Universidad Zaragoza

1542

Trabajo fin de grado

Generación automática de código de verificación en SystemVerilog

Autor
Lucas López Azcona
Ingeniería informática – Especialidad: ingeniería de computadores

Directores
Denis Navarro
Jesús Javier Resano Ezcaraz

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

24 – Noviembre – 2016

Declaración de autoría y originalidad

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Lucas López Azcona,

con nº de DNI 25192427 K en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Generación automática de la verificación en System Verilog de un banco de registros VHDL,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 24 de Noviembre de 2016

Fdo: Lucas López Azcona

Generación de la verificación en SystemVerilog de un banco de registros en VHDL

Resumen

Este proyecto ha sido realizado a partir de una propuesta de la empresa BSH que convocó dos becas para generar código automático de un diseño de bancos de registros hardware. Ambas becas parten de un mismo fichero CSV que define el sistema. La primera beca, realizada por Adrián Alcolea Moreno se encarga de generar el entorno de funcionamiento en VHDL del sistema. La segunda beca, la mía, se encarga de generar el entorno en SystemVerilog que verifique que el código en VHDL funcione correctamente. Además les va a permitir acercarse a un nuevo lenguaje de descripción y verificación de hardware como es SystemVerilog.

Para desarrollar el proyecto he utilizado los conocimientos de la asignatura "Procesadores de lenguaje" que me han servido para realizar la lectura del fichero CSV y trasladar su contenido a una estructura en memoria que me facilite realizar la generación de los ficheros.

Para generar los ficheros he necesitado adquirir conocimientos de SystemVerilog que he obtenido a partir de su estándar definido en el IEEE [1] además de realizar tutoriales online sobre los diversos componentes de SystemVerilog [2] y [3]. Por otro lado para adquirir conocimientos sobre la metodología de verificación y tratamiento de las aserciones que realiza SystemVerilog he leído dos libros proporcionados por el director del proyecto [4] y [5].

Con todas esas herramientas he creado un programa que lee el fichero CSV con las especificaciones del sistema, traslada este a un sistema de estructuras de memoria y a partir de estas estructuras generar varios ficheros que van a permitir realizar una verificación del sistema para comprobar que los ficheros generados en VHDL funcionan correctamente.

Índice

1 Introducción....1
2 Descripción del sistema a verificar....3
2.1 Banco de lectura....4
2.2 Banco de escritura....5
2.2.1 Ejemplo de funcionamiento del banco de escritura....6
3 Generación del traductor....7
3.1 Analizador léxico....7
3.2 Analizador sintáctico....8
3.3 Analizador semántico....9
4 Generación de los ficheros de verificación....10
4.1 El proceso de verificación....10
4.2 Banco de lectura simulado....12
4.3 Pruebas del banco de lectura....13
4.4 Banco de escritura simulado....14
4.5 Pruebas del banco de escritura....15
4.6 Fichero de cabecera....16
5 Pruebas realizadas....17
5.1 Pruebas de corrección del código generado....17
5.2 Pruebas de verificación del código VHDL....18
6 Conclusiones....19
7 Bibliografía....20
8 Anexos....21
8.1 Estandarización del fichero de especificaciones....21
8.1.1 Cabecera....21
8.1.2 Inicio de las especificaciones....21
8.1.3 Especificaciones de los bloques....21
8.1.4 Definición del bloque....21
8.1.5 Definición de los registros de los bloques de tipo lectura....22
8.1.6 Definición de los registros de los bloques de tipo escritura....22
8.1.7 Final de las especificaciones....23
8.1.8 Normas de estilo léxicas....23
8.2 Analizador semántico....24
8.2.1 Estructuras auxiliares....29
8.2.1.1 Clase Block....30
8.2.1.2 Clase Register....33
8.2.1.3 Clase Field....35
8.3 Contenido de los ficheros de verificación....38
8.3.1 read_bank_shadow.sv....38
8.3.2 read_bank_test.sv....39
8.3.3 write_bank_shadow.sv....41
8.3.4 write_bank_test.sv....45
8.3.5 definitions.sv....48
8.4 Traductor completo....53
8.5 Fichero de pruebas de corrección del código generado....60
8.6 Diagrama de Gantt....61

1 Introducción

BSH Hausgeräte GmbH es el mayor fabricante de electrodomésticos en Europa y una de las empresas líderes en el mundo. Con una cartera de 14 marcas, BSH cubre un amplio espectro de las necesidades de los consumidores. Entre las marcas internacionales que forman parte del grupo BSH se encuentran Bosch y Siemens y como marcas españolas se encuentran Balay y Ufesa. [6]

La división BSH-Balay colabora con la universidad de Zaragoza en proyectos de investigación y desarrollo de nuevos productos desde el año 1981. Desde entonces se han realizado 30 proyectos de investigación, 15 tesis doctorales y mas de 300 patentes sobre todo relacionadas con las cocinas de inducción. [7]
El pasado marzo convocó dos becas para realizar TFGs relacionados con un próximo desarrollo en el que están trabajando. En el proyecto intervienen dos bancos de registros, uno de lectura y otro de escritura. Cada banco tiene su funcionalidad particular que explicaré en el capítulo 2. Estos bancos están descritos en una hoja de cálculo que define los bloques que lo componen, los registros que forman parte de cada bloque y en qué campos están subdivididos estos registros. La composición del fichero de especificaciones se encuentra en el anexo 8.1.
Las dos becas parten de este fichero de especificaciones, una de ellas se encarga de realizar el código VHDL que describe el sistema definido en el fichero. Esta beca está realizada por Adrián Alcolea Moreno. VHDL es un lenguaje de descripción de hardware que se utiliza como entrada para las herramientas de diseño de circuitos integrados.
La segunda beca, la que estoy presentado en esta memoria, se encarga de generar un entorno en SystemVerilog que permita verificar que el código generado previamente en VHDL funciona de manera correcta.
Estas dos becas van a permitir ahorrar mucho tiempo ya que tanto la generación del código como la generación de la verificación son tareas tediosas y repetitivas que se pueden desarrollar de manera muy eficaz si se ejecutan de forma automática. También van a ahorrar dinero ya que una verificación previa de los proyectos evitará la construcción de chips defectuosos y en el ámbito de diseño de circuitos integrados la verificación automática es especialmente importante ya que aproximadamente el 90% del tiempo de desarrollo de un chip es debido al proceso de verificación.

Los objetivos de este proyecto son los siguientes:

- Crear un programa que a partir de un fichero de especificaciones genere un entorno de verificación en SystemVerilog para el sistema generado en VHDL a partir del mismo fichero de especificaciones.
- Definir junto con Denis Navarro, director del proyecto y representante de BSH-Balay, y Adrián Alcolea Moreno, encargado de realizar el código VHDL, este fichero de especificaciones, que estará representado en una hoja de cálculo, de forma estándar con la definición de los contenidos y tamaños de cada fila de manera que sea fácil de comprender por parte de un humano pero que también se pueda realizar una lectura automática por parte de una máquina sin necesidad de realizar un trabajo previo sobre el fichero.
- Como objetivo secundario también se analizará si merece la pena que BSH-Balay realice un cambio en el lenguaje de descripción de hardware que han utilizado hasta ahora, VHDL, y que empiecen a utilizar SystemVerilog.

Como metodología que he utilizado para llevar a cabo el proyecto se encuentra la siguiente:

- Adquirir conocimientos de SystemVerilog ya que es un lenguaje que desconozco. Este aprendizaje se ha realizado inicialmente a través de tutoriales online como asic-world [2] y doulos [3].
- Posteriormente con la base otorgada por estos tutoriales he leído los capítulos más interesantes para la realización del proyecto del estándar IEEE. Entre estos capítulos se encuentran:
    - 5. Lexical conventions.
    - 6. Data types.
    - 7. Aggregate data types.
    - 8. Classes.
    - 9. Processes.
    - 13. Tasks and functions.
    - 16. Assertions.
    - 23. Modules and hierarchy.
    - 26. Packages.
- Leer libros recomendados por el director del proyecto para conocer mas a fondo el tema de la metodología de verificación en SystemVerilog y las aserciones.
- Realizar diseños pequeños de SystemVerilog para poder adaptarme a su forma de describir y ejecutar procesos ya que es un lenguaje que se ejecuta en paralelo.
- Utilizar el simulador QuestaSim para probar las diferentes programas de prueba que he ido realizando y las diferentes partes del entorno de verificación.

A continuación muestro las diferentes tareas que he seguido para desarrollar el proyecto:

- Adquirir conocimientos sobre creación de programas en SystemVerilog y sus técnicas de verificación.
- Generar un entorno sencillo que me permita utilizar estos conocimientos.
- Consensuar las especificaciones del fichero de especificaciones.
- Realizar a mano la estructura de los diferentes ficheros del entorno de verificación para conocer las funciones a desarrollar y el número de ficheros a generar.
- Generar un traductor que me permita leer el fichero de especificaciones y trasladarlo a una estructura de datos en memoria.
- Crear un programa que lea esa estructura de memoria y la transcriba en los ficheros pensados previamente.
- Realizar pruebas que permitan generar entornos de verificación para varias configuraciones del fichero de especificaciones.

Esta memoria está formada por los siguientes capítulos:

- Capítulo 2: Descripción del sistema a verificar. Explica los dos bancos de registros en los que esta compuesto.
- Capítulo 3: Generación del traductor. Explica que pasos he ido realizando para conseguir el traductor que lea el fichero de especificaciones y lo transforme en una estructura en memoria.
- Capítulo 4: Generación de los ficheros de verificación. Explica los diferentes ficheros que van a ser generados tras ejecutar el programa y la utilidad de los bloques que los componen.
- Capítulo 5: Pruebas realizadas.
- Capítulo 6: Conclusiones.

2 Descripción del sistema a verificar

El documento en el que están escritas las especificaciones de los bancos de lectura y escritura se definió en varias reuniones con el compañero que realiza la otra beca y con el director del proyecto que es representantes de Balay. El resultado de las reuniones con las normas que tiene que cumplir el fichero de especificaciones se puede consultar en el anexo 8.1. A continuación pongo un ejemplo de fichero que servirá de ejemplo para el resto de la memoria:

```
      --INICIO_ESPECIFICACIONES

      bloque_lectura     RD     1
(1)   registro_1         fase_1 reloj_1 campo_1(8)  campo_2(8) #
(2)   registro_2         fase_1 reloj_2 campo_3(10) $(6)       #
(3)   registro_3         fase_2 reloj_1 campo_4(15) campo_5    #

      bloque_escritura   WR     2
(4,7) registro_1         fase_1 reloj_1 campo_1(8)  campo_2(8) # s111  u0
(5,8) registro_2         fase_1 reloj_2 $(10)       campo_3(6) # $(10) u33 24 48 SI
(6,9) registro_3         fase_2 reloj_2 campo_4(14) campo_5(2) # s0    b11

      --FIN_ESPECIFICACIONES
```

Tabla 1: Ejemplo de fichero de especificaciones

Este fichero describe tres bloques: uno de lectura, que será identificado en el sistema como bloque_lectura_0, y otro de escritura que se repite dos veces y cada uno tiene entidad propia, llamados bloque_escritura_0 y bloque_escritura_1. En el banco de lectura hay tres registros: el primero tiene dos campos de 8 bits, el segundo tiene un campo de 10 bits y el resto es vacío y el tercer registro tiene un campo de 15 bits y otro de 1 bit.
En los bancos de escritura hay tres registros similares al banco de lectura con el añadido de los valores de inicialización y rango. El primer registro esta compuesto por dos campos de 8 bits, el primero es un campo con signo y tiene un valor de inicialización de 111 mientras que el segundo es un campo sin signo con un valor de inicialización de 0. El segundo registro tiene un campo útil de 6 bits sin signo con un valor de inicialización de 33 y unos valores válidos que van del 24 al 48, el "SI" indica que cualquier valor que se salga de ese rango se limite al valor mas cercano. El tercer registro esta formado por dos campos: el primero es un campo con signo con valor de inicialización de 0 y el segundo es un campo de máscara de bits de 2 bits con un valor de inicialización de '11'.

El acceso en el sistema a los diferentes campos se realiza concatenando los nombres de bloque, registro y campo unidos por un punto o bien por las direcciones indicadas entre paréntesis. Ejemplos:

Dirección	Acceso
1	bloque_lectura_0.registro1.campo_1
5	bloque_escritura_0.registro2.campo_3
9	bloque_escritura_1.registro3.campo_5

2.1 Banco de lectura

El banco de lectura funciona como un multiplexor. Tiene como entradas todos los campos de los registros definidos dentro de un bloque de lectura. Como selector tiene de la entrada tiene una linea con el número de bits suficientes para representar las direcciones útiles. En el ejemplo serían 4 bits por que hay 9 direcciones, indicadas entre paréntesis, ya que el bloque de escritura se repite 2 veces. Este selector indicará la dirección del registro que se quiere obtener como salida. Esta salida se obtendrá al generar el registro seleccionado tal y como esta representado en el fichero de especificaciones, concatenando los campos pertenecientes al registro y añadiendo ceros en los campos vacíos ($).

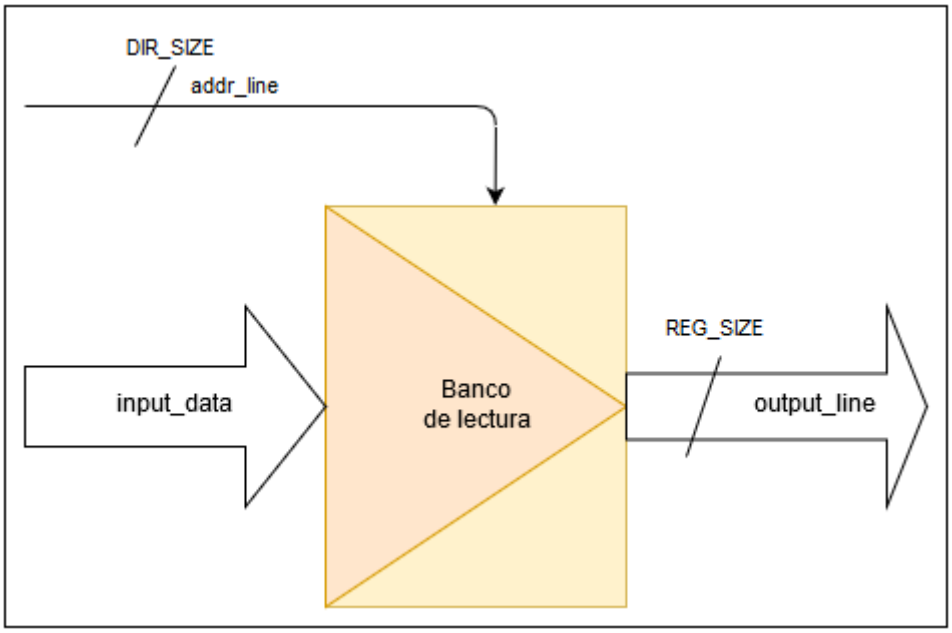

Figura 1: Diagrama del banco de lectura

2.2 Banco de escritura

El banco de escritura funciona como una memoria donde se van almacenando los registros deseados para luego obtener un flujo de bits con todos los registros pertenecientes a los bloques de escritura. Esta formado por dos almacenes internos: uno temporal donde se guardan los registros de entrada y otro de salida donde se vuelcan los registros desde el temporal.
El funcionamiento de carga de registros en el almacén temporal es el siguiente. Si 'RESET_N' está activado (valor 0) se carga el valor de inicialización en todos los campos de los registros del almacén. Si 'RESET_N' está deshabilitado entonces cada tick del reloj principal 'CLK' se verifica si esta activo el bit 'ENABLE'. Si está activo entonces se carga en la dirección indicada por la linea 'addr_line' del almacén temporal el valor procedente de la linea 'input_line'. Si el valor de 'ENABLE' es cero no se modifica ningún valor del almacén temporal.
El almacén de salida esta cortocircuitado con la salida del banco de escritura por lo que la salida siempre es coherente con el contenido de este almacén. El funcionamiento es similar al almacén temporal pero cambiando el reloj principal y el 'ENABLE' por el reloj y la fase asociados a los registros. Estos están definidos en el fichero de especificaciones. Si 'RESET_N' está activado se carga el valor de inicialización en todos los campos de los registros del almacén. Si 'RESET_N' esta deshabilitado entonces cada flanco positivo del reloj asociado a cada registro se verifica si esta activa la fase asociada al registro. Si está activa entonces se carga en el almacén de salida el registro correspondiente del almacén temporal.

