Synthesis

ZeBu-Server Training

*THE **FASTEST** VERIFICATION*

Agenda

- Overview
- Standalone synthesis
- Design entry
- RTL Front-End
- zFAST

Synthesis Overview

- Synthesis for ZeBu system can be achieved in 3 ways:
  - Standalone synthesizer
    - FPGA synthesizer
    - ASIC synthesizer
    - User is on his own
      - Must write synthesis scripts
      - Must produce EDIF netlists
  - RTL Front-End
    - Use a third-party FPGA synthesizer
    - Integrated in a graphical environment
      - No scripts to write
    - Synthesis job can be split and run in parallel
      - Manage large design
  - zFAST
    - ZeBu fast synthesizer
      - Very fast but not best netlist optimization
    - Integrated in a graphical environment
      - No scripts to write
    - Synthesis job can be split and run in parallel
      - Manage large and very large design
      - Supports emulation-specific features

Agenda

- Overview
- Standalone synthesis
- Design entry
- RTL Front-End
- zFAST

Standalone synthesizer

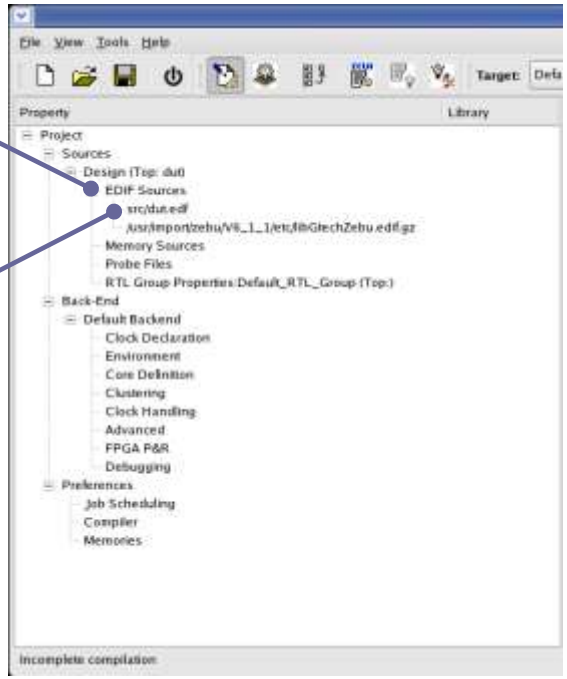
- Use a third-party synthesizer
- User is on his own
  - Must write synthesis scripts
    - Contact your local support to get examples
  - Must produce EDIF netlists
- Can use (preferred) an FPGA synthesizer
  - Target Xilinx Virtex5 library
  - Must configure the synthesizer to omit pad I/O cells
- Can use (not recommended) an ASIC synthesizer
  - Target GTECH library
  - Will not produce optimal netlists

Standalone synthesizer

- Add the synthesized EDIF netlists to the zCui project
- If using the GTECH library, add the `$ZEBU_ROOT/etc/libGtechZebu.edif.gz` EDIF library to the project

Right-click on "EDIF Sources" to add EDIF files

Right-click on EDIF design file and select "Extract Toplevel" to get design top-level name automatically



Agenda

- Overview
- Standalone synthesis
- Design entry
- RTL Front-End
- zFAST

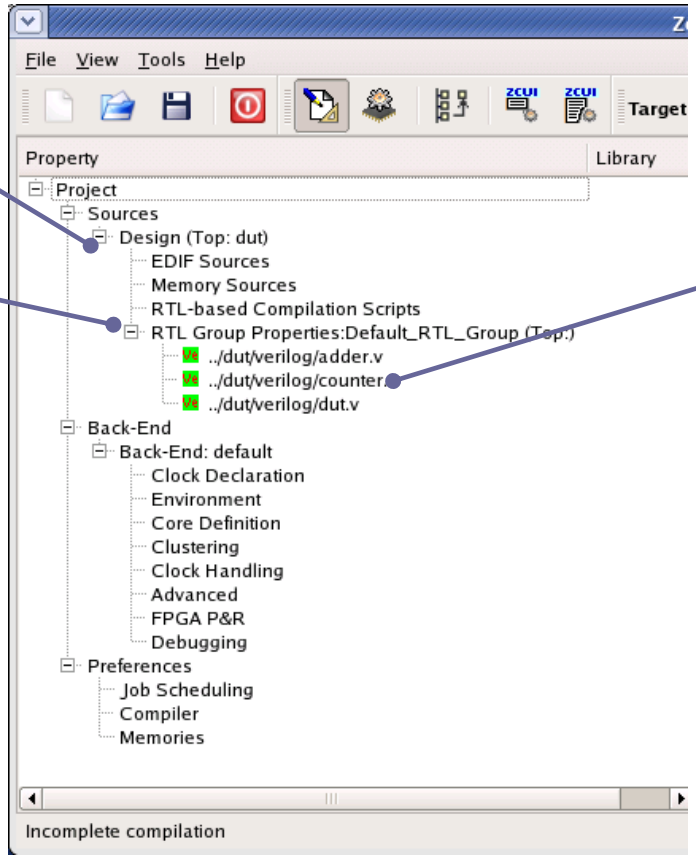
Design entry

- RTL source files must be listed in `zCui`
- Source language is selected
- A compilation library may be defined for each file
- RTL source files may be grouped in RTL groups
  - At least one group exists: `Default_RTL_Group`
  - Files in different groups will be synthesized in different synthesis tasks
  - Each group can have different synthesis parameters

Design entry
Adding source files

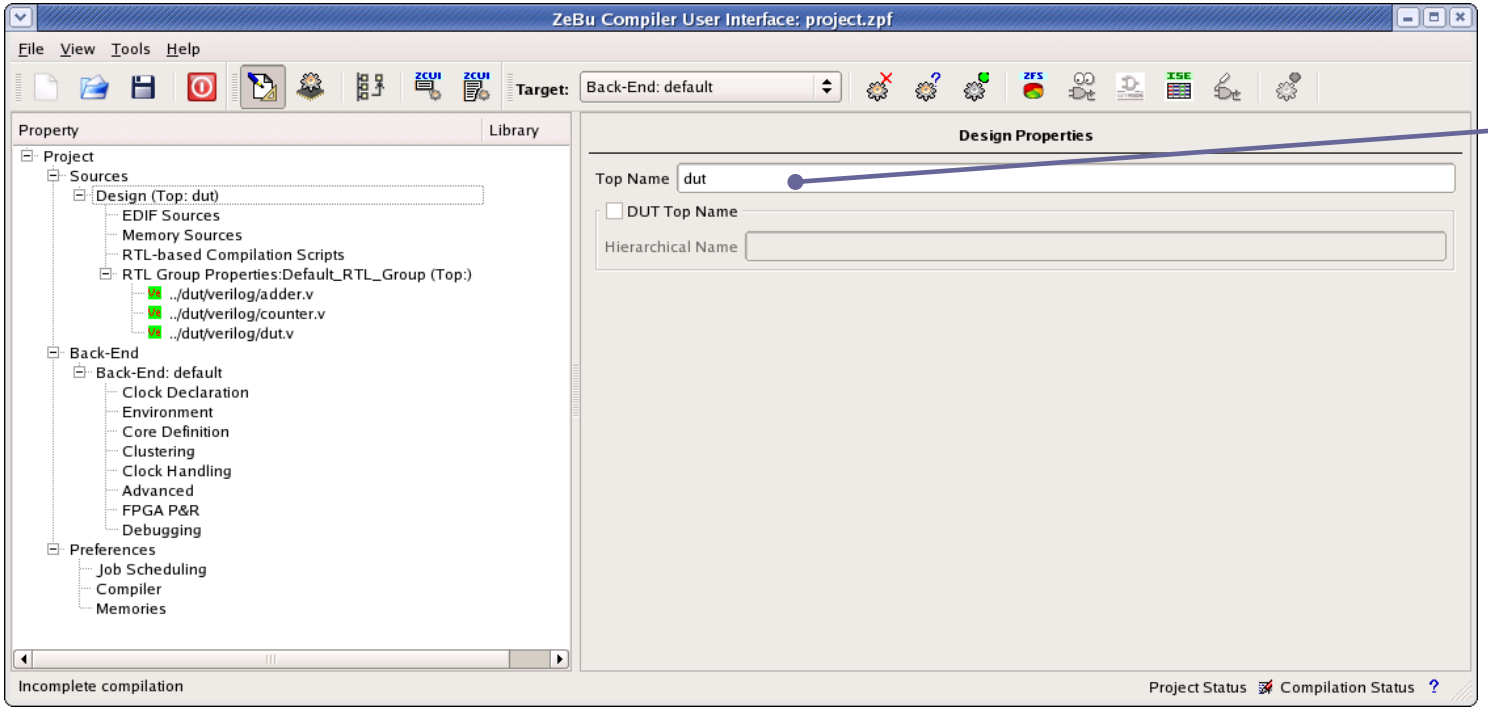
Right-click on "Design" to add RTL groups

Right-click on "RTL Group" to add RTL source files



Right-click on RTL source file to define compilation library

Design entry
Specify top

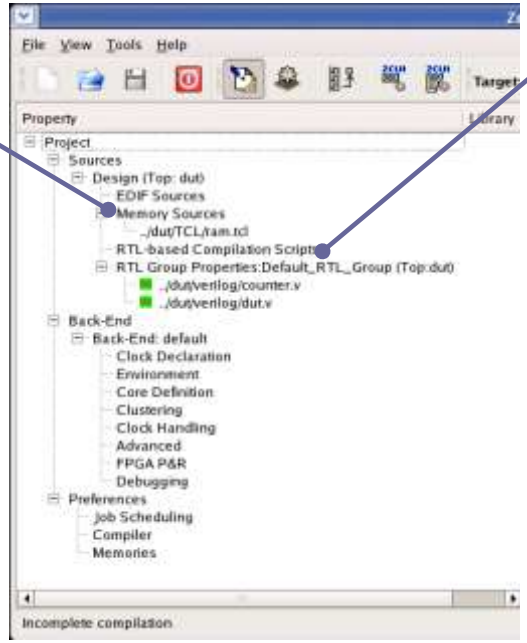


Add design top name in the design properties panel

Design entry
Memories and probes specifications

- **Possibly add memories and probes specifications**
  - **Memory commands file format will be introduced in different training modules**
  - **For synthesis, defining a probe will force it to keep the RTL signal name and not optimize it**

Right-click on "Memory Sources" to add memories description files



Right-click on "RTL-based Compilation Scripts" to add probes description files

Design entry
Probe file specifications

- List of probes must be declared in a TCL file added to the project under the “RTL-based Compilation Scripts” property in `zCui`
- The command name is `probe_signals`. Its syntax is somehow complex, use the on-line help for reference:
  `zTopBuild -help probe_signals`

Design entry
Probe file specifications

- `probe_signals -instance top.design -port foo* -select is_output -fnmatch`
  Inserts dynamic probes on all output ports whose name starts with "`foo`" of cores under hierarchy "`top.design`"
- `probe_signals -port top.hier0.hier1.and0.o`
  Inserts a dynamic probe on port "`top.hier0.hier1.and0.o`"
- `probe_signals -port top.hier0.hier1.and0.o -type static`
  Inserts a static probe on port "`top.hier0.hier1.and0.o`"
- `probe_signals -port_file ports.prb -type flexible -group A`
  Inserts flexible probes on ports given in file "`ports.prb`". Only one port per line is allowed. The declared ports will belong to group "A"
- `probe_signals -wire top.hier0.hier1.wand_out`
  Inserts a dynamic probe on wire "`top.hier0.hier1.wand_out`"

Design entry
Simulation command replacement

- It is possible to build the RTL source part of a `zCui` project automatically
- If you have a simulation script, you can replace the RTL source code analysis and elaboration commands by ZeBu simulation command replacement:
  - `zRFEvlog`: analyze Verilog code
  - `zRFEvhdl`: analyze VHDL code
  - `zRFEelab`: elaborate the design
- Supports common simulator options such as: `-y`, `-f`, `+define+macro`, ...

