Overview

ZeBu-Server Training

*THE **FASTEST** VERIFICATION*

Agenda

- Overview
- Synthesis
- Compilation
- HDL co-simulation
- C/C++ co-simulation
- SystemC co-simulation
- Using transactors
- Designing zcei transactors
- Designing ZEMI-3 transactors
- SVA

Agenda

- Overview
  - Company overview
  - ZeBu-Server hardware overview
  - ZeBu-Server software overview
  - Smart Debug methodology

Company Background

- Founded in 2000 by leading experts in emulation
  - Since 1994, developed 8 generations of emulation systems
- #1 worldwide HW/SW co-emulation provider
- #2 worldwide emulation provider
- Offices in San Jose, Paris, Tokyo, Seoul, Bangalore, Hsinchu
  - Distributors in Israel, China
- Currently 100+ employees (50% R&D)
- 300+ systems shipped to 60+ customers

Used by #1 Companies

Wireless/Mobile

QUALCOMM
SiRF
[redacted]
NXP founded by Philips
RENESAS
MARVELL
ETRI
NEC
Panasonic
Quartics
ST

Multifunction Printers

Major MFP Company
KONICA MINOLTA
TOSHIBA

Graphics/Audio/Video

SANYO
ST
AMD
cea
FUJITSU
THE AEROSPACE CORPORATION
Magnum SEMICONDUCTOR
DiBcom The Heart of Mobile TV
Major Camcorder Company
LSI
SUNPLUS
OLYMPUS
LG Life's Good
CONNEX TECHNOLOGY
GENNUM CORPORATION
BROADCOM
NORTHROP GRUMMAN

Processors

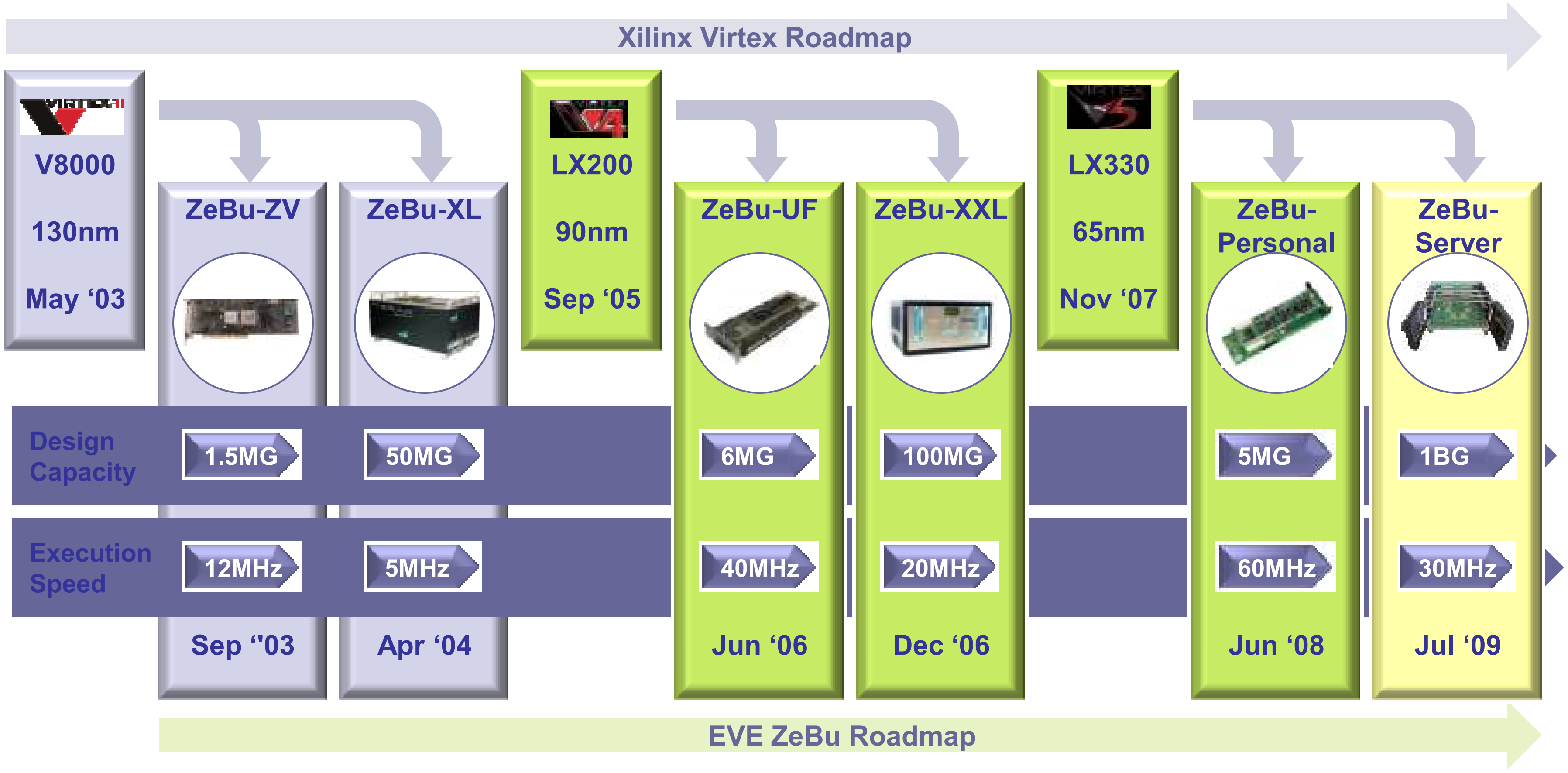
Major Processor Company
ARM
TOSHIBA
PA SEMI
tensilica
Sun microsystems
FUJITSU
Transmeta CORPORATION