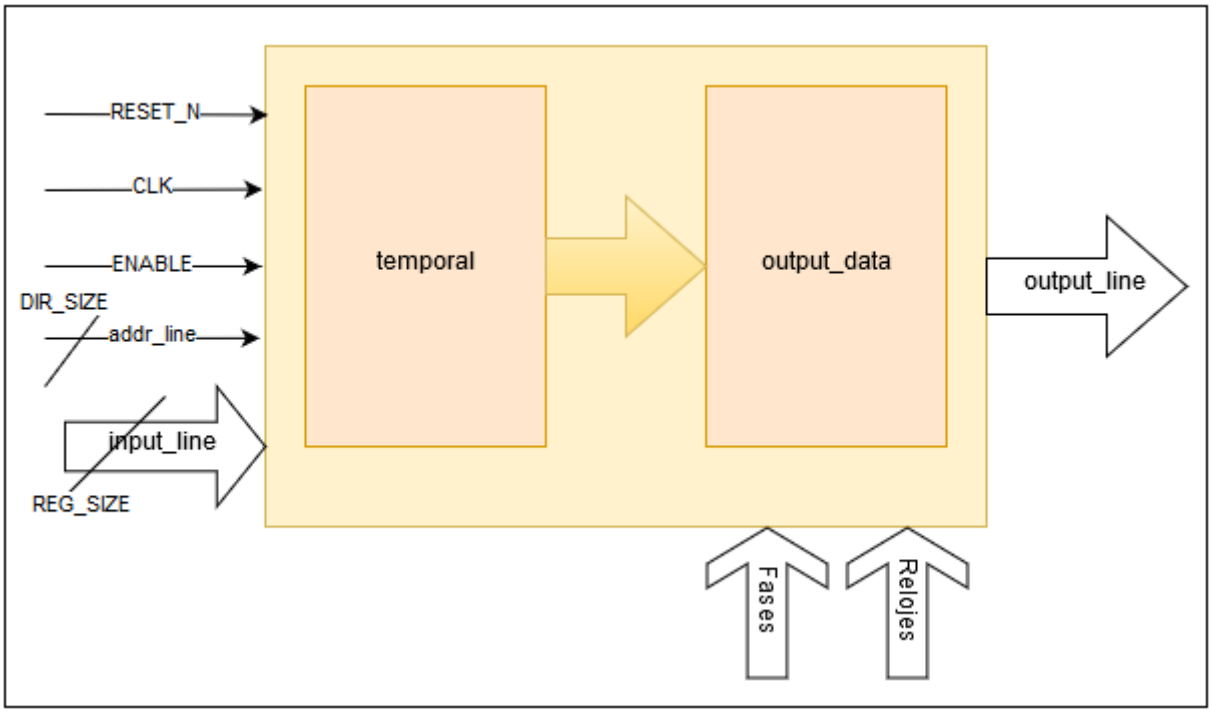

Figura 2: Diagrama del banco de escritura

2.2.1 Ejemplo de funcionamiento del banco de escritura

En la siguiente figura se muestra en detalle el funcionamiento de la carga de los distintos almacenes del banco de escritura. En el segmento superior muestro la carga del almacén temporal. Para que esta se produzca se dan las condiciones necesarias en los instantes 3, 5, 7, 9 pero en el instante 3 el 'reset' esta activado por lo tanto no se produce la actualización del almacén temporal.
En el segmento inferior se muestra la carga en el almacén de salida a partir de los valores del almacén temporal. Mientras esta el 'reset' activo se da la condición para cargar el valor del registro_1 pero el 'reset' lo impide, instante 2. Cuando termina el 'reset', instante 4, tanto el reloj_1 como el reloj_2 presentan un flanco de subida y tanto la fase_1 como la fase_2 están activas se actualizan los valores de todos los registros. En el siguiente flanco del reloj_1, instante 6, esta activa la fase_1 y se actualiza el registro_1. En el instante 8 hay un flanco de subida para ambos relojes pero solo esta activa la fase_2 por lo tanto se actualiza el registro_3 pero no el registro_1 ni el registro_2 ya que dependen de la fase_1.

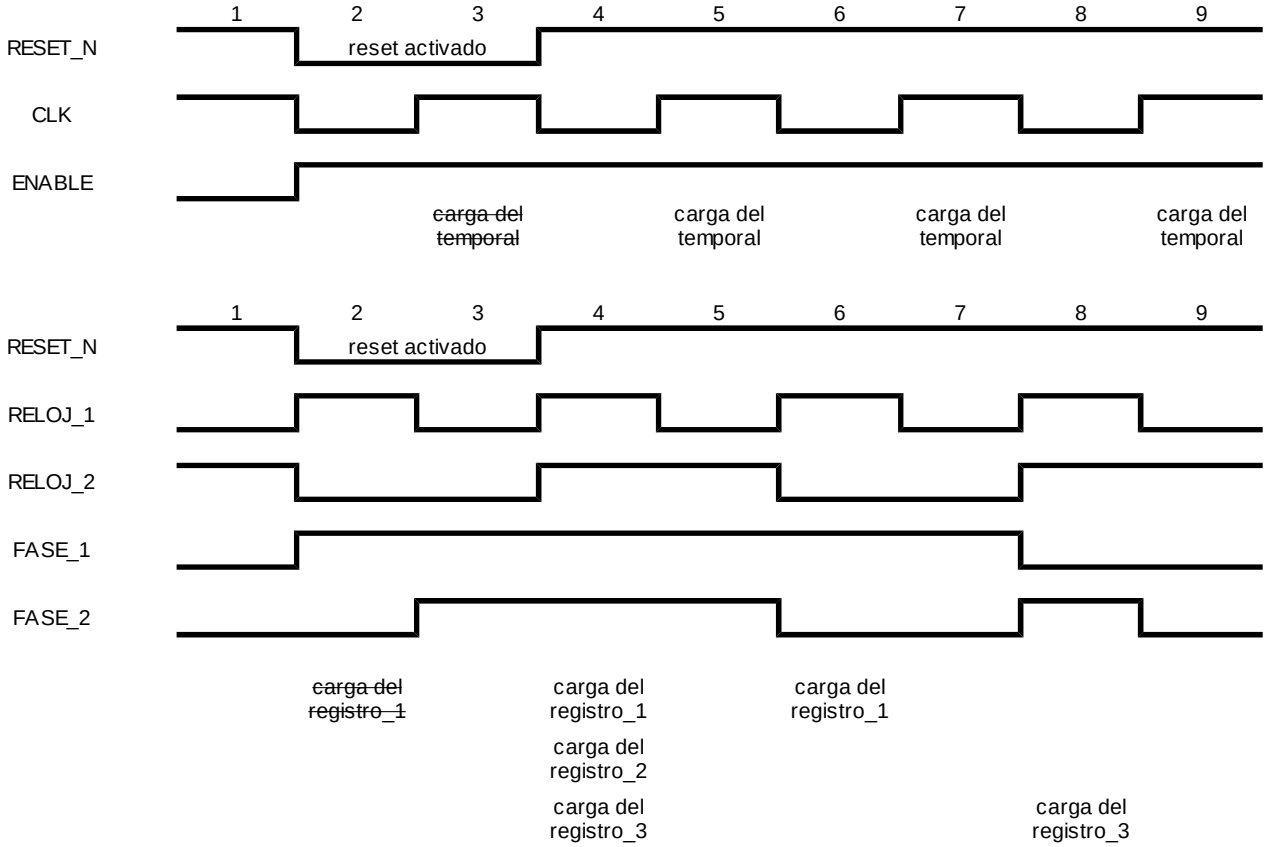

Figura 3: Ejemplo de funcionamiento del banco de escritura

3 Generación del traductor

Tras la definición del fichero de especificaciones el siguiente paso es traducir ese contenido a contenido digital para poder trabajar de manera mas rápida ya que son necesarios varios accesos para generar los ficheros de verificación.
Para realizar esta traducción utilicé una herramienta ya vista en la asignatura "Procesadores de lenguajes" que es el "Java Compiler Compiler" o JavaCC. Esta herramienta permite crear compiladores o traductores a través de la definición de una gramática BNF que describa el texto a analizar y posteriormente genera automáticamente un programa Java que reconoce esa gramática. Además de la gramática hay que añadir un analizador semántico que genere los ficheros de salida deseados. Para crear el traductor en JavaCC hay que definir tres analizadores:

3.1 Analizador léxico

Su trabajo es leer la entrada, en este caso un fichero, carácter a carácter y emparejar los caracteres encontrados con alguna de las reglas definidas. Estas reglas generan los 'tokens' que serán enviados a los otros dos analizadores. Por lo tanto el primer paso para generar el traductor es proporcionar las reglas que permitan generar los 'tokens'.
En JavaCC hay varios tipos de 'tokens' que permiten generar complejas reglas anidadas. En el proyecto he utilizado unicamente dos de ellas, SKIP y TOKEN.
SKIP permite ignorar caracteres mientras va leyendo de la entrada, esta regla se utiliza para ignorar elementos como espacios en blanco, tabuladores o saltos de linea en la generación de compiladores, en el proyecto la he utilizado para ignorar el carácter "," ya que el fichero de entrada es un CSV y cada campo de la hoja de cálculo esta separado por este carácter. El resultado de la regla es el siguiente:

```
SKIP:
{
   < COMMA:          "," >
}
```

TOKEN es el núcleo del analizador léxico y permite definir las reglas que generan las cadenas que se van a utilizar. Para generar estas reglas he analizado el documento de especificaciones y he obtenido las siguientes:

```
TOKEN:   /* PALABRAS RESERVADAS */
{
   < BEGIN:          ( ~[] )* "--INICIO_ESPECIFICACIONES" ( ~["\n"] )* >
|  < END:            "--FIN_ESPECIFICACIONES" >

|  < READ:           "RD" >
|  < WRITE:          "WR" >
|  < YES:            "SI" >
|  < NO:             "NO" >

|  < DOLLAR:         "$" >
|  < SEPARATOR:      "#" >
|  < LPAR:           "(" >
|  < RPAR:           ")" >
|  < EOL:            ("\r\n" | "\r" | "\n") >
```

```
|  < #LETTER:        (["a"-"z"]) >
|  < #DIGIT:         (["0"-"9"]) >
|  < INTEGER:        (["-"])? (<DIGIT>)+ >
|  < BIT_NUMBER:     "B" (<DIGIT>)+ >
|  < SIG_NUMBER:     "S" (<INTEGER>) >
|  < USG_NUMBER:     "U" (<DIGIT>)+ >
|  < ID:             (<LETTER> | (<LETTER> | "_") (<LETTER> | <DIGIT> | "_")+) >
}
```

3.2 Analizador sintáctico

Recibe los 'tokens' del analizador léxico y evalúa si el 'token' puede formar parte de la regla activa ya sea porque tiene una disyuntiva y es el comienzo de alguna de las reglas a elegir o porque hay una única rama y coincide con el siguiente elemento. Si el 'token' no concuerda con ninguna opción de las esperadas dará un error. Como el fichero es previamente verificado no genero rutinas de gestión de errores ya que no dará ningún error sintáctico.

Para generar el analizador sintáctico hay que generar primero la gramática que describa el comportamiento descrito en el fichero de especificaciones. La forma mas sencilla de describir esta gramática es en la forma BNF. [8] La gramática obtenida a partir del análisis de la descripción del fichero de especificaciones es la siguiente:

```
input ::=
   BEGIN { EOL } { register_block { EOL } } END

register_block ::=
   ID
   [
      READ INTEGER EOL { read_register_definition }
   |
      WRITE INTEGER EOL { write_register_definition }
   ]

read_register_definition ::=
      ID ID ID { fields_definition } SEPARATOR EOL

write_register_definition ::=
      ID ID ID { fields_definition } SEPARATOR { values_definition }
      ( INTEGER INTEGER [ YES | NO ] )
      EOL

fields_definition ::=
   [
      ID ( LPAR INTEGER RPAR )
   |
      DOLLAR ( LPAR INTEGER RPAR )
   ]

values_definition ::=
   [
      BIT_NUMBER
   |
      SIG_NUMBER
   |
      USG_NUMBER
   |
      DOLLAR ( LPAR INTEGER RPAR )
   ]
```

Notación utilizada:

- Los elementos entre ( ) son de elección opcional.
- Los elementos entre [ ] son de elección única. Hay que elegir uno y solo uno de los elementos que están separados por una barra vertical, "|".
- Los elementos entre { } son de elección múltiple. Se pueden repetir 1 o mas veces.
- Los elementos en MAYÚSCULAS son 'tokens'.
- Los elementos en minúsculas son no terminales.

3.3 Analizador semántico

Recibe los 'tokens' del analizador sintáctico y se encarga de producir la estructura de datos que contendrá el fichero CSV en memoria lo que permitirá un acceso mas rápido a la hora de generar los ficheros de verificación.
Este analizador estará integrado dentro del analizador sintáctico y entrará en funcionamiento mientras se estén verificando las distintas reglas de la gramática. Irá transformando los distintos 'tokens' en las variables necesarias para ir creando los diferentes elementos, ya sean los nombres de los bloques, registros, campos, fases y relojes como los valores numéricos tanto de inicialización como de rango de valores mínimos y máximos.

Debido a su extensión el resultado de los analizadores sintáctico y semántico completo se encuentra en el anexo 8.2.

4 Generación de los ficheros de verificación

4.1 El proceso de verificación

Como ya he comentado este proyecto tiene un proyecto complementario que es el va a crear un programa que generará automáticamente un diseño hardware en VHDL de un sistema de registros que sería procesado posteriormente para generar un chip real. Estos chips tienen que ser sometidos a diversas pruebas antes de realizar la producción a gran escala ya que las producciones a gran escala requieren de grandes inversiones iniciales para iniciar la fabricación de las obleas que posteriormente darán los chips. Estas pruebas se aseguran de que el chip se comporta de forma que cumpla con las especificaciones.
Por otro lado, en este proyecto tengo que generar también automáticamente un entorno en SystemVerilog que verificara el diseño hardware generado anteriormente. Ambos programas han sido diseñados por dos personas distintas que solo comparten las especificaciones de entrada para evitar que un diseño vago pudiera incurrir en errores solapados en ambos programas al reutilizar el mismo código en las dos partes.