Design entry
Simulation command replacement

- **Example:**

```
zRFEvlog src/verilog/adder.v src/verilog/counter.v src/verilog/dut.v
zRFEelab dut
```

- **This will create a `zfc.work` directory in which you will find:**
  - **A binary database**
  - **Elaborated version of the RTL source code**
  - **Two files:**
    - **`top_user.zpf`**
      - **A `zCui` project file pointing to the user source code**
    - **`top_elab.zpf`**
      - **A `zCui` project file pointing to the elaborated source code**
    - **These files can be used as starting point to build a full `zCui` project**

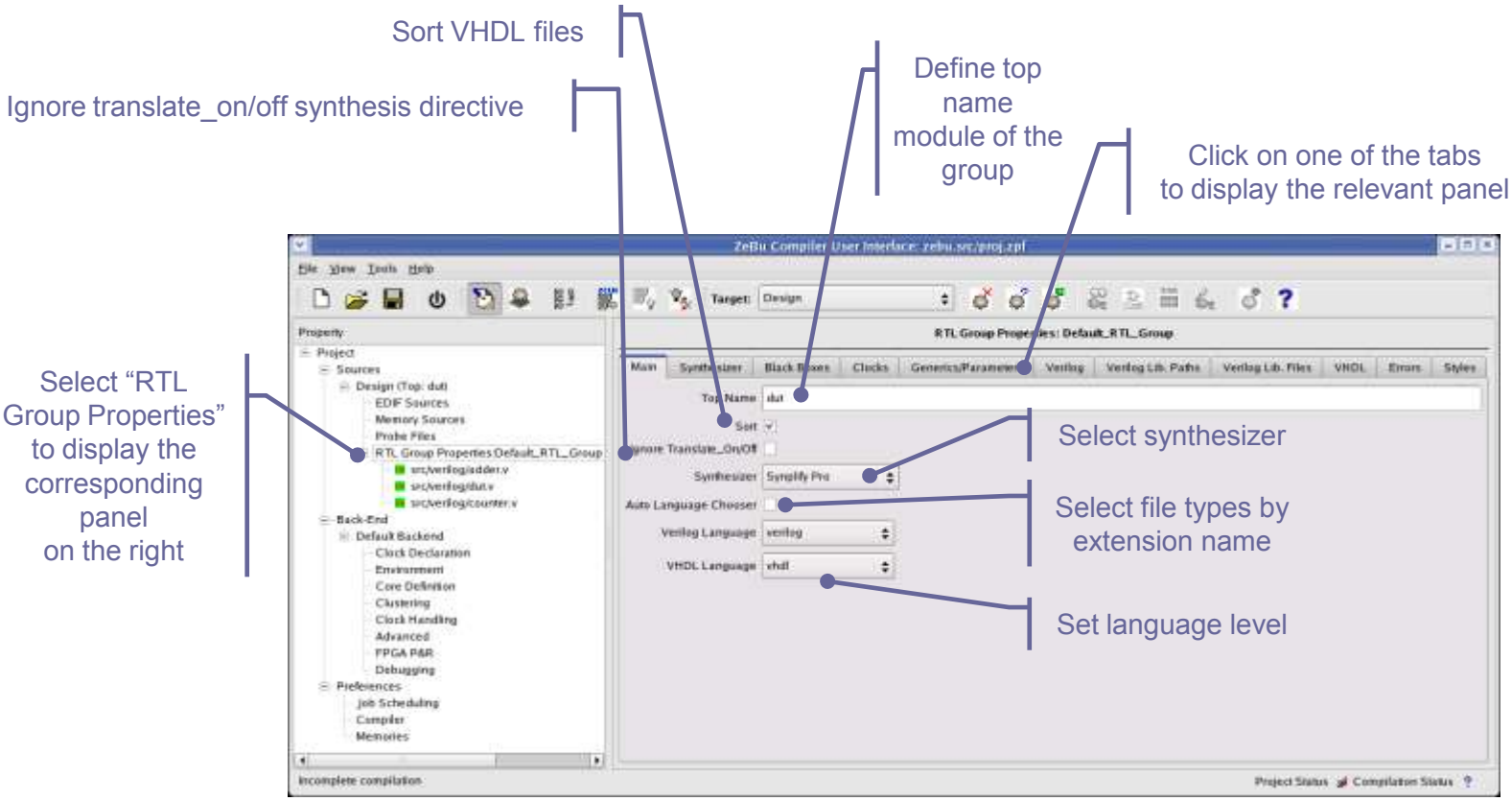
Agenda

- Overview
- Standalone synthesis
- Design entry
- RTL Front-End
- zFAST

RTL Front-End Overview

- RTL Front-End is a synthesis environment
- It automatically:
  - Elaborates the design
  - Writes synthesis scripts
  - Splits the synthesis in multiple tasks
  - Launches synthesis tasks in parallel
- It is integrated in `zCui`
- The following 3rd party synthesizers are supported:
  - Precision
  - Synplify, Synplify Pro, Synplify Premier, Synplify Premier DP
  - XST
- It consumes one synthesizer license per parallel task
- Synthesis licenses must be purchased from their respective vendors

RTL Front-End
Main panel

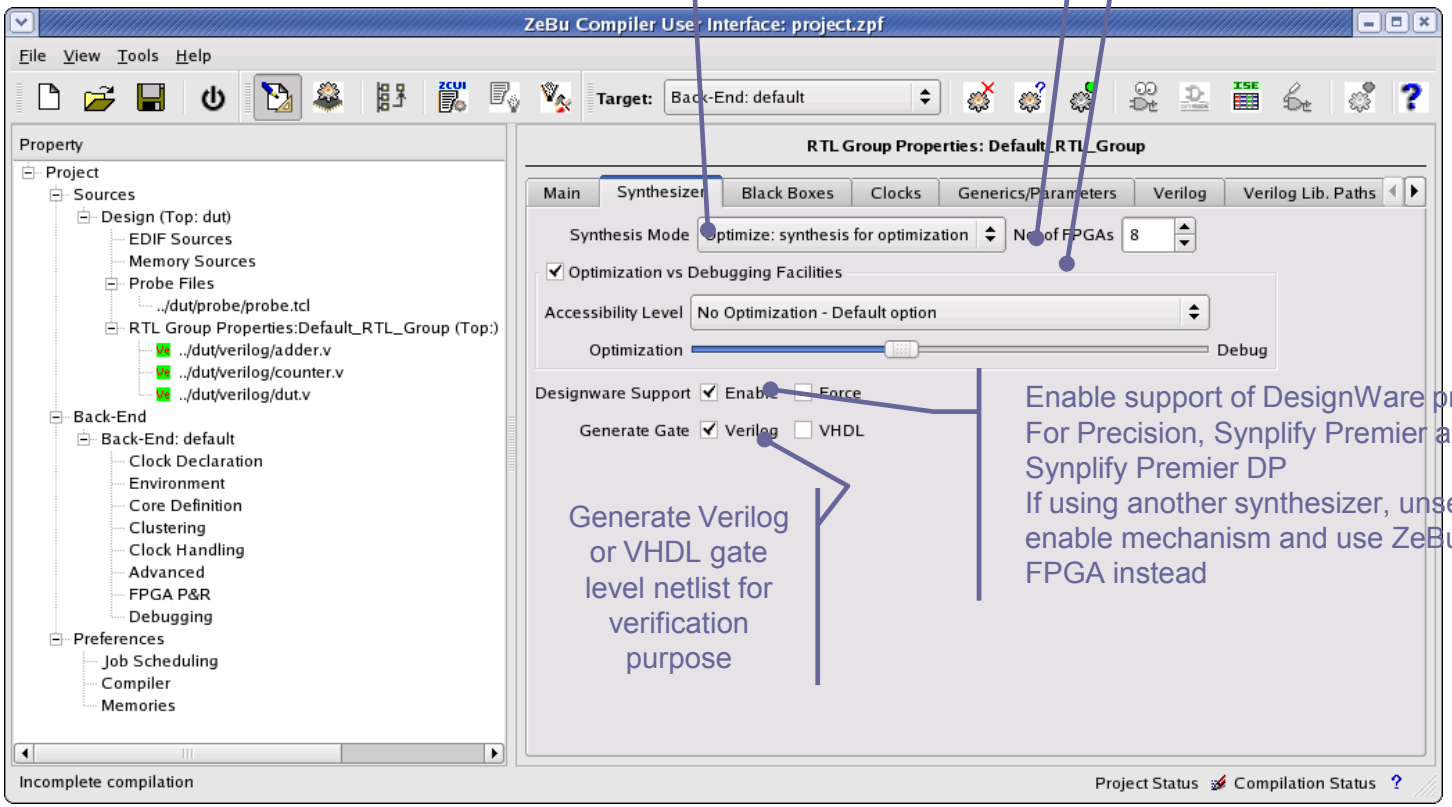


RTL Front-End Synthesizer panel

Select synthesis mode:
- Top-Down: full design in one single path
- Optimize: synthesize for optimization
- Block-based: synthesize each block separately (debug)
- Quick: accelerate synthesis

Set number of "FPGAs"

Set the accessibility level:
- Optimize for area
- Optimize for speed
- No optimization
- Keep all registers
- Keep all registers and insert dynamic probes for combinational signals

