SISTEMAS DIGITALES PROGRAMABLES

2022/23

Tema 2 - Verificación lógica de sistemas digitales

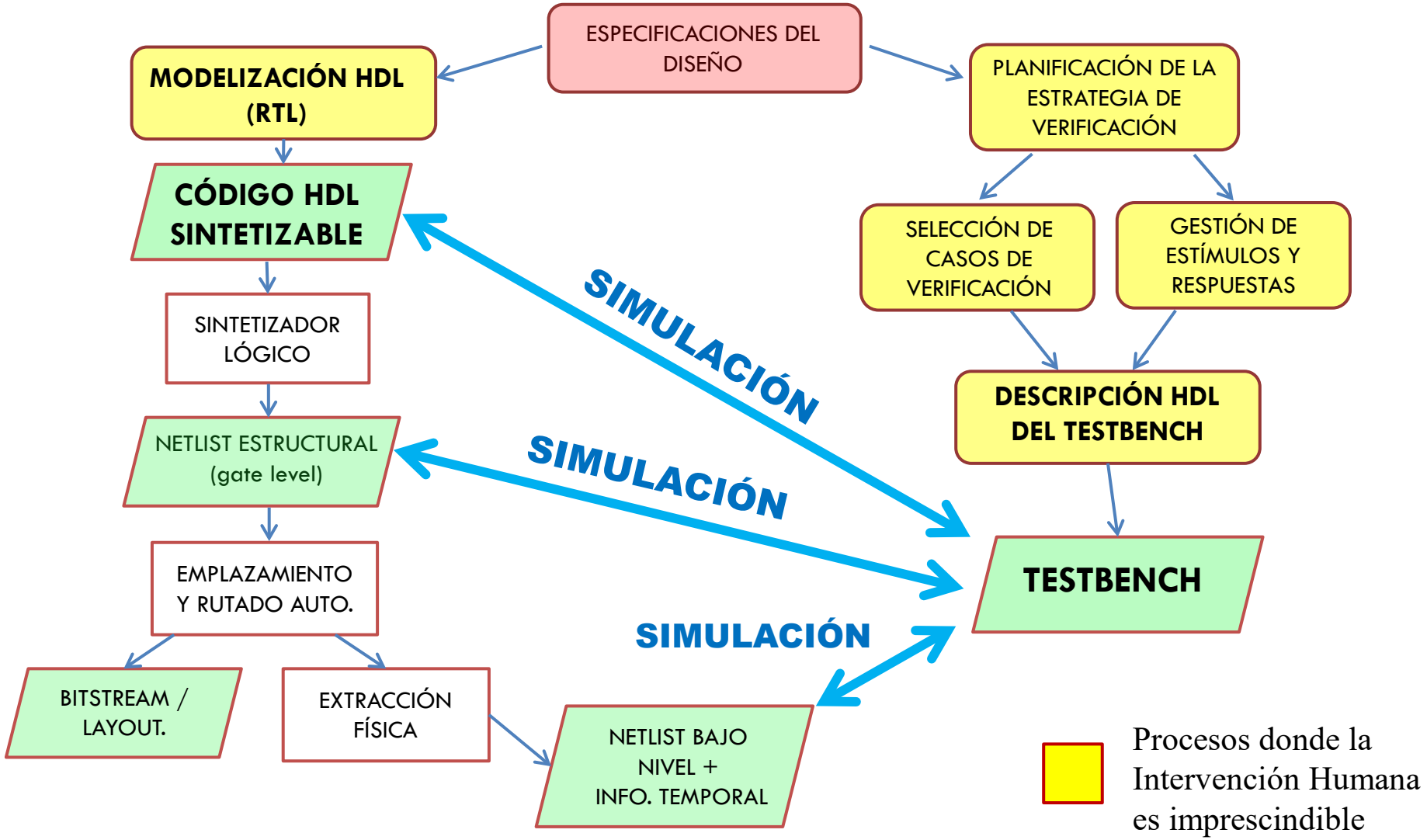
Objetivos

- Conocer la metodología de verificación lógica empleando Verilog basada en un Testbench estructurado.
- Aplicar las técnicas para generación de estímulos y comprobación de respuestas en un Testbench.

Estructura

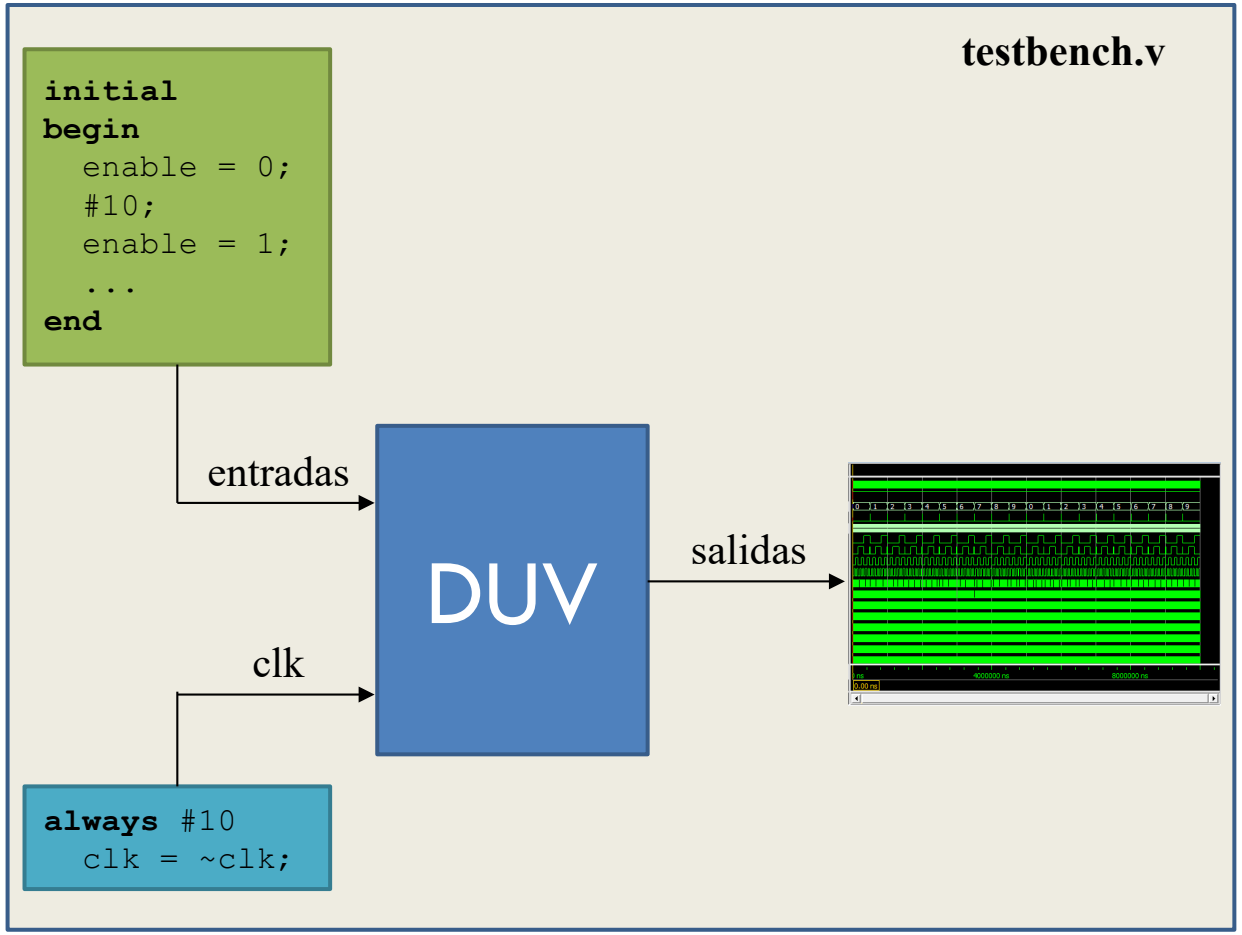
- Planificación de la Verificación
- Estructura del Testbench
- Verilog para Verificación
  - Generación de Estímulos
  - Comprobación de Respuestas
  - Ejemplos

Verificación Lógica con Verilog HDL



Testbench básico

- **Testbench básico:**
  - "Mesa de laboratorio" virtual en la que probaremos el funcionamiento del circuito
  - Contiene:
    - Instancia del circuito a verificar (DUV)
    - Generación de señal de reloj
    - Generación de señales de entrada
    - Observación de las salidas puede ser visual
  - No tiene entradas ni salidas



Testbench básico


```
`timescale 1ns/100ps

module tb_counter_with_divider ();

   localparam T=20;

   // DUV instance

   reg CLK, RSTn;
   wire [3:0] COUNT;

   counter_with_divider duv (
      .CLK(CLK),
      .RSTn(RSTn),
      .COUNT(COUNT)
      );

   // Clock generation

   always
   begin
      #(T/2) CLK = ~CLK;
   end

   // Test procedure

   initial
   begin
      CLK = 0;
      RSTn = 0;
      #(T*2)
      RSTn = 1;
      #10000000; // 10 ms

      $display("Test finished");
      $stop;
   end

endmodule
```

