Boost your productivity in FPGA & ASIC design and verification

Bart Brosens
Application Engineering Lead

Sigasi;
• VHDL?
• Verilog?
• SystemVerilog?
• Mixed?
Focus

More productive
Focus on user

read
write
modify

Most EDA tools
Focus on chip

- SystemVerilog
- Verilog
- VHDL
- Mixed

simulation
synthesis
p&r
...
Focus

More productive
Focus on user

read
write
modify

Most EDA tools
Focus on chip

● SystemVerilog
● Verilog
● VHDL
● Mixed

→ simulation
→ synthesis
→ p&r
...

“Upfront Verification” : you design, we verify, let’s keep moving.
Topics

- Upfront Verification
- Project Exploration
- VS Code extension
- CI/CD
Topics

- Upfront Verification
  - Direct feedback
  - Language aware code checks
  - Intelligent content assist

- Project Exploration
- VS Code extension
- CI/CD
Upfront Verification

Direct feedback
User-focused - relevant
No need to switch context
Upfront Verification

Language aware code checks

- Code validation: Checking whether code is syntactically correct
- Code linting: Checking for problems in syntactically correct code
  - programming errors
  - bugs
  - stylistic errors (coding and naming style, indentation, header comments...)
  - suspicious or problematic constructs
Upfront Verification

Intelligent content assist

- Quick Fixes to resolve common language issues
- Help with a consistent formatting of your code
- Use templates for common declarations and statements
- Use the content of the project to suggest appropriate identifiers as autocompletion candidates
Upfront Verification

Demo
if (rst == '1') then
elsif rising_edge(clk) then

-- TDD "QuickFix"
-- On the line below, we typed a syntax error (data out is a port so the
-- assignment must be used). Note that the error marker in the gutter
-- has a small light bulb icon. Click on the light bulb and select
-- "Fix Assignment Operator".
-- Note how the error gets automatically fixed.

data out <= data in;
end if;
end process name;
end architecture RTL;
Topics

- Upfront Verification
- Project Exploration
  - HDL objects -> hyperlinks
  - Hovers show information
  - Multiple views for more insight
- VS Code extension
- CI/CD
Project Exploration

HDL objects -> hyperlinks

- Select Identifier then
  - Right-Click “Open Declaration”
  - F3

- Hyperlinks
  - Hold Ctrl and Click on Identifier
Project Exploration

Hovers show information

A tooltip shows information about a data object. No need to navigate away from the code.

- Object definition
- Object type
- Useful hyperlinks
Project Exploration - views

Multiple views for more insight

Views highlight particular aspects of the project

Some examples in the next slides
Project Exploration - views

Views help navigation

- View $\Rightarrow$ editor
  
  Double-click in view $\Rightarrow$ editor opens corresponding code

- Editor $\Rightarrow$ view

  Enable or disable link between view and editor
  
  If enabled: view highlights element being edited
Project Exploration - views

Outline View (single file)
- Libraries
- Entities
- Modules
- Architectures
- Ports & signals
- Assignments
- Instantiations
- Processes
Project Exploration - views

Hierarchy View (entire design)
- Similar to outline view
- Spans multiple files

Navigation:
- Hierarchy => editor
- Editor => hierarchy
Project Exploration - views

Dependencies between files in a project
Configurable scope & details
Navigate from the dependencies view to the editor
Project Exploration - views

- Overview of all libraries in a project
- List of design units in libraries
- Navigate between libraries view and editor
Project Exploration - views

List of errors and warnings in open projects in the workspace

- Navigate to editor
- Apply Quick Fix
Project Exploration - views

List of tasks in open projects in the workspace

- Configurable tags: TODO, FIXME, ...
- Navigate to editor
Project Exploration - views

SystemVerilog class hierarchy

Overview of
  ● (derived) classes
  ● Members

Navigate to editor
Project Exploration - views

Preprocessor View shows file in the editor, after preprocessing

- Include files are included
- Preprocessor macros expanded

Focus of editor and preprocessor view are linked
Project Exploration - views

Block diagram

- Navigation
- Documentation

```
begin
assert data_in /= data_out;
assert iterations <= MAX_COUNT;
data_out <= result;
valid <= counting and busy(3);
gen_subblock : for subcount in 0 to 7 generate
  a_block : entity work.my_block
  generic map(
    block_number => subcount
  )
  port map(
    clk => clk,
    rst => rst,
    d => start,
    q => busy(subcount)
  );
end generate;
COUNTER : process(clk, rst) is
variable state : state_t;
begin
if rst = '1' then
  state := idle;
```

```
Project Exploration - views

State machine
- Navigation
- Documentation

