Lista de Features

- Ruteo de paquetes.
- 16 terminales externos conectados al borde de la malla con mapeo fijo.
- FIFOs de entrada/salida en cada router / puerto.
- Handshake de flujo:
  - Lado entrada: pndng_i_in[] / popin[].
  - Lado salida: pndng[] / pop[].
- Reset global del sistema .

TestPlan

Casos de uso común

1. Entrega básica de paquetes
   - Un solo terminal fuente activo, destino válido en el borde.
2. Lectura de ruteo para todos los terminales de borde
   - Para cada receptor 0–15, generar al menos un paquete.
3. Tráfico aleatorio en toda la malla
   - Activar todos los terminales con número aleatorio de paquetes y retardo aleatorio.
4. Múltiples fuentes hacia destinos variados
   - Varios terminales activos simultáneamente con destinos diferentes.
5. Reset
   - Aplicar reset al inicio.

Casos de esquina

6. Todos los orígenes hacia el mismo destino
   - 16 terminales enviando al mismo de borde.
   - Verificar que todas las fuentes logren entregar y que no se disparen aserciones de pndng no atendido.
7. Tráfico concentrado en esquinas
   - Generar muchos paquetes hacia coordenadas de borde

- Verificar que no haya bloqueo en routers de esquina y que el scoreboard siga pasando.

8. FIFO al límite

   - Generar secuencias que llenen la FIFO de uno o varios puertos casi hasta fifo_depth.
   - Verificar que pndng_i_in permanezca alto mientras no haya espacio y que popin sólo se active cuando se puede aceptar datos.
   - Comprobar que no haya overflow/underflow (no salen más paquetes de los que entraron).

9. Patrones de tráfico cruzado

   - Mitad de las fuentes enviando hacia un lado de la malla y la otra mitad hacia el lado opuesto.

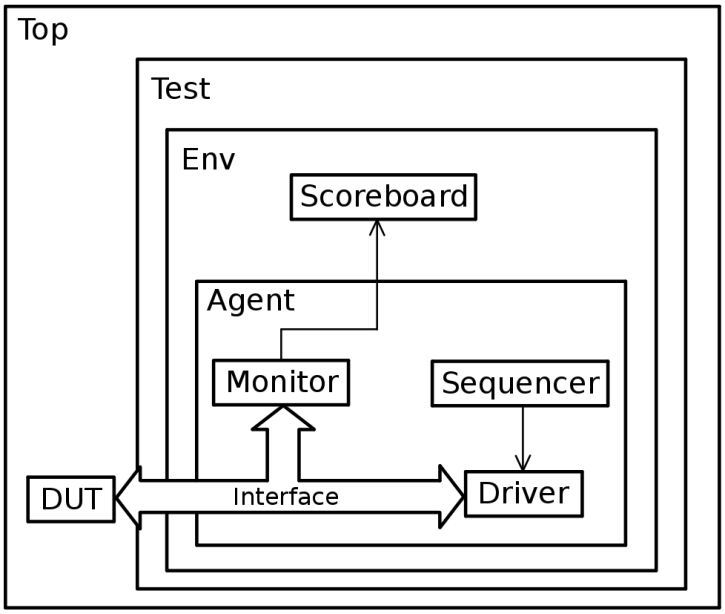
10. Reset en medio del tráfico

    - Iniciar tráfico aleatorio, dejar que se llene parte de la red y aplicar reset.

11. Modos de operación (mode)

    - Forzar mode=0 y mode=1 con el mismo patrón de destinos.
    - Verificar que la red entregue correctamente en ambos modos

Estructura del ambiente



Item de transacción

- target_row_out
- target_column_out
- mode
- payload
- next_jump
- packet
- sender
- receptor
- send_gap
- inject_error

Pruebas

Para las pruebas se tiene el ambiente montado en EDA Playground

https://www.edaplayground.com/x/bfF9

Donde originalmente se hace una prueba aleatoria para observar el correcto comportamiento

El primer sequence

```
class gen_sequence extends uvm_sequence;
  `uvm_object_utils(gen_sequence)
  function new(string name = "gen_sequence");
    super.new(name);
  endfunction

  rand int num_per_terminal[16];

  constraint c_counts {
    foreach (num_per_terminal[i]) num_per_terminal[i] inside {[3:10]};
  }

  virtual task body();
    int t;
    foreach (num_per_terminal[t]) begin
      for (int k = 0; k < num_per_terminal[t]; k++) begin
        trans_item m_item = trans_item::type_id::create($sformatf("m_item_%0d_%0d", t, k));
        start_item(m_item);
        assert(m_item.randomize() with { sender == t; });
        `uvm_info("SEQ", $sformatf("Generate item from terminal %0d with gap of %0d payload
=%0h ", t, m_item.send_gap, m_item.payload), UVM_LOW)
        finish_item(m_item);
      end
    end
  endtask
endclass
```

El cual simplemente envia ítems desde todas las terminales y una cantidad aleatoria desde cada una de estas, para asi probar todas las terminales con diferentes targets.

Luego se corre el caso de esquina

```
class corner_seq_same_dst extends uvm_sequence #(trans_item);
  `uvm_object_utils(corner_seq_same_dst)

  function new(string name = "corner_seq_same_dst");
    super.new(name);
  endfunction

  virtual task body();
    for (int k = 0; k < 50; k++) begin
      trans_item it = trans_item::type_id::create($sformatf("corner_it_%0d", k));
      start_item(it);
      assert(it.randomize() with {
        target_row_out    == 0;
        target_column_out == 1;
        send_gap          == 0;
      });
      `uvm_info("SEQ", $sformatf("Generate item from terminal %0d with gap of %0d payload =
%0h", it.sender, it.send_gap, it.payload), UVM_LOW)
      finish_item(it);
    end
  endtask
endclass
```

De todos los paquetes hacia una misma terminal desde diferentes terminales.

Además, también se corre

```
class all_src_same_dst_seq extends uvm_sequence #(trans_item);
  `uvm_object_utils(all_src_same_dst_seq)

  function new(string name = "all_src_same_dst_seq");
    super.new(name);
  endfunction

  virtual task body();
    for (int s = 0; s < 16; s++) begin
      for (int k = 0; k < 5; k++) begin
        trans_item it = trans_item::type_id::create($sformatf("it_%0d_%0d", s, k));
        start_item(it);
        assert(it.randomize() with {
          sender           == s;
          target_row_out    == 0;
          target_column_out == 1;
          send_gap          == 0;
        });
        `uvm_info("SEQ", $sformatf("Generate item from terminal %0d with gap of %0d payload =
%0h", s, it.send_gap, it.payload), UVM_LOW)
        finish_item(it);
      end
    end
  endtask
endclass
```

El cual es otra secuencia para el caso esquina de enviar diferentes paquetes desde un mismo source hasta un mismo destino, para asi probar el llenado de las fifos.

Además, se tienen 2 tests

```
run_test("trans_test");
//run_test("trans_test_reset_mid_traffic");
```

El primero hace un reset al inicio, y corre las 3 secuencias.

El segunda hace tanto un reset al inicio, inicia la secuencia principal y hace un reset en medio del tráfico.

También es posible correr el programa utilizando el servidor, clonando el repositorio

https://github.com/jcur02/Proyecto2_Verificacion.git

Y luego correr el siguiente comando

vcs -Mupdate -full64 -sverilog +incdir+$UVM_HOME/src $UVM_HOME/src/uvm_pkg.sv testbench.sv -o salida -debug_acc+all -l log_test +lint=TFIPC-L -cm line+tgl+cond+fsm+branch+assert +incdir+Archivos_proyecto_2

Esto generará el archivo de salida, el cual se puede ejecutar simplemente con

./salida +UVM_TIMEOUT=100000000ms +UVM_TESTNAME=trans_test -cm line+tgl+cond+fsm+branch+assert -cm_name cov_test -l test.log

El cual correra el ambiente con el primer test.

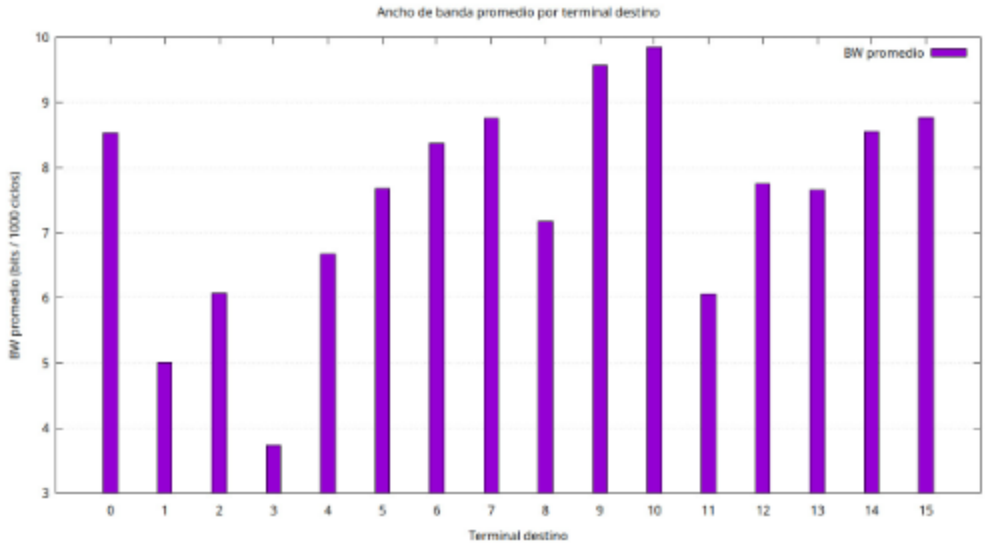
Para correr el ambiente con el segundo test simplemente se cambia el TESTNAME a 'trans_test_reset_mid_traffic'

Esto generará el archivo 'reporte_paquetes.csv'

Resultados los cuales se pueden graficar con el uso del comando

Gnuplot plot_ret.gnu

Con este se genera el gráfico para el retraso promedio en la entrega de paquetes x terminal, por ejemplo

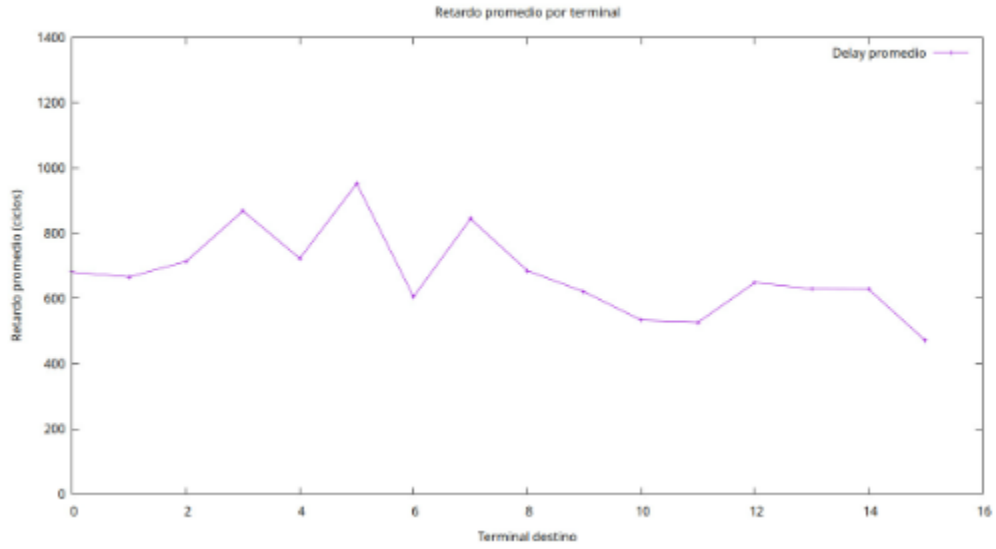


Y luego para generar el gráfico para el ancho de banda promedio, se utilizan los comandos

Gnuplot bw_por_terminal.gnu

Gnuplot plot_bw.gnu

Por ejemplo



Y para ver el coverage con
verdi -cov salida.vdb