YamlNet

Table of Contents

Purpose

Describe a network with one or more text files. These files can describe either hierarchical or parallel information (adding properties to an already defined network). The file format should be easy to read by any programming language and by humans.

Files can be read in any order and compiled into a single structure.

Network rules depends on its usage so only a few network rules are defined within YamlNet. User defined rules are easy to add.

Information like wires and ports are added by keywords. YamlNet have strict definitions of these keywords to avoid confusion.

Background

A network specifies how instances of blocks are connected by wires. Both instances and wires may have properties. The problem is that a common limitation of available network descriptions is that they only supports a certain kind of properties. For instance, if the network describe a system using several different abstraction levels of blocks, the result is that the hierarchy is being described in several places, often with additional overlapping information. Also when designing a system where several different tools are used, they all need their own network description format.

A simple example is when designing an ASIC. HDL files describe how wires and block instances are connected, sometimes twice because of synthesis and simulation use different block models. Performance simulation describe the network once more since this is usually not done with tools understanding HDL. Tcl describe timing constraints, another Tcl file describe control commands for the synthesis, and at least one set of files are used just to know where to find the source code. When all these files are located at different locations, written in different file formats by different people things go wrong.

If all this information could be gathered by a single top level file and sub level files can be located wherever best suited for the current organization, development will become both safer and faster. Since all information can be extracted from a single top level file it will also become easier to follow up on the progress. Do all blocks have both a C and a gate-level model defined? Do all ports have timing contraints? ...

Definition

The YamlNet main structure is a dictionary with four keywords (INCLUDE, HIERARCHY, WIRES and PORTS). INCLUDE is a list and the rest defines three sub dictionaries.

Any words starting with $ will be treated as variables and are expected to be defined either in the OS environment or as an argument to the YamlNet parser (the latter has the highest priority and will override any OS environment settings)

The parser will warn if any properties are redefined when processing several input files.
See network rules for more information.

INCLUDE – Parallel YamlNet files.

This is a list of additional YamlNet files that will be processed as if their text was part of this file.

Example:
INCLUDE:
- *topLevelTimingConstraint.yamlnet*
- *topLevelSynthesis.yamlnet*

HIERARCHY - The hierarchy data structure.

This dictionary define an instance to a hierarchic location and its properties.

Each key is a dot separated name where the dot ('.') shows the hierarchic level.
The corresponding value to a key is yet another dictionary. Each sub key here defines a property for an instance.

Currently only three properties/keywords are defined: *RTL*, *SG* and *SGARG*

- *RTL* specify a Register Transfer Model, a.k.a. a block description in HDL possible to synthesize to a gate level model. File endings (.vhd, .vhdl, v, sv, …) will tell what language used.
- *SG* (simGlue) specify an executable file expecting simGlue arguments.
- SGARG (simGlue arguments) specify additional arguments the the executable file.

Both *RTL* and *SG* will automatically detect if the specified file ends with .yamlnet which means a sub YamlNet file is used to describe the block. All nodes in a recursive YamlNet file will inherit its ancestor instance path.

Feel free to add more keys with additional information about your instances.

Example:
HIERARCHY:
    topLevel.stimuli:
        SG: $TB/sender.pl
        SGARG: --in_file ./muxTest.txt

mux1: *{ SG: $TB/mux.pl , RTL: $RTL/mux.v}*
sub1: *{ SG: $HOME/test.yamlnet }*
ram0: *{ SG: $TB/ram.pl }*
ram1: *{ SG: $TB/ram.pl }*
ram2: *{ SG: $TB/ram.pl }*
demux0: *{ SG: $TB/demux.pl }*

WIRES - The connections between instance ports. Wire perspective.

This dictionary define a wire to a hierarchic location and its properties.

Each key is a dot separated name where the dot ('.') shows the hierarchic level.
The corresponding value to a key is yet another dictionary. Each sub key here defines a property for a wire.

Currently only one property/keyword is defined: *PORTS*

- *PORTS* specify a list of instance ports the wire connects to.

Feel free to add more keys with additional information about your wires.

Example:
WIRES:
wire1: *{ PORTS: [topLevel.muxIn, dbg1, sub1.log] }*
subInst7.sladd2: *{ PORTS: [sub1.iirOut, iirIn2] }*
output: *{ PORTS: [inst47.dout] }*

PORTS - The connections between instance ports. Port perspective.

This dictionary define an instance port and its properties.

Each key is a dot separated name where the dot ('.') shows the hierarchic level. A port is by definition the last name after the last dot.
The corresponding value to a key is yet another dictionary. Each sub key here defines a property for a wire.

Currently only two properties/keywords are defined: WIRES and DIR

- *WIRES* specify a list of wires the port connects to.
- *DIR* specify the direction of a port

Feel free to add more keys with additional information about your wires.

Example:
PORTS:
subInstans2.inst1.port33: *{DIR: out, WIRES: [wire22, wire34]}*

Network rules

Different networks may need different rules. Some believe that two outputs to a single wire should be forbidden while others believe this is twice as good.

This is way within the supplied YamlNet parser only as few rules as possible are checked.

Adding additional checks with separate scripts is easy since the combining of several YamlNet files into one is already provided. So reading the resulting structure into a scripts to find out if your particular rule being violated should be fairly simple.

Anyway, some rules must exist and here is the list:

- A property being redefined will result in an error.

Complete example

This is a comment.

See XXX for more information about Yaml.

INCLUDE:

- *topLevelTimingConstraint.yamlnet*
- *topLevelSynthesis.yamlnet*

HIERARCHY:

topLevel.stimuli:
SG: $TB/sender.pl
SGARG: --in_file ./muxTest.txt

mux1: { SG: $TB/mux.pl , RTL: $RTL/mux.v}
sub1: { SG: $HOME/test.yamlnet }
ram0: { SG: $TB/ram.pl }
ram1: { SG: $TB/ram.pl }
ram2: { SG: $TB/ram.pl }
demux0: { SG: $TB/demux.pl }

WIRES:

wire1: { PORTS: [topLevel.muxIn, dbg1, sub1.log] }
subInst7.sladd2: { PORTS: [sub1.iirOut, iirIn2] }
output: { PORTS: [inst47.dout] }

PORTS:

subInstans2.inst1.port33: {DIR: out, WIRES: [wire22, wire34]}

TODO

- Present graphical output of a network.
- Override a property should be a severe error. Instead supply a way of manually allowing a redefinition. Also make sure the yaml parser notify about this?
- Check for tab since it is never allowed in yaml. (so we get nicer error message)
- Check for endless loops between yamlet files.
- Enabling the possibility to assign data to several keys with ?,* sounds like a good idea.
- MAKE A PROPER EXAMPLE AND ADD THE COMPLETE FILE AT THE END OF THIS DOCUMENT.