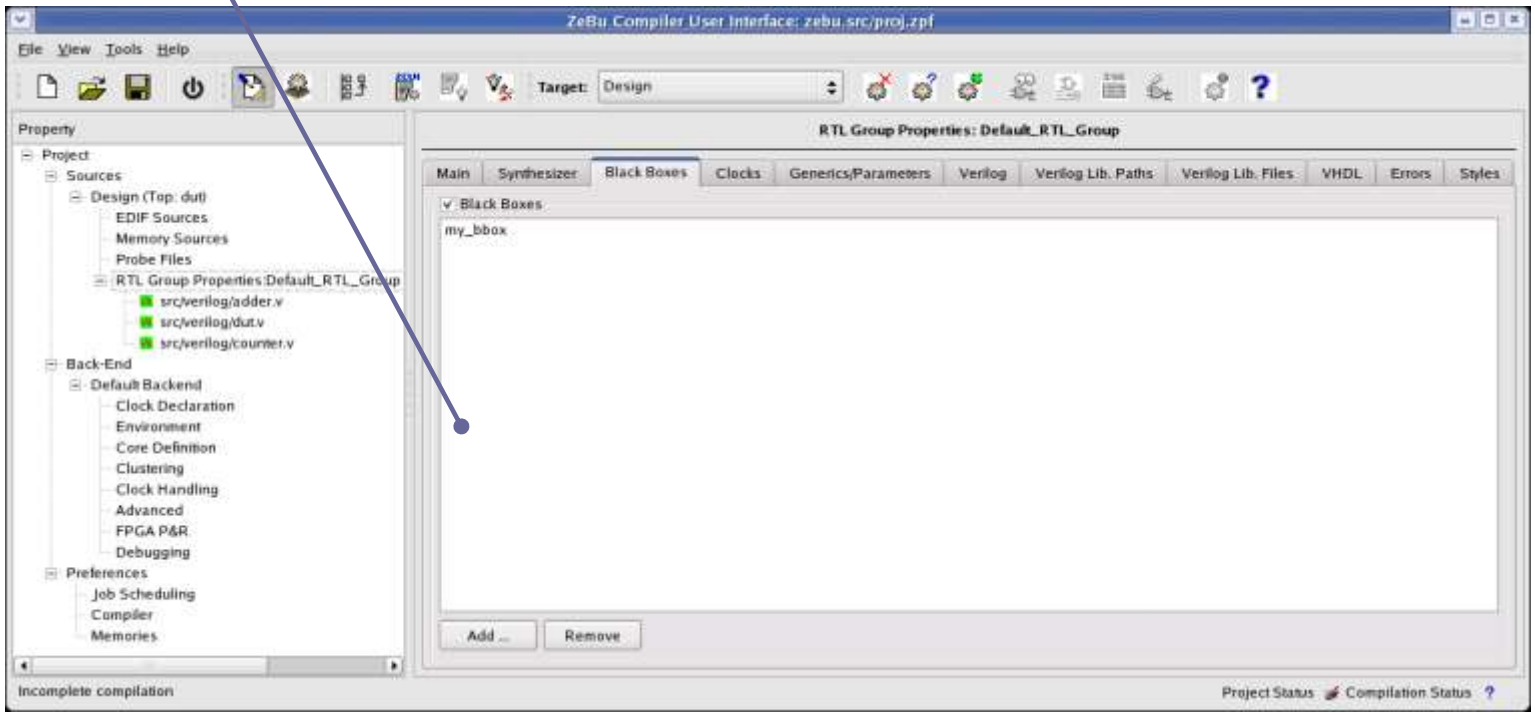


Enable support of DesignWare primitives
For Precision, Synplify Premier and Synplify Premier DP
If using another synthesizer, unset the enable mechanism and use ZeBuWare FPGA instead

