SISTEMAS ELECTRÓNICOS DIGITALES

MEMORIA FPGAs

APELLIDOS, NOMBRE:

González Cabeza, César (56398)

Aragoneses Hernández, Esther (56265)

De Antonio Sánchez, Enrique (56330)

GRUPO: A404 - Ingeniería Electrónica Industrial y Automática (UPM - ETSIDI)

PROFESOR: Giuseppe Conti

Contenido

Introducción.

En este trabajo se ha realizado una simulación de una Ascensor en consonancia con la oferta 6 de la gama media. Sin embargo, pese a que la premisa siga siendo la misma se han realizado ligeras modificaciones para aumentar la dificultad del código para obtener un trabajo más completo.

La primera parte de la propuesta se ha mantenido, obteniendo como salida dos bits del motor y un bit para la puerta, añadiendo distintas señales para la representación de información clave en el display (expandido en explicación de identidad LedGestor). La primera modificación se toma de la primera propuesta, en la cual en vez de mantener el ascensor abierto se toma un tiempo de referencia para que se detengan las puertas, una vez cumplido estas se vuelven a cerrar. Esta modificación viene ligada a la segunda que hemos introducido. En vez de poder atender únicamente a una llamada a la vez, el ascensor se aproxima más a un ascensor común. El ascensor registra todas las llamadas y las atiende en función de la proximidad y la dirección, como veremos más adelante. De este modo obtenemos un código más complejo que se aproxima más al funcionamiento corriente de un ascensor.

Cabe clarificar que en este trabajo hemos atendido más al ejercicio académico que a la aproximación real a un ascensor real. Un ejemplo se estos es el registro de las posiciones, que en vez de tomarse como una entrada de sensores se le dedica una entidad propia más aproximada a una máquina simple.

Esta memoria se organiza de modo que se estudian las entidades divididas en dos grupos, entidades síncronas y asíncronas. De este modo podemos observar que decisiones toman las entidades y como estas se transmiten al resto del programa. Una vez estudiadas todas las entidades procederemos a estudiar el programa en su conjunto y como estas se implementan.

Entidades asíncronas.

Gestor (Decididor del direccionamiento)

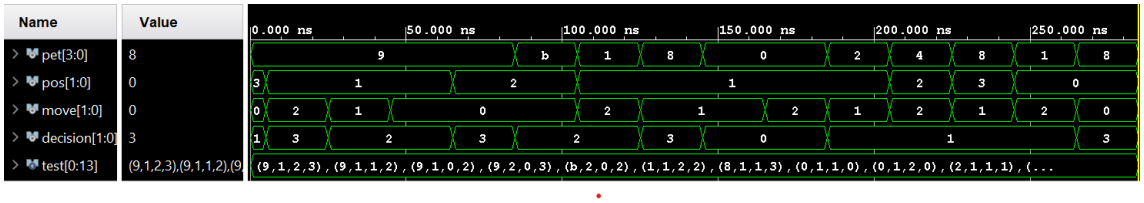
La entidad Gestor es la encargada de dictaminar hacia donde se tiene que mover el ascensor. Sin embargo, está no tiene ningún control sobre los motores o la puerta, solamente dice dónde podemos encontrar una petición. Esto es debido a que muchas veces a pesar de tener una petición arriba debemos realizar alguna acción para mantener el comportamiento adecuado del ascensor. Este comportamiento del ascensor se decide en la máquina simple.

Para poder decidir hacia dónde queremos movernos debemos observar el vector de peticiones. Este vector debe recorrerse de una manera concreta de modo que si

estamos yendo para arriba podamos recoger a pasajeros que aparezcan por el camino. Por tanto, dependiendo de que motores estén activados observaremos de una manera u otra. Si esta activado el motor de subida observaremos primero nuestro piso, después los pisos por encima nuestra y por último los pisos debajo nuestro, siempre por orden de proximidad. Si el motor de bajada esta activo observaremos primeros nuestro piso, luego los pisos debajo nuestra y por último los pisos encima nuestra. Si nos encontramos parados va alternando, primero mira abajo, luego arriba y así hasta observar todos los pisos.

Una vez encontrada una petición, se calcula donde se encuentra y dependiendo de su posición decidimos subir (“11”), bajar (“10”), abrir la puerta (“01”) o detenernos (“00”) si no encontramos ninguna petición.