A la hora de construir el entorno de verificación he contemplado dos escenarios:
El primer escenario consiste en crear una tabla con los valores de entrada a proporcionar y los valores de salida esperados. Pero este escenario no permite realizar comprobaciones sobre las conexiones entre los registros del sistema y verificar que no existen cortocircuitos entre los distintos registros e incluso en el interior de un registro.
El otro escenario consiste en crear un sistema idéntico al que se ha generado en VHDL pero con la ventaja de que es software y por lo tanto mas fácil de comprobar que funciona de manera correcta. Este escenario además permite comprobar que solo se han alterado los elementos deseados ya que se puede hacer un recorrido por todos los registros y comprobar sus valores temporales. Esto era imposible de realizar con la tabla de valores.

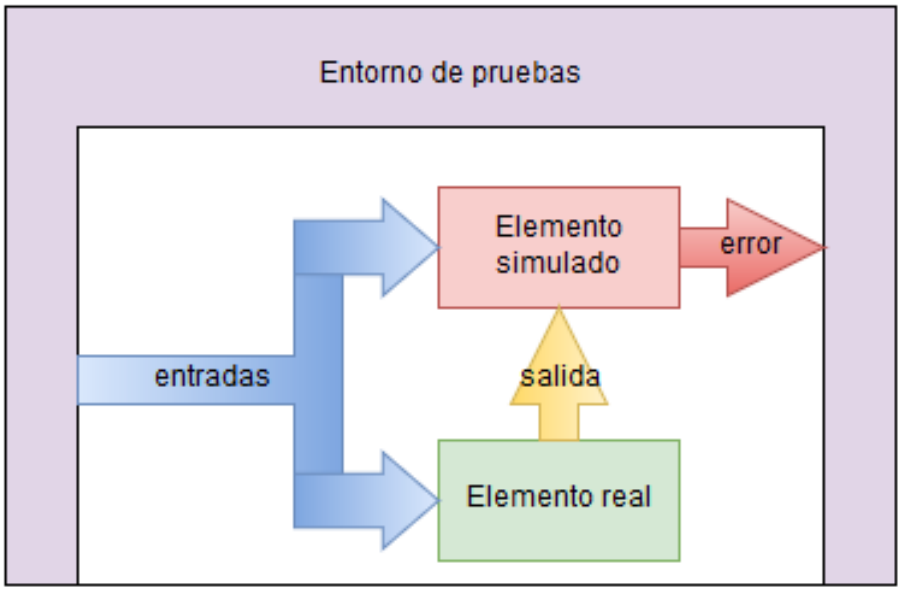

Figura 4: Diagrama del proceso de verificación

SystemVerilog tiene varias formas de definir bloques de código, entre todas ellas solamente he utilizado dos:

- Módulos (modules): Es el bloque de construcción básico de un diseño en SystemVerilog. El principal propósito de un módulo es encapsular los datos, funcionalidad y temporización de objetos 'hardware'. [1]
  Lo he utilizado para representar los bancos de lectura y escritura y los programas de prueba de los respectivos bancos.
- Paquetes (packages): Los paquetes SystemVerilog proveen al sistema de un mecanismo adicional para compartir parámetros, valores, tipos, tareas, funciones, secuencias, propiedades y declaraciones entre múltiples módulos, interfaces, programas y 'checkers'. [1]
  Lo he utilizado para generar un fichero con los elementos comunes a los dos bancos como el tamaño de los registros o la definición de las clases que definen los diferentes campos. En las siguientes subsecciones muestro los diferentes componentes del entorno de verificación utilizando como ejemplo el fichero de prueba que he indicado en el capítulo 2.

Otros elementos que también van a salir de forma reiterada son los siguientes:

- Funciones: Una función en SystemVerilog es como en cualquier otro lenguaje. Puede recibir parámetros en su invocación y puede devolver un elemento cuando acaba su ejecución. Puede invocar a otras funciones pero no puede llamar a una tarea o entrar en modo de espera. Tiene que ejecutarse sin interrupciones.
- Tarea: Es similar a la función pero no puede devolver ningún valor directo al terminar la ejecución, si que puede cambiar el valor de algún parámetro que haya sido definido como elemento de salida o entrada-salida. Puede invocar cualquier función o tarea y puede entrar en modo de espera o sincronización.
- Procesos: En SystemVerilog los módulos están divididos en diferentes bloques denominados procesos. Estos procesos pueden ser de tres tipos:
  - Initial: Estos bloque se ejecutan una única vez al ser iniciado el módulo. Se utilizan para inicializar las variables y demás elementos del módulo y para dar comienzo a los entornos de pruebas como he utilizado en este caso.
  - Always: Estos bloque se activan al iniciar el módulo y se ejecutan cuando se cumple la condición que tienen definida. Una vez terminada la ejecución se vuelven a activar y esperan la siguiente vez que se cumpla la definición.
  - Final: Estos bloque se ejecutan una única vez al terminar el módulo. Se utilizan para mostrar posibles resúmenes de la ejecución del módulo.

4.2 Banco de lectura simulado

Está contenido en el módulo “read_bank_shadow” en el fichero “read_bank_shadow.sv” y contiene la especificación del funcionamiento del banco de lectura del sistema en versión software. Lo utilizo para simular el comportamiento del banco de lectura y comparar el resultado obtenido por el banco de lectura real ‘hardware’ y de esta forma verificar el funcionamiento correcto. El proceso de verificación consiste en
El banco de lectura hardware funciona como un multiplexor que recibe todas las entradas indicadas por los registros en el fichero CSV y la dirección de la que se desea obtener la salida. Si la dirección de salida no concuerda con ninguna de las direcciones correspondientes a un registro de un banco de lectura la salida es indiferente.

El módulo tiene como entradas la dirección, los datos de entrada y la salida generada por el banco de lectura real. Esta compuesto por dos procesos: uno para actualizar el valor de salida y otro para comparar los valores de salida del simulador y del entorno real.
El proceso que verifica cada ciclo de simulación el valor generado en el banco de lectura real, ‘output_line’, con el valor generado en el simulador, ‘output_data’ muestra la siguiente información si se da un error.

```
# ESTABLECIENDO VALOR: '01'
# ** Warning: RB @: 1, NAME: read_bank.bloque_lectura_0.registro_1,
     DATA: 21845, LINE: 1
#    Time: 46 ns  Scope: read_bank_test.rbs.value_check File:
     /home/lucas/examples/BSH_V0/read_bank_shadow.sv Line: 72
```

En el mensaje se puede identificar la dirección del registro, su nombre y los valores que han proporcionado el banco simulado, valor DATA, y el banco real, valor LINE. A priori no se puede saber de donde proviene el error. Por lo tanto se tendrá que buscar el fallo a mano. Primero comparando estos datos con el fichero de especificaciones para comprobar si la dirección y el nombre concuerdan. Y si esto es correcto entonces analizar el fichero que describe el banco real para buscar el fallo en la definición del registro que ha dado el fallo.

El contenido completo de este fichero está en el anexo 8.3.1.

4.3 Pruebas del banco de lectura

Está contenido en el módulo “read_bank_test” en el fichero “read_bank_test.sv”.
La prueba consiste en asignar un valor inicial a todos los registros, recorrer todos los registros uno a uno estableciendo el valor complementario y cambiando la dirección de entrada a todas las direcciones asociadas a registros de lectura para observar que la salida es correcta y luego volver a poner el valor inicial en el registro. Esto permite comprobar que no se han generado cortocircuitos en la producción del banco de lectura real y que todas las líneas funcionan correctamente ya que se muestrean con ambos valores, tanto el 0 como el 1.

Este módulo no contiene cabecera ya que no va a ser invocado desde fuera, va a ser instanciado como elemento inicial en el simulador. Si contiene las variables que se van a utilizar como entrada en el banco de lectura real y en el simulado: ‘addr_line’, ‘input_data’ y ‘output_line’.
Está compuesto por tres tareas. La primera, “set_value”, asigna un valor a todos los campos de los registros de los bancos de lectura real y simulado. La siguiente tarea, “change_address”, no se genera automáticamente, es única para todos los sistemas. Y se encarga de actualizar la dirección del registro activo para poder evaluar todos los registros. La última tarea, “update_value”, es la que actualiza el valor registro a registro y cambia las direcciones utilizando la tarea anterior recorriendo todas las correspondientes al banco de lectura.

El último elemento es un proceso que sirve de punto de entrada para programa de prueba e inicializa el banco de lectura simulado y efectúa las pruebas. El proceso de verificación explicado de forma genérica en el capítulo 4.1 y consiste en este caso en cargar un valor inicial en todos los registros, tarea “set_value”. Para luego actualizar registro a registro el valor complementario, tarea “update_value”. Tras cada actualización fuerzo al banco de lectura a que muestre el valor de todos los registros pertenecientes al banco, tarea “change_address”.
Por petición del director repito las pruebas dos veces, la primera vez cargo el valor ’0101...01’ y actualizo con el valor ’1010...10’ y la segunda cargo el valor ’1010...10’ y actualizo los campos con el valor ’0101...01’. Estos valores se han tomado para comprobar que no existe ningún cortocircuito interno en los registros ya que si utilizara los valores ’0000..00’ y ’1111..11’ se podría dar el caso de que dos bits contiguos estén cortocircuitados y no habría forma de notificar ese error.

El contenido completo de este fichero está en el anexo 8.3.2.

4.4 Banco de escritura simulado

Está contenido en el módulo “write_bank_shadow” en el fichero “write_bank_shadow.sv” y contiene la especificación del funcionamiento del banco de escritura del sistema en versión software. Lo utilizo para simular el comportamiento del banco de escritura y comparar el resultado obtenido por el banco de escritura real ‘hardware’ y de esta forma verificar el funcionamiento correcto. Como no se puede acceder al contenido del fichero mientras se realizan las pruebas únicamente puedo comprobar que el banco funciona bien evaluando las salidas y no evaluando el contenido de los dos almacenes en todo momento.

El módulo tiene como entradas los relojes de los registros y el de sistema, las fases de los registros y el ‘enable’, el ‘reset_n’, la dirección, los datos de entrada y la salida generada por el banco de escritura real. Esta compuesto por cuatro procesos fijos: uno para la instanciación de los campos de los almacenes, otro para la realizar la inicialización de los registros cuando se activa ‘reset_n’, otro para gestionar la carga del almacén temporal y el último para la verificación del vector de salida del banco real y el simulado. A estos procesos hay que añadir uno mas por cada reloj propio de los registros que detecta los flancos de estos relojes y realiza la carga de los registros si se cumplen las condiciones especificadas.
El proceso que verifica cada ciclo de simulación el valor generado en el banco de escritura real con el valor generado en el simulador muestra la siguiente información si se da un error.

```
# HABILITA LA FASE 'FASE_1' Y EL RELOJ 'RELOJ_1'.
# ** Warning: WR @: 4, NAME: write_bank.bloque_escritura.registro_1.campo_1,
     TMP: 85, DATA: 85, LINE: 170.
#    Time: 352 ns  Scope: write_bank_test.wbs.value_check File:
/home/lucas/examples/BSH_V0/write_bank_shadow.sv Line: 1086
```

En el mensaje se puede identificar la dirección del registro, su nombre y los valores que han proporcionado los almacenes simulados, valores TMP y DATA, y el banco real, valor LINE. Debido a que puede haber diferentes causas de fallo en el banco de escritura también se proporcionan en el entorno de pruebas del banco de escritura las indicaciones de cambios de fases y relojes. Tal y como ocurría en el banco de lectura esta información no indica que elemento del sistema es el que ha producido el error y hay que buscarlo a mano a partir de esta información. Primero comprobando si la definición indicada en el mensaje de error concuerda con el fichero de especificaciones y después analizando el código del que define el banco real.

El contenido completo de este fichero está en el anexo 8.3.3.

4.5 Pruebas del banco de escritura

Está contenido en el módulo “write_bank_test” en el fichero “write_bank_test.sv”.
La prueba consta de cuatro fases, combinando todos los valores posibles de ‘reset_n’ y ‘enable’. En las dos primeras ‘reset_n’ esta activado y por lo tanto el banco esta deshabilitado. En las tercera tanto ‘reset_n’ como ‘enable’ están desactivados. En la cuarta ‘reset_n’ esta desactivado y ‘enable’ está activado. Por lo tanto las tres primeras pruebas sirven para comprobar que el banco real funciona siguiendo las especificaciones ya que no debe dar ninguna salida distinta del valor de inicialización de los distintos campos. En la última fase los dos bancos trabajan de manera normal y se comprueba que los registros están generados de forma correcta. La prueba realizada es conceptualmente similar a la realizada en el banco de lectura, se carga un valor en el almacén temporal y se actualiza registro a registro con el valor complementario. Después se activan una a una todas las combinaciones encontradas en el fichero de especificaciones de fases y relojes y se comprueba que solo se actualizan los registros definidos con esa combinación.

Este módulo no contiene cabecera ya que no va a ser invocado desde fuera, va a ser instanciado como elemento inicial en el simulador. Si contienen las variables que se van a utilizar como entrada en el banco de escritura real y en el simulado.
Está compuesto por dos tareas y un proceso. La primera tarea, “rise_and_fall”, se encarga de realizar las activaciones de todas las combinaciones de fase y relojes que hay en el CSV. Los valores cambian solo cuando hay un flanco negativo del reloj del sistema para que la entrada sea estable cuando llegue el flanco positivo. Se van mostrando por pantalla los mensajes de que fase y reloj se van activando. Si se produce un error se puede saber si el registro que ha dado el error se ha activado en el momento correcto o no. La segunda tarea, “do_test”, se encarga de realizar la batería de pruebas, cargar el valor inicial en el almacén temporal y actualizarlo con el valor complementario. El elemento restante es el proceso que ejecuta las cuatro fases definidas anteriormente y va invocando la batería de pruebas en cada una de las fases.

El contenido completo de este fichero está en el anexo 8.3.4.

4.6 Fichero de cabecera

Para terminar con la descripción de los ficheros únicamente queda el fichero que contiene los elementos comunes a los ficheros de verificación, el fichero es “definitions.sv" y contiene el paquete "definitions". Este paquete contiene los siguientes elementos:

- Tamaños de la linea de direcciones y la linea de datos.
- Tipos que definen la linea de direcciones y la linea de datos.
- Valores complementarios para realizar las pruebas.
- Listas de las direcciones de los registros pertenecientes a cada banco.
- Función que traduce las direcciones en el nombre de registro asociado.
- Funciones que calculan el máximo y el mínimo dependiendo del tamaño del campo en bits y del tipo, 'signed' o 'unsigned'.
- Definiciones de las clases que describen los campos de tipo signed y unsigned.
- Definición de los bloques de lectura que componen el CSV.
- Definición de los bloques de escritura que componen el CSV.
- Instanciación de los bloques dentro de la definición de los bancos de lectura y escritura.
- Tareas que inicializan los campos de las estructuras que forman los bancos de lectura y escritura.

