SESIÓN 3:

MÓDULO *UC_DIR* –

MÓDULO *PISOS_DEMANATS*

En esta sesión de prácticas desarrollaremos dos módulos del controlador del ascensor. Esta será la última sesión en la que se desarrollen dos módulos.

El primer módulo que se desarrollará será uno de los dos módulos que conforman la unidad de control del ascensor, concretamente el módulo **UC_DIR**. Como ya se comentó en la presentación de las prácticas, la unidad de control es algo así como el "cerebro" del controlador del ascensor. El módulo **UC_DIR** tiene como finalidad determinar la dirección del movimiento del ascensor. Este módulo consiste en un circuito secuencial que implementa una pequeña máquina de estados.

El segundo módulo a desarrollar es el módulo **Pisos_demanats**, perteneciente a la Unidad de Proceso. Este módulo consiste básicamente en un registro de 8 bits en el que se memorizan los pisos que han sido solicitados para que el ascensor vaya a ellos.

Aquellos equipos que vayan a trabajar en un ordenador del laboratorio, antes de iniciar la sesión tienen que:

-descargar de la nube la carpeta del proyecto comprimida y moverla a **U:\FCPract**

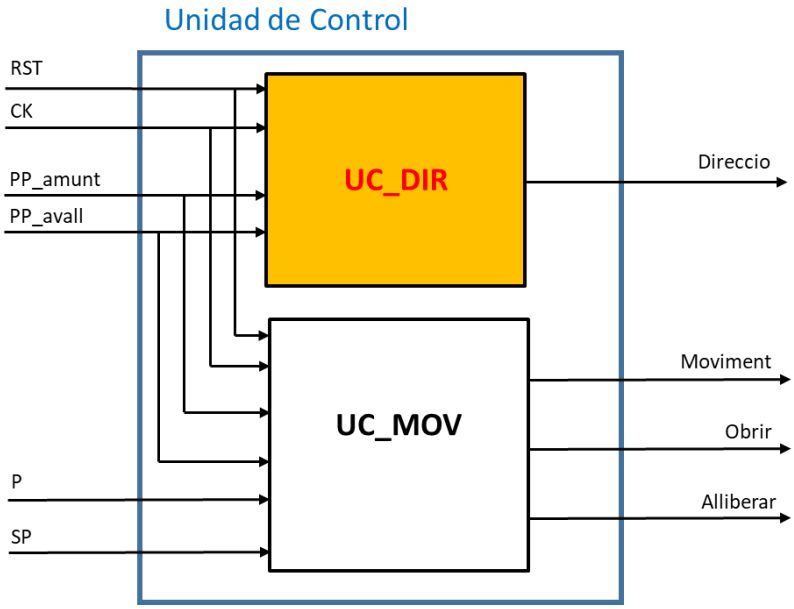
-una vez está la carpeta comprimida en **U:\FCPract**, descomprimir la carpeta

-borrar la carpeta comprimida

1 MÓDULO *UC_DIR*

1.1 DESCRIPCIÓN Y DISEÑO

Tal como se comentaba en la introducción, este módulo, perteneciente a la Unidad de Control, tiene como función determinar la dirección del movimiento del ascensor. Consiste en un circuito secuencial que implementa una sencilla máquina de estados.



Tal como se puede ver en la imagen anterior, el módulo **UC_DIR** tiene una sola salida, *Direccio*, la cual indica la dirección del movimiento del ascensor: 'hacia arriba' (*Direccio*=1) o 'hacia abajo' (*Direccio*=0). Evidentemente, la señal *Direccio* solo es tenida en cuenta si el ascensor se mueve; si el ascensor permanece parado la señal *Direccio* no tiene ningún efecto sobre el comportamiento del ascensor. Es el bloque **UC_MOV**, el otro módulo de la Unidad de Control, quien decide si el ascensor se mueve o permanece parado mediante la generación de la señal *Moviment*.

La determinación de la dirección de movimiento (*Direccio*) se hace en función de la dirección de movimiento actual y de si hay pisos solicitados por encima o por debajo del piso actual (entradas *PP_amunt* y *PP_avall*). Más concretamente:

- Si la dirección de movimiento actual es hacia arriba, y hay pisos solicitados por encima del piso actual (*PP_amunt*=1), la dirección de movimiento ha de seguir siendo "hacia arriba" (*Direccio*=1).
- Si la dirección de movimiento actual es hacia arriba, y no hay pisos solicitados por encima del piso actual (*PP_amunt*=0), pero sí hay pisos solicitados por debajo (*PP_avall*=1), se debe cambiar la dirección de movimiento a "hacia abajo" (*Direccio*=0).
- Si la dirección de movimiento actual es hacia abajo, y hay pisos solicitados por debajo del piso actual (*PP_avall*=1), la dirección de movimiento ha de continuar siendo "hacia abajo" (*Direccio*=0).

- Si la dirección de movimiento actual es hacia abajo, y no hay pisos solicitados por debajo del piso actual (*PP_avall*=0), pero sí hay pisos solicitados por arriba (*PP_amunt*=1),se debe cambiar la dirección de movimiento a "hacia arriba" (*Direccio*=1).
- Si *PP_amunt*=*PP_avall*=0, la dirección de movimiento actual se mantiene, es decir, si es hacia arriba sigue siendo "hacia arriba" (*Direccio*=1), y si es hacia abajo sigue siendo "hacia abajo" (*Direccio*=0).

Es interesante recordar que las señales *PP_amunt* y *PP_avall* son generadas por el módulo **Senyals_UC** de la Unidad de Proceso, el cual ya fue diseñado en la sesión 2.

Para materializar este módulo que es un circuito secuencial se va a aplicar la metodología que ha sido estudiada en la parte de teoría/problemas de la asignatura y que queda resumida en la siguiente diapositiva que fue comentada en una de las clases de problemas.

RECORDEMOS la secuencia de pasos metodológicos para materializar SISTEMAS SECUENCIALES o MEF (Máquinas de Estados Finitos)

1. ¿Cuáles son las entradas y salidas?
2. ¿Qué información deberemos guardar en el estado?
3. ¿Cuántos estados necesitaremos? ¿Cuántos bits necesitamos para guardarlos? El número de bits me dice cuantos flip flops serán necesarios
4. ¿Qué valores de salidas estarán asociadas a cada estado (máquina de Moore) o a cada transición entre estados (máquina de Mealy)?
5. ¿Cuál deberá ser la frecuencia del reloj de sincronización?
6. **Dibujar el grafo de comportamiento del circuito**
7. Extraer del grafo la **tabla de transición de estados** y la **tabla de salidas**
8. Asignar códigos binarios a los estados
9. A partir de las tablas de transición de estados y de salidas en las que los estados ya se han sustituido por sus códigos, diseñar las funciones de entrada a cada uno de los flip flops (=función estado siguiente) y las salidas del circuito
10. **Dibujar el circuito**

1. ¿Cuáles son las entradas y las salidas?

Teniendo en cuenta lo que se ha explicado en párrafos anteriores respecto al comportamiento deseado del módulo **UC_DIR**, este tendrá como entradas de datos las señales *PP_amunt* y *PP_avall*. *PP_amunt* indica si hay solicitudes al ascensor pendientes de ser atendidas en los pisos superiores al piso en el que se encuentra el ascensor. Por su parte, *PP_avall* indica si hay solicitudes al ascensor pendientes de ser atendidas en los pisos inferiores al piso en el que se encuentra el ascensor.

