Synthesis with GHDL

FSiC 2022, Tristan Gingold

GHDL as a simulator

What is GHDL ?

- A VHDL simulator
- Command line tool

https://github.com/ghdl/ghdl

- Open source (GPLv2)
- Binaries for Linux (x86/x86-64), Windows, Mac

1 slide demo

hello.vhdl

```
entity hello is
end hello;

architecture behav of hello is
begin
  assert false
   report "Hello VHDL world" severity note;
end behav;
```

```
$ ghdl -a hello.vhdl    # analysis
$ ghdl -e hello         # elaboration
$ ghdl -r hello         # simulation
hello.vhdl:6:3:@0ms:(assertion note): Hello VHDL world
```

Main features

- Compiled (gcc, llvm or internal back-ends)

- Full support of 87, 93, and 02 standards
- 08 standard mostly supported
- PSL (Property Specification Language)

Some features

- vcd dump
- ghw waveform dump
  - Support all VHDL types
  - Can be read by gtkwave
- VPI interface
  - To support cocotb
- VHPIDIRECT
  - Call C functions from VHDL

What is GHDL?

- Works on virtually any design
  - Option '`-frelaxed`' to be compatible with bugs or deviations of commercial simulators
  - Rares issues concerning < 2008 standards

New: language server

- Analyze your file on every key stroke (fast enough)
- Very useful to improve error recovery
- Very valuable to navigate
- Written in python, using libghdl
- Many features could be added!
- VS Code and emacs extensions
- https://github.com/ghdl/ghdl-language-server

GHDL – Synthesis

Recent development: Synthesis

- Synthesis was the most requested feature
  - Not as github issues, but during informal talks
  - Pushed by open HW projects
- Missing block in FOSS EDA
- Support of PSL for formal proofs.

GHDL synth

- Standalone

  ```
  $ ghdl --synth FILES -e TOP
  ```

  - VHDL design to a simpler VHDL netlist
  - To check if your design could be synthesized
  - For regression tests
  - Can output VHDL or Verilog

GHDL synth – Yosys plugin

- As a Yosys plugin, provide a new command

  ```
  yosys> ghdl FILES -e TOP
  ```

  - Import a VHDL design as a netlist
  - Then the normal flow can be used
  - The most common usage!
- https://github.com/ghdl/ghdl-yosys-plugin

GHDL Synthesis

- It’s a front-end
  - Does not perform optimizations, done by ABC/ Yosys
- Can already handle large designs
  - Retro-uC (z80 + 6502)
  - Microwatt (PowerPc cpu)

Synthesis: improvements (1/2)

- Memory inference
  - Handle multi-port memories
  - Multi-clock
- Verilog interoperability in Yosys
  - Can instantiate a blackbox
  - Can instantiate a Verilog Module
    - (Need to deal with parameters)

Synthesis: improvements (2/2)

- VHDL 2008
  - Partial
  - Package with generics
- Improve error messages

What is synthesis ?

For research

- Netlist optimization, netlist mapping
  - That’s very important
  - But you need a netlist to start from…

- `ghdl --synth` is about netlist generation
  - No logic optimization

Netlist generation

- Partial evaluation
- Going functional
- Inference

Partial evaluation

- The size of all the objects must be known
  - Bus width
  - Register size
  - Memory size
- Evaluation is needed
  - at compile-time
  - during elaboration
  - during synthesis
- Partial evaluation is not a simple problem

Partial evaluation - examples

```
constant N : natural := 64;  --  Value is obvious
```

```
signal s2 : std_logic_vector (N - 1 downto 0);
```

```
variable V : natural;
…
V := to_integer(unsigned (Addr));
…
V := 0;
V := V + 1;
```

```
constant W : natural := log2 (N);
```

```
signal Addr : std_logic_vector (1 to W);
```

Partial evaluation – ghdl rules

- A signal is never constant
- A variable is constant when wholly assigned to a constant value
- A function is expected to return a constant value if called only with constant arguments.

- But another synthesizer may have different rules.