Networking/Telecom/Storage/Automotive/Testers

Teranetics
CONEXANT
nDisk
Seagate
[redacted]
AQUANTIA
BROADCOM
LTX CORPORATION

ZeBu: A Family of Products With a Proven Track Record of Innovation

- *Scalable Architecture enables New Product Releases Every 12-18 months, an unrivaled pace in this industry*



V8000	130nm	May ‘03
LX200	90nm	Sep ‘05
LX330	65nm	Nov ‘07

	Design Capacity	Execution Speed	
ZeBu-ZV	1.5MG	12MHz	Sep ‘’03
ZeBu-XL	50MG	5MHz	Apr ‘04
ZeBu-UF	6MG	40MHz	Jun ‘06
ZeBu-XXL	100MG	20MHz	Dec ‘06
ZeBu-Personal	5MG	60MHz	Jun ‘08
ZeBu-Server	1BG	30MHz	Jul ‘09

Agenda

- Overview
  - Company overview
  - ZeBu-Server hardware overview
  - ZeBu-Server software overview
  - Smart Debug methodology

Three Types of Units

- Each slot can load a FPGA module
- Each module can be connected to a host PC
- At least one PC needs to be connected in each system

2 slots

5 slots (mono-unit)

5 slots (multi-unit)

In-Circuit Emulation Connectivity

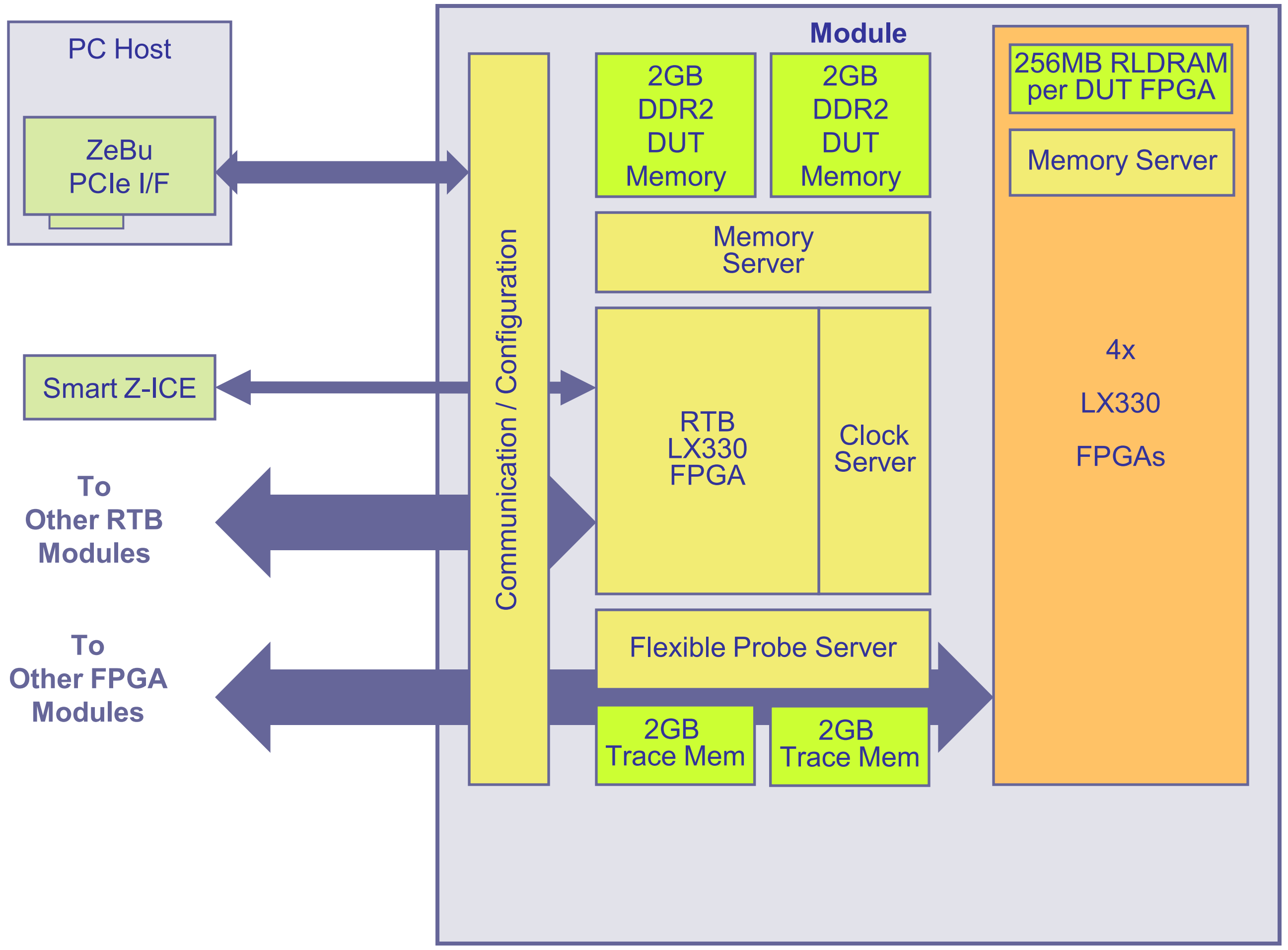
- **Smart Z-ICE**
  - On the back of unit
  - 4 connectors in 2-slot units
  - 5 connectors in 5-slot units
  - 16 data (bidir) + 1 clock per connector
  - Typical usage: can be used to connect a software debugger probe

- **Direct ICE**
  - On the top of unit
  - Optional board
  - Compatible with ZeBu-XXL Direct ICE connector
  - 1,200 data (bidir)
  - Needs to be connected internally to 1 or 2 8-FPGA Direct ICE modules
    - (1,200 or 600+600 data)
  - Typical usage: can be used to connect a user target board or system