Especifica escala temporal

Ni entradas ni salidas

Declaración de señales internas del testbench:
entradas y salidas del DUV

- Salidas → **wire**
- Entradas → **reg** normalmente (damos valor en un proceso)

Instanciación del DUV

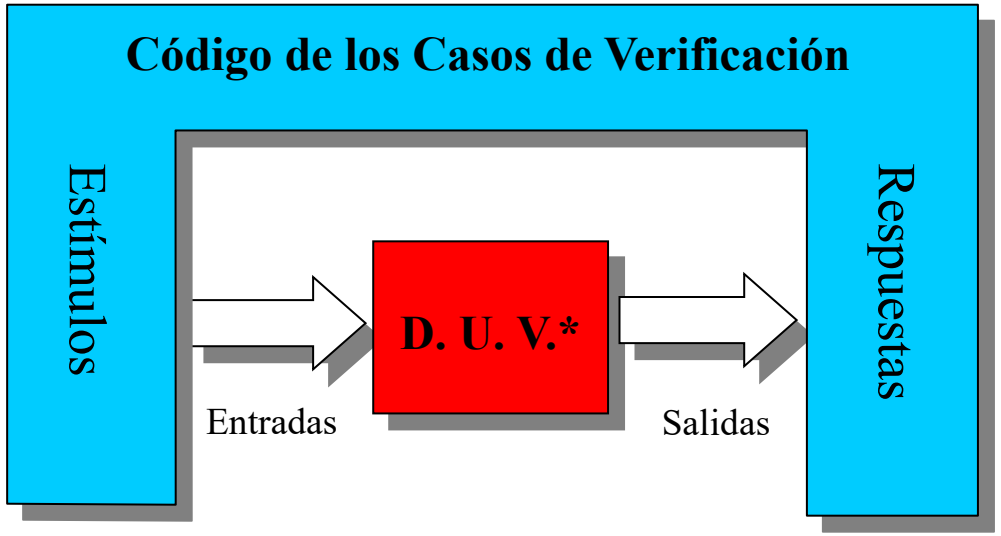
Generación de
reloj cuadrado

Generación de
entradas y salidas

Verificación Lógica con Verilog HDL

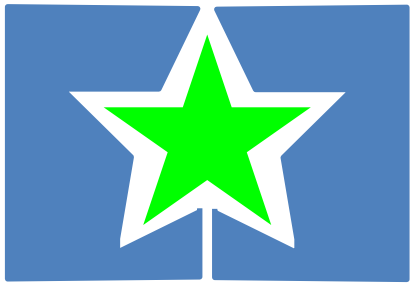
- **Planificación de la Verificación**

TESTBENCH



*Device Under Verification

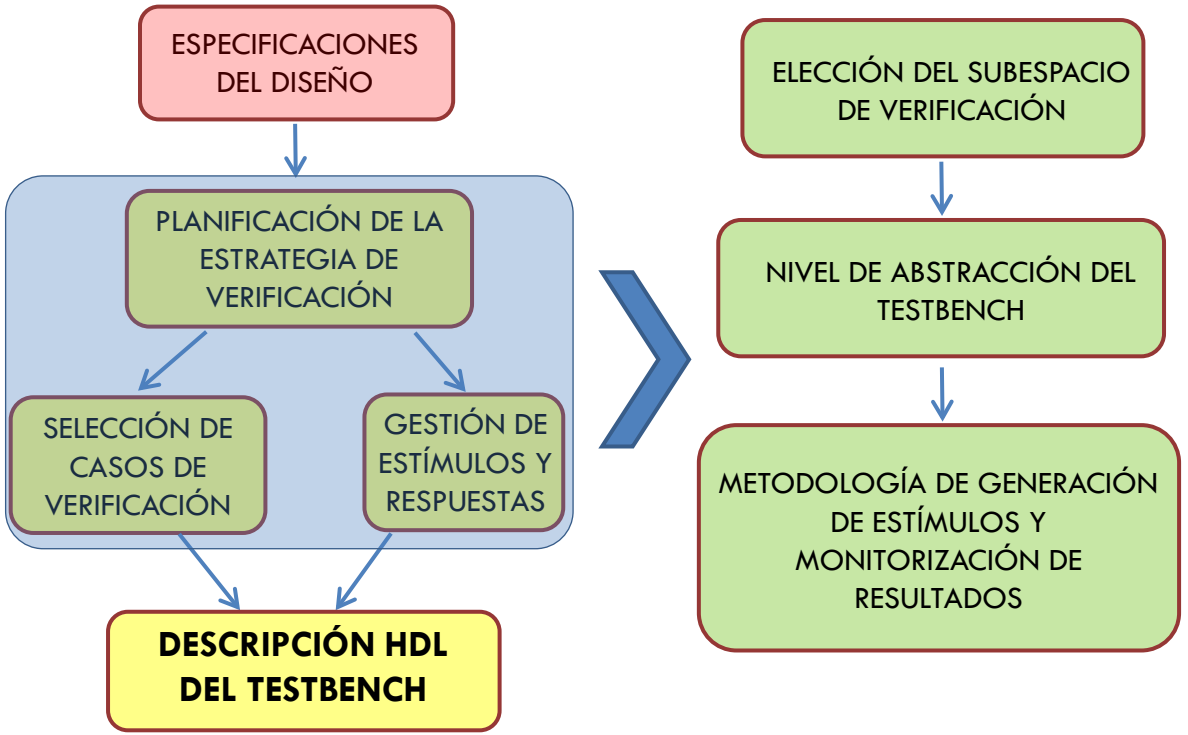
El proceso de verificación funcional depende del diseño y sus características



TESTBENCH

Verificación Lógica con Verilog HDL

Planificación de la Verificación



Espacio completo de verificación demasiado grande para una cobertura total

Operaciones de Bajo Nivel — **− Nivel Abstracción +** → **Operaciones más complejas**

"Golden Models"
Generación Aleatoria / Dirigida de estímulos
Aserciones

Verificación Lógica con Verilog HDL

Planificación de la Verificación

Elección del Subespacio de Verificación

ELECCIÓN DEL SUBESPACIO DE VERIFICACIÓN

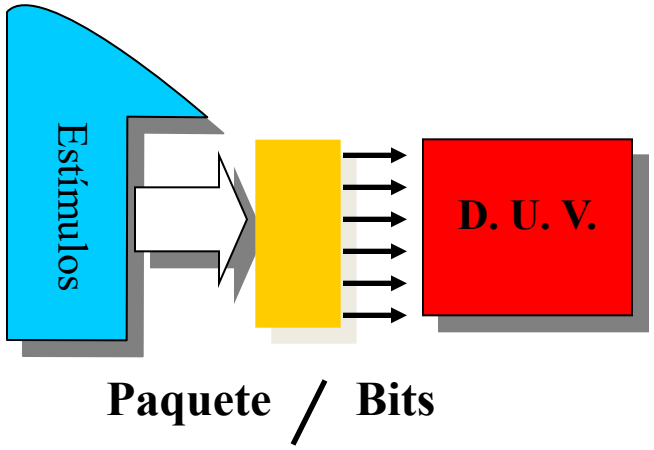
- **VERIFICACIÓN POR CARACTERÍSTICAS**
  Cobertura de las **especificaciones** iniciales
- **VERIFICACIÓN POR INTERFACES**
  Funcionamiento de los **interfaces de comunicación** (Buses Estándar de Comunicación)
- **VERIFICACIÓN POR *CORNER CASES***
  Casos Extremos **no cubiertos por las especificaciones**
  Relacionados con la estabilidad del diseño (pueden depender de la implementación)

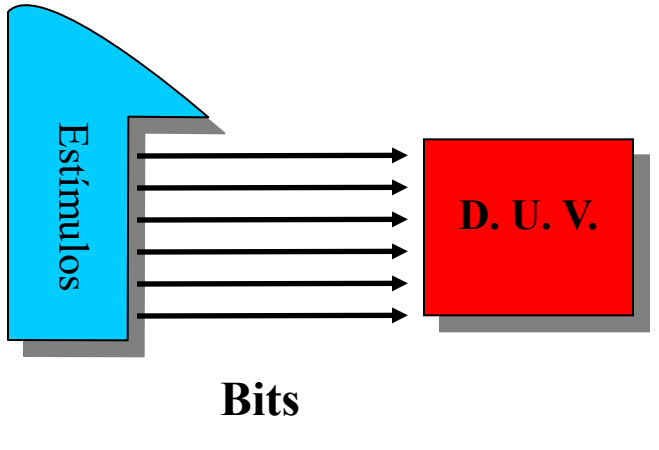
Verificación Lógica con Verilog HDL

- **Planificación de la Verificación**
  - **Nivel de Abstracción del Testbench**

Mayor Abstracción → Menor Control pero Mayor flexibilidad

Menor Abstracción → Mayor Control pero Menor flexibilidad





Verificación Lógica con Verilog HDL

Planificación de la Verificación

Generación de Estímulos y Monitorización de Resultados

METODOLOGÍA DE GENERACIÓN DE ESTÍMULOS Y MONITORIZACIÓN DE RESULTADOS

"Golden Models"
Fase de especificación en un lenguaje común entre cliente y diseñador (Python, C++ etc.)