Simulación



Como podemos observar la prueba se ha realizado correctamente. No se detecta ningún error. Estas decisiones son completamente asíncronas lo que deberemos tener en cuenta al realizar la maquina simple del ascensor.

LedDecoder (Decodificador de los displays)

LedDecoder es otro programa sencillo que utiliza una sintaxis dataflow. Su principal objetivo es dictaminar que valor se muestra en los displays dependiendo de la posición, de su movimiento y de la apertura de la puerta.

Para simular el ascensor se utilizan dos valores el 0 indicando que la puerta está abierta y el 8 indicando que la puerta está cerrada. Para indicar la posición mostraremos el número del piso y para mostrar el movimiento 7 si vamos hacia arriba, L si vamos hacia abajo y – si el ascensor está detenido. Estos valores se alimentar a la entidad LedGestor que se encargara de mostrar estos valores en la pantalla.

Entidades síncronas

Ahora vamos a estudiar todas las entidades síncronas. Estas serán las encargadas de determinar el comportamiento del ascensor en función de los resultados en las entidades síncronas y los impulsos a los que sometamos la placa.

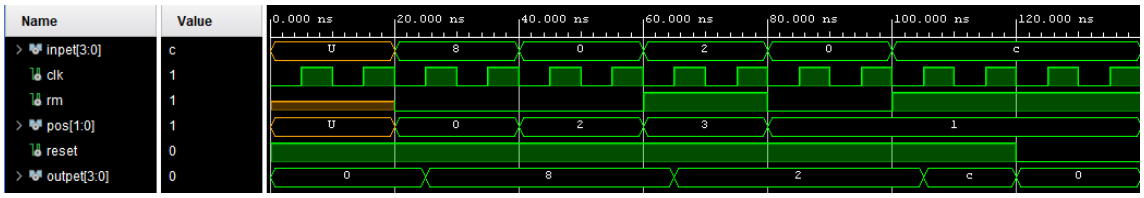
Petition (Registrador de peticiones)

La entidad Petition es la entidad a cargo de recibir las señales sincronizadas de cada uno de los pisos. Esta las guarda en un vector de dimensión 4 donde cada bit representa la petición a un piso a nivel alto. Esta entidad también recibe una señal para eliminar una petición que se conectará a la apertura de la puerta, así como la posición en la que se elimina que se conecta al registro de la posición. De este modo cada vez que se abra la puerta se eliminará la petición.

La entidad funciona con un proceso principal que cambia los valores de una señal intermedia según la entrada inpet (vector conectado a las señales sincronizadas de entrada), rm (señal de eliminación de una petición) y reset (señal de reinicio). Si recibe un '1' en algún bit del vector de entrada inscribe un '1' en el bit correspondiente a la señal intermedia. Por otro lado, cuando recibimos la señal rm inscribimos un '0' en el bit situado en la posición recibida. Por último, al recibir la señal de reinicio se inscribe 0 en todos los bits. Fuera del proceso asignamos a la señal de salida la señal intermedia.

Cabe destacar también la necesidad de una señal de reloj debido a su comportamiento de memoria. Esta necesidad a resultado clave para el correcto desarrollo del programa.

Simulación



Como podemos ver, todas las entradas y salidas se actualizan correctamente con los flancos positivos del reloj.

SYNCHRNZR (Sincronizador) y EDGEDTCTR (Detector de flancos)

Estas dos primeras entidades trabajan en bloque para sincronizar las entradas y detectar estas una vez sean desactivadas. El sincronizador estabiliza la señal y evita que se produzca la metaestabilidad. Por otro lado, el detector de flancos es el encargado de indicar un flanco negativo a su entrada. De este modo una vez soltemos el botón se parará un valor alto a la correspondiente entrada de la petición.

Stabledeci (Estabilizador de la dirección)

Stabledeci es una entidad sencilla que traduce las decisiones de la entidad gestor y se las devuelve como señales del motor para que pueda decidir en función de ellas. Esta entidad se creó para separar la salida del motor de las decisiones que