Generate Verilog or VHDL gate level netlist for verification purpose

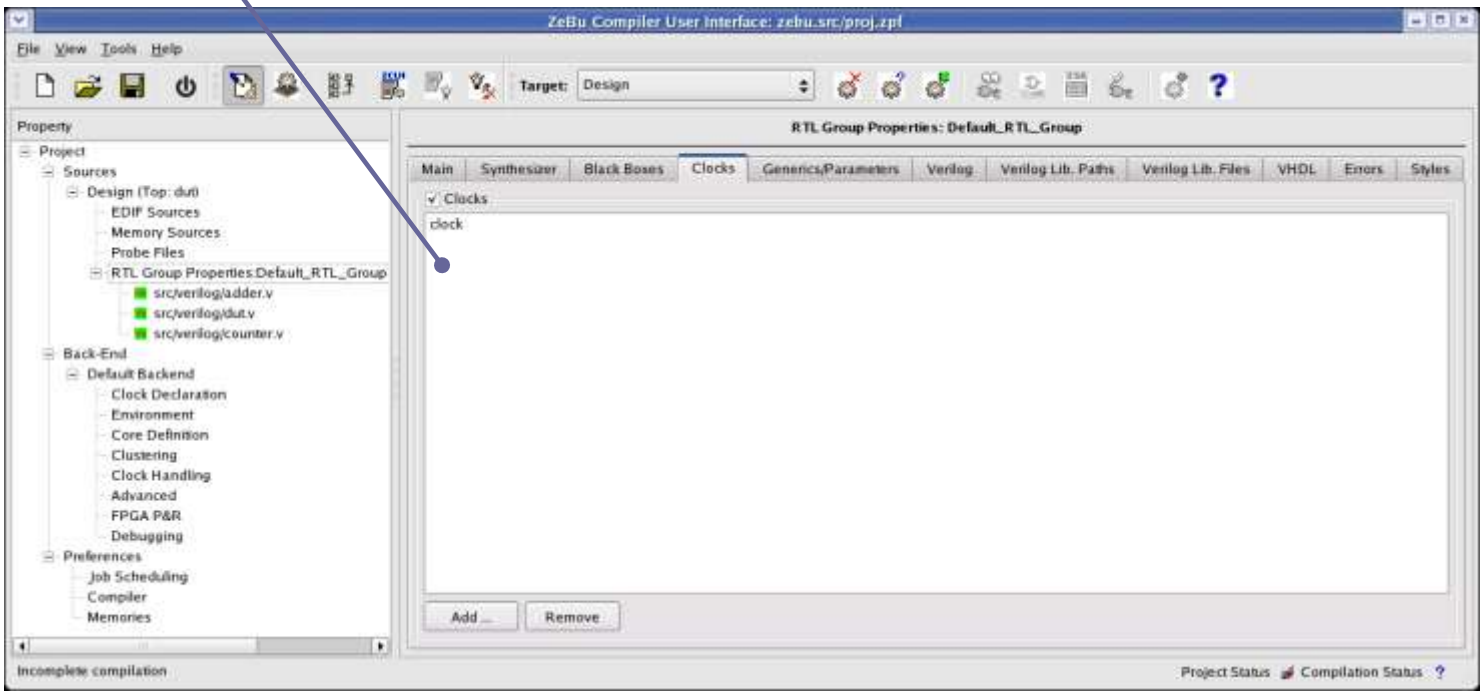
RTL Front-End
Black Boxes panel

Add black boxes if any, module name



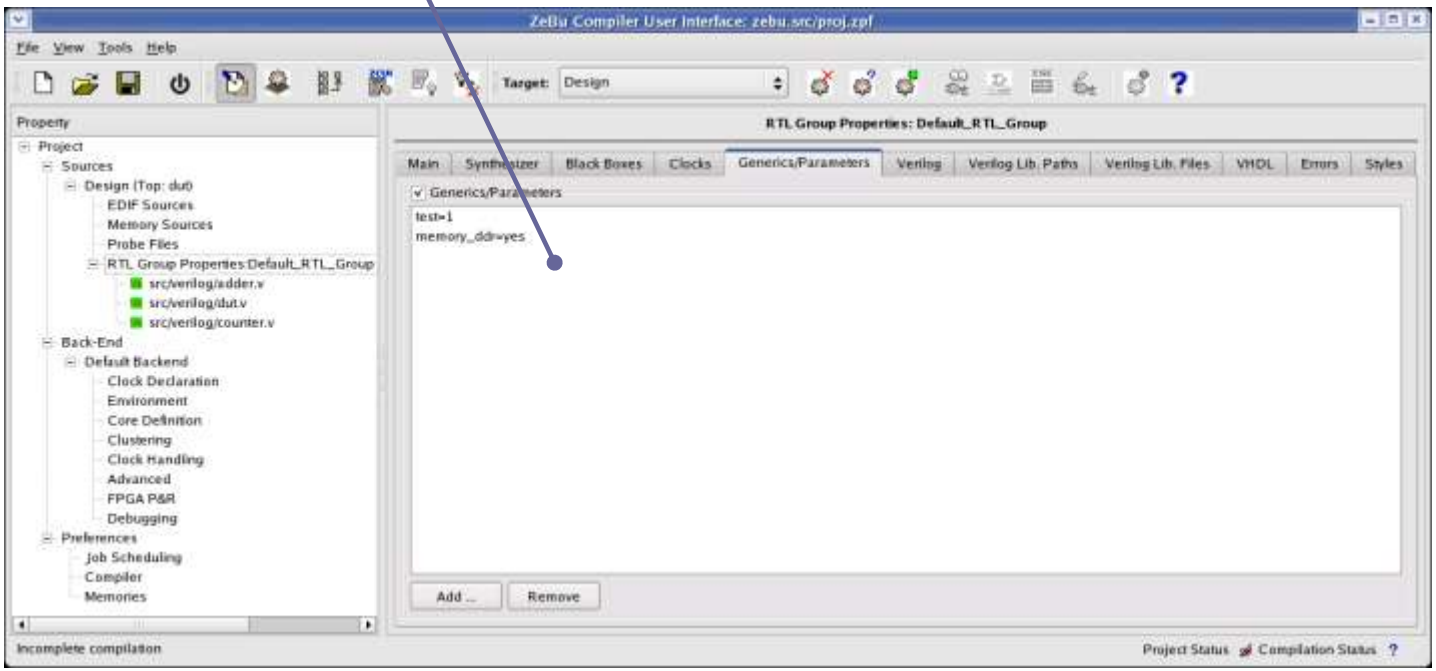
RTL Front-End
Clocks panel

Add clock names



RTL Front-End
Generics/Parameters panel

Add top level generics for VHDL, parameters for Verilog

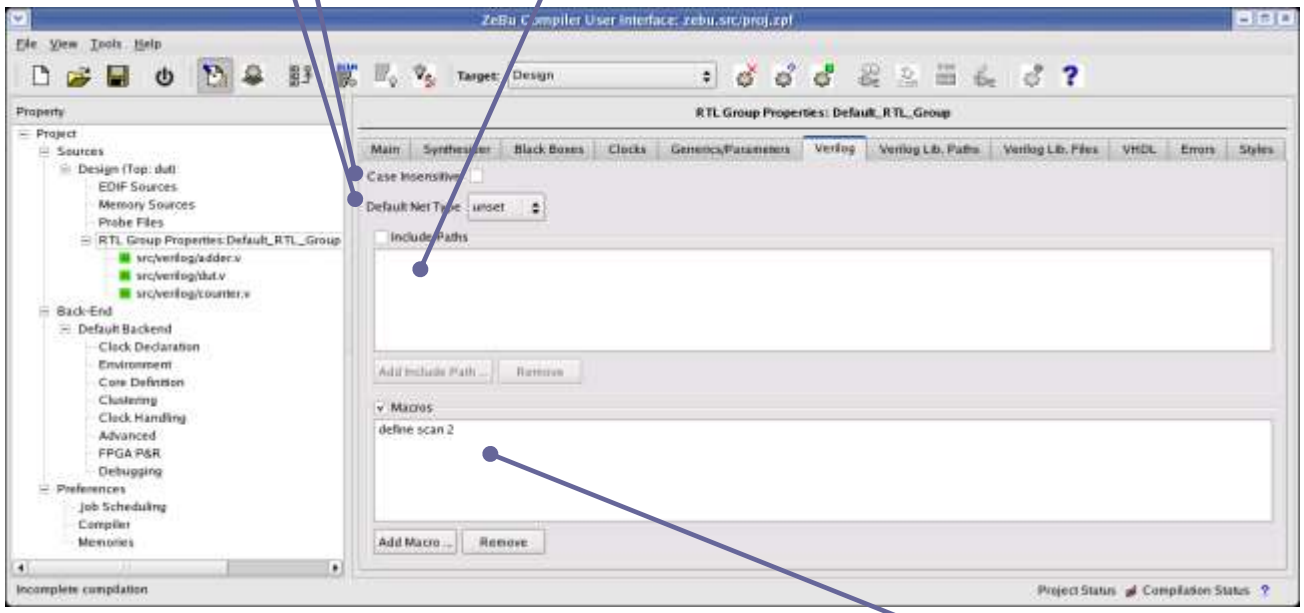


RTL Front-End
Verilog panel

Set Verilog synthesis case insensitive, does not respect standard!

Set Verilog default net type for undefined wires

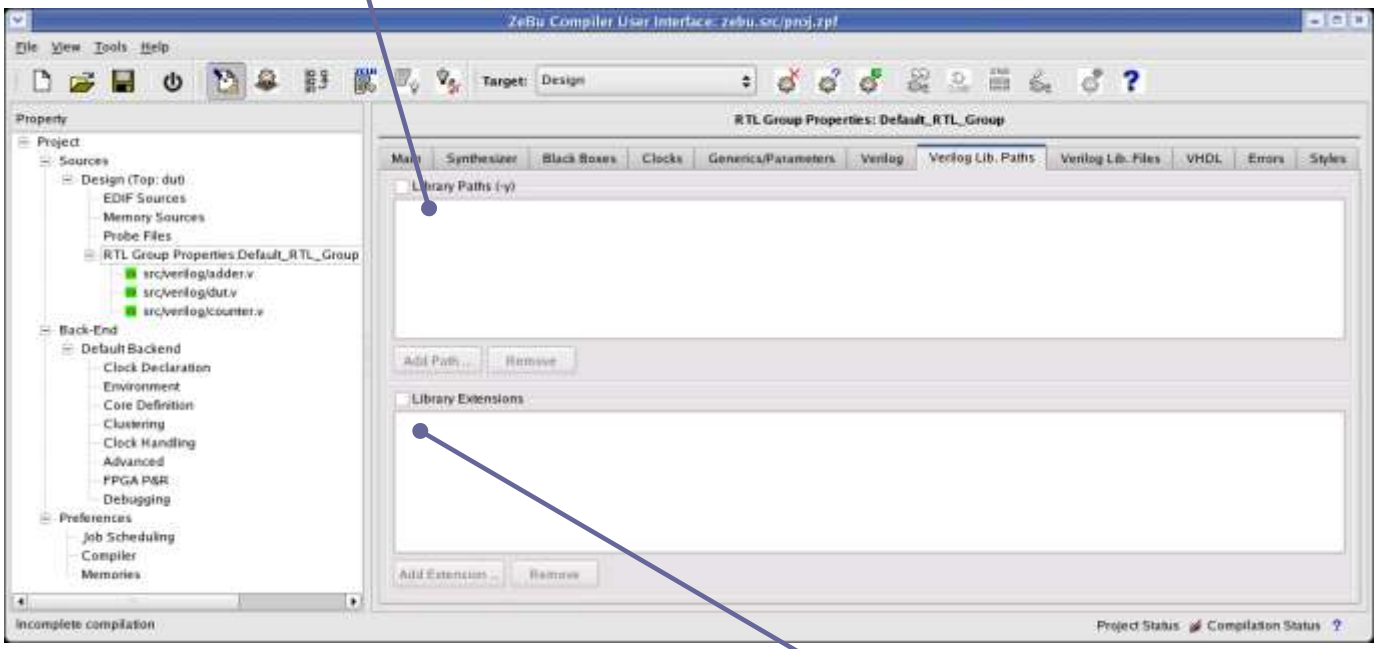
Add include paths



Add top-level Verilog macro

RTL Front-End
Verilog Lib. Paths panel

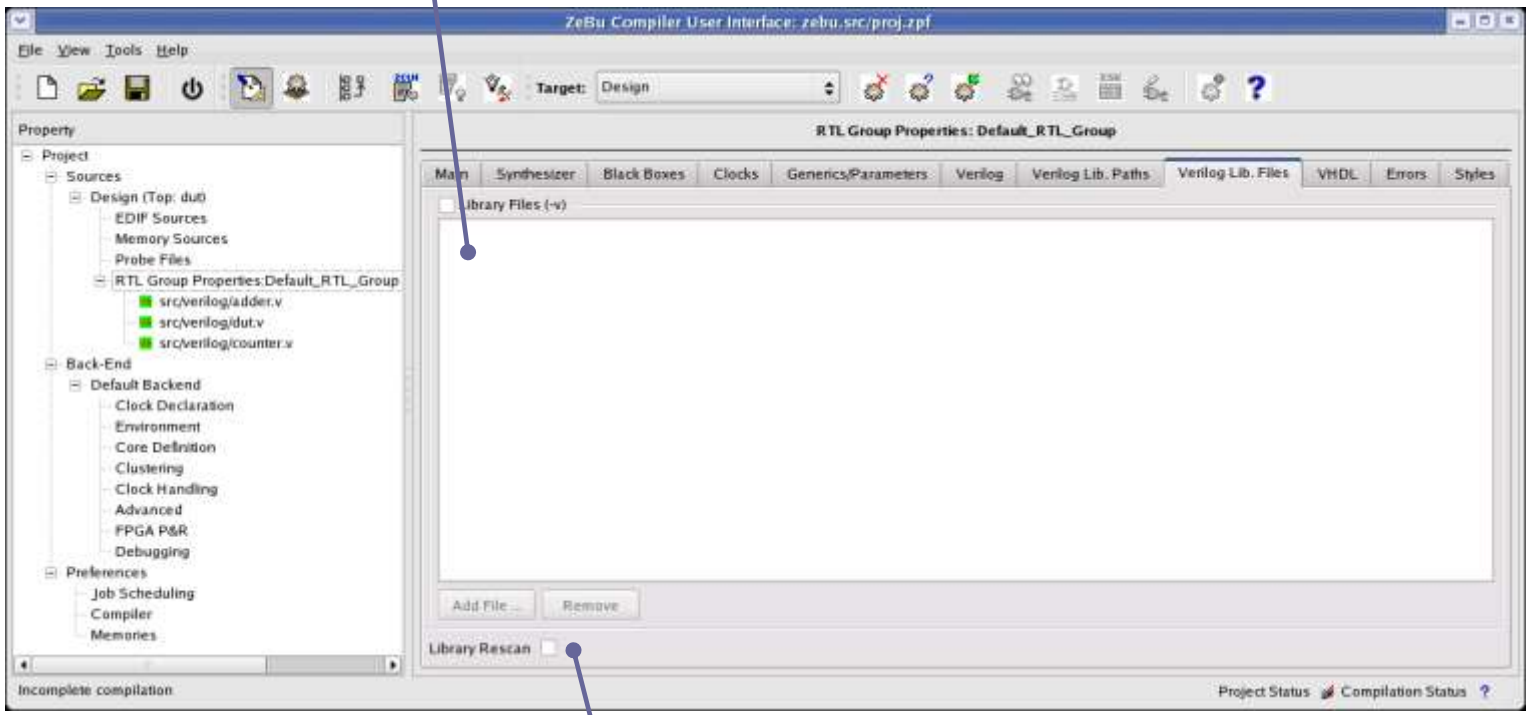
Add library paths



Define Verilog library file name extension

RTL Front-End
Verilog Lib. Files panel

Add library files

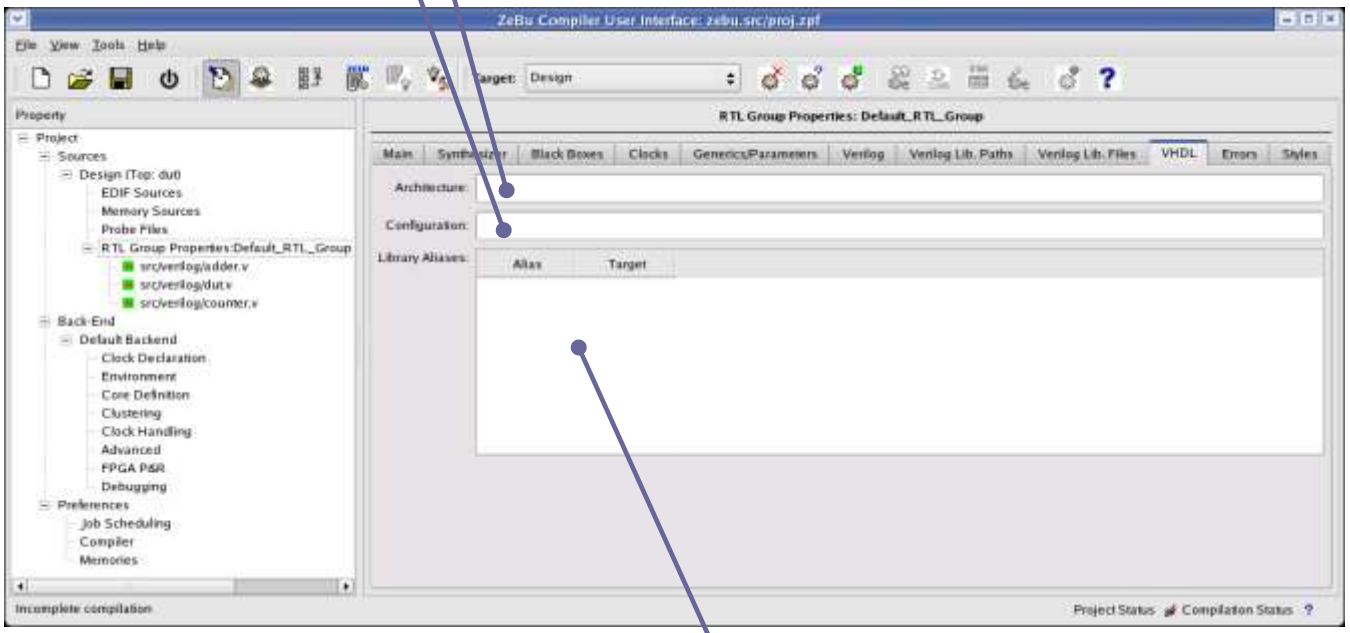


Force library file rescan

RTL Front-End
VHDL panel

Possibly, specify architecture name

Possibly, specify configuration name

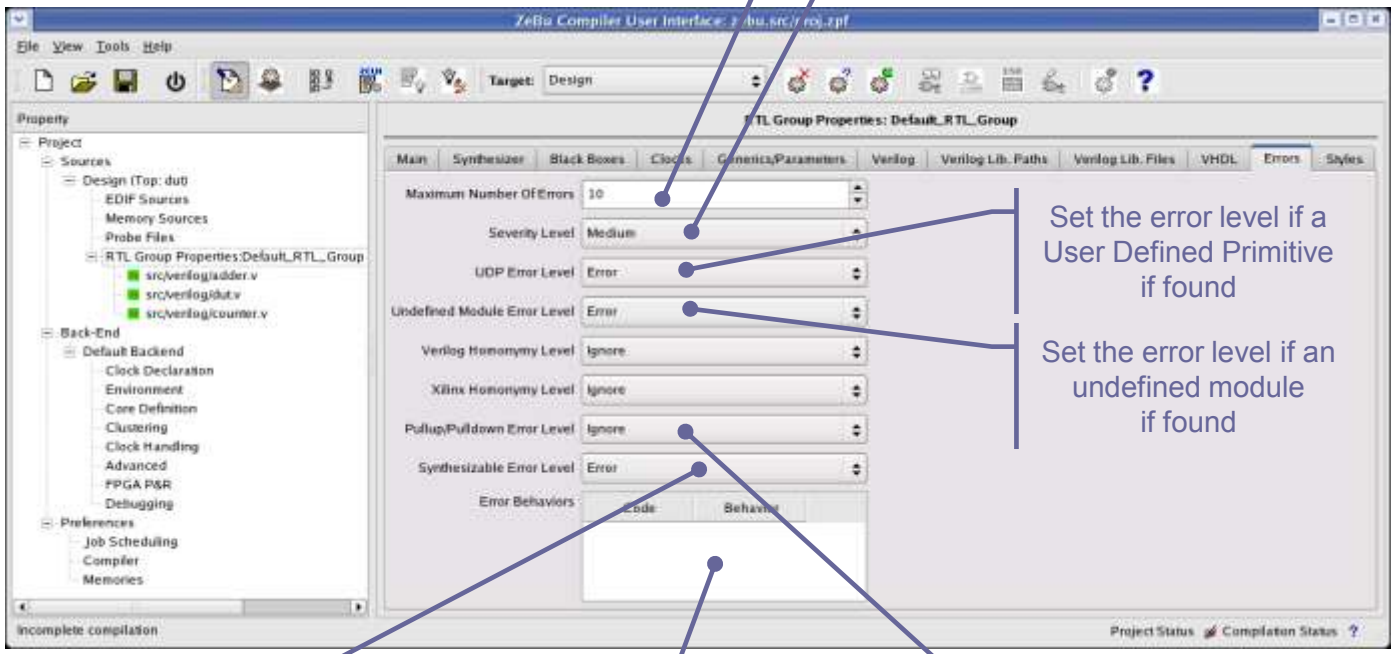


Add library aliases
or specify aliases file name (VCS format)

RTL Front-End
Errors panel

Maximum number of errors to process

Set the severity level:
- High: stops on warnings
- Medium: stops on errors
- Low: stops on some errors
- No: never exits on error



Set the error level if a User Defined Primitive if found

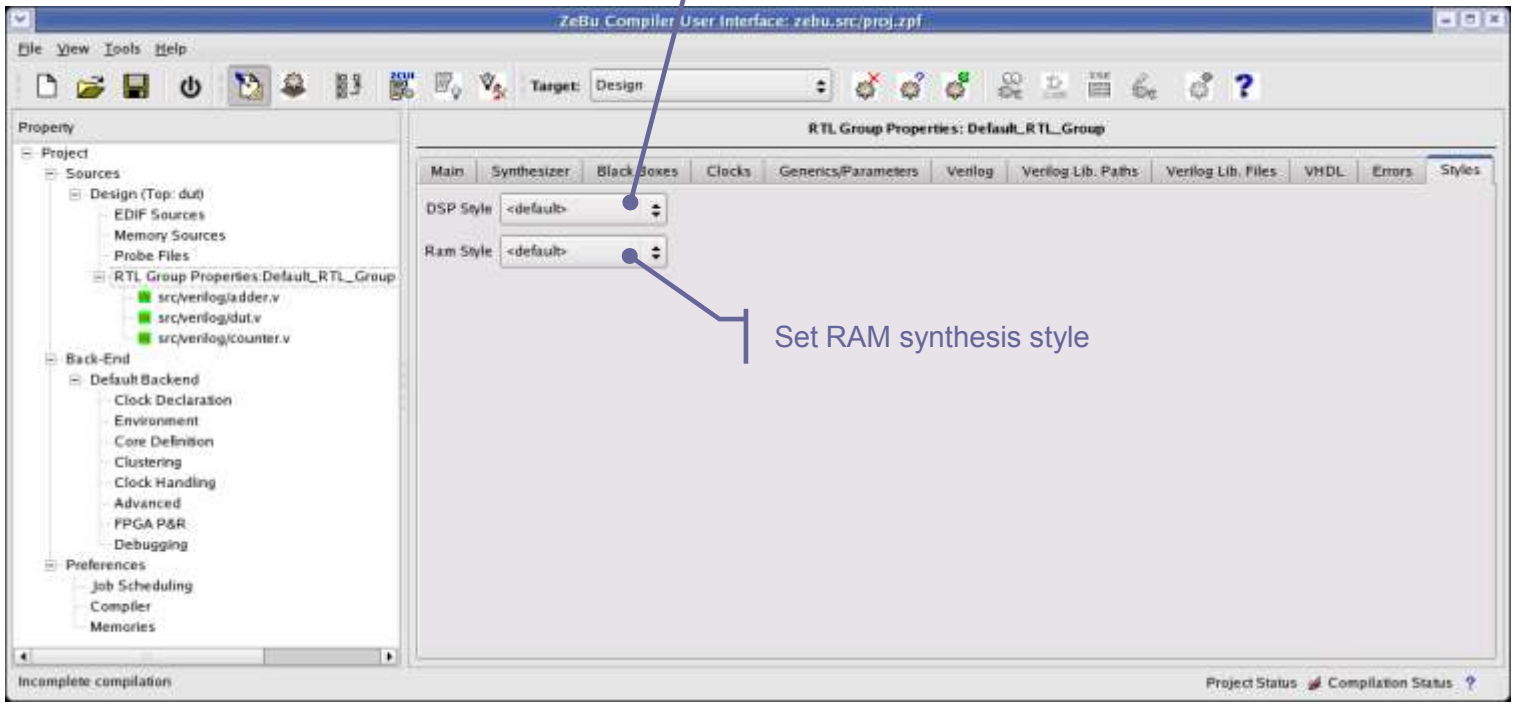
Set the error level if an undefined module if found

Set the error level if a non-synthesizable construct if found

Define behavior per error code

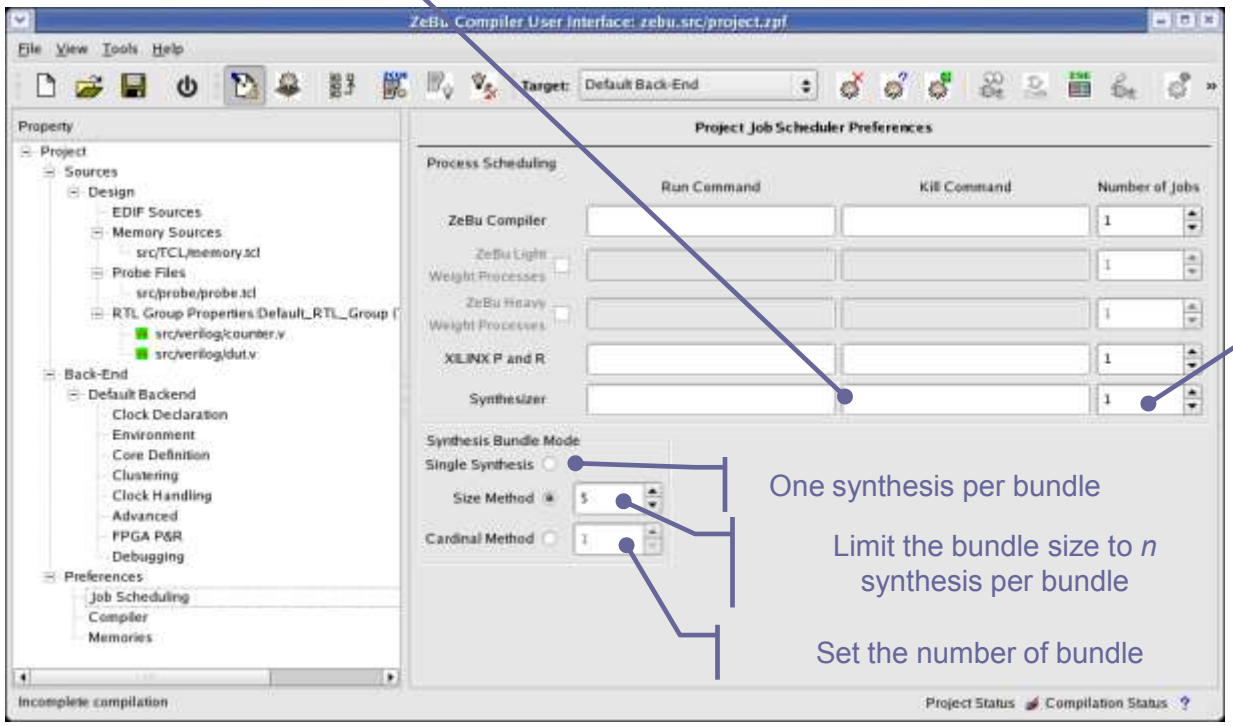
Set the error level if a `pullup` or `pulldown` if found
Can be set to repair to actually synthesize it

RTL Front-End
Styles panel



RTL Front-End
Job Scheduler Preferences panel

Possibly, specify a scheduler pre-command for synthesis



One synthesis per bundle

Limit the bundle size to *n* synthesis per bundle

Set the number of bundle

Specify the maximum number of synthesis jobs to be run in parallel

RTL Front-End
Lab1

- **In this lab you will learn how to set up a basic project and launch synthesis using third-party synthesizers**

```
Trainee/lab1
|-- RtlFrontEnd
`-- dut
    |-- probe
    `-- verilog
        |-- adder.v
        |-- counter.v
        `-- dut.v
```

RTL Front-End
Lab1

- Go into the `RtlFrontEnd` directory
- Launch `zCui`
- Declare the 3 DUT source files:
  - Right-click on the RTL group, then select “Add Verilog Sources …”
  - In the file browser, go into `../dut/verilog` and select the 3 DUT source files: `adder.v`, `counter.v` and `dut.v`. Click on “Open”
- Set RTL group top name and select the synthesizer to be used:
  - Left-click on the RTL group, in the “Main” tab:
    - Enter the DUT top name
    - Select the synthesizer of choice

RTL Front-End
Lab1

- Set the synthesizer options:
  - Click on the “Synthesizer” tab
  - Select “Block-based” mode
  - Turn on the “Optimization vs Debugging Facilities” checkbox
  - Select “Keep all Registers” as “Accessibility Level”
- Set the DUT top name:
  - Left-click on the “Design” property
  - Set the DUT top name
- Save the project file, name it “`project`”

RTL Front-End
Lab1

- Create a probe file
  - With a text editor create a file named `probe.tcl` in the `../dut/probe` directory
  - Declare the wire `dut.count2.cnt` as static probe
  - Declare the wires `dut.adder.inA` and `dut.adder.inB` as flexible probes, give `"thegroup"` as name to the flexible probe group
  - In `zCui`, right-click on the “RTL-based Compilation Scripts” property and select “Add RTL-based Compilation Scripts …” and choose the file `../dut/probe/probe.tcl` in the file browser. Click on “Open”

RTL Front-End
Lab1

- Select the target hardware configuration file
  - Left-click on the “Back-End : default” property
  - Open the file browser and select the target hardware configuration file appropriate for your ZeBu system
- Set the job scheduling preferences
  - Left-click on the “Job Scheduling” property
  - Fill the form in accordance with your network configuration
- Save the project
- Launch the synthesis:
  - Select “Design” as target
  - Click on the “Compile” button
- Once the synthesis is done, quit `zCui`

Agenda

- Overview
- Standalone synthesis
- Design entry
- RTL Front-End
- zFAST

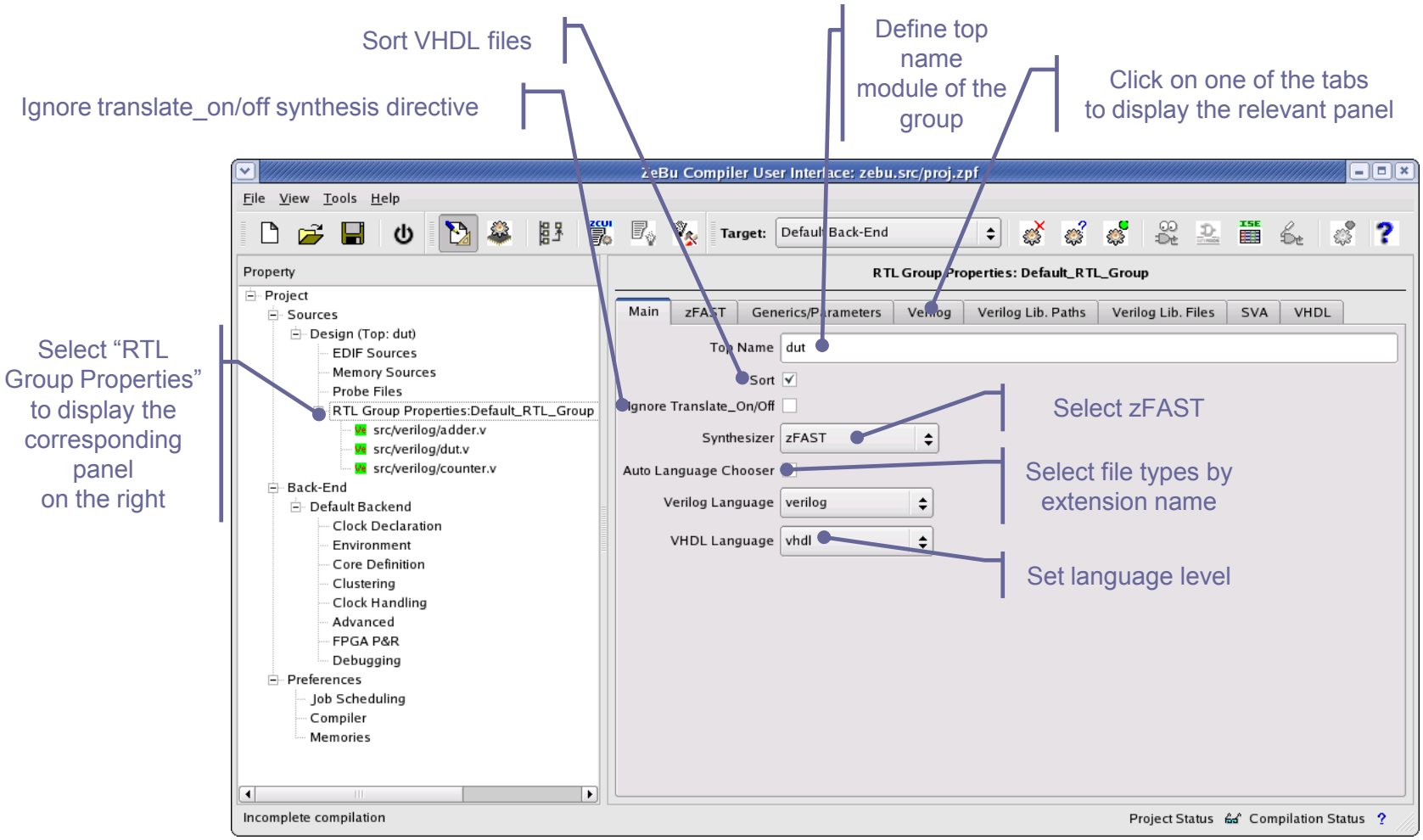
zFAST Overview

- zFAST is a fast synthesizer dedicated to the ZeBu platform
- It automatically:
  - Elaborates the design
  - Writes synthesis scripts
  - Splits the synthesis in multiple tasks
  - Launches synthesis tasks in parallel
- It is integrated in `zCui`
- It consumes one synthesizer license per parallel task
- Licenses for Verilog, for VHDL and for SystemVerilog

zFAST Overview

- 10X to 20X faster than third-party synthesizers
- Synthesizes hundreds of millions of ASIC gates in minutes, not hours
- Supports User Defined Primitives (UDP)
- Automatic memory inference
- Supports SystemVerilog Assertions (SVA)
- Emulation-friendly
  - Preserves RTL names
  - Inferred memory content available at runtime
- Area is comparable to third-party synthesizer (-10% to +25%)
- Emulation speed is equivalent

zFAST
Main panel



zFAST
zFAST panel

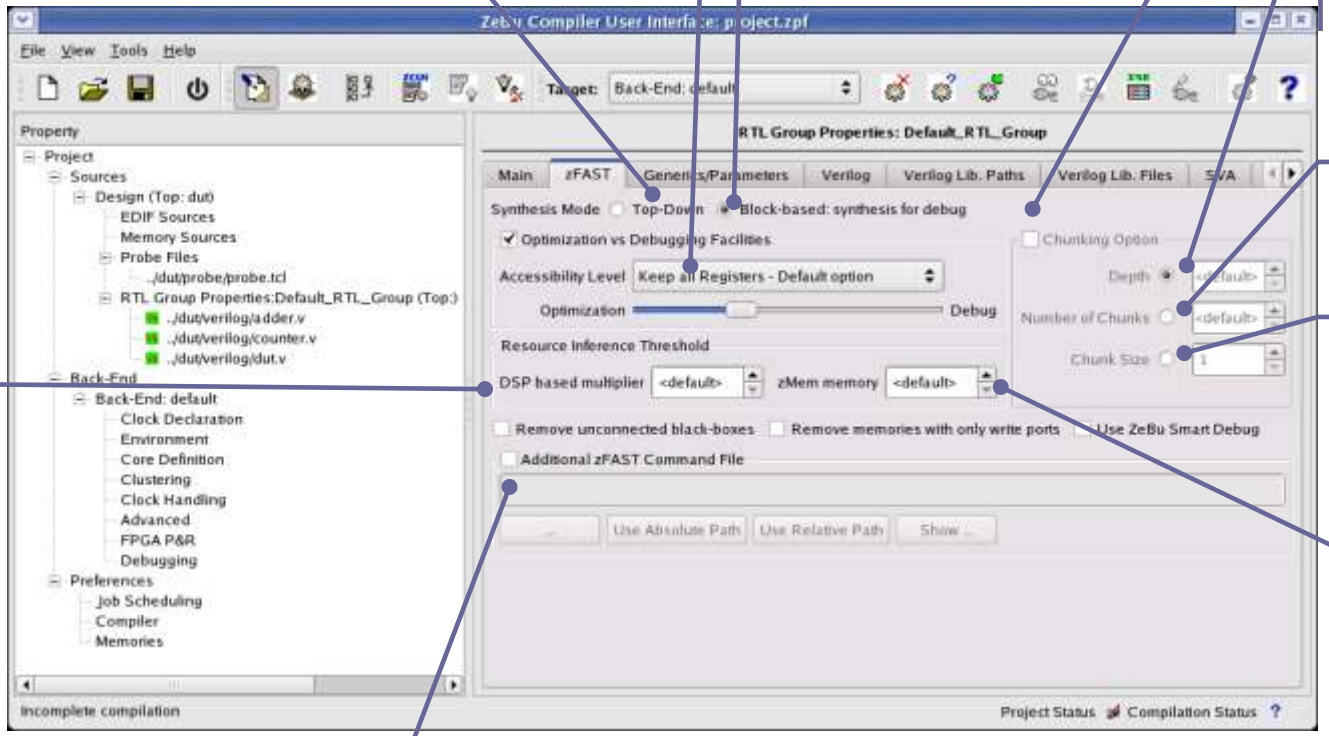
Select accessibility level:
- Max optimization
- Keep all registers
- Keep all registers and simulate combinational signals
- Keep all registers and add probes for combinational signals

Define top name module of the group

In Top-Down mode, it is possible to split the synthesis into chunks

Select Top-Down or Block based synthesis

Cut the design by groups of *n* levels of hierarchy



Cut the design in *n* chunks

Cut the design in *n* chunks based on a proportion of number of code lines

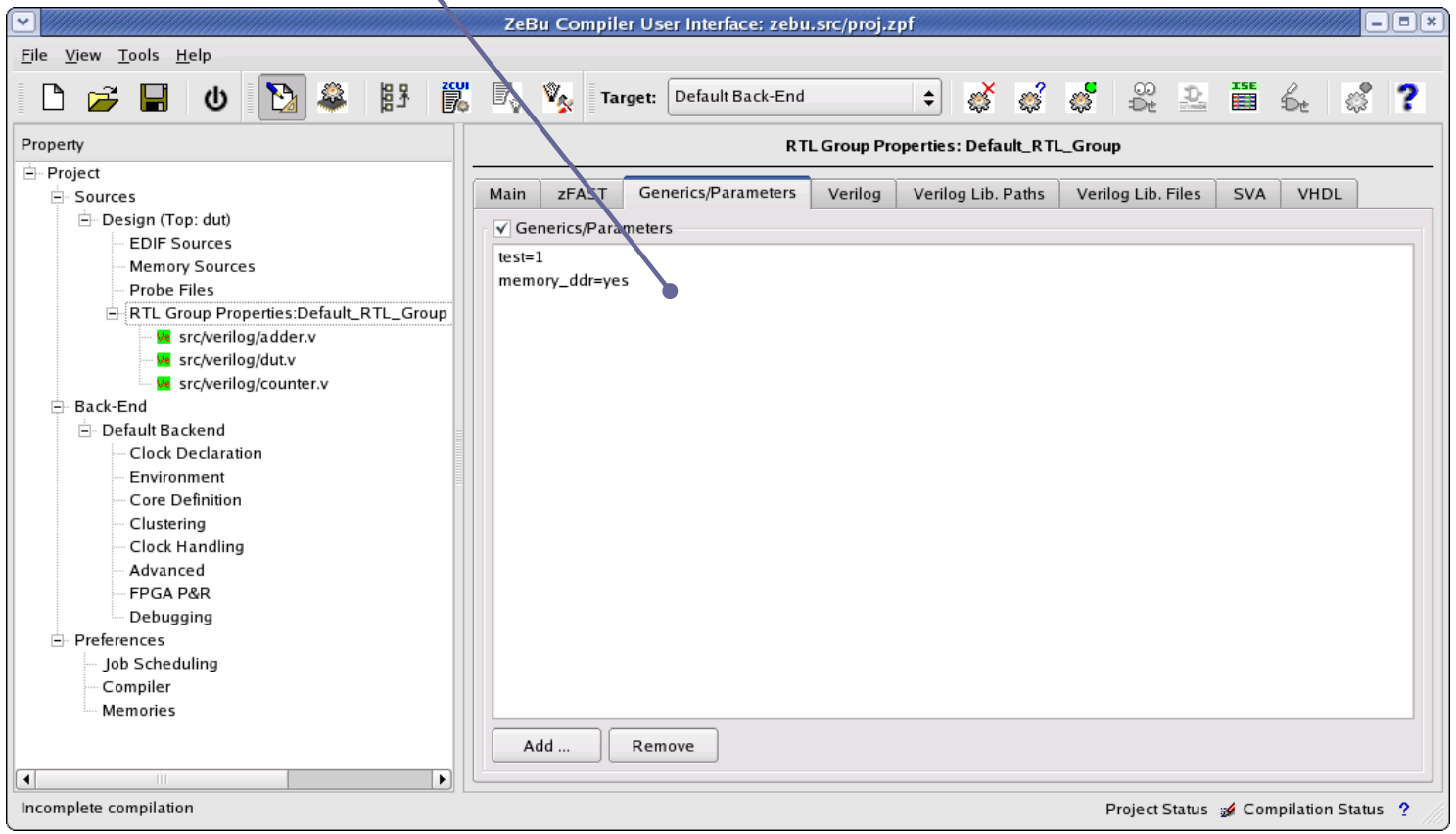
Threshold for DSP inference

Threshold for memory inference as registers

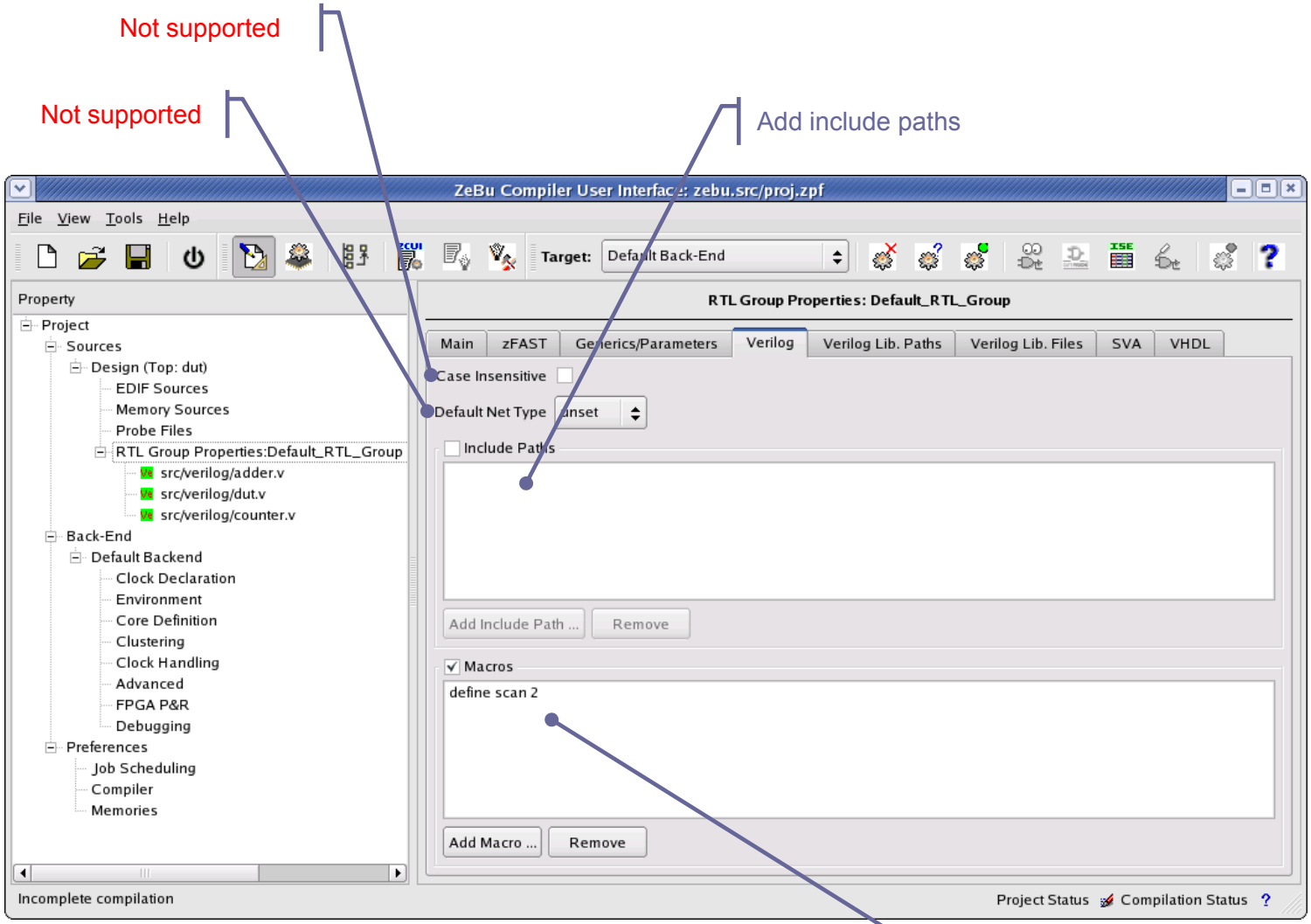
Advanced zFAST command file

zFAST
Generics/Parameters panel

Add top level generics for VHDL, parameters for Verilog

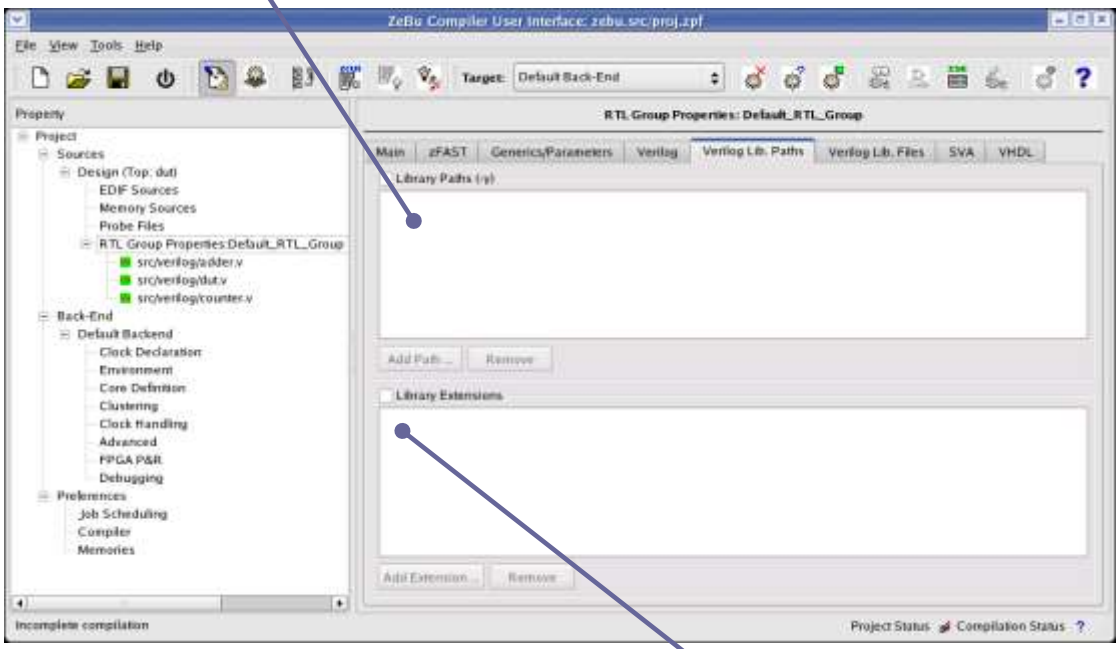


zFAST
Verilog panel



zFAST
Verilog Lib. Paths panel

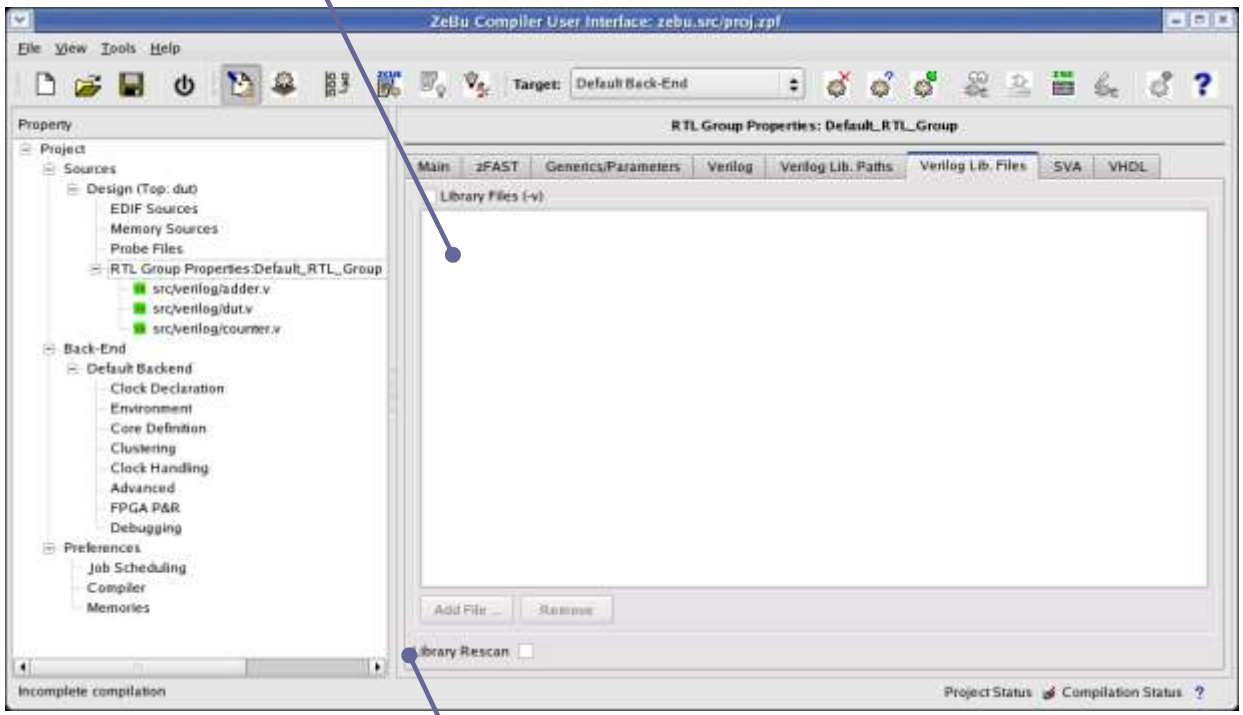
Add library paths



Define Verilog library file name extension

zFAST
Verilog Lib. Files panel

Add library files

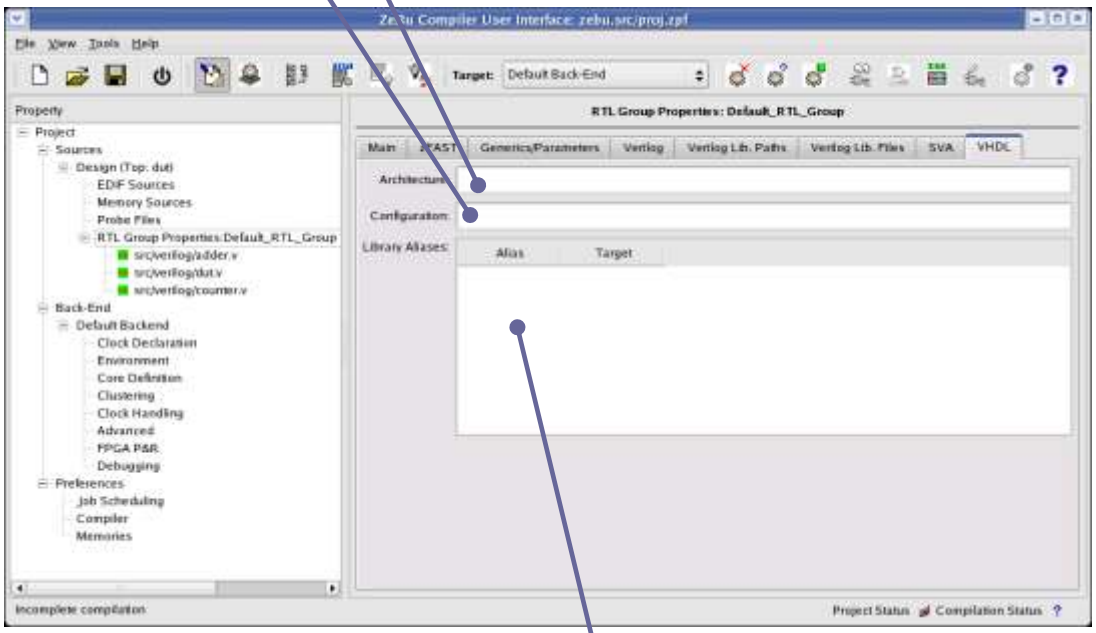


Force library file rescan

zFAST
VHDL panel

Possibly, specify architecture name

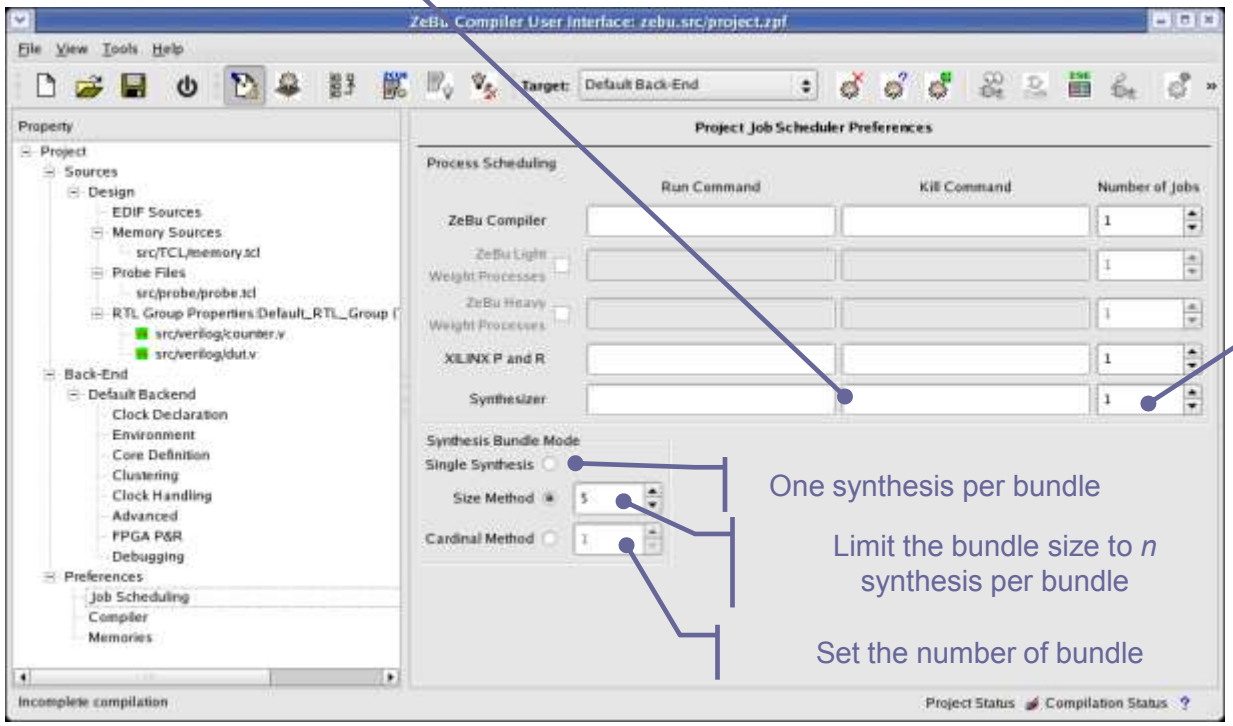
Possibly, specify configuration name



Add library aliases
or specify aliases file name (VCS format)

zFAST
Job Scheduler Preferences panel

Possibly, specify a scheduler pre-command for synthesis



One synthesis per bundle

Limit the bundle size to *n* synthesis per bundle

Set the number of bundle

Specify the maximum number of synthesis jobs to be run in parallel

zFAST
Lab1

- **In this lab you will learn how to set up a basic project and launch synthesis using zFAST synthesizer**

```
Trainee/lab1
|-- zFAST
`-- dut
    |-- probe
    `-- verilog
        |-- adder.v
        |-- counter.v
        `-- dut.v
```

zFAST
Lab1

- Go into the `zFAST` directory
- Launch `zCui`
- Declare the 3 DUT source files:
  - Right-click on the RTL group, then select “Add Verilog Sources …”
  - In the file browser, go into `../dut/verilog` and select the 3 DUT sources files: `adder.v`, `counter.v` and `dut.v`. Click on “Open”
- Set RTL group top name and select the synthesizer to be used:
  - Left-click on the RTL group, in the “Main” tab:
    - Enter the DUT top name
    - Select the zFAST as synthesizer

zFAST
Lab1

- Set the zFAST options:
  - Click on the “zFAST” tab
  - Select “Block-based” mode
  - Turn on the “Optimization vs Debugging Facilities” checkbox
  - Select “Keep all Registers” as “Accessibility Level”
- Set the DUT top name:
  - Left-click on the “Design” property
  - Set the DUT top name
- Save the project file, name it “`project`”

zFAST Lab1

- Create a probe file
  - With a text editor create a file named `probe.tcl` in the `../dut/probe` directory
  - Declare the wire `dut.count2.cnt` as static probe
  - Declare the wires `dut.adder.inA` and `dut.adder.inB` as flexible probes, give `"thegroup"` as name to the flexible probe group
  - In `zCui`, right-click on the “RTL-based Compilation Scripts” property and select “Add RTL-based Compilation Scripts …” and choose the file `../dut/probe/probe.tcl` in the file browser. Click on “Open”

zFAST
Lab1

- Select the target hardware configuration file
  - Left-click on the “Back-End : default” property
  - Open the file browser and select the target hardware configuration file appropriate for your ZeBu system
- Set the job scheduling preferences
  - Left-click on the “Job Scheduling” property
  - Fill the form in accordance with your network configuration
- Save the project
- Launch the synthesis:
  - Select “Design” as target
  - Click on the “Compile” button
- Once the synthesis is done, quit `zCui`