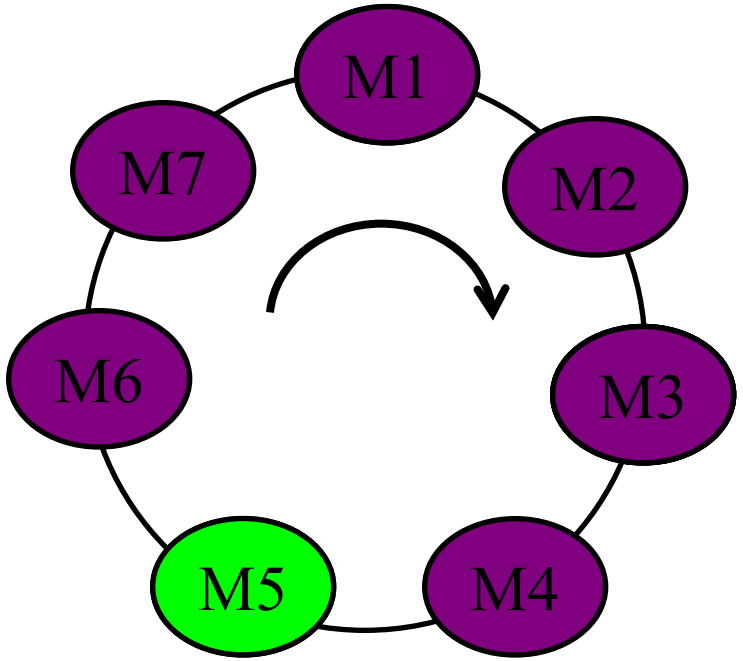
Co-simulación del modelo con el DUV → Adaptación de la filosofía de simulación (event-based / time-based)

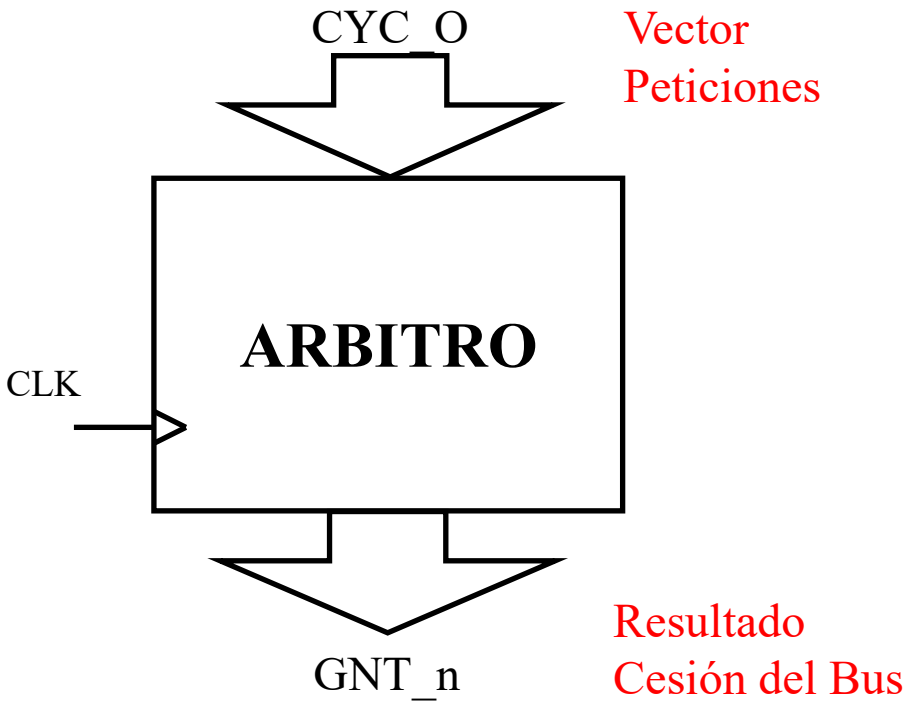
Generación Dirigida / Aleatoria de estímulos
Incrementar el espacio de verificación añadiendo una componente aleatoria

Aserciones
Mecanismos autónomos de verificación insertados en el código RTL

Verificación Lógica con Verilog HDL

- **Planificación de la Verificación**
  - **Ejemplo: Arbitro Prioridad "*ROUND-ROBIN*"**





Verificación Lógica con Verilog HDL

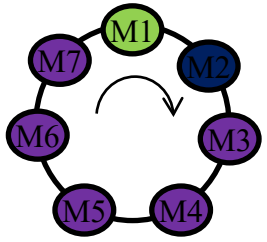
Planificación de la Verificación

Ejemplo: Arbitro Prioridad "*ROUND-ROBIN*" (II)

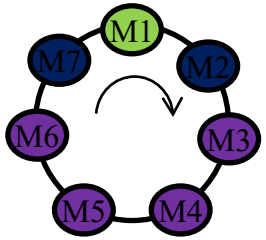
CASOS DE VERIFICACIÓN

POR CARACTERÍSTICAS

- Distintas situaciones de paso de testigo



Cada posición del bus (exhaustiva)

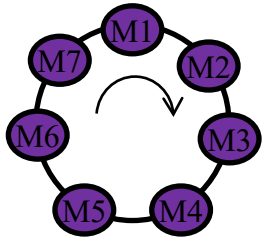


Sentido de asignación de prioridad

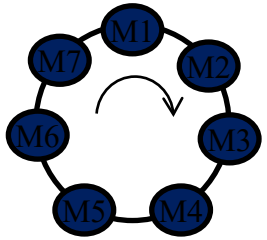
- Asignación del bus en un ciclo de reloj

POR CASOS EXTREMOS

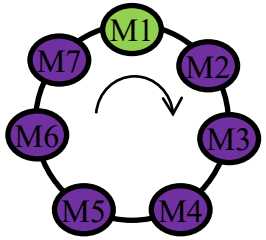
- Casos no descritos en especificaciones



