(202110-IELE2210-SED) PROYECTO FROGGER. Grupo 13 Sec1

PROYECTO FROGGER - SECCIÓN 1

Grupo 13

Descripción:

Este artículo presenta los resultados del trabajo del grupo. **Leer más**.

Autores:

Montes Buitrago, Tony Santiago (t.montes@uniandes.edu.co (mailto:t.montes@uniandes.edu.co))

Ortegon Chacon, Mateo Felipe (mf.ortegonc@uniandes.edu.co (mailto:mf.ortegonc@uniandes.edu.co))

Pasachoa Aranguren, Alejandro (a.pasachoa@uniandes.edu.co (mailto:a.pasachoa@uniandes.edu.co))

Contents

ESPECIFICACIONES

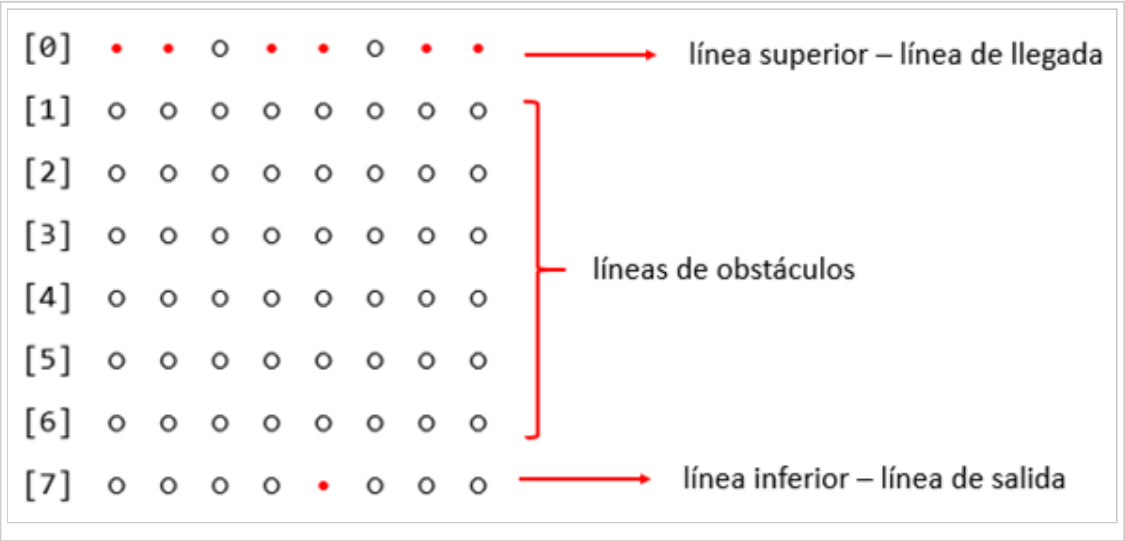
Descripción del producto

[Collapse]

PRODUCTO DE CALIDAD: El estudiante identifica las especificaciones y restricciones (enunciadas y no enunciadas del producto) para comprender y garantizar e requerimiento solicitado.

- a. La descripción del producto está redactada en palabras propias de los estudiantes, organizada lógica y claramente.
- b. Las especificaciones y restricciones responden totalmente al producto solicitado, demostrando originalidad y aportes propios. Se propone un macroalgoritmo de operación/funcionamiento del producto que responde totalmente al producto solicitado.
- c. La búsqueda e identificación de características similares de proyectos base utilizados como referencia es clara justificando su utilidad para el desarrollo del proyecto.
- d. El lenguaje disciplinar es preciso y adecuado, las frases son gramaticalmente correctas y no hay errores ortográficos.

En la implementación del juego se maneja un único personaje (frogger) y además del personaje principal existen ciertos obstáculos (carros) que el personaje debe evitar, y entradas a las que debe llegar. El juego se diseña sobre una matriz led de 8x8, y maneja 6 señales básicas de control: Reset, Start, Up, Down, Left, Right. Las señales Up, Down, Left, Right permiten mover a frogger por la matriz de leds, mientras que Reset permite reiniciar todo el juego en cualquier momento y Start permite iniciar el juego o pasar de nivel.



En la matriz, el personaje inicia el juego en la línea inferior [7] en la posición que se muestra en la Figura 1, y a lo largo de las líneas [1-6] empiezan a moverse los obstáculos, tanto de izquierda a derecha como al revés. El juego tiene 4 niveles y el objetivo del juego es llevar a frogger a llenar las dos entradas superiores que se muestran en la Figura 1 hasta llenar la línea superior [0]. Cada vez que frogger llegue a una de las casas el led se encenderá indicando que ya se llenó dicha casa. Una vez se llenen las dos casas, se apagarán todos los leds y se encenderán indicando el número del nivel al que pasa; si es el último nivel, se muestra una "W" que indica que ha ganado el juego. Para continuar con el juego luego de mostrar el número, el jugador debe presionar el botón Start, y se iniciará una pantalla idéntica, con las casas vacías, pero cuyos carros tienen diferentes características.

Las características de los obstáculos o carros a lo largo de los niveles se definieron de la siguiente forma:

Nivel 1.

- 3 carros distribuidos uno en cada línea: [1], [3], [5]
- Longitud de 2 leds cada carro
- A una velocidad de 0.75 led/s por las líneas

Nivel 2.

- 4 carros distribuidos uno en cada línea: [1], [2], [4], [6]
- Longitud de 3 leds cada carro
- A una velocidad de 1.5 led/s por las líneas

Nivel 3.

- 5 carros distribuidos uno en cada línea: [1], [2], [4], [5], [6]
- Longitud de 3 leds cada carro
- A una velocidad de 1.5 led/s por las líneas

Nivel 4.

- 8 carros distribuidos uno en las líneas: [2], [3], [5], [6] y dos en las líneas [1], [4]

- Longitud de 2 leds cada carro
- A una velocidad de 3 led/s por las líneas

Dichas son las características de los obstáculos, sin embargo, estos se generan de forma aleatoria, es decir que no tienen un orden de aparición. Esto con el fin de evitar que sea totalmente simétrico y predecible.

El personaje frogger tiene 3 vidas en cada nivel, esto quiere decir que puede chocar con un obstáculo hasta 3 veces en un mismo nivel. Cada vez que choque se apagarán todos los leds y se mostrará un mensaje de un signo – más el número de vidas restantes, se mantendrá el mensaje hasta que el personaje presione el botón start, luego se quitará y se iniciará la pantalla en el nivel en el que iba; si es cero, se mostrará "0" y se reiniciará el juego desde el nivel 1.

Restricciones:

- El frogger no podrá chocarse con ninguno de los obstáculos, los cuales serán los carros ya que si lo hace el juego automáticamente reinicia el nivel.
- Sino se presiona el botón START entonces no podrá empezar el nivel que quiere jugar.
- No se podra pasar de nivel, si uno de los dos froggers no llegan a la casa. Si uno de los dos llega a ser atropellado entonces el nivel se repite.
- No deben de haber más de dos casas cuando lleguen los froggers.
- Al iniciar el juego, el frogger no podrá estar en otra posicion que no sea la del medio de la fila 1.
- El juego no será multijugador.

Referencias: Basamos nuestra implementación de esta versión del frogger en los proyectos BASE y EJEMPLO brindados por el curso dde Sistemas Electrónicos Digitales.

Cronograma de Actividades

[Collapse]

PRODUCTO DE CALIDAD: Presenta un cronograma de actividades y la distribución de tareas entre los integrantes del grupo. Se deben consignar los logros y problemas (RESUMIDOS) presentados día a día en el desarrollo del proyecto. Se debe presentar como una tabla muy resumida.

Parrafo / Tabla: Cronograma de actividades y la distribución de tareas.

Fecha	Actividad			Problemas	Logros
	Principal	Específica	Encargado		
Lunes 12/04/2021	Primera Entrega	Descripción del producto	Tony	♦ Dado que al momento no conocemos mucho sobre cómo interpretar todo el código y el lenguaje de descripción de hardware, no tenemos muy claros los límites sobre qué es posible y está realmente al alcance de nuestro conocimiento, y qué no.	♠ Definir completamente la implementación que esperamos lograr para el juego.
		Cronograma de actividades	Tony		♠ Identificar las restricciones de nuestra implementación.
		Diagrama de caja negra	Mateo		♠ Organizar las próximas reuniones del grupo.
		Macroalgoritmo general de la solución	Todos		♠ Realizar el macroalgoritmo de nuestra propuesta para su futura implementación.
		Pasar datos del Word a la WIKI	Alejandro		
Domingo 18/04/2021	Segunda Entrega	Arquitectura del Sistema (Parte Circuital): Diagrama de bloques, Señales, e	Mateo	♦ Dado que desconocíamos totalmente el funcionamiento de cada componente en el proyecto de ejemplo, tuvimos que analizar a detalle el código para comprender.	♠ Definir los componentes que vamos a utilizar, para facilitar la implementación en
		Arquitectura del Sistema (Parte Hardware): Diagrama de bloques, Señales e	Tony y Mateo		♠ Comprender a más detalle la función de cada componente definido.
		Bloque X	Alejandro		
Lunes 19/04/2021	Tercera Entrega	Memorias de Cálculo	Tony	♦ Comprender el funcionamiento del preescaler.	♠ Definir correctamenta las diferentes velocidades propuestas para cada nivel.
		Definición de vectores de prueba y simulaciones parciales de respaldo	Mateo y Alejandro		♠ Definir casos de prueba útiles para verificar los detalles importantes del proyecto.
		Indicadores de utilización de recursos y rendimiento	Tony		
Martes 27/04/2021	Entrega Final	Descripción de lenguajes HW y SW (Códigos Fuente)	Tony	♦ Las principales complicaciones de este proyecto se dieron en la entrega final, en la que debíamos implementar el código, dado que hasta esta etapa comenzamos a escribir nuestro propio código. Saber qué código reutilizar sobre el código ejemplo fue sencillo, gracias a la segunda entrega, sin embargo implementar nuestros propios componentes tales como RegLIVES, RegLEVEL, COLLISIONCOMPARATOR, y demás, fue complejo dado que estabamos acostumbrados a escribir lenguaje de software.	♠ Lograr la funcionalidad total de nuestra implementación propuesta.
		Dispositivo 1	Todos		
		Dispositivo 2	Todos		
		Funcionalidad global del sistema	Todos		
		Vídeos y fotos de demostración del prototipo final	Todos		
		Funcionalidad modular	-		
		Vídeos y fotos de demostración de módulos del sistema	Todos		
		Resultados y lecciones aprendidas	Alejandro		
		Dispositivos Hardware	Tony		
		Herramientas Software	Tony		
		Bibliografía	Mateo		

Cronograma de Actividades

Diagrama de caja negra

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de realizar un diagrama de caja negra para identificar señales de entrada/salida del producto.

- a. El diagrama de caja negra muestra todas las señales de entrada y salida con sus correspondientes nombres estructurados (In/Out) y tamaños (bit/bus).
- b. El diagrama de caja negra relaciona dicho componente con el diagrama de caracterización (test-bench).

*Parrafo 1:*Se propone el siguiente diagrama de caja negra en donde se plantean 7 señales de entrada del sistema (INPUTS) y 3 señales de salida (OUTPUTS)

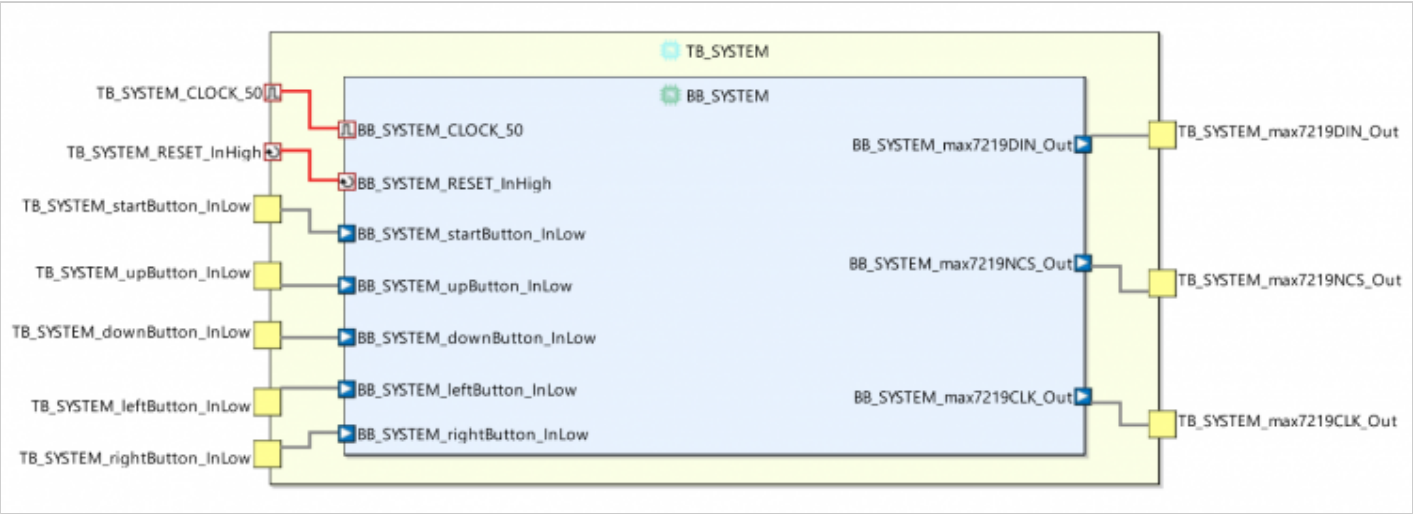


Diagrama de caja negra

Imagen 1: Imagen de diagrama de caja negra.

