Para este proyecto se realizó un testbench escrito en SystemVerilog, con las librerías de Universal Verification Methodology (UVM), para un un multiplicador combinacional de punto flotante que debe cumplir con el estándar IEEE-754 Single Precision Format de 32 bits. Este documento abarca el Test Plan y una serie de pasos a seguir para correr el código.

Para el desarrollo de este banco de pruebas se implementó la siguiente lista de tareas:

- Analizar el DUT.
- Búsqueda de información relacionada al DUT.
- Diseñar el objetivo del Testbench.
- Diseñar un test plan.
- Planificación del desarrollo del Testbench.
- Desarrollo del Ambiente de pruebas.
- Implementación de los diferentes escenarios y pruebas.
- Implementación de los reportes del Testbench.

A continuación se presenta un diagrama de bloques general, donde la clase agente fue combinada con la clase generador.

Test Plan

El Test Plan se diseñó con el objetivo de ejercitar el DUT de forma tal que se cubra la mayor cantidad de estados posibles, tanto para las máquinas de estado como para las señales. Para esto se busca implementar dos escenarios. El primero contiene pruebas en las que se busca alta alternancia entre los dispositivos y entre las señales que ingresan al DUT. El segundo busca cubrir casos en los que las entradas al DUT contienen valores inválidos para ejercitar su comportamiento en dichos casos.

Escenario 1: Uso común del dispositivo

Test 1.1 Ambos operandos, multiplicador y operando aleatorios:

Objetivo: Probar la capacidad del DUT que este caso es un multiplicador combinacional de punto flotante de 32 bits de precisión simple, por medio de multiplicación de operandos aleatorios.

Descripción: En este test se realizan un total de 500 multiplicaciones aleatorizando los operandos X y Y procurando no repetir eventos ni provocar errores como underflow, overflow y NaN.
Los bits correspondientes a sign, exponent y fraction se aleatorizan.
Las transacciones se generarán con retardos aleatorios independientes para cada operación.
Se correrán 50 semillas de esta prueba.

Criterio de aprobación: Se considera que el test será correcto si el comportamiento del dispositivo bajo prueba concuerda con el dispositivo simulado (Golden Reference).

Reportes: Al final de la prueba se imprimirá un reporte de cada una de las operaciones realizadas con sus respectivos resultados, incluidos el tiempo de creación de la operación y tiempo de retardo.

Requerimientos para el ambiente:

- Capacidad de aleatorización de: operandos X y Y, tiempo de retardo de las operaciones individuales.
- Constraint para evitar underflows.
- Constraint para evitar overflows.
- Constraint para evitar NaN.

Test 1.2 Máxima alternancia en los operandos:

Objetivo: Con este test se pretende maximizar el consumo energético y obtener datos en función del tamaño de la máxima alternancia de valores del DUT, además con esta prueba se busca llegar a un valor de cobertura medio-alto, ya que se forzarán valores de los operandos poco probables en las secuencias.

Descripción:
En este test se enviarán valores de los operandos tratando de buscar una máxima alternancia de bits, de manera que para el valueX y el valueY tendrán valores de F's, el siguiente en 0's, el siguiente en patrones de A's, el siguiente en patrones de 5's y luego se repetirá la secuencia. Cabe destacar que se correrán 50 semillas de esta prueba.

Criterio de aprobación: Se considera que el test es exitoso si las operaciones calculadas por el DUT son iguales a las del modelos de referencia.

Reportes: Al final de la prueba se imprimirá un reporte de cada una de las operaciones realizadas con sus respectivos resultados, incluidos el tiempo de creación de la operación y tiempo de retardo.

Requerimientos para el ambiente:

- Capacidad de aleatorización de: operandos X y Y, tiempo de retardo de las operaciones individuales.
- Constraint para evitar underflows.
- Constraint para evitar overflows.
- Constraint para evitar NaN.

Escenario 2: Comportamiento del DUT en casos de error

Test 2.1 Overflow

Objetivo: Verificar la respuesta del DUT ante distintas combinaciones de entrada que generen como resultado de la multiplicación un desbordamiento.

Descripción: Para este caso el DUT será excitado con valores para el multiplicando y el multiplicador con valores que vayan a causar un desbordamiento de bits en el resultado de la operación (valueZ).

Criterio de aprobación: Se considera que el test será correcto si se obtiene el valor Inf a la salida para las combinaciones de X y Y que deban generarlo, a la vez que se activa la bandera de overflow.

Reportes: Al final de la prueba se imprimirá un reporte de cada uno de los paquetes enviados incluidos el tiempo de envío en la fuente, tiempo de recibido en el destino y retardo total de transito del paquete.

Requerimientos para el ambiente:

- Capacidad de aleatorización de: operandos X y Y, tiempo de retardo de las operaciones individuales.
- Constraint para evitar underflows.
- Constraint para evitar NaN.

Test 2.2 Underflow.

Objetivo: Ejercitar el DUT creando situaciones variadas (diferentes combinaciones) entre el valueX y el valueY donde se de como resultado de la operación una situación de underflow.

Descripción:
En este test se realizarán operaciones que den como resultado de la multiplicación un underflow, esto se logra mediante la combinación de un multiplicando y el multiplicador.

Criterio de aprobación: Se considera que el test será correcto si para las combinaciones de X y Y cuyo producto (en valor absoluto) sea menor al valor mínimo permitido por el estándar, se obtenga un resultado de 0 a la salida y se active la bandera de underflow.

Reportes: Al final de la prueba se imprimirá un reporte de cada uno de los paquetes enviados incluidos el tiempo de envío en la fuente, tiempo de recibido en el destino, dirección de destino, dirección de fuente y retardo total de transito del paquete.