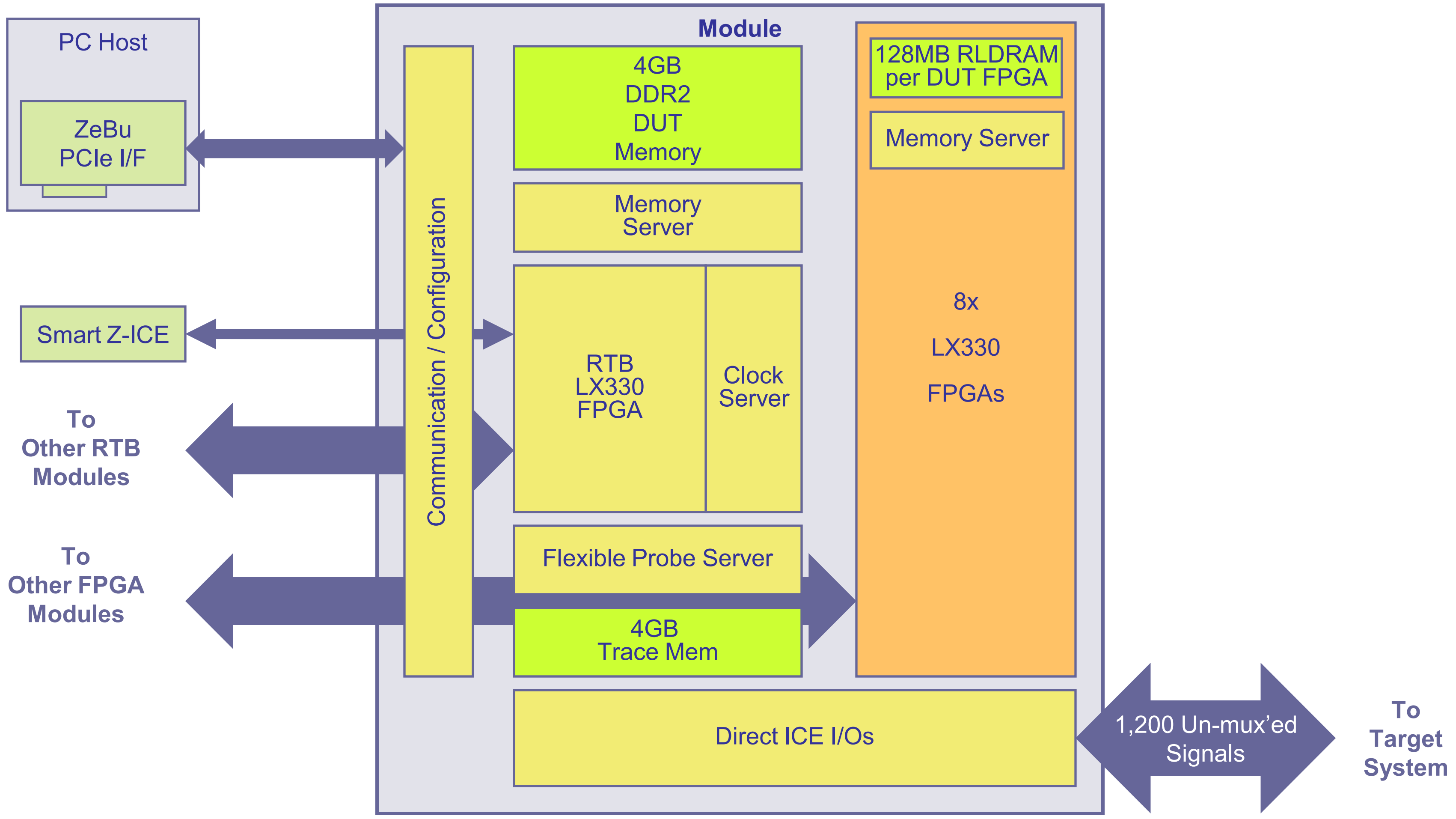
Three Types of Modules

- 4-FPGA modules
- 8-FPGA / Direct ICE modules
- 16-FPGA modules
- All modules include RAM