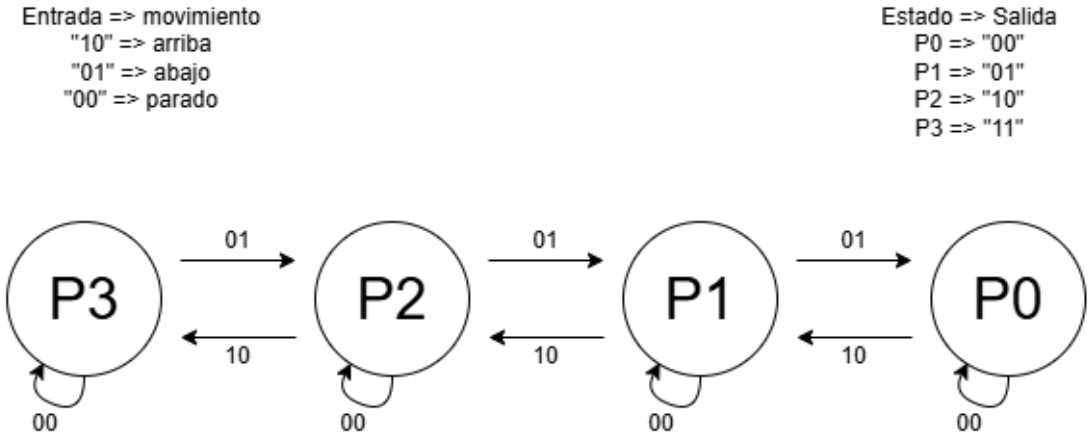
toma la entidad gestor. Como hemos mencionado antes, no siempre la decisión que tomamos resultará en la actuación directa del motor, sino que hay un protocolo a seguir mientras tanto. Este problema que soluciona se explica en detalle en la sección de problemas y mejoras.

Position (Registrador de la posición)

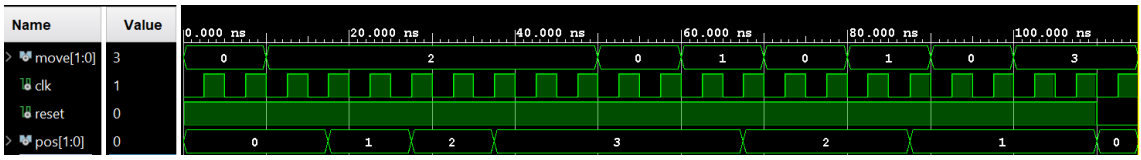
La entidad Position es la encargada de registrar la posición en la que se encuentra nuestro ascensor. Al prescindir de los sensores para este ejercicio académico incrementamos la dificultad del código cambiando la posición en función de un contador sincronizado con la señal de reloj. La entidad recibe un elemento generic que se configura para que coincida con la frecuencia del reloj (en nuestro caso 100MHz). La posición cambiará cada dos segundos dependiendo de que motor este encendido.

Para llevar a cabo esta tarea hemos diseñado una máquina simple que se actualiza cada vez que el contador llega a la frecuencia de reloj estimada. Por tanto, tenemos tres procesos. El primero es para generar la señal del temporizador, de modo que creamos una nueva señal de reloj con un periodo de dos segundos. Los otros tres procesos corresponden al desarrollo normal de una máquina de estados simple. Por un lado, un registrador de estados que estará sincronizado con el temporizador, el decodificador de estados que calcula el siguiente estado y el decodificador de salidas que da valor a la posición en función del estado.

Diagrama de estados



Simulación



En esta simulación hemos indicado que la frecuencia de reloj es uno, por lo tanto, por cada dos ciclos de reloj se actualiza la posición. Podemos observar como el comportamiento de la máquina es el adecuado.

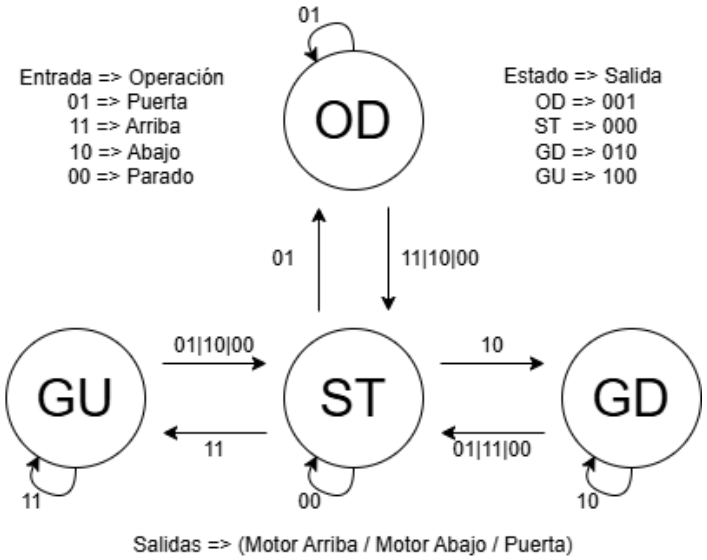
FMS (Maquina simple del ascensor)

La máquina simple del ascesor recibe las decisiones emitidas en el gestor y modifica las señales de los motores y la puerta en función de ellas. Su funcionamiento se basa en una maquina simple corriente con una ligera modificación. Es ella se ha añadido un proceso que activa una espera siempre que el ascensor se detenga o abra una puerta. En el apartado problemas y mejoras estudiamos más a fondo este proceso y porque se necesita.

