

Enhancing Digital Hardware Quality in LLM-Driven Design Flows through Strong Typing

Luca De Santis – Principal Design Engineer, Micron, Italy

The problem

- Industry-level digital hardware design tool-chains are based on Hardware Description Languages (HDLs, mainly Verilog and VHDL).
- These languages, introduced in the 80s, are showing weaknesses , the main one being no semantics of wires (everything can be connected everywhere).
- In the agentic AI era, this basic fact is enlightening a «paradox»: despite of a precise and formal description of design intent, the quality of final result is delegated to the ability of agents to infer an effective verification suite.

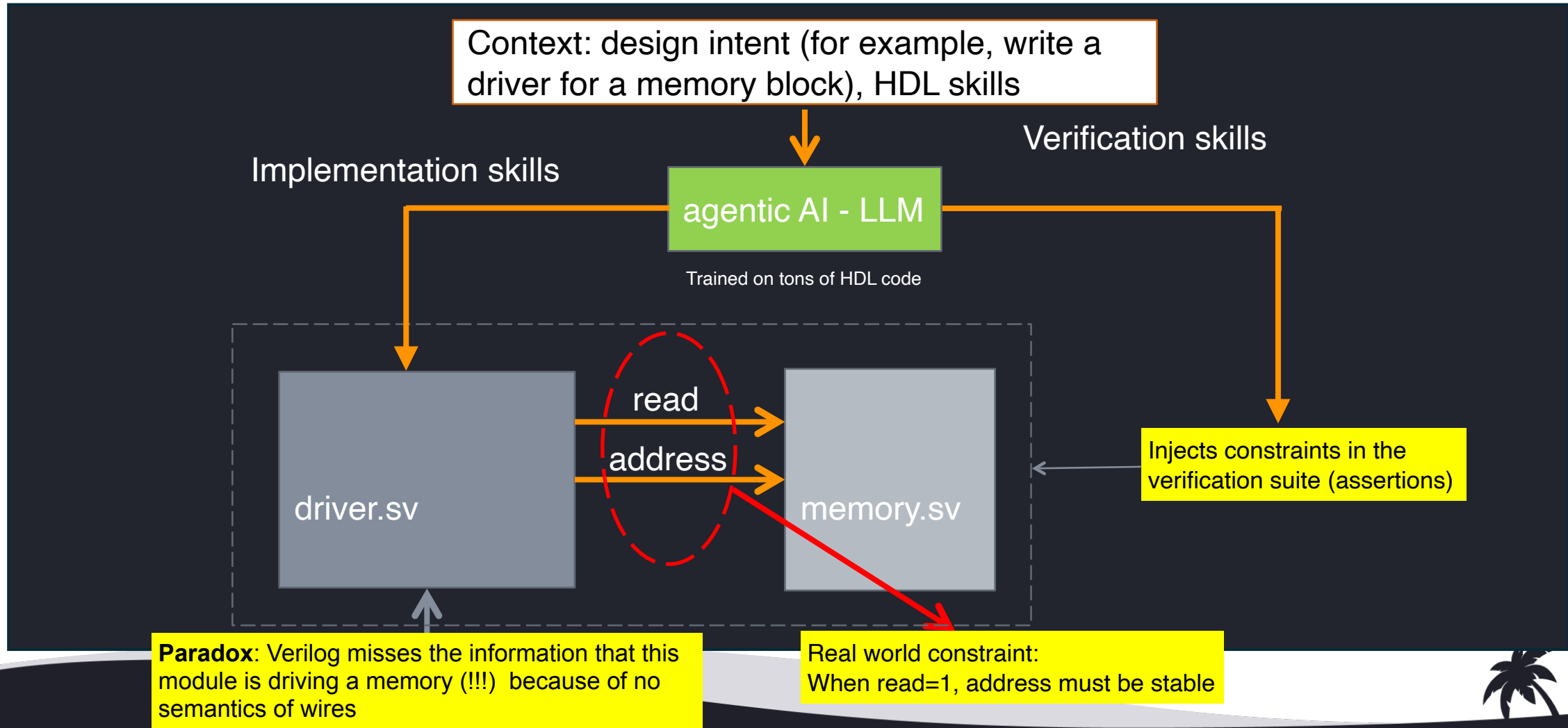


How a HDL works

- Design of digital systems starts from a high-level description through a HDL
- At a fundamental level, after compilation and synthesis , they are represented like a hierarchical connection of blocks
- Blocks are connected by uncommitted, untyped wires
- Typically the only check at the lowest level is the proper direction from an output port to one or more input ports
- All other structural checks are delegated to assertions on wires and/or specialised ex-post tools.



A paradox with agentic flow



Quality of artifacts

Quality of final result depends on :

1. Best-known-practices as learnt from training material
2. Ability of LLM to infer constraints
3. Ability of LLM to infer a proper verification suite
4. Ability/Experience of designer to catch corner cases and unusual sequences



What is typing

...in computer science

The main purpose of a type system in a programming language is to reduce possibilities for bugs in computer programs due to mismatches in how values are interpreted in different parts of a program. The aim is to prevent operations expecting a certain kind of value from being applied to values for which that operation does not make sense.