Solo hay una salida, *Direccio*, que con su valor determina la dirección del movimiento del ascensor (1 -> hacia arriba; 0 -> hacia abajo).

Evidentemente, el módulo tendrá también una entrada para la señal de sincronización (*CK*) y una para la entrada de reset asíncrono (*RST*), el cual será **activo a alta** (es decir, cuando valga 1).

2. ¿Qué información deberemos guardar en el estado?

Para que el módulo pueda cumplir su función, debe "recordar" ("conocer") la dirección del movimiento actual del ascensor. Esa es la información que se deberá guardar en el estado.

3. ¿Cuántos estados necesitaremos? ¿Cuántos bits necesitamos para guardarlos? ¿Cuántos flip-flops van a ser necesarios?

Tal como se decía en la pregunta anterior, en el estado se va a guardar la dirección del movimiento actual del ascensor. La dirección solo puede ser o bien "hacia abajo" o bien "hacia arriba". Por tanto, solo habrá **dos estados** (**DIR_ABAJO** y **DIR_ARRIBA**). Para guardar esos dos estados es suficiente con un solo bit. En consecuencia, el subbloque de memoria del circuito secuencial estará formado solamente por **un flip-flop D**.

4. ¿Será la máquina de estados una máquina de Moore o una máquina de Mealy? ¿Qué valores de las salidas estarán asociadas a cada estado (en el caso de una máquina de Moore) o a cada transición entre estados (en el caso de una máquina de Mealy)?

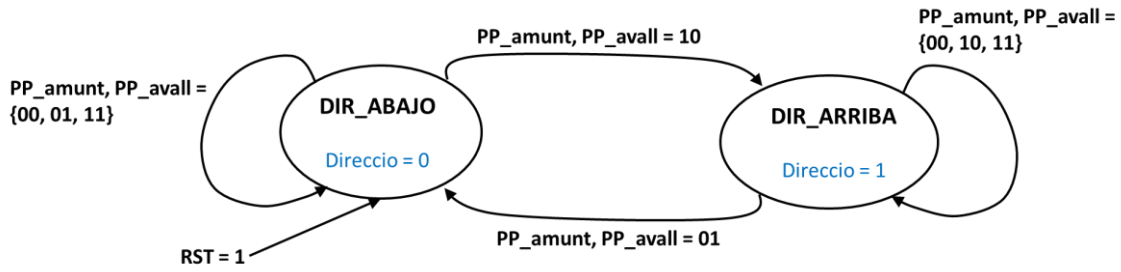
Vamos a decidir implementar la máquina de estados como una máquina de Moore. Por tanto, la salida solo depende del estado en que se encuentra la máquina. En el estado DIR_ABAJO, *Direccio* valdrá 0. En el estado DIR_ARRIBA, *Direccio* valdrá 1.

5. ¿Cuál deberá ser la frecuencia del reloj de sincronización?

En el enunciado no se aporta información específica de la cual se pueda derivar la frecuencia (mínima) que ha de tener el reloj de sincronización. Se usará un reloj de periodo 20 ns.

6. Dibujar el grafo de comportamiento del circuito.

El grafo de comportamiento siguiente describe el funcionamiento del módulo **UC_DIR**, teniendo en cuenta lo explicado al final de la página 2 e inicio de la página 3.



7. Extraer del grafo la tabla de transición de estados y la tabla de salidas

Tabla de transición de estados

Estado actual	*PP_amunt*	*PP_avall*	Estado siguiente
DIR_ABAJO	0	0	DIR_ABAJO
DIR_ABAJO	0	1	DIR_ABAJO
DIR_ABAJO	1	0	DIR_ARRIBA
DIR_ABAJO	1	1	DIR_ABAJO
DIR_ARRIBA	0	0	DIR_ARRIBA
DIR_ARRIBA	0	1	DIR_ABAJO
DIR_ARRIBA	1	0	DIR_ARRIBA
DIR_ARRIBA	1	1	DIR_ARRIBA

Tabla de salidas

Estado actual	*Direccio*
DIR_ABAJO	0
DIR_ARRIBA	1

8. Asignación de estados

La asignación (codificación) de estados en este caso es muy sencilla teniendo en cuenta que solo hay dos estados (**DIR_ABAJO** y **DIR_ARRIBA**). Como ya se indicó en el punto 3, para representar los dos estados es suficiente con un solo bit. La asignación de estados que haremos será la siguiente:

DIR_ABAJO = **0**

DIR_ARRIBA = **1**

Cuando el único flip-flop que conforma el subbloque de memoria almacene un 0 entonces la máquina de estados se hallará en el estado DIR_ABAJO. Cuando el flip-flop almacene un 1 entonces la máquina de estados se hallará en el estado DIR_ARRIBA.

9. A partir de las tablas de transición de estados y de salidas en la que los estados ya se han sustituido por sus códigos, diseñar las funciones de entrada a cada uno de los flip-flops (función estado siguiente) y las salidas del circuito.

Vamos a reescribir las tablas de transición de estados y la tabla de salidas teniendo en cuenta la codificación de los estados. Se va a denominar **q** el estado actual del flip-flop que conforma el subbloque de memoria y **qsig** el estado siguiente de dicho flip-flop.

Tabla de transición de estados

q	*PP_amunt*	*PP_avall*	qsig
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tabla de salidas

q	*Direcció*
0	0
1	1

Observando la tabla de transición de estados, se puede ver que **qsig** es una función booleana de tres variables: **q**, **PP_amunt** y **PP_avall**. Si introducimos esa tabla de verdad en la herramienta "Analizar Circuito" de VerilUOC_Desktop, podemos obtener la expresión minimizada de la misma.

$$qsig = q \cdot \overline{PP\_avall} + q \cdot PP\_amunt + \overline{PP\_avall} \cdot PP\_amunt$$

El flip-flop que se va a utilizar es un flip-flop D. La ecuación característica de los flip-flops D es **qsig = D**. Con ella se indica que el estado siguiente en un flip-flop D es el valor presente en la entrada D del mismo. Por ello, para conseguir que el flip-flop se comporte de la manera que establece la tabla de transición de estados, lo único que hay que hacer es implementar la función booleana **qsig** y conectarla a la entrada D del flip-flop:

$$D = q \cdot \overline{PP\_avall} + q \cdot PP\_amunt + \overline{PP\_avall} \cdot PP\_amunt$$

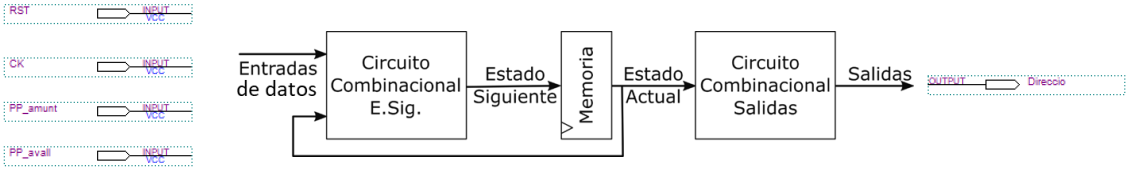
Finalmente, si se observa la tabla de salidas resulta obvio que la salida *Direccio* equivale a q.