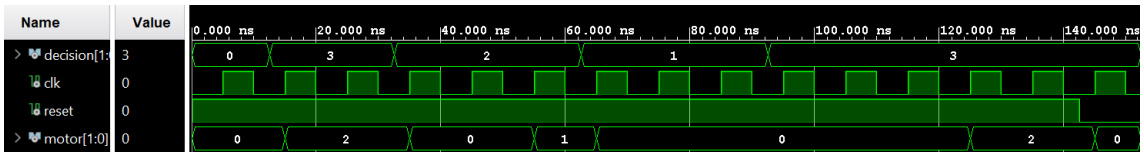
Por tanto, esta entidad funciona con los siguientes procesos. En primera instancia un proceso síncrono que, si existe una espera, realiza una cuenta que al acabar actualiza el estado y si no existe dicha espera actualiza directamente. Después se definen los decodificadores de estados y salidas. Por último, se define un proceso que, si están activos los estados de parada o de puerta abierta, activa la espera.

La cuenta que se realiza dependerá de los segundos que nos queramos detener multiplicados por la frecuencia del reloj. Por defecto nos detendremos durante 3 s, 2 segundos parados y entre medias un segundo para la apertura de la puerta.

Diagrama de estados



Simulación.

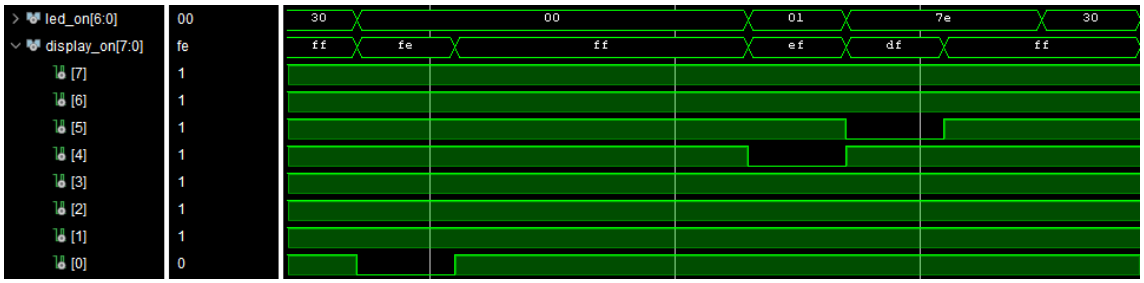


LedGestor (Gestor de los displays)

LedGestor es la entidad final de nuestro trabajo. Su trabajo consiste en recibir los códigos 7 segmentos de nuestro programa y mostrarlos en los displays de la placa. Esto lo hace multiplexando dichos displays y mediante una frecuencia determinada (en este caso 1kHz) va pasando los valores de siete segmentos a cada uno. La entidad recibe la frecuencia del reloj y la convierte en una señal de reloj de 1kHz (Si la frecuencia es menor de 1kHz se debe dar un valor de 200 kHz para que la entidad funcione correctamente).

En el primer proceso se realiza este ajuste. Una vez se completa la cuenta se incrementa una señal count global de la arquitectura que indica a que display le pasamos la información. En el segundo proceso utilizamos la señal de posición para determinar que displays van a estar encendidos en la representación final. Los primeros cuatro display representan el ascensor con su apertura y cierre de puerta. Los siguientes dos muestran primero la posición con número y en la segunda posición el movimiento. La 7 posición siempre se mantiene apagada y la octava se enciende en caso de error de posición mostrando una 'E'. En el último proceso se inicia el vector de salida para los displays a uno y se asigna en el número de cuenta el valor respectivo de la señal donde indicamos que displays están encendidos. Seguidamente dependiendo del valor de la cuenta, mostramos un valor de 7 segmentos u otro.

Simulación.