El contenido completo de este fichero está en el anexo 8.3.5.

5 Pruebas realizadas

Dado que el programa va a generar ficheros de forma automática que verifiquen el funcionamiento de otro código he tenido que realizar dos tipos de pruebas. En primer lugar he ejecutado pruebas para analizar el correcto funcionamiento del programa y que genera de forma correcta los ficheros y en segundo lugar que el entorno de verificación funciona correctamente y genera una salida acorde con el sistema introducido, ya fuera con errores o sin ellos.

5.1 Pruebas de corrección del código generado

Para realizar la comprobación del código generado he utilizado el programa para generar entornos con distintas configuraciones para luego analizar los ficheros creados de forma manual.
Para comprobar la correcta generación de los ficheros generé un fichero de especificaciones con todas las posibles características de los registros aunque no todas las posibles combinaciones de campos posibles ya que estas combinaciones dan mas de 2^{16} posibilidades. Debido a que los registros del banco de lectura solo difieren en la composición de los campos lo utilicé para probar diferentes combinaciones de estos: varios campos útiles seguidos, varios campos vacíos seguidos, ambos campos útiles y vacíos intercalados, etc. Mientras que para los registros del banco de escritura me centré mas en probar diferentes combinaciones de relojes, fases y los tres tipos de valores contenidos en los campos, 'signed', 'unsigned' y máscara de bits. El contenido de este fichero se encuentra en el anexo 8.5.

Una vez comprobado que el programa generaba de forma correcta todas las posibles combinaciones de elementos presentes realicé tres programas que generaban ficheros de especificaciones con diferentes configuraciones: solo banco de lectura, solo banco de escritura y ambos bancos. Utilicé estos programas para crear ficheros con un número de registros creciente. Primero con un número pequeño de registros y duplicando este número de forma constante. Inicialmente estas pruebas me sirvieron para detectar un error del programa que no está incluido en las especificaciones y es la posibilidad de generar entornos de pruebas para sistemas que solo incluyen uno de los dos bancos. También me sirvieron para realizar una tabla con el tiempo invertido por el programa para generar cada salida. Véase tabla 2.

La última prueba consistió en la carga de los ficheros generados en las pruebas anteriores en el simulador QuestaSim y comprobar que producía los valores esperados. Dado que el diseño hardware no estaba presente decidí que se mostrase la salida que iba generando cada iteración del entorno de pruebas.

Numero de registros totales	Tiempo invertido (s)		
	Solo banco de lectura	Solo banco de escritura	Ambos bancos
128	0,5	1	0,8
256	1,5	3,5	2
512	6	14	8
1024	25	64	37
2048	115	310	175
4096	530	1400	825
8192	2400 (40m)	6600 (1h50m)	3450 (≈1h)
16384	10350 (≈3h)	≈30000 (≈8h)	15625 (4h20m)

Tabla 2: Coste en tiempo de generación

El número de registros indicado en cada fila es el número total de registros en el documento, no el número de registros por banco. Cuando realizo las pruebas con ambos bancos estos se reparten los registros indicados, la mitad en cada uno.

5.2 Pruebas de verificación del código VHDL

Para realizar estas pruebas conté con un fichero real proporcionado por la empresa.
En primer lugar generé el sistema VHDL con el programa creado en el otro proyecto y generé el entorno de verificación con mi proyecto para comprobar que se podían realizar todas las conexiones desde el entorno de pruebas al sistema VHDL. Estas pruebas señalaron una diferencia entra ambos lenguajes de definición de hardware y es el tratamiento de las mayúsculas, mientras que en VHDL no se tienen en cuenta las mayúsculas en SystemVerilog si que se tienen en cuenta y esto provocó un reajuste de los nombres de los bloques, registros y campos para transformar todos los caracteres a minúsculas.
Tras solventar este problema se realizó la primera carga del sistema completo en el simulador y se verificó la salida generada para un sistema correcto. Esta salida fue de acuerdo a lo esperado ya que no produjo ningún aviso de error. Posteriormente se realizaron cambios sobre el fichero de especificaciones para generar a mano errores en los diferentes registros.
En el banco de lectura los únicos errores que se pueden provocar son el de una dirección incorrecta en el registro o la alteración de los campos que lo componen. Ambos errores fueron generados y un ejemplo de la salida de este error se puede revisar en el capítulo 4.2.
En el banco de escritura se pueden provocar los siguientes errores:

- Error en la dirección de un registro.
- Error en los campos que componen un registro.
- Error en la configuración de los campos: tipo, valor de inicialización y rango de valores.
- Error en el reloj o el tipo correspondiente al registro.

De la misma forma que en el banco de lectura se generaron todos los tipos de errores y se comprobó que el sistema mostraba un aviso de error en los momentos en que el sistema real y el simulado contenían valores diferentes. Hay un ejemplo de la salida de error en el capítulo 4.4.

6 Conclusiones

El trabajo me ha servido para conocer un nuevo lenguaje de descripción de hardware relativamente moderno que utiliza conceptos de los lenguajes de programación actuales como la orientación a objetos e introduce un concepto nuevo en los lenguajes de verificación que son las aserciones dinámicas. Las aserciones son comprobaciones que se evalúan siempre a verdadero o en caso contrario dan un error. Se utilizan para comprobar la correcta ejecución de un programa. En otros lenguajes las aserciones son estáticas, es decir que simplemente comprueban la veracidad de la aserción en un instante dado, pero en SystemVerilog esta comprobación se puede alargar en el tiempo a través de varios ciclos de simulación para delimitar el tiempo de ejecución de una tarea o para comprobar que se cumple con la secuencia de funcionamiento establecido. Aunque no las he tenido que utilizar en este proyecto y me ha bastado con las aserciones estáticas para comprobar que los valores producidos por los bancos real y simulado coincidían.

También me ha servido para tener un primer contacto con la forma de trabajar de las empresas a la hora de tener que realizar un documento con la estandarización del documento de especificaciones y de producir un proyecto cerrado con unos requisitos fijos que tiene que funcionar perfectamente.

Como resumen del trabajo realizado tengo que comentar que me ha llevado mas tiempo del que había valorado en un primer momento ya que he tardado 5 meses en realizarlo, aunque no a tiempo completo, cuando esperaba tenerlo terminado en 3 meses. Como valoración de los objetivos cumplidos estos se han realizado por encima de los requisitos indicados ya que he habilitado la opción de que el programa trabaje con ficheros de especificaciones con un tamaño de registros superior al indicado en los requisitos que es de 16 bits.

7 Bibliografía

[1] Documento del estándar del SystemVerilog del IEEE.
- https://standards.ieee.org/getieee/1800/download/1800-2012.pdf

[2] Tutoriales de SystemVerilog en Asic-World.
- http://www.asic-world.com/systemverilog/tutorial.html

[3] Tutoriales de SystemVerilog en Doulos.
- https://www.doulos.com/knowhow/sysverilog/tutorial/

[4] Bergeron, J.; Cerny, E.; Hunter, A. y Nightingale, A. (2006).
Verification methodology manual for SystemVerilog. Springer

[5] Vijayaraghavan S. y Ramanathan, M. (2005).
A practical guide for SystemVerilog assertions. Springer

[6] Página web del grupo BSH.
- https://www.bsh-group.com/index.php?page=100325

[7] Página web de la cátedra BSH de la universidad de Zaragoza
- http://www.catedrabsh-uz.es/historia/index.php

[8] Descripción de la gramática BNF.
- https://en.wikipedia.org/wiki/Backus–Naur_form

8 Anexos

8.1 Estandarización del fichero de especificaciones

8.1.1 Cabecera

Las primeras lineas de la hoja de cálculo se utilizarán para incluir comentarios que indican el significado de las columnas o cualquier otra cosa que no se tendrá en cuenta para generar los ficheros de verificación.

8.1.2 Inicio de las especificaciones

Estos comentarios se extenderán hasta una linea que contendrá en la primera celda la cadena: "--INICIO_ESPECIFICACIONES". Esta linea no contendrá ninguna otra celda con contenido. Tras esta linea habrá una o mas lineas con todas las celdas en blanco y posteriormente se incluirán los bloques que forman parte del proyecto.

8.1.3 Especificaciones de los bloques

La especificación de un bloque estará contenida entre una o mas lineas en blanco tanto antes como después. La especificación del bloque contendrá una linea con la definición del bloque y una linea para definir cada registro perteneciente al bloque. La especificación del bloque no contendrá ninguna linea en blanco en su interior.

8.1.4 Definición del bloque

Ocupará tres celdas:

- La primera contendrá el nombre del bloque. No podrá haber otro bloque con el mismo nombre y deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- La segunda indicará el tipo del bloque ya sea escritura(WR) o lectura(RD).
- La tercera indicará el número de veces que el bloque estará repetido en el sistema.
- La linea no contendrá nada mas.

8.1.5 Definición de los registros de los bloques de tipo lectura

Ocuparán 20 celdas:

- La primera contendrá el nombre del registro. No podrá haber otro registro en el bloque con el mismo nombre y deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- La segunda contendrá el nombre de la fase asociada al registro. Deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- La tercera contendrá el nombre del reloj asociado al registro. Deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- Las siguientes 16 celdas contendrán la definición de los campos pertenecientes al registro. Los campos pueden ser de dos tipos campos útiles o campos nulos. Los campos útiles se definen con el nombre seguido del número de bits que utilizan entre paréntesis, esto no será necesario en caso de que solo utilice un bit. Los campos nulos se definen de la misma manera pero el nombre será en todos los casos la cadena "$".
  Un campo, ya sea útil o nulo, deberá ocupar un número de celdas idéntico al número de bits que ocupa en el registro, por lo tanto habrá que concatenar tantas celdas como hagan falta para indicar esta situación. El contenido del campo solo estará en la primera celda y el resto deberán estar vacías.
- Tras la definición de los campos la siguiente celda contendrá el simbolo '#'.
- La linea no contendrá nada mas.

8.1.6 Definición de los registros de los bloques de tipo escritura

Ocuparán 39 celdas:

- La primera contendrá el nombre del registro. No podrá haber otro registro en el bloque con el mismo nombre y deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- La segunda contendrá el nombre de la fase asociada al registro. Deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- La tercera contendrá el nombre del reloj asociado al registro. Deberá cumplir las normas de estilo léxicas indicadas mas abajo.
- Las siguientes 16 celdas contendrán la definición de los campos pertenecientes al registro. Los campos pueden ser de dos tipos campos útiles o campos nulos. Los campos útiles se definen con el nombre seguido del número de bits que utilizan entre paréntesis, esto no será necesario en caso de que solo utilice un bit. Los campos nulos se definen de la misma manera pero el nombre será en todos los casos la cadena "$".
  Un campo, ya sea útil o nulo, deberá ocupar un número de celdas idéntico al número de bits que ocupa en el registro, por lo tanto habrá que concatenar tantas celdas como hagan falta para indicar esta situación. El contenido del campo solo estará en la primera celda y el resto deberán estar vacías.
- Tras la definición de los campos la siguiente celda contendrá el símbolo '#'.

- Las siguientes 16 celdas contendrán el valor de inicialización de los campos definidos anteriormente. Estos valores deberán ocupar las mismas celdas que ocupan las definiciones de los campos respectivos. En los campos vacíos se escribirá la misma cadena que se ha puesto en la definición. En los campos útiles se escribirá lo siguiente:
  - Si el valor de inicialización es una máscara de bits se escribirá la cadena “b” seguida del valor en binario.
  - Si el campo es una variable sin signo se escribirá la cadena “u” seguida del valor en decimal.
  - Si el campo es una variable con signo se escribirá la cadena “s” seguida del valor en decimal.
- Tras los valores de inicialización vienen tres celdas opcionales que indicarán que los campos tienen un rango de valores aceptado. Rellenar estas celdas es opcional como un todo, o estan todas o no esta ninguna. Estos valores se suponen acordes a las definiciones dadas anteriormente, ya sea que el tamaño en bits no ocupe mas que la definición del campo como que no se definan números negativos para un campo ‘unsigned’. Estas restricciones se indicarán de la siguiente manera:
  - En la primera se indicará el valor mínimo.
  - En la siguiente se indicará el valor máximo.
  - En la última se indicará la acción a tomar si el valor introducido no se encuentra en el rango indicado. Si se indica “YES” el valor se truncará al valor mas cercano ya sea el máximo o mínimo. Si se indica “NO” se ignorará cualquier valor que no esté entre el valor mínimo y máximo.
- La linea no contendrá nada mas.

8.1.7 Final de las especificaciones

La definición de los bloques se extenderán hasta una linea que contendrá en la primera celda la cadena: "-- FIN_ESPECIFICACIONES". Esta linea no contendrá ninguna otra celda con contenido. Tras esta linea todo los que se incluya se ignorará.

8.1.8 Normas de estilo léxicas

Los nombres estarán compuestos unicamente por letras, números o guiones bajos(“_”) con las siguientes salvedades:

- ✔ Si el nombre esta formado unicamente por un carácter deberá ser obligatoriamente una letra.
- ✔ Si el nombre esta formado por mas de un carácter el primero será una letra o un guión bajo.

8.2 Analizador semántico

Recibe los 'tokens' del analizador sintáctico y se encarga de producir la estructura de datos que contendrá el fichero CSV en memoria lo que permitirá un acceso mas rápido a la hora de generar los ficheros de verificación.
Este analizador estará integrado dentro del analizador sintáctico y entrará en funcionamiento mientras se estén verificando las distintas reglas de la gramática. Irá transformando los distintos 'tokens' en las variables necesarias para ir creando los diferentes elementos, ya sean los nombres de los bloques, registros, campos, fases y relojes como los valores numéricos tanto de inicialización como de rango de valores mínimos y máximos.

El resultado de los analizadores sintáctico y semántico es el siguiente:

- input ::= BEGIN { EOL } { register_block { EOL } } END

```
ArrayList<Block> Input():
{
   int numReps, dir = 1;
   Block block_data = new Block();
   ArrayList<Block> block_list = new ArrayList<Block>();
}
{
   <BEGIN>
   (
      <EOL>
   )+
   (
   // Captura todos los datos del bloque activo y genera las repeticiones necesarias.
      numReps = register_block(block_data)
                           {
                              for (int i = 0; i < numReps; i++)
                              {
                                 Block tmp = new Block(block_data.getName() + "_" + i,
                                                       block_data.getMode(),
                                                       block_data.getRegisterList());
                                 tmp.setRegsDir(dir);
                                 dir += tmp.getRegisterNum(tmp.getMode());
                                 block_list.add(tmp);
                              }
                           }
      (
         <EOL>
      )+
   )+

   <END>                      {  return block_list; }
}
```

Tras capturar cada bloque lo añade a la lista de bloques que va a devolver el analizador sintáctico duplicándolo las veces indicadas en el fichero. Cada bloque se diferenciará de los duplicados en que su nombre terminará en un número distinto empezando el primer bloque en "_0", el siguiente en "_1" y así sucesivamente. Además se numerarán los distintos registros del bloque en orden ascendente.

