1. Simulación y verificación

1.1. Verificación

El proceso de verificación de un procesador consiste en asegurar que su comportamiento funcional coincide con la arquitectura y especificación definidas. Este proceso se realiza antes de la síntesis o implementación en hardware, y tiene como objetivo identificar errores de diseño, comprobar la correcta interacción entre módulos internos y garantizar que las instrucciones se ejecuten según lo establecido en el conjunto de instrucciones.

La verificación se lleva a cabo mediante la simulación del procesador en un entorno controlado, donde se generan estímulos de entrada, se observan las señales internas y se comparan los resultados obtenidos con los valores esperados. Esta comparación permite validar que el procesador responde correctamente bajo diferentes condiciones.

1.1.1. Testbench

Se desarrolló un testbench encargado de instanciar el procesador completo y de proveer los estímulos necesarios para la verificación del correcto funcionamiento del procesador.

El objetvio del testbench es generar instrucciones aleatorias que entran al sistema y que por medio de aserciones se realiza un análisis de las entradas que se colocan con las salidas que se obtienen y estas se comparan las salidas esperadas, y por medio de estas pruebas se busca el escenario en el caso que se presentan errores en caso contrario se demuestra que efectivamente el sistema se comparta como se espera.

1.1.2. Generación de instrucciones.

Para la generación de instrucciones se define un *interface* que permite generar y enviar instrucciones al procesador durante la verificación y simulación, este solo recibe como señales externas el reloj y un reset. Dentro de este se declara una señal que se encarga de contener la instrucción RISC-V generada, para luego ser enviada al procesador.

Para el objetivo de este procesador, que es ejecutar la Secuencia de Fibonacci se usaron 4 tipos de instrucciones ADD, ADDI, BEQ y JAL.

1.1.3. Formatos de instrucción

El *inteface* incluye funciones para construir cada instrucción de forma específica. Todas las instrucciones se arman colocando los campos correspondientes en sus posiciones dentro del formato binario según la arquitectura RISC-V. Y para las instrucciones usadas estos son los formatos a utilizar.

funct7	rs2	rs1	funct3	rd	opcode

Cuadro 1: Formato ADD

imm[11:0]	rs1	funct3	rd	opcode

Cuadro 2: Formato ADDI

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode

Cuadro 3: Formato BEQ

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode

Cuadro 4: Formato JAL

1.1.4. Ejecución aleatoria de instrucciones

Se implementa un módulo que es un entorno de verificación para un procesador RISC-V. Su objetivo es generar instrucciones aleatorias del conjunto `ADD`, `ADDI`, `BEQ` y `JAL`, enviarlas al procesador y comprobar que cada módulo interno funcione correctamente.

Se instancia la interfaz `microprocessor_if`, la cual construye instrucciones combinando registros e inmediatos aleatorios. Cada instrucción se asigna a `instruction` para ser enviada al procesador. Mediante directivas `define se crean accesos abreviados a señales internas como la ALU, el PC y el banco de registros.

Después de liberar el reset, el testbench ejecuta un millón de iteraciones. En cada una, `randcase` selecciona aleatoriamente una instrucción con probabilidades iguales. Para cada instrucción se calculan los valores esperados:

- **ADD/ADDI:** suma de operandos.
- **BEQ:** actualización del PC según la comparación de registros.
- **JAL:** dirección de salto y valor `PC+4` para `rd`.

1.1.5. Monitoreo de entradas y salidas de los módulos internos.

Los resultados de la ejecusión aleatoria de instrucciones alimentan las propiedades SVA, que comparan automáticamente el comportamiento real del procesador con el esperado, validando así la ALU, el PC y el banco de registros y a partir del monitoreo de señales se permite observar en tiempo real el comportamiento interno del procesador durante la ejecución de cada instrucción. A través de la herramienta *SimVision* se visualizan señales clave como el valor del PC, las operaciones realizadas por la ALU, el contenido de los registros y los cambios en las líneas de control. Esto facilita identificar si el procesador está decodificando, ejecutando y escribiendo resultados correctamente en cada ciclo.