Going functional

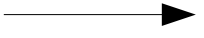
- Expressions can be naturally synthesized
  - Or, and, not, +, - …
- A synthesizer needs to remove the assignments
  - In particular the sequential assignments
- It’s like transforming your HDL into a function
- Use standard technics
  - VN (value numbering)
  - SSA (Static Single Assignment)
  - Loop unrolling

Going functional - examples

```
V := V0;
if Cond = '1' then
   V := V1;
end if;
```

```
V := Cond ? V1 : V0;
```

```
if Cond = '1' then
   V := V1;
else
   V := V0;
end if;
```



```
V := Cond ? V1 : V0;
```

```
for I in 1 to 5 loop
   Stmts;
end loop;
```

```
I := 1;
Stmts;
I := 2;
...
I := 5;
Stmts;
```

Inference

- A design is rarely fully functional
  - As a design is rarely only combinational
- There are storage elements
  - DFF, memories, latches…
- Internally they appear as violations of functional rules
  - And then a storage element is inferred
- Storage element: keep its previous value
  - A combinational loop

Inference - example

```
if rising_edge (clk) then
  S <= F (S, A)
end if;
```

```
S <= rising_edge (clk) ? F (S, A) : S;
```

Loop!

And edge detection

```
S <= DFF (clk, F (S, A));
```

DFF Inference

- Logical loop
- Path from the signal to itself
- Edge detection
  - Signal modified only on an edge
- (Latch if no edge)
- Possible async set/reset
- No special handling for sync set/reset
  - Normal logic

Memory Inference

- A little bit like DFF
- But the target is partially updated
  - At a non-static offset
  - Disjoint offsets
- Also partially read
  - Same criteria
- No reset
- Initial value (ROM or initialized RAM)

Memory Inference

- Multiple ports
  - With priority/order
- Multiple implementations
  - One process
  - Multiple processes (shared variable)
- Synch or Async read
- (Usually write is synchronous)

Memory Inference

- Different ports width
- Write enable
- Byte enable
- Single clock, dual/many clocks
- Content described by a record
- Memory within a record

Conclusion

Future

- Improve support of Verilog
  - Instantiation of Verilog modules in VHDL
  - Instantiation of VHDL entities in Verilog
- Improve memory inference
- Few ieee.numeric_std functions to implement
- What about ieee.numeric_bit
  - Unused ?

Backup: Extra Ideas

Work to be done...

- VHDL-SystemVerilog mixed simulators/synthesis
- IDE / Debuggers
- Project file format
- Schematic viewer
- Encrypted IPs/Cores/Blocks
- Coverage
- Constrained random verification
- AMS

VHDL + SystemVerilog simul (1)

- Many designs use both VHDL + (System)Verilog
  - At least for VHDL designs
- There are FOSS simulators for both languages
- But none that can handle both at the same time

VHDL + SystemVerilog simul (2)

- Handling just the synthesis part is simple
  - icarus could do it for simple designs
  - yosys could generate c/c++ simulation code
  - Adding GHDL front-end to Verilator should not be a lot of work
- But what about full language support ?

VHDL + SystemVerilog simul (3)

- Starting point ?
- Which FOSS SystemVerilog simulator ?
  - There are many!
  - None of them is complete
- (For VHDL, I have the answer!)

IDE + Debugger

- It would be nice to have an interactive debugger
  - Waveforms
  - Breakpoints
  - Forcing signals
  - Variable inspection
- It is possible to use gdb with GHDL, but it’s a difficult experience…

Project file format (1)

- A design has several files
- For synthesis or simulation, you need to provide the list of files
  - Plus general options, per file options...
- Each tool has its own project file format
- There are some project manager FOSS:
  - FuseSoC
  - hdlmake

Schematic viewer

- Current state of the art: graph + viewer
  - Graphviz
  - ELK/ELKjs
  - D3.js
  - …
- Only flatten designs ?
- Doesn’t scale well

Encrypted IPs

- Some FPGA vendors provide their IP simulation models through encrypted HDL.
- Use IEEE1735
- Support commercial simulators
- What about FOSS ?
- Is there a possible solution ?

Constrained Random Verification

- Strong argument from SystemVerilog
- Are there any FOSS implementation ?

AMS

- Analog Mixed Simulation
  - Like spice + logic + multi-domains
- Existing standards:
  - VHDL-AMS
  - Verilog-AMS
- Niche market ?