(source: Wikipedia)

...in HW design

A type system in a HDL prevents designers from constructing circuits where signals, modules, or state elements are combined in ways that violate their intended electrical or structural meaning. Its role is to eliminate design errors caused by mismatches in widths, protocols, timing assumptions, or interface contracts, ensuring that operations expecting a specific kind of signal or resource are never applied to hardware elements for which that operation is undefined or physically nonsensical.

(courtesy: Copilot)



The proposal

Author's proposal is to introduce a proper typing into the HDL (mainly on wires) to increase the quality of final artifact. The proposal follows the principle of :

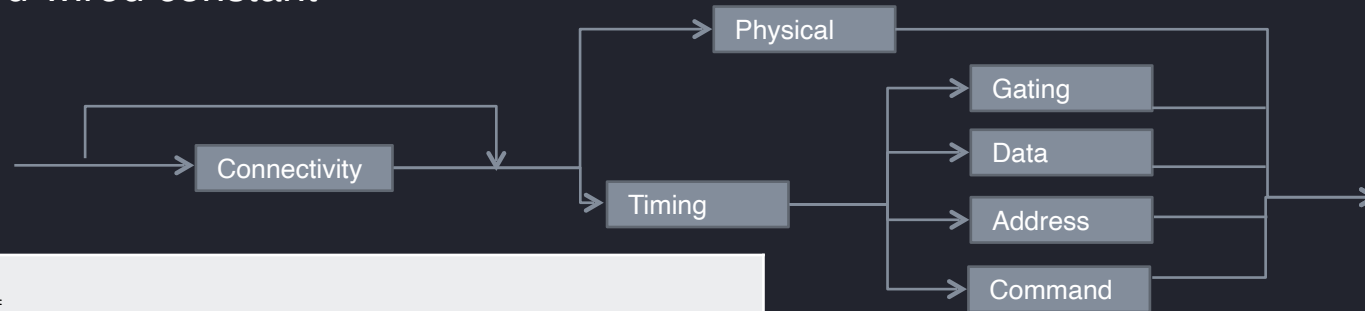
1. Forbid generic uncommitted wires
2. Categorize common functionalities associated to wires
3. Make clock and reset domains clear and segregated
4. Force designer to «cast» wire type on unconventional connections
5. Create a small set of building blocks with a strong semantics (for instance: memory modules, state machines ...)



Proposed Categorization

Different purposes and functionalities are associated to wires :

- Physical Event
- Power gating
- Usage
- Connectivity
- Timing
- Data values
- Hard-wired constant



```
Type :=  
Connectivity?  
(Physical  
| ( Timing? (Gating | Data | Address | Command )) )  
| ConstData ;
```

Category	Type
Physical event	clock
	derived(<clock>)
	reset
Power Gating	enabler
	gating
Usage	address
	command
Connectivity	connection
	input
	output
Timing	static
	synchr(<clock>)
	asynch(<clock>)
Data	boolean
	string [char_width, str_size]
	enum
	<i>numeric</i>
	unsigned [bit_size]
	signed [bit_size]
	float [m_size, e_size]
	fixed [int_size,dec_size]
	complex [m_size, e_size]

Enabling new possibilities

A richer semantics on HDL opens new possibilities :

- Automatic synchronizations
- Automatic power-gating
- Automatic pipelining
- Compile-time correctness checks on memory modules wiring (i.e. avoiding glitching logic)
- Enforce domains discipline through strong segregation
- Consistency checks on chains of processing logic on arbitrary numerical typed values
- Complex multi-clock architectures
- Clear identification of states and transitions (avoiding hidden and distributed states).



Example

The proposed approach is being used to describe an industry-level processor with Konyk-HDL language. A snapshot :

```
clock clk
reset rst_
```

Explicit definition of clock and reset
Domain segregation

```
// Machine Architecture
```

```
domain clk, rst_ {
  State  globState [3]  :: enum(IDLE , RUN)
  Reg    pc [CODE_ADDR_SIZE] :: address
  Instance dataram of mem [32,DATA_SIZE]
  Instance progmam of mem [CODE_WIDTH,CODE_SIZE]
  Instance rf    of bank [DW,REG_SIZE] { rch(2) }
}
```

only allowed state values

Define pc as an address;
Checks if logic generating it is in
the same clock domain ... **at compile time !!**

```
table exe_next_pc ( EXE_action , instrType -> forcepc) {
  _DO , JALtype      -> 1 , pc := res ;
  _DO , JALRtype     -> 1 , pc := res ;
  _DO , Btype , flag -> 1 , pc := nextpc ;
  any , any , do_IF  -> 0 , pc := pc + 4 ;
  any , any          -> 0 , pc := pc ;
}
```

Conclusion

In the agentic AI era, conventional HDLs show a semantic poorness that force designers to bet totally on the quality of training material and quality of verification suite.

A more robust flow can be implemented introducing a richer semantics on HDL definitions, especially on wires, avoiding generic ones and make clear clock and reset domains.

Next steps :

- Type inference and checking
- Integrate into an agentic flow
- Measure quality improvement





Thanks !!

<https://www.linkedin.com/in/luca-de-santis-42a87a5>