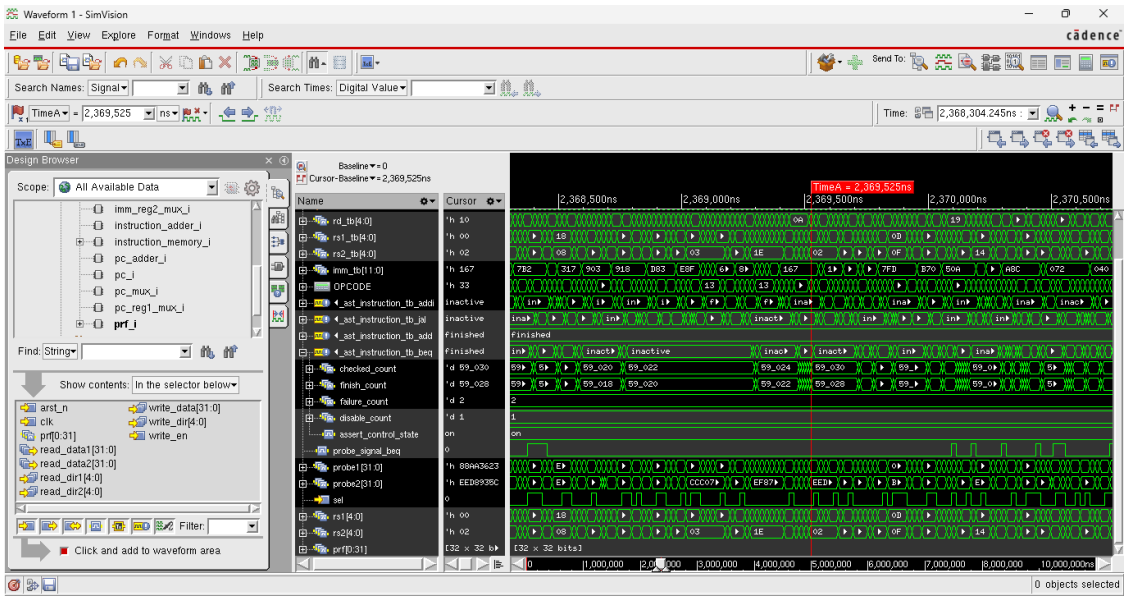

Figura 1: Señales de entradas y salidas

De esta manera, es posible analizar el comportamiento de cada módulo en el sistema.

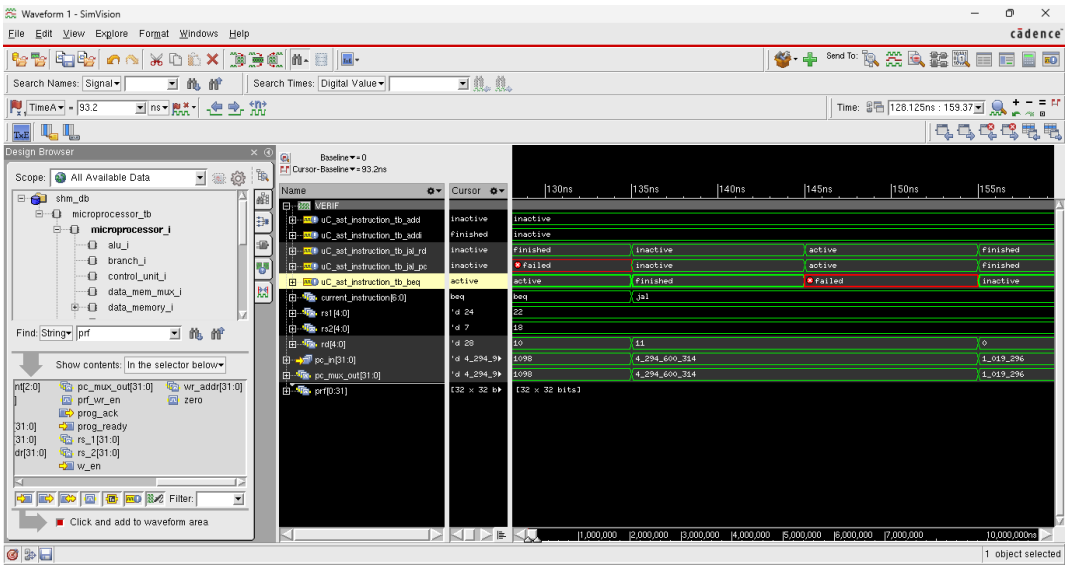

Figura 2: Resultados de verificación

Finalmente, la simulación se detiene automáticamente, garantizando que el análisis cubra un número suficiente de ciclos sin extender innecesariamente el tiempo de ejecución.