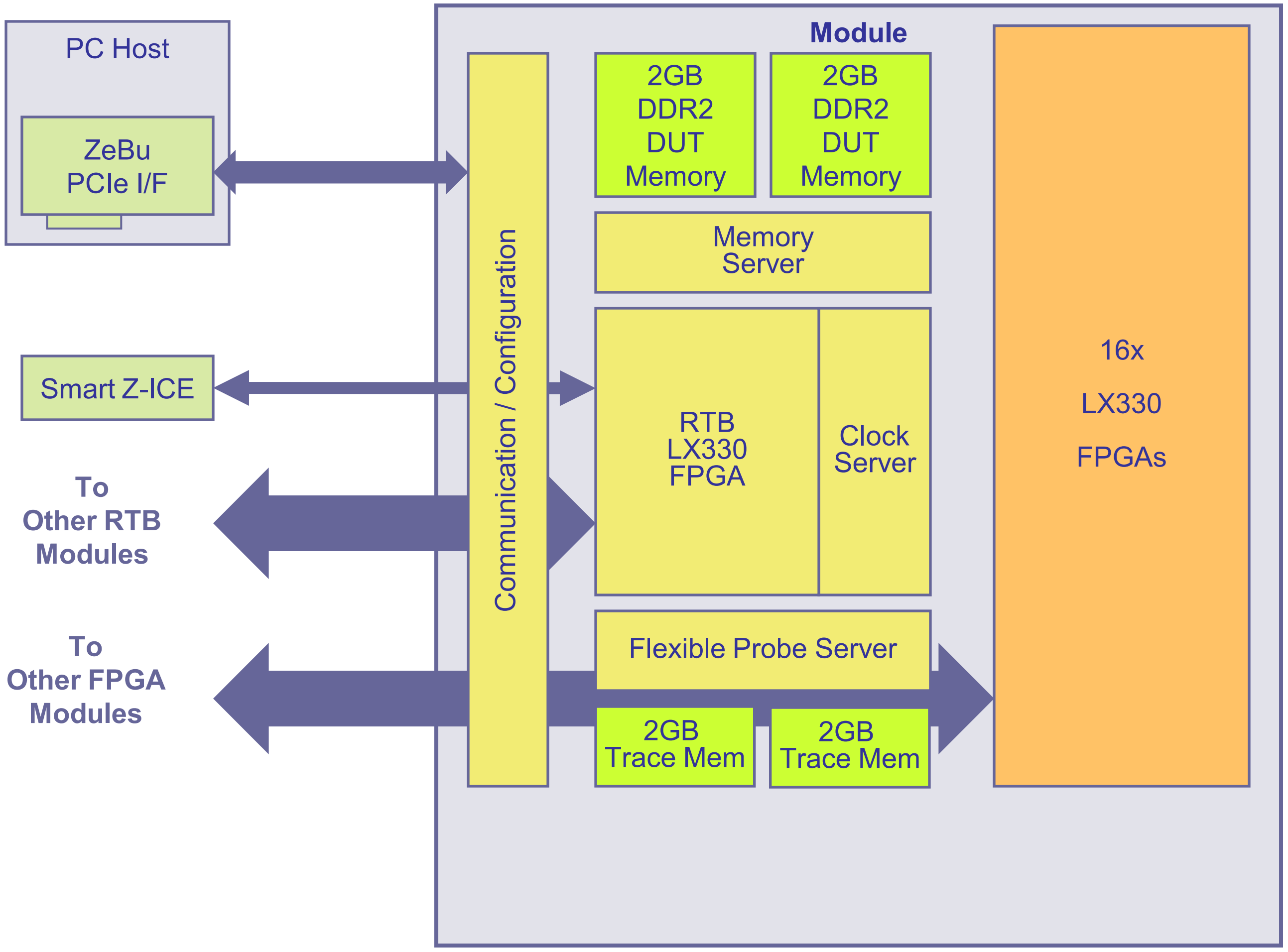
4-FPGA module: Physical Diagram



8-FPGA / Direct ICE module: Physical Diagram



16-FPGA module: Physical Diagram



Agenda

- Overview
  - Company overview
  - ZeBu-Server hardware overview
  - ZeBu-Server software overview
  - Smart Debug methodology