Nadie tiene el bus y nadie lo pide



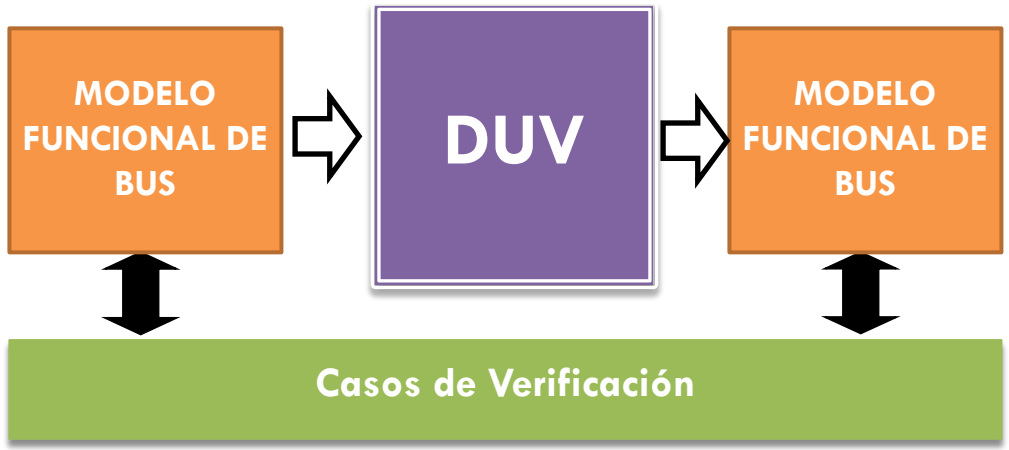
Nadie tiene el bus y todos lo piden



Un máster termina con el bus y nadie lo pide

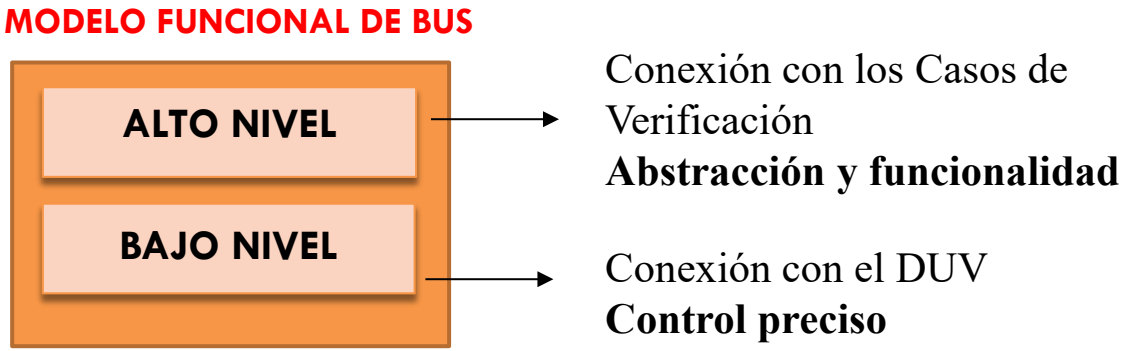
Verificación Lógica con Verilog HDL

Estructura del TESTBENCH



Modelo Funcional de Bus:
Interfaz de los Casos de Verificación con el DUV

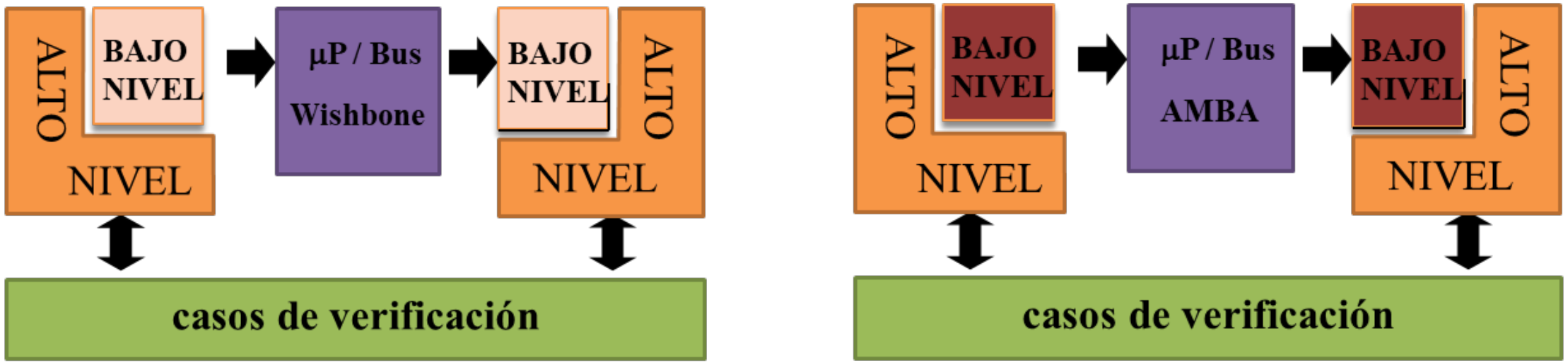
Posible de Reuso
Posible Incremento del Nivel de Abstracción



Se define un interfaz con las operaciones de alto nivel realizables.
En caso de reuso se cambia la implementación de bajo nivel de esas operaciones.

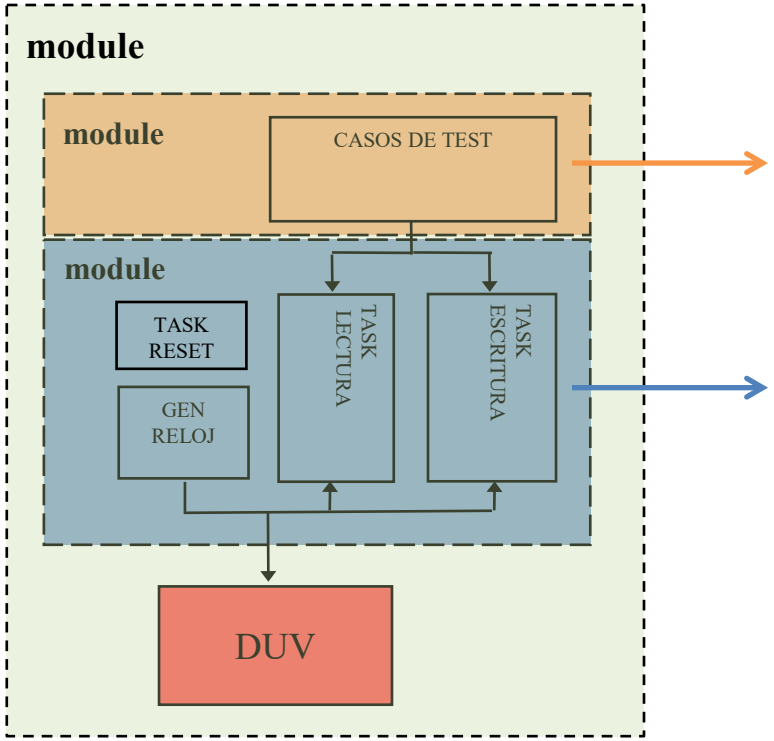
Verificación Lógica con Verilog HDL

- **Estructura del TESTBENCH. Ejemplo de Reuso**



Verificación Lógica con Verilog HDL

Estructura del TESTBENCH. Ejemplos (I)



Nivel de Abstracción lo más elevado posible

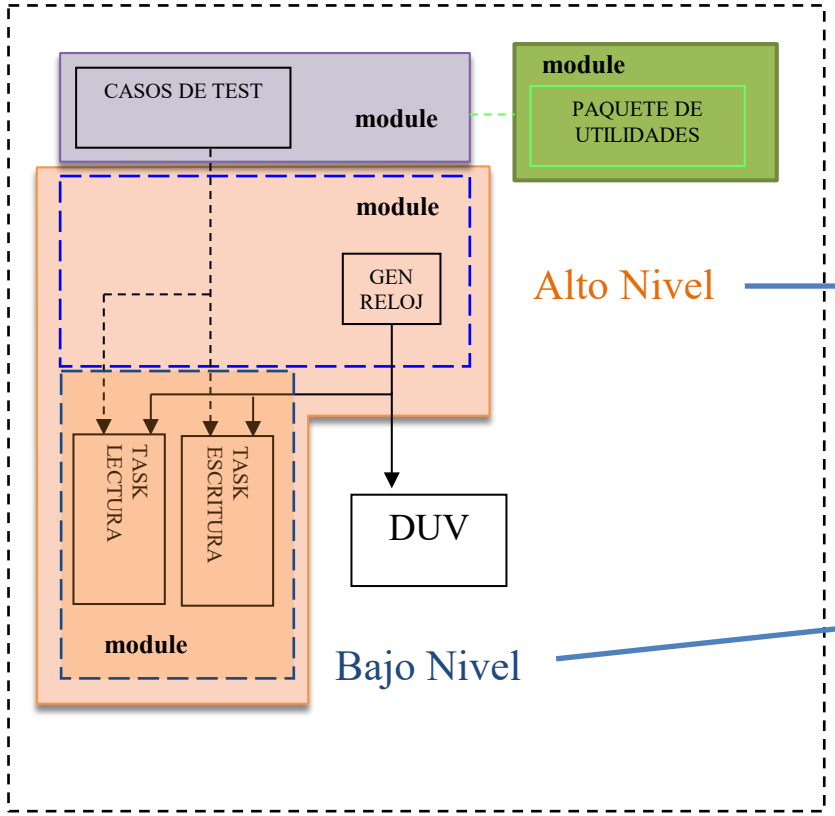
Interfaz con el nivel de implementación del DUV

Los resultados se comprobarán mediante:

- **Modelo Funcional de Referencia**
- **Conjunto de Vectores (*golden vectors*) de respuestas conocidas**

Verificación Lógica con Verilog HDL

Estructura del TESTBENCH. Ejemplos (II)