$$Direccio = q$$

10. Dibujar el esquema lógico del circuito

Finalmente debéis realizar el circuito secuencial que constituye el módulo UC_DIR y que implementa la máquina de Moore que hemos descrito en los apartados anteriores. En la imagen del final de la página tenéis el esquema genérico de una máquina de Moore. **Vosotros debéis implementar el módulo UC_DIR sustituyendo cada subbloque por lo que corresponda.** Antes de hacerlo tened en cuenta los siguientes aspectos:

- La entrada **RST** del circuito es activa a alta, es decir, debe resetear el flip-flop cuando vale 1. Sin embargo, como ya se vio en la sesión 2, los flip-flops de Quartus tienen una entrada de reset asíncrono (denominada **CLRN**) que es activa a baja (es decir, para resetear el flip-flop hay que introducir un 0 por la entrada **CLRN**). Por ello, lo que se ha de hacer es negar la señal **RST** antes de conectarla a la entrada **CLRN** del flip-flop.
- La entrada **PRN** del flip-flop (que es la entrada de set asíncrono) se puede quedar sin que se le conecte nada. Ello se puede hacer porque internamente tiene lo que se denomina un *pull-up*. Un *pull-up* es una conexión interna por defecto a 1. Si no se le conecta nada, el pull-up hace que esa entrada esté conectada a 1 y, por tanto, el set queda deshabilitado (se debe recordar que la entrada **PRN** es activa a baja).
- El flip-flop ha de tener asignado como nombre de instancia el nombre **ff**. Para poner ese nombre de instancia, hay que seleccionar el flip-flop, hacer clic con el botón derecho del ratón, escoger la opción **Properties** y editar el campo **Instance name**. El nombre de instancia del flip-flop (**ff**) nos será útil posteriormente en la simulación para conocer cuál es el estado del flip-flop (es decir, para saber si tiene almacenado un 0 o un 1).



Entra el esquema lógico del circuito y guárdalo en el fichero **UC_DIR.bdf** en la carpeta del proyecto (U:\FCPract\4XX_YY). A continuación, define el módulo como *top-level entity* y realiza la compilación, asegurándote de que no hay errores técnicos en el módulo. Finalemente, genera un símbolo del módulo.

Para concluir se procederá a simular funcionalmente el módulo para verificar que funciona de manera correcta. Ello se explica en el siguiente apartado.

1.2 SIMULACIÓN DEL MÓDULO UC_DIR

1.2.1 CREACIÓN Y EDICIÓN DE LOS VECTORES DE SIMULACIÓN

Como ya se vio en las anteriores sesiones de prácticas, lo primero que hay que hacer cuando queremos simular un esquemático es generar el llamado "fichero de vectores de simulación" (también llamado "fichero de estímulos" o "fichero de formas de onda").

Se empezará creando el fichero de vectores de simulación. Para ello, ejecuta la orden:

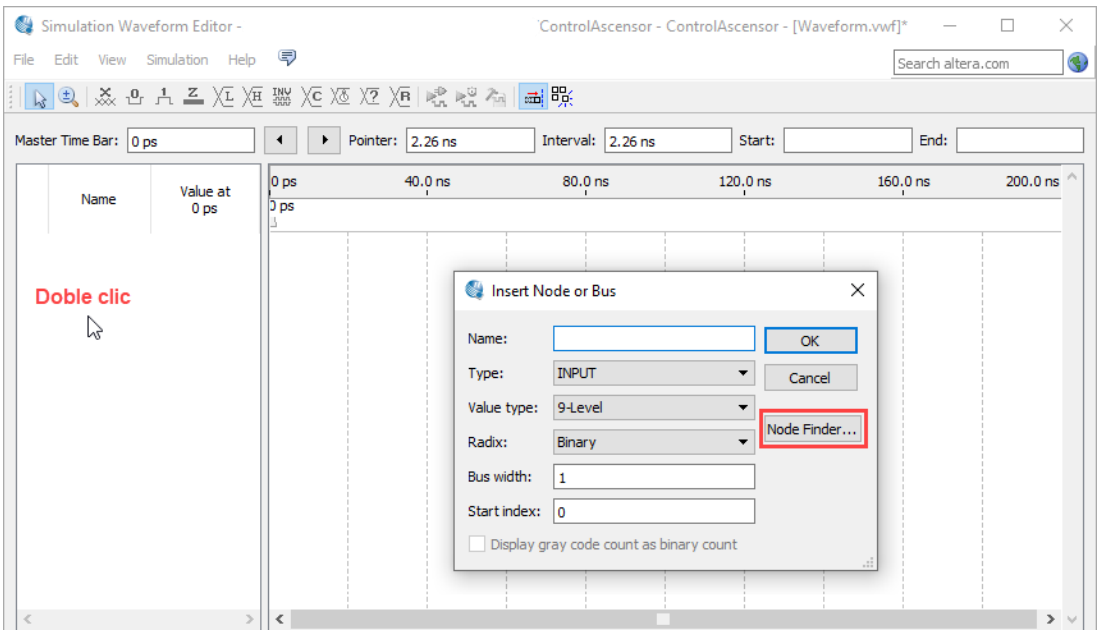
File → New

En el menú desplegable que aparece, selecciona ***University Program VWF*** y clica en **OK**.

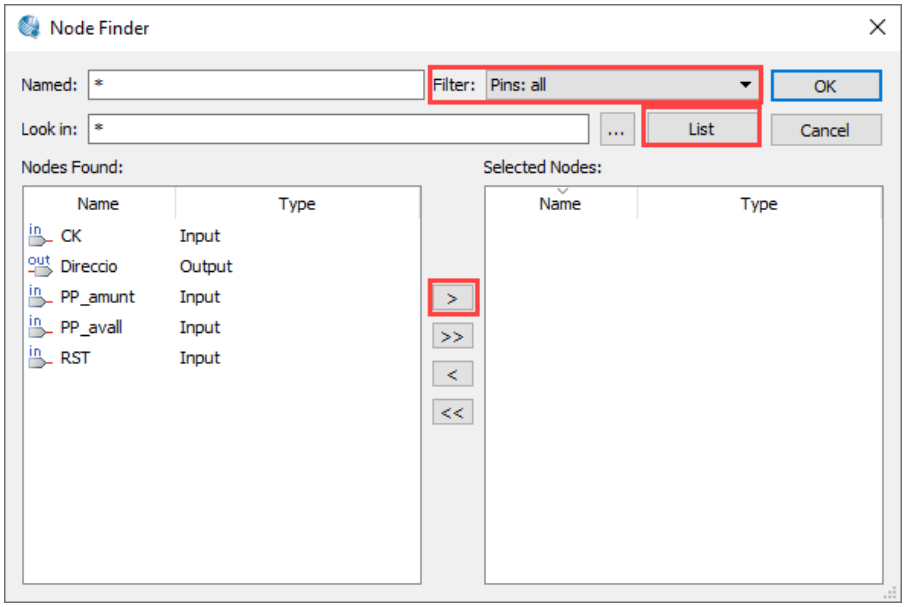
Establece la anchura de las franjas temporales del editor de formas de onda (20 ns) y la duración de la simulación (200 ns):

Edit → Grid size… → 20 ns
Edit → Set End Time… → 200 ns

Para introducir las entradas y sus valores, haz doble clic debajo de la columna *Name*. En el formulario que aparece, clica en el botón **Node Finder**.