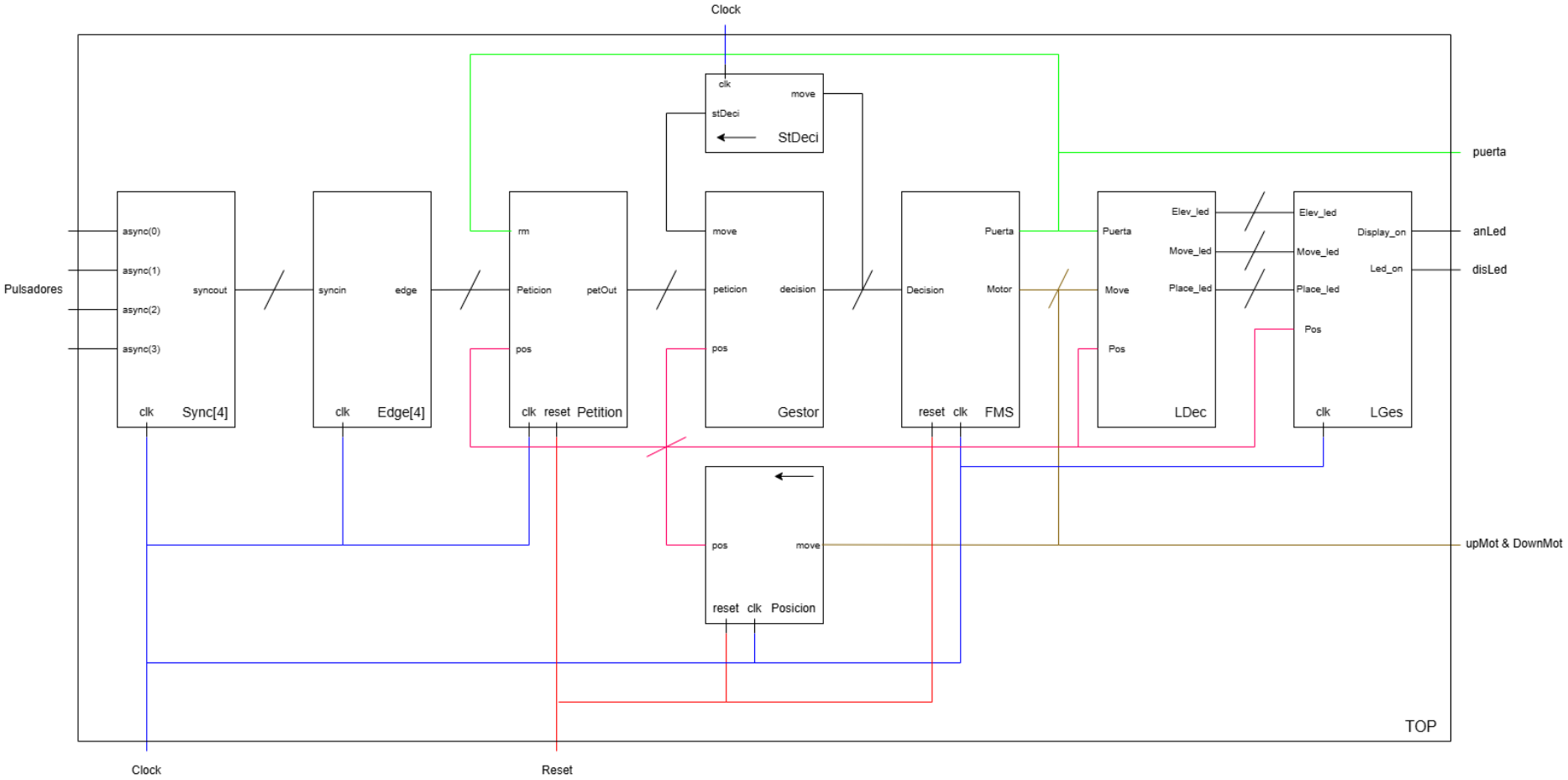


Como podemos observar el display se activa periódicamente dependiendo del valor de la posición y el movimiento.

TOP (Entidad final)

La entidad todo se define mediante una sintaxis estructural y agrupa todas las entidades que hemos creado. En la siguiente página se expone su estructura. Las flechas simbolizan donde se encuentran las salidas.

Diagrama de bloques



Problemas y mejoras

Problemas.

En este proyecto no hemos encontrado con bastantes problemas por el camino que hemos ido solucionando. Entre estos se encuentran distintos errores de diseño y conceptos que hemos sabido identificar y resolver, como la necesidad de una frecuencia inferior en la entidad LedGestor o la necesidad de una señal síncrona en la entidad Petition por su condición de memoria. Sin embargo, en esta memoria nos gustaría resaltar tres problemas principales.