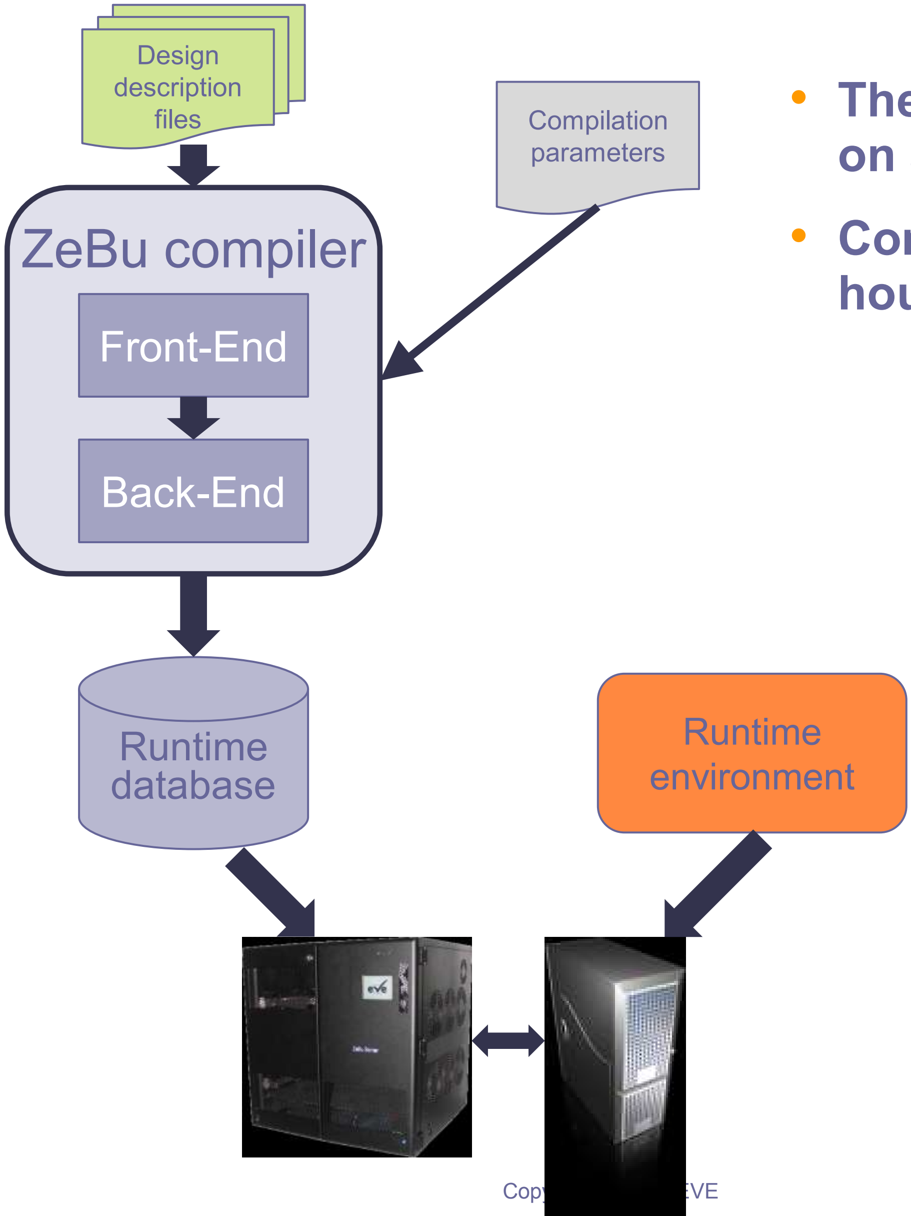
Emulation Modes

Verification mode	Specification	ZeBu-Server
HDL co-simulation	Commercial HDL simulators •ModelSim •NC-Sim •VCS	5K-100KHz
Signal level co-simulation	C/C++, SystemC	100K-500KHz
Transactional level co-simulation	C/C++, SystemC	200K-20MHz
Test vectors emulation	Vector files	100K-500KHz
Stand-alone emulation	Synthesizable testbench	2MHz-20MHz
In-Circuit emulation	Connected to target system	2MHz-20MHz