```verilog
module the_fsm(
    input clk,
    input rst,
    input energetic
);

typedef enum {sit, stand, walk, run} t_state;

always @(posedge clk) begin
    if (rst == 1)
        state = sit;
    else
        case (state)
            sit : state = stand; // get up
            stand : state = walk; // get going
            walk:
            if (energetic == 1)
                state = run;
            else
                state = sit; // halt!
            run:
            if (energetic == 1)
                state = walk; // easy!
            else
                state = sit;
            default : state = sit;
        endcase
end
endmodule
```
Project Exploration

Demo
```
begin
  "Open declaration"
  -- In the line below, place your cursor on the word 'dut' and press 'F3'.
  This takes you to the declaration of the entity 'dut'.

  dut inst : component dut
  generic map
    iterations => iterations
  }
  port map(    
    data out => data out,    
    valid => valid,    
    start => start,    
    clk => clk,    
    rst => rst,    
  );

  dut inst ent : entity work.dut
  generic map
    iterations => iterations
  }
  port map(    
    data out => data out,    
    data in => data in,    
    valid => valid,    
    start => start,    
    clk => clk,    
    rst => rst,    
  );

  assert valid = '0' or data out /= "000"
end architecture STR;
```
Topics

- Upfront Verification
- Project Exploration
- VS Code extension
  - Learning from software world
  - Language Server Protocol (LSP)
  - Available on marketplace
- CI/CD
VS Code extension

Learning from Software IDEs:

- The Software industry is much larger by number of engineers than the Hardware industry
- Wider market drives larger tool investments
- Adopting Proven Software Practices to Hardware needs is a common trend (i.e., version control, linting, coverage, unit testing)
- Software IDEs are currently a decade ahead
- Catching IDE trends from Software is a natural evolution path for the Hardware domain
Language Server Protocol (LSP)

Pre-LSP: Every Language x Every Editor

N x M

LSP: Editors and Language Servers Decouple

N + M

Suggested by Microsoft, adopted by many vendors

images: medium.datadriveninvestor.com
LSP: a Unified IDE-to-Server Protocol

**LSP Highlights:**

- LSP is a high-level message protocol defining typical operations/requests about documents, positions, ranges
- **IDE-agnostic & Language-agnostic**
- Covers IDE operations: hover, auto-complete, jump to definition, error checking, formatting, refactoring, folding, ...
- Language servers provide rich editing capabilities (not necessarily all of them)
- At low-level based on JSON-RPC communication, standardized message exchange structures and sequences

[Link to LSP website](https://langserver.org/)

---

code.visualstudio.com
VisualStudio Code: LSP-based tool example

**VisualStudio Code**
- Cross-platform browser-like app
- Free & Open Source
- Highly customizable with extra plugins
- Supports numerous languages (via LSP)
- Most editing capabilities of desktop IDE, but lightweight in nature
- Targets small projects / components, most popular for web-frontend parts

**Other references**
- Eclipse Theia
- Atom
- IntelliJ
- Sublime
- Vim + LSP
- Emacs + LSP
Sigasi Language Server

VS Code

SystemVerilog
Verilog
VHDL

Autocompletes
Errors
Navigation info
Rich hovers
Semantic highlighting

Analyzes your code while you type and gives instant context-aware feedback
Sigasi LSP: **VS Code extension**

www.sigasi.com/vscode/
VS Code extension

Demo
Sigasi extension for Visual Studio Code

An extension developed by Sigasi with rich support for VHDL and SystemVerilog, including features such as code navigation, project management, linting, code formatting, tooltips, refactoring and much more.

Getting started

- **Step 1. Set up your Sigasi license:**
  1. Open the command palette (Ctrl+Shift+P)
  2. Type Settings, select "Preferences: Open Settings (UI)" and confirm with enter
  3. Search for Sigasi and navigate to Path To License.
  4. Enter the path to your Sigasi license. This can either be a node locked license (path to license file) or a floating license (port=server).
Topics

- Upfront Verification
- Project Exploration
- VS Code extension
- CI/CD
  - Same rules as in Sigasi Studio (Eclipse and VS Code)
  - Align your team
  - Gatekeeper for the CI/CD flow
CI/CD

Learning from Software IDEs methodologies:

- The Software industry is much larger by number of engineers than the Hardware industry
- Wider market drives larger tool investments
- Adopting Proven Software Practices to Hardware needs is a common trend (i.e., version control, linting, coverage, unit testing)
- Software IDEs methodologies are currently a decade ahead
- Catching IDE trends from Software is a natural evolution path for the Hardware domain
CI/CD

Demo
Try it yourself

Explore on your own code what you’ve seen.
Get your free trial of Sigasi Studio XPRT on:

sigasi.com/try
Questions?

You can contact me on

bart.brosens@sigasi.com or support@sigasi.com