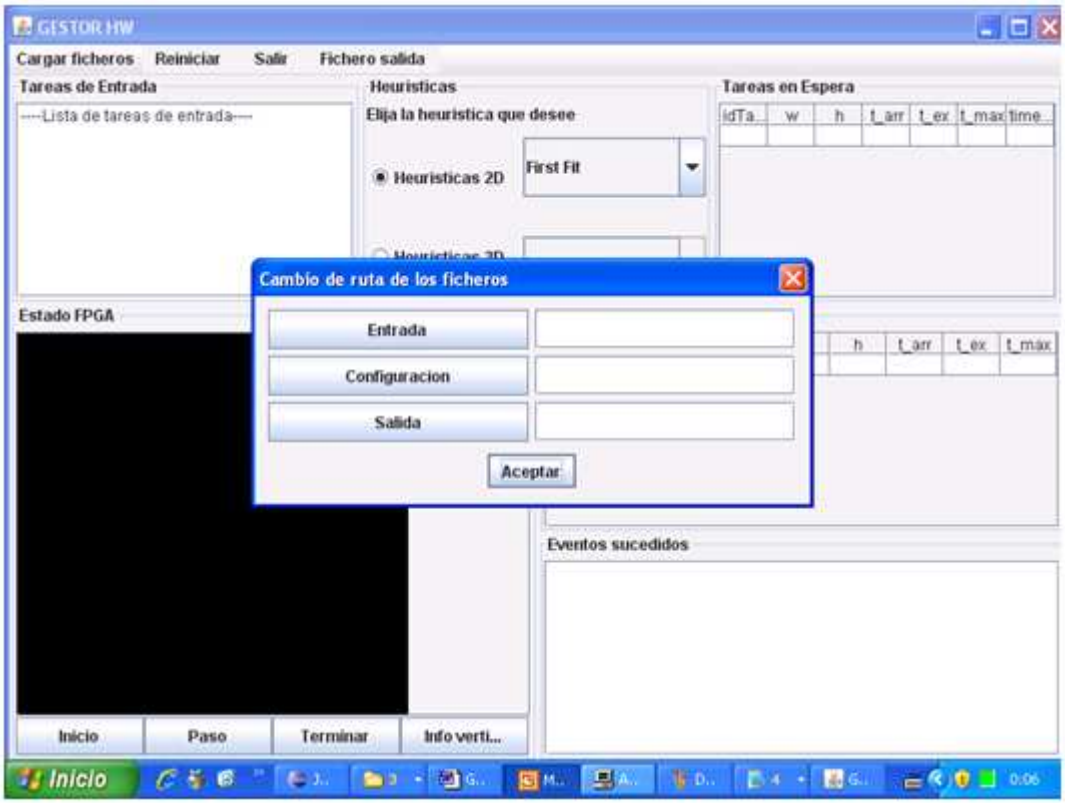


Fig. 88. Tras pulsar "Elegir ficheros"

Después pulsar sobre "Entrada", para cambiar el fichero de entrada de tareas; tras lo cual se verá una pantalla como la que se muestra en la figura 89 y elegir el fichero que se desee.

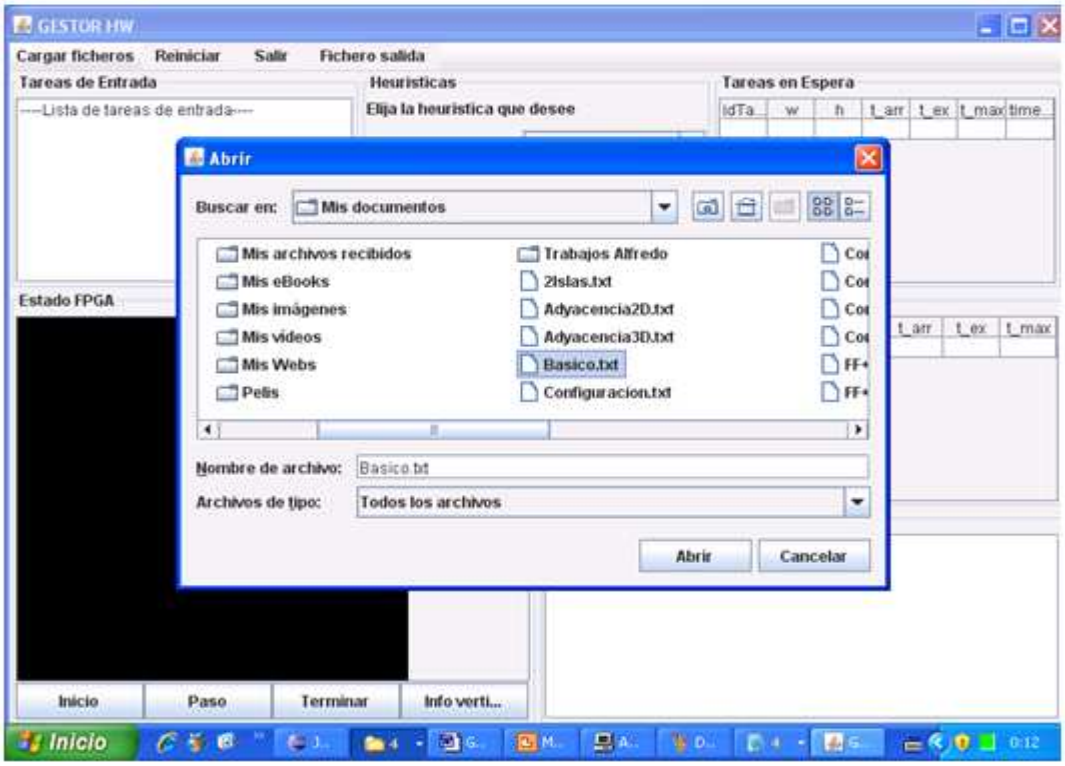


Fig. 89.

Pulsar en “Abrir” y aparece la siguiente ventana.

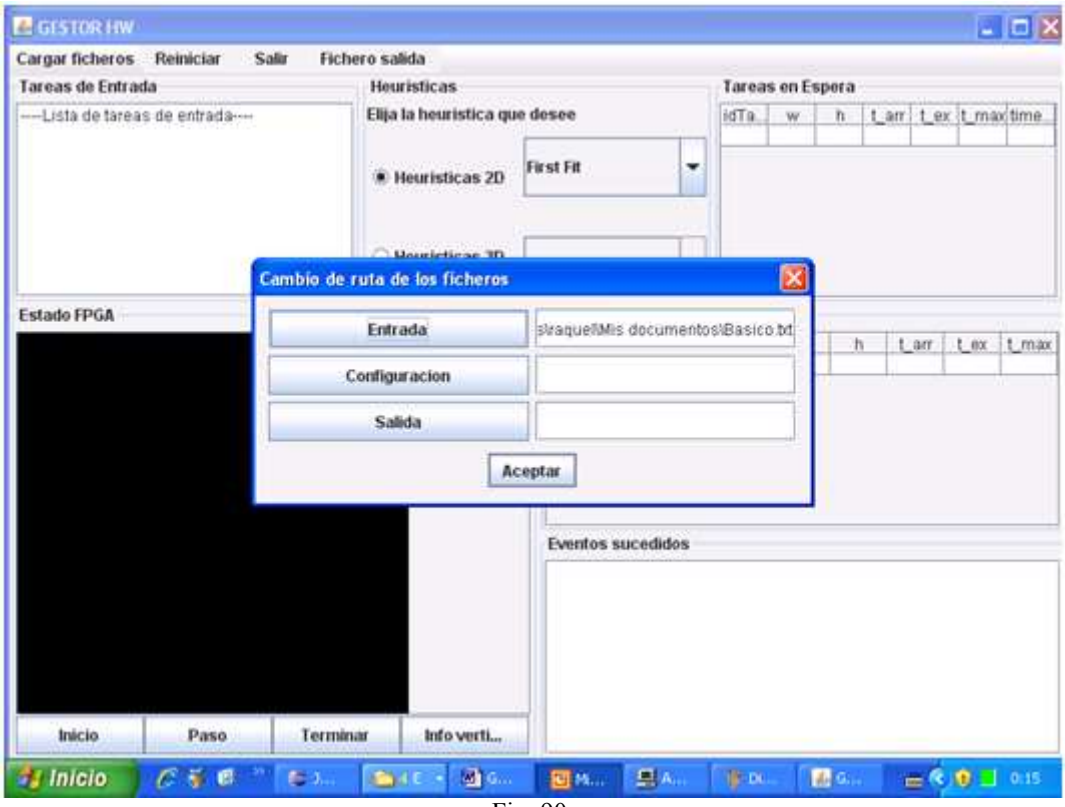


Fig. 90.

Pulsar en “Aceptar”, para que solo se cambie el fichero de datos de entrada y el resto permanezcan con la ruta por defecto; y se muestra la siguiente pantalla.

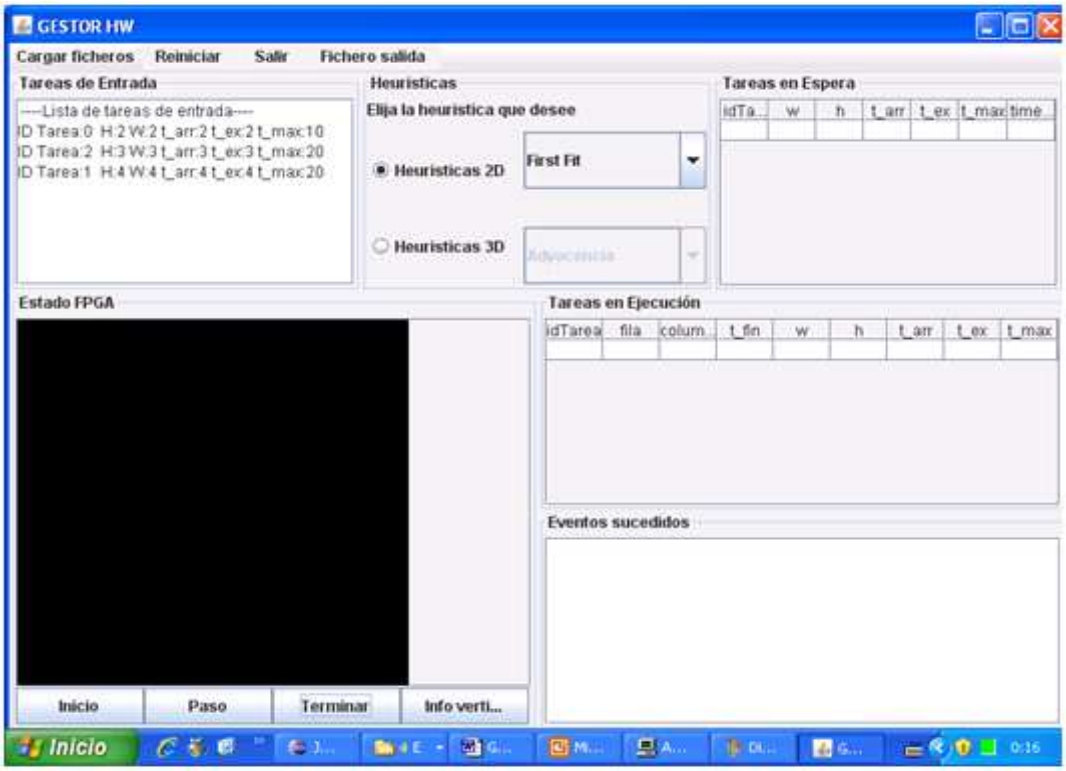


Fig. 91.

En esta ocasión, quiero seleccionar la heurística de Adyacencia 2D para la ejecución, así que tal y como se muestra en la siguiente figura, hay que pulsar sobre el combo box y seleccionar Adyacencia 2D.

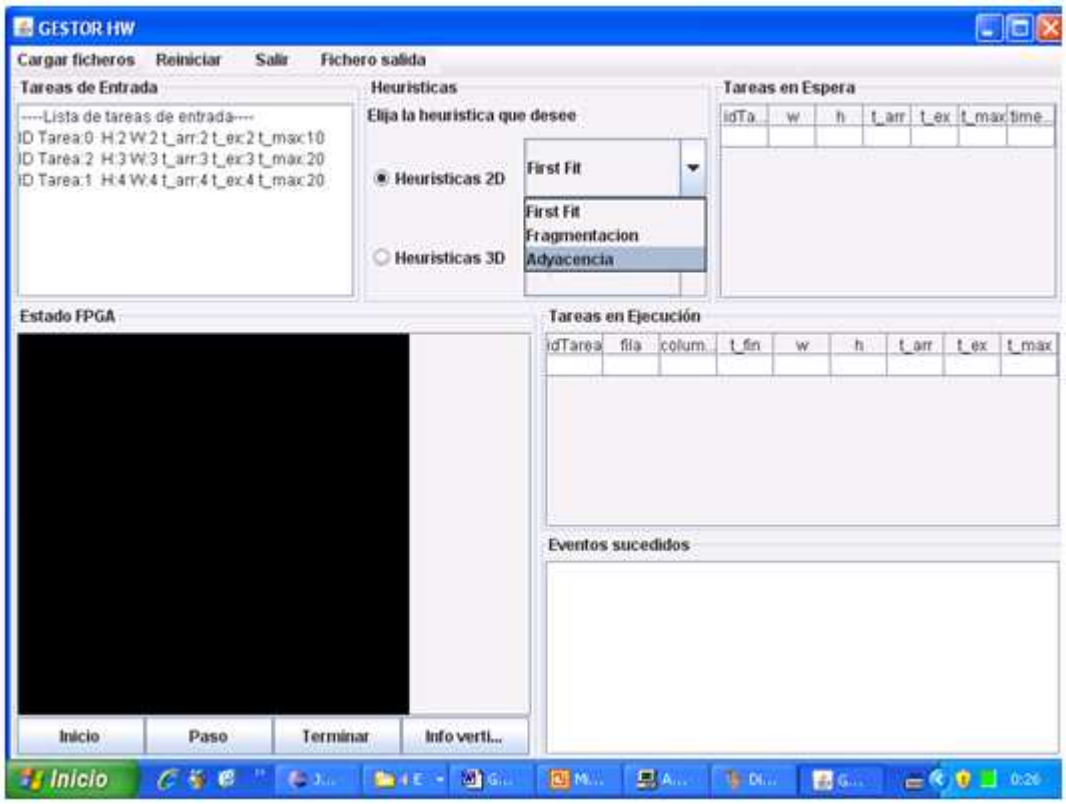


Fig. 92.

A continuación, hay que pulsar sobre “Inicio”, y la ventana del Gestor aparece con la siguiente apariencia.

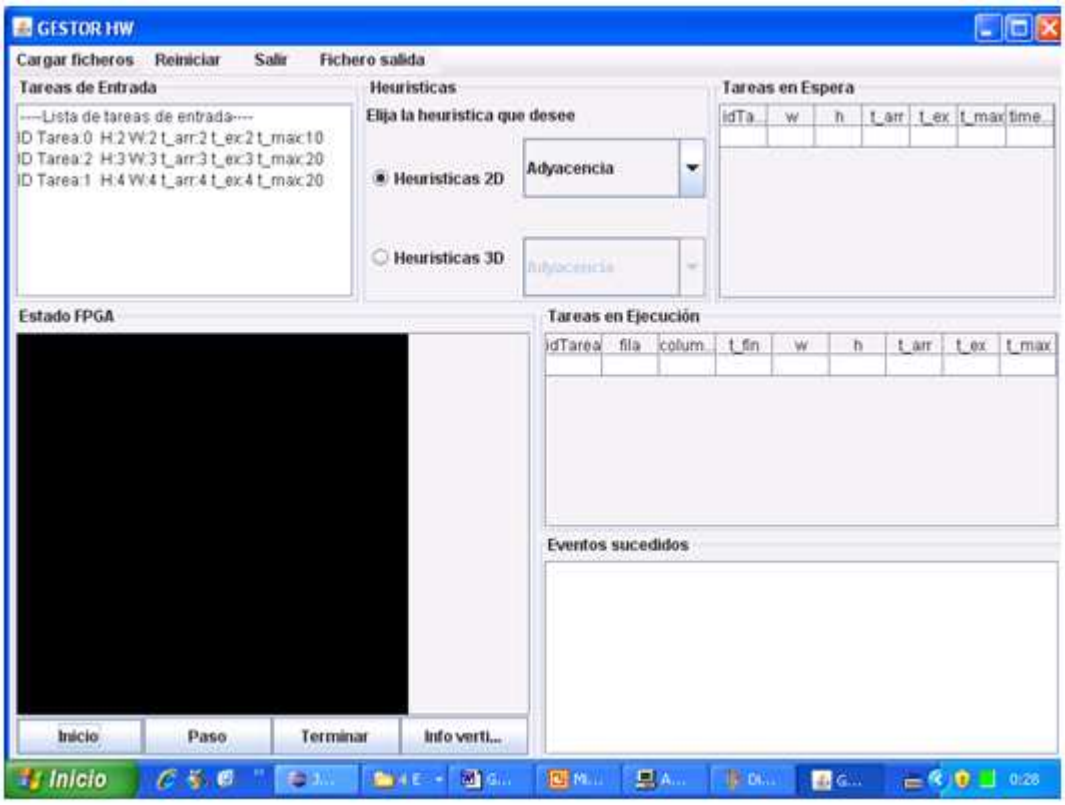


Fig. 93.

Esta vez, en vez de hacer una ejecución paso a paso, opto por hacerla completa de una vez, así que pulso sobre “Terminar” y la pantalla se ve como en la siguiente figura.

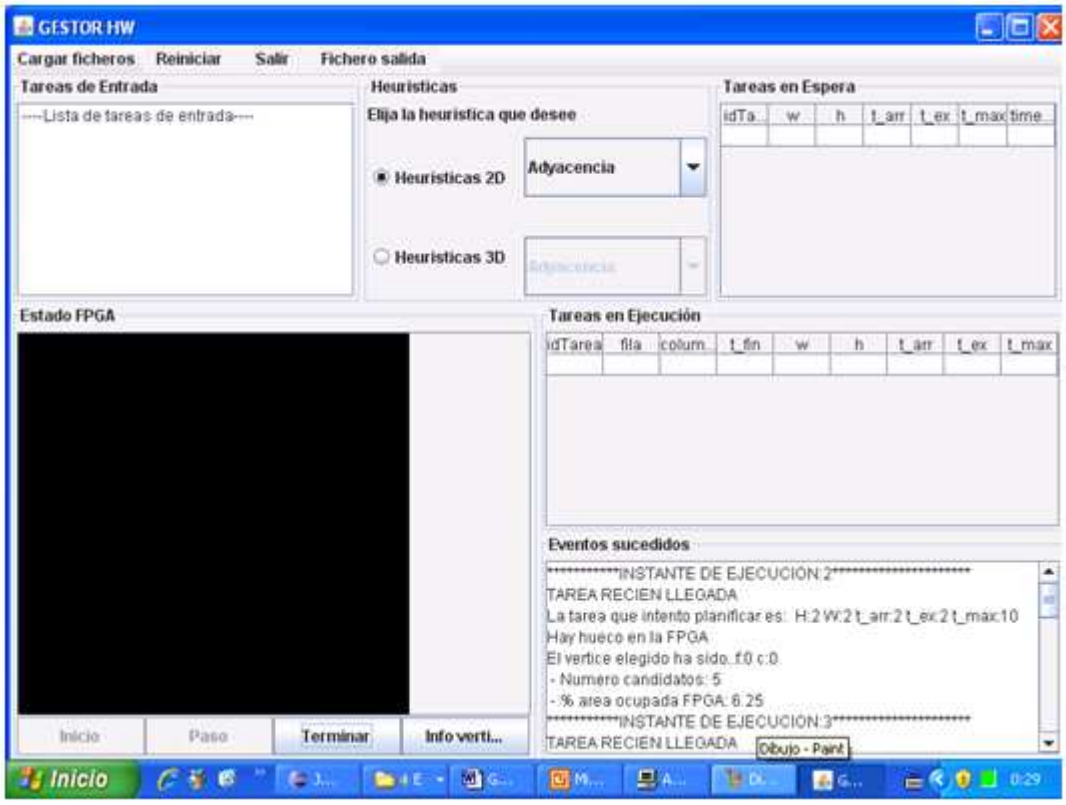


Fig. 94.

Para ver la información sobre la ejecución pulso sobre “Fichero salida” y el fichero se muestra en la figura número 95.

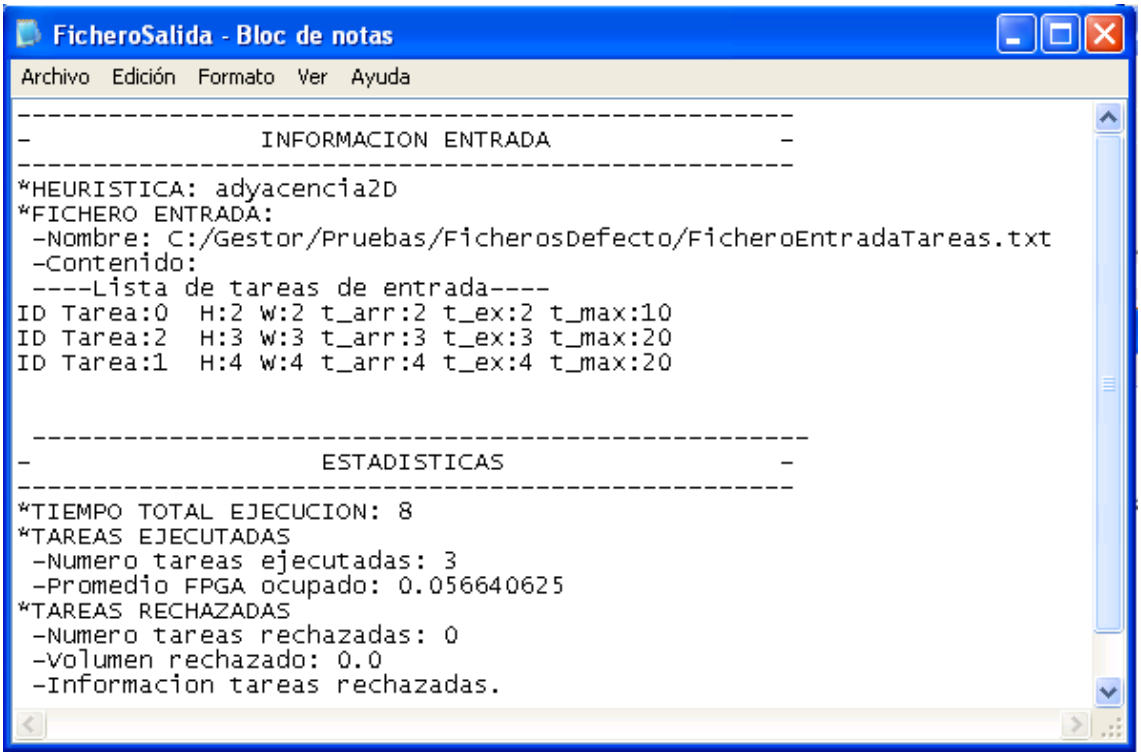


Fig. 95.

Por último, para salir del Gestor se puede hacer mediante la X de la ventana o del botón “Salir”.

Hay que poner de manifiesto que en cada instante de la ejecución, las estructuras auxiliares de la Lista de Tareas en Espera y la Lista de Tareas en Ejecución se actualizan, y por tanto, también su representación en la ventana de la interfaz grafica. Además, en el panel de mensajes, se van añadiendo los nuevos mensajes que surgen de cada paso de la ejecución.

D.Implementación

En este apartado, se introducen trozos de código de algunas de las clases java más relevantes del proyecto como son:

- Principal.java
- Planificador.java
- VertexListUpdater.java
- VertexSelector.java

Principal.java

Es la clase desde la que se lanza la interfaz grafica del gestor.

```
package gui;
/**
 * @author Raquel Sanchez
 * */

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.awt.event.MouseEvent;
import java.awt.event.WindowEvent;
import java.io.IOException;

import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import proyecto.LineaFicheroConfiguracion;
import proyecto.LineaFicheroSalida;
import proyecto.LineaFicheroTareas;
import proyecto.MatrizFPGA;
import proyecto.PlanificadorTareas;


public class Principal extends JFrame {
  private static final long serialVersionUID = 10L;

  private String heuristica="FF";
  private boolean fin=false;
  private boolean reiniciar=false;

  /************Atributos de la aplicacion**********************/
  private String ruta="";
  private String nombreFichEntrada="";
  private String nombreFichConfiguracion="";
  private String nombreFichSalida="";

  private PlanificadorTareas planificador;


  /*************Atributos de la interfaz grafica**************/
  private BorderLayout border;
  private GridLayout gridSuperior;
  private GridLayout gridCentral;
  private GridLayout gridCentralIzq;
  private GridLayout gridCentralDer;

  private JPanel panelSuperior;
  private JPanel panelCentral;
  private JPanel panelCentralIzq;
  private PanelTareasEntrada jPanelTareasEntrada;
  private JPanel panelCentralDer;
```

```
  private PanelFPGA jPanelFPGA;
  private PanelHeuristicas jPanelHeuristicas;
  private PanelTareasEspera jPanelTareasEspera;
  private PanelMensajes jPanelMensajes;
  private PanelTareasEjecucion jPanelTareasEjecucion;

  public Principal() {
    /*Inicializacion de los atributos de la aplicacion*/
    planificador=new PlanificadorTareas(8,8);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  public void inicio(){
    boolean pulsadoInicio = false;
    while (!pulsadoInicio) {
      pulsadoInicio = this.jPanelFPGA.getPulsadoInicio();
    }
    String h=this.jPanelHeuristicas.getHeuristica();
    System.out.println(h);
    if (h.equals(" ")){
       h=heuristica;
    }
    System.out.println(h);
    planificador.setHeuristica(h);
    Dimension tamPantalla = Toolkit.getDefaultToolkit().getScreenSize();
    int altoPantalla=tamPantalla.height;
    int altoPanel=((2*altoPantalla)/3);
    int anchoCasilla=((altoPanel-100)/planificador.getColumnasFPGA());
    jPanelFPGA.setAnchoCasilla(anchoCasilla);
    planificador.inicio();
    System.err.println("El fichero de configuracion
es:"+this.planificador.getFilasFPGA()+" "+this.planificador.getColumnasFPGA());
    actualizaPaneles();
    jPanelFPGA.setPulsadoInicio(false);
    System.out.println("Fuera de inicio");
  }

  public void reiniciar(){
         this.planificador=new PlanificadorTareas(8,8);
         this.actualizaPaneles();
         this.jPanelFPGA.setPulsadoInicio(false);
         this.jPanelFPGA.setPulsadoPaso(false);
         this.jPanelFPGA.setPulsadoTerminar(false);
         this.jPanelFPGA.setEnabledBotonInfo(true);
         this.jPanelFPGA.setEnabledBotonInicio(true);
         this.jPanelFPGA.setEnabledBotonPaso(true);
         this.jPanelFPGA.setEnabledBotonTerminar(true);
  }

  public void paso(){
    System.out.println("Dentro de paso");
    planificador.paso();
    actualizaPaneles();
    getPanelFPGA().setPulsadoPaso(false);
    System.out.println("Fuera de paso");
  }

  public void termina(){
    System.out.println("Dentro de termina");
    planificador.termina();
    this.actualizaPaneles();
    this.jPanelFPGA.setPulsadoTerminar(true);
    this.jPanelFPGA.setEnabledBotonPaso(false);
    System.out.println("Fuera de termina");
  }

  public boolean finEstructuras(){
    boolean fin=false;
    if ((planificador.getListaTareasEntrada().size()>0)||
       (planificador.getListaTareasEjecucion().getTamanyo()>0)||
       (planificador.getListaTareasEspera().getTamanyo()>0)){
```

```
      fin=false;
    }
    else{
      fin=true;
    }
    return fin;
  }

  public static void main(String [] args){
    Principal gui=new Principal();
    gui.pack();
    gui.setVisible(true);

    //Siempre lo voy a ejecutar una primera vez
    gui.inicio();
    boolean pulsadoTermina=false;
    boolean pulsadoPaso=false;
    boolean pulsadoReiniciar=false;

    LineaFicheroSalida lfs=new LineaFicheroSalida();

    while(!pulsadoTermina){
      pulsadoPaso=gui.getPanelFPGA().getPulsadoPaso();
      if (pulsadoPaso){
        gui.paso();
      }
      pulsadoTermina=gui.getPanelFPGA().getPulsadoTerminar();
      pulsadoReiniciar=gui.getReiniciar();
      if ((gui.finEstructuras())||(pulsadoReiniciar)){
        pulsadoTermina=true;
      }
      if (pulsadoTermina){
        gui.termina();
      }
    }

       //Miro a ver si quiero salir de la aplicacion
       boolean finAplicacion=gui.getFin();;
    boolean reinicio=false;
    while(!finAplicacion){
       reinicio=gui.getReiniciar();
       if (reinicio){
              gui.reiniciar();
              gui.inicio();
              pulsadoTermina=false;
              pulsadoPaso=false;

           while(!pulsadoTermina){
             pulsadoPaso=gui.getPanelFPGA().getPulsadoPaso();
             if (pulsadoPaso){
               gui.paso();
             }
             pulsadoTermina=gui.getPanelFPGA().getPulsadoTerminar();
             if (gui.finEstructuras()){
               pulsadoTermina=true;
             }
             if (pulsadoTermina){
               gui.termina();
             }
           }
           gui.setReiniciar(false);
        }
       finAplicacion=gui.getFin();
    }

    lfs.escribeFicheroSalida(gui.getNombreFichSalida(),
                      gui.getNombreFichEntrada(),
              gui.getHeuristica(),
              gui.getPlanificador().getMensajes(),
              gui.getPlanificador().getTareasEjecutadas(),
              gui.getPlanificador().getTareasRechazadas(),
              gui.getPlanificador().getT(),
              gui.getPlanificador().getFilasFPGA(),
              gui.getPlanificador().getColumnasFPGA());
  }

}
```

Planificador.java

```
package proyecto;
/**
 * @author Raquel Sanchez
 * */
/**Este es el metodo principal del programa
 * Se encarga de hacer la planificacion de las tareas
 *
 * Vamos a tomar como entrada un fichero de texto donde estan las tareas que
 * queremos planificar junto con las caracteristicas de cada una
 *
 * Otra entrada va a ser un fichero independiente que he creado yo, donde
 * se especifica la configuracion de la FPGA, como el tamaño, el parametro k...
 *
 * Como salida vamos a devolver un fichero de texto donde se especificaran
 * los resultados y la configuracion que hemos utilizado
 */
/**Primero comprobamos si podemos ejecutar en la FPGA una tarea que se encuentre
 * en la lista de tareas en espera
 * Nos podemos encontrar en distintas situaciones:
 * - la lista esta vacia=> se desestima la posibilidad de elegir una tarea
 * de la lista de tareas en espera
 * - la primera tarea de la lista tiene time_out=0 => se saca la tarea de la lista
 * y se intenta colocar la siguiente tarea en la FPGA si se puede
 * - la primera tarea tiene time_out>0 y tiene hueco en la FPGA => entonces
 * se elige esa tarea y se pasa a planificarla
 * -la primera tarea tiene time_out>0 pero no coge en la FPGA => se intenta
 * planificar la siguiente tarea que haya en el fichero de entrada cuyo tiempo
 * de llegada coincida con el instante actual
 */

/********* UNA VEZ QUE TENGO ELEGIDA LA TAREA A PLANIFICAR HAY QUE
 * -ELEGIR VERTICE SEGUN LA HEURISTICA
 * -INSERTAR LA TAREA EN LA LISTA DE TAREAS EN EJECUCION
 * -INSERTAR LA TAREA EN LA FPGA
 * -ACTUALIZAR LA LISTA DE VERTICES
 * -ANALIZAR LA LISTA DE VERTICES
 * -OBTENER LA METRICA DE FRAGMENTACION
 * -GESTIONAR POSIBLE FRAGMENTACION
 *
 */


import java.util.LinkedList;
import java.util.Vector;

import comun.listaTareasEjecucion.ListaTareas;
import comun.listaTareasEjecucion.NodoTarea;
import comun.listaTareasEjecucion.ParametrosTarea;
import comun.listaTareasEspera.ListaTareasEspera;
import comun.listaTareasEspera.TareasEspera;
import comun.listaVertices.ListasVertices;
import comun.listaVertices.NodoLista;
import comun.listaVertices.NodoVertice;
import comun.listaVertices.OperacionesVLS;

public class PlanificadorTareas {

  private String mensajes = "";
  private Vector tareasEjecutadas=new Vector();
  private Vector tareasRechazadas=new Vector();
  private Vector tareasLookAheadEsperando=new Vector();

  private int t; //Es un contador que va a simular el paso del tiempo, por unidades de
tiempo
  private int t_anterior;
  private String ficheroEntradaTareas;
  private String ficheroEntradaConfiguracion;
  private String ficheroSalida;
  private int idTarea; //el identificador de la tarea sera el numero de linea que ocupe
en el fichero de entrada
  private int t_conf; //viene determinado por el fichero de entrada de configuracion
```

```
  private int filasFPGA;
  private int columnasFPGA;
  private ListaTareasEspera listaTareasEspera;
  private ListaTareas listaTareasEjecucion;
  private VertexSelector selectorVertice;
  private VertexListUpdater actualizadorVLS;

  private ListasVertices vls;
  private Vector candidatos;
  private Vector listaTareasEntrada;
  private OperacionesVLS ops;
  private MatrizFPGA m;

  //Cuando haya la interfaz grafica hay que quitarlo
  private String heuristica;


  //Constructoras
  public PlanificadorTareas(int filasFPGA, int columnasFPGA) {
    idTarea = 1;
    t = 0;
    t_conf = 0; //Habra que inicializarlo como corresponda
    this.filasFPGA = filasFPGA;
    this.columnasFPGA = columnasFPGA;
    heuristica = " ";
    ficheroEntradaTareas = " ";

    listaTareasEspera = new ListaTareasEspera();
    listaTareasEjecucion = new ListaTareas();

    int[][] matrizFPGA = new int[filasFPGA][columnasFPGA];
    for (int f=0;f<filasFPGA;f++){
      for (int c=0;c<columnasFPGA;c++){
        matrizFPGA[f][c]=-1;
      }
    }

    ops = new OperacionesVLS(filasFPGA, columnasFPGA);
    m = new MatrizFPGA();
    m.setMatriz(matrizFPGA);
    vls = ops.primeraVLS();

    actualizadorVLS = new VertexListUpdater(filasFPGA,columnasFPGA);
    selectorVertice = new VertexSelector(filasFPGA, columnasFPGA);
    listaTareasEntrada = new Vector();

    candidatos = new Vector();
  }

  public void inicio() {
    vls = ops.primeraVLS();
    candidatos = ops.listaCandidatos(vls, m.getMatriz());
  }

  public void paso() {
    int t_llegada, t_espera, t_salida;
    idTarea = -1;

    t_llegada = 1000;
    t_espera = 1000;
    t_salida = 1000;

    int tareaListaEspera = -1;
    /** -1: valor por defecto
     *  0 : realizo la operacion con una tarea recien llegada o con una tarea que
finaliza
     *  1 : alcanzado time_out de una tarea de la lista de espera
     *  2 : una de las tareas de la lista tiene hueco en la FPGA
     */

    if (listaTareasEntrada.size() > 0) {
      t_llegada = ( (ParametrosTarea) ( (LineaFicheroTareas)
listaTareasEntrada.elementAt(0)).getParametros()).getT_arr();
    }
    if (listaTareasEspera.getTamanyo() > 0) {
       tareaListaEspera = planificarTareaListaEspera();
      if (tareaListaEspera==1){
```

```
            t_espera=t;
          }
          else{
             if (tareaListaEspera==2){
                    t_espera=t_anterior;
             }
          }
        }
        if (listaTareasEjecucion.getTamanyo() > 0) {
          NodoTarea tarea = (NodoTarea) listaTareasEjecucion.getTareaConMenorTFin();
          t_salida = tarea.getTfin();
        }

        t = Math.min(t_salida, Math.min(t_llegada, t_espera));

        System.out.println("********* INSTANTE DE EJECUCION:" + t
+"**********************");
        mensajes = mensajes + "***********INSTANTE DE EJECUCION:" + t
+"*********************\r\n";
        if (tareaListaEspera == 1) {
           sacaTareaListaEspera();
        }
        else {
          if (tareaListaEspera == 2) {
            //Una tarea de las que estan en la lista de espera tiene hueco para
            //ejecutarse en la FPGA
           t=t_anterior;
           colocaTareaListaEspera();
          }
          else {
            if (t == t_salida) { //Ha terminado de ejecutarse una tarea de la FPGA
              System.out.println("TAREA FINALIZADA");
              mensajes = mensajes + "TAREA FINALIZADA" + "\r\n";
              NodoTarea
tareaSaliente=(NodoTarea)listaTareasEjecucion.getTareaConMenorTFin();
              idTarea = tareaSaliente.getIdTarea();
              System.out.println("Ha terminado la tarea con id: " + idTarea);
              mensajes = mensajes + "Ha terminado la tarea con id:" + idTarea +"\r\n";
              sacaTareaFPGA(m.getMatriz(),tareaSaliente);
              vls = actualizadorVLS.actualizaVLS(listaTareasEjecucion, m.getMatriz());
              candidatos = ops.listaCandidatos(vls, m.getMatriz());
            }
            else {
              if (t == t_llegada) {
                idTarea = ( (LineaFicheroTareas)
listaTareasEntrada.elementAt(0)).getIdTarea();
                ParametrosTarea paramTarea = (
(LineaFicheroTareas)listaTareasEntrada.elementAt(0)).getParametros();
                System.out.println("TAREA RECIEN LLEGADA");
                mensajes = mensajes + "TAREA RECIEN LLEGADA\r\n";
                System.out.print("La tarea que intento planificar es: ");
                mensajes = mensajes + "La tarea que intento planificar es: ";
                System.out.println("Parametros tarea.w:" + paramTarea.getW() +
                                   " ,h:" + paramTarea.getH() +
                                   " ,t_ex:" + paramTarea.getT_ex() +
                                   " ,t_arr:" + paramTarea.getT_arr() +
                                   " ,t_max:" + paramTarea.getT_max());
                mensajes = mensajes + (paramTarea.escribeParametros()) + "\r\n";
                colocaTareaLlegada(paramTarea);
              }
            }
          }
        }
        mensajes=mensajes+obtenInfoTrazaComun();
        t_anterior=t;
      }

      public int planificarTareaListaEspera() {
        int n = 0;
        int i = 0;
        boolean encontrado = false;
        TareasEspera tareaAux = new TareasEspera();
        //para cuando solo quedan tareas en la lista de espera
        if ((this.getListaTareasEspera().getListaEspera().size()>0)&&
                  (this.getListaTareasEjecucion().getTamanyo()==0)&&
                  (this.getListaTareasEntrada().size()==0)&&
```

```
       (!huecoTarea((ParametrosTarea)((TareasEspera)this.getListaTareasEspera().getLista
Espera().get(0)).getParam()))){

       t=((TareasEspera)this.getListaTareasEspera().getListaEspera().get(0)).getT_timeou
t();
       n=1;
    }
    else{
           if ((t >
(((TareasEspera)this.getListaTareasEspera().getListaEspera().get(0)).getT_timeout())){
             n = 1;
           }
    }
    if (n==0){
           while ( (i < (this.getListaTareasEspera().getTamanyo())) && (!encontrado)) {
             tareaAux =
(TareasEspera)this.getListaTareasEspera().getListaEspera().get(i);
             int idTareaAux=tareaAux.getIDTarea();
             if ((huecoTarea( (ParametrosTarea) tareaAux.getParam()))&&
                       ((!estaEsperando(idTareaAux))||

((estaEsperando(idTareaAux))&&(siguienteEvento(idTareaAux))))){
               encontrado = true;
               n = 2;
             }
             else {
               i++;
             }
           }
    }
    return n;
  }


  public void termina() {
    while ( (this.listaTareasEjecucion.getTamanyo() > 0) ||
           (this.listaTareasEntrada.size() > 0) ||
           (this.listaTareasEspera.getTamanyo() > 0)) {
      paso();
    }
  }

  public void colocaTareaLlegada(ParametrosTarea p) {
    candidatos = ops.listaCandidatos(vls, m.getMatriz());
    if (huecoTarea(p)) {
      System.out.println("-Hay hueco en la FPGA");
      mensajes = mensajes + "Hay hueco en la FPGA\r\n";
      NodoVertice vElegido = selectorVertice.selectorVertice(m, heuristica, vls,p,this);
      //La distincion de casos es necesaria por la heuristica de look_ahead
      if ((ops.estaEnListaCandidatos(vElegido,
candidatos))&&(ops.esCandidatoEspacio(m.getMatriz(),
vElegido,ops.devuelveVLdeVertice(this.vls, vElegido), p))){
             System.out.println("El vertice elegido ha sido...f:" + vElegido.getFila()
+" c:" + vElegido.getColumna());
             mensajes = mensajes + "El vertice elegido ha sido..f:" + vElegido.getFila()
+" c:" + vElegido.getColumna() + "\r\n";
             int t_fin = t + t_conf + (p.getT_ex());
             NodoTarea nodoTareaElegida = new NodoTarea(idTarea,
t_fin,vElegido.getFila(),vElegido.getColumna(), p,t);
             listaTareasEjecucion.addTarea(nodoTareaElegida);
             colocaTareaFPGA(m.getMatriz(), nodoTareaElegida,vls);
             vls = actualizadorVLS.actualizaVLS(listaTareasEjecucion, m.getMatriz());
             candidatos = ops.listaCandidatos(vls, m.getMatriz());
             borraPrimeraListaTareasEntrada();
      }
      else{
             System.out.println("LOOK_AHEAD.EJECUCION RETRASADA.El vertice elegido ha
sido...f:" + vElegido.getFila() +" c:" + vElegido.getColumna());
             mensajes = mensajes + "LOOK_AHEAD.EJECUCION RETRASADA.El vertice elegido ha
sido..f:" + vElegido.getFila() +" c:" + vElegido.getColumna() + "\r\n";
             TareasEspera tareaEspera = new TareasEspera();
             tareaEspera.setIDTarea(idTarea);
             tareaEspera.setParam(p);
             tareaEspera.setSiguiente(null);
             int t_timeOut = p.getT_max() - p.getT_ex()-t_conf;
             tareaEspera.setT_timeout(t_timeOut);
```

```
              listaTareasEspera.addTareaOrdenada(tareaEspera);
              int [] info=new int [2];
              info[0]=idTarea;

info[1]=((NodoTarea)getListaTareasEjecucion().getTareaConMenorTFin()).getTfin();
              tareasLookAheadEsperando.add(info);
              borraPrimeraListaTareasEntrada();
      }
    }
    else { //se coloca en la lista de tareas en espera
      System.out.println("-No hay hueco en la FPGA y meto la tarea en la lista de
espera");
      mensajes = mensajes +"No hay hueco en la FPGA y meto la tarea en la lista de
espera\r\n";
      TareasEspera tareaEspera = new TareasEspera();
      tareaEspera.setIDTarea(idTarea);
      tareaEspera.setParam(p);
      tareaEspera.setSiguiente(null);
      int t_timeOut = p.getT_max() - p.getT_ex();
      tareaEspera.setT_timeout(t_timeOut);
      listaTareasEspera.addTareaOrdenada(tareaEspera);
      borraPrimeraListaTareasEntrada();
    }
  }

  public void colocaTareaListaEspera() {
    TareasEspera tarea=new TareasEspera();
    ParametrosTarea p = new ParametrosTarea();
    NodoVertice vElegido = new NodoVertice();
    int t_fin = 0;

    boolean encontrado = false;
    int i = 0;
    LinkedList listaAux = listaTareasEspera.getListaEspera();

    while ( (!encontrado) && (i < listaAux.size())) {
      tarea=(TareasEspera) listaAux.get(i);
      p = (ParametrosTarea)tarea.getParam();
      boolean sePuede=true;
      //En el caso de que sea una tarea que esta esperando por look ahead hay que ver
mas cosas
      int idTareaLookAhead=-1;
      int tTareaLookAhead=-1;
      int [] arrayAux=new int[2];
      for(int k=0;k<tareasLookAheadEsperando.size();k++){
         arrayAux=(int [])tareasLookAheadEsperando.get(k);
         idTareaLookAhead=arrayAux[0];
         tTareaLookAhead=arrayAux[1];
         if (idTareaLookAhead==tarea.getIDTarea()){
                if (t<=tTareaLookAhead){
                       sePuede=false;
                }
         }
      }

      if ((huecoTarea(p))&&(sePuede)) {
        vElegido = selectorVertice.selectorVertice(m, heuristica, vls, p,this);
        if (ops.estaEnListaCandidatos(vElegido, candidatos)){
            System.out.println("LISTA DE TAREAS EN ESPERA");
            mensajes = mensajes + "LISTA DE TAREAS EN ESPERA\r\n";
               System.out.println("-Coloco la tarea que ocupa la posicion " + i +" en la
lista de tareas en espera");
               mensajes = mensajes + "Coloco la tarea que ocupa la posicion " + i +" en
la lista de tareas en espera\r\n";
               System.out.println("El vertice elegido ha sido...f:" + vElegido.getFila()
+" c:" + vElegido.getColumna());
               mensajes = mensajes + "El vertice elegido ha sido...f:"
+vElegido.getFila() + " c:" + vElegido.getColumna() + "\r\n";
               t_fin = t + t_conf + (p.getT_ex());
               idTarea = ( (TareasEspera) listaAux.get(i)).getIDTarea();
               NodoTarea nodoTareaElegida = new NodoTarea(idTarea,
t_fin,vElegido.getFila(), vElegido.getColumna(), p,t);
               listaTareasEjecucion.addTarea(nodoTareaElegida);
               colocaTareaFPGA(m.getMatriz(), nodoTareaElegida,vls);
               vls = actualizadorVLS.actualizaVLS(listaTareasEjecucion,
m.getMatriz()/*,this.vls,true,nodoTareaElegida*/);
               candidatos = ops.listaCandidatos(vls, m.getMatriz());
```

```
                listaTareasEspera.removeTareaPorIdentificador(idTarea);
                //muestraEstructuras();
                encontrado = true;
          }
          else{
                System.err.println("Tarea de look ahead esperando");
          }
        }
        else {
             System.out.println("LISTA DE TAREAS EN ESPERA");
             mensajes = mensajes + "LISTA DE TAREAS EN ESPERA\r\n";
          System.out.println(
              "-He intentado meter una tarea de la lista de espera pero no entra");
          mensajes = mensajes +
              "He intentado meter una tarea de la lista de espera pero no entra\r\n";
          i++;
        }
      }
    }

    public void sacaTareaListaEspera() {
      boolean distinto = false;
      int i = 0;
      LinkedList listaAux = (LinkedList) listaTareasEspera.getListaEspera();
      TareasEspera tarea = new TareasEspera();

      System.out.println("LISTA DE TAREAS EN ESPERA");
      mensajes = mensajes + "LISTA DE TAREAS EN ESPERA\r\n";
      System.out.println("Se ha alcanzado el time_out de una tarea de la lista de
espera");
      mensajes = mensajes + "Se ha alcanzado el time_out de una tarea de la lista de
espera\r\n";

      while ( (!distinto) && (i < listaAux.size())) {
        tarea = (TareasEspera) listaAux.get(i);
        if (tarea.getT_timeout() <= t) {
             System.out.println("La tarea a desestimar tiene las siguientes
carcateristicas: " +
                        "Posicion."+i+" "+tarea.escribeTareaEspera());
             mensajes = mensajes + "La tarea a desestimar tiene las siguientes
carcateristicas: " +
                        "Posicion."+i+" "+tarea.escribeTareaEspera()+"\r\n";
             tareasRechazadas.add((TareasEspera)tarea);
             listaTareasEspera.removeTareaPorTimeout(tarea.getT_timeout());
          i++;
        }
        else {
          distinto = true;
        }
      }
    }

}
```

VertexListUpdater.java

Implementa el módulo que se corresponde con el Actualizador de Listas de Vértices. Dentro de esta clase se hace uso de un algoritmo de busqueda de arboles, que es Primero en Profundidad, para comprobar la existencia de islas y poder tartarlas de un modo diferente

```
package proyecto;
/**
 * @author Raquel Sanchez
 * */
import java.util.Vector;

import utilidades.Busquedas;

import comun.listaTareasEjecucion.ListaTareas;
import comun.listaVertices.ListasVertices;
import comun.listaVertices.NodoLista;
import comun.listaVertices.NodoVertice;
import comun.listaVertices.OperacionesVLS;
```

```
public class VertexListUpdater {

    private int filasFPGA;
    private int columnasFPGA;

    OperacionesVLS ops;

    public VertexListUpdater(int filasFPGA, int columnasFPGA){
        this.filasFPGA = filasFPGA;
        this.columnasFPGA = columnasFPGA;
        ops = new OperacionesVLS (filasFPGA, columnasFPGA);
    }

/*****************************************************************************************
*****************************/

  public ListasVertices actualiza(int[][] m) {
    Vector aux = new Vector();
    ListasVertices vlsNueva = new ListasVertices();
    NodoLista vlNueva = new NodoLista();

    Vector vAux=new Vector();
    /**Para actualizar la vls los pasos son los siguientes:
     * -crear los nodoVertice
     * -añadirlos a un vector
     * -añadir el vector como VL a la VLS
     * */
    Vector casillasOcupadas = ops.sacaCasillasOcupadas(m);
    aux = ops.saca4VerticesDeOcupadas(casillasOcupadas, m);
    for (int j=0;j<aux.size();j++){
        NodoVertice verticeAux=(NodoVertice)aux.get(j);
        int grados=ops.dameGradosVertice(verticeAux, m);
        if (grados!=-1){
                if (ops.esta(vAux, verticeAux)){
                        if (seNecesitaRepetido(verticeAux,m,grados)){
                                vAux.add(verticeAux);
                                //Meto el repetido en la estructura de operacionesVLS
                                Vector elemento=new Vector();
                                elemento.add(verticeAux);
                                Vector repetidosAux=ops.getVerticesRepetidos();
                                repetidosAux.add(elemento);
                                ops.setVerticesRepetidos(repetidosAux);
                        }
                }
                else
                        vAux.add(verticeAux);
        }
    }

    if (vAux.size()>0){
            vlNueva.setVL(vAux);
            Vector vlOrdenadas=ordenaVL(vlNueva,m);
            NodoLista vlNuevaOrdenada;
            for (int i=0;i<vlOrdenadas.size();i++){
                vlNuevaOrdenada=(NodoLista)vlOrdenadas.get(i);
                vlsNueva.addVL(vlNuevaOrdenada);
            }
    }
    else{
        // No hay ningun vertice => no hay ningun hueco => FPGA llena => VLS es vacia
        vlsNueva = new ListasVertices();
    }
    return vlsNueva;
  }

  public ListasVertices actualizaConIslas(int[][] m,Vector tareasIslas) {
        Vector aux = new Vector();
        ListasVertices vlsNueva = new ListasVertices();
        NodoLista vlNueva = new NodoLista();
        /**Para actualizar la vls los pasos son los siguientes:
         * -crear los nodoVertice
         * -añadirlos a un vector
         * -añadir el vector como VL a la VLS
         * */

        Vector casillasOcupadas = ops.sacaCasillasOcupadas(m);
```

```
        aux = ops.saca4VerticesDeOcupadas(casillasOcupadas, m);

        Vector vAux=new Vector ();
        ListasVertices vlsNuevaAux = new ListasVertices();
        for (int j=0;j<aux.size();j++){
                NodoVertice verticeAux=(NodoVertice)aux.get(j);
                int grados=ops.dameGradosVertice(verticeAux, m);
                if (grados!=-1){
                        if (ops.esta(vAux, verticeAux)){
                        }
                        else
                                vAux.add(verticeAux);
                }
        }

        vlNueva.setVL(vAux);
        Vector vlOrdenadas=ordenaVL(vlNueva,m);
        NodoLista vlNuevaOrdenada;
        for (int i=0;i<vlOrdenadas.size();i++){
                vlNuevaOrdenada=(NodoLista)vlOrdenadas.get(i);
                vlsNuevaAux.addVL(vlNuevaOrdenada);
        }

        if (vlsNuevaAux.getTam()>=2){
                //Si la tarea que estamos manejando es una isla
                vlsNuevaAux=construyeVerticesVirtuales(vlsNuevaAux,tareasIslas,m);
            NodoLista vl=new NodoLista();
            vl.setVL(vlsNuevaAux.getVLS());
            Vector vOrdenado=ordenaVLIsla(vl,m);

            for (int j=0;j<vOrdenado.size();j++){
                vlNuevaOrdenada=(NodoLista)vOrdenado.get(j);
                vlsNueva.addVL(vlNuevaOrdenada);
            }
        }
        else{//es una tarea que da lugar a varios huecos
                System.err.println("Es una tarea que da lugar a dos huecos");

        }
        return vlsNueva;
    }

    public ListasVertices actualizaVLS(ListaTareas listaTareas, int [][] m) {

            ListasVertices vlsNueva = new ListasVertices();
            /*Sino hay ninguna tarea ejecutandose*/
            if (listaTareas.getTamanyo()==0){
                vlsNueva = ops.primeraVLS();
            }
            /*Si hay al menos una tarea ejecutandose en la FPGA*/
            else{

                 Vector tareasIslas=ops.queTareasSonIslas(m);
                 if (ops.hayIsla(m)){
                        System.err.println("HAY UNA ISLA");
                        vlsNueva=actualizaConIslas(m,tareasIslas);
                 }
                 else{
                        vlsNueva=actualiza(m);
                 }
            }
            return vlsNueva;
          }

/*****************************************************************************************
**************************/

    private Vector ordenaVL(NodoLista vl,int [][]matriz){
        Vector v = new Vector();

        NodoLista vl1 = new NodoLista();

        NodoVertice v1 = ops.buscaVerticeInicio(vl);
        vl1.addNodoAlFinal(v1);
        vl.removeNodo(v1.getFila(), v1.getColumna());
```

```
        NodoVertice v2 = ops.buscaV2(v1, vl.getVL(), matriz);
        vl1.addNodoAlFinal(v2);
        vl.removeNodo(v2.getFila(), v2.getColumna());

        NodoVertice v3 = ops.buscaV3(v2, vl.getVL(), matriz);
        vl1.addNodoAlFinal(v3);
        vl.removeNodo(v3.getFila(), v3.getColumna());

        NodoVertice v4 = buscaV4(v3, vl.getVL(), matriz,false);
        vl1.addNodoAlFinal(v4);
        vl.removeNodo(v4.getFila(), v4.getColumna());

        if (vl.getTam() == 0){ // El tamaño de la VL es de 4, que es el minimo
                v.add(vl1);
        }
        else{
                boolean vl1Insertada = false;
                NodoVertice v4Referencia = v4;

                while (vl.getTam() > 0) {
                        NodoVertice v5 = buscaV5
(v4Referencia,vl.getVL(),matriz,false,vl1.getVL());
                        if (((v5.getFila()==-1)||(v5.getColumna()==-1))&&(vl.getTam()>0)){
                                // Hay que dividir la VLS en varios huecos, o varias VL
                                v.add(vl1);
                                vl1Insertada = true;

                                Vector vl2 = ordenaVL(vl,matriz);
                                NodoLista vlAux = new NodoLista();
                                for (int i=0;i<vl2.size();i++){
                                        vlAux = (NodoLista)vl2.get(i);
                                        v.add(vlAux);
                                }
                        }
                        else{
                                // Tal vez habria que distinguir el caso de que pueda haber
errores
                                vl1.addNodoAlFinal(v5);
                                vl.removeNodo(v5.getFila(), v5.getColumna());

                                NodoVertice v6 = buscaV6(v5,vl.getVL(),matriz,vl1.getVL());
                                vl1.addNodoAlFinal(v6);
                                vl.removeNodo(v6.getFila(), v6.getColumna());

                                v4Referencia = v6;
                        }
                }

                if (!vl1Insertada){
                        v.add(vl1);
                }
        }

        return v;
  }

  private Vector ordenaVLIsla(NodoLista vl, int [][] matriz) {
           /*Tengo que ordenar la vl de una isla de forma distinta, siendo que va a
haber mas casos
                y distintos, a la hora de sacar izquierda y arriba*/

              Vector v=new Vector();

            NodoLista vlOrdenada = new NodoLista();
            Vector vOrdenado = new Vector();

            /**Busco el menor vertice, y le meto en la lista ordenada*/
            NodoVertice vMenor = ops.buscaVerticeInicio(vl);
            vl.removeNodo(vMenor.getFila(), vMenor.getColumna());
            vOrdenado.add(vMenor);

            /**Busco el v2*/
            //Busco un vertice que tenga el mismo numero de fila
            NodoVertice v2 = ops.buscaV2(vMenor, vl.getVL(), matriz);
        NodoVertice v3=new NodoVertice();
        NodoVertice v4=new NodoVertice();
            //Lo meto en la lista
```

```
            /*vl.removeNodo(v2.getFila(), v2.getColumna());
            vOrdenado.add((NodoVertice)v2);*/

            NodoVertice vReferencia=v2;
            boolean par = false;
            while(vl.getTam()>0){
                if (vReferencia.getFila()!=-1){
                                vOrdenado.add(vReferencia);

       vl.removeNodo(vReferencia.getFila(),vReferencia.getColumna());
                        if (vReferencia.getEsVirtual()){

                                vOrdenado=completaVLIsla(vOrdenado,vReferencia,vl,matriz);
                                vReferencia=(NodoVertice)vOrdenado.get(vOrdenado.size()-1);
                                //Elimino las dos apariciones de los vertices virtuales
                                vl.removeNodo(((NodoVertice)vOrdenado.get(vOrdenado.size()-
1)).getFila(), ((NodoVertice)vOrdenado.get(vOrdenado.size()-1)).getColumna());
                                vl.removeNodo(((NodoVertice)vOrdenado.get(vOrdenado.size()-
1)).getFila(), ((NodoVertice)vOrdenado.get(vOrdenado.size()-1)).getColumna());
                                vl.removeNodo(((NodoVertice)vOrdenado.get(vOrdenado.size()-
2)).getFila(), ((NodoVertice)vOrdenado.get(vOrdenado.size()-2)).getColumna());
                                vl.removeNodo(((NodoVertice)vOrdenado.get(vOrdenado.size()-
2)).getFila(), ((NodoVertice)vOrdenado.get(vOrdenado.size()-2)).getColumna());
                                //Busco el vertice de vuelta de la isla
                                if (vl.getTam()>0){
                                        // Para controlar la posibilidad de que el ultimo
vertice de la VL sea virtual
                                        NodoVertice
verticeVuelta=sacaVerticeVuelta((NodoVertice)vOrdenado.get(vOrdenado.size()-
2),vReferencia,vl);
                                        vReferencia=verticeVuelta;
                                }
                        }
                        else{
                                if (vReferencia.getEsVirtual()){
                                        if (((NodoVertice)vOrdenado.get(vOrdenado.size()-
2)).getEsVirtual()){

                                        }
                                        else{
                                                par=!par;
                                        }
                                }
                                if (par){
                                        v4 =buscaV4(vReferencia, vl.getVL(), matriz,
false);
                                        vReferencia=v4;
                                }
                                else{
                                        v3=buscaV5(vReferencia, vl.getVL(), matriz,
false,vOrdenado);
                                        vReferencia=v3;
                                }

                                par=!par;
                        }
                }
                else{
                                //La busqueda no ha sido fructifera y hago una llamada
recursiva
                                //para la busqueda en el resto de la lista de elementos
                            vlOrdenada.setVL(vOrdenado);
                                v.add(vlOrdenada);
                                Vector vector=ordenaVLIsla(vl,matriz);
                                for (int i=0;i<vector.size();i++){
                                        v.add((NodoLista)vector.get(i));
                                }
                                return v;
                }
            }

            vlOrdenada.setVL(vOrdenado);
                v.add(vlOrdenada);
            return v;
          }

  public NodoVertice sacaVerticeVuelta(NodoVertice v0,NodoVertice v1,NodoLista vl){
```

```
          NodoVertice verticeVuelta=new NodoVertice();
          int sentido=dameNumSentido(v0,v1);
        NodoVertice vertAux=new NodoVertice();
        switch (sentido){
        case 1:
                verticeVuelta=ops.buscaCualquierVerticeMismaFilaColumnaMayor(v1,
vl.getVL());
                for (int i=0;i<vl.getTam();i++){
                        vertAux=vl.getVertice(i);
                        if
((vertAux.getFila()==v1.getFila())&&(vertAux.getColumna()<verticeVuelta.getColumna())&&(
vertAux.getColumna()>v1.getColumna())){
                                verticeVuelta=vertAux;
                        }
                }
                break;
        case 2:
                verticeVuelta=ops.buscaCualquierVerticeMismaColumnaFilaMayor(v1,
vl.getVL());
                for (int i=0;i<vl.getTam();i++){
                        vertAux=vl.getVertice(i);
                        if
((vertAux.getColumna()==v1.getColumna())&&(vertAux.getFila()<verticeVuelta.getFila())&&(
vertAux.getFila()>v1.getFila())){
                                verticeVuelta=vertAux;
                        }
                }
                break;
        case 3:
                verticeVuelta=ops.buscaCualquierVerticeMismaFilaColumnaMenor(v1,
vl.getVL());
                for (int i=0;i<vl.getTam();i++){
                        vertAux=vl.getVertice(i);
                        if
((vertAux.getFila()==v1.getFila())&&(vertAux.getColumna()>verticeVuelta.getColumna())&&(
vertAux.getColumna()<v1.getColumna())){
                                verticeVuelta=vertAux;
                        }
                }
                break;
        case 4:
                verticeVuelta=ops.buscaCualquierVerticeMismaColumnaFilaMenor(v1,
vl.getVL());
                for (int i=0;i<vl.getTam();i++){
                        vertAux=vl.getVertice(i);
                        if
((vertAux.getColumna()==v1.getColumna())&&(vertAux.getFila()>verticeVuelta.getFila())&&(
vertAux.getFila()<v1.getFila())){
                                verticeVuelta=vertAux;
                        }
                }
                break;
        }
        return verticeVuelta;
  }

  public Vector completaVLIsla(Vector vOrdenado,NodoVertice verticeVirtual,NodoLista
vl,int [][] m){
          Vector v=new Vector();
          for (int n=0;n<vOrdenado.size();n++){
                v.add((NodoVertice)vOrdenado.get(n));
          }
          //Busco el otro vertice virtual y lo inserto en la lista ordenada
          NodoVertice verticeVirtual2=new NodoVertice();
          boolean encontrado=false;
          int i=0;
          NodoVertice verticeAux;

          boolean vertical=false;
          boolean horizontal=false;

          while((!encontrado)&&(i<vl.getTam())){
                verticeAux=(NodoVertice)vl.getVertice(i);
                if (verticeAux.getEsVirtual()){
                        if
((verticeVirtual.getFila()==0)||(verticeVirtual.getFila()==filasFPGA)){
                                vertical=true;
```

```
                        }
                        else if
((verticeVirtual.getColumna()==0)||(verticeVirtual.getColumna()==columnasFPGA)){
                                horizontal=true;
                        }

                        if
((vertical)&&((verticeAux.getFila()!=verticeVirtual.getFila())&&(verticeAux.getColumna()
==verticeVirtual.getColumna()))){
                                encontrado=true;
                                verticeVirtual2=(NodoVertice)verticeAux;
                        }
                        else if
((horizontal)&&((verticeAux.getColumna()!=verticeVirtual.getColumna())&&(verticeAux.getF
ila()==verticeVirtual.getFila()))){
                                encontrado=true;
                                verticeVirtual2=(NodoVertice)verticeAux;

                        }
                        else{
                                i++;
                        }
                }
                else{
                        i++;
                }
        }
        if (!encontrado){
                i=0;
                System.err.println("completaVLIsla. buscando el segundo virtual,
partiendo de F:"+verticeVirtual.getFila()+" C:"+verticeVirtual.getColumna());
                while((!encontrado)&&(i<vl.getTam())){
                        verticeAux=(NodoVertice)vl.getVertice(i);
                        if (verticeAux.getEsVirtual()){
                                if
(((verticeAux.getFila()!=verticeVirtual.getFila())&&(verticeAux.getColumna()==verticeVir
tual.getColumna()))||

((verticeAux.getColumna()!=verticeVirtual.getColumna())&&(verticeAux.getFila()==verticeV
irtual.getFila()))){
                                        encontrado=true;
                                        verticeVirtual2=(NodoVertice)verticeAux;

                                }
                                else{
                                        i++;
                                }
                        }
                        else{
                                i++;
                        }
                }
        }

        NodoVertice verticeVirtual2Aux=new
NodoVertice(verticeVirtual2.getFila(),verticeVirtual2.getColumna());
        v.add(verticeVirtual2Aux);

        /**Busco el segundo vertice de la isla*/
        boolean arriba=ops.arribaAbajo(verticeVirtual2, m,v);
        NodoVertice segundoIsla=new NodoVertice();

        if (arriba)
segundoIsla=ops.buscaCualquierVerticeMismaColumnaFilaMayor(verticeVirtual2, vl.getVL());
        else
segundoIsla=ops.buscaCualquierVerticeMismaColumnaFilaMenor(verticeVirtual2, vl.getVL());
        //Hallamos el vertice que buscamos
        for (int j=0;j<vl.getTam();j++){
                verticeAux=(NodoVertice)vl.getVertice(j);
                if ((verticeAux.getFila()!=segundoIsla.getFila())&&
                                (verticeAux.getColumna()==segundoIsla.getColumna())&&
                                (!verticeAux.getEsVirtual())&&
                                //Para asegurarme de que sea distinto del virtual
                                (verticeAux.getFila()!=verticeVirtual2.getFila())){
                        if
((arriba)&&(verticeAux.getFila()<segundoIsla.getFila())&&(verticeAux.getFila()>verticeVi
rtual2.getFila())){
```

```
                                segundoIsla=(NodoVertice)verticeAux;
                        }
                        else{
                                if
((!arriba)&&(verticeAux.getFila()>segundoIsla.getFila())&&(verticeAux.getFila()<verticeV
irtual2.getFila())){
                                        segundoIsla=(NodoVertice)verticeAux;
                                }
                        }
                }
        }
        //Lo añado a la lista ordenada y lo quito de la vl
        v.add(segundoIsla);
        vl.removeNodo(segundoIsla.getFila(),segundoIsla.getColumna());

        /**Busco el tercer vertice de la isla*/
        NodoVertice tercerIsla=new NodoVertice();
        boolean izquierda=ops.izquierdaDerecha2(segundoIsla, m);

        if (izquierda)
tercerIsla=ops.buscaCualquierVerticeMismaFilaColumnaMayor(segundoIsla, vl.getVL());
        else tercerIsla=ops.buscaCualquierVerticeMismaFilaColumnaMenor(segundoIsla,
vl.getVL());
        //Hallamos el vertice que buscamos
        for (int j=0;j<vl.getTam();j++){
                verticeAux=(NodoVertice)vl.getVertice(j);
                if ((verticeAux.getFila()==tercerIsla.getFila())&&
                                (verticeAux.getColumna()!=tercerIsla.getColumna())&&
                                (!verticeAux.getEsVirtual())){
                        if
((izquierda)&&(verticeAux.getColumna()<tercerIsla.getColumna())&&(verticeAux.getColumna(
)>segundoIsla.getColumna())){
                                tercerIsla=(NodoVertice)verticeAux;
                        }
                        else{
                                if
((!izquierda)&&(verticeAux.getColumna()>tercerIsla.getColumna())&&(verticeAux.getColumna
()<segundoIsla.getColumna())){
                                        tercerIsla=(NodoVertice)verticeAux;
                                }
                        }
                }
        }
        //Lo añado a la lista ordenada y lo quito de la vl
        v.add(tercerIsla);
        vl.removeNodo(tercerIsla.getFila(),tercerIsla.getColumna());

        /**Busco el cuarto vertice de la isla*/
        NodoVertice cuartoIsla=new NodoVertice();
        arriba=ops.arribaAbajo(tercerIsla, m,v);
        if (arriba)
cuartoIsla=ops.buscaCualquierVerticeMismaColumnaFilaMayor(tercerIsla, vl.getVL());
        else cuartoIsla=ops.buscaCualquierVerticeMismaColumnaFilaMenor(tercerIsla,
vl.getVL());
        //Hallamos el vertice que buscamos
        for (int j=0;j<vl.getTam();j++){
                verticeAux=(NodoVertice)vl.getVertice(j);
                if (verticeAux.getColumna()==58){
                        System.err.println("fila:"+
verticeAux.getFila()+"columna"+verticeAux.getColumna());
                }
                if ((verticeAux.getFila()!=cuartoIsla.getFila())&&
                                (verticeAux.getColumna()==cuartoIsla.getColumna())&&
                                (!verticeAux.getEsVirtual())){
                        if
((arriba)&&(verticeAux.getFila()<cuartoIsla.getFila())&&(verticeAux.getFila()>tercerIsla
.getFila())){
                                cuartoIsla=(NodoVertice)verticeAux;
                        }
                        else{
                                if
((!arriba)&&(verticeAux.getFila()>cuartoIsla.getFila())&&(verticeAux.getFila()<tercerIsl
a.getFila())){
                                        cuartoIsla=(NodoVertice)verticeAux;
                                }
                        }
                }
```

```
		}
		//Lo añado a la lista ordenada y lo quito de la vl
		v.add(cuartoIsla);
		vl.removeNodo(cuartoIsla.getFila(),cuartoIsla.getColumna());

		NodoVertice vRef=cuartoIsla;
		NodoVertice v4=new NodoVertice();
		NodoVertice v5=new NodoVertice();
		boolean fin=false;
		while((!fin)&&(vl.getTam()>0)){
			v4=this.buscaV4(vRef, vl.getVL(), m,false);
			if
((v4.getFila()==verticeVirtual2.getFila())&&(v4.getColumna()==verticeVirtual2.getColumna
())){
				fin=true;
			}
			else{
				v.add(v4);
				vl.removeNodo(v4.getFila(), v4.getColumna());
				v5=buscaV5(v4,vl.getVL(),m,false,v);
				if ((v5.getFila()==-1)&&(v5.getColumna()==-1)){
					fin=true;
				}
				else{
					v.add(v5);
					vl.removeNodo(v5.getFila(), v5.getColumna());
					vRef=v5;
				}
			}
		}

		/**Ahora añado los vertices virtuales de nuevo*/
		v.add(verticeVirtual2);
		vl.removeNodo(verticeVirtual2.getFila(), verticeVirtual2.getFila());
		vl.removeNodo(verticeVirtual2.getFila(), verticeVirtual2.getFila());
		v.add(verticeVirtual);
		vl.removeNodo(verticeVirtual.getFila(), verticeVirtual.getFila());

		return v;
	}

	private ListasVertices construyeVerticesVirtuales(ListasVertices vls,Vector
tareasIslas,int [][]m){
		ListasVertices vlsNueva=new ListasVertices();
		Vector vlUnion=new Vector();
		//La primera VL no representa una isla
		NodoLista vlBase=vls.getVL(0);
		NodoLista vlIsla;
		NodoVertice vertIsla=new NodoVertice();
		Vector posiblesVerticesVirtuales=new Vector();
		Vector verticesVirtuales=new Vector();
		Vector ternasVirtuales=new Vector();

		for (int i=1;i<vls.getTam();i++){
			/* Hay que inicializar para que si hay varias islas, solo se tengan en
cuenta
			 * los posibles candidatos a vertices virtuales de esa isla
			 */
			posiblesVerticesVirtuales=new Vector();
			//Tomo una vl que representa una isla
			vlIsla=vls.getVL(i);
			//Recorro los vertices de la isla
			for (int j=0;j<vlIsla.getTam();j++){
				vertIsla=vlIsla.getVertice(j);
				vlUnion.add((NodoVertice)vertIsla);
				//Asocio al vertice de la isla,el vertice de la vlBase
				//que este a menor distancia, y la informacion de la
distancia(distancia,filas,columnas)
				Vector vAux=asociaVerticeIsla(vertIsla,vlBase,m);
				//Añado el vertice de la isla, el de la vlBase
				//y la distancia(que sean menores)
				posiblesVerticesVirtuales.add(vAux);

			}

ternasVirtuales.add((Vector)buscaTernaVirtualMejor(posiblesVerticesVirtuales,filasFPGA,c
olumnasFPGA));
```

```
        }

        //Ya tengo una terna en la que se me dice el vertice base,
        //la informacion distancia(distancia,filas,columnas)
        //y el virtual de la isla
        Vector ternaAux=new Vector();
        for (int n=0;n<ternasVirtuales.size();n++){
                ternaAux=(Vector)ternasVirtuales.get(n);
                verticesVirtuales.add((Vector)calculaVirtuales(ternaAux));
        }

        //Meto los vertice de la vl base
        for(int k=0;k<vlBase.getTam();k++){
                vlUnion.add((NodoVertice)vlBase.getVertice(k));
        }
        //Meto los vertice virtuales
        Vector vAux=new Vector();
        for (int l=0;l<verticesVirtuales.size();l++){
                vAux=(Vector)verticesVirtuales.get(l);
                for(int n=0;n<vAux.size();n++){
                        //Meto el vertice virtual de la isla dos veces,
                        //pero antes elimino la aparicion que habia; y asi lo marcare
como virtual
                        NodoVertice v1=(NodoVertice)vAux.get(n);
                        vlUnion.remove(v1);
                        v1.setEsVirtual(true);
                        vlUnion.add(v1);
                        vlUnion.add(v1);
                        //Meto el vertice virtual de la vlBase 2 veces
                        NodoVertice v2=(NodoVertice)vAux.get(n+1);
                        vlUnion.remove(v2);
                        v2.setEsVirtual(true);
                        vlUnion.add(v2);
                        vlUnion.add(v2);
                        n++;
                }
        }
        vlsNueva.setVLS(vlUnion);

        return vlsNueva;
  }

  public NodoVertice buscaV4(NodoVertice v3, Vector v, int [][] m,boolean
tratandoVirtual) {
            NodoVertice v4 = new NodoVertice();
            NodoVertice vAux = new NodoVertice();
            NodoVertice vReferencia = new NodoVertice();

            NodoLista vl=new NodoLista();
            vl.setVL(v);
            boolean izquierda = ops.izquierdaDerecha2(v3, m);
            if (tratandoVirtual){
               izquierda=!izquierda;
            }

            if (izquierda) {
              vReferencia = ops.buscaCualquierVerticeMismaFilaColumnaMayor(v3, v);
            }
            else {
              vReferencia = ops.buscaCualquierVerticeMismaFilaColumnaMenor(v3, v);
            }
            for (int i = 0; i < v.size(); i++) {
              vAux = (NodoVertice) v.elementAt(i);
              if (vAux.getFila() == v3.getFila()) {
                if (izquierda) {
                  if ( (vAux.getColumna() <= vReferencia.getColumna()) &&
                      (vAux.getColumna() > v3.getColumna())) {
                    vReferencia = vAux;
                  }
                }
                else {
                  if ( (vAux.getColumna() >= vReferencia.getColumna()) &&
                      (vAux.getColumna() < v3.getColumna())) {
                    vReferencia = vAux;
                  }
                }
```

```
            }
          }
          v4 = vReferencia;
          return v4;
  }

  public NodoVertice buscaV5(NodoVertice v4, Vector v, int [][] m,boolean
tratandoVirtual,Vector vectorComprobacion) {
          NodoVertice v5 = new NodoVertice();
          NodoVertice vAux = new NodoVertice();
          NodoVertice vReferencia = new NodoVertice();

          boolean arriba = ops.arribaAbajo(v4, m,vectorComprobacion);
              if (ops.estaRepetido(v4)){
                      if (ops.estaAsignadoRepetidos(v4)){
                              arriba=!(ops.devuelveAsignacionRepetidos(v4));
                      }
                      else{
                              ops.asignarRepetidos(v4,arriba);
                      }
              }
          if (tratandoVirtual){
              arriba=!arriba;
          }

          if (arriba) {
            vReferencia = ops.buscaCualquierVerticeMismaColumnaFilaMayor(v4, v);
          }
          else {
            vReferencia = ops.buscaCualquierVerticeMismaColumnaFilaMenor(v4, v);
          }
          for (int i = 0; i < v.size(); i++) {
            //System.out.println("");
            vAux = (NodoVertice) v.elementAt(i);
            if (vAux.getColumna() == v4.getColumna()) {
              if (arriba) {
                if ( (vAux.getFila() <= vReferencia.getFila()) &&
                    (vAux.getFila() > v4.getFila())) {
                  vReferencia = vAux;
                }
              }
              else {
                if ( (vAux.getFila() >= vReferencia.getFila()) &&
                    (vAux.getFila() < v4.getFila())) {
                  vReferencia = vAux;
                }
              }
            }
          }
          v5 = vReferencia;
          return v5;
  }

  public NodoVertice buscaV6(NodoVertice v5, Vector v, int [][] m,Vector
vectorComprobacion) {
          NodoVertice v6 = new NodoVertice();
          v6 = buscaV4(v5, v, m,false);
          if ((v6.getFila()==-1)&&(v6.getColumna()==-1)){
              v6=buscaV5(v5,v,m,true,vectorComprobacion);
          }
          return v6;
  }

}
```

VertexSelector.java

Se encarga de implementar el modulo de Selector de Vértices.

```
package proyecto;
/**
 * @author Raquel Sanchez
 * */
import comun.listaVertices.*;
import comun.listaTareasEjecucion.ParametrosTarea;
```

```
import proyecto.algoritmos.algoritmos2D.*;
import proyecto.algoritmos.algoritmos3D.*;

public class VertexSelector {

	private int filasFPGA;
	private int columnasFPGA;
	private OperacionesVLS ops;

	public VertexSelector (int filasFPGA, int columnasFPGA){
		this.filasFPGA = filasFPGA;
		this.columnasFPGA = columnasFPGA;
		ops = new OperacionesVLS(filasFPGA,columnasFPGA);
	}

  public NodoVertice selectorVertice(MatrizFPGA m, String heuristica,ListasVertices vls,
ParametrosTarea tarea,PlanificadorTareas planificador) {
    NodoVertice vSeleccionado = new NodoVertice();
    if ( (heuristica == "FF") || (heuristica == "fragmentacion") ||(heuristica ==
"adyacencia2D")) {
      vSeleccionado = heuristica2D(m, heuristica, vls, tarea,planificador);
    }
    else {
      if ( (heuristica == "adyacencia3D") || (heuristica == "look_ahead")) {
        vSeleccionado = heuristica3D(m,heuristica, vls, tarea,planificador);
      }
    }
    return vSeleccionado;
  }

  public NodoVertice heuristica2D(MatrizFPGA m, String heuristica,ListasVertices vls,
ParametrosTarea tarea,PlanificadorTareas planificador) {
    NodoVertice vSeleccionado = new NodoVertice();
    if (heuristica == "FF") {
      vSeleccionado = eleccionFF(m, vls, tarea);
    }
    else {
      if (heuristica == "fragmentacion") {
        vSeleccionado = eleccionFragmentacion(m, vls, tarea,planificador);
      }
      else {
        if (heuristica == "adyacencia2D") {
          vSeleccionado = eleccionAdyacencia2D(m, vls, tarea);
        }
      }
    }
    return vSeleccionado;
  }

  public NodoVertice heuristica3D(MatrizFPGA m,String heuristica, ListasVertices vls,
                                  ParametrosTarea tarea,PlanificadorTareas planificador)
{
    NodoVertice vSeleccionado = new NodoVertice();
    if (heuristica == "adyacencia3D") {
      vSeleccionado = eleccionAdyacencia3D(m,vls, tarea,planificador);
    }
    else {
      if (heuristica == "look_ahead") {
        vSeleccionado = eleccionLookAhead(m,vls, tarea,planificador);
      }
    }
    return vSeleccionado;
  }

  /* Se elige el vertice en funcion de la heuristica 2D First Fit,
   * Se elige el primer vertice que sirva
   */
  public NodoVertice eleccionFF(MatrizFPGA m, ListasVertices vls,ParametrosTarea tarea)
{
    NodoVertice vSeleccionado = new NodoVertice();
    NodoVertice vAux = new NodoVertice();
    NodoLista vl = new NodoLista();
    boolean encontrado = false;
    int cLista = 0;

    while ( (cLista < vls.getTam()) && (!encontrado)) {
      vl = vls.getVL(cLista);
```

```
      int posVertice = 0;
      while ( (!encontrado) && (posVertice < vl.getTam())) {
        vAux = vl.getVertice(posVertice);
        if (ops.esCandidatoEspacio(m.getMatriz(), vAux, vl, tarea)) {
          vSeleccionado = vAux;
          encontrado = true;
        }
        else {
          posVertice++;
        }
      }
      cLista++;
    }
    return vSeleccionado;
  }

  /* Se elige el vertice en funcion de una heuristica 2D Best Fit,
   * en este caso la heuristica esta basada en Fragmentacion
   *
   * Se estima la fragmentacion producida en el area libre de la FPGA
   * por cada uno de los vertices candidatos; y finalmente
   * elige el vertice donde se produzca menor nivel de fragmentacion
   */
  public NodoVertice eleccionFragmentacion(MatrizFPGA m,ListasVertices
vls,ParametrosTarea tarea,PlanificadorTareas planificador) {
    NodoVertice vSeleccionado = new NodoVertice();

    Fragmentacion frag=new Fragmentacion(filasFPGA, columnasFPGA);
    vSeleccionado=frag.algoritmoFragmentacion(m,vls,tarea,planificador);

    return vSeleccionado;
  }

  /* Se elige el vertice en funcion de una heuristica 2D Best Fit,
   * en este caso la heuristica esta basada en Adyacencia
   *
   * Se estima el nivel de adyacencia que habria en cada
   * uno de los candidatos
   * Elegimos el candidato que provoque mayor nivel de adyacencia
   */
  public NodoVertice eleccionAdyacencia2D(MatrizFPGA m, ListasVertices
vls,ParametrosTarea tarea) {
    NodoVertice vSeleccionado = new NodoVertice();
    Adyacencia2D ady2D=new Adyacencia2D(filasFPGA, columnasFPGA);
    vSeleccionado=ady2D.algoritmoAdyacencia2D(m,vls,tarea);
    return vSeleccionado;
  }

  /* Se elige el vertice en funcion de una heuristica 3D Best Fit,
   * en este caso la heuristica esta basada en Adyacencia
   *
   * Se estima el nivel de adyacencia que habria en cada
   * uno de los candidatos
   * Elegimos el candidato que provoque mayor nivel de adyacencia
   */
  public NodoVertice eleccionAdyacencia3D(MatrizFPGA m, ListasVertices
vls,ParametrosTarea tarea,PlanificadorTareas planificador) {
    NodoVertice vSeleccionado = new NodoVertice();
    Adyacencia3D ady3D=new Adyacencia3D(filasFPGA, columnasFPGA);
    vSeleccionado=ady3D.algoritmoAdyacencia3D(m,vls,tarea,planificador);
    return vSeleccionado;
  }
  /* Se elige el vertice en funcion de una heuristica 3D Best Fit,
   * en este caso la heuristica esta basada en Look Ahead
   *
   * Se usa el valor de la adyacencia 3D, se calcula para todos los posibles candidaros
   * pero ahora se calcula en el instante actual (t_curr) y en
   * el proximo evento (cuando el fin de la siguiente tarea ocurres)
   */
  public NodoVertice eleccionLookAhead(MatrizFPGA m,ListasVertices vls,ParametrosTarea
tarea,PlanificadorTareas planificador) {
    NodoVertice vSeleccionado = new NodoVertice();
    LookAhead look_ahead=new LookAhead(filasFPGA, columnasFPGA);
    vSeleccionado=look_ahead.algoritmoLookAhead(m,vls,tarea,planificador);
    return vSeleccionado;
  }
}
```