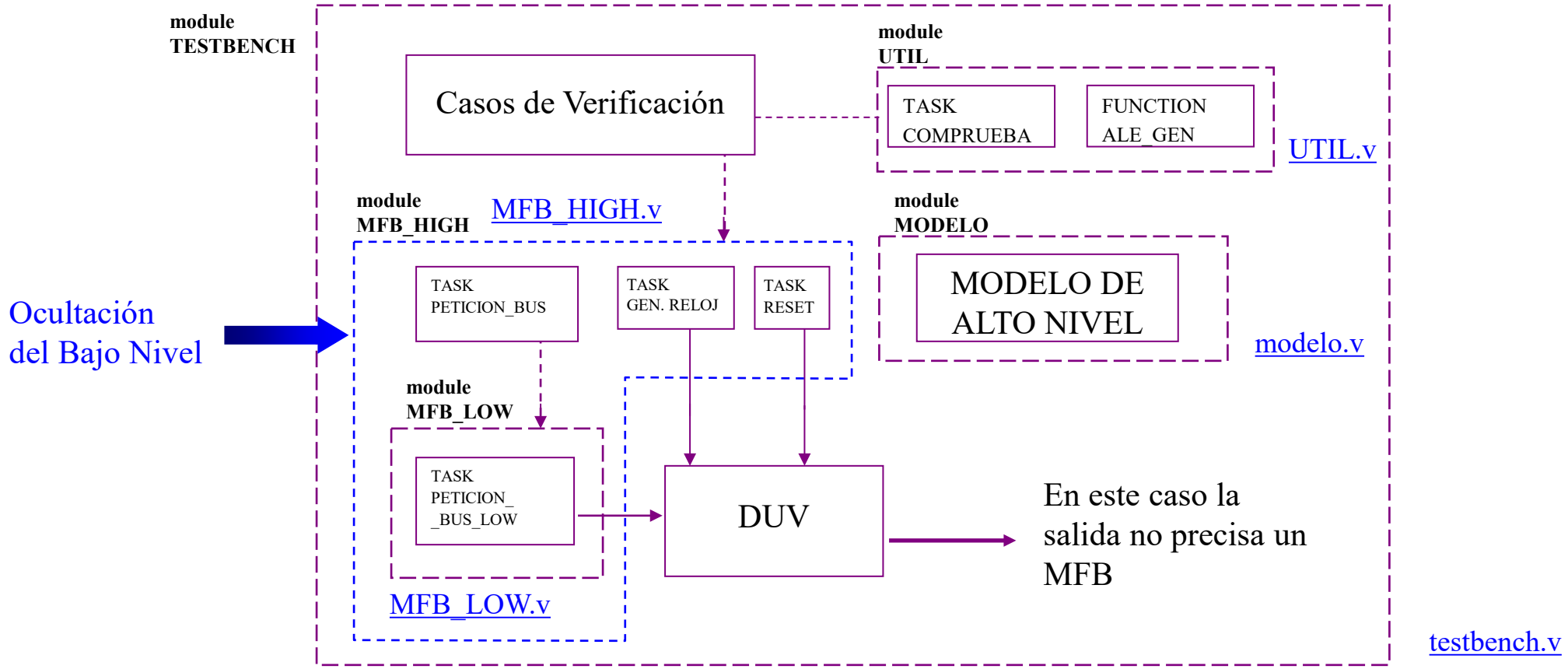


Ocultación de las Task de Bajo Nivel + Generación de Señales Globales

Interfaz con el nivel de implementación del DUV

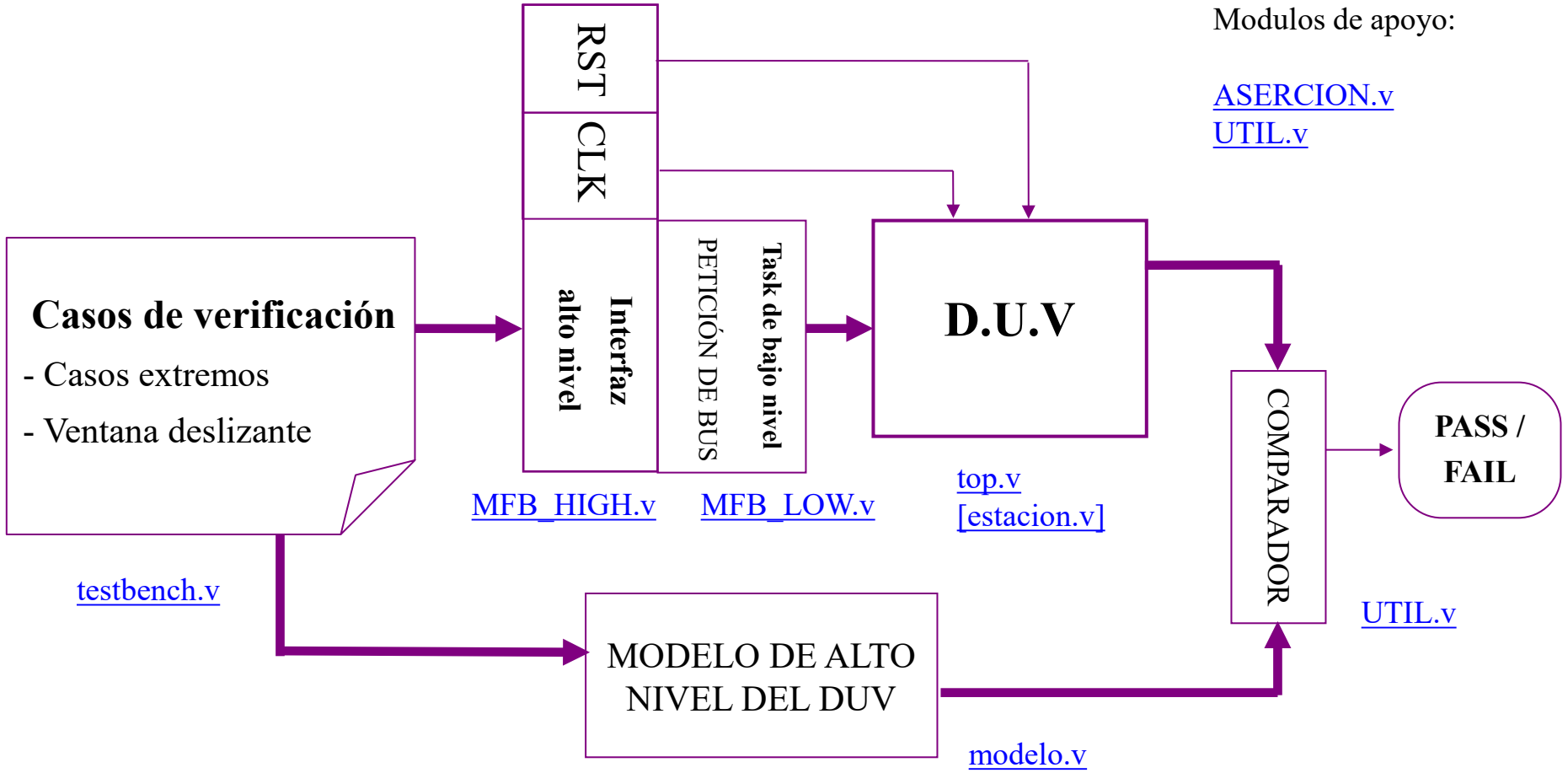
Verificación Lógica con Verilog HDL

Estructura del TESTBENCH. Ejemplos (III)



Verificación Lógica con Verilog HDL

Estructura del TESTBENCH – Flujo de Datos. Ejemplos (IV)

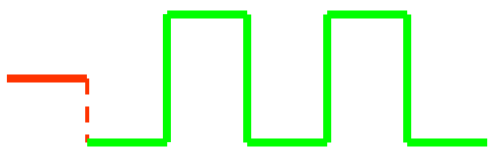


Verificación Lógica con Verilog HDL

Verilog para Verificación

Generación de Estímulos. Relojes

```
`timescale 1 ns / 100 ps;
reg clk;
parameter periodo = 10;
// Periodo = 10ns
always
begin
#(periodo/2);
clk = 1'b0;
#(periodo/2);
clk = 1'b1;
end
```



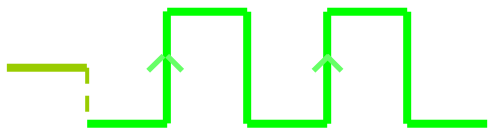
- Relojes asimétricos
- Frecuencia configurable
- Uso de la directiva *timescale* (precisión temporal)

Verificación Lógica con Verilog HDL

Verilog para Verificación

Generación de Estímulos. Generación del RESET

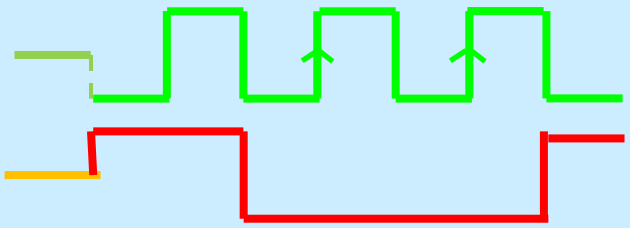
```
always
begin
#50 clk  =  1'b0;
#50 clk  =  1'b1;
end
```



```
initial
Begin
rst = 1'b1;
rst = 1'b0;
#150 rst  =  1'b1;
end
```

POSIBLE CONDICIÓN DE CARRERA EN EL DUV

```
task reset;
        //Generador de reset
        begin
        rst <= 1'b1;
        wait (clk !== 1'bx);
        @(negedge clk);
        rst <= 1'b0;
        @(negedge clk);
        @(negedge clk);
        rst <= 1'b1;
        end
endtask
```



Verificación Lógica con Verilog HDL

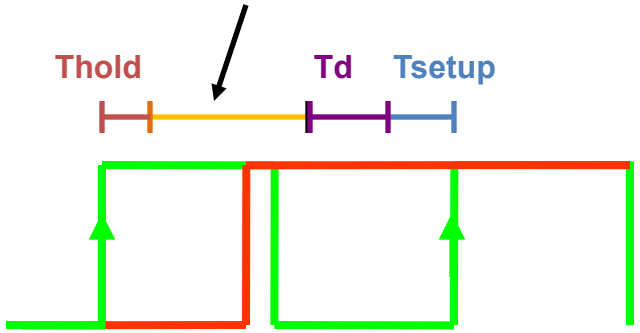
Verilog para Verificación

Generación de Estímulos. Temporización Aleatoria

```
task peticion_bus_low;

        input [(masters-1):0] vector_masters;

        real A,B;

        begin

        A = ({$random} % 100);
        B = A / 100;

        @(posedge clk);
        #(Thold);
        #((periodo -Tsetup -Td-Thold)*B);
        // Temporizacion aleatoria de las señales

        CYC_O <= vector_masters;

        end
endtask
```