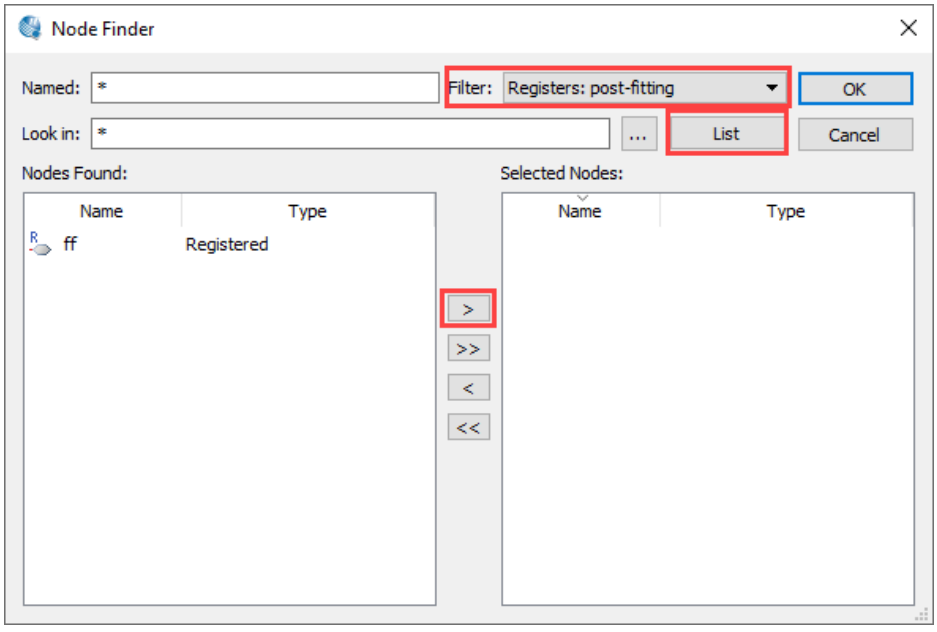


Se abrirá una nueva ventana desde donde podrás seleccionar los nodos de entrada (*RST*, *CK*, *PP_amunt* y *PP_avall*) y los nodos de salida (*Direccio*):



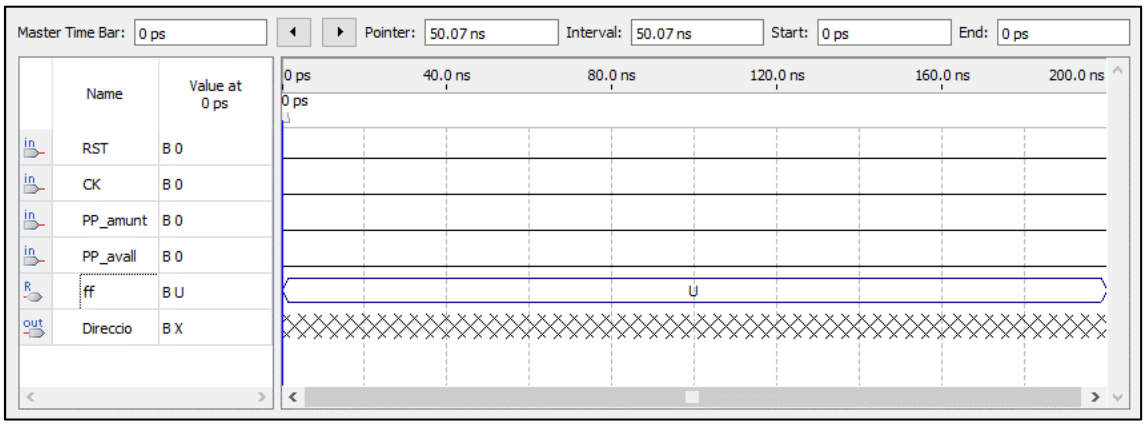
Selecciona la opción "**Pins: all**" en Filter, y clica en "**List**". En la subventana "**Nodes Found**" aparecerán todos los nodos de entrada y salida de **UC_DIR**. Selecciona el nodo *RST* y clica en **>** . Repite el mismo proceso para los nodos *CK*, *PP_amunt*, *PP_avall* y *Direccio*. Para acabar clica en **OK**.

A continuación, selecciona la opción "**Registers: post-fitting**" en Filter, y clica en "**List**". En la subventana "**Nodes Found**" aparecerá el nodo *ff* de **UC_DIR** (que equivale a la salida **q** del flip-flop **ff**). Selecciona el nodo *ff* y clica en **>** . Finalmente clica en **OK**.



Una vez seleccionados los nodos que te interesan, clica en ***OK***. Te volverá a aparecer la ventana **Insert Node or Bus**. Vuelve a clicar en ***OK***. Los nodos se incorporarán a la ventana de simulación.

Vuelve al simulador y verás todos los nodos. En general, es recomendable poner siempre en primer lugar los nodos de entrada, después los nodos internos y después los nodos de salida ya que ello facilitará la legibilidad del resultado de la simulación. Para conseguirlo puedes arrastrar un nodo a la posición deseada. En este caso arrastra el nodo *ff* para situarlo antes de *Direccio*. Tras ellos los nodos deberían quedar como indica la imagen.



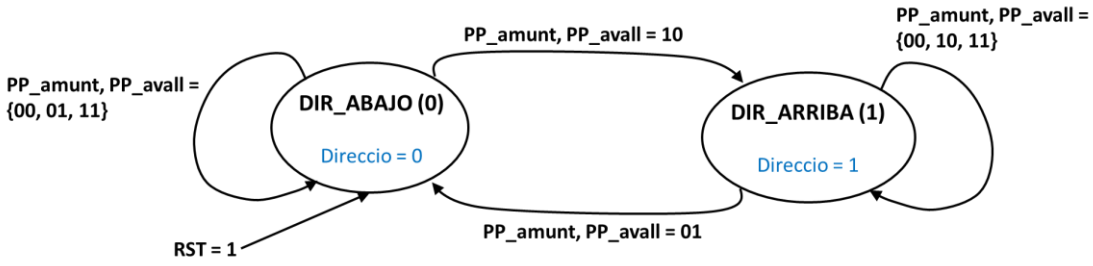
Observa que todas las entradas están a 0. Por su parte el nodo *ff* y la salida *Direccio* están fijados a un valor indeterminado (concretamente *ff* tiene el valor *Uninitialized* [U] y *Direccio* tiene el valor *Unknown* [X], respectivamente).

Lo siguiente que hemos de hacer es definir el valor de las entradas en cada instante de tiempo. Pero antes hay que establecer una estrategia de test.

Estrategia de test

Para comprobar que el circuito secuencial (que implementa la máquina de estados) funciona correctamente haremos lo siguiente:

- En primer lugar, resetearemos el circuito lo cual debería provocar que el flip-flop **ff** pase al estado 0 (recuerda que 0 es la codificación del estado DIR_ABAJO).
- En la simulación, la máquina de estados tiene que pasar por todos sus estados (en este caso, los dos estados que tiene) y ha de realizar todas las transiciones de estados que aparecen en el grafo de comportamiento de la máquina de estados:



Una posible forma de conseguirlo sería aplicar los valores en las entradas *PP_amunt* y *PP_avall* que se citan a continuación. En rojo se indica el comportamiento que debería verse en el flip-flop *ff* (el estado de la máquina) tras el flanco de subida de *CK,* cuando se ejecute la simulación. Se puede observar que la máquina pasa por sus dos estados y realiza la totalidad de las transiciones entre estados indicadas en el grafo de comportamiento.