Introduction to ZeBu Software

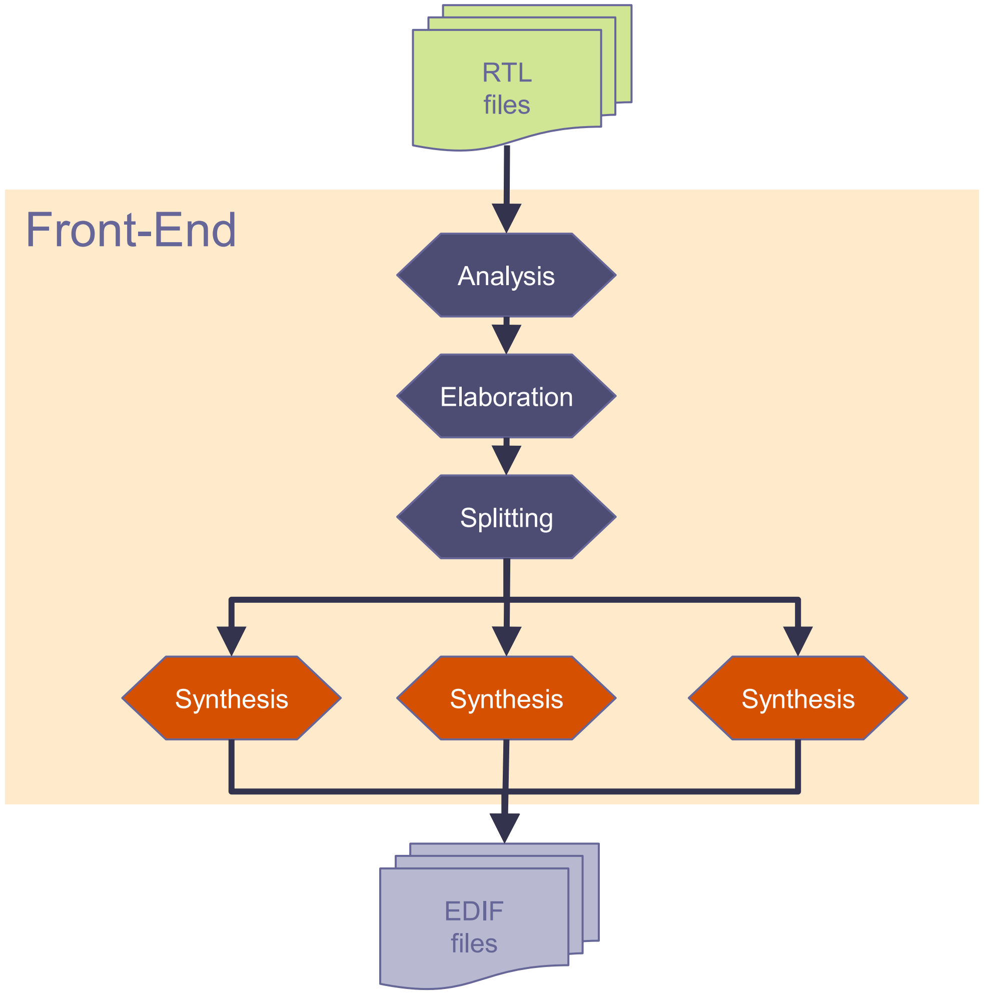
- Compilation
  - Front-End
    - RTL Front-End + 3rd party synthesizer
    - or zFAST (Verilog/VHDL/SystemVerilog)
      - SVA
    - ZEMI-3
    - ZW-FPGA
  - Back-End
    - ZeBu Back-End Compiler
    - Xilinx ISE FPGA Place & Route
- Runtime

