Apellidos, Nombre: _____

Sistema digital para el control de una barrera

Estructura de Computadores
Ingeniería Informática. Tecnologías Informáticas
Dpto. de Tecnología Electrónica
Jorge Juan Chico
Rev. 2024

1 Material

- Ordenador con el entorno ISE de Xilinx instalado.
- Placa de desarrollo FPGA Digilent Basys 2 y documentación.
- Archivos base del diseño (kit del laboratorio).
- Servomotor SG90 o similar (opcional).
- Sensor digital detector de obstáculos por infrarrojos (opcional).

2 Descripción

El objetivo de esta práctica es diseñar un sistema de control de una barrera de acceso a un aparcamiento. El sistema completo se compone de un mando de entrada (un pulsador), una barrera accionada por un servomotor y un sensor de obstáculos que detecta si hay algún vehículo bajo la barrera (Figura 1).

El sistema debe operar de la siguiente forma:

- Al accionar el mando de apertura (entrada *open*) la barrera se abrirá y permanecerá abierta 10s para permitir el paso de un vehículo. Al finalizar ese tiempo, la barrera se cerrará automáticamente.
- Si se detecta un obstáculo bajo la barrera (señal *obs*) cuando esta está abierta o cerrándose, volverá a abrirse y se cerrará tan pronto desaparezca el obstáculo.
- En caso de que se accione el mando de apertura cuando la barrera está abierta o cerrándose, permanecerá abierta 10s adicionales.

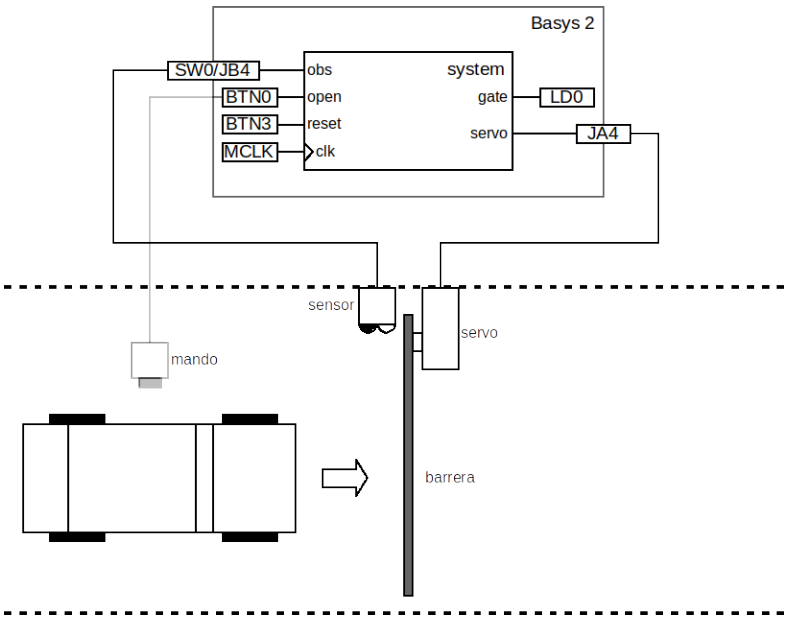

Figura 1: Esquema del sistema completo.

Internamente el sistema se compone de tres módulos interconectados, según la Figura 2.

La función de cada módulo es el siguiente:

- **timer**: es un contador que se usa como temporizador para medir el tiempo que la barrera permanece abierta. Es controlado mediante las señales de entrada *clear* (puesta a cero) y *enable* (habilitación) y activa la señal *eoc* (fin de cuenta) cuando ha pasado el tiempo establecido de 10s.
- **servo_gate**: este módulo se entrega ya diseñado y genera la señal adecuada para controlar un servomotor que mueve una barrera. El servomotor se moverá a la posición inicial cuando *gate = 0* (barrera cerrada) y a la posición final cuando *gate = 1* (barrera abierta). El movimiento de la barrera tarda 4s de extremo a extremo. Cuando finaliza el movimiento, la señal *finished* pasa a valer '1' a modo de señal de fin de carrera.
- **control**: este módulo contiene la unidad de control del sistema que se encarga de valorar las entradas *open* y *obs*, y las señales de estado de los otros módulos, *eoc* y *finished*, y generar las señales de control para los otros módulos con objeto de obtener la funcionalidad requerida.
- **system**: este módulo resulta de la interconexión de los módulos anteriores, como se muestra en la Figura 2.