- Señales de entrada del sistema (INPUTS)
  - *Señal 1:* BB_SYSTEM_CLOCK_50
  - *Señal 2:* BB_SYSTEM_RESET_InHigh
  - *Señal 3:* BB_SYSTEM_startButton_InLow
  - *Señal 4:* BB_SYSTEM_upButton_InLow
  - *Señal 5:* BB_SYSTEM_downButton_InLow
  - *Señal 7:* BB_SYSTEM_leftButton_InLow
  - *Señal 6:* BB_SYSTEM_rightButton_InLow
- Señales de Salida (OUTPUTS)
  - *Señal 1:* BB_SYSTEM_max7219DIN_Out
  - *Señal 2:* BB_SYSTEM_max7219NCS_Out
  - *Señal 3:* BB_SYSTEM_max7219CLK_Out

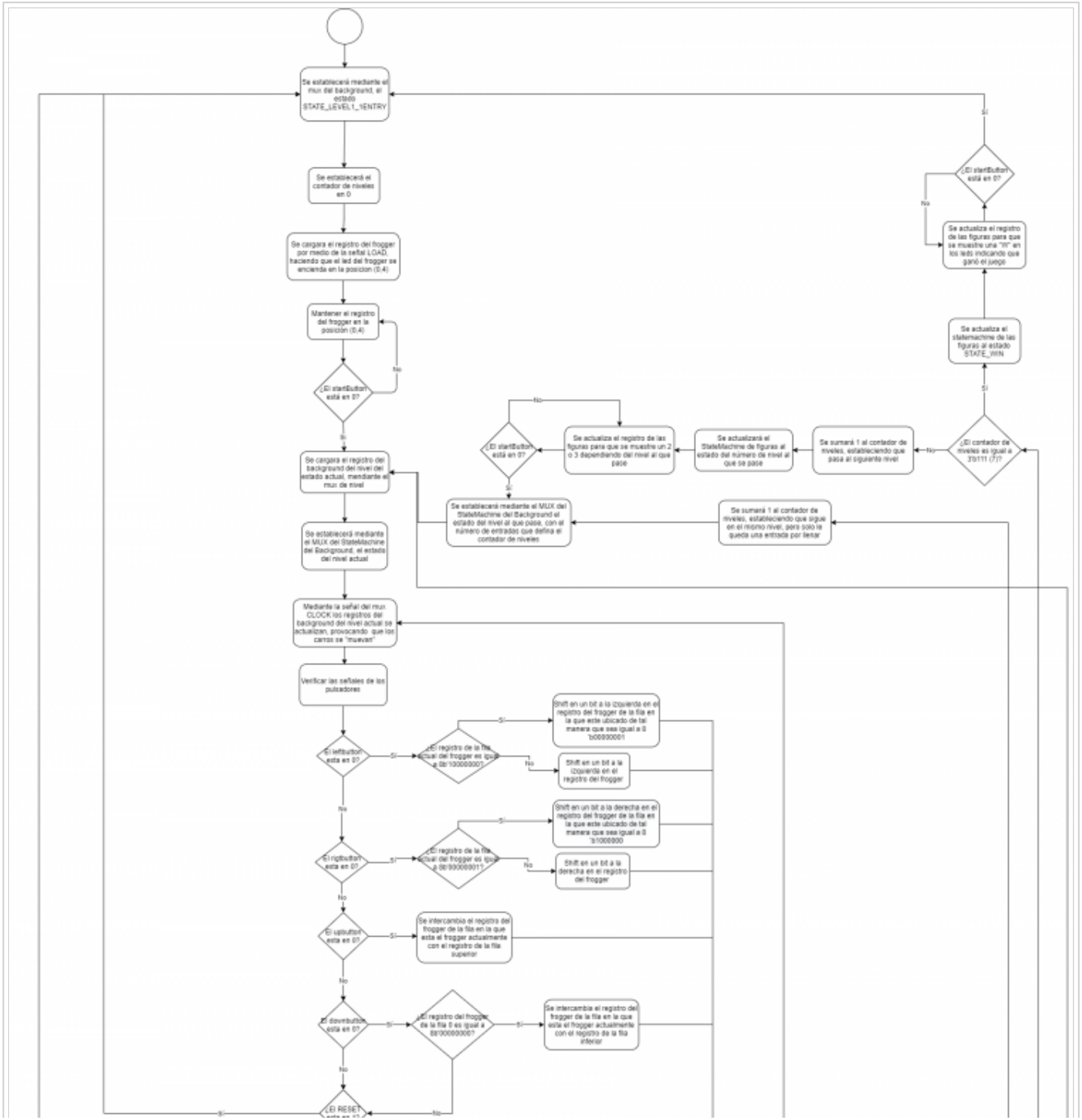
Macro-algoritmo general de solución

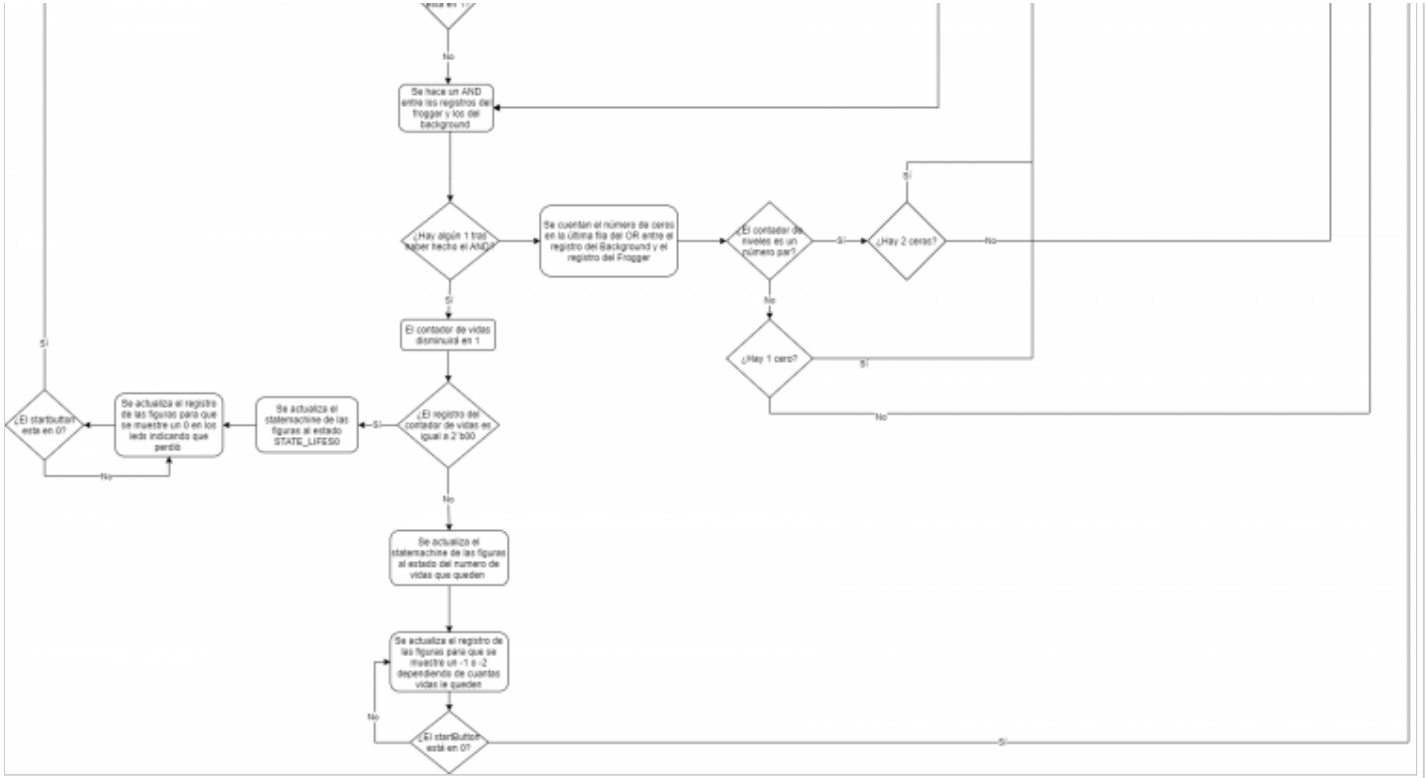
[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de proponer macroalgoritmos descomponiendo el problema en un conjunto de pasos para detallar la funcionalidad esperada del producto.

- a. El macro-algoritmo de solución describe correctamente la funcionalidad del producto y se representa adecuadamente con una explicación detallada en donde cada paso es menos complejo que el producto solicitado.
- b. En cada paso del macro-algoritmo se detalla correctamente un pseudo-algoritmo que describe una posible implementación.

Descripción: Este macroalgoritmo define la funcionalidad general de nuestra implementación propuesta.


Se establecerá mediante el mux del background, el estado STATE_LEVEL1_1ENTRY
Se establecerá el contador de niveles en 0
Se cargara el registro del frogger por medio de la señal LOAD, haciendo que el led del frogger se encienda en la posicion (0,4)
Mantener el registro del frogger en la posición (0,4)
¿El startButton está en 0?
No
Si
Se cargara el registro del background del nivel del estado actual, mendiante el mux de nivel
Se establecerá mediante el MUX del StateMachine del Background, el estado del nivel actual
Mediante la señal del mux CLOCK los registros del background del nivel actual se actualizan, provocando que los carros se "muevan"
Verificar las señales de los pulsadores
El leftbutton esta en 0?
¿El registro de la fila actual del frogger es igual a 8b'10000001'?
Shift en un bit a la izquierda en el registro del frogger de la fila en la que este ubicado de tal manera que sea igual a 8 'b00000001
Shift en un bit a la izquierda en el registro del frogger
El rigtbuttun esta en 0?
¿El registro de la fila actual del frogger es igual a 8b'00000011'?
Shift en un bit a la derecha en el registro del frogger de la fila en la que este ubicado de tal manera que sea igual a 8 'b10000000
Shift en un bit a la derecha en el registro del frogger
El upbuttun esta en 0?
Se intercambia el registro del frogger de la fila en la que esta el frogger actualmente con el registro de la fila superior
El downbuttun esta en 0?
¿El registro del frogger de la fila 0 es igual a 8b'00000000'?
Se intercambia el registro del frogger de la fila en la que esta el frogger actualmente con el registro de la fila inferior
¿El RESET
¿El startButton está en 0?
Se actualiza el registro de las figuras para que se muestre un 2 o 3 dependiendo del nivel al que pase
Se actualizará el StateMachine de figuras al estado del número de nivel al que se pase
Se sumará 1 al contador de niveles, estableciendo que pasa al siguiente nivel
¿El contador de niveles es igual a 3'b111 (7)?
Se actualiza el statemachine de las figuras al estado STATE_WIN
Se actualiza el registro de las figuras para que se muestre una "W" en los leds indicando que ganó el juego
¿El startButton está en 0?
Se sumará 1 al contador de niveles, estableciendo que sigue en el mismo nivel, pero solo le queda una entrada por llenar
Se establecerá mediante el MUX del StateMachine del Background el estado del nivel al que pase, con el número de entradas que defina el contador de niveles



Macroalgoritmo General

Imagen 1: Pasos del macroalgoritmo general de solución.

Arq del Sistema (PARTE CIRCUITAL): Diagrama de Bloques, Señales e Inter-conexiones

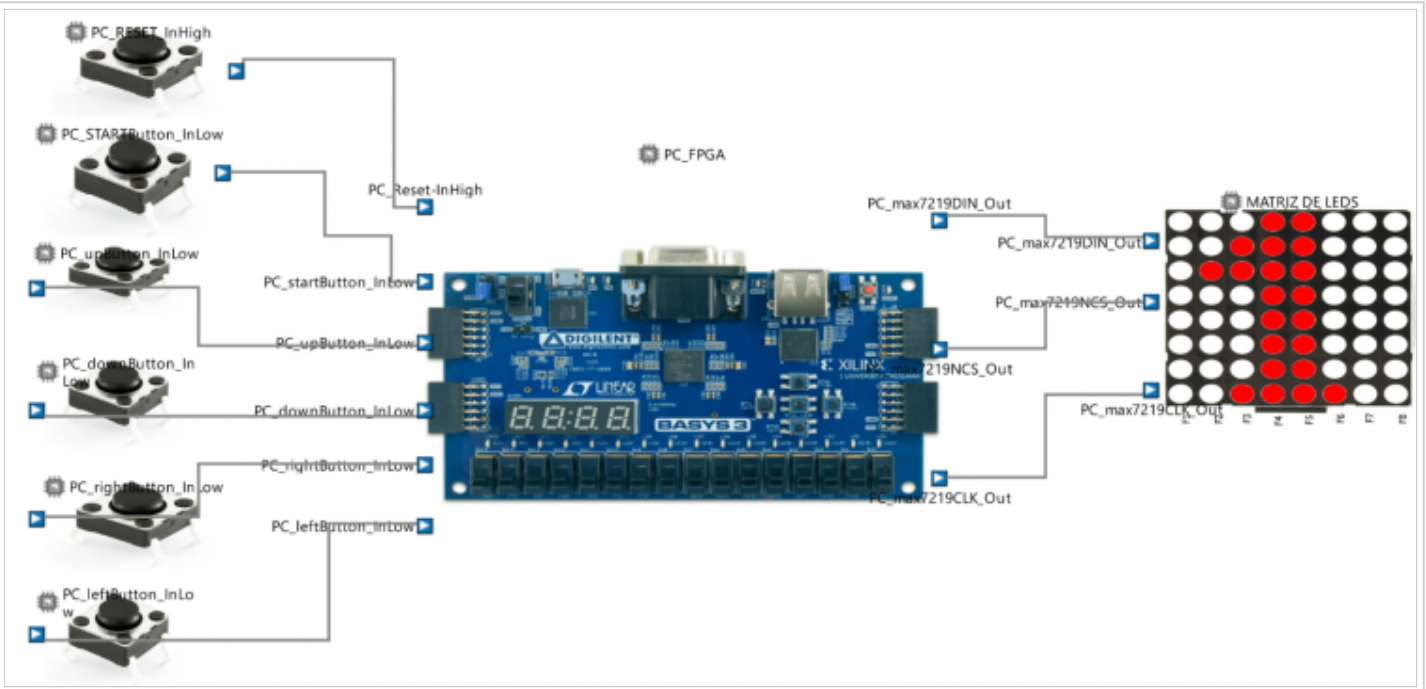
[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de identificar los componentes circuitales constitutivos del producto y su forma de comunicación para comprender y evidenciar la estructura del requerimiento solicitado en términos circuitales.

- a. Se muestran todas las señales de entrada, salida e internas con sus correspondientes nombres estructurados (In/Out) y tamaños (Bit/Bus) para todos los componentes que forman el sistema.
- b. Se presenta una descripción (qué es y cómo funciona) de cada uno de los componentes físicos del producto solicitado, describiendo las conexiones físicas a realizar.

En esta imagen podemos observar los componentes físicos de nuestro circuito, estos son los pulsadores, la tarjeta FPGA y la matriz de LEDS. En este caso el componente mas importante será la tarjeta FPGA debido al hecho que esta es la que programaremos en quartus para que nuestro frogger y nuestro background se comporten de la manera que queramos. Las señales de entrada y salida son las mismas que las del Black Box del circuito, la razón de esto es que para que nuestra tarjeta pueda coordinarse con los demas elementos debe estar conectado a la señal de cada elemento de las piezas que queremos controlar.