Respetamos los tiempos de Hold y Set -Up

Margen de incertidumbre temporal en la introducción del dato



Recordatorio del Tema 1…

Elementos del Módulo. Funciones y Tareas

```
function <rango> <nombre>;
        <declaraciones entradas>
        <declaraciones variables>
        begin
          <estamentos>
        end
endfunction
```

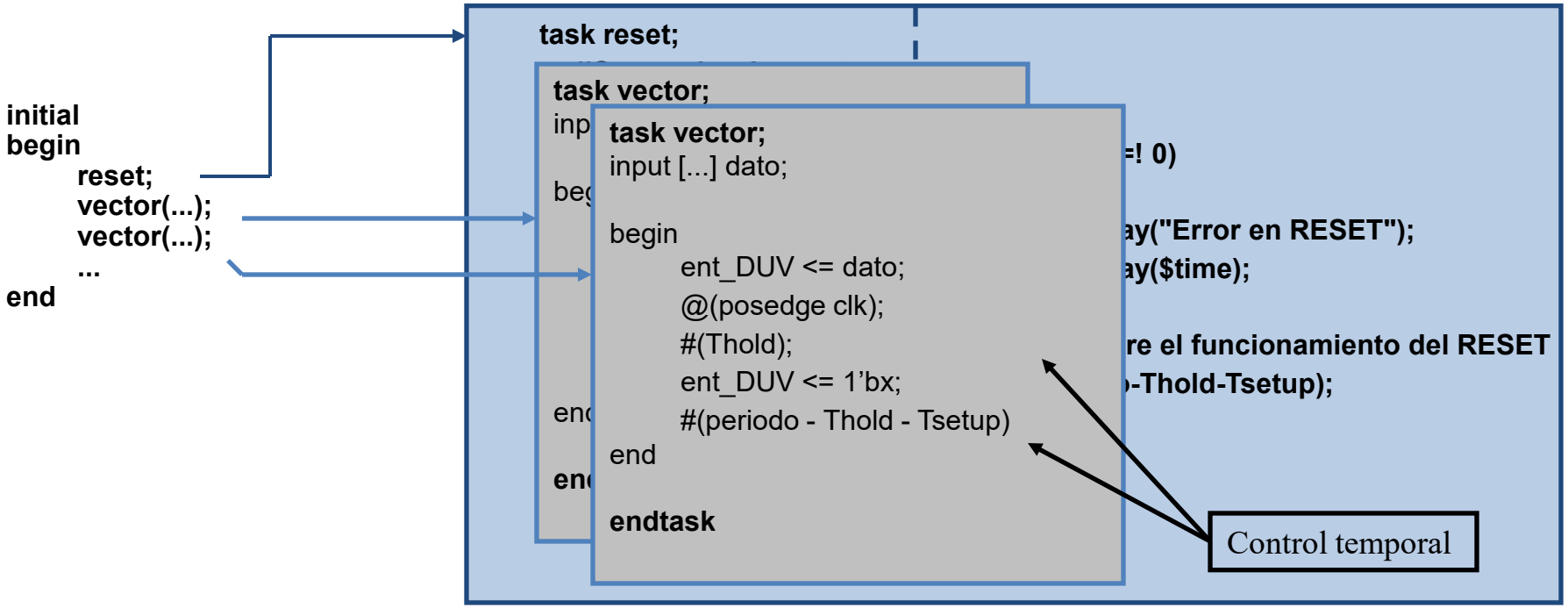
- Se ejecuta en 0 de tiempo de simulación
- No admite controles temporales
- Debe tener al menos una entrada
- Solo puede devolver un valor

```
task <task_name>;
        <declaraciones entradas/salidas>
        <declaraciones variables>
        begin
          <estamentos>
        end
endtask
```

- Puede ejecutarse durante un tiempo de simulación diferente de 0
- Puede contener controles temporales
- Puede tener argumentos de entrada y salida arbitrarios (o no tener argumentos)

Verificación Lógica con Verilog HDL

- **TestBench**
  - **Encapsulado de la generación de estímulos**



Las construcciones *task* son idóneas para el encapsulado pues permiten un control temporal y una ejecución paralela.

Verificación Lógica con Verilog HDL

- **Testbench**
  - **Encapsulado de la Comprobación de Respuestas (Golden Vectors)**

```
task comprueba_vector;
input [...] datos_ent;
input [...] datos_sal;
begin
        ent_DUV <= datos_ent;
        @(posedge clk);
        fork
                begin
                #(Thold);
                ent_DUV <= ...'bx;
                end

                begin
                #(Td);
                if (sal_DUV !== datos_sal) ... ;
                end
        join

        #(periodo - Td - Tsetup);
end
endtask
```

Introduce el dato y espera el siguiente Flanco Activo

Desactiva la entrada de datos (Opcional)

Comprueba las salidas

Procesos Paralelos

Obliga la espera hasta que sea posible introducir otros datos

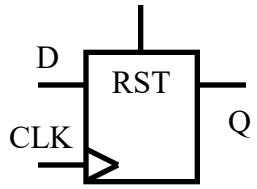
Verificación Lógica con Verilog HDL

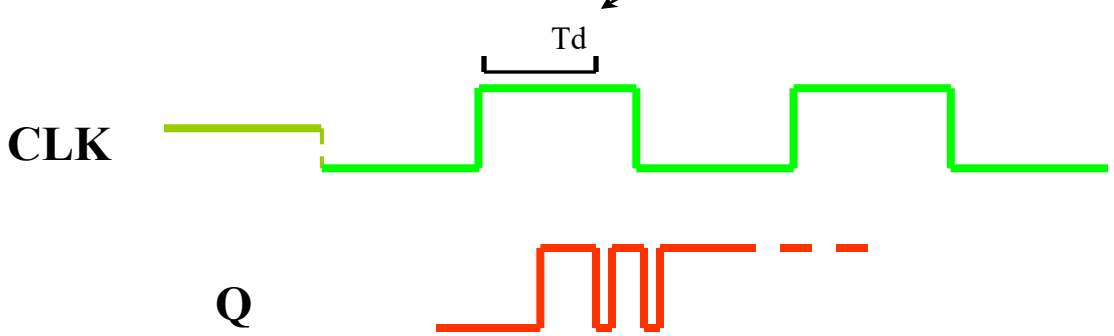
Verilog para Verificación

Comprobación de Respuestas. Monitor de Estabilidad

```
initial
begin
        @ (negedge rst);  //Espera a que el reset esté inactivo
        @ (posedge clk);     //Espera al siguiente ciclo activo
      forever
      begin
          #(Td);
          fork: Monitor_Estabilidad
              @ (Q) $display('Problemas de estabilidad con Q');
              @ (posedge clk) disable Monitor_Estabilidad;
          join
      end

end
```

} Condición para que comience el chequeo de estabilidad



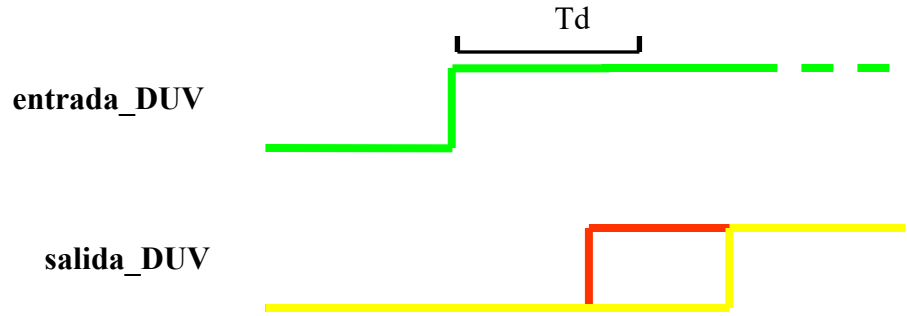


Verificación Lógica con Verilog HDL

Verilog para Verificación

Comprobación de Respuestas. Delay Check

```
task Chequeo_temporal;
        output correcto;
begin
        entrada_DUV = 1'b1;              // Se genera la señal de entrada
        fork: bomba_tiempo
            #Td                          disable bomba_tiempo;
            @(posedge salida_DUV)        disable bomba_tiempo;
        join
        correcto = ( salida_DUV == 1'b1 );
end
endtask
```