Registro de la posición.

El problema más interesante al que nos hemos enfrentado es llevar un control de la posición. Queríamos intentar que el ascensor fuese lo más automático posible por lo que evitamos la necesidad de intervenir con switches o botones para determinar la posición. Para resolver este problema introdujimos la entidad position, que utiliza la entrada de los motores y un temporizador para llevar un control de la posición. Esta entidad es una especie de realimentación que permite a la entidad Gestor, que decide hacia donde debe ir el ascensor, conocer en todo momento la posición virtual. También es un dato importante en la eliminación de las peticiones o en la representación de la posición.

Decisión sobre múltiples peticiones.

Al introducir la posibilidad de recibir múltiples posiciones se debía llevar un orden lógico para atenderlas. Además, al recibir el sistema peticiones mientras opera debía poder afrontar la llegada de estas nuevas peticiones, atendiendo a todas y de la manera más optima posible. La solución que se propuso fue separar la lógica de la decisión del protocolo de actuación en dos entidades distintas Gestor y FMS. El gestor permitía realizar una lógica compleja para poder elegir que peticiones deben atenderse primero, pudiendo cambiar esta decisión al momento sin perjudicar al movimiento del ascensor. Esto se debe a que se encuentra aislada de la entidad FMS, que es la que regula el movimiento del ascensor.

Cambios de rumbo.

Rápidamente descubrimos que para poder atender todas las peticiones de manera eficiente debíamos saber hacia dónde nos estábamos moviendo. Por ejemplo, si estamos avanzando hacia arriba es mucho mejor ir recogiendo pasajeros en nuestro camino que luego tener que volver a por ellos. Además, si por ejemplo tuviésemos una petición más cercana, pero debemos cambiar el sentido de nuestra marcha esto podría hacer que alguna peticiones quedasen desatendidas. Si periódicamente señalásemos peticiones en la planta 0 y 1 podríamos desatender una petición en la planta 3 si solo tuviésemos en cuenta la cercanía.

En primera instancia colocamos una realimentación de los motores a la entidad gestor, de modo que si avanzásemos para arriba recogiésemos todas las peticiones antes de volver a bajar. No obstante, al poder interrumpir la marcha para recoger pasajeros los motores llegan a parase, y al priorizar en ese caso la cercanía nos encontraríamos con la misma situación. La solución pasaba por realimentar la propia decisión, de este modo si decidíamos ir para arriba hasta que no hubiese más peticiones en esa dirección y decidiésemos otra opción el Gestor seguiría priorizando las peticiones en esa dirección.

Mejoras

Las mejoras principales que hemos realizado sobre el proyecto original se han propuesto en la introducción. En ellas se puede resaltar sobre todo la recepción de múltiples peticiones y la gestión de estas, que ya hemos atacado en los problemas anteriores.

En cuanto a futuras mejoras podríamos resaltar especialmente una que rompe un poco con el esquema principal, por lo que no se ha implementado, pero si se quisiese llevar este programa a una solución más práctica debería implementarse. Esta mejora consistiría en dejar de considerar a las plantas como estados discretos y tenerlas en cuenta como estados continuos, es decir, establecer etapas de transición entre plantas. La primera manera sería crear nuevas estados en la entidad posición para cada posición entre plantas, lo cual es poco óptimo ya que se debería ampliar más todavía la lógica de programación. En la segunda manera nos evitamos la creación de nuevos estados, implementado una señal de salida de reloj que se conectaría a la entidad Decisión, sincronizándola. Esta señal se activaría siempre que el siguiente estado de la posición coincida con posición actual, es decir una vez se establezca la posición. Es una proposición interesante para una posible mejora para una aplicación práctica, pero para el propósito de la simulación en placa el programa actual funciona correctamente.

Conclusiones

En este trabajo se han programado en todas las sintaxis existentes, estructural, behavioral y dataflow. También se han realizado distintas simulaciones para las entidades más relevantes. Se han encontrado distintos problemas de conceptos y teoría que se han podido solucionar, aprendiendo de los errores y consiguiendo un resultado final satisfactorio. Se ha trabajado con las placas FPGA para realizar pruebas y ajustar las restricciones. Este proyecto ha sido clave para asentar los conocimientos aprendidos acerca del VHDL.