- register_block ::= ID [READ INTEGER EOL { read_register_definition }
  | WRITE INTEGER EOL { write_register_definition }]

```
int register_block(Block block):
{
   int reps;
   Token token;
   String name;
   Block.Mode mode;
   ArrayList<Register> register_list;
}
{
   token = <ID>            {  name = token.image; }
   (
      <READ>               {  mode = Block.Mode.READ; }
      token = <INTEGER>    {  reps = Integer.parseInt(token.image); }
      <EOL>

      register_list = read_register_definition()
   |
      <WRITE>              {  mode = Block.Mode.WRITE; }
      token = <INTEGER>    {  reps = Integer.parseInt(token.image); }
      <EOL>

      register_list = write_register_definition()
   )
                           {
                              block.setName(name);
                              block.setMode(mode);
                              block.setRegisterList(register_list);
                              return reps;
                           }
}
```

Para gestionar la captura de los bloques tiene que saber si el bloque es de escritura o lectura, por eso después de capturar el nombre de este hay dos caminos para indicar esta disyuntiva. En cada caso se captura el modo y el número de repeticiones y se llama a la función correspondiente que se encarga de capturar la lista de registros pertenecientes al bloque. Una vez que ha terminado de capturar todos los datos, el bloque rellena el objeto de tipo 'Block' que ha recibido de la función "input" con el nombre, modo y lista de registros. Además devuelve el número de repeticiones para que se generen los bloques duplicados necesarios.

- read_register_definition ::= ID ID ID { fields_definition } SEPARATOR EOL

```
ArrayList<Register> read_register_definition():
{
   Token token;
   String name, phase, clock;
   ArrayList<Field> field_list;
   ArrayList<Register> register_list = new ArrayList<Register>();
}
{
   (
      token = <ID>            {  name = token.image; }
      token = <ID>            {  phase = token.image; }
      token = <ID>            {  clock = token.image; }

      field_list = fields_definition()
      <SEPARATOR>
      <EOL>
                 {  register_list.add(new Register(name, phase, clock, field_list)); }
   )+
                              {  return register_list; }
}
```

La gestión de los registros de un bloque de tipo lectura consiste en capturar el nombre del registro, de la fase y del reloj. Luego se delega la captura de los campos que componen el registro y cuando se tienen todos los datos se genera un objeto de tipo 'Register' que se añade a la lista de registros que se va a devolver a la función "register_block". Esto se repetirá hasta que el analizador encuentre una linea en blanco, que indica el final de la definición de un bloque, momento en el cual devolverá la lista de registros.

- write_register_definition ::= ID ID ID { fields_definition } SEPARATOR
  { values_definition } (INTEGER INTEGER [YES | NO]) EOL

```
ArrayList<Register> write_register_definition():
{
   Token token;
   String name, phase, clock;
   int minValue = 0, maxValue = 0;
   boolean trunk_value = false, range_values = false;

   ArrayList<Field> field_list;
   ArrayList<Field> value_list;
   ArrayList<Register> register_list = new ArrayList<Register>();
}
{
   (
                              {
                                 field_list = new ArrayList<Field>();
                                 value_list = new ArrayList<Field>();
                                 minValue = 0; maxValue = 0;
                                 trunk_value = false; range_values = false;
                              }
      token = <ID>            {  name = token.image; }
      token = <ID>            {  phase = token.image; }
      token = <ID>            {  clock = token.image; }

      field_list = fields_definition()
      <SEPARATOR>
      value_list = values_definition()

      (
         token = <INTEGER>    {  minValue = Integer.parseInt(token.image); }
         token = <INTEGER>    {  maxValue = Integer.parseInt(token.image); }
         (
            <YES>             {  trunk_value = true; }
         |
            <NO>              {  trunk_value = false; }
         )?
                              {  range_values = true; }
      )?
      <EOL>
                              {
                                 // Joins the fields and the values.
                                 for(int i = 0; i < field_list.size(); i++)
                                 {
                                    Field field = field_list.get(i);
                                    Field value = value_list.get(i);

                                  if (range_values)
                                   field.join(value, minValue, maxValue, trunk_value);
                                  else
                                       field.join(value);
                                    field_list.set(i, field);
                                 }

                       // Creates the register readed and adds it to the return array.
                      register_list.add(new Register(name, phase, clock, field_list));
                              }
   )+
                              {  return register_list; }
}
```

La gestión de captura de los registros de un bloque de tipo escritura es similar a la de los registros de un bloque de lectura. Esta consiste en capturar el nombre del registro, de la fase y del reloj, luego se delega la captura de los campos que componen el registro y la captura de los valores de inicialización de cada uno y de los valores que indican el mínimo, máximo y truncado si los hay.
Cuando se tienen todos los datos se junta la lista de la definición de los campos con la lista de valores de los campos y esto da la definición completa de cada campo.
Con todos los campos completamente definidos se genera un objeto de tipo 'Register' que se añade a la lista de registros que se va a devolver a la función "register_block". Esto se repetirá hasta que el analizador encuentre una linea en blanco, que indica el final de la definición de un bloque, momento en el cual devolverá la lista de registros.

- fields_definition ::= [ID (LPAR INTEGER RPAR)
  | DOLLAR (LPAR INTEGER RPAR)]

```
ArrayList<Field> fields_definition():
{
   Token token;
   String field_name;
   int bit_length, num_dollar = 0;
   ArrayList<Field> field_list = new ArrayList<Field>();
}
{
   (
      token = <ID>            {
                                 field_name = token.image;
                                 bit_length = 1;
                              }
      (
         <LPAR>
         token = <INTEGER>    {  bit_length = Integer.parseInt(token.image); }
         <RPAR>
      )?
                              {  field_list.add(new Field(field_name, bit_length)); }
   |
      <DOLLAR>                {  bit_length = 1; }
      (
         <LPAR>
         token = <INTEGER>    {  bit_length = Integer.parseInt(token.image); }
         <RPAR>
      )?
                    {  field_list.add(new Field("$", bit_length, Field.Type.DOLLAR)); }
   )+
                              {  return field_list; }
}
```

La captura de las definiciones de los campos se encargan de recoger si es un campo útil o vació, su nombre en caso de que sea un campo útil y su tamaño en bits definido en el fichero CSV. Tras esta captura de datos se genera un objeto de tipo 'Field' que se añade a la lista de campos. Esto se repetirá hasta que el analizador encuentre el separador que indica el final de la definición de campo, momento en el cual devolverá la lista de campos.

- values_definition ::= [BIT_NUMBER | SIG_NUMBER | USG_NUMBER
  | DOLLAR (LPAR INTEGER RPAR)]

```
ArrayList<Field> values_definition():
{
   int value;
   Token token;
   ArrayList<Field> value_list = new ArrayList<Field>();
}
{
   (
      token = <SIG_NUMBER>    {
                                 value = Integer.parseInt(token.image);
                                 value_list.add(new Field(Field.Type.SIGNED, value));
                              }
   |
      token = <USG_NUMBER>    {
                                 value = Integer.parseInt(token.image);
                                 value_list.add(new Field(Field.Type.UNSIGNED,
value));
                              }
   |
      token = <BIT_NUMBER>    {
                                 value = Integer.parseInt(token.image, 2);
                                 value_list.add(new Field(Field.Type.UNSIGNED,
value));
                              }
   |
      <DOLLAR>
      (
         <LPAR>
         <INTEGER>
         <RPAR>
      )?
                              {  value_list.add(new Field(Field.Type.DOLLAR, 0)); }
   )+
                              {  return value_list; }
}
```

La captura de las definiciones de valores de los campos se encarga de recoger si es un campo esta definido como 'signed', 'unsigned', máscara de bits o vacío. Tras esto se genera un objeto de tipo 'Field' con la información del tipo y el valor de inicialización que se añade a la lista de valores de los campos. Esto se repetirá hasta que el analizador encuentre el final de linea o un número entero que indica que hay definición de mínimo, máximo y truncado. Después devolverá la lista de campos.

8.2.1 Estructuras auxiliares

Para que el analizador semántico guarde la información en memoria son necesarias tres estructuras que ya he ido comentando en los párrafos anteriores: la clase 'Block', la clase 'Register' y la clase 'Field'. Estas estructuras también serán útiles a la hora de generar los ficheros de verificación ya que incorporan métodos que generan sus propias definiciones en el sistema.

8.2.1.1 Clase Block

Block

```
public Block()

Constructor vacío para inicializar los campos mediante métodos 'set'.
```

Block

```
public Block(String name,
       Block.Mode mode,
       ArrayList<Register> register_list)

Constructor con todos los campos inicializados.

Parameters:
    name - Nombre del bloque.
    mode - Modo de funcionamiento del bloque.
    register_list - Lista de registros pertenecientes al bloque.
```

getMode

```
public Block.Mode getMode()

Devuelve el modo del bloque.

Returns:
    El modo del bloque.
```

getName

```
public String getName()

Devuelve el nombre del bloque.

Returns:
    El nombre del bloque.
```

getRegisterList

```
public ArrayList<Register> getRegisterList()

Devuelve la lista de registros del bloque.

Returns:
    La lista de registros del bloque.
```

getRegisterNum

```
public int getRegisterNum()

Devuelve el número de registros del bloque.

Returns:
    El número de registros del bloque.
```

getRegisterNum

```
public int getRegisterNum(Block.Mode mode)

Devuelve el número de registros que contiene el bloque si es del modo buscado.

Parameters:
    mode - Modo esperado de funcionamiento del bloque.

Returns:
    El número de registros que contiene el bloque si es del modo buscado.
    0 si no es del modo buscado.
```

setMode

```
public void setMode(Block.Mode mode)

Establece el modo del bloque.

Parameters:
    mode - Modo a guardar.
```

setName

```
public void setName(String name)

Establece el nombre del bloque.

Parameters:
    name - Nombre a guardar.
```

setRegisterList

```
public void setRegisterList(ArrayList<Register> register_list)

Establece un listado de registros para el bloque.

Parameters:
    register_list - Lista de registros a guardar.
```

setRegsDir

```
public void setRegsDir(int dir)

Establece la dirección de los registros incluidos en el bloque.

Las direcciones se almacenarán de manera secuencial dentro del bloque.

Parameters:
    dir - Número de la dirección del primer registro del bloque.
```

checkMode

```
public boolean checkMode(Block.Mode mode)

Comprueba que el modo del bloque coincide con el dado.

Parameters:
    mode - Modo esperado para comparar.

Returns:
    TRUE si el modo del bloque es el esperado.
    FALSE si no lo es.
```

getDefinition

```
public StringBuilder getDefinition()

Genera la producción de la estructura de datos del bloque.

Esta estructura genera un tipo de dato para cada bloque del sistema. En el interior de
este tipo de datos inserta las definiciones de los registros y dentro de estas inserta
las definiciones de los campos que contienen los registros.

Returns:
    La cadena con el contenido de la producción del bloque.
```

getInstantiation

```
public String getInstantiation()

Genera la producción que instancia el bloque dentro del banco.

Returns:
    La cadena que define la instanciación del bloque.
```

8.2.1.2 Clase Register

Register

```
public Register(Register register)

Constructor a partir de otro objeto de tipo registro.
```

Register

```
public Register(String name,
       String phase,
       String clock,
       ArrayList<Field> fields)

Constructor con todos los campos del registro excepto la dirección.

Parameters:
    name - Nombre del registro.
    phase - Fase asociada al registro.
    clock - Reloj asociado al registro.
    fields - Lista de campos que contiene el registro.
```

getClock

```
public String getClock()

Devuelve el reloj asociado al registro.

Returns:
    El reloj asociado al registro.
```

getDir

```
public int getDir()

Devuelve la dirección del registro.

Returns:
    La dirección del registro.
```

getFieldList

```
public ArrayList<Field> getFieldList()

Devuelve la lista de campos del registro.

Returns:
    La lista de campos del registro.
```

getFieldsNum

```
public int getFieldsNum()

Devuelve el número de campos del registro.

Returns:
    El número de campos del registro.
```

getName

```
public String getName()

Devuelve el nombre del registro.

Returns:
    El nombre del registro.
```

getPhase

```
public String getPhase()

Devuelve la fase asociada al registro.

Returns:
    La fase asociada al registro.
```

setDir

```
public void setDir(int dir)

Establece la dirección del registro.

Parameters:
    dir - La dirección a guardar.
```

getDefinition

```
public StringBuilder getDefinition()

Genera la producción de la estructura de datos del registro.

Esta estructura genera una estructura para cada registro. Dentro de esta se insertan
las definiciones de los campos mediante objetos.

Returns:
    La cadena con el contenido de la producción del registro.
```

8.2.1.3 Clase Field

Field

```
public Field(String name,
       int size)

Constructor a partir de la definición de los campos, parte izquierda del CSV.

Parameters:
    name - Nombre del campo.
    size - Tamaño en bits que ocupa el campo
```

Field

```
public Field(Field.Type type,
       int reset)

Constructor a partir de la definición de los valores de reset, parte derecha del CSV.

Parameters:
    type - Tipo del campo.
    reset - Valor de inicialización en decimal.
```

Field

```
public Field(String name,
       int size,
       Field.Type type)

Constructor a partir de la definición de los valores de reset en el CSV.

Parameters:
    name - Nombre del campo.
    size - Tamaño en bits que ocupa el campo
    type - Tipo del campo.
```

getName

```
public String getName()

Devuelve el nombre del campo.

Returns:
    El nombre del campo.
```

getReset

```
public int getReset()

Devuelve el valor de inicialización.

Returns:
    El valor de inicialización.
```

getSize

```
public int getSize()

Devuelve el tamaño en bits del campo.
```

```
Returns:
    El tamaño en bits del campo.
```

getType

```
public Field.Type getType()

Devuelve el tipo del campo.

Returns:
    El tipo del campo.
```

checkType

```
public boolean checkType(Field.Type type)

Comprueba que el tipo del campo coincide con el dado.

Parameters:
    type - Tipo esperado para comparar.

Returns:
    TRUE si el tipo del campo es el esperado.
    FALSE si no lo es.
```

getDefinition

```
public String getDefinition()

Genera la producción de la estructura de datos del campo.

Esta estructura es un objeto para cada campo. Este objeto puede ser de dos clases:
SignedField o UnsignedField. No se pueden inicializar las clases mediante el método
'new' porque las instancias están generadas en un fichero de tipo 'package'.

Returns:
    La cadena con el contenido de la producción del campo.
```

getInstantiation

```
public String getInstantiation()

Genera la producción que instancia el campo dentro del registro.

Esto genera la inicialización del objeto mediante el método 'new'.

Returns:
    La cadena que define la instanciación del campo.
```

join

```
public void join(Field values)

Junta este campo con la definición de los valores de reset.

Es necesario realizar esta unión ya que la definición del campo y del valor de
inicialización esta separada en el CSV.

Parameters:
    values - Campo con los valores de tipo e inicialización.
```

join

```
public void join(Field values,
       int min,
       int max,
       boolean trunk)

Junta este campo con la definición de los valores de reset, mínimo, máximo y truncado.

Es necesario realizar esta unión ya que la definición del campo y del valor de
inicialización, mínimo, máximo y truncado esta separada en el CSV.

Parameters:
    values - Campo con los valores de tipo e inicialización.
    min - Rango de valor mínimo para el campo
    max - Rango de valor máximpo para el campo.
    trunk - Valor con la decisión de hacer truncado o no.
```