Verificación Lógica con Verilog HDL

Verilog para Verificación

Comprobación de Respuestas. Watchdog

```
task comprueba;
input resultado;
input timeout;
time timeout;

fork: bloque_con_watchdog
        begin
        #(timeout);
        $display("Error de timeout");
        disable bloque_con_watchdog;
        end

        begin
        @ (señal_validadora_salida_DUV);
        if (salida_DUV !== resultado)    $display("Error en salida del DUV");
        disable bloque_con_watchdog;
        end
join
endtask
```

Watchdog

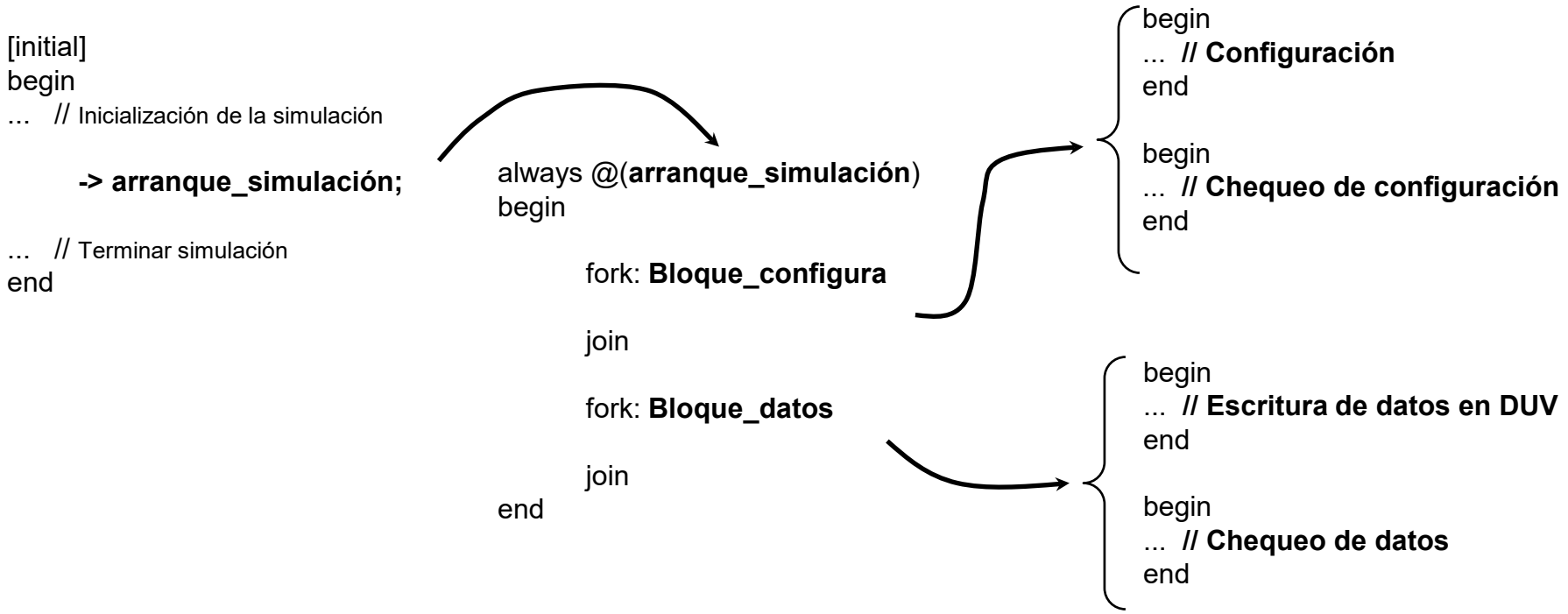
Si el diseño tiene una latencia variable podemos emplear una señal de validación de datos de salida como aquí.

Si conocemos el delay del DUV podemos emplear directamente un retardo del tipo "#(delay)"

Verificación Lógica con Verilog HDL

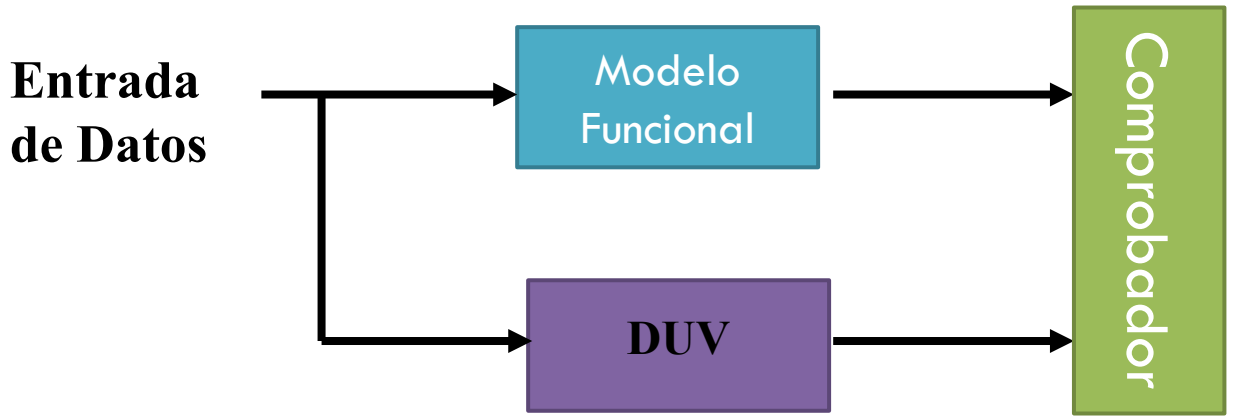
Verilog para Verificación

Comprobación de Respuestas. Diseños con Latencia Variable



Verificación Lógica con Verilog HDL

- **Testbench**
  - **Encapsulado de la Comprobación de Respuestas (Golden Models)**



Diversos tipos de modelos funcionales →

- Alto nivel (HDL, C, System C y otros).
- *Bus functional*.
- Modelos funcionales a nivel de ciclos.

Verificación Lógica con Verilog HDL

Verilog para Verificación

Comprobación de Resultados con Modelos Funcionales

```
task comprueba;
  begin
        fork:bloque_1
        join
        semaforo = 1'b0;
  end
endtask
```

```
begin
        #timeout;
        $display("Tiempo de espera agotado");
        $display($time);
        $stop;
        disable bloque_1;
end
```

Watchdog para el caso de emplear señal de validación

Latencia variable → Señal de validación de datos de salida.

Latencia Fija → Emplear directamente "#(delay)"

```
begin
        @(posedge senal_validacion);
        #Thold;
        if (salida_DUV ==! salida_modelo)
            begin
            $display ("Error de coincidencia con el modelo");
            $display($time);
            $stop;
            end
        if (salida_DUV == salida_modelo)
            begin
            disable bloque_1;
            end
        // Para el caso de 'x'
        $display ("Error de coincidencia con el modelo");
        $display($time);
        $stop;
        disable bloque_1;
end
```

Verificación Lógica con Verilog HDL

Verilog para Verificación

Comprobación de Respuestas. Aserciones

Mecanismos 'autónomos' de verificación en el propio código RTL

```
`define VERIFICACION                    *
// Activado el proceso de verificacion

module top (clk, rst, CYC_O, GNT_n);
        …

        `ifdef VERIFICACION

        ASER_UNIMASTER #(masters) VERIF_1(GNT_n);

        `endif
        ...

endmodule
```

** Uso de un TAG para instanciar el módulo de aserción de forma condicional*

```
module ASER_UNIMASTER(GNT_n);

parameter masters = 4;
input [(masters-1):0] GNT_n;
integer aux;
integer i;

always @( GNT_n )
        begin
        aux = 0;
        i = 0;
        for(i = 0; i < masters; i = i + 1)
                begin
                if (GNT_n[i] == 1) aux = aux+1;
                end
        if (aux > 1)
                begin
                $display("Error en asercion:
                        Más de un master toma el bus");
                $display($time);
                $stop;
                end
        end
endmodule
```

Verificación Lógica con Verilog HDL

Ejemplo Simple TESTBENCH

```
`timescale 1 ns/ 1 ps
module contador_param_vlg_tst();

localparam T = 10;

reg CLK; reg ENA; reg RST_A;
wire [3:0]  COUNT; wire TC;

contador_param #(.BITS(4),.cuenta(4'd14)) i1 (
                        .CLK(CLK),
                        .COUNT(COUNT),
                        .ENA(ENA),
                        .RST_A(RST_A),
                        .TC(TC)
);

initial
                        begin
                        RST_A=0;
                        CLK=0;
                        ENA=0;
                        $display("SIMULANDO!!!");

                         CASO_1(); // Caso de Verificacion 1
                         #(T*5)
                        CASO_2(); // Caso de Verificacion 2. RST_A con ENABLE
                         #(T*5)
                         CASO_3(); // Caso de Verificacion 3. ENABLE con TC

                        end
```

```
 task CASO_1;
     begin
              #T RST_A = 1'b1;
             @(negedge CLK) ENA  = 1'b1;
              #(T*20)
             @(negedge CLK) ENA  = 1'b0;
     end
endtask

task CASO_2;
       begin
              fork
                   RST_A     = 1'b0;
                   #(T*4) RST_A = 1'b1;
                   ENA      = 1'b1;
                   #(T*2) ENA   = 1'b0;
              join
        end
endtask

task CASO_3;
       begin
              ENA = 1'b1;
              #(T*14)
              @(negedge CLK)  ENA = 1'b0;
              #(T*5)
              @(negedge CLK)  ENA = 1'b1;
             end
endtask


always
              begin
              #(T/2) CLK <= ~CLK;
              end

endmodule
```