Requerimientos para el ambiente:

- Capacidad de aleatorización de: operandos X y Y, tiempo de retardo de las operaciones individuales.
- Constraint para evitar overflows.
- Constraint para evitar NaN.

Test 2.2 NaN.

Objetivo: Ejercitar el DUT en situaciones variadas donde se de resultados de NaN.

Descripción: En esta prueba se pretende 'generar un error ' por medio de la asignación de un valueX y un valueY donde se genere un resultado NaN (Not a Number). Esto se logra por medio de la multiplicación de cero por infinito ó infinito por cero.

Criterio de aprobación: Esta prueba debe de encender una bandera de NaN.

Reportes: Al final de la prueba se imprimirá un reporte de cada uno de los paquetes enviados incluidos el tiempo de envío en la fuente, tiempo de recibido en el destino, dirección de destino, dirección de fuente y retardo total de transito del paquete.

Requerimientos para el ambiente:

- Capacidad de aleatorización de: operandos X y Y, tiempo de retardo de las operaciones individuales.
- Constraint para evitar overflows.
- Constraint para evitar underflows.

Pasos para Correr el TestBench

Para correr el Testbench es necesario especificar el escenario y el número de prueba que se desea realizar, tales se muestran en el archivo llamado *Top.sv* en la clase Top. En la línea 40 se cambia dicho dato. Una vez compilado el programa se corre el archivo *testbench.sv*, además se debe de generar un archivo .csv como reporte de paquetes enviados y recibidos. Por medio de la aplicación GNUplot se grafican los datos.

Pasos que se siguieron para encontrar y solucionar el Bug

Identificación: Las aserciones agregadas en la interfaz arrojaron mensajes de error que indicaban que las banderas de overflow y underflow no se encendían en algunos casos.

```
18    property und;
19      @(negedge clk) (~|cb.fp_Z[30:23] & ~cb.fp_Z[22]) |-> cb.udrf;
20    endproperty
21
22    property ovr;
23      @(negedge clk) (&cb.fp_Z[30:23] & ~cb.fp_Z[22]) |-> cb.ovrf;
24    endproperty
25
26     a_und: assert property (und) else $display("Underflow Flag Error");
27     c_und: cover property (und) $display("Underflow Flag Pass");
28
29
30     a_ovr: assert property (ovr) else $display("Overflow Flag Error");
31     c_ovr: cover property (ovr) $display("Overflow Flag Pass");
32
33
34  endinterface
```

```
"interface.sv", 30: top_testbench._if.a_ovr: started at 420ns failed at 420ns
        Offending 'dut_if.cb.ovrf'
Overflow Flag Error
```

```
"interface.sv", 26: top_testbench._if.a_und: started at 220ns failed at 220ns
        Offending 'dut_if.cb.udrf'
Underflow Flag Error
```

Descripción: Observando los operandos multiplicando y multiplicador se pudo notar que los errores se daban en dos circunstancias específicas: Cuando uno de los operando presentaba underflow y el otro no y cuando uno de los operando presentaba overflow y el otro no. En el primer caso se obtiene un resultado con underflow pero sin generación de bandera de underflow, mientras que en el segundo de manera similar se obtiene un resultado con overflow pero sin la debida generación de la bandera de overflow

Solución: De la línea 709 a la 729, donde se daba la generación de las señales de overflow y underflow, se le agregó la funcionalidad de poder generar estas señales cuando alguno de los dos operandos presenta esta condición

```
709 module EXP(
710   input norm,
711   input [7:0]exp_X, exp_Y,
712   output [7:0]exp_Z,
713   output ovrf, udrf);
714
715   wire [8:0]buffer;
716
717   assign buffer = exp_X + exp_Y;
718
719   wire [7:0]bias;
720
721   assign bias = {7'b0111111, !norm};
722
723-  assign ovrf = {buffer >= {255 + bias}};
724-  assign udrf = {buffer <= bias};
725
726   assign exp_Z = exp_X + exp_Y - bias;
727
728 endmodule
729
```

```
709 module EXP(
710   input norm,
711   input [7:0]exp_X, exp_Y,
712+  input nan,
713   output [7:0]exp_Z,
714   output ovrf, udrf);
715
716   wire [8:0]buffer;
717+  wire inf_X, inf_Y;
718+  wire zer_X, zer_Y;
719
720   assign buffer = exp_X + exp_Y;
721
722   wire [7:0]bias;
723
724+  assign zer_X = ~|exp_X;
725+  assign zer_Y = ~|exp_Y;
726+
727+  assign inf_X = &exp_X;
728+  assign inf_Y = &exp_Y;
729+
730   assign bias = {7'b0111111, !norm};
731
732+  assign ovrf = {!nan}&{{buffer >= {255 + bias}}|{inf_X}|{inf_Y}};
733+  assign udrf = {!nan}&{{buffer <= bias}|{zer_X}|{zer_Y}};
734+
735
736   assign exp_Z = exp_X + exp_Y - bias;
737
738 endmodule
739
```

```
820
821   EXP EXP(.norm(norm_n | norm_r),
822           .exp_X(exp_X),
823           .exp_Y(exp_Y),
824
825           .exp_Z(exp_Z),
826           .ovrf(ovrf),
827           .udrf(udrf));
```

```
830
831   EXP EXP(.norm(norm_n | norm_r),
832           .exp_X(exp_X),
833           .exp_Y(exp_Y),
834+          .nan(nan),
835
836           .exp_Z(exp_Z),
837           .ovrf(ovrf),
838           .udrf(udrf));
```

Proyecto 2–Testbench con aleatorización controlada

DUT: Multiplicador

A. Delgado, A. Vega, F. Rojas

Verificación Funcional de Circuitos Integrados

Tecnológico de Costa Rica

II Semestre, 2020