Desde el punto de vista de la técnica de diseño de sistemas digitales, los módulos *timer* y *servo_gate* forman la *unidad de datos* del sistema, y el módulo *control* es la unidad de control del sistema.

El objetivo de la práctica es diseñar, implementar y probar el sistema descrito. Para ello es necesario diseñar los módulos *timer*, *control* y la conexión de todos los módulos para obtener el módulo *system*.

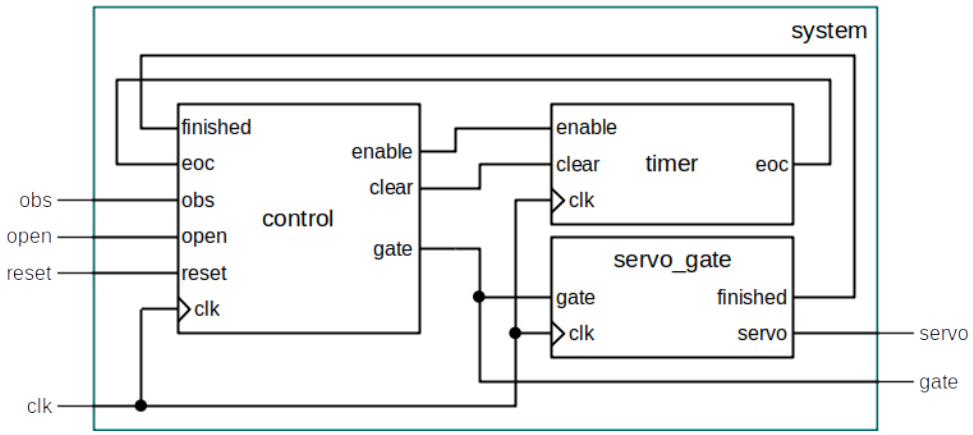

Figura 2: Sistema digital completo.

3 Resultados del aprendizaje

- Aprender a diseñar sistemas digitales sencillos basados en el modelo unidad de control-unidad de datos.
- Representar problemas secuenciales mediante máquinas de estados finitos.
- Describir máquinas de estado en Verilog.
- Simular máquinas de estados y sistemas digitales sencillos.
- Experimentar con el uso de módulos sensores y actuadores.

4 Kit del laboratorio

El kit del laboratorio se compone de los siguientes archivos:

- **control.v**: plantilla del módulo de control.
- **timer.v**: plantilla del módulo temporizador.
- **servo_gate.v**: diseño completo del módulo de generación de señales para el servo.
- **servo_gate_sim.v**: modelo simplificado del módulo de señales del servo para uso en simulación.
- **system.v**: plantilla del módulo de interconexión del sistema completo.
- **system1_tb.v**: banco de pruebas completo par la primera fase del diseño.
- **system2_tb.v**: banco de pruebas completo par la segunda fase del diseño.
- **Basys2_100_250General.ucf**: archivo de restricciones de síntesis para la placa Basys 2.

5 Trabajo previo

La práctica se desarrolla en dos fases. En la primera fase se diseña un sistema básico solo con la función de cierre temporizado (Figura 3). En la segunda fase se añade la función de detección de obstáculos y detección de fin de movimiento de la barrera.

