FSIC 2023 - TRISTAN GINGOLD

TOWARD MULTI-LANGUAGE HDL SIMULATION

WHY ? (1)

- Verilog and SystemVerilog are very dominent in ASICs
- VHDL is still very popular in Europe and in FPGAs
- Clearly most vendors push for SystemVerilog
    - But it’s already old-fashioned, they now push for HLS!
- Most designs contain IPs (sub-blocks designed by other teams)
    - Even on an FPGA (e.g. transceivers)
- (Obvious question -> obvious answer !)

WHY ? (2)

- Maybe Verilog/SystemVerilog people don’t need it
  - SV with UVM offers a powerful/good(?) verification framework
  - Many IPs are available
- Not exactly the same situation for VHDL users
- BUT:
  - mixing VHDL and Verilog has been possible for years
  - No compelling reasons to rewrite your own designs

GHDL

- Open Source VHDL simulator since ~2002
- Synthesis since ~2017
  - Can be used as a yosys plugin
  - Support PSL for formal verification
- But limited support for mixed language

WHAT DO USERS WANT ?

- The minimum:
  - Instantiate Verilog modules from VHDL
  - Instantiate VHDL entities from Verilog
- Probably more…

THE PROBLEMS ?

- Case sensitivity is different
- Type systems are different

- Existing tools already support mixed language designs
- But the rules are not described

CASE SENSITIVITY: IMPORT FROM VHDL

(EASIER CASE: IT'S SURJECTIVE)

VHDL

```
entity fifo1 is …
```

VERILOG

```
fifo1 inst1 (…);  // OK
Fifo1 inst2 (…); // Warning ?
FIFO1 inst3 (…); // Warning ?
```

Simplest solution:

- do case insensitive matching
- map extended identifiers with escaped identifiers (eg: \Name\ with \Name)

CASE SENSITIVITY: IMPORT FROM VERILOG

(COMPLEX CASE: IT'S INCOMPLETE)

VERILOG

```
module fifo1 …
module Fifo1 …
module FIFO1 …
```

VHDL

```
inst1: entity work.fifo1 …
```

Possible solutions:

- Do case insensitive matching
  - What if several modules have the same (case insensitive) name ?
- Only match lower case (or upper case verilog names)
  - Probably not very convenient, arbitrary choice
- Do case sensitive matching
  - Might be surprising to the users (as VHDL is case insensitive)
- Case insensitive matching if there is only one module, sensitive otherwise
  - Surprising effect if the user adds a new module

SIDE DISCUSSION

LET’S BE TECHNICAL

A FEW SLIDES ABOUT TYPE SYSTEMS

VERILOG TYPE SYSTEM

- Verilog types are based on the logic type
  - (Well, was retro-defined by SystemVerilog)
  - Together with the real type (less useful for synthesis)
  - Limited support of arrays
  - An integer can be seen as a packed logic array
- The logic type is builtin, you cannot define a similar type.
- Many implicit conversions (type, width, sign…)

VHDL TYPE SYSTEM (1)

- VHDL (Ada based) has a more complex type system
- No builtin types, any type can be defined by the user
- Some basic types are defined in the Standard package
  - In particular the boolean type (an enumerated type):
    - type boolean is (False, True);
- `(I like to speak about VHDL in my presentations…)`

VHDL TYPE SYSTEM (2)

- Any type can be declared by the user
  - enumeration, arrays, records (of course)
  - But also integer, real types:
    - type my_int is range 0 to 2**16 - 1;
- It is possible to do a design without using any standard types
  - Except maybe the boolean type
- Strong typing
  - Types integer and my_int cannot be freely mixed

VHDL TYPE SYSTEM (3)

- In practice nobody redeclare basic types, and for good reasons:
  - Limited use (at least in hardware)
  - You want interoperability
  - Synthesizers handle bit and std_logic specially
  - Users (and teachers) are often not aware of that
    - (For the same reasons)
- (Subtypes are a different matter, they just add bounds).

MOST USEFUL TYPE SYSTEM ?

- Implicit conversions in Verilog make the description less clear
    - a + b (what is the sign of the result ? How it is extended ?)
- VHDL requires many conversions
    - For width, sign
    - Those are cheap in hardware (or even transparent)
    - Could be heavy: std_logic_vector(unsigned(a) + x"12")
    - Some packages help to reduce the number of conversions

END OF SIDE DISCUSSION

SO, YOU WANT TO CONNECT VHDL AND VERILOG,
BUT THE TYPES ARE DIFFERENT!

MIXING VERILOG AND VHDL TYPES

General approach:

- Compatibility with existing tools
  - But the details are barely described
- Need to handle std_logic as an exception
  - The synthesis semantic is not a regular semantic
- Obvious:
  - Allow what makes sense
  - Reject what doesn’t make sense

MIXING SCALAR TYPE RULES

Verilog vectors can be associated with:

- VHDL vectors of bit, std_logic or boolean
- VHDL integers
- VHDL strings (very useful for parameters)
- VHDL physical types (at least for time, also useful for parameters)

Automatic width conversion

Note: as a consequence, you cannot associate literals ‘0’ or ‘1’ to verilog (ambiguity on the type, bit or std_logic)

- Annoying…
- But you could use a component to avoid ambiguity

MIXING OTHER SCALAR TYPE RULES

Verilog reals can be associated with VHDL reals

- Also very useful for parameters

Probably no mixing of enumerations

- (Except bit/boolean/std_logic)
- Semantic is too different
- Anyway, it concerns only SystemVerilog

MIXING COMPOSITE TYPE RULES

Associating records and structures should probably be allowed if they are compatible:

- elements/fields are compatible
- same names, same order

Likewise for arrays:

- Elements must be compatible
- Same dimensions ?

Note: VHDL has multidimensional arrays and arrays of arrays.

SystemVerilog interfaces with VHDL-2019 views ?

CURRENT IMPLEMENTATION

Currently it’s mainly work in progress (WIP)

Initial support of limited verilog synthesis.

Initial support of synthesis of verilog in VHDL.

To follow: synthesis of VHDL in verilog.

Future:

- SystemVerilog
- Mixed language simulation

REDUCING BOARDERS

Both VHDL and SystemVerilog support packages.

Is it possible to import a package declared in a different language ?

- Some tools allow this
- Probably doable for type/typedef declarations
- Probably doable for constant declarations
- Not sure how you can go further…

THANK YOU!

ANY FEEDBACK OR QUESTIONS ?