8.3 Contenido de los ficheros de verificación

8.3.1 read_bank_shadow.sv

```
import definitions::*;

module read_bank_shadow
   (
      // Valores de entrada de datos.
      input dir_vector_t addr_line,
      ref   read_bank_t  input_data,

      // Valor de salida esperado.
      input reg_vector_t output_line
   );

   reg_vector_t output_data = '0;


   always_comb begin : update_output
      unique case (addr_line)
         1: output_data = { input_data.bloque_lectura_0.registro_1.campo_1.get_value(),
                            input_data.bloque_lectura_0.registro_1.campo_2.get_value() };
         2: output_data = { input_data.bloque_lectura_0.registro_2.campo_3.get_value(), 6'b0 };
         3: output_data = { input_data.bloque_lectura_0.registro_3.campo_4.get_value(),
                            input_data.bloque_lectura_0.registro_3.campo_5.get_value() };
         default: output_data = '0;
      endcase
   end : update_output


   always_comb begin : value_check
      assert #0 (output_data ==? output_line)
      else $warning("RB @: %1d, NAME: %s, DATA: %1d, LINE: %1d",
                    addr_line, dir_2_reg(addr_line), output_data, output_line);
   end : value_check

endmodule : read_bank_shadow
```

8.3.2 read_bank_test.sv

```
import definitions::*;

module read_bank_test;
   // Valores de entrada generados.
   dir_vector_t addr_line  = '0;
   read_bank_t  input_data;  // Entrada simulada.
   memoria_especificaciones::t_lectura_i input_line;  // Entrada real.

   // Valor de salida obtenido.
   reg_vector_t output_line;

   read_bank_shadow rbs(.*);  // Banco de lectura simulado.
   memoria_banco_lectura rb(.*);  // Banco de lectura real.


   task set_value(input reg_vector_t value);
      input_data.bloque_lectura_0.registro_1.campo_1.set_value(value);
      input_line.bloque_lectura_0.registro_1.campo_1         = value;
      input_data.bloque_lectura_0.registro_1.campo_2.set_value(value);
      input_line.bloque_lectura_0.registro_1.campo_2         = value;
      input_data.bloque_lectura_0.registro_2.campo_3.set_value(value);
      input_line.bloque_lectura_0.registro_2.campo_3         = value;
      input_data.bloque_lectura_0.registro_3.campo_4.set_value(value);
      input_line.bloque_lectura_0.registro_3.campo_4         = value;
      input_data.bloque_lectura_0.registro_3.campo_5.set_value(value);
      input_line.bloque_lectura_0.registro_3.campo_5         = value;
   endtask : set_value


   task change_address();
      foreach(read_register_list[i])
         #1 addr_line = read_register_list[i];
      #1 addr_line = '0;
   endtask : change_address


   task update_value(input reg_vector_t new_value);
      automatic reg_vector_t old_value = ~new_value;

      $display();
      $display("ACTUALIZANDO CAMPO: read_bank.bloque_lectura_0.registro_1.campo_1");
      input_data.bloque_lectura_0.registro_1.campo_1.set_value(new_value);
      input_line.bloque_lectura_0.registro_1.campo_1         = new_value;
      change_address();
      $display();
      $display("ACTUALIZANDO CAMPO: read_bank.bloque_lectura_0.registro_1.campo_2");
      input_data.bloque_lectura_0.registro_1.campo_2.set_value(new_value);
      input_line.bloque_lectura_0.registro_1.campo_2         = new_value;
      change_address();
      input_data.bloque_lectura_0.registro_1.campo_1.set_value(old_value);
      input_line.bloque_lectura_0.registro_1.campo_1         = old_value;
      input_data.bloque_lectura_0.registro_1.campo_2.set_value(old_value);
      input_line.bloque_lectura_0.registro_1.campo_2         = old_value;

      $display();
      $display("ACTUALIZANDO CAMPO: read_bank.bloque_lectura_0.registro_2.campo_3");
      input_data.bloque_lectura_0.registro_2.campo_3.set_value(new_value);
      input_line.bloque_lectura_0.registro_2.campo_3         = new_value;
      change_address();
      input_data.bloque_lectura_0.registro_2.campo_3.set_value(old_value);
      input_line.bloque_lectura_0.registro_2.campo_3         = old_value;
```

```
      $display();
      $display("ACTUALIZANDO CAMPO: read_bank.bloque_lectura_0.registro_3.campo_4");
      input_data.bloque_lectura_0.registro_3.campo_4.set_value(new_value);
      input_line.bloque_lectura_0.registro_3.campo_4         = new_value;
      change_address();
      $display();
      $display("ACTUALIZANDO CAMPO: read_bank.bloque_lectura_0.registro_3.campo_5");
      input_data.bloque_lectura_0.registro_3.campo_5.set_value(new_value);
      input_line.bloque_lectura_0.registro_3.campo_5         = new_value;
      change_address();
      input_data.bloque_lectura_0.registro_3.campo_4.set_value(old_value);
      input_line.bloque_lectura_0.registro_3.campo_4         = old_value;
      input_data.bloque_lectura_0.registro_3.campo_5.set_value(old_value);
      input_line.bloque_lectura_0.registro_3.campo_5         = old_value;
   endtask : update_value


   initial begin
      init_read_bank(input_data);

      $display();
      $display("MUESTREANDO VALORES DE RESET...");
      change_address();

      $display();
      $display("INICIANDO PRUEBAS PARA EL BANCO DE LECTURA.");

      $display();
      $display("ESTABLECIENDO VALOR: '01'");
      set_value(value01);
      change_address();

      $display();
      $display("CARGANDO VALOR '10'");
      update_value(value10);

      $display();
      $display("ESTABLECIENDO VALOR: '10'");
      set_value(value10);
      change_address();

      $display();
      $display("CARGANDO VALOR '01'");
      update_value(value01);

      $display();
      $display("PRUEBAS FINALIZADAS PARA EL BANCO DE LECTURA.");
      #1 $finish;
   end
endmodule : read_bank_test
```

8.3.3 write_bank_shadow.sv

```
import definitions::*;

module write_bank_shadow
   (
      // Entrada de fases.
      input logic fase_1,
      input logic fase_2,

      // Entrada de relojes.
      input logic reloj_1,
      input logic reloj_2,

      // Valores de entrada de datos.
      input logic clk,
      input logic enable,
      input logic reset_n,
      input dir_vector_t addr_line,
      input reg_vector_t input_line,

      // Valor de salida esperado.
      ref memoria_especificaciones::t_escritura_o output_line
   );

   // Almacenes con los valores de verificación.
   write_bank_t temporal;
   write_bank_t output_data;

   initial begin
      init_write_bank(temporal);
      init_write_bank(output_data);
   end


   always @(negedge (reset_n)) begin : reset_load
      output_data.bloque_escritura_0.registro_1.campo_1.set_reset();
      temporal.bloque_escritura_0.registro_1.campo_1.set_reset();
      output_data.bloque_escritura_0.registro_1.campo_2.set_reset();
      temporal.bloque_escritura_0.registro_1.campo_2.set_reset();
      output_data.bloque_escritura_0.registro_2.campo_3.set_reset();
      temporal.bloque_escritura_0.registro_2.campo_3.set_reset();
      output_data.bloque_escritura_0.registro_3.campo_4.set_reset();
      temporal.bloque_escritura_0.registro_3.campo_4.set_reset();
      output_data.bloque_escritura_0.registro_3.campo_5.set_reset();
      temporal.bloque_escritura_0.registro_3.campo_5.set_reset();

      output_data.bloque_escritura_1.registro_1.campo_1.set_reset();
      temporal.bloque_escritura_1.registro_1.campo_1.set_reset();
      output_data.bloque_escritura_1.registro_1.campo_2.set_reset();
      temporal.bloque_escritura_1.registro_1.campo_2.set_reset();
      output_data.bloque_escritura_1.registro_2.campo_3.set_reset();
      temporal.bloque_escritura_1.registro_2.campo_3.set_reset();
      output_data.bloque_escritura_1.registro_3.campo_4.set_reset();
      temporal.bloque_escritura_1.registro_3.campo_4.set_reset();
      output_data.bloque_escritura_1.registro_3.campo_5.set_reset();
      temporal.bloque_escritura_1.registro_3.campo_5.set_reset();
   end : reset_load
```

```
always @(posedge (reloj_1) iff (reset_n)) begin : reloj_1_output_load
   if (fase_1) begin
      output_data.bloque_escritura_0.registro_1.campo_1
         .set_value(temporal.bloque_escritura_0.registro_1.campo_1.get_value());

      output_data.bloque_escritura_0.registro_1.campo_2
         .set_value(temporal.bloque_escritura_0.registro_1.campo_2.get_value());
      output_data.bloque_escritura_1.registro_1.campo_1
         .set_value(temporal.bloque_escritura_1.registro_1.campo_1.get_value());
      output_data.bloque_escritura_1.registro_1.campo_2
         .set_value(temporal.bloque_escritura_1.registro_1.campo_2.get_value());
   end
end : reloj_1_output_load

always @(posedge (reloj_2) iff (reset_n)) begin : reloj_2_output_load
   if (fase_1) begin
      output_data.bloque_escritura_0.registro_2.campo_3
         .set_value(temporal.bloque_escritura_0.registro_2.campo_3.get_value());
      output_data.bloque_escritura_1.registro_2.campo_3
         .set_value(temporal.bloque_escritura_1.registro_2.campo_3.get_value());
   end

   if (fase_2) begin
      output_data.bloque_escritura_0.registro_3.campo_4
         .set_value(temporal.bloque_escritura_0.registro_3.campo_4.get_value());
      output_data.bloque_escritura_0.registro_3.campo_5
         .set_value(temporal.bloque_escritura_0.registro_3.campo_5.get_value());
      output_data.bloque_escritura_1.registro_3.campo_4
         .set_value(temporal.bloque_escritura_1.registro_3.campo_4.get_value());
      output_data.bloque_escritura_1.registro_3.campo_5
         .set_value(temporal.bloque_escritura_1.registro_3.campo_5.get_value());
   end
end : reloj_2_output_load

always @(posedge (clk) iff (reset_n && enable)) begin : temporal_load
   unique case (addr_line)
      4:
         begin
             temporal.bloque_escritura_0.registro_1.campo_1
                .set_value(input_line[REG_SIZE-1 : REG_SIZE-8]);
             temporal.bloque_escritura_0.registro_1.campo_2
                .set_value(input_line[REG_SIZE-9 : REG_SIZE-16]);
         end

      5:
         begin
             temporal.bloque_escritura_0.registro_2.campo_3
                .set_value(input_line[REG_SIZE-11 : REG_SIZE-16]);
         end

      6:
         begin
             temporal.bloque_escritura_0.registro_3.campo_4
                .set_value(input_line[REG_SIZE-1 : REG_SIZE-15]);
             temporal.bloque_escritura_0.registro_3.campo_5
                .set_value(input_line[REG_SIZE-16 : REG_SIZE-16]);
         end

      7:
         begin
             temporal.bloque_escritura_1.registro_1.campo_1
                .set_value(input_line[REG_SIZE-1 : REG_SIZE-8]);
             temporal.bloque_escritura_1.registro_1.campo_2
                .set_value(input_line[REG_SIZE-9 : REG_SIZE-16]);
         end
```

```
      8:
         begin
             temporal.bloque_escritura_1.registro_2.campo_3
                .set_value(input_line[REG_SIZE-11 : REG_SIZE-16]);
         end

      9:
         begin
             temporal.bloque_escritura_1.registro_3.campo_4
                .set_value(input_line[REG_SIZE-1 : REG_SIZE-15]);
             temporal.bloque_escritura_1.registro_3.campo_5
                .set_value(input_line[REG_SIZE-16 : REG_SIZE-16]);
         end

      default: ;
   endcase
end : temporal_load


always @(negedge(clk)) begin : value_check
   assert #0 (output_data.bloque_escritura_0.registro_1.campo_1.get_value()
                ==? output_line.bloque_escritura_0.registro_1.campo_1)
   else
      $warning("WR @: 4, NAME: write_bank.bloque_escritura_0.registro_1.campo_1,
                TMP: %1d, DATA: %1d, LINE: %1d.",
               temporal.bloque_escritura_0.registro_1.campo_1.get_value(),
               output_data.bloque_escritura_0.registro_1.campo_1.get_value(),
               output_line.bloque_escritura_0.registro_1.campo_1);

   assert #0 (output_data.bloque_escritura_0.registro_1.campo_2.get_value()
                ==? output_line.bloque_escritura_0.registro_1.campo_2)
   else
      $warning("WR @: 4, NAME: write_bank.bloque_escritura_0.registro_1.campo_2,
                TMP: %1d, DATA: %1d, LINE: %1d.",
               temporal.bloque_escritura_0.registro_1.campo_2.get_value(),
               output_data.bloque_escritura_0.registro_1.campo_2.get_value(),
               output_line.bloque_escritura_0.registro_1.campo_2);

   assert #0 (output_data.bloque_escritura_0.registro_2.campo_3.get_value()
                ==? output_line.bloque_escritura_0.registro_2.campo_3)
   else
      $warning("WR @: 5, NAME: write_bank.bloque_escritura_0.registro_2.campo_3,
                TMP: %1d, DATA: %1d, LINE: %1d.",
               temporal.bloque_escritura_0.registro_2.campo_3.get_value(),
               output_data.bloque_escritura_0.registro_2.campo_3.get_value(),
               output_line.bloque_escritura_0.registro_2.campo_3);

   assert #0 (output_data.bloque_escritura_0.registro_3.campo_4.get_value()
                ==? output_line.bloque_escritura_0.registro_3.campo_4)
   else
      $warning("WR @: 6, NAME: write_bank.bloque_escritura_0.registro_3.campo_4,
                TMP: %1d, DATA: %1d, LINE: %1d.",
               temporal.bloque_escritura_0.registro_3.campo_4.get_value(),
               output_data.bloque_escritura_0.registro_3.campo_4.get_value(),
               output_line.bloque_escritura_0.registro_3.campo_4);

   assert #0 (output_data.bloque_escritura_0.registro_3.campo_5.get_value()
                ==? output_line.bloque_escritura_0.registro_3.campo_5)
   else
      $warning("WR @: 6, NAME: write_bank.bloque_escritura_0.registro_3.campo_5,
                TMP: %1d, DATA: %1d, LINE: %1d.",
               temporal.bloque_escritura_0.registro_3.campo_5.get_value(),
               output_data.bloque_escritura_0.registro_3.campo_5.get_value(),
               output_line.bloque_escritura_0.registro_3.campo_5);
```

```
      assert #0 (output_data.bloque_escritura_1.registro_1.campo_1.get_value()
                    ==? output_line.bloque_escritura_1.registro_1.campo_1)
      else
         $warning("WR @: 7, NAME: write_bank.bloque_escritura_1.registro_1.campo_1,
                   TMP: %1d, DATA: %1d, LINE: %1d.",
                  temporal.bloque_escritura_1.registro_1.campo_1.get_value(),
                  output_data.bloque_escritura_1.registro_1.campo_1.get_value(),
                  output_line.bloque_escritura_1.registro_1.campo_1);

      assert #0 (output_data.bloque_escritura_1.registro_1.campo_2.get_value()
                    ==? output_line.bloque_escritura_1.registro_1.campo_2)
      else
         $warning("WR @: 7, NAME: write_bank.bloque_escritura_1.registro_1.campo_2,
                   TMP: %1d, DATA: %1d, LINE: %1d.",
                  temporal.bloque_escritura_1.registro_1.campo_2.get_value(),
                  output_data.bloque_escritura_1.registro_1.campo_2.get_value(),
                  output_line.bloque_escritura_1.registro_1.campo_2);

      assert #0 (output_data.bloque_escritura_1.registro_2.campo_3.get_value()
                    ==? output_line.bloque_escritura_1.registro_2.campo_3)
      else
         $warning("WR @: 8, NAME: write_bank.bloque_escritura_1.registro_2.campo_3,
                   TMP: %1d, DATA: %1d, LINE: %1d.",
                  temporal.bloque_escritura_1.registro_2.campo_3.get_value(),
                  output_data.bloque_escritura_1.registro_2.campo_3.get_value(),
                  output_line.bloque_escritura_1.registro_2.campo_3);

      assert #0 (output_data.bloque_escritura_1.registro_3.campo_4.get_value()
                    ==? output_line.bloque_escritura_1.registro_3.campo_4)
      else
         $warning("WR @: 9, NAME: write_bank.bloque_escritura_1.registro_3.campo_4,
                   TMP: %1d, DATA: %1d, LINE: %1d.",
                  temporal.bloque_escritura_1.registro_3.campo_4.get_value(),
                  output_data.bloque_escritura_1.registro_3.campo_4.get_value(),
                  output_line.bloque_escritura_1.registro_3.campo_4);

      assert #0 (output_data.bloque_escritura_1.registro_3.campo_5.get_value()
                    ==? output_line.bloque_escritura_1.registro_3.campo_5)
      else
         $warning("WR @: 9, NAME: write_bank.bloque_escritura_1.registro_3.campo_5,
                   TMP: %1d, DATA: %1d, LINE: %1d.",
                  temporal.bloque_escritura_1.registro_3.campo_5.get_value(),
                  output_data.bloque_escritura_1.registro_3.campo_5.get_value(),
                  output_line.bloque_escritura_1.registro_3.campo_5);

   end : value_check

endmodule : write_bank_shadow
```

8.3.4 write_bank_test.sv

```
import definitions::*;

module write_bank_test;
   // Fases.
   logic fase_1 = 0;
   logic fase_2 = 0;

   // Relojes.
   logic reloj_1 = 0;
   logic reloj_2 = 0;

   // Valores de entrada generados.
   logic clk     = 0;
   logic enable  = 0;
   logic reset_n = 1;
   dir_vector_t addr_line  = '0;
   reg_vector_t input_line = '0;

   // Valor de salida obtenido.
   memoria_especificaciones::t_escritura_o output_line;

   write_bank_shadow wbs(.*);  // Banco de escritura simulado.
   memoria_banco_escritura wb(.*);  // Banco de escritura real.

   always #1 clk++;


   /**
    * Gestiona la secuencia de activaciones progresivas de todas las combinaciones de fases y
relojes aceptadas.
    *   Para cada secuencia activa la fase, en el siguiente tick activa el reloj y en el
siguiente apaga ambas.
    */
   task rise_and_fall;
      @(negedge (clk))
         $display();
         $display("HABILITA LA FASE 'FASE_1' Y EL RELOJ 'RELOJ_1'.");
         fase_1  = 1;
      @(negedge (clk))
         reloj_1 = 1;
      @(negedge (clk))
         fase_1  = 0;
         reloj_1 = 0;
      @(negedge (clk));

      @(negedge (clk))
         $display();
         $display("HABILITA LA FASE 'FASE_1' Y EL RELOJ 'RELOJ_2'.");
         fase_1  = 1;
      @(negedge (clk))
         reloj_2 = 1;
      @(negedge (clk))
         fase_1  = 0;
         reloj_2 = 0;
      @(negedge (clk));

      @(negedge (clk))
         $display();
         $display("HABILITA LA FASE 'FASE_2' Y EL RELOJ 'RELOJ_2'.");
         fase_2  = 1;
      @(negedge (clk))
         reloj_2 = 1;
```

```
      @(negedge (clk))
         fase_2  = 0;
         reloj_2 = 0;
      @(negedge (clk));
   endtask : rise_and_fall


   task do_test(input reg_vector_t value);
      automatic reg_vector_t altvalue = ~value;

      $display();
      $display("CARGA EN EL TEMPORAL EL VALOR '%b'", value[1:0]);
      @(negedge (clk))
         input_line = value;

      for (dir_vector_t i = 9; i > 0; i--) begin
         @(negedge (clk))
            $display("   EN LA DIRECCION: %0d.", i);
            addr_line = i;
      end
      @(negedge (clk)) addr_line = '0;

      $display();
      $display("ACTUALIZA EL VALOR DE SALIDA.");
      rise_and_fall;

      // Cambia el valor de cada registro al valor complementario.
      for (dir_vector_t i = 9; i > 0; i--) begin
         $display();
         $display("CAMBIA EL TEMPORAL AL VALOR '%b' EN LA DIRECCION %0d(%s)",
                  altvalue[1:0], i, dir_2_reg(i));
         @(negedge (clk))
            addr_line  = i;
            input_line = altvalue;

         // Cambia todas las fases una a una.
         rise_and_fall;

         @(negedge (clk))
            input_line = value;
      end
   endtask : do_test


   initial begin
      $display();
      $display("INICIANDO PRUEBAS PARA EL BANCO DE ESCRITURA.");

      $display();
      $display("ENABLE ACTIVADO.");
      #1 enable = 1;

      $display();
      $display("RESET ACTIVADO.");
      #1 reset_n = 0;

      do_test(value01);
      do_test(value10);

      $display();
      $display("ENABLE DESACTIVADO.");
      #1 enable = 0;

      do_test(value01);
      do_test(value10);
```

```
      $display();
      $display("RESET DESACTIVADO.");
      #1 reset_n = 1;

      do_test(value01);
      do_test(value10);

      $display();
      $display("ENABLE ACTIVADO.");
      #1 enable = 1;

      do_test(value01);
      do_test(value10);

      $display();
      $display("PRUEBAS FINALIZADAS PARA EL BANCO DE ESCRITURA.");
      #1 $finish;
   end
endmodule : write_bank_test
```

8.3.5 definitions.sv

```
package definitions;
   parameter DIR_SIZE = 4;
   parameter REG_SIZE = 16;

   typedef logic [DIR_SIZE-1:0] dir_vector_t;
   typedef logic [REG_SIZE-1:0] reg_vector_t;

   parameter reg_vector_t value01 = { REG_SIZE { 2'b01 }};
   parameter reg_vector_t value10 = { REG_SIZE { 2'b10 }};

   dir_vector_t read_register_list [3]  = '{ 1, 2, 3 };
   dir_vector_t write_register_list [6]  = '{ 4, 5, 6, 7, 8, 9 };

   function string dir_2_reg(dir_vector_t dir);
      unique case (dir)
         1: return "read_bank.bloque_lectura_0.registro_1";
         2: return "read_bank.bloque_lectura_0.registro_2";
         3: return "read_bank.bloque_lectura_0.registro_3";

         4: return "write_bank.bloque_escritura_0.registro_1";
         5: return "write_bank.bloque_escritura_0.registro_2";
         6: return "write_bank.bloque_escritura_0.registro_3";

         7: return "write_bank.bloque_escritura_1.registro_1";
         8: return "write_bank.bloque_escritura_1.registro_2";
         9: return "write_bank.bloque_escritura_1.registro_3";
         default: return "";
      endcase
   endfunction : dir_2_reg

   function logic signed [REG_SIZE-1:0] MIN_SIGNED_VALUE(byte bit_length);
      static logic signed [REG_SIZE-1:0] ans = '1;
      return (ans << (bit_length - 1));
   endfunction : MIN_SIGNED_VALUE

   function logic signed [REG_SIZE-1:0] MAX_SIGNED_VALUE(byte bit_length);
   return (1 << (bit_length - 1)) - 1;
   endfunction : MAX_SIGNED_VALUE

   function logic [REG_SIZE:0] MIN_UNSIGNED_VALUE();
      return '0;
   endfunction : MIN_UNSIGNED_VALUE

   function logic [REG_SIZE:0] MAX_UNSIGNED_VALUE(byte bit_length);
      return (1 << bit_length) - 1;
   endfunction : MAX_UNSIGNED_VALUE

   class SignedField #(byte size);
      string                  name;
      string                  ftype;
      logic signed [size-1:0] value;
      logic signed [size-1:0] reset;
      logic signed [size-1:0] minValue;
      logic signed [size-1:0] maxValue;
      bit                     doTrunks;
      bit                     doChanged;

      function new(input string                  name,
                   input logic signed [size-1:0] reset    = '0,
                   input logic signed [size-1:0] minValue = MIN_SIGNED_VALUE(size),
                   input logic signed [size-1:0] maxValue = MAX_SIGNED_VALUE(size),
                   input bit                     doTrunks = 0);
```

```
      this.name      = name;
      this.ftype     = "signed";
      this.reset     = reset;
      this.value     = reset;
      this.minValue  = minValue;
      this.maxValue  = maxValue;
      this.doTrunks  = doTrunks;
      this.doChanged = 0;
   endfunction : new

   task print();
      $display("N: %s", name);
      $display("S: %d", size);
      $display("T: %s", ftype);
      $display("K: %d", doTrunks);
      $display("R: %6d - %0b", reset, reset);
      $display("I: %6d - %0b", minValue, minValue);
      $display("A: %6d - %0b", maxValue, maxValue);
      $display();
   endtask

   function string get_name();
      return this.name;
   endfunction : get_name

   function string get_ftype();
      return this.ftype;
   endfunction : get_ftype

   function logic signed [size-1:0] get_value();
      return this.value;
   endfunction : get_value

   function logic [size-1:0] get_size();
      return this.size;
   endfunction : get_size

   function bit has_changed();
      bit ans = doChanged;
      doChanged = 0;
      return ans;
   endfunction : has_changed

   task set_value(logic signed [size-1:0] value);
      unique case(doTrunks)
         0:
            if ((value >= this.minValue) && (value <= this.maxValue)) begin
               this.value = value;
               doChanged = 1;
            end

         1: begin
                     if (value > this.maxValue) this.value = this.maxValue;
               else if (value < this.minValue) this.value = this.minValue;
               else                            this.value = value;
               doChanged = 1;
            end
      endcase
   endtask : set_value

   task set_reset();
      this.value = this.reset;
   endtask : set_reset
endclass : SignedField
```

```
class UnsignedField #(byte size);
   string           name;
   string           ftype;
   logic [size-1:0] value;
   logic [size-1:0] reset;
   logic [size-1:0] minValue;
   logic [size-1:0] maxValue;
   bit              doTrunks;
   bit              doChanged;

   function new(input string           name,
                input logic [size-1:0] reset    = '0,
                input logic [size-1:0] minValue = MIN_UNSIGNED_VALUE(),
                input logic [size-1:0] maxValue = MAX_UNSIGNED_VALUE(size),
                input bit              doTrunks = 0);

      this.name      = name;
      this.ftype     = "unsigned";
      this.reset     = reset;
      this.value     = reset;
      this.minValue  = minValue;
      this.maxValue  = maxValue;
      this.doTrunks  = doTrunks;
      this.doChanged = 0;
   endfunction : new

   task print();
      $display("N: %s", name);
      $display("S: %d", size);
      $display("T: %s", ftype);
      $display("K: %d", doTrunks);
      $display("R: %6d - %0b", reset, reset);
      $display("I: %6d - %0b", minValue, minValue);
      $display("A: %6d - %0b", maxValue, maxValue);
      $display();
   endtask

   function string get_name();
      return this.name;
   endfunction : get_name

   function string get_ftype();
      return this.ftype;
   endfunction : get_ftype

   function logic [size-1:0] get_value();
      return this.value;
   endfunction : get_value

   function logic [size-1:0] get_size();
      return this.size;
   endfunction : get_size

   function bit has_changed();
      bit ans = doChanged;
      doChanged = 0;
      return ans;
   endfunction : has_changed

   task set_value(logic [size-1:0] value);
      unique case(doTrunks)
         0:
            if ((value >= this.minValue) && (value <= this.maxValue)) begin
               this.value = value;
```

```
                doChanged = 1;
             end

          1: begin
                     if (value > this.maxValue) this.value = this.maxValue;
                else if (value < this.minValue) this.value = this.minValue;
                else                            this.value = value;
                doChanged = 1;
             end
       endcase
    endtask : set_value

    task set_reset();
       this.value = this.reset;
    endtask : set_reset
 endclass : UnsignedField


 // BLOQUE_LECTURA_BLOCK_DEFINITION_INSTANTIATION
 typedef struct
 {
    struct
    {
       UnsignedField #(8) campo_1;
       UnsignedField #(8) campo_2;
    } registro_1;

    struct
    {
       UnsignedField #(10) campo_3;
    } registro_2;

    struct
    {
       UnsignedField #(15) campo_4;
       UnsignedField #(1) campo_5;
    } registro_3;

 } bloque_lectura_tb;

 // BLOQUE_ESCRITURA_BLOCK_DEFINITION_INSTANTIATION
 typedef struct
 {
    struct
    {
         SignedField #(8) campo_1;
       UnsignedField #(8) campo_2;
    } registro_1;

    struct
    {
       UnsignedField #(6) campo_3;
    } registro_2;

    struct
    {
         SignedField #(15) campo_4;
       UnsignedField #(1) campo_5;
    } registro_3;

 } bloque_escritura_tb;


 // READ_BANK_INSTANTIATION
 typedef struct
```

```
   {
      bloque_lectura_tb  bloque_lectura_0;
   } read_bank_t;


   // WRITE_BANK_INSTANTIATION
   typedef struct
   {
      bloque_escritura_tb  bloque_escritura_0;
      bloque_escritura_tb  bloque_escritura_1;
   } write_bank_t;


   task init_read_bank(inout read_bank_t rb);
      rb.bloque_lectura_0.registro_1.campo_1 = new(.name("campo_1"), .reset(0));
      rb.bloque_lectura_0.registro_1.campo_2 = new(.name("campo_2"), .reset(0));
      rb.bloque_lectura_0.registro_2.campo_3 = new(.name("campo_3"), .reset(0));
      rb.bloque_lectura_0.registro_3.campo_4 = new(.name("campo_4"), .reset(0));
      rb.bloque_lectura_0.registro_3.campo_5 = new(.name("campo_5"), .reset(0));
   endtask : init_read_bank

   task init_write_bank(inout write_bank_t wb);
      wb.bloque_escritura_0.registro_1.campo_1 = new(.name("campo_1"), .reset(111));
      wb.bloque_escritura_0.registro_1.campo_2 = new(.name("campo_2"), .reset(0));
      wb.bloque_escritura_0.registro_2.campo_3 = new(.name("campo_3"), .reset(0),
                                                  .minValue(1024), .maxValue(2048), .doTrunks(1));
      wb.bloque_escritura_0.registro_3.campo_4 = new(.name("campo_4"), .reset(0));
      wb.bloque_escritura_0.registro_3.campo_5 = new(.name("campo_5"), .reset(1));

      wb.bloque_escritura_1.registro_1.campo_1 = new(.name("campo_1"), .reset(111));
      wb.bloque_escritura_1.registro_1.campo_2 = new(.name("campo_2"), .reset(0));
      wb.bloque_escritura_1.registro_2.campo_3 = new(.name("campo_3"), .reset(0),
                                                  .minValue(1024), .maxValue(2048), .doTrunks(1));
      wb.bloque_escritura_1.registro_3.campo_4 = new(.name("campo_4"), .reset(0));
      wb.bloque_escritura_1.registro_3.campo_5 = new(.name("campo_5"), .reset(1));
   endtask : init_write_bank

endpackage : definitions
```