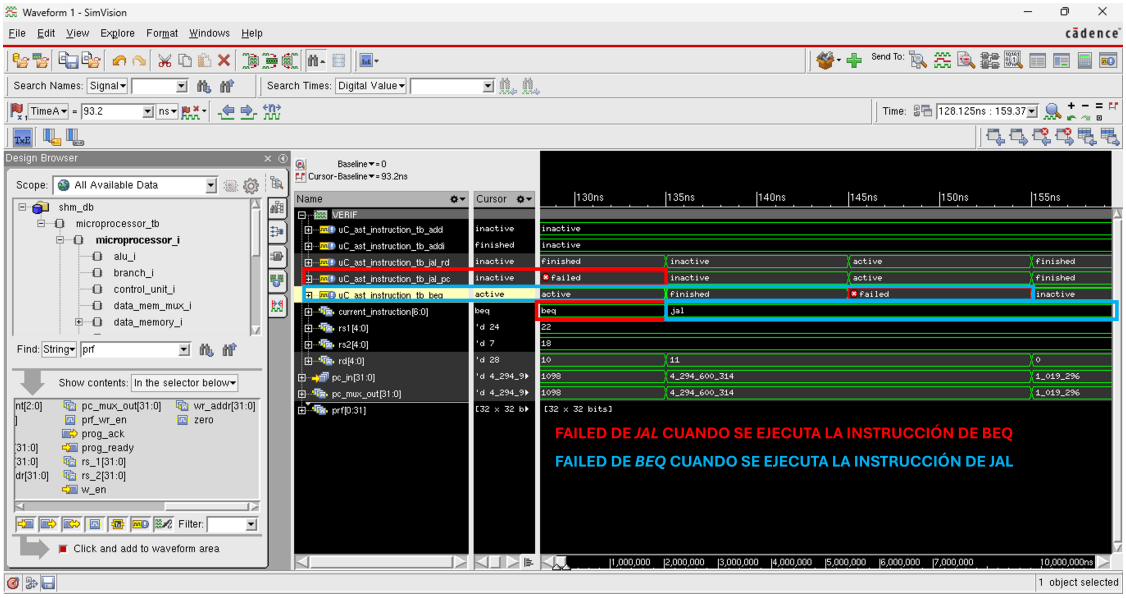

Figura 3: Enfoque de error en los resultados de la verificación

Como puede apreciarse en la Figura 3, la simulación de la verificación mediante aserciones sólo reporta un tipo de error: fallas en las instrucciones `jal` cuando en realidad se está ejecutando una beq, y viceversa. Tras un análisis exhaustivo de las señales involucradas y de su evolución temporal, se identificó que este comportamiento se debía a que la aserción evaluaba un flanco de señal posterior al ciclo en el que la instrucción correspondiente estaba activa, provocando así una desincronización entre la condición verificada y la instrucción efectivamente en ejecución.

1.2. Simulación

La simulación consiste en ejecutar el procesador con una secuencia de instrucciones definida, para comprobar que cumple una tarea específica. En este proyecto, el objetivo de la simulación es verificar que el procesador ejecuta correctamente la secuencia Fibonacci.

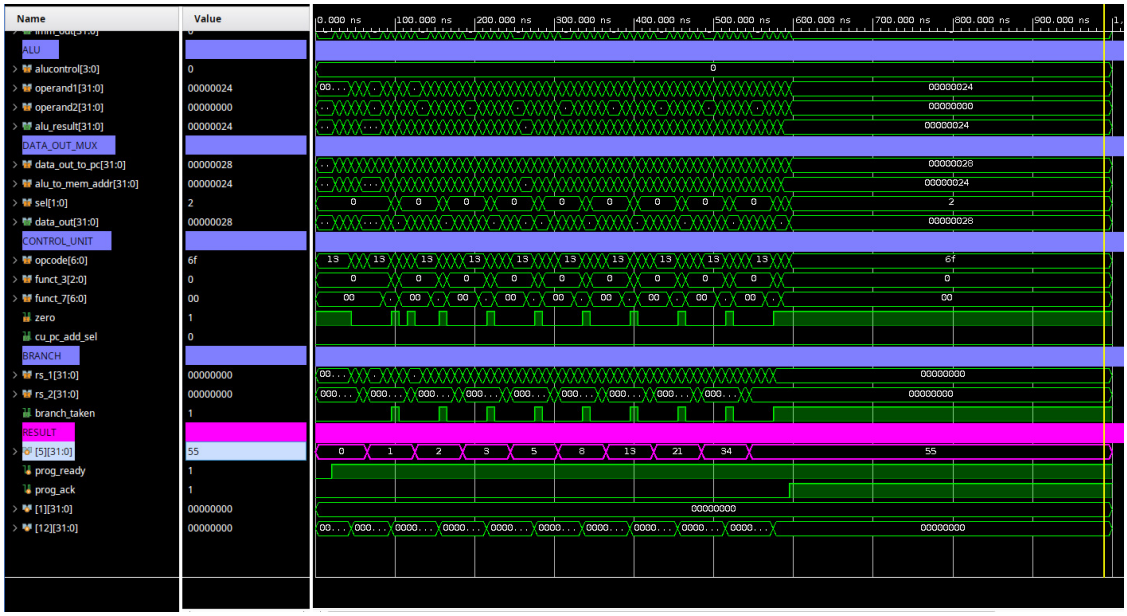

Figura 4: Resultados de Simulación en 10 iteraciones