Marked Graph Simulation

• Create a VHDL package/entities to support analysis of marked graphs

• Requirements
  – Must be able to simulate a marked graph of arbitrary topology
  – Report the cycle firing times of the marked graph
  – Support fixed or random delay times for nodes
  – Detect safety violations

• In your vhdl files, do not ever refer to your library name by anything other than ‘work’. This means that you can name your library anything you want, and it will not mess up my scripts.
What is a Marked Graph?

A **Marked Graph** (MG) is a restricted form of a Petri Net (PN). It is useful for modeling data movement within complex systems. A MG consists of **nodes**, **arcs**, and an **initial token marking** (the placement of tokens on nodes).
Firing Rule

A token represents input data.

A node fires when there are tokens on all input arcs; firing removes one token from each input arc and places one token on each output arc. An arc can hold more than token.

Before firing

After firing, the node consumes the tokens on its input arcs.
Liveness and Safety

A marked graph is **live** if there is always a node somewhere in the graph ready to fire.

A marked graph is **safe** if the token count of an arc never exceeds 1.

A dead system does not circulate data.

A node in an unsafe system can produce new data before its output data has been consumed by its successor nodes, resulting in loss of data.
Liveness and Safety Theorems

A MG is live, if every node is part of a directed circuit with token count > 0.

A MG is safe, if every node is part of a directed circuit with token count = 1.

live and safe

non-live and unsafe
Delays in Marked Graphs

In Marked Graphs, nodes are assumed to have no delay.

However, can assign delays to nodes to form timed Marked Graphs. We are interested in timed Marked Graphs, because real systems have delays.

Marked Graphs can be used to model asynchronous pipelines (asynchronous means that the system does not have a global clock).
Pipeline Example

Dotted lines are feedbacks or acknowledgements that indicate data has been received.

Nodes n1, n2 are registers in the pipeline, so tokens (initial data) placed on outputs. Other blocks are combinational blocks. Note this forms a live and safe marked graph.

Assume n1, n2 have delay = 5 units, all other blocks have delay = 10 units, what is average cycle time of this pipeline?
This is a valid arrangement for feedback. Might be advantageous as it reduces the number of feedback signals. However, does this affect the average cycle time of the pipeline?
Modeling Requirements

• Write a single entity/architecture to represent a node
  – Number of input is parameterized
  – No difference between normal inputs and feedback inputs; all are treated the same
  – Define a \textit{gdly} generic of type time; if this is non-zero then this specifies the firing delay of the node. If this delay is zero then node firing will either be a global constant, or a random value between some min/max bounds (more on this later).

• Write an entity that can be attached to a signal in the marked graph and display the delta times between token arrivals on the signal, as the average time of token arrivals
Modeling a Token

How do you represent a token on an arc?

Let each arc either be a ‘1’ or ‘0’ (arc phase). Let a state bit (node phase) in each node either be a ‘1’ or ‘0’.

An arc contains a token if the arc phase \(\equiv\) node phase.

All nodes have a reset line; all phases reset to either ‘0’ or ‘1’ at reset. Choice is arbitrary, but all must reset to same value.
Modeling Firing

A node fires if the phase bits of all inputs match the node phase. A node has two outputs, one whose phase always matches gate phase, and one whose phase is always opposite the node phase.

ready to fire

after firing

ready to fire

after firing
Initial Token Marking

Initial token marking is just a wiring choice

This token representation limits us to one token at a time on the arc. This is ok, anything $> 1$ is unsafe and we will only want to detect an unsafe condition, but not model it.
Other Modeling Requirements

Define a package called ‘mg’ that has at least the following variables in it:

```plaintext
shared variable min_delay: time := 10 ns;
shared variable max_delay: time := 100 ns;
shared variable gate_delay: time := 10 ns;
shared variable do_rand_delay: boolean := FALSE
```

The delay for a node will default to gate_delay if the gdly generic for the node is 0 ns, and do_rand_delay is false.

The delay for a node will be a random value between min_delay and max_delay if the gdly generic for the node is 0 ns, and do_rand_delay is true. For random delays, a new random delay is chosen **each time the gate fires**.
**rnd2 package**

The *rnd2* package contained in the ZIP archive contains a set of functions/procedures for generating random numbers.

The code below shows how to use this package to generate 10 random numbers between 0 and 999 inclusive.

```plaintext
process
    variable bound_h: Real := 999.0;
    variable bound_l: Real := 0.0;
    Variable rnd_rec: rnd_rec_t;
    variable ll: line;
begin
    rnd_rec.distribution := rnd_uniform_d;
    rnd_rec.seed := rnd_seeds(5); --random seed
    rnd_rec.bound_l := bound_l;
    rnd_rec.bound_h := bound_h;
    for i in 0 to 9 loop
        Rnd_Random(rnd_rec);
        write (ll, integer(rnd_rec.rnd));
        writeln(OUTPUT,ll);
    end loop;
wait;
```

Record type passed to *rnd_random* proc. Fill in the bounds before first call.

Init random seed using a value from *rnd_seeds* array; 50 seeds available

Generate random number, uniform distribution.
Detecting A Safety Violation

If the input to a node changes twice before the node fires, that is a safety violation. Your node mode must detect this, and print out the node.
Sample Output

<table>
<thead>
<tr>
<th>Cycle number</th>
<th>Avg (ns)</th>
<th>Delta (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>33</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>50</td>
</tr>
</tbody>
</table>

All nodes in a marked graph will have the same average cycle time.

After an initial transient after reset, the firing pattern for a node will reach a steady state.
Unsafe Condition Detection

# run 1 us;quit
# 1: Avg= 110 ns  Delta= 120 ns
# 180 ns: SAFETY Violation, on :tb_net2pipe_unsafe:bg1:a, input 0

Code that I used to write this string (‘a’ is an input signal to the node)

# run 1 us;quit
write(ll,now);
write(ll, string'(": SAFETY Violation, on "));
write(ll, a(i)'path_name');
write (ll,string'(", input "));
write (ll, i);
writeline(output,ll);
Test Cases:

For all test cases, shaded boxes have delay = 8 ns, others = 16 ns. Test cases for random delay specified later.
Test Cases (cont)

\[
\begin{align*}
\text{tb}_n & \quad n_1 \\
& \quad \quad \text{arrow} \quad n_2 \\
& \quad \quad \quad \text{arrow} \quad n_3 \\
& \quad \quad \quad \quad \text{arrow} \quad n_4 \\
& \quad \quad \quad \quad \quad \text{arrow} \quad n_5 \\
& \quad \quad \quad \quad \quad \quad \text{arrow} \quad n_6 \\
& \quad \quad \quad \quad \quad \quad \quad \text{arrow} \quad n_7
\end{align*}
\]
Test Cases (cont)

Note that n9 has a different delay from other non-shaded blocks.
Test Cases (cont)

n1 n2 n3
s1

n4 s4

n5

n6

n7

n8

s1

s2 s3

s4

s5

s6

s7

tb_n5

BR V0.1
Test Cases (cont)

n1 → n2 → n3 → n4 → s4 → n5 → n6 → n7

s1 → s2 → s3 → s4 → s5 → s6 → s7

n8 → n9

tb_n6
Test Cases

gdly = 200 ns, fixed. Other nodes have random delays from 10 ns to 100 ns max. This should create a safety violation.

tb_n7

---

tb_n8: network of tb_n2 with random node delays with min=10 ns, max = 100 ns for all nodes.

tb_n9: network of tb_n7 with random node delays with min = 10 ns, max = 100 ns for all nodes.

For random seed, use last digit of your student_id + 1.