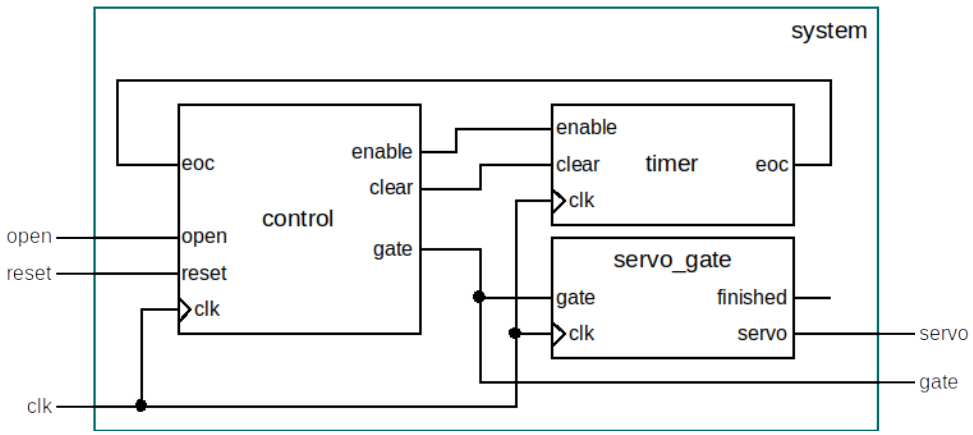

Figura 3: Módulos del sistema básico.

Como trabajo previo es posible realizar los diseños del sistema básico y el sistema completo y simularlos con cualquier simulador Verilog: EDA playground, Icarus Verilog, Xilinx ISE, etc. Algunos ejemplos más abajo se realizan con Icarus Verilog.

5.1 Fase 1. Sistema básico

El sistema básico se muestra en la Figura 3. En este sistema, la unidad de control esperará a que se active la entrada *open*, luego abrirá la barrera (*gate*=1) y activará el temporizador. Cuando el temporizador expire (*eoc*=1) la unidad de control cerrará la barrera (*gate*=0), reiniciará y detendrá el contador y volverá a esperar a que se active la entrada. Si se activa la entrada *open* mientras la barrera está abierta, se reiniciará el contador para que vuelva a esperar un tiempo de apertura completo.

1. Complete el diseño del módulo *timer* en el archivo `timer.v` según la tabla siguiente.

clear	**enable**	**count** ←
1	-	0
0	1	count + 1
0	0	count

   eoc=1 si, y sólo si, *count*=MAX_COUNT

   Observe el uso de parámetros para definir la frecuencia de reloj y el tiempo de espera, y razone cómo se ha calculado el valor máximo de cuenta del contador. De la forma en que se ha hecho, el temporizador puede adaptarse fácilmente a otras frecuencias de reloj y también a cambios en el tiempo de espera. Compruebe que no hay errores de sintaxis en su código Verilog. Por ejemplo, con Icarus Verilog:

```
$ iverilog timer.v
```

2. Dibuje la carta ASM de la unidad de control según la descripción dada al principio de este apartado (5.1). Incluya en la carta al menos dos estados (llamados, por ejemplo, CLOSED y OPEN). El primer estado debe corresponder al estado inicial cuando la barrera está cerrada.

3. Complete el diseño de la unidad de control en el archivo `control.v` a partir de la carta ASM dibujada y compruebe que el código Verilog es correcto:

```
$ iverilog control.v
```

4. Interconecte los módulos del sistema tal como indica la Fig. 3. Use el archivo `system.v` como plantilla. Observe que el módulo *servo_gate* ya está instanciado y conectado a modo de ejemplo. Observe como se pasa como parámetro la frecuencia del sistema al módulo *servo_gate*. En el sistema básico no se conecta la salida *finished* tal como se indica en la Fig. 3. Al instanciar el módulo *timer* también debe pasarle la frecuencia del sistema como parámetro.
5. Compruebe que no hay errores de sintaxis en el código Verilog del diseño completo. Por ejemplo, si usa *Icarus Verilog* ejecute lo siguiente:

```
$ iverilog system.v control.v timer.v servo_gate_sim.v
```

   El archivo `system1_tb.v` contiene un banco de pruebas ya diseñado para el sistema básico. Observe cómo el banco de pruebas instancia el módulo *system* cambiando el parámetro de frecuencia de reloj *SYS_FREQ* a 100Hz. De esta forma la simulación se realiza suponiendo que la frecuencia del sistema es mucho menor, lo que reduce notablemente el tiempo de simulación y el volumen de datos generados.

   Observe también cómo definir la frecuencia mediante parámetros facilita el control de los tiempos en los eventos de simulación. Por ejemplo, el siguiente fragmento de código hará que el simulador espere a que se produzca un número de ciclos de reloj igual a 3 veces la frecuencia de reloj, esto es, 3 segundos exactos.

```
repeat(3*SYS_FREQ) @(negedge clk);      // esperamos 3s
```

   Por otro lado, el módulo *servo_gate* no funcionará correctamente a la frecuencia de reloj reducida empleada en el banco de pruebas. Por este motivo, el módulo tiene una versión simplificada sólo para simulación en el archivo `servo_gate_sim.v`, y es la versión que debe emplearse en las simulaciones. Tenga en cuenta que la salida *servo* generada por este módulo no es real, por lo que no es necesario visualizarla.
6. Simule el sistema simplificado completo. Por ejemplo, si usa *Icarus Verilog* ejecute lo siguiente:

```
$ iverilog system1_tb.v system.v control.v timer.v servo_gate_sim.v
$ vvp a.out
VCD info: dumpfile system_tb.vcd opened for output.
```

7. Observe los posibles errores generados por el banco de pruebas y visualice las formas de onda de los resultados. Corrija los errores que pueda haber hasta obtener unos resultados correctos. Una simulación correcta debe generar unas ondas como en la Figura 4.

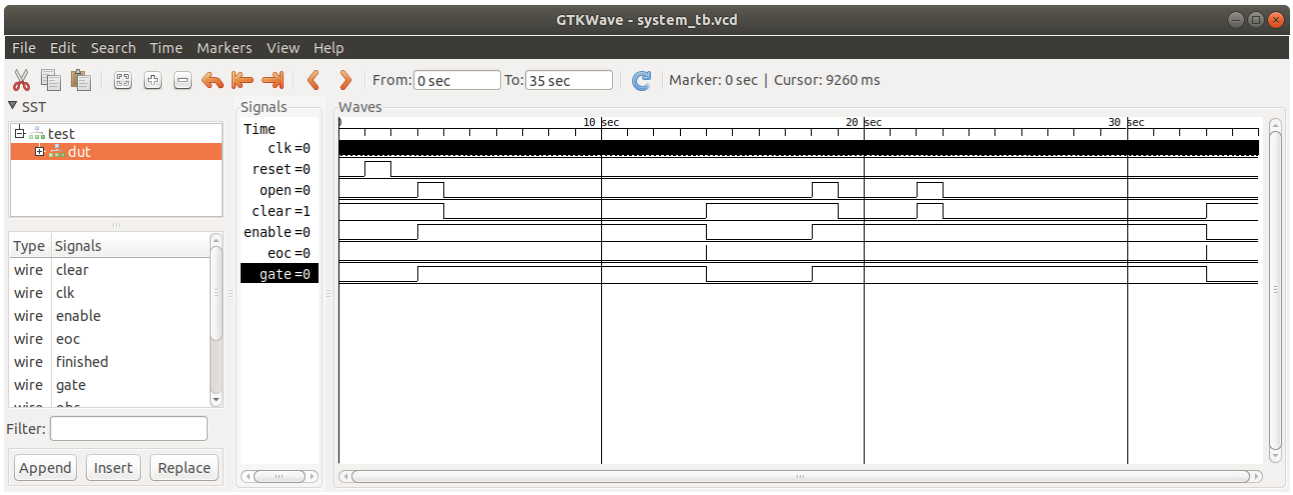

Figura 4: Simulación del sistema simplificado.

5.2 Fase 2. Sistema completo

El sistema completo añade una señal externa *obs* que se pondrá a cero cuando un sensor detecte un obstáculo que impida que se cierre la barrera. También añade una conexión de la salida *finished* del módulo *servo_gate* a la unidad de control. Esta señal se activa con valor '1' cuando la barrera llega al final de su recorrido, tanto al abrirse como al cerrarse.

De esta forma, aparte del funcionamiento del sistema básico, el sistema completo debe:

- Dejar la barrera abierta cuando se detecte un obstáculo, aunque el temporizador haya expirado, y cerrarla tan pronto desaparezca el obstáculo.
- Volver a abrir la barrera si se detecta un obstáculo mientras la misma se está cerrando pero todavía no está totalmente cerrada.

El esquema del sistema completo ya se presentó en la Figura 2.

1. Dibuje la carta ASM de la nueva unidad de control considerando las nuevas señales de entrada *obs* y *finished*. Tenga en cuenta que la señal *obs* es activa en nivel bajo y la señal *finished* activa en nivel alto.
2. Tome como partida el sistema básico y modifique la unidad de control añadiendo las nuevas señales de entrada y rehaciendo la máquina de estados a partir de la nueva carta ASM. Verifique que el código no tiene errores de sintaxis.
3. Modifique el módulo *system* con la nueva señal de entrada *obs* y las nuevas conexiones entre módulos. Compruebe que no hay errores de sintaxis en el diseño completo.
4. Simule el sistema completo con el banco de pruebas que está en el archivo `system2_tb.v`. Observe las formas de onda de salida y los posibles errores del simulador. Haga las correcciones necesarias. La salida de simulación correcta debe ser similar a la de la figura siguiente. En la figura se ha incluido la señal de estado *state*. Los valores de esta señal dependerán de la codificación de estados realizada y no tienen por qué coincidir con los de la Figura 5.

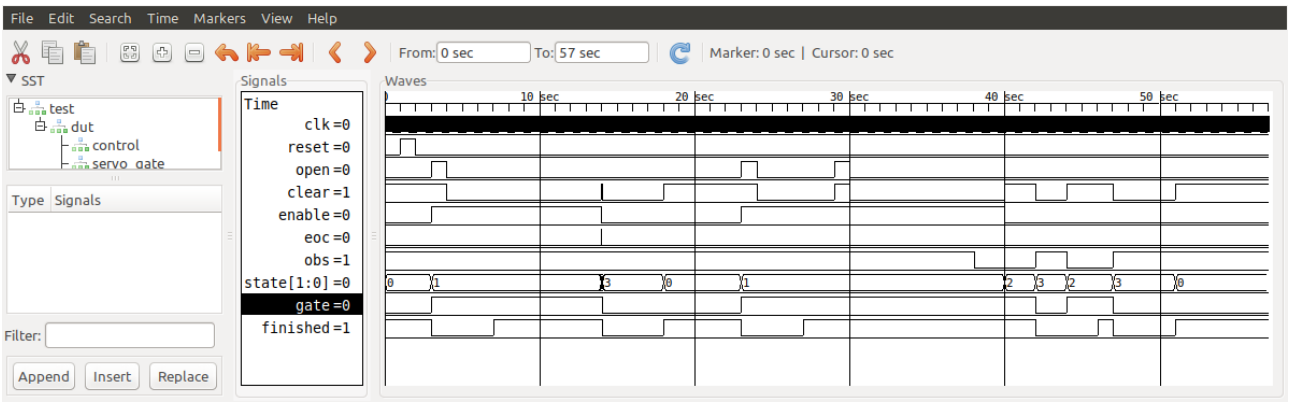

Figura 5: Simulación del sistema completo.

6 Trabajo práctico

El trabajo práctico consiste en llevar los diseños de las fases 1 y 2 a la placa de desarrollo Basys 2.

6.1 Fase 1. Sistema básico

1. Cree un proyecto en ISE con nombre **gate_control**. Use las propiedades del proyecto adecuadas para la placa Basys 2:
   - General Purpose
   - Family: Spartan 3E
   - Device: XC3S100E
   - Package: CP132
   - Speed grade: -5
2. Añada al proyecto los archivos del sistema básico `timer.v`, `control.v`, `system.v` y `system1_tb.v`. Añada además una copia del archivo de restricciones proporcionado (UCF). Los archivos `timer.v`, `control.v` y `system.v` deben completarse (si no los ha completado aún) según se explica en el apartado 5.1
3. Añada al proyecto los archivos `servo_gate.v` y `servo_gate_sim.v`. Preste atención y **vincule el primer archivo solo con el proceso de implementación y el segundo archivo sólo con el proceso de simulació**n. De esta forma las herramientas utilizarán la versión adecuada del módulo *servo_gate* según se trate de simular o sintetizar el sistema.
4. Simule el diseño con ISIM (el simulador del entorno ISE). Tenga en cuenta que, por defecto, ISIM solo simulará 1µs. Para que la simulación sea completa seleccione en el menú Simulation la opción “Run All” o pulse una vez F5, observará que la simulación ahora termina en la línea $finish. Vuelva a seleccionar la pantalla de ondas (Default.wcfg). Para hacer Zoom a la simulación completa pulse F6. Compruebe si la simulación es correcta.
5. Seleccione la vista *Implementation*. Edite el archivo UCF y realice las conexiones adecuadas de las señales del diseño con los periféricos de la placa según la siguiente tabla:

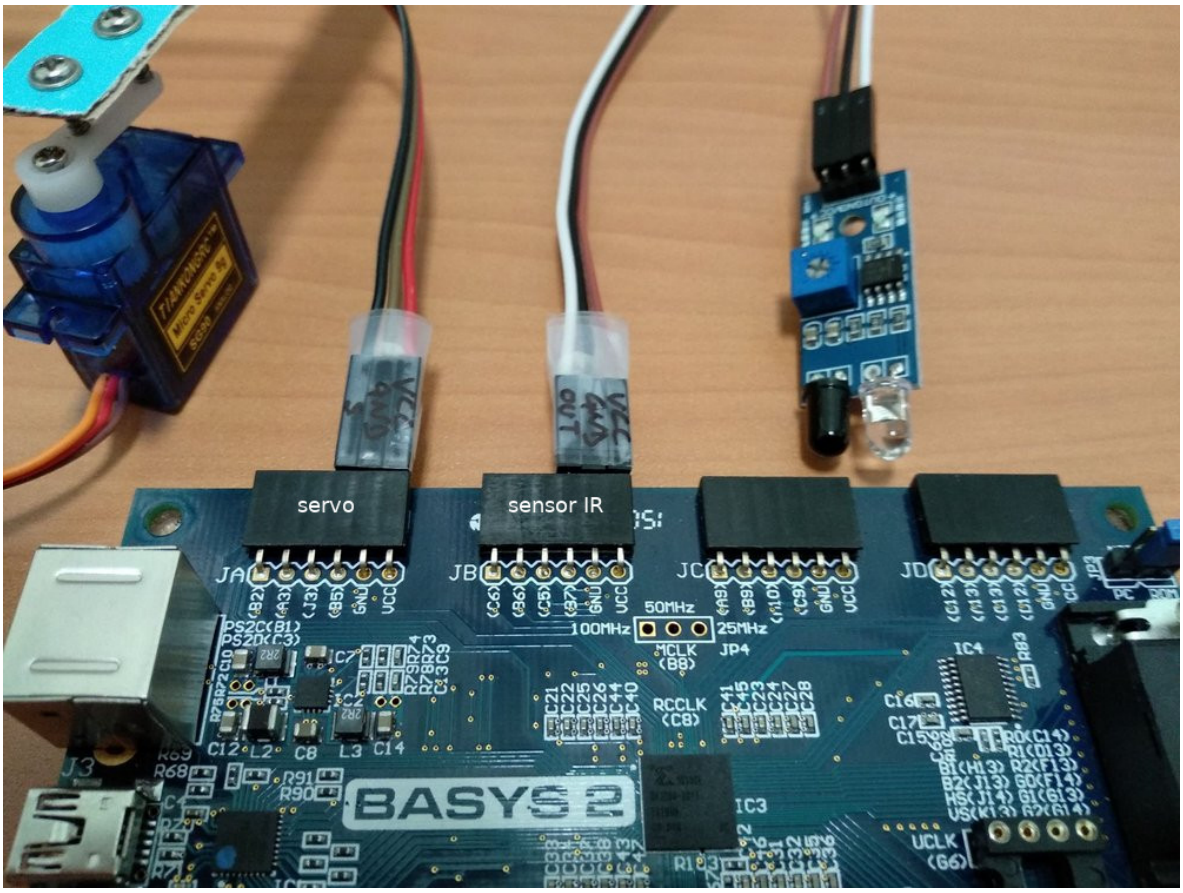

Figura 6: Conexión de servomotor y sensor IR.

Señal circuito	Periférico
clk	Reloj del sistema (MCLK)
reset	Botón 3 (BTN3)
open	Botón 0 (BTN0)
gate	LED 0 (LD0)
servo	Bit 4 del conector A (JA4)

6. Ejecute el proceso de síntesis e implementación del diseño para obtener el archivo de configuración de la FPGA (*bitstream*). Para ello seleccione el modo *Implementation* y ejecute los pasos *Synthesize – XST*, *Implement Design* y *Generate Programming File*. Haga las correcciones necesarias en el caso de que haya errores.

7. Programe el diseño en la placa. Para ello, conéctela mediante el puerto USB, asegúrese de tener accionado el conmutador de encendido y ejecute la herramienta Adept. Cargue el archivo *system.bit*, que se ha generado en el paso anterior. Compruebe que el circuito funciona correctamente tal como se indica el la descripción del sistema para la fase 1. Compruebe que la salida *gate* conectada al LED 0 de la placa se activa durante 10s al activar la entrada *open* mediante el pulsador 0. Compruebe que al activarse la señal *reset* mediante el pulsador 3 el sistema vuelve al estado inicial inmediatamente.

8. Si hay disponibles servomotores, conecte un servomotor al conector de expansión JA de la placa como en la Figura 6. Localice el conector y realice las siguientes conexiones:

Conector JA	Servomotor SG90
VCC	Cable rojo (+)
GND	Cable marrón (-)
JA4	Cable naranja (señal)

Tenga en cuenta que el terminal JA4 es el más próximo al terminal de masa (GND) en el conector.

9. Monte un brazo de madera o cartón en el eje del servomotor y compruebe que funciona a modo de barrera.

6.2 Fase 2. Sistema completo

1. Modifique el proyecto de la fase 1 añadiendo la nueva funcionalidad descrita en el apartado de 5.2.
2. Sustituya el banco de pruebas por el correspondiente a la fase 2 en el archivo `system2_tb.v`.
3. Simule el diseño con ISIM (el simulador del entorno ISE). Compruebe que la simulación es correcta.
4. Edite el archivo UCF y realice o complete las conexiones adecuadas de las señales del diseño con los periféricos de la placa según la siguiente tabla:

Señal circuito	Periférico
clk	Reloj del sistema (MCLK)
reset	Botón 3 (BTN3)
open	Botón 0 (BTN0)
obs	Conmutador 0 (SW0)
gate	LED 0 (LD0)
servo	Bit 4 del conector A (JA4)

Observe como el conmutador 0 (SW0) emula la entrada del sensor de obstáculos.

5. Ejecute el proceso de síntesis e implementación del diseño para obtener el archivo de configuración de la FPGA (*bitstream*). Haga las correcciones necesarias en el caso de que haya errores.
6. Programe el diseño en la placa y compruebe que el circuito funciona correctamente tal como se indica el la descripción del sistema para la fase 2. Compruebe el funcionamiento de la entrada *obs* tanto cuando la barrera está abierta como cuando está cerrándose.
7. Si hay disponibles sensores digitales de infrarrojos, conecte un sensor al conector de expansión JB de la placa como en la Figura 6. Localice el conector y realice las siguientes conexiones:

Conector JB	Sensor
VCC	VCC

Conector JB	Sensor
GND	GND
JB4	OUT

Tenga en cuenta que el terminal JB4 es el más próximo al terminal de masa (GND) en el conector.

8. Gire el potenciómetro de calibración del sensor hasta conseguir que el sensor se active al colocar un obstáculo frente a él, como un trozo de papel blanco, pero permanezca inactivo cuando no haya obstáculo.
9. Modifique el archivo UCF para asociar la entrada *obs* a la señal JB4 de la placa. Implemente de nuevo el diseño en la placa y compruebe la operación del sistema.