8.4 Traductor completo

```
options
{
   Lookahead   = 1;
   Ignore_Case = true;
}

PARSER_BEGIN(SystemVerilogParser)
package javacc;

import extra.Normalizer;
import java.io.File;
import java.util.ArrayList;
import structures.Block;
import structures.Field;
import structures.Register;
import workers.MixedAuxBank;
import workers.MixedReadBank;
import workers.MixedWriteBank;

public class SystemVerilogParser
{
   private static int               size      = 16;
   private static boolean           doTest    = false;
   private static boolean           doFolder  = false;
   private static ArrayList<String> filenames = new ArrayList<String>();

   public static void main(String args[])
   {
      SystemVerilogParser SVParser = new SystemVerilogParser(System.in);
      SVParser.readCommandLine(args);

      for (String filename : filenames)
      {
         try
         {
            File   fileID     = new File(filename);
            String fileName   = fileID.getName().substring(0,
                                        fileID.getName().lastIndexOf("."));
            String filePath   = fileID.getCanonicalPath();

            String folderPath = filePath.substring(0, filePath.lastIndexOf("."));
            if (doFolder)
            {
               File folder = new File(folderPath);
               folder.mkdir();
            }
            else
               folderPath = fileID.getCanonicalFile().getParent();


            Normalizer.println("Analizando: " + filePath + ".\n\n");
            SVParser.ReInit(new java.io.FileInputStream(filePath), "UTF-8");

            long before = System.currentTimeMillis();
            ArrayList<Block> block_list = SVParser.Input();
            long after = System.currentTimeMillis();
            if (doTest)
               Normalizer.println("Tiempo analisis CSV: "
                                  + (after - before) + " ms. \n");

            boolean existsReadBank  = SVParser.checkReadBank(block_list);
            boolean existsWriteBank = SVParser.checkWriteBank(block_list);
```

```
            MixedAuxBank.createFiles(folderPath, size, block_list,
                                     existsReadBank, existsWriteBank, doTest);

            if (existsReadBank)
               MixedReadBank.createFiles(folderPath, fileName, block_list, doTest);

            if (existsWriteBank)
               MixedWriteBank.createFiles(folderPath, fileName, block_list, doTest);
         }

         catch (java.io.FileNotFoundException fnfe)
         {
            System.out.printf("Fichero no encontrado: %s.\n\n", filename);
         }
         catch (Exception e)
         {
            e.printStackTrace();
         }
      }
   }

   private static boolean checkReadBank(ArrayList<Block> block_list)
   {
      for (Block block : block_list)
         if (block.checkMode(Block.Mode.READ))
            return true;

      return false;
   }

   private static boolean checkWriteBank(ArrayList<Block> block_list)
   {
      for (Block block : block_list)
         if (block.checkMode(Block.Mode.WRITE))
            return true;

      return false;
   }

   private static void readCommandLine(String[] args)
   {
      boolean setHelp = false;

      for (int i = 0; i < args.length; i++)
         switch(args[i])
         {
            case "-folder":
               doFolder = true;
               break;

            case "-help":
               setHelp = true;
               break;

            case "-size":
               size = Integer.parseInt(args[++i]);
               break;

            case "-test":
               doTest = true;
               break;

            default:
               filenames.add(args[i]);
         }
```

```
      if (setHelp || filenames.size() == 0 || args.length == 0)
      {
         Normalizer.showHelpMenu();
         System.exit(11);
      }
   }
}
PARSER_END(SystemVerilogParser)


/**
 * Analizador léxico
 */
SKIP:
{
   < COMMA:          "," >
}

TOKEN:   /* PALABRAS RESERVADAS */
{
     // Ignora todas las lineas previas a INICIO*
   < BEGIN:          ( ~[] )* "--INICIO_ESPECIFICACIONES" ( ~["\n"] )* >
     // Evita elementos posteriores al código.
|  < END:            "--FIN_ESPECIFICACIONES" >
}

TOKEN:   /* PALABRAS RESERVADAS */
{
   < READ:           "RD" >
|  < WRITE:          "WR" >
|  < YES:            "SI" >
|  < NO:             "NO" >
}

TOKEN:   /* SIMBOLOS */
{
   < DOLLAR:         "$" >
|  < SEPARATOR:      "#" >
|  < LPAR:           "(" >
|  < RPAR:           ")" >
|  < EOL:            "\n" >
}

TOKEN:   /* RESTO DE ELEMENTOS */
{
   < #LETTER:        (["a"-"z"]) >
|  < #DIGIT:         (["0"-"9"]) >
|  < INTEGER:        (["-"])? (<DIGIT>)+ >
|  < BIT_NUMBER:     "B" (<DIGIT>)+ >
                     {  matchedToken.image = new String(image.deleteCharAt(0)); }
|  < SIG_NUMBER:     "S" (<INTEGER>) >
                     {  matchedToken.image = new String(image.deleteCharAt(0)); }
|  < USG_NUMBER:     "U" (<DIGIT>)+ >
                     {  matchedToken.image = new String(image.deleteCharAt(0)); }
|  < ID:             (<LETTER> | (<LETTER> | "_") (<LETTER> | <DIGIT> | "_")+) >
                     {  matchedToken.image = new String(image).toLowerCase(); }
}
```

```
/**
 * Analizador gramatical
 */
ArrayList<Block> Input():
{
   int numReps, dir = 1;
   Block block_data = new Block();
   ArrayList<Block> block_list = new ArrayList<Block>();
}
{
   <BEGIN>
   (
      <EOL>
   )+

   (
      // Captura todos los datos del bloque leido y
      //    genera los bloques necesarios debidos a las repeticiones.
      numReps = register_block(block_data)
                           {
                              for (int i = 0; i < numReps; i++)
                              {
                                 Block tmp = new Block(block_data.getName() + "_" + i,
                                                       block_data.getMode(),
                                                       block_data.getRegisterList());
                                 tmp.setRegsDir(dir);
                                 dir += tmp.getRegisterNum(tmp.getMode());
                                 block_list.add(tmp);
                              }
                           }
      (
         <EOL>
      )+
   )+

   <END>                      {  return block_list; }
}

int register_block(Block block):
{
   int reps;
   Token token;
   String name;
   Block.Mode mode;
   ArrayList<Register> register_list;
}
{
   token = <ID>            {  name = token.image; }
   (
      <READ>               {  mode = Block.Mode.READ; }
      token = <INTEGER>    {  reps = Integer.parseInt(token.image); }
      <EOL>

      register_list = read_register_definition()
   |
      <WRITE>              {  mode = Block.Mode.WRITE; }
      token = <INTEGER>    {  reps = Integer.parseInt(token.image); }
      <EOL>

      register_list = write_register_definition()
   )
```

```
                           {
                              block.setName(name);
                              block.setMode(mode);
                              block.setRegisterList(register_list);
                              return reps;
                           }
}

ArrayList<Register> read_register_definition():
{
   Token token;
   String name, phase, clock;
   ArrayList<Field> field_list;
   ArrayList<Register> register_list = new ArrayList<Register>();
}
{
   (
      token = <ID>            {  name  = token.image; }
      token = <ID>            {  phase = token.image; }
      token = <ID>            {  clock = token.image; }

      field_list = fields_definition()
      <SEPARATOR>
      (
         values_definition()
      )?
      <EOL>
                              {  register_list.add(new Register(name, phase,
                                                                  clock, field_list)); }
   )+
                              {  return register_list; }
}

ArrayList<Register> write_register_definition():
{
   Token token;
   String name, phase, clock;
   int minValue = 0, maxValue = 0;
   boolean trunk_value = false, range_values = false;

   ArrayList<Field> field_list;
   ArrayList<Field> value_list;
   ArrayList<Register> register_list = new ArrayList<Register>();
}
{
   (
                              {
                                 field_list = new ArrayList<Field>();
                                 value_list = new ArrayList<Field>();
                                 minValue = 0; maxValue = 0;
                                 trunk_value = false; range_values = false;
                              }
      token = <ID>            {  name  = token.image; }
      token = <ID>            {  phase = token.image; }
      token = <ID>            {  clock = token.image; }

      field_list = fields_definition()
      <SEPARATOR>
      value_list = values_definition()
```

```
      (
         token = <INTEGER>    {  minValue = Integer.parseInt(token.image); }
         token = <INTEGER>    {  maxValue = Integer.parseInt(token.image); }
         (
            <YES>             {  trunk_value = true; }
         |
            <NO>              {  trunk_value = false; }
         )?
                              {  range_values = true; }
      )?
      <EOL>

                              {
                                 // Joins the fields and the values.
                                 for(int i = 0; i < field_list.size(); i++)
                                 {
                                    Field field = field_list.get(i);
                                    Field value = value_list.get(i);

                                    if (range_values)
                                       field.join(value, minValue, maxValue,
trunk_value);
                                    else
                                       field.join(value);
                                    field_list.set(i, field);
                                 }

                       // Creates the register readed and adds it to the return array.
                      register_list.add(new Register(name, phase, clock, field_list));
                              }
   )+
                              {  return register_list; }
}

ArrayList<Field> fields_definition():
{
   Token token;
   int bit_length;
   String field_name;
   ArrayList<Field> field_list = new ArrayList<Field>();
}
{
   (
      token = <ID>            {
                                 field_name = token.image;
                                 bit_length = 1;
                              }
      (
         <LPAR>
         token = <INTEGER>    {  bit_length = Integer.parseInt(token.image); }
         <RPAR>
      )?
                              {  field_list.add(new Field(field_name, bit_length)); }
   |
      <DOLLAR>                {  bit_length = 1; }
      (
         <LPAR>
         token = <INTEGER>    {  bit_length = Integer.parseInt(token.image); }
         <RPAR>
      )?
                       {  field_list.add(new Field("$", bit_length, Field.Type.DOLLAR));
}
   )+
                              {  return field_list; }
}
```

```
ArrayList<Field> values_definition():
{
   int value;
   Token token;
   ArrayList<Field> value_list = new ArrayList<Field>();
}
{
   (
      token = <SIG_NUMBER>    {
                                 value = Integer.parseInt(token.image);
                                 value_list.add(new Field(Field.Type.SIGNED, value));
                              }
   |
      token = <USG_NUMBER>    {
                                 value = Integer.parseInt(token.image);
                                 value_list.add(new Field(Field.Type.UNSIGNED,
value));
                              }
   |
      token = <BIT_NUMBER>    {
                                 value = Integer.parseInt(token.image, 2);
                                 value_list.add(new Field(Field.Type.UNSIGNED,
value));
                              }
   |
      <DOLLAR>
      (
         <LPAR>
         <INTEGER>
         <RPAR>
      )?
                              {  value_list.add(new Field(Field.Type.DOLLAR, 0)); }
   )+
                              {  return value_list; }
}
```

8.5 Fichero de pruebas de corrección del código generado

```
--INICIO_ESPECIFICACIONES,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,
lectura,RD,1,,,,,,,,,,,,
registro_1,fase_1,reloj_1,$,$(1),campo_2(2),$(3),campo_4(4),$(5),#,,,,,,
registro_2,fase_1,reloj_1,$(1),campo_2(2),$(3),campo_4(4),$(5),$,#,,,,,,
registro_3,fase_1,reloj_1,campo_2(2),$(3),campo_4(4),$(5),$,$(1),#,,,,,,
registro_4,fase_1,reloj_1,$(3),campo_4(4),$(5),$,$(1),campo_2(2),#,,,,,,
registro_5,fase_1,reloj_1,campo_4(4),$(5),$,$(1),campo_2(2),$(3),#,,,,,,
registro_6,fase_1,reloj_1,$(5),$,$(1),campo_2(2),$(3),campo_4(4),#,,,,,,
,,,,,,,,,,,,,,,
escritura,WR,1,,,,,,,,,,,,
registro_1,fase_1,reloj_1,$,$(1),campo_2(2),$(3),campo_4(4),$(5),#,$,$(1),u0,$(3),u0,$(5),,
registro_2,fase_1,reloj_2,$(1),campo_2(2),$(3),campo_4(4),$(5),$,#,$(1),u2,$(3),u4,$(5),$,,
registro_3,fase_2,reloj_1,campo_2(2),$(3),campo_4(4),$(5),$,$(1),#,s0,$(3),s0,$(5),$,$(1),,
registro_4,fase_2,reloj_2,$(3),campo_4(4),$(5),$,$(1),campo_2(2),#,$(3),s4,$(5),$,$(1),s2,,
registro_5,fase_1,reloj_1,campo_4(4),$(5),$,$(1),campo_2(2),$(3),#,b0,$(5),$,$(1),b0,$(3),,
registro_6,fase_1,reloj_1,$(5),$,$(1),campo_2(2),$(3),campo_4(4),#,$(5),$,$(1),b11,$(3),b1111,,
,,,,,,,,,,,,,,,
truncado,WR,1,,,,,,,,,,,,
registro_1,fase_1,reloj_1,$,$(1),campo_2(2),$(3),campo_4(4),$(5),#,$,$(1),u0,$(3),u0,$(5),0,1,no
registro_2,fase_1,reloj_2,$(1),campo_2(2),$(3),campo_4(4),$(5),$,#,$(1),u2,$(3),u4,$(5),$,0,1,si
registro_3,fase_2,reloj_1,campo_2(2),$(3),campo_4(4),$(5),$,$(1),#,s0,$(3),s0,$(5),$,$(1),0,1,no
registro_4,fase_2,reloj_2,$(3),campo_4(4),$(5),$,$(1),campo_2(2),#,$(3),s4,$(5),$,$(1),s2,0,1,si
registro_5,fase_1,reloj_1,campo_4(4),$(5),$,$(1),campo_2(2),$(3),#,b0,$(5),$,$(1),b0,$(3),0,1,no
registro_6,fase_1,reloj_1,$(5),$,$(1),campo_2(2),$(3),campo_4(4),#,$(5),$,
$(1),b11,$(3),b1111,0,1,si
,,,,,,,,,,,,,,,
--FIN_ESPECIFICACIONES,,,,,,,,,,,,,,
```

8.6 Diagrama de Gantt

Task Name	Start Date	End Date	Q2			Q3			Q4		
			April	May	June	July	August	September	October	November	December
Trabajo fin de grado	01/06/16	17/11/16									
Aprender SystemVerilog	01/06/16	15/07/16									
Estandarizar especificaciones	01/07/16	31/08/16									
Generar traductor	01/09/16	07/10/16									
Generar los ficheros a mano	03/10/16	28/10/16									
Automatizar la generación de ficheros	17/10/16	17/11/16									
Realizar pruebas	01/11/16	17/11/16									