ZeBu Compilation Flow

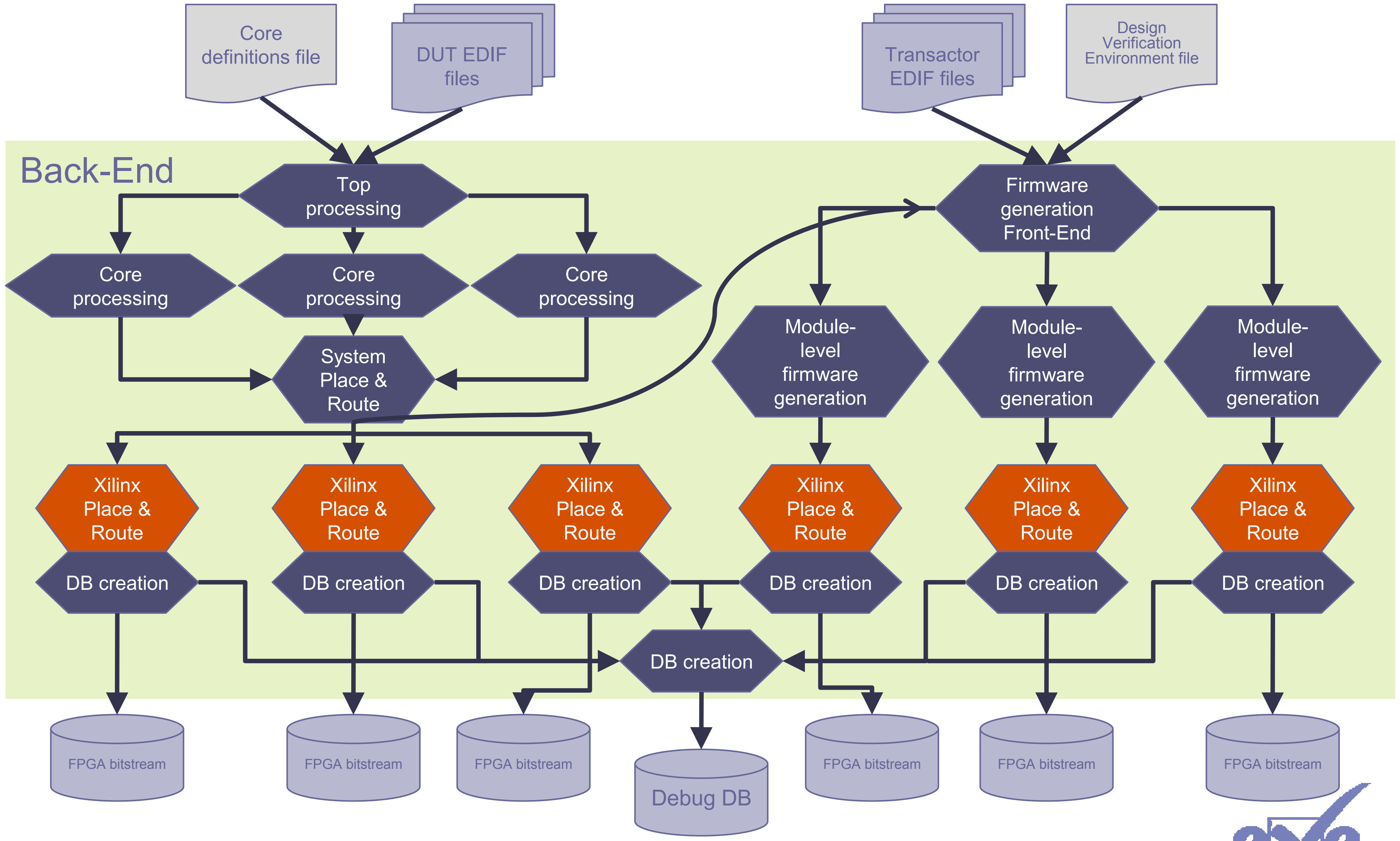


- The ZeBu compiler can run on a PC farm
- Compilation time is about 2 hours on a PC farm

ZeBu Front-End Compilation Flow



ZeBu Back-End Compilation Flow (1/2)



ZeBu Back-End Compilation Flow (2/2)

- Top processing
  - Executes netlist editing commands
  - Splits cores
- Core processing
  - Netlist optimization
  - Tristate busses resolution
  - Clock handling
  - Memory handling
  - FPGA clustering

- Firmware generation Front-End
  - Clock allocation
- Module-level firmware generation
  - RