BLOQUE TEMÁTICO 1

TÍTULO DE LA ACTIVIDAD: **Diseño del módulo de control**	**CÓDIGO:** **BT1.A3_P6**

FECHA:			
NOMBRE:		APELLIDOS:	

MODALIDAD:	Libre. Resolución de problemas.	TIPO:	Presencial	DURACIÓN:	120 minutos

CALENDARIO:	PS10	REQUISITOS:	

CRITERIO DE ÉXITO:	

COMENTARIOS E INCIDENCIAS:

TIEMPO DEDICADO:	Minutos	AUTOEVALUACIÓN: [entre 0 y 10 puntos]	No procede

Introducción

En esta actividad se completará el diseño del módulo de control del master I2C. Para ello se procederá de la siguiente manera: se partirá de un código incompleto del modelo del circuito de control; para completarlo tendrá que ir resolviendo una serie de cuestiones que se le formulan y que consistirán en que elija un fragmento de código entre varios que se le ofrecen (puede haber más de una opción correcta) y que cumpla con el requisito funcional que corresponda. El objetivo es que comprenda el funcionamiento del módulo a medida que va completando el modelo. Para realizar la actividad, deberá descargar el fichero BT1_A3_Z3.zip que contiene el código incompleto del modelo del circuito de control (*ctrl_i2c.vhd*). También puede resultarle de utilidad consultar la información de la BT1_A3_P3, particularmente las figuras 3 y 9.

Operación del módulo

Para empezar, debe entender el funcionamiento del módulo, que está descrito, mediante comentarios VHDL, en la cabecera del fichero *ctrl_i2c.vhd*. Dedique 10 o 15 minutos a leer y comprender el funcionamiento del módulo leyendo los mencionados comentarios. Durante la lectura, revise las entradas y salidas de la interfaz del módulo que forman parte de la declaración de entidad.

Modelo VHDL sintetizable del control del master I2C

Para la realización del módulo de control se recurre al diseño de un autómata. Para el modelado del autómata se emplea una técnica ligeramente distinta a la que usted está habituado a utilizar. La diferencia que va a encontrar en el modelo consiste en que en el proceso que modela las transiciones de estado (un proceso para el modelado de circuitos secuenciales síncronos), se asigna además valor a varias señales, algunas de las cuáles son salidas del módulo (*tx_ok*, *fin_tx* y *fin_byte*) y, el resto, señales internas (*cnt_pulsos_SCL*, *nWR* y *nWR_op*). Cada señal a la que se asigna valor de esta manera modela un *flip-flop;* son, por tanto, salidas *registradas* del autómata. Esta técnica de modelado de autómatas, muy empleada *profesionalmente*, permite obtener modelos claros y compactos del circuito.

El modelo del autómata cuenta también con salidas *combinacionales* modeladas mediante sentencias concurrentes y que se corresponden con el estilo de modelado al que usted está habituado.

A continuación se va a ir explicando la estructura del código y se irán planteando cuestiones para que usted lo complete.

Señales internas

En el cuerpo de arquitectura del modelo se declara el tipo para definir los seis estados del autómata, la señal de estado y tres señales internas *cnt_pulsos_SCL*, *nWR* y *nWR_op*, necesarias para la ejecución de las operaciones de control; se declara también otra señal, *ACK_lectura*, cuyo propósito es simplificar una expresión *farragosa* que se revisará más adelante.

Proceso síncrono

El proceso que modela las transiciones de estado y asigna valor a algunas señales se encuentra al principio del cuerpo de arquitectura. El reset asíncrono inicializa el circuito de la siguiente manera:

1. La señal de estado toma el valor *libre*; en este estado el control está preparado para aceptar la orden de inicio de una transacción (activación de la entrada *ini*); cualquier otro valor de la señal *estado* implica que hay una transacción en curso y no se atiende a la activación de *ini*.
2. c*nt_pulsos_SCL* que, como se verá, modela un contador que se emplea para contar los pulsos del reloj I2C que componen la transferencia de un byte, se inicializa a "0000".
3. La salida *fin_tx* se pone a 1, indicando que el master I2C está preparado para aceptar la orden de inicio de una transacción, las salidas *tx_ok* y *fin_byte* se ponen a 0 (el valor de *tx*_ok es irrelevante después de un reset y *fin*_byte sólo debe activarse cada vez que se completa la transferencia de un byte)
4. Las señales *nWR* y *nWR_op*, de las que hablaremos pronto, se inicializan a 0.

Por tanto, tras el reset asíncrono, el autómata de control se encuentra en el estado *libre*, esperando a que llegue la orden de realizar una transacción.

a) *Estado libre*: Este estado se alcanza después del reset asíncrono o, también cuando, tras completar una transacción (después de la señal de STOP, que se genera en el estado *stop*) transcurre el tiempo indicado por el estándar I2C para que se pueda iniciar otra (hay un tiempo mínimo establecido que debe separar un STOP del subsiguiente START). En este estado se permanece a la espera de que se active *ini*; cuando se activa esta entrada se desencadenan las siguientes operaciones:
   - Se pasa al estado *cargar_byte*, estado en el que se ordenará (en el siguiente ciclo de reloj) la carga del registro de escritura de la línea SDA con el byte de dirección I2C (que debe mantenerse válido en la entrada *dato_in* del registro dos ciclos de reloj, el de activación de *ini* y el siguiente)
   - Se ponen a cero las salidas *fin_tx* y *tx_ok* (aunque la puesta a 0 de *tx_ok* no resulta imprescindible ya que el valor de esta señal sólo tienen validez cuando *fin_tx* vale 1)
   - Se actualiza el valor de *nWR y nWR_op* (véase el siguiente ejercicio)

Ejercicio 1

Para que el autómata de control ejecute la transferencia de un byte, debe saber si tiene que escribirlo o leerlo; como sabe, en las transacciones de escritura el master escribe todos los bytes, mientras que en las de lectura escribe el primer byte (el byte de dirección) y el resto los lee.

Cuando se activa *ini*, en la entrada *tipo_op_nW_R* se ingresa el valor del bit de menor peso del byte de dirección, que es el primer byte que se transfiere; este byte se escribe independientemente de que la transacción sea de escritura o lectura. La señal *nWR* indica si el byte que se está transfiriendo hay que escribirlo (0) o leerlo (1), la señal *nWR_op* debe guardar el tipo de operación que hay que hacer con el resto de los bytes de la transferencia, escritura (0) o lectura (1), para que el valor de esta señal se pueda transferir a *nWR* (en el estado ACK, cuando se complete la escritura del byte de dirección I2C).

Teniendo en cuenta esto, elija de entre las siguientes opciones aquella que le parezca que debe emplear para completar el código del estado *libre*:

a) *nWR <= '0';*
   nWR_op <= '0';

b) *nWR <= '0';*
   nWR_op <= tipo_op_nW_R;

c) *nWR <= tipo_op_nW_R;*
   nWR_op <= '0';

d) *nWR <= '1';*
   nWR_op <= tipo_op_nW_R;

b) Estado cargar_byte: Este estado se alcanza después de una orden de incio de una transacción o cuando al completar la transferencia de un byte la entrada *last_byte* vale 0 (esto indica que quedan bytes por transferir, que no era el último byte de la transferencia). En este estado, al que se llega antes del primero de los nueve pulsos de SCL que transcurren durante la transferencia de un byte, se realizan las siguientes acciones:

- Se pone a 0 el contador de pulsos de SCL (*cnt_pulsos_SCL*)
- Se pone a 0 la salida *fin_byte*, porque cuando se llega a este estado tras haber completado la transferencia de un byte, esta salida vale 1 y sus pulsos deben tener una duración de un ciclo de reloj
- Se ordena la transición al estado *tx_byte* (donde se transfieren los bits del byte)
- Se ordena, si procede, la carga del registro de escritura en SDA: véase el Ejercicio 2

Ejercicio 2

El registro de escritura en la línea SDA debe cargar el dato que tiene en su entrada única y exclusivamente cuando el byte que se va a transferir hay que escribirlo. El módulo de control ordena la carga activando la salida *carga_reg_out_SDA* (salida combinacional que se modela con una sentencia concurrente) en el estado *cargar_byte* si se va a escribir el byte. Teniendo en cuenta esto, elija de entre las siguientes opciones la que considere correcta para modelar el funcionamiento de esta salida en la sentencia concurrente (edite el código con ella):

a) carga_reg_out_SDA <= not nWR_op when estado = cargar_byte
else '0';

b) carga_reg_out_SDA <= nWR_op when estado = cargar_byte
else '0';

c) carga_reg_out_SDA <= not nWR when estado = cargar_byte
else '0';

d) carga_reg_out_SDA <= nWR when estado = cargar_byte
else '0';

c) *Estado tx_byte*: A este estado se llega desde *cargar*_byte, por lo que *cnt_SCL_pulsos* vale 0, y se debe permanecer en él durante los 8 pulsos de SCL en que se realiza la transferencia de un byte. La permanencia en este estado condiciona, como se verá más adelante, la generación de las señales de control (*desplaza_reg_out_SDA* y *leer_bit_SDA*) que ordenan a los registros de lectura y escritura en SDA leer o escribir, respectivamente, un bit de SDA.

Ejercicio 3

El autómata alcanza el estado *tx_byte* justo antes de que comience el primer ciclo de SCL correspondiente a la transferencia de un nuevo byte. Y debe salir de él y pasar al estado *ACK* una vez que el módulo *gen_SCL* haya generado la señal de temporización para la lectura de bits de SDA (*ena_in_SDA*) correspondiente al octavo (último) bit del byte transferido. Teniendo en cuenta esto, elija, de entre las siguientes opciones, aquella que le parezca necesaria para incrementar la cuenta de *cnt_pulsos_SCL*, sabiendo que el estado se abandona cuando *cnt_pulsos_SCL* vale 8 (edite el código con ella):

a) ena_in_SDA = '1'

b) SCL_up = '1'

c) ena_in_SDA = '1' and SCL_up = '1'

d) *ena_in_SDA = '1' or SCL_up = '1'*

Ejercicio 4

En el estado *tx_byte* hay que dar la orden de leer bits en SDA, desplazando el registro de entrada cada vez que se active la señal de temporización *ena_in_SDA* generada por *gen_SCL*. Teniendo en cuenta esto y que cuando esta señal es generada el autómata sólo puede estar en los estados *tx*_byte y ACK (que se revisará a continuación), elija, de entre las siguientes opciones, aquella que le parezca necesaria para activar la orden de desplazamiento del registro de lectura, *leer_bit_SDA*, y edite el código con ella:

a) leer_bit_SDA <= ena_in_SDA when estado = ACK
else '0';

b) leer_bit_SDA <= ena_in_SDA;
;

c) leer_bit_SDA <= ena_in_SDA when estado /= ACK
else '0';

d) leer_bit_SDA <= ena_in_SDA when estado = cargar_byte
else '0';

Ejercicio 5

En el estado *tx_byte* hay que dar la orden de escribir bits en SDA, desplazando el registro de salida cada vez que se active la señal de temporización *ena_out_SDA* generada por *gen_SCL*. Teniendo en cuenta esto, elija, de entre las siguientes opciones, aquella que le parezca necesaria para activar la orden de desplazamiento del registro de escritura (*desplaza_reg_out_SDA*) y edite el código con ella:

a) *desplaza_reg_out_SDA <= ena_out_SDA when estado = tx_byte or estado = ACK*
else '0';

b) *desplaza_reg_out_SDA <= ena_out_SDA when estado = tx_byte*
else '0';

c) *desplaza_reg_out_SDA <= ena_out_SDA when estado = cargar_byte*
else '0';

d) *desplaza_reg_out_SDA <= ena_out_SDA;*

d) *Estado ACK*: Este es el estado que tiene una operación más compleja, así que lea con atención la descripción de su funcionamiento y contrástelo con el código del modelo.

- La operación más importante que se hace en este estado es la comprobación del nivel de SDA en el bit de ACK. Al activarse la señal de muestreo (*ena_in_SDA*), el último byte transferido se dará por bueno si SDA vale 0, o si vale 1 y es una transferencia de lectura (el master indica que es el último byte leído en una lectura con un NACK); en ambos casos se activa *fin_byte*, que se desactivará en otro estado en el siguiente ciclo de reloj.

- Por otra parte, en este estado se decide si la transferencia continúa o finaliza. Finaliza en caso de no asentimiento de una escritura (hay un error y se finaliza la transferencia con *tx_ok a 0*), del asentimiento de una escritura con la entrada *last_byte* a 1 (es el último byte de una escritura, se pone *tx_ok* a 1) o del no asentimiento a una lectura con *last_byte* a 1 (es el último byte de una lectura y se finaliza con *tx_ok* a 1). La transferencia continúa únicamente si hay asentimiento y *last_byte* vale 0 (el último byte se ha transferido correctamente y no era el último.

- Si la transferencia finaliza, se va al estado *inhabilitar_SCL*. Si continúa, se inicia el ciclo de transferencia (escritura o lectura) de un nuevo byte; para ello se actualiza el tipo de operación para el siguiente byte (en las lecturas el byte que se acaba de transferir puede haber sido el de dirección I2C y hay que actualizar el valor de *nWR*) y se realiza una transición al estado *cargar_byte* (observe que los estados *cargar_byte*, *tx_byte* y *ACK* forman un bucle que permite ir transfiriendo los bytes uno a uno), en *cargar_byte* se desactiva *tx_byte*, se reinicia el contador de pulsos y, en el siguiente ciclo de reloj se pasa a *tx_byte*.

e) *Estado inhabilitar_SCL*: Este estado forma parte de la secuencia de finalización de una transición. La salida de este estado marca el momento en que se debe inhabilitar la generación del reloj SCL mediante la desactivación de la salida *ena_SCL*. La inhabilitación de *gen_SCL* (que ya se ha analizado) provoca que deje la señal SCL a nivel alto y que se generen las señales de temporización correspondientes a la señalización de STOP (*ena_stop_i2c)* y al transcurso del tiempo mínimo entre STOP y START (*ena_start_i2c)*. El estado se abandona, para alcanzar el estado *stop*, cuando la señal *SCL_up*, que se activa en los flancos de subida de SCL, identifica el último flanco de subida de SCL en la transacción.

Observe también que, tal y como se indicó en el análisis del estado *ACK*, en este estado se desactiva *fin_byte* para que su pulso activo dure únicamente un ciclo de reloj.

Ejercicio 6

La salida *ena_SCL* debe activarse en el mismo ciclo de reloj en que se carga byte de dirección I2C en el registro de escritura y debe desactivarse, tal y como se acaba de indicar, en cuanto se abandona el estado *inhabilitar_SCL*. Teniendo en cuenta esto, elija, de entre las siguientes opciones, aquella que le parezca necesaria para para asignar valor a *ena_SCL* y edite el código con ella:

a)
```
ena_SCL <= '1' when estado = cargar_byte or estado = tx_byte or estado = ACK else
           '1' when estado = inhabilitar_SCL                                 else
           '0';
```

b)
```
ena_SCL <= '1' when estado /= libre or estado /= stop     else
           '0';
```

c)
```
ena_SCL <= '1' when estado /= libre and estado /= stop else
           '0';
```

d)
```
ena_SCL <= '1' when estado = cargar_byte  and estado = tx_byte   else
           '0' when estado = inhabilitar_SCL                     else
           '0';
```

f) *Estado stop*: Este estado se alcanza después de inhabilitar la generación de SCL; después de la inhabilitación de SCL, el módulo *gen_SCL* genera dos señales de temporización:

- primero, activa *ena_stop_i2c* que señala el ciclo de reloj en que se debe generar la señal de STOP que cierra la transición I2C
- después, *ena_start_i2c* (cuando ha transcurrido el tiempo indicado en el estándar desde la señal de STOP), que indica que puede ya generarse una nueva transacción sobre el bus

Cuando la entrada *ena_start_i2c* se activa, se evoluciona al estado *libre*, en el que se espera la orden de inicio de una nueva transacción y se activa la salida *fin_tx* que indica que ha finalizado la última transacción y el master I2C está preparado para acometer otra.

Ejercicio 7

La señal de STOP se genera poniendo a nivel alto la salida SDA mientras SCL está a nivel alto al final de la transacción I2C. Para generar la señal de STOP hay que poner a 1 la salida SDA haciendo uso del preset síncrono del registro de escritura en SDA. Teniendo en cuenta esto, elija, de entre las siguientes opciones, aquella que le parezca necesaria para para activar el preset síncrono del registro y edite el código con ella:

a) *preset_SDA <= ena_stop_i2c when estado = inhabilitar_SDA*
else '0';

b) *preset_SDA <= ena_stop_i2c;*

c) *preset_SDA <= ena_stop_i2c when estado = stop*
else '0';

d) *preset_SDA <= ena_start_i2c when estado = stop*
else '0';

Reset del registro de escritura en SDA

Para terminar con la realización del módulo de control, sólo falta por analizar la actuación que realiza sobre el reset síncrono del registro de escritura en la línea SDA. Este reset se emplea para poner la línea SDA a nivel bajo en los siguientes tres casos:

- Cuando hay que generar la señal de START de inicio de una transacción; esta acción se realiza cuando, estando la interfaz preparada para realizar una transacción (en el estado *libre*), se activa *ini*.

- Después del bit de ACK del último byte de una transacción; en este caso, la puesta a 0 de SDA viene dada porque después del último flanco de súbida de SCL en una transacción (cuando se inhabilita la generación de SCL) hay que generar la señal de STOP, que consiste en que SDA pase de 0 a 1, lo que obliga a que SDA esté a 0 antes del STOP

- Cuando el master tiene que generar un ACK a un byte leído.

Ejercicio 8

En el código del módulo de control ser emplea una sentencia concurrente para generar el reset del registro de escritura en SDA:

```
reset_SDA <= ini          when estado = libre              else  --START
             ena_out_SDA  when estado = inhabilitar_SCL    else  -- Prepara STOP
             ena_out_SDA  when ACK_lectura = '1'           else  -- ACK lectura
             '0';
```

Hay una línea en la sentencia concurrente por cada una de las condiciones de reset expuestas. La primera corresponde a la condición de START, la segunda a la puesta a 0 de SDA para preparar el STOP y la tercera a la generación del ACK de lectura. Esta última condición se

apoya en la señal *ACK_lectura*, que debe valer 1 cuando estando en el estado *ACK* se está leyendo (no escribiendo) un byte y no es el último de la transacción. Teniendo en cuenta esto, elija, de entre las siguientes opciones, aquella que le parezca necesaria para manejar la señal *ACK_lectura* y edite el código con ella:

a) *ACK_lectura <= nWR and last_byte when estado = ACK*
else '0';

b) *ACK_lectura <= (not nWR and last_byte) when estado = ACK*
else '0';

c) *ACK_lectura <= (not nWR) and (not last_byte) when estado = ACK*
else '0';

d) *ACK_lectura <= (nWR and (not last_byte)) when estado = ACK*
else '0';

Una vez resuelto este ejercicio habrá completado el código del módulo de control del master I2C. Realice una compilación para comprobar que no ha cometido errores sintácticos al editar el modelo. Puede resultar conveniente que lo repase, volviendo a consultar este documento cuando le resulte necesario, para consolidar lo aprendido sobre su funcionamiento.