Verificación Lógica con Verilog HDL

Ejemplo Elaborado TESTBENCH.

```
////////////////////////////////////////////////////////////////
// FICHERO:          testbench.v
//
// Descripción: Proyecto de Arbitro de Bus Wishbone.
//       Testbench. Instanciación del MFB y Casos de Test
////////////////////////////////////////////////////////////////

`timescale 1ns / 100ps

module testbench;
…

MFB_HIGH T_MFB_HIGH(          .clk(clk),
                              .rst(rst),
                              .GNT_n(T_salida_DUV),
                              .CYC_O(CYC_O));

MODELO T_MODELO(              .CYC_O(CYC_O),
                              .GNT_n(T_salida_DUV_reg),
                              .GNT_n_next(T_GNT_n_next));

UTIL T_UTIL(                  .salida_DUV(T_salida_DUV),
                              .salida_modelo(T_GNT_n_next),
                              .senal_validacion(clk));

top T_TOP(                    .clk(clk),
                              .rst(rst),
                              .CYC_O(CYC_O),
                              .GNT_n(T_salida_DUV));
always @(posedge clk)
                    begin
                    T_salida_DUV_reg <= T_salida_DUV;
                    end

//Se registra la salida del DUV para la entrada al modelo
```

```
// BLOQUE DE CASOS DE TEST

reg aux_1;
reg [(masters-1):0] posiciones_fijas;

event casos_extremos;
event ventana_deslizante;
event fin_bloque;



always @(clk)
                    begin

                    ->casos_extremos;
                    @(fin_bloque);
                    $display("Verificación del DUV por casos extremos correcta");
                    $display($time);

                    ->ventana_deslizante;
                    @(fin_bloque);
                    $display("Verificación del DUV por ventana deslizante
correcta");
                    $display("Verificación del DUV correcta");
                    $display($time);
                    $stop;
                    end
```

Verificación Lógica con Verilog HDL


```
always @(casos_extremos)
begin
// Comienzan los casos extremos
        T_MFB_HIGH.reset;

        posiciones_fijas = 4'b1111;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;    // Nadie tiene el bus y todos lo piden
        // Caso de test 1

        posiciones_fijas = 4'b0000;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_MFB_HIGH.reset;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;  // Nadie tiene el bus (BUS_IDLE) y nadie pide
        // Caso de test 2

        posiciones_fijas = 4'b0110;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;
        posiciones_fijas = 4'b0000;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;   // Alguien tiene el bus y después nadie pide
        // Caso de test 3
        posiciones_fijas = 4'b0001;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;
        posiciones_fijas = 4'b1000;
        T_MFB_HIGH.peticion_bus(posiciones_fijas);
        T_UTIL.comprueba;    // Prueba la conexión en anillo
        // Caso de test 4

        -> fin_bloque;

end
```

```
always @(ventana_deslizante)
begin
        T_MFB_HIGH.reset;

        posiciones_fijas = 4'b1000;

        repeat (1000)
        begin
                if (posiciones_fijas == 4'b0001)
                posiciones_fijas = 4'b1000;
                else
                posiciones_fijas = posiciones_fijas >> 1;

                T_MFB_HIGH.peticion_bus(posiciones_fijas);
                T_UTIL.comprueba;

              T_MFB_HIGH.peticion_bus(T_UTIL.ALEA_GEN(posiciones_fija
        s));
                T_UTIL.comprueba;
        end

        -> fin_bloque;

end

endmodule
```

Verificación Lógica con Verilog HDL


```
// FICHERO: MFB_HIGH.v
// Autor: Vicente Herrero Bosch
// Descripción: Proyecto de Arbitro de Bus Wishbone.
// Comentarios: En este módulo se sitúan la generación de reloj
// el interfaz para reuso con otro tipo de buses (p.e.)
////////////////////////////////////////////////////////
`timescale 1ns / 100ps

module MFB_HIGH(clk, rst, GNT_n, CYC_O);
parameter periodo = 50, masters = 4, Thold = 1, Tsetup = 1;
parameter Td = 20;

input [(masters-1):0] GNT_n;
output reg rst, clk;
output [(masters-1):0] CYC_O;

MFB_LOW #(.periodo(periodo),.masters(masters),
          .Thold(Thold),.Tsetup(Tsetup),.Td(Td))
        H_MFB_LOW(.clk(clk),.CYC_O(CYC_O));

always begin #(periodo/2);
clk <= 0; #(periodo/2);
clk <= 1; end // Generador de reloj

task peticion_bus;
     input [(masters-1):0] vector_masters;
begin
     H_MFB_LOW.peticion_bus_low(vector_masters);
end endtask // Encapsulamiento para reuso
```

```
task reset; //Generador de reset
begin
disable peticion_bus; rst <= 1'b0;
wait (clk !== 1'bx);
@(posedge clk); rst <= 1'b1; @(posedge clk); @(posedge clk); rst <= 1'b0;
#(Thold);
if (GNT_n ==! 0)
     begin $display("Error en RESET"); $display($time);
     end //Asercion RESET
#(periodo-Thold-Tsetup);
end endtask endmodule
```

```
// FICHERO: MFB_LOW.v
// Autor: Vicente Herrero Bosch
// Descripción: Proyecto de Arbitro de Bus Wishbone.
// Testbench. Modelo Funcional de Bus. Modulo de Bajo Nivel
// Comentarios: Tareas de bajo nivel de acceso al bus.
module MFB_LOW( clk, CYC_O );
parameter periodo = 50, masters = 4, Thold = 1, Tsetup = 1, Td = 20;
input clk;
output reg [(masters-1):0] CYC_O;
task peticion_bus_low; // Tarea para simular peticiones de bus
input [(masters-1):0] vector_masters;
     real A; real B;
     begin
          A = ({$random} % 100); B = A / 100;
          @(posedge clk);
          #((periodo-Tsetup-Td)*B); // Temporizacion aleatoria
          CYC_O <= vector_masters;
          #(Thold)  semaforo = 1'b0;
     end
endtask endmodule
```

Verificación Lógica con Verilog HDL


```
// FICHERO: top.v
// Autor: Vicente Herrero Bosch
// Descripción: Proyecto de Arbitro de Bus Wishbone.
// Instanciación de las estaciones
`timescale 1ns / 100ps
`define VERIFICACION // Activado el proceso de verificacion

module top (clk, rst, CYC_O, GNT_n);
parameter masters = 4; // Numero de masters
input clk, rst;
input [(masters-1):0] CYC_O;
output [(masters-1):0] GNT_n;
wire FLAG_IDLE;
wire [(masters-1):0] ring_A;
wire [(masters-1):0] ring_B;
assign FLAG_IDLE = (GNT_n == 0 ) ? 1'b1 : 1'b0; // Condicion BUS IDLE

estacion M[(masters-1):0] (.clk(clk), .rst(rst), .CYC_O(CYC_O[(masters-1):0]),
                           .PRIO_n(ring_A[(masters-1):0]),
                           .PRIO_np1(ring_B[(masters-1):0]),
                           .GNT(GNT_n[(masters-1):0]));
// Instanciacion en array con indice de mayor a menor prioridad del master

assign ring_A[(masters-2):0] = ring_B[(masters-1):1];
assign ring_A[(masters-1)] = (FLAG_IDLE == 1'b1) ? 1'b1 : ring_B[0];


`ifdef VERIFICACION
            ASER_UNIMASTER #(masters) VERIF_1(GNT_n);
`endif // Asercion para asegurar un único master en el bus


endmodule
```

ASERCIÓN

```
// FICHERO: asercion.v
// Autor: Vicente Herrero Bosch
// Descripción: Proyecto de Arbitro de Bus Wishbone.
// Testbench. Modulo auxiliar de aserciones
// Comentarios: En este módulo se sitúan las aserciones incluibles
// en el código RTL
`timescale 1ns / 100ps
module ASER_UNIMASTER(GNT_n);
parameter masters = 4;
input [(masters-1):0] GNT_n;
integer aux,i;
always @( GNT_n )
      begin
            aux = 0; i = 0;
            for (i = 0; i < masters; i = i + 1)
      begin
            if (GNT_n[i] == 1) aux = aux+1;
      end
      if (aux > 1)
      begin
            $display("Error en asercion: Más de un master toma el
bus");
            $display($time); $stop;
end end endmodule
```