Tras el reset, *ff* = 0 (estado DIR_ABAJO)

PP_amunt = 0, *PP_avall* = 0

ff = 0 (estado DIR_ABAJO)

PP_amunt = 0, *PP_avall* = 1

ff = 0 (estado DIR_ABAJO)

PP_amunt = 1, *PP_avall* = 1

ff = 0 (estado DIR_ABAJO)

PP_amunt = 1, *PP_avall* = 0

ff = 1 (estado DIR_ARRIBA)

PP_amunt = 0, *PP_avall* = 0

ff = 1 (estado DIR_ARRIBA)

PP_amunt = 1, *PP_avall* = 0

ff = 1 (estado DIR_ARRIBA)

PP_amunt = 1, *PP_avall* = 1

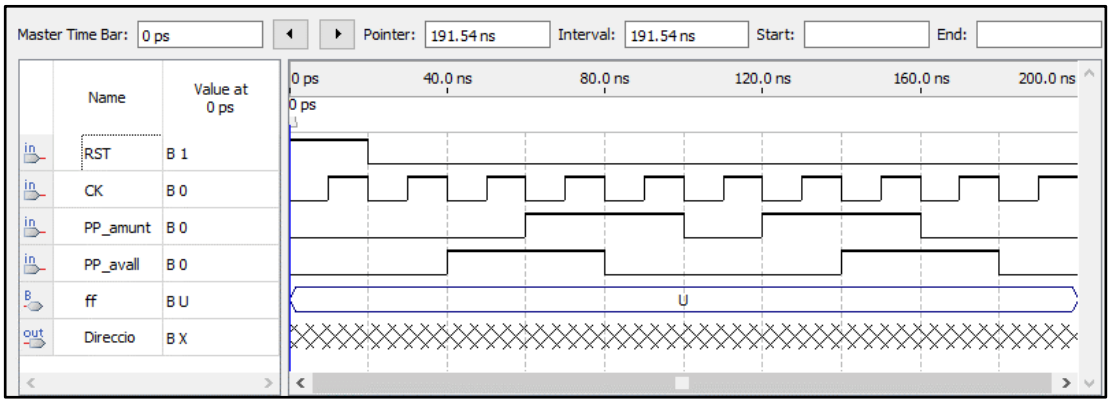
ff = 1 (estado DIR_ARRIBA)

PP_amunt = 0, *PP_avall* = 1

ff = 0 (estado DIR_ABAJO)

- La duración de las señales *PP_amunt* y *PP_avall* será de un periodo de reloj, de flanco de bajada a flanco de bajada del reloj (CK). De esa forma cuando se produzca el flanco de subida

del reloj, esas señales estarán perfectamente estables. En la siguiente imagen se puede observar la totalidad de los estímulos de entrada.



Ahora que ya hemos establecido la estrategia de test que nos permitirá determinar si el circuito se comporta correctamente, hemos de introducir los valores en los nodos de entrada. Para ello, tenéis que aplicar lo que ya aprendisteis en sesiones previas. Recordad que el reloj no es necesario dibujarlo manualmente ya que puede ser generado automáticamente mediante el botón  .

Entra todos los valores que hemos definido para las entradas (es decir, *CK, RST, PP_amunt* y *PP_avall*), y guarda el fichero de simulación (**File → Save**) con el nombre **UC_DIR.vwf** en la carpeta del proyecto (**U:\FCPract\4XX_YY**).

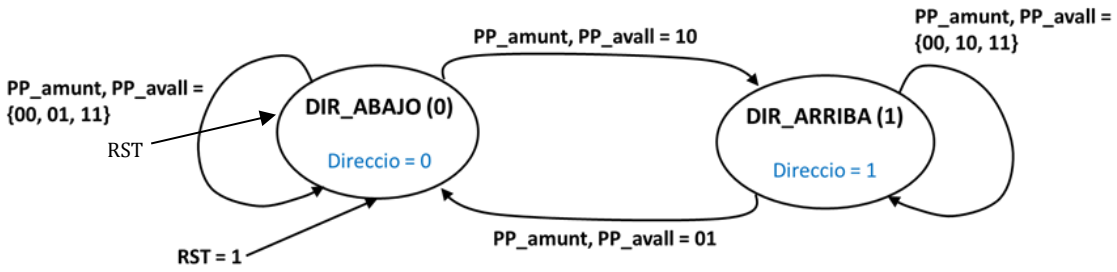
1.2.2 SIMULACIÓN DEL CIRCUITO

Selecciona el simulador propio de Quartus II (*qsim*) como el simulador que vamos a utilizar:

Simulation → Options → Seleccionar Quartus II Simulator → Pulsar OK

Realiza una simulación funcional. Para ello clica en el botón  .

El resultado de la simulación aparece en una nueva ventana. Se ha de comprobar que los valores de las salidas que da el simulador coinciden con los valores esperados en todos los casos. En primer lugar, se ha de comprobar que la transición de estados se realiza correctamente (ver el grafo de comportamiento). Para ello hay que observar el valor almacenado en el flip-flop *ff* tras cada flanco de subida de reloj.



Una vez comprobado que las transiciones de estado se realizan correctamente, hay que verificar que la salida *Direccio* adopta siempre el valor correcto. Ello es muy sencillo de comprobar ya que como recordaréis *Direccio* equivale a la salida *q* del flip-flop **ff** (por tanto, *Direccio* y *ff* han de valer siempre lo mismo).

Finalmente, guarda el resultado de la simulación ejecutando el comando:

File → Save As …

En el formulario que aparece **selecciona la subcarpeta simulation\qsim del proyecto (U:\FCPract\4XX_YY\simulation\qsim)** e indica como nombre del fichero **UC_DIR_sim.vwf**. Finalmente pulsa el botón **Abrir**.

El fichero con el resultado de la simulación (**UC_DIR_sim.vwf**) no se considera que forme parte del proyecto y por ese motivo no aparece en la sección *Files* del *Project Navigator*. Para añadirlo al proyecto se ejecutará el comando:

Project → Add/Remove Files in Project…

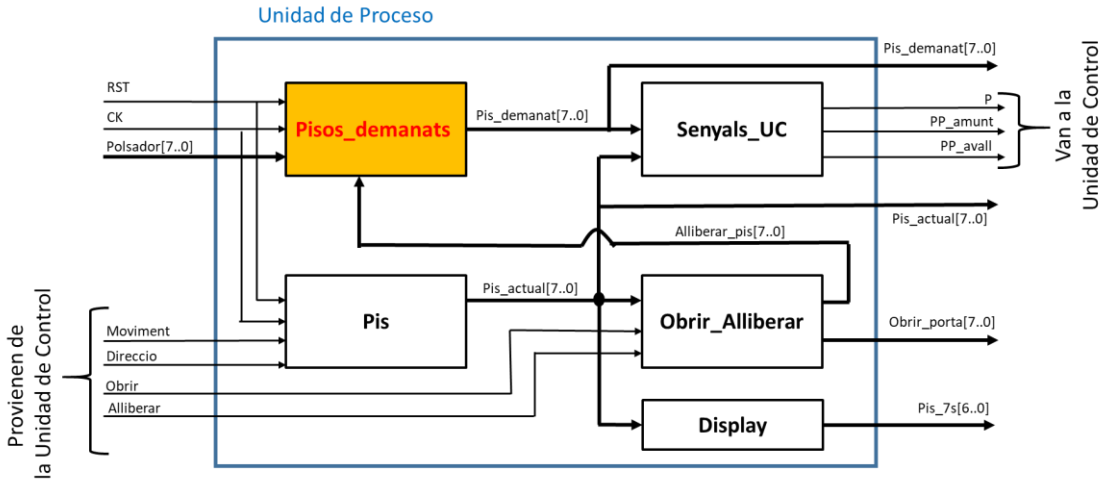
y se seguirá el mismo procedimiento que se siguió para añadir el fichero **Obrir_Alliberar_sim.vwf** al proyecto en la sesión 1.

Si has hecho correctamente el proceso ahora se debería ver el fichero simulation/qsim/UC_DIR_sim.vwf en la sección *Files* del *Project Navigator*.

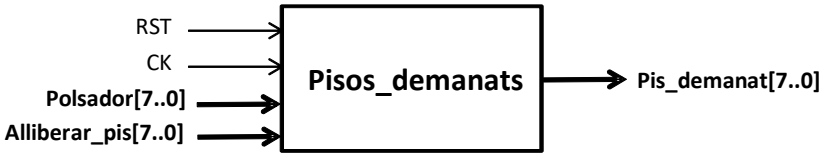
2 MÓDULO *PISOS_DEMANATS*

2.1 DESCRIPCIÓN Y DISEÑO

Este módulo memoriza en un registro de 8 bits los pisos que han sido solicitados para que el ascensor vaya a ellos. Tras ser atendida una solicitud, dicha solicitud es eliminada. El registro está formado por 8 flip-flops D.

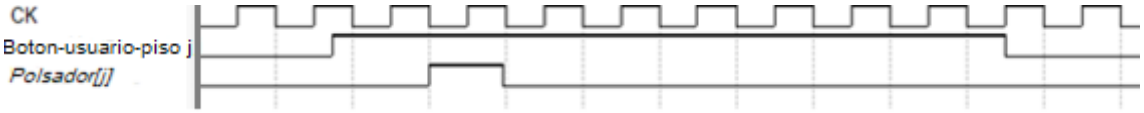


Veamos cuáles son las entradas y salidas del circuito:



- La entrada *RST* tiene como finalidad resetear asíncronamente el registro. La señal RST es activa a alta, es decir, debe realizar su misión cuando adopta el valor 1.
- La entrada *CK* es la señal de sincronización (reloj) que rige el funcionamiento del registro.
- La entrada *Polsador[7..0]* es un bus de 8 bits. Cada bit está vinculado a un piso. Cuando un usuario del ascensor realiza una solicitud para que el ascensor vaya a un determinado piso, el bit correspondiente a ese piso en el bus *Polsador* se pone a 1.
- La entrada *Alliberar_pis[7..0]* es un bus de 8 bits que proviene del módulo **Obrir_Alliberar** (implementado en la sesión 1). Cada bit está vinculado a un piso. Cuando un bit vinculado a un piso se pone a 1, ello quiere decir que se ha de proceder a liberar (eliminar) la solicitud para que el ascensor vaya a ese piso, puesto que esa solicitud ya ha sido atendida.
- La salida *Pis_demanat[7..0]* es un bus de ocho bits. Cada bit está vinculado a un piso e indica si hay una solicitud para que el ascensor vaya a ese piso pendiente de ser atendida. El bus *Pis_demanat* muestra el estado del registro de 8 bits presente en el módulo **Pisos_demanats**.

Las 8 señales *Polsador[7..0]* provienen de los botones de llamada de ascensor de cada uno de los pisos del edificio y de los propios botones internos del ascensor. Cuando se selecciona el piso *j*, la entrada *Polsador[j]* toma el valor 1. Se supone que el circuito de llamada a los pisos (el cual no se estudiará aquí) garantiza que cuando un usuario llama a un piso, en el bit correspondiente de la entrada *Polsador* se genera un valor 1 durante un único periodo de reloj (es decir, *Polsador[j]* vale 1 solamente durante un periodo de reloj). Ello se muestra en el siguiente cronograma.



El funcionamiento general del módulo **Pisos_demanats** es el siguiente. Cuando un usuario hace una solicitud para que el ascensor vaya al piso j, *Polsador[j]* se pone a 1 durante un periodo de reloj. Esa solicitud debe ser almacenada en el flip-flop j del registro presente en el módulo, es decir, el flip-flop j deberá cargar síncronamente el valor 1, significando ello la existencia de la solicitud. Una vez la solicitud ha sido atendida, *Alliberar_pis[j]* tomará el valor 1 indicando con ello que la solicitud debe ser eliminada. Ello provocará que el flip-flop j cargue síncronamente el valor 0, quedando así la solicitud eliminada.

Más concretamente, el módulo **Pisos_demanats** funciona así:

- Si RST=1 ⇒ los ochos flip-flops del registro adoptan asíncronamente el valor 0.
- Supongamos que RST = 0. En tal caso:
  - Si *Polsador[j]*=1 y *Alliberar_pis[j]*=0, el flip-flop j ha de cargar síncronamente el valor 1, indicando la existencia de una solicitud en el piso j.
  - Si el flip-flop j vale 1, *Polsador[j]*=0 y *Alliberar_pis[j]*=1 ⇒ el flip-flop j ha de cargar síncronamente el valor 0, eliminando la solicitud.
  - Si *Polsador[j]*=0 y *Alliberar_pis[j]*=0 ⇒ el flip-flop j ha de mantener el valor que tiene almacenado (ya sea este 0 o 1).
  - Si el flip-flop j vale 1, *Polsador[j]*=1 y *Alliberar_pis[j]*=1 ⇒ el flip-flop j ha de cargar síncronamente el valor 0, eliminando la solicitud, y no tendrá en cuenta la pulsación del usuario. Este es un caso un tanto especial.

El comportamiento descrito quedaría reflejado en la tabla de la siguiente página. En ella *q* representa la salida *q* del flip-flop j, *P* representa *Pulsador[j]* y *A* representa *Alliberar_pis[j]*. Por su parte, *qsig* representa el siguiente estado del flip-flop j.

q	*P*	*A*	*qsig*
0	0	0	0
0	0	1	x
0	1	0	1
0	1	1	x
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

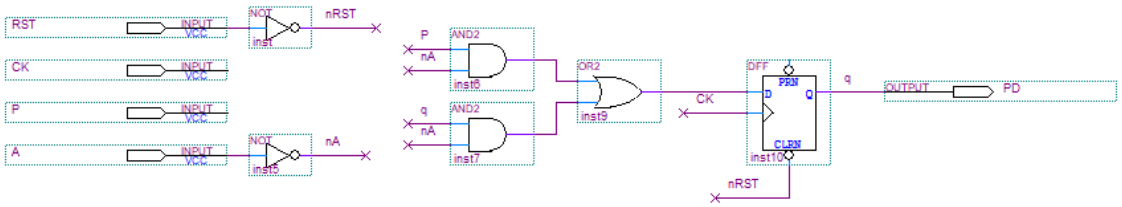
De la tabla anterior se puede deducir que:

$$qsig = P \cdot \bar{A} + q \cdot \bar{A}$$

Los flip-flops que usamos son de tipo D, siendo su ecuación característica *qsig* = *D*. Por tanto, a la entrada *D* se le deberá conectar la implementación de la función booleana *qsig*:

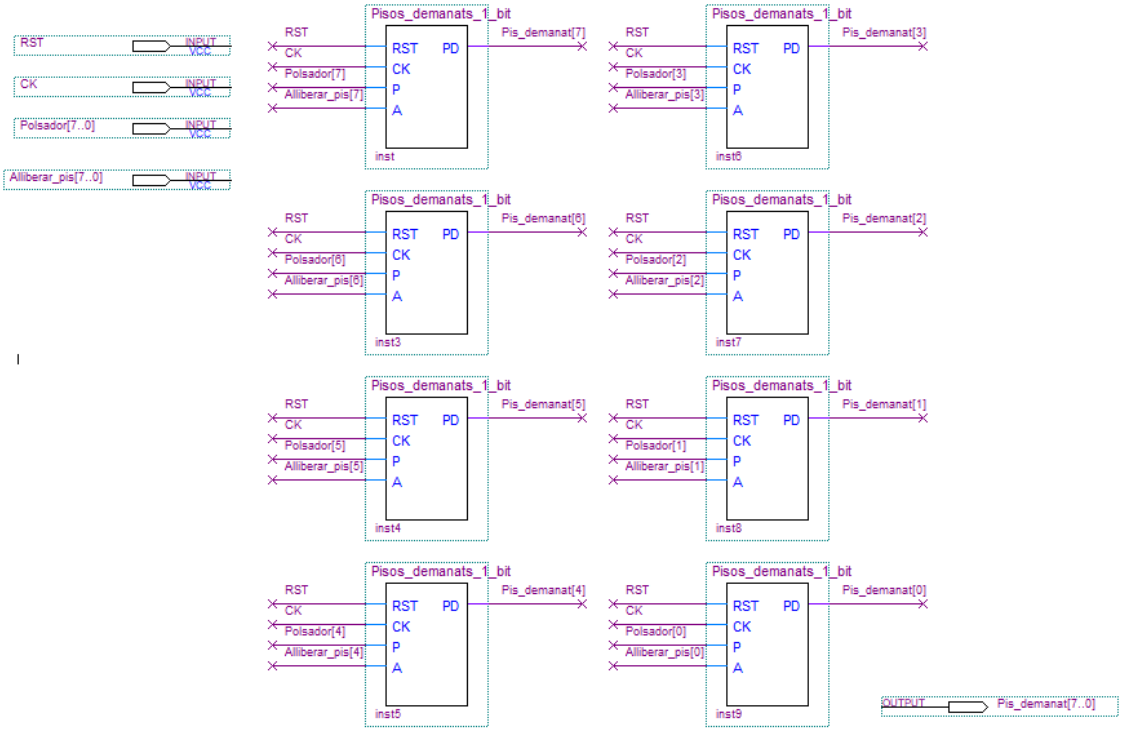
$$D = P \cdot \bar{A} + q \cdot \bar{A}$$

Crearemos un pequeño submódulo para cada piso que contendrá un flip-flop D y la circuitería combinacional de la función booleana que hemos obtenido anteriormente. Posteriormente utilizaremos el símbolo de dicho circuito para construir el módulo **Pisos_demanats**. El circuito para cada piso, al que llamaremos **Pisos_demanats_1_bit**, tendrá 4 entradas (CK, RST, P –de *Polsador*- y A –de *Alliberar_pis*-), y una salida PD (de *Pis_demanat*):



Una vez hayas hecho este esquema lógico, guárdalo en el fichero **Pisos_demanats_1_bit.bdf** dentro de la carpeta del proyecto (U:\FCPract\4XX_YY) y crea un símbolo para él.

Replicando este circuito 8 veces tendremos el módulo **Pisos_demanats**:



Como siempre, crea el esquemático y sálvalo con el nombre **Pisos_demanats.bdf** en la carpeta del proyecto (U:\FCPract\4XX_YY). A continuación, crea un símbolo para él. Finalmente, defínelo como *top-level entity* y compílalo para verificar que no tiene errores técnicos.

En el siguiente apartado se explica cómo realizar su simulación.

2.2 SIMULACIÓN

2.2.1 CREACIÓN Y EDICIÓN DE LOS VECTORES DE SIMULACIÓN

Puesto que este es el cuarto módulo que realizas, ya no se van a detallar los pasos a seguir para realizar una simulación. Si tienes alguna duda respecto a algún paso consulta la documentación relativa a los módulos que has hecho anteriormente.

Crea un fichero de estímulos en el que se hallen todas las entradas y la salida. Posiciona primero las entradas (*RST*, *CK*, el bus *Polsador* y el bus *Alliberar_pis*) y, a continuación, el bus de salida *Pis_demanat*.

Establece la anchura de las franjas temporales del editor de formas de onda (20 ns) y la duración de la simulación (1.3 us):

Edit → Grid size… → 20 ns
Edit → Set End Time… → 1.3 us

No dibujes manualmente la señal de reloj. Para crearla, selecciona la señal *CK* y, a continuación, usa el botón  que genera señales periódicas. En el formulario que se abre indica el periodo de la señal *CK* en el campo **Period** (20 ns).

Estrategia de test

Como estrategia de test se seguirá la siguiente (en rojo se indica lo que debería verse en el resultado de la simulación computado por el simulador). Consiste en comprobar que cada bit del registro se comporta de acuerdo a la tabla de verdad mostrada en la página 15, fila a fila:

- Las entradas *Polsador [7..0]* y *Alliberar_pis[7..0]* inicialmente valen 0. En primer lugar, resetearemos el circuito (RST = 1) lo cual debería provocar que los ocho flip-flops del registro adopten asíncronamente el valor 0 (es decir, *Pis_demanat[7..0]* = 00000000).

- A continuación, para cada bit j haremos lo siguiente:

q	*P*	*A*	*qsig*
0	1	0	1

  - Polsador[j] se pondrá a 1 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería ponerse a 1 (es decir, *Pis_demanat[j]* = 1).

q	*P*	*A*	*qsig*
1	0	0	1

  - Polsador[j] se pondrá a 0 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería mantener el valor 1 (es decir, *Pis_demanat[j]* = 1).

q	*P*	*A*	*qsig*
1	0	1	0

- Alliberar_pis[j] se pondrá a 1 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería ponerse a 0 (es decir, *Pis_demanat[j]* = 0).

q	*P*	*A*	*qsig*
0	0	0	0

- Alliberar_pis[j] se pondrá a 0 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería mantener el valor 0 (es decir, *Pis_demanat[j]* = 0).

q	*P*	*A*	*qsig*
0	1	0	1

- Polsador[j] se pondrá a 1 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería ponerse a 1 (es decir, *Pis_demanat[j]* = 1).

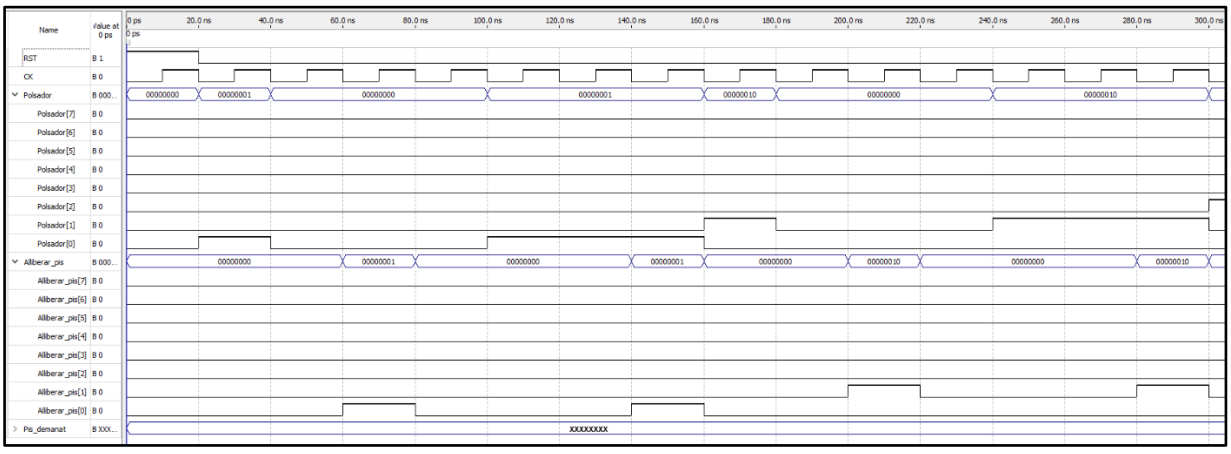
q	*P*	*A*	*qsig*
1	1	0	1

- Polsador[j] se pondrá a 1 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería mantener el valor 1 (es decir, *Pis_demanat[j]* = 1).

q	*P*	*A*	*qsig*
1	1	1	0

- Polsador[j] y Alliberar_pis[j] se pondrán ambos a 1 durante un periodo de reloj (de flanco de bajada a flanco de bajada del reloj). Con ello, tras el flanco de subida de reloj, el flip-flop j del registro debería ponerse a 0 (es decir, *Pis_demanat[j]* = 0).

Para que te resulte más sencillo de comprender lo especificado en el párrafo anterior te muestro cómo sería el fichero de estímulos desde el inicio y para los casos j=0 y j=1. Relaciona la imagen con el contenido del párrafo anterior. **Evidentemente, tú lo has de hacer para los 8 bits**.

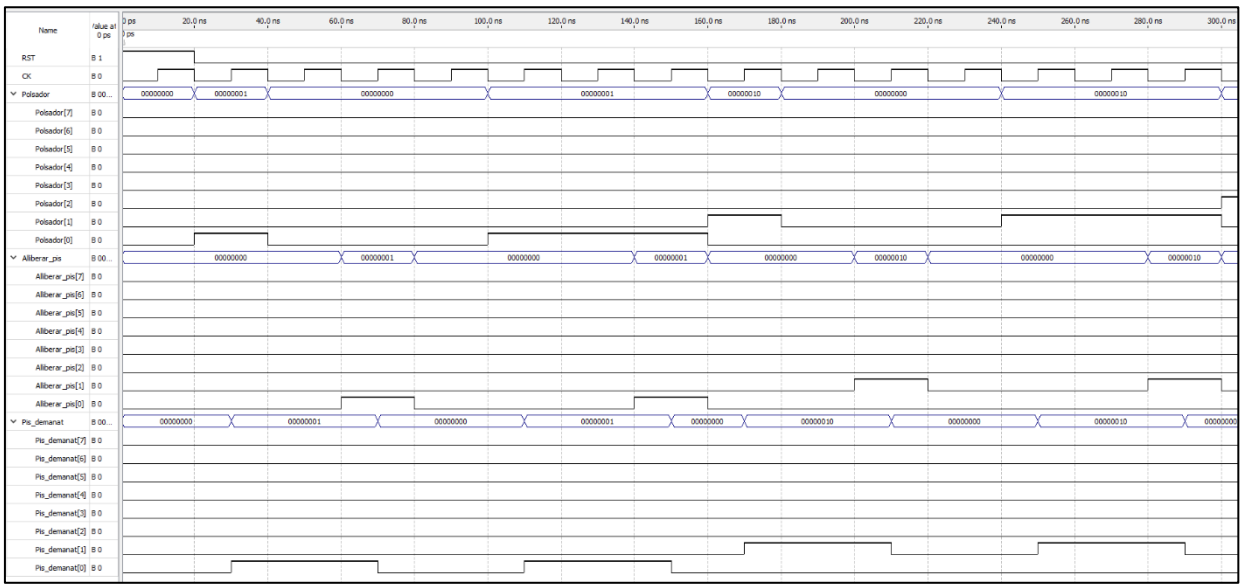


Entra todos los valores que hemos definido en el apartado de **Estrategia de test**, y graba el fichero de simulación (**File → Save**) con el nombre **Pisos_demanats.vwf** en la carpeta del proyecto (U:\FCPract\4XX_YY).

2.2.2 SIMULACIÓN DEL CIRCUITO

Selecciona el simulador propio de Quartus II (*qsim*) como el simulador que vamos a utilizar y realiza una simulación funcional.

El resultado de la simulación aparece en una nueva ventana. Te muestro el resultado para los casos j=0 y j=1. Relaciónalos con lo que se explicaba en el punto 2.2.1 (texto en color rojo).



Finalmente, guarda los resultados de la simulación ejecutando el comando:

File → Save As …

En el formulario que aparece **selecciona la subcarpeta simulation\qsim del proyecto (U:\FCPract\4XX_YY\simulation\qsim)** e indica como nombre del fichero **Pisos_demanats_sim.vwf**. Finalmente pulsa el botón **Abrir**.

El fichero con el resultado de la simulación (**Pisos_demanats_sim.vwf**) no se considera que forme parte del proyecto y por ese motivo no aparece en la sección *Files* del *Project Navigator*. Para añadirlo al proyecto se ejecutará el comando:

Project → Add/Remove Files in Project…

y se seguirá el mismo procedimiento que se siguió para añadir el fichero **Obrir_Alliberar_sim.vwf** al proyecto en la sesión 1.

Si has hecho correctamente el proceso ahora se debería ver el fichero simulation/qsim/Pisos_demanats_sim.vwf en la sección *Files* del *Project Navigator*.

Entrega correspondiente a esta tercera sesión

La entrega consistirá en un fichero .zip cuyo nombre será **FC-S3-4XX_YY.zip**, donde **4XX_YY** es el identificador del equipo de trabajo (por ejemplo, **FC-S3-411_01.zip**), que ha de contener seis ficheros:

- el fichero **UC_DIR.bdf**

- el fichero **UC_DIR_sim.vwf**

- el fichero **Pisos_demanats_1_bit.bdf**

- el fichero **Pisos_demanats.bdf**

- el fichero **Pisos_demanats_sim.vwf**

- un fichero denominado **autoría.txt** con el nombre y el NIU del autor/autores de la entrega.

La entrega del fichero **FC-S3-4XX_YY.zip** se ha de realizar a través del Campus Virtual.