Imagen 1: Imagen de diagrama circuital.



Arquitectura del sistema parte circuital.

Descripción de cada componente.

- Botones: Estos componentes son los que le permiten a la persona que esté jugando que manipule el movimiento del frogger y realice otras acciones tales como iniciar el juego (start) o reiniciarlo (reset).
- FPGA: Es la tarjeta que programamos en lenguaje de hardware para que realice ciertas operaciones sobre la tarjeta dependiendo de sus máquinas de estado y de los botones oprimidos
- Matriz LED 8x8: Este componente nos permite visualizar de una mejor manera el juego que se programó en la FPGA.

Arq del Sistema (PARTE HARDWARE): Diagrama de Componentes, Señales e Inter-conexiones

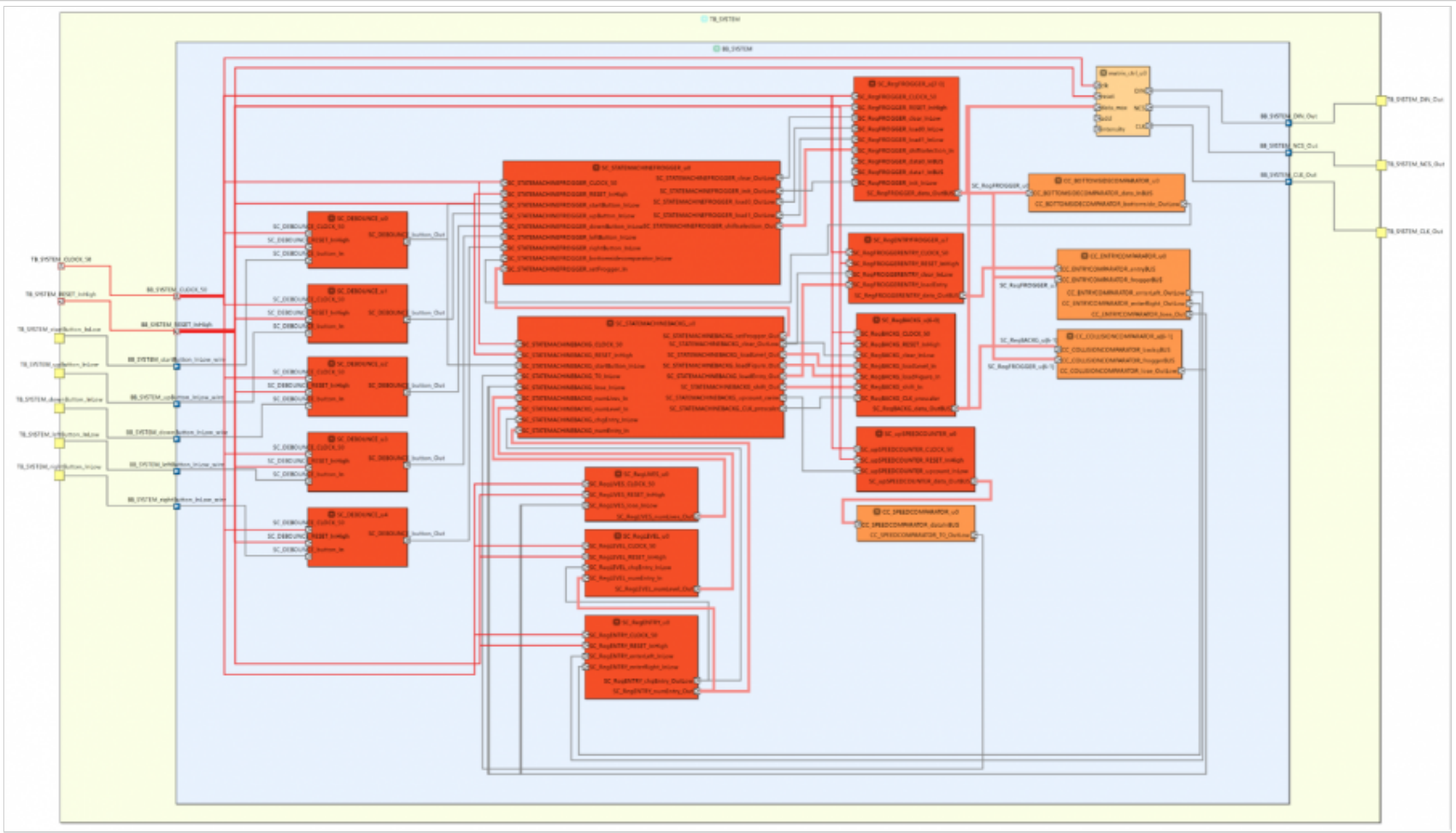
[Collapse]

PRODUCTO DE CALIDAD: : El estudiante es capaz de proponer una arquitectura estructurada usando componentes funcionales (combinacionales y secuenciales) y su forma de comunicación para comprender y evidenciar la estructura y funcionalidad del requerimiento solicitado en términos de componentes.

- a. Se muestran todas las señales de entrada/salida e internas con sus correspondientes nombres estructurados (In/Out) y tamaños (Bit/Bus) para todos los componentes de sistema.
- b. La arquitectura corresponde a una solución eficiente en cuanto a recursos, número de componentes y elementos y algoritmo de solución. Los componentes internos son menos complejos que los de mayor jerarquía.
- c. Se diferencian los componentes combinacionales y secuenciales al igual que las señales de reloj y reset que deben ser compartidas por todos los bloques secuenciales. La arquitectura presenta un módulo de control principal (máquina de estados finita).
- d. Se presenta una descripción (qué es y cómo funciona) de cada uno de los componentes constitutivos del producto solicitado, describiendo sus señales.

Descripción: A continuación se presenta el diagrama Hardware realizado para la solución propuesta, sobre el cual se basará el código que desarrollaremos en Verilog proximamente, y sobre el cual se fundamenta todo el funcionamiento del programa.

Imagen 1: Imagen de diagrama de componentes, señales e inter-conexiones.



Pie de imagen

Diagrama Hardware [1] (https://drive.google.com/file/d/1Bz7nEKTon4D4OWPDuGtdqPRzCUmzjyGc/view?usp=sharing)

COMPONENTE SECUENCIAL DEBOUNCE

Descripción Funcionalidad: Las 5 instancias del bloque DEBOUNCE se utilizan para rectificar las señales que se reciben de los pulsadores, con el fin de generar una señal continua en el diagrama de tiempos

Descripción Algoritmo: El algoritmo del bloque secuencial DEBOUNCE no se ha estudiado en profundidad en el curso.

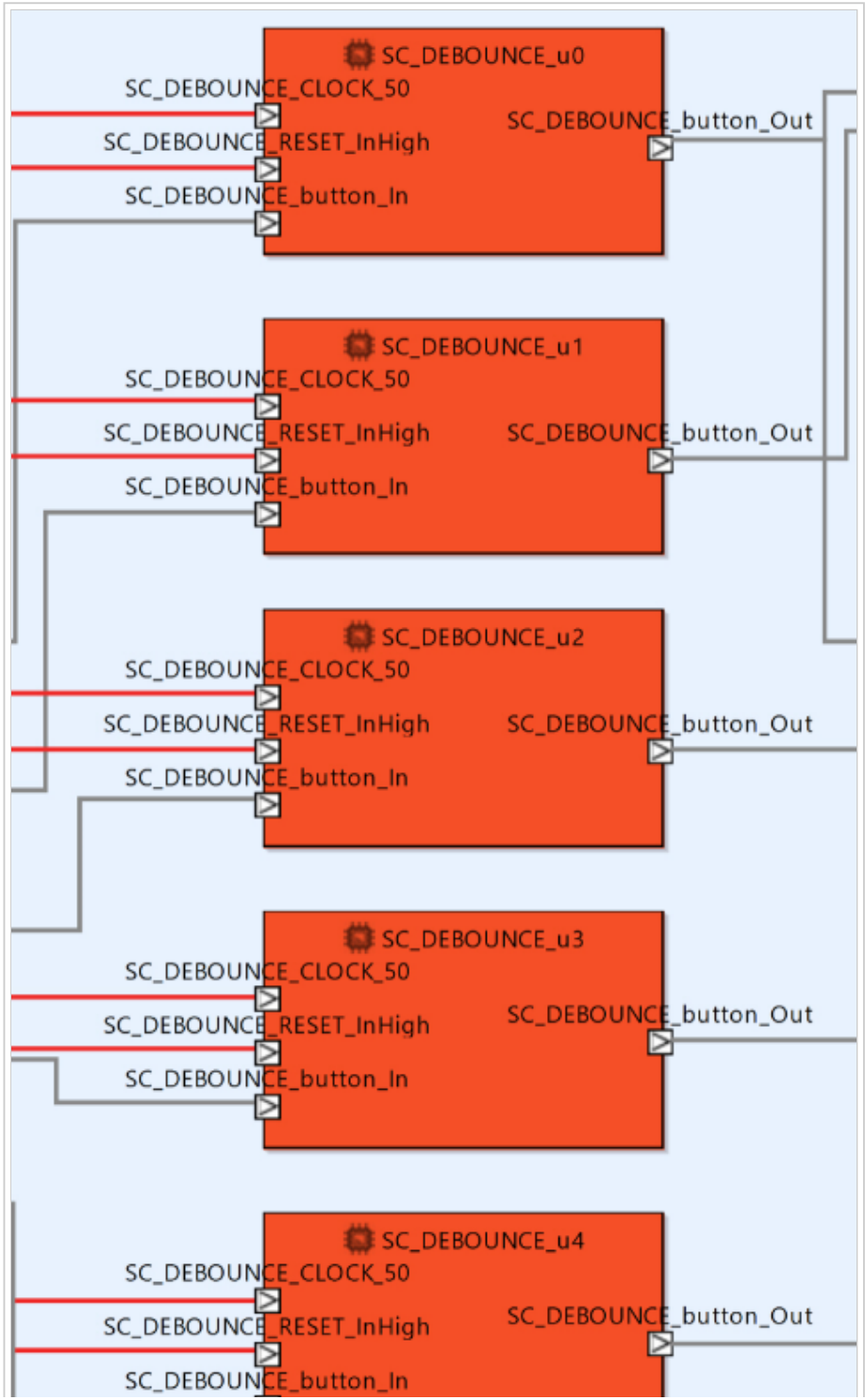

SC_DEBOUNCE_u0
SC_DEBOUNCE_CLOCK_50
SC_DEBOUNCE_RESET_InHigh
SC_DEBOUNCE_button_In
SC_DEBOUNCE_button_Out
SC_DEBOUNCE_u1
SC_DEBOUNCE_CLOCK_50
SC_DEBOUNCE_RESET_InHigh
SC_DEBOUNCE_button_In
SC_DEBOUNCE_button_Out
SC_DEBOUNCE_u2
SC_DEBOUNCE_CLOCK_50
SC_DEBOUNCE_RESET_InHigh
SC_DEBOUNCE_button_In
SC_DEBOUNCE_button_Out
SC_DEBOUNCE_u3
SC_DEBOUNCE_CLOCK_50
SC_DEBOUNCE_RESET_InHigh
SC_DEBOUNCE_button_In
SC_DEBOUNCE_button_Out
SC_DEBOUNCE_u4
SC_DEBOUNCE_CLOCK_50
SC_DEBOUNCE_RESET_InHigh
SC_DEBOUNCE_button_In
SC_DEBOUNCE_button_Out

Imagen de diagrama de caja negra, bloques secuenciales DEBOUNCE

- Entradas
  - *SC_DEBOUNCE1_CLOCK_50 :* Entrada de un bit que funciona como el reloj del sistema, permitiendo coordinar las acciones del sistema.
  - *SC_DEBOUNCE1_RESET_InHigh* Señal activa alta que da la indicación al sistema cuando se oprime el pulsador reset a fin de resetear el juego.
  - *SC_DEBOUNCE1_button_In:* Señal directa del pulsador.
- Salidas
  - *SC_DEBOUNCE1_button_Out:* Señal del pulsador rectificada.

COMPONENTE SECUENCIAL STATEMACHINEFROGGER

*Descripción Funcionalidad:*Este componente tiene como función evaluar el comportamiento de las entradas tales como reset, clock, starButton, upButton, downButton, lefttButton, rightButton, y bottomsidecomparator para generar las salidas para configurar posteriormente los registros de manera esperada a como funcionaría en el juego.

Descripción Algoritmo: En el STATEMACHINEFROGGER se manejan dos MUX, uno de estados, y uno de salidas. El MUX de estados, se encarga de definir dependiendo de las entradas definidas más abajo, cuál es el estado futuro del punto frogger en la matriz. Por otro lado, el MUX de salidas depende del estado presente en el que se encuentre el STATEMACHINE, y en base a este define las salidas que tendrá, que se definen más abajo.

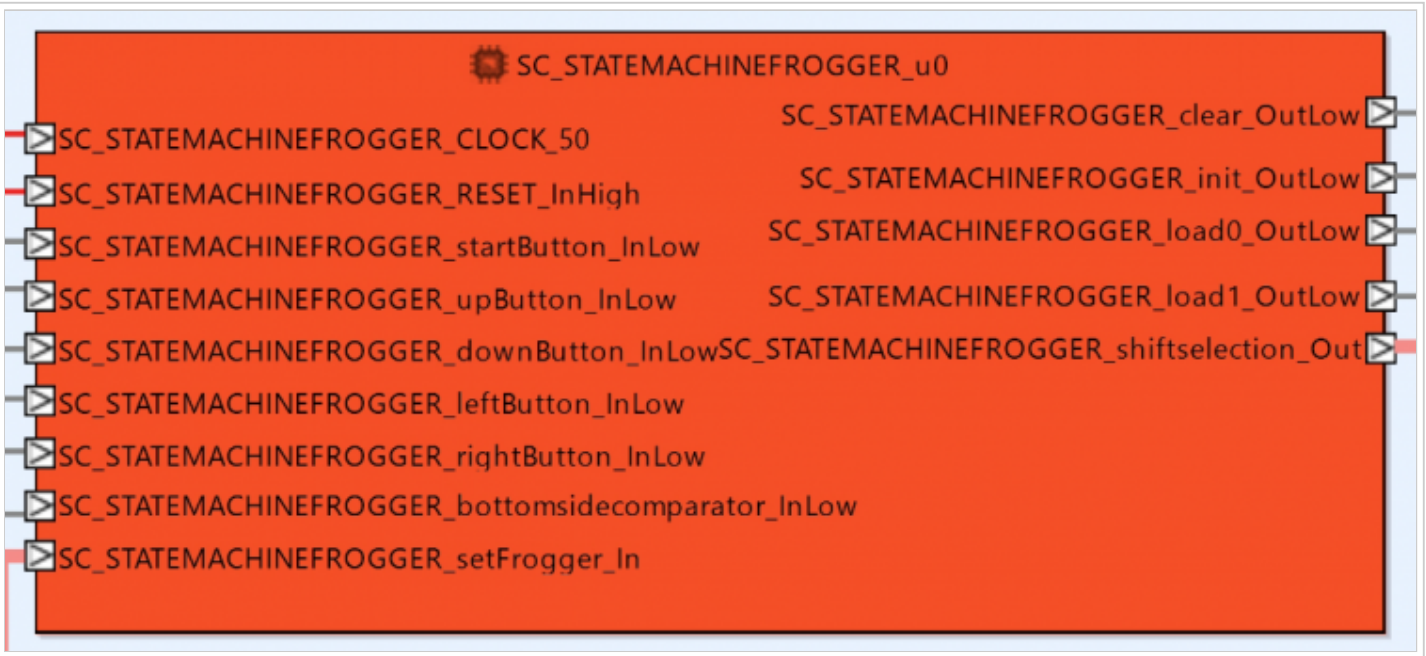


Imagen de diagrama de caja negra, bloque secuencial STATEMACHINEFROGGER

- Entradas
  - *SC_STATEMACHINEFROGGER_CLOCK_50:* Esta señal nos permitirá coordinar las acciones que queremos que tenga nuestro circuito, en este caso será coordinar los movimientos del frogger.
  - *SC_STATEMACHINEFROGGER_RESET_InHigh:* Esta señal indica que el jugador ha oprimido el botón RESET con lo cual se deben volver al estado inicial.
  - *SC_STATEMACHINEFROGGER_upButton_InLow:* Esta es la entrada de la señal del botón que moverá al frogger hacia arriba; si este botón es presionado, se enviará la señal load1 a todos los registros del frogger, haciendo que este suba. El STATEMACHINE pasará al estado: "STATE_UP" .

- *SC_STATEMACHINEFROGGER_downButton_InLow:* Esta es la entrada de la señal del botón que moverá al frogger hacia abajo; si este botón es presionado, se enviará la señal load0 a todos los registros del frogger (después de verificar con el bottomsidecomparator que esto sea posible), haciendo que este baje. El STATEMACHINE pasará al estado: "STATE_DOWN".
- *SC_STATEMACHINEFROGGER_leftButton_InLow:* Esta es la entrada de la señal del botón que moverá al frogger hacia la izquierda; si este botón es presionado, se enviará mediante la señal shiftselection, el bus 2'b01 indicando que el registro del frogger debe hacer un shift a la izquierda. El STATEMACHINE pasará al estado: "STATE_LEFT".
- *SC_STATEMACHINEFROGGER_rightButton_InLow:* Esta es la entrada de la señal del botón que moverá al frogger hacia la derecha; si este botón es presionado, se enviará mediante la señal shiftselection, el bus 2'b10 indicando que el registro del frogger debe hacer un shift a la derecha. El STATEMACHINE pasará al estado: "STATE_RIGHT".
- *SC_STATEMACHINEFROGGER_bottomsidecomparator_InLow:* Esta señal permite saber si el frogger se encuentra en la fila inferior; si es así, no está permitido que el frogger haga un desplazamiento hacia abajo (downButton).
- *[1:0] SC_STATEMACHINEFROGGER_setFrogger_In:* Esta señal es enviada por el STATEMACHINEBACKG y es la que le permite controlar el frogger mediante los sigueintes 4 casos:
  - 2'b00 -> Permite al frogger estar "libre", esto significa que se puede mover con los botones y las entradas left, right, up, down.
  - 2'b01 -> Fija al frogger en la posición actual, esto impide que esté atento a las señales de los botones.
  - 2'b10 -> Envía el frogger a la posición inicial, que es la en la fila de abajo.
  - 2'b11 -> Desaparece el frogger; esta señal es útil para cuando se estén mostrando figuras por ejemplo, no queremos que salga el frogger.

- Salidas
  - *SC_STATEMACHINEFROGGER_clear_OutLow* La señal clear permite inicializar el contador a su estado de cuenta 0 cuando este es activado dicho en otras palabras, esta salida hará que nuestro frogger regrese al valor inicial predeterminado. Esta señal se envía cuando el estado del STATEMACHINE sea "STATE_INIT_0".
  - *SC_STATEMACHINEFROGGER_init_OutLow:* La salida init permite reiniciar todos los registros a la señal definida como inicial, en este caso envía el frogger de vuelta abajo.
  - *SC_STATEMACHINEFROGGER_load0_OutLow:* La salida load0 bajara todos los registros de frogger de la matriz. Esta señal se envía cuando el estado del STATEMACHINE sea "STATE_DOWN".
  - *SC_STATEMACHINEFROGGER_load1_OutLow:* La salida load1 subirá todos los registros de frogger de la matriz. Esta señal se envía cuando el estado del STATEMACHINE sea "STATE_UP".
  - *[1:0] SC_STATEMACHINEFROGGER_shiftselection_Out:* Esta salida es un BUS de 2 bits, que tiene 3 casos:
    - 2'b11 -> Los registros deben mantenerse quietos.
    - 2'b01 -> Los registros deben hacer un shift a la izquierda. Esta señal se enviará cuando el estado del STATEMACHINE sea "STATE_LEFT".
    - 2'b10 -> Los registros deben hacer un shift a la derecha. Esta señal se enviará cuando el estado del STATEMACHINE sea "STATE_RIGHT".

COMPONENTE SECUENCIAL SC_STATEMACHINEBACKG

Descripción Funcionalidad: Este componente tiene como funcionalidad permitir sectorizar las diferentes configuraciones del background esperadas para el funcionamiento del juego; dentro de estas están incluidas: El manejo de los carros (obstáculos de frogger), y sus definiciones dependiendo del nivel actual, además de las casas a las que debe llegar frogger, y las figuras que se muestran cuando este pierde, gana o pasa al siguiente nivel. El StateMachine del Background es el PRINCIPAL dispositivo del programa, dado que este es el encargado de definir la señal no solo de los registros del background sino también el de las entradas, y además puede fijar en un punto o reiniciar los registros del frogger mediante la señal *setFrogger_Out*.

Descripción Algoritmo: En el STATEMACHINEBACKG se manejan dos MUX, uno de estados, y uno de salidas. El MUX de estados, se encarga de definir dependiendo de las entradas definidas más abajo, cuál es el estado futuro del fondo de la matriz led, la cual incluye tanto los carros como las casas de los frogger y las figuras de fondo. Por otro lado, el MUX de salidas depende del estado presente en el que se encuentre el STATEMACHINE, y en base a este define las salidas que tendrá, que se definen más abajo.

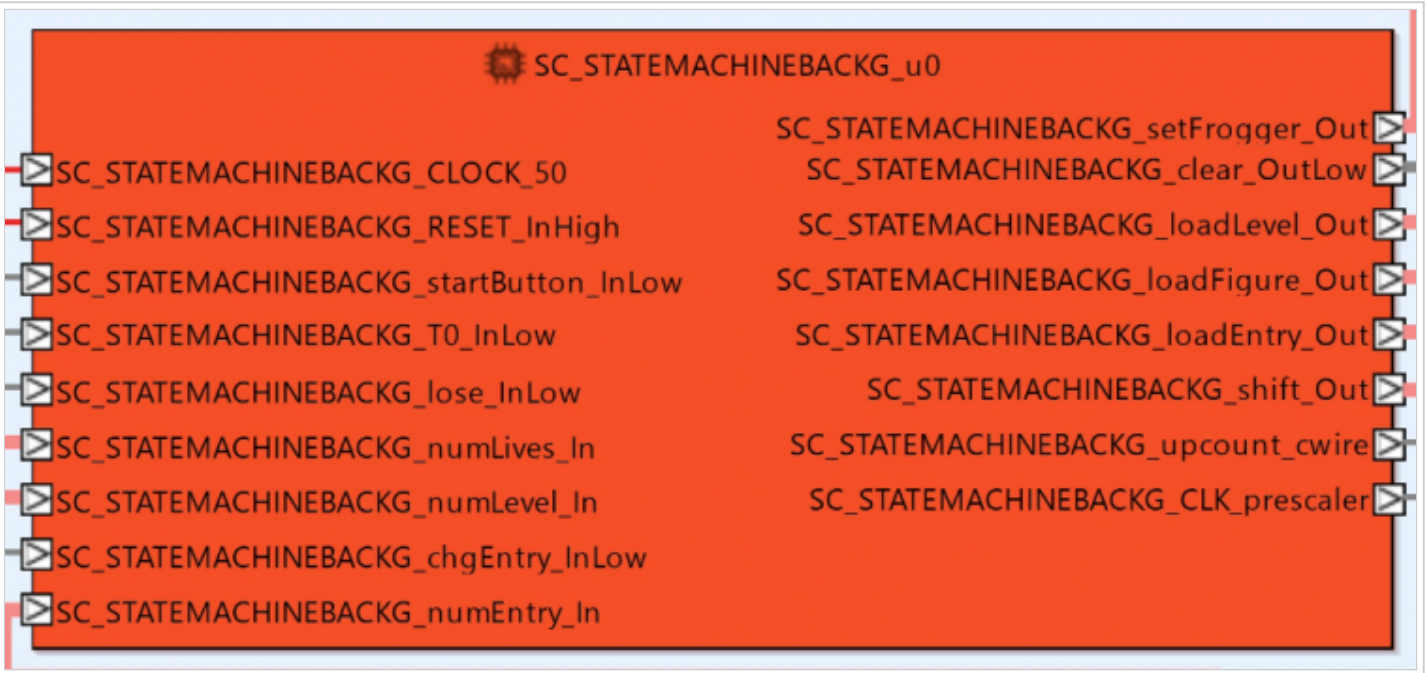


Imagen de diagrama de caja negra, bloque secuencial STATEMACHINEBACKG

- Entradas
  - *SC_STATEMACHINEBACKG_CLOCK_50:* Esta señal nos permitirá coordinar las acciones que queremos que tenga nuestro circuito, en este caso será coordinar el comportamiento del background que incluirá el movimiento de los LEDS.
  - *SC_STATEMACHINEBACKG_RESET_InHigh:* Esta señal indica que el jugador ha oprimido el botón RESET con lo cual se deben volver al estado inicial.
  - *SC_STATEMACHINEBACKG_startButton_InLow:* Esta entrada es la que dará inicia al juego, cargando todos los bits de los registros del background para el primer nivel provocando que los leds se empiecen a mover de izquierda a derecha.
  - *SC_STATEMACHINEBACKG_T0_InLow:* Es la proporción que existe entre la potencia de señal que se transmitirá en el background y el ruido que haya de fondo.
  - *SC_STATEMACHINEBACKG_lose_InLow* Esta entrada indica si llega a haber un choque entre el frogger y los carros, para hacerlo el sistema debe verificar si los bits del frogger y los bits del background son iguales, si llegan a ser iguales entonces mandara la señal 0 indicando que hubo un choque y el jugador perdió.
  - *SC_STATEMACHINEBACKG_chgEntry_InLow:* Esta señal que recibe es la que indica si hubo algún cambio en las entradas, así se sabrá cuándo el frogger haya llegado a alguna nueva entrada o haya ganado.
  - *[1:0] SC_STATEMACHINEBACKG_numEntry_In:* Esta entrada es un BUS de 2 bits que indica cuales frogger han entrado a la casa, y tiene 4 casos:
    - 2'b00 -> Ningún frogger ha entrado a la casa.
    - 2'b01 -> Únicamente el frogger de la derecha ha entrado a la casa.
    - 2'b10 -> Únicamente el frogger de la izquierda ha entrado a la casa.
    - 2'b11 -> Los dos frogger han entrado a la casa; es decir que el jugador ha pasado el nivel.
  - *[1:0] SC_STATEMACHINEBACKG_numLives_In:* Esta entrada es un BUS de 2 bits que indica cuántas vidas le quedan al usuario. Los casos de este BUS son evidentes, dado que solo puede tener 3,2,1 o 0 vidas.
  - *[2:0] SC_STATEMACHINEBACKGK_numLevel_In:* Esta señal de entrada viene del registro de niveles, e indica el nivel actual en el que se esté. Cabe resaltar que esta entrada se maneja con un bus de 3 bits debido a que así se puede saber cuándo el frogger ha ganado definitivamente (pasa al nivel 5).
- Salidas
  - *[1:0] SC_STATEMACHINEBACKG_setFrogger_Out:* Esta señal le permite al StateMachine del Background controlar la posición acrtual de la rana mediante 4 posibles señales que se describieron previamente en el StateMachine del Frogger.

- *SC_STATEMACHINEBACKG_clearBakg_OutLow:* La señal clear permite inicializar el contador a su estado de cuenta 0 cuando este es activado dicho en otras palabras, esta salida hará que el background regrese al valor inicial predeterminado (vacío). Esta señal se envía cuando el estado del STATEMACHINE sea "STATE_INIT".
- *[2:0] SC_STATEMACHINEBACKG_loadLevel_OutLow:* Esta salida es un BUS de 3 bits que indica a los registros si deben pasar de nivel, y tiene 5 casos:
  - 3'b000 -> Se mantiene en el estado actual, sin pasar de nivel.
  - 3'b001 -> Se empiezan a cargar y mover los carros con las configuraciones del nivel 1.
  - 3'b010 -> Se empiezan a cargar y mover los carros con las configuraciones del nivel 2.
  - 3'b011 -> Se empiezan a cargar y mover los carros con las configuraciones del nivel 3.
  - 3'b100 -> Se empiezan a cargar y mover los carros con las configuraciones del nivel 4.
- *[2:0] SC_STATEMACHINEBACKG_loadFigure_Out:* Esta salida es un BUS de 3 bits que indica si se debe cargar alguna figura estática, y tiene 8 casos:
  - 3'b000 -> No se carga ninguna figura estática
  - 3'b001 -> Se carga la figura estática '2' mostrando que ha pasado al nivel 2.
  - 3'b010 -> Se carga la figura estática '3' mostrando que ha pasado al nivel 3.
  - 3'b011 -> Se carga la figura estática '4' mostrando que ha pasado al nivel 4.
  - 3'b100 -> Se carga la figura estática 'W' mostrando que ha ganado el juego.
  - 3'b101 -> Se carga la figura estática '-2' mostrando que ha perdido una vida y le quedan 2.
  - 3'b110 -> Se carga la figura estática '-1' mostrando que ha perdido una vida y le queda 1.
  - 3'b111 -> Se carga la figura estática '0' mostrando que ha perdido el juego porque se ha quedado sin vidas.
- *[1:0] SC_STATEMACHINEBACKG_loadEntry_Out:* Esta salida es un BUS de 2 bits identico al bus de entrada "SC_STATEMACHINEBACKG_numEntry", el cual le indica al registro de entradas (fila superior) cuáles frogger han entrado y por ende, cuáles entradas debe tapar.
- *[1:0] SC_STATEMACHINEBACKG_shift_Out:* Esta salida es un BUS de 2 bits que indica si el background debe moverse a la derecha o si se debe mantener estático (como por ejemplo si se está mostrando una figura), y tiene 2 casos:
  - 2'b10 -> Mover el escenario (los carros) a la derecha.
  - 2'b11 -> Mantener estático el escenario.
- *SC_STATEMACHINEBACKG_upcount_out:* Esta salida es que la que determinara la velocidad a la que irán nuestros carros por medio de un pre scaler, esta velocidad ira cambiando a medida que cambiemos de nivel, entre mayor sea el nivel mayor será la velocidad.
- *SC_STATEMACHINEBACKG_CLK_prescaler:* Esta salida envía la nueva señal de clock proveniente del prescaler, que el STATEMACHINE decide, según el número del nivel en el que el jugador esté actualmente.

COMPONENTE SECUENCIAL RegFROGGER

Descripción Funcionalidad: Las 8 instancias de este bloque en el proyecto se encargan de almacenar y modificar los registros de las 8 filas de la matriz que se encargan de identificar como un único punto, la ubicación en la matriz del frogger ignorando la ubicación de los demás objetos del juego.

Descripción Algoritmo: En el RegFROGGER se maneja un único MUX encargado de la evaluación de las entradas y la modificación de las salidas y el registro que se almacena.

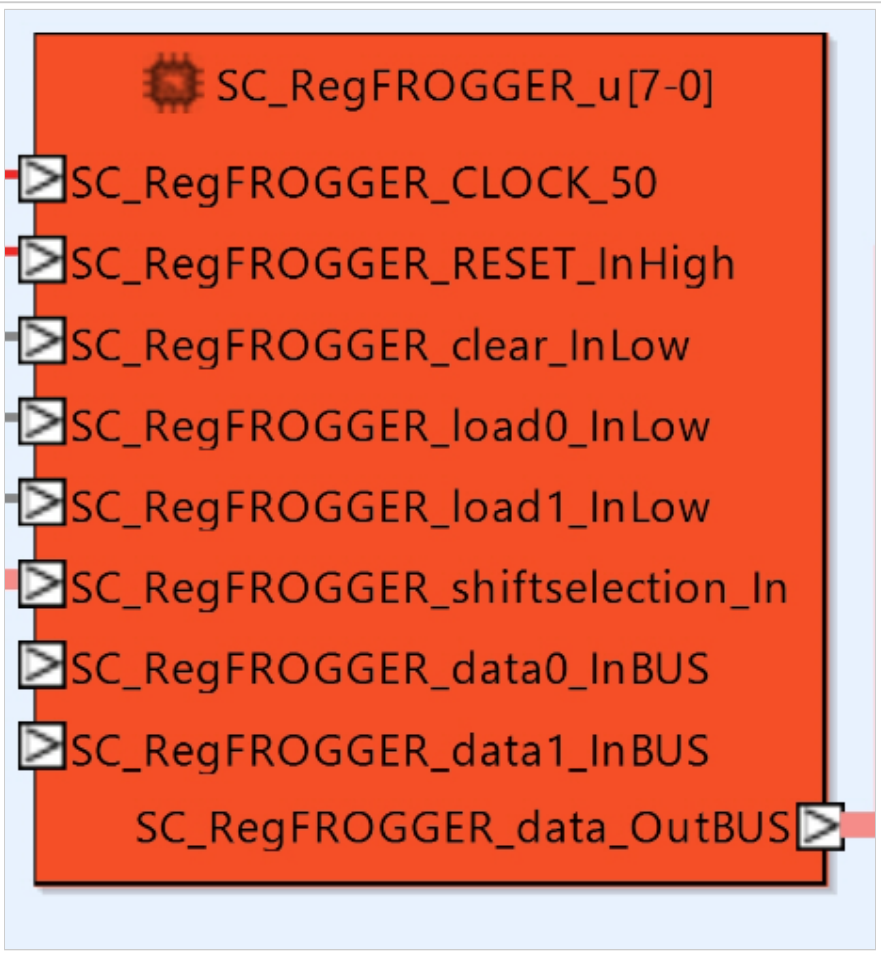


Imagen de diagrama de caja negra, bloque secuencial RegFROGGER

- Entradas
  - *SC_RegFROGGER_CLOCK_50:* Entrada de un bit que funciona como el reloj del sistema, permitiendo coordinar las acciones del sistema.
  - *SC_RegFROGGER_RESET_InHigh:* Entrada de un bit que se encarga de enviar el registro a 0 si es activada (reiniciar el registro).
  - *SC_RegFROGGER_clear_InLow:* Señal que "elimina" el frogger de cada registro, dejándolos llenos de 0s.
  - *SC_RegFROGGER_init_InLow:* Señal que
  - *SC_RegFROGGER_load0_InLow:* La entrada load0 intercambia el registro actual de la fila, con el de la fila inmediatamente inferior para simular que el frogger bajó sin importar en qué registro esté.
  - *SC_RegFROGGER_load1_InLow:* La entrada load1 intercambia el registro actual de la fila, con el de la fila inmediatamente superior para simular que el frogger subió sin importar en qué registro esté.
  - *[1:0] SC_RegFROGGER_shiftselection_In:* Esta entrada es un BUS de 2 bits que controla si el todos los registros del frogger se mueven hacia la izquierda, hacia la derecha o se quedan quietos.
  - *[7:0] SC_RegFROGGER_data0_InBUS:* Esta entrada es un BUS de 8 bits que es el bus de datos de la fila inferior del frogger.
  - *[7:0] SC_RegFROGGER_data1_InBUS:* Esta entrada es un BUS de 8 bits que es el bus de datos de la fila superior del frogger.
- Salidas

- *[7:0] SC_RegFROGGER_data_OutBUS:* Señal final actualizada del bus de bits corresondientes a la posicion del frogger (el registro actual después de la evaluación).

COMPONENTE SECUENCIAL RegFROGGERENTRY

Descripción Funcionalidad: La única instancia de este bloque se encarga de almacenar y modificar el registro de la fila de entradas del frogger, como cuáles entradas deben estar encendidas indicando que el frogger ya llegó, y cuáles deben estar apagadas y por llenar.

Descripción Algoritmo: En el RegFROGGERENTRY se maneja un único MUX encargado de la evaluación de las entradas y la modificación de las salidas y el registro que se almacena.

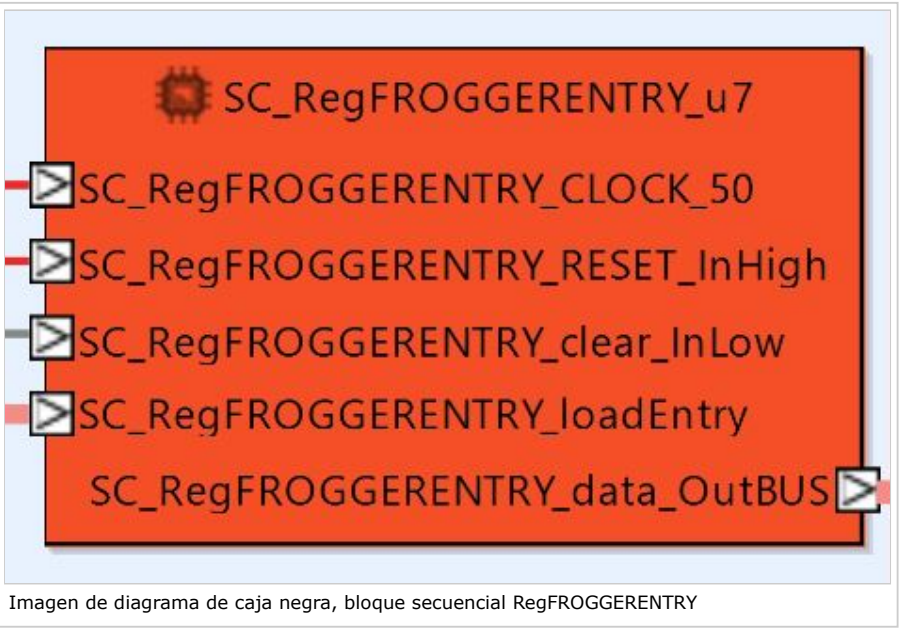


Imagen de diagrama de caja negra, bloque secuencial RegFROGGERENTRY

- Entradas
  - *SC_RegFROGGERENTRY_CLOCK_50:* Entrada de un bit que funciona como el reloj del sistema, permitiendo coordinar las acciones del sistema.
  - *SC_RegFROGGERENTRY_RESET_InHigh:* Entrada de un bit que se encarga de enviar el registro a 0 si es activada (reiniciar el registro).
  - *SC_RegFROGGERENTRY_clear_InLow:* Señal que reinicia al bus de bits al definido como inicial, es decir que apaga las 2 entradas del frogger (estado inicial predeterminado).
  - *[2:0] SC_RegFROGGERENTRY_loadEntry:* Esta entrada es un BUS de 2 bits que indica cuáles entradas del frogger ya se han llenado, y cuáles hacen falta llenar, de modo que indica si debe encender en el registro las 2 entradas, solo la de la izquierda, solo la de la derecha, o ya se pasó al siguiente nivel. Los casos de este BUS ya fueron definidos anteriormente.
- Salidas
  - *SC_RegFROGGERENTRY_data_OutBUS:* Señal final actualizada del bus de bits corresondiente a las entradas del frogger (el registro actual después de la evaluación).

COMPONENTE SECUENCIAL RegBACKG

Descripción Funcionalidad: Las 7 instancias del RegBACKG se encargan de guardar y actualizar el bus de bits de cada fila (sin incluir la superior que es la de las entradas) que corresponde a la posición en la cual se encuentran los carros del juego dependiendo del nivel en que se encuentra.

Descripción Algoritmo: En el RegBACKG se maneja un único MUX encargado de la evaluación de las entradas y la modificación de las salidas y el registro que se almacena. El RegBACKG recibe como parámetro inicial los diferentes valores que tiene tanto para representar las figuras, como de la cantidad y longitud de los carros de cada nivel, y dentro de dicho MUX decide si debe cargar alguno de estos valores, y si debe mantenerse estático o debe mover los carros.

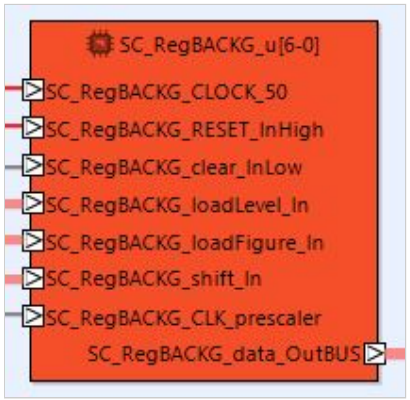


Imagen de diagrama de caja negra, bloque secuencial RegBACKG

- Entradas
  - *SC_RegBACKG_CLOCK_50:* Entrada de un bit que funciona como el reloj del sistema, permitiendo coordinar las acciones del sistema.
  - *SC_RegBACKG_RESET_InHigh:* Entrada de un bit que se encarga de enviar el registro a 0 si es activada (reiniciar el registro).
  - *SC_RegBACKG_clear_InLow:* Señal que reinicia al bus de bits al definido como inicial, es decir que envía el frogger a la posición que tiene al comienzo del juego.
  - *[2:0] SC_RegBACKG_loadLevel_In:* Esta entrada es un BUS de 3 bits que indica a los registros si deben pasar de nivel. Los casos de esta señal fueron definidos anteriormente.
  - *SC_RegBACKG_loadFigure_In:* Esta entrada es un BUS de 3 bits que indica si se debe cargar alguna figura estática. Los casos de esta señal fueron definidos anteriormente.
  - *[1:0] SC_RegBACKG_shift_In:* Esta entrada es un BUS de 2 bits que indica si el registro del background debe moverse a la derecha o si se debe mantener estático (como por ejemplo si se está mostrando una figura).
- Salidas
  - *[7:0] SC_RegBACKG_data_OutBUS:* Señal final actualizada de cada bus de bits correspondiente a los carros o figura del background (el registro actual después de la evaluación).

COMPONENTE SECUENCIAL RegLIVES

Descripción Funcionalidad: El componente RegLIFES tiene como única función almacenar el número de vidas actuales que tiene el jugador.

Descripción Algoritmo: En el RegLIFES se maneja un único MUX bastante sencillo encargado de la evaluación de la única entrada que avisa si el jugador perdió, en caso afirmativo disminuye el registro en 1, de lo contrario lo deja igual.

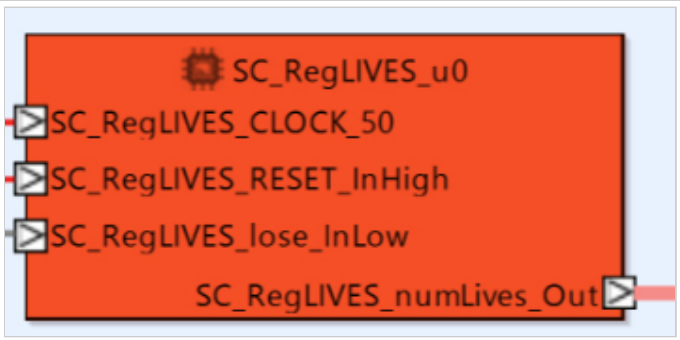


Imagen de diagrama de caja negra, bloque secuencial RegLIFES

- Entradas
  - *SC_RegLIVES_CLOCK_50:* Señal que le indica al componente cuando es hay un flanco de reloj para actualizar la salida..
  - *SC_RegLIVES_RESET_InHigh:* Esta señal indica que el jugador ha oprimido el botón RESET con lo cual se deben volver al estado inicial.
  - *SC_RegLIVES_lose_InLow:* Esta entrada indica si hubo alguna colisión del frogger, es decir si perdió una vida.
- Salidas
  - *[1:0] SC_RegLIVES_numLives_Out:* Esta salida es el registro actual de vidas actualizado según si perdió alguna vida o si se mantiene igual.

COMPONENTE SECUENCIAL RegLEVEL

Descripción Funcionalidad: El componente RegLEVEL tiene como única función almacenar el nivel actual.

Descripción Algoritmo: En el RegLEVEL se maneja un único MUX bastante sencillo encargado de la evaluación de las entradas que avisa si el jugador paso de nivel, en caso afirmativo el registro se suma en 1, de lo contrario lo deja igual.

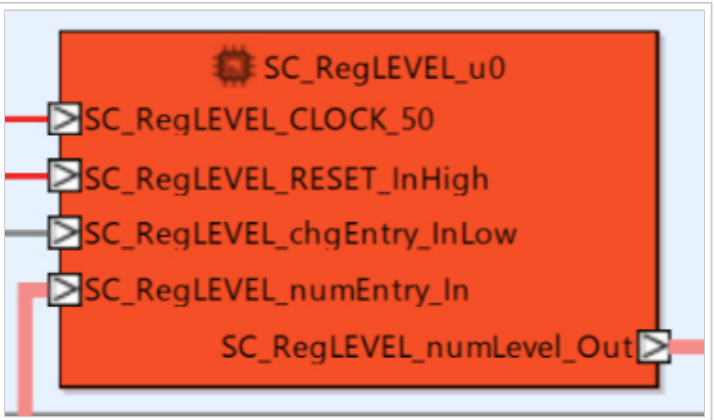


Imagen de diagrama de caja negra, bloque secuencial RegLEVEL

- Entradas
  - *SC_RegLEVEL_CLOCK_50:* Señal que le indica al componente cuando es hay un flanco de reloj para actualizar la salida.
  - *SC_RegLEVEL_RESET_InHigh:* Esta señal indica que el jugador ha oprimido el botón RESET con lo cual se deben volver al estado inicial.
  - *SC_RegLEVEL_chgEntry_InLow:* Esta entrada indica si hubo algún cambio en las entradas indicando si llegó a alguna, con lo cual debe revisar la señal *numEntry_In*.
  - *[1:0] SC_RegLEVEL_numEntry_In:* Esta señal de entrada indica el número de entradas llenadas por el frogger hasta ahora; esta señal viene directamente del RegENTRY.
- Salidas
  - *[3:0] SC_RegLEVEL_numLEVEL_Out:* Esta salida es el registro actual del nivel en el que se encuentra el jugador.
    - 3b'001 nivel 1.
    - 3b'010 nivel 2.
    - 3b'011 nivel 3.
    - 3b'100 nivel 4.
    - 3'b101 nivel 5 (ganó el juego).

COMPONENTE SECUENCIAL RegENTRY

Descripción Funcionalidad: El componente RegENTRY tiene como única función almacenar el número de entradas llenadas por el frogger hasta el momento y actualizarlo dependiendo si este entra a alguna entrada.

Descripción Algoritmo: En el RegENTRY se maneja un único MUX bastante sencillo encargado de la evaluación de las entradas que verifica si se entró en alguna casa, y si así es, enciende la señal chgEntry y modifica el registro.

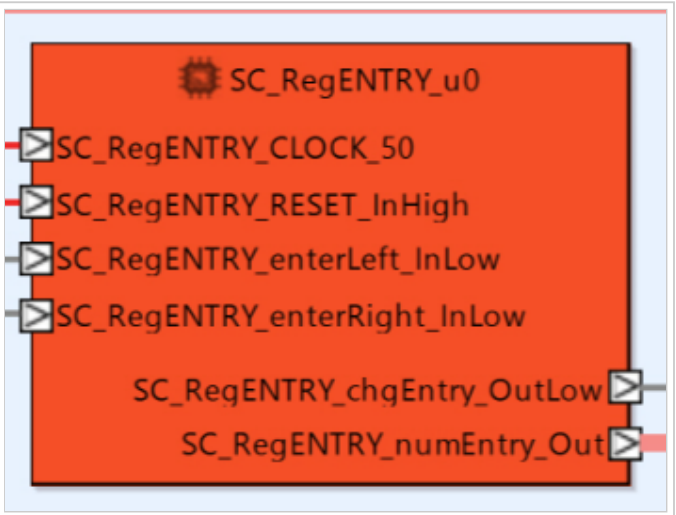


Imagen de diagrama de caja negra, bloque secuencial RegENTRY

- Entradas
  - *SC_RegLEVEL_CLOCK_50:* Señal que le indica al componente cuando es hay un flanco de reloj para actualizar la salida.
  - *SC_RegLEVEL_RESET_InHigh:* Esta señal indica que el jugador ha oprimido el botón RESET con lo cual se deben volver al estado inicial.
  - *SC_RegENTRY_enterLeft_InLow:* Esta señal indica 0 si el comparador de entradas verificó que el frogger ha entrado a la casa de la izquierda.
  - *SC_RegENTRY_enterRight_InLow:* Esta señal indica 0 si el comparador de entradas verificó que el frogger ha entrado a la casa de la derecha.
- Salidas
  - *SC_RegLEVEL_chgEntry_OutLow:* Esta salida indica si hubo algún cambio en las entradas enterLeft o enterRight indicando si llegó a alguna y hubo algún cambio en el registro.

COMPONENTE SECUENCIAL SPEEDCOUNTER

Descripción Funcionalidad: Componente secuencial necesario para darle diferentes velocidades a los carros que va sumando las señales a medida que el clock contenga flancos de subida

Descripción Algoritmo: Descripción de algoritmo.

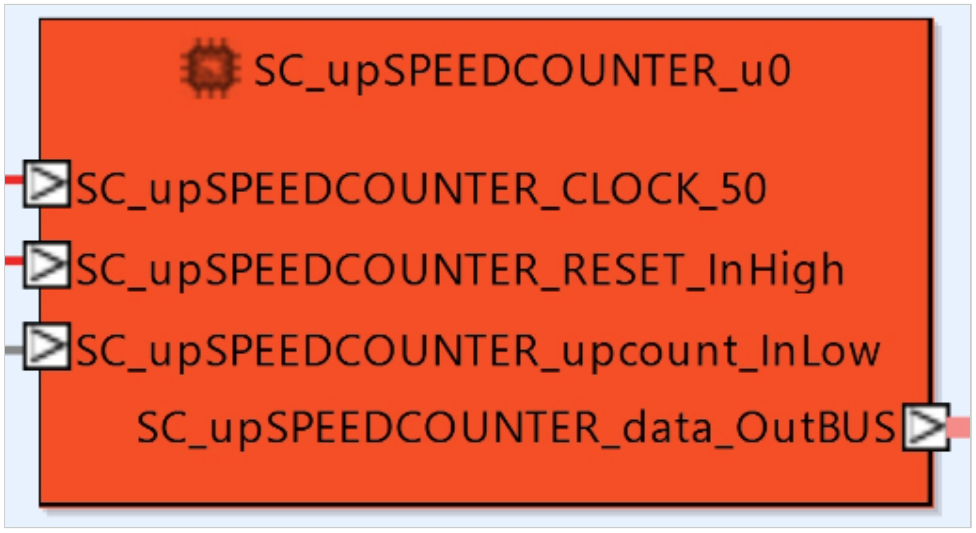


Imagen de diagrama de caja negra, bloque secuencial SPEEDCOUNTER

Imagen de diagrama de flujo / pasos macro-algoritmo componente X.

- Entradas
  - *SC_upSPEEDCOUNTER _CLOCK_50:* Señal que le indica al componente cuando es hay un flanco de reloj para actualizar la salida.
  - *SC_ upSPEEDCOUNTER _RESET_InHigh* Señal que al estar en 1 reiniciara la salida y por ende el conteo generado por la señal del upcount.
  - *SC_ upSPEEDCOUNTER _upcount_InLow:* Señal que va sumando de a una señal al registro cuando esta pulsada los los valores .
- Salidas
  - *SC_ upSPEEDCOUNTER_data_OutBus:* señal que almacena el conteo generado por la señal upcount y que es actualizada por el clock o reseteada por la señal reset.

COMPONENTE COMBINACIONAL SPEEDCOMPARATOR

Descripción Funcionalidad: .

Descripción Algoritmo: Descripción de algoritmo.

Imagen de diagrama de caja negra, bloque combinacional SPEEDCOMPARATOR

Imagen de diagrama de flujo / pasos macro-algoritmo componente X.

- Entradas
  - *CC_SPEEDCOUNTER_data_InBus:* Sirven para darle diferentes velocidades a los carros.

- Salidas
  - *CC_SPEEDCOUNTER_data_OutLow:* Este tomara el registro de las velocidades que irán cambiando a medida que avancemos de nivel.

COMPONENTE COMBINACIONAL BOTTOMSIDECOMPARATOR

Descripción Funcionalidad: El bloque BOTTOMSIDECOMPARATOR tiene como fin evitar evaluar si el frogger está en el RegFROGGER de la fila de abajo, para evitar que este pueda bajar más.

Descripción Algoritmo: El algoritmo del BOTTOMSIDECOMPARATOR es bastante simple, este consiste en analizar la señal del registro 0 del frogger, y retorna 0 si el frogger está en esta, 1 de lo contrario.

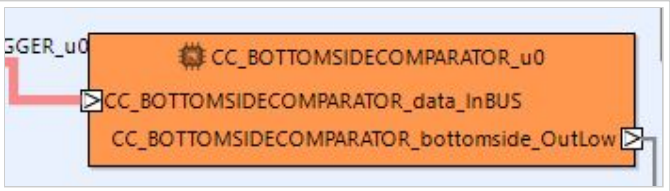


Imagen de diagrama de caja negra, bloque combinacional BOTTOMSIDECOMPARATOR

Imagen de diagrama de flujo / pasos macro-algoritmo componente X.

- Entradas
  - *[7:0] CC_BOTTOMSIDECOMPARATOR_data_InBus:* Este es el registro de salida del registro de la fila 0 del FROGGER sobre el cual se va a comparar.
- Salidas
  - *CC_BOTTOMSIDECOMPARATOR_bottomside_OutLow:* Esta salida indica si el frogger se encuentra en la fila inferior, si es así envía un 0 indicando que no puede bajar más.

COMPONENTE COMBINACIONAL ENTRYCOMPARATOR

Descripción Funcionalidad: El fin del componente ENTRYCOMPARATOR es de comparar la fila de llegada (superior) del frogger y el registro RegENTRY, y revisar si el frogger ha llegado a alguna casa, y retorna un bus que indica cuáles casas se han llenado y cuáles hacen falta.

Descripción Algoritmo: El algoritmo del ENTRYCOMPARATOR es bastante simple, este consiste en analizar la señal del registro 7 del frogger y el registro de entradas del background y revisar si el frogger chocó con otra entrada o si este entró en alguna entrada válida.

Imagen de diagrama de caja negra, bloque combinacional ENTRYCOMPARATOR

Imagen de diagrama de flujo / pasos macro-algoritmo componente X.

- Entradas

- *[7:0] CC_ENTRYCOMPARATOR_entryBUS:* Este es el bus de la entrada a la que debe llegar el frogger.
- *[7:0] CC_ENTRYCOMPARATOR_froggerBUS:* Este es el bus de la fila superior del frogger.

- Salidas
  - *CC_ENTRYCOMPARATOR_enterLeft_OutLow:* Esta salida indica si el frogger justo ha entrado en la casa izquierda.
  - *CC_ENTRYCOMPARATOR_enterRight_OutLow:* Esta salida indica si el frogger justo ha entrado en la casa derecha.
  - *CC_ENTRTYCOMPARATOR_lose_Out:* Esta salida indica si el frogger tuvo una colisión con otra entrada no esperada.

COMPONENTE COMBINACIONAL COLLISIONCOMPARATOR

Descripción Funcionalidad: Las 6 instancias del componente COLLISIONCOMPARATOR tienen como fin comparar los registros [1 a 6] del frogger con los registros [1 a 6] del background para detectar cualquier tipo de colisión entre el frogger y los carros y así detectar si perdió.

Descripción Algoritmo: El algoritmo del COLLISIONCOMPARATOR es bastante simple, este consiste en analizar la señal de entrada del registro del frogger y el del background y hacer una comparación bit a bit (AND) y así verificar si coinciden, lo cual indicaría que perdió.

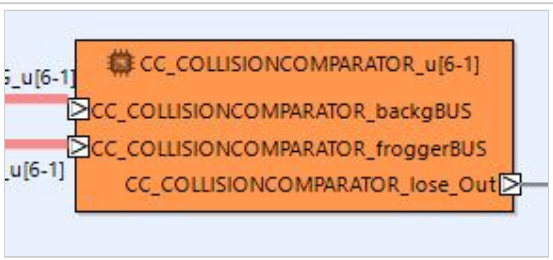


Imagen de diagrama de caja negra, bloque combinacional COLLISIONCOMPARATOR

Imagen de diagrama de flujo / pasos macro-algoritmo componente X.

- Entradas
  - *[7:0] CC_COLLISIONCOMPARATOR_backgBUS:* Este es el bus del background de la fila correspondiente de comparación.
  - *[7:0] CC_COLLISIONCOMPARATOR_froggerBUS:* Este es el bus del frogger de la fila correspondiente de comparación.
- Salidas
  - *CC_COLLISIONCOMPARATOR_lose_Out:* Esta salida indica si el frogger tuvo una colisión con algún bit del background (carro).

REPORTES TÉCNICOS

Memorias de Cálculo

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de realizar cálculos de parámetros cuantitativos de módulos funcionales para dar una solución matemática y científica a los parámetros que lo requieran.

- a. Se presentan memorias de cálculo completas del proyecto, que incluyen justificaciones de su pertinencia para todos los productos que las requieran (ejemplo: cálculos de variables temporales, de espacio de memoria, entre otros).

Memorias de Cálculo: Dentro de los cálculos que se necesitaron para nuestra implementación del juego frogger fueron las siguientes.

- *Velocidad Preescalers:* Fue necesario calcular la cantidad de ciclos del clock que eran necesarios para avanzar una cierta cantidad de bits/s y así dar cierta velocidad a los carros, mucho más lenta que la del clock general, esto se realizó mediante la siguiente ecuación:

$$v_{carros}\left(\frac{shift}{s}\right) = \frac{f_{CLOCK}\left(\frac{ciclo}{s}\right)}{2^{x+1} - 1\left(\frac{ciclo}{shift}\right)}$$

Esta ecuación se obtiene analizando el mecanismo utilizado en los preescalers, el cual consiste en crear un vector de x bits el cual se inicializa en 0 (x' b000…0) y se inicia el conteo en unos cada ciclo de reloj, con lo cual el vector se llenará de unos en 2^(x+1)-1 ciclos de reloj. De modo que al llenarse de unos se hace un shift y se reinicia el contador para el siguiente shift. Teniendo en cuenta que la incógnita es x, se despeja de la ecuación quedando:

$$x = log_2\left(\frac{f}{v} + 1\right) - 1$$

Y teniendo en cuenta que la frecuencia del reloj del sistema es constante, se reemplaza:

$$x = log_2\left(\frac{50000000}{v} + 1\right) - 1$$

Y se obtienen los siguientes valores, los cuales son la longitud que deben tener los vectores para tardar la velocidad esperada:

- $v = 0.75\frac{shift}{s} \Rightarrow x = log_2\left(\frac{50000000}{0.75} + 1\right) - 1 = 24.99 \approx 25bits$
- $v = 1.5\frac{shift}{s} \Rightarrow x = log_2\left(\frac{50000000}{1.5} + 1\right) - 1 = 23.99 \approx 24bits$
- $v = 3\frac{shift}{s} \Rightarrow x = log_2\left(\frac{50000000}{2.5} + 1\right) - 1 = 22.99 \approx 23bits$

- *Carga de Figuras:* Otro cálculo mucho más sencillo que se hizo fue el de la carga de figuras (manejado por la señal de 3 bits loadFigure del componente STATEMACHINEBACKG), dado que esta señal tiene los siguientes casos:
  - (0) 3'b000: No carga ninguna figura
  - (1) 3'b001: Carga la figura de que pasa al nivel 2
  - (2) 3'b010: Carga la figura de que pasa al nivel 3
  - (3) 3'b011: Carga la figura de que pasa al nivel 4
  - (4) 3'b100: Carga la figura de que ganó el juego (W)
  - (5) 3'b101: Carga la figura de que le quedan 2 vidas
  - (6) 3'b110: Carga la figura de que le queda 1 vida
  - (7) 3'b111: Carga la figura de que perdió el juego (0)

Esta señal se utiliza para manejar las figuras cuando gana y cuando pierde, y se deben hacer 2 cálculos:

1.¿Qué figura de que perdió debe mostrar en función del número de vidas (que se lleva en registro) que tiene? 0 vidas -> figura 7 1 vida -> figura 6 2 vidas -> figura 5 Por ende, la función termina siendo:

$$figura = 7 - n\,(vidas\ restantes)$$

2.¿Qué figura de que ganó debe mostrar en función del nivel al que pasó? nivel 2 -> figura 1 nivel 3 -> figura 2 nivel 4 -> figura 3 ganó el juego (nivel 5) -> figura 4 Por ende, la función termina siendo:

$$figura = n\,(nivel\ al\ que\ pasa) - 1$$

Definición de vectores de prueba y simulaciones de respaldo

[Collapse]

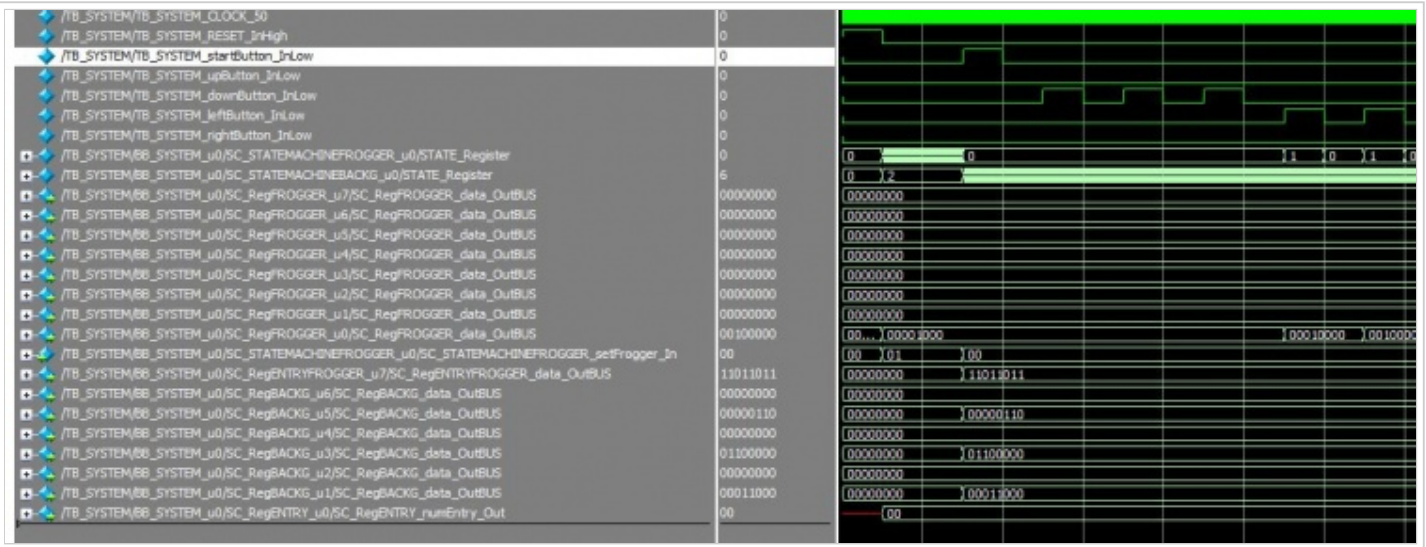
PRODUCTO DE CALIDAD: El estudiante es capaz de proponer una estrategia de identificación de vectores de prueba para validar la funcionalidad del producto.

- a. Los vectores de prueba se seleccionan describiendo en un párrafo una estrategia explícita y claramente definida y estos vectores permiten comprobar totalmente la funcionalidad.
- b. Se presentan resultados de simulación para el producto solicitado, explicando tres o más casos de funcionamiento sobre el diagrama de simulación. La simulación contiene marcadores en la gráfica que señalan situaciones específicas del prototipo.

Vectores de prueba: A fin de comprobar el correcto funcionamiento de la implementación del frogger, se establecieron diferentes vectores de prueba de casos específicos que demostraran el funcionamiento general de frogger. Los casos específicos serán descritos a continuación con su respectiva implementación en quartus.

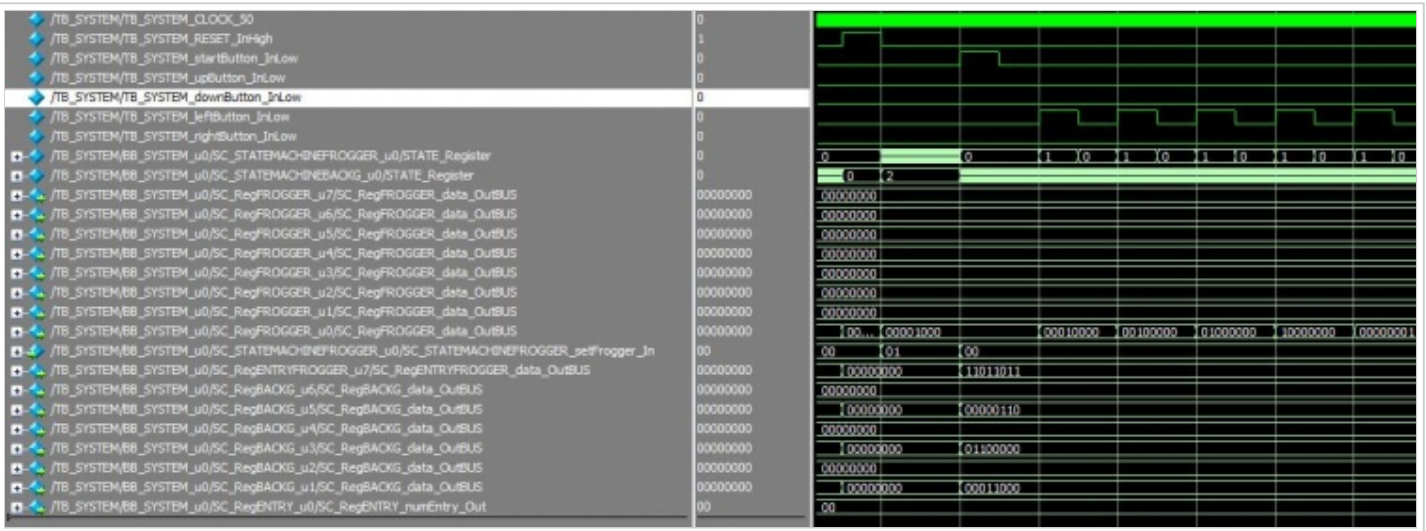
Identificar los vectores de prueba relevantes de bloques/funciones individuales y de todo el sistema.

- Vectores movimiento del frogger:
  - *Botón abajo desde el punto inicial:* El objetivo de esta prueba es comprobar que el frogger al momento de darle señales por medio del boton reset luego inicializarlo con start y posteriormente señal down por medio del pulsador abajo no se moverá el frogger a una posición diferente a la fila en donde se encuentra.
    - simulación: En la captura presentada a continuación se puede observar lo que sucede en las señales del registro después de haber pulsado los botones que emiten la señal de reset y strat, posteriormente de inicializar el juego se pulsan los botones de down para comprobar que el frogger no desaparezca o se cruce con la fila de las casas. Las señales evidenciadas en los registros comprueban el correcto funcionamiento después de ejecutado el vector.



Primera simulación para evaluar el movimiento botón abajo desde el punto inicial.

- *Mover el frogger muchas vece hacia un lado*: El objetivo de esta prueba es el correcto funcionamiento del frogger al llegar a una ultima columna situada en alguno de los lados de la matriz de leds. El resultado esperado tras realizar esta prueba es que el registro de frogger salte de columna final a inicial o viceversa.
  - simulación: En la captura presentada a continuación se puede observar lo que sucede en las señales del registro después de haber pulsado los botones que emiten la señal de reset y strat, posteriormente de inicializar el juego se pulsan los botones left para comprobar en repetidas ocasiones para que el frogger de desplace hasta finalizar la matriz y saltar de la posición 10000000 a la posición 00000001. En la señal de statemachinefrogger se puede evidenciar que los resultados son los esperados por ende se comprueban el correcto funcionamiento después de ejecutado el vector.

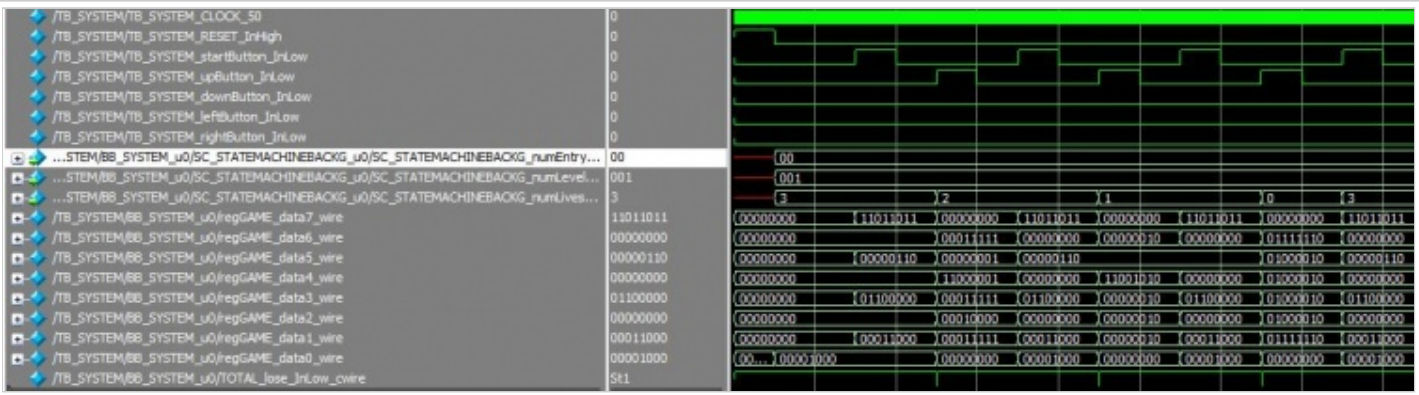


Segundo vector para evaluar el frogger despues de muchas vece hacia un lado.

- Vectores Backgtype:
  - *Colisión de frogger con carro, descuento de vidas y señales del registro:* Este vector de prueba tiene como objetivo comprobar la funcionalidad del descuento de vidas, la actualización de señales de registro y los saltos siguientes de la maquina de estados. El método empleado para esta prueba consiste en dar señales de

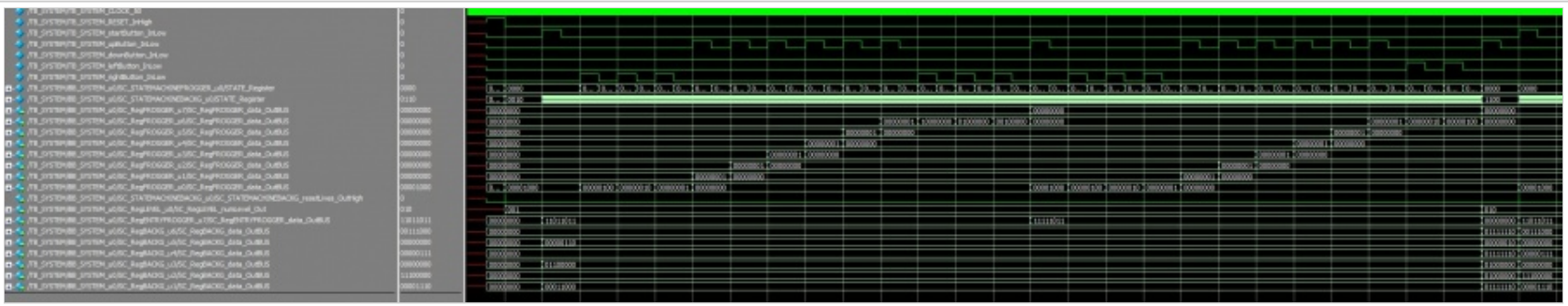
start, prosteriomente generar una señal de up para que muera el frogger y repetir este proceso 3 veces que son el numero de vidas del frogger.

- simulación: En la captura presentada a continuación se puede observar lo que sucede en las señales del registro después de haber pulsado los botones que emiten la señal de reset y strat, posteriormente de inicializar el juego se pulsan los botones up para ocasionar una colisión, esto se repite tres veces debido a que es ese el numero de vidas. En las señales de registros se evidencia correctamente el reinicio de nivel tras la colisión, además en las señales del statemachine que lleva el conteo de las vidas también se puede observar la resta de las vidas de 3 a 2 y de 2 a 1 correctamente después de no tener mas vidas reinicia el juego y empieza de nuevo con tres vidas. Los resultados son los esperados, por ende, se comprueban el correcto funcionamiento después de ejecutado el vector.



Tercer vector para evaluar la colisión de frogger con carro, descuento de vidas y señales del registro.

- *Verificar pasada de nivel:" El objetivo de este vector de prueba es comprobar si son correctas las señales y la respuesta de la arquitectura después de la llegada del primer y segundo frogger a la casal. Las respuestas esperadas son el cambio de nivel con el aviso de nivel siguiente.*
  - simulación: En la captura presentada a continuación se puede observar lo que sucede en las señales del registro después de haber pulsado los botones que emiten la señal de reset y strat, posteriormente de inicializar el juego se pulsan los botones de righ y up tal forma que el frogger llega a la primera casa correctamente y después ocurra lo mismo co el siguiente frogger pero en este caso en la casa faltante. Las señales del registro del frogger evidencian el correcto fincionamiento en el movimiento de los dos froggers y la señal de regENTRY en primera medida pasa de su bus en 11011011 a 11111011 después de las señales de movimiento. Esto indica que el primer frogger ha llenado la señal que corresponde a la primera casa correctamente. Después de que el segundo frogger llega a la segunda casa el regBACKG actualiza sus señales mostrando y despues de start continuando el siguiente nivel. Los resultados son los esperados, por ende, se comprueban el correcto funcionamiento después de ejecutado el vector de prueba.


Cuarto vector de prueba para evaluar la pasada de nivel.

Indicadores de utilización de recursos y rendimiento de los dispositivos utilizados

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de identificar indicadores de utilización de recursos para dimencionar la complejidad de la solución.

- a. Se presenta una tabla con indicadores de utilización de recursos para el producto solicitado de forma ordenada que incluya al menos (1) el porcentaje de utilización de recurso de lógica programable y recursos utilizados (caso hardware), (2) porcentaje de utilización de la memoria de instrucciones (caso software) y (3) frecuencia de operación del dispositivo. Se presenta un párrafo que relaciona las características del diseño con los valores de la tabla.

Parrafo 1: La tabla presentada a continuación se brinda automaticamente por el software Quartus al compilar el código HDL, y brinda información sobre el rendimiento actual, y los recursos utilizados al compilar el programa.

Tabla indicadores

Flow Summary

<<Filter>>

Flow Status	Successful - Fri May 14 11:58:16 2021
Quartus Prime Version	19.1.0 Build 670 09/22/2019 SJ Lite Edition
Revision Name	BB_SYSTEM
Top-level Entity Name	BB_SYSTEM
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	789 / 22,320 (4 %)
Total registers	296
Total pins	18 / 154 (12 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Tabla del rendimiento de la solución final

IMPLEMENTACIÓN

Descripción en lenguajes HW y SW (Códigos fuente)

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de describir lo solicitado en lenguajes de HDL para evidenciar la utilización de los elementos de la herramienta de diseño y del lenguaje y contruir su solución en tecnología de dispositivos de lógica programable.

- a. La descripción en lenguajes hardware (complejidad del código, diferencia combinacional y secuencial) y software (complejidad del código, diferencia tipos de variables funciones métodos) es correcta y corresponde al producto solicitado.
- b. Se incluye documentación completa para estructurar y/o entender el código claramente (indentación y sintaxis de los lenguajes), nombrando correcta y adecuadamente todas las señales, variables y demás elementos relevantes.

Dispositivo 1

- Archivos Programación QUARTUS (Descargar (https://www.mediafire.com/file/riw55r9a1o99mll/PR-FROGGER_G13_FuncionalidadTotal.zip/file))
- Archivo .SOF para enviar a la FPGA conectada a los botones y la matriz (Descargar (https://www.mediafire.com/file/11w18h726wlg6pq/BB_SYSTEM.sof/file))

Funcionalidad modular

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de identificar una agrupación de componentes para evidenciar una funcionalidad parcial del producto para concretar avances efectivos.

- a. Se definen módulos (que sumados corresponden a todo el prototipo) sobre los cuales se propusieron pruebas independientes y todas funcionaron adecuadamente y completamente según las especificaciones.

Vídeos y fotos de demostración de módulos del sistema

Se logró la funcionalidad global.

Funcionalidad global del sistema

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de integrar todos los componentes en un producto funcional que cumple con toda las especificaciones y restricciones para evidenciar una funcionalidad global de la solución.

- a. El prototipo funciona adecuadamente y completamente según las especificaciones del producto solicitado.

Vídeos y fotos de demostración del prototipo final

Descripción: En el video 1 se puede observar el funcionamiento total de nuestra implementación propuesta del proyecto; es posible que debido a algunas fallas de internet al comienzo del video no se pueda ver mucho, sin embargo a partir del segundo 25 aproximadamente se puede observar tanto el caso en el que pierde, que se muestra que se pierde una vida (y se muestra la figura del número de vidas restantes), de cuando pierde todas las vidas que se muestra un "0" y se pasa de vuelta a

primer nivel; cuando llena una casa se puede observar cómo se enciende el led y se activa solamente la otra entrada, y al completar las 2 entradas se muestra una figura del nivel al que pasa, y se cambian los carros del nivel.

Video 1

- https://www.youtube.com/watch?v=bemeV4KzBMI&feature=youtu.be

REFLEXIÓN

Resultados y lecciones aprendidas

[Collapse]

PRODUCTO DE CALIDAD: El estudiante es capaz de fundamentar/explicar el proceso de diseño realizado evidenciando las mejoras o problemas de la soluciór propuesta para demostrar la comprensión y dominio de las actividades propias del diseño.

- a. Se proponen nuevas especificaciones y aplicaciones del trabajo realizado (ejemplo: mayores niveles de complejidad y usos en otros contextos)..
- b. En caso de no lograr el item de funcionamiento glogal; identifica y argumenta las razones principales del no funcionamiento.
- c. El lenguaje disciplinar es preciso y adecuado, haciendo uso de frases gramaticalmente correctas, sin errores ortográficos.

- A lo largo del proyecto se presentaron diferentes dificultades con algunas de las especificaciones que se plantearon desde el inicio. Sin embargo, al finalizar el proyecto fue posible cumplir con todas las especificaciones inicialmente planteadas. El proceso para llegar al cumplimiento total de los requerimientos mostró ser de mucha utilidad para el correcto entendimiento e interiorización de los procesos y conocimientos necesarios para el desarrollo de un aplicativo usando la programación de hardware.
- A lo largo del proyecto se presentaron diferentes dificultades que podrían caber dentro de lo normal en la realización de un proyecto. En primera medida, para la primera entrega nos encontramos con dificultades para establecer los lineamientos generales que cupiesen dentro de los limites posibles de desarrollo, acorde a nuestros conocimientos en la programación de hardware usando verilog. Era necesario aterrizar nuestras ideas a la realidad y a los posibles de implementación. Al finalizar el proyecto pudimos evidenciar que las ideas planteadas desde el inicio fueron ejecutables y bien logradas por lo que se podría decir que se suplió exitosamente con la primera problemática. Posteriormente, en la elaboración del macro algoritmo, fue necesaria la reelaboración de este por faltas de especificación a detalle de la funcionalidad interna del producto. Esta dificultad nos incitó a profundizar en la solución a detalle de cada paso interno del producto.
- En la segunda entrega la dificultad más notable la tuvimos con el desarrollo de la arquitectura del sistema parte hardware en gran medida por no tener completamente claro que era lo que significaba esta y los requerimientos específicos. Por esto, la tarea se hizo mas larga y extensa de lo que en principio debería. sin embargo, al finalizar este contratiempo se pudo tener claro el significado de una arquitectura de sistema de hardware y su utilidad para la posterior programación del producto.
- Para la tercera entrega fue necesaria la implementación de cálculos que modelaran las diferentes velocidades de los carros teniendo en cuenta la frecuencia del clock y la definición del shift dependiendo del nivel en el que se encuentre. Dichos cálculos representaron cierta dificultad de modelamiento, sin embargo al finalizar estas memorias de cálculo fue posible hacer un acercamiento a la necesidad del uso de las matemáticas coo una herramienta muy util para el quehacer cotidiano de un ingeniero.
- Ya adentrándonos en la programación, dentro de las dificultades más destacadas encontramos el interiorizar y acostumbrarse al cambio de la programación de software a hardware y las implicaciones que esto representan el cuanto a la secuencialidad de los diversos componentes. Además de esto, el planteamiento de los estados en el statemachine, como componente que controla en general muchas funciones a las que es necesario designar el papel principal con sus estados, salidas y su conexiones con los demás componentes como los registros del frogger y del backgrown. El registro de las vidas fue otra dificultad notable que requirió de tiempo para su ejecución exitosa.
- Dentro de todas estas dificultades mencionadas anteriormente se pudieron superar con éxito y con el propósito de adentrarnos en un acercamiento del funcionamiento de las diferentes herramientas que son usadas para el desarrollo de un proyecto de implementación hardware. La implementación de prescalers para

medir el tiempo de una manera diferente fue una solución bastante importante que permitió sobrellevar de una mejor manera contratiempos y dificultades con el movimiento de los registros.

- Al finalizar el proyecto de manera exitosa es posible afirmar que este proyecto fue de un aporte muy significativo para la comprensión de conceptos importantes vistos en la clase. Se concluyo exitosamente con el movimiento del frogger en todas las direcciones permitidas por los pulsadores del componente. También fue posible el correcto descuento de vidas después de las colisiones del restablecimiento de las mismas después de pasar el nivel. Se logro la interacción del juego con el jugador para comunicarle el numero de vidas disponibles, el numero del nivel al que el jugador pasaría después de la correcta llegada con los dos froggers a las casas y la comunicación de ganador después de haber cumplido satisfactoriamente con todos los niveles o de perdedor al haber perdido sus tres vidas en un nivel, además se cumplió exitosamente con el cambio de la velocidad y extensión de vehículos después de pasar cada nivel. Por esto, el proyecto funciona correctamente y según lo esperado y planteado inicialmente.

Trabajo Colaborativo

[Collapse]

PRODUCTO DE CALIDAD:Los estudiantes son capaces de cooperar y contribuir en el proceso de diseño para sopesar las utilidades y alcance del trabajo en equipo, en la obtención de metas comunes. a. Participación: todos los miembros del grupo contribuyeron por igual y asistieron a las reuniones programadas y cumplieron sus tareas designadas.

Breve descripción del trabajo colaborativo: A lo largo del proyecto de implementación del frogger se mantuvo una constante comunicación mediante distintas reuniones que se realizaron, se distribuyó el trabajo para cada uno de los integrantes, y se hacían actualizaciones de cómo iba cada integrante con su parte asignada. Como se puede observar en la parte del Cronograma de Actividades, cada integrante del grupo tuvo una parte de cada entrega, y entre todos realizamos algunas partes más complejas tales como la revisión del código HW y los diagramas que se realizaron en general, tanto en la parte de Arquitectura Hardware como el MacroAlgoritmo. Finalmente se puede concluir que hubo un buen trabajo en equipo que nos permitió sacar el proyecto adelante, explicarlo bien y lograr la funcionalidad total de la implementación que habíamos planeado desde un comienzo.

Inconvenientes encotrados: A lo largo de la entrega tuvimos que realizar diversos cambios de la implementación planeada que teníamos, dado que a lo largo que avanzabamos en el proyecto, encontrabamos que algunas cosas planteadas inicialmente no eran posibles de hacer, u otras se estaban pensando de otra forma completamente diferente a como estaban descritas en el lenguaje Hardware. Un gran inconveniente que provocó lo anteriormente mencionado, fue que como se mencionó en la sección de Logros y Problemas del Cronograma de Actividades, al principio entramos con bastantes fallas de comprensión del código Hardware lo cuál hizo que lo imaginaramos de una forma diferente y no tuvieramos muy claros ciertos conceptos importantes del lenguaje, haciendo que adicional a seguir avanzando con el programa, tuvieramos que hacer diversas modificaciones sobre las anteriores entregas constantemente. Sin embargo, con el paso del tiempo y más entrados en esta última entrega logramos salir adelante con todas las dudas que teníamos y comprender correctamente el funcionamiento del lenguaje.

Recomendaciones: Consideramos que seria una buena idea que se pudiera probar el proyecto por nuestra cuenta en algún momento disponible ya que de esta manera podemos controlar nuestro frogger e identificar los problemas producidos para luego llegar a tener dudas concretas. De esta manera sería posible avanzar mucho más rápido y tendríamos mayor seguridad para saber que lo que hicimos es correcto.

MATERIALES

Dispositivos Hardware

- Pulse button
  - Enlace documentación Descargar... (https://datasheet.octopart.com/G13KC-Alcoswitch-datasheet-8823464.pdf)
- FPGA
  - Enlace documentación Descargar... (https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_datasheet.pdf)
- 8x8 Module Matrix

- Enlace documentación Descargar... (http://www.farnell.com/datasheets/29075.pdf)

Dispositivos software

- Dispecto
  - Enlace documentación Descargar... (https://bloqueneon.uniandes.edu.co/content/enforced/24637-202110_IELE2210_01/docs/mod02/DiSpecTo-MANUAL-V08.pdf)
- Quartus:
  - Enlace documentación Descargar... (https://software.intel.com/content/www/us/en/develop/support.html)

BIBLIOGRAFIA

- DiSpecTo - Manual de Uso. - Jhoan Esteban León - Isabella Salgado Pinzón - F.Segura-Quijano (Descargar (https://bloqueneon.uniandes.edu.co/content/enforced/24637-202110_IELE2210_01/docs/mod02/DiSpecTo-MANUAL-V08.pdf))
- Intel® Developer Zone - intel (Descargar (https://software.intel.com/content/www/us/en/develop/support.html))
- Pushbutton Switches - Tyco electronics (Descargar (https://datasheet.octopart.com/G13KC-Alcoswitch-datasheet-8823464.pdf))
- 8 by 8 dot matrix LED displays with Cascadable Serial driver - Nexus machines (Descargar (http://www.farnell.com/datasheets/29075.pdf))
- Intel® MAX® 10 FPGA Device Datasheet - Intel (Descargar (https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/max-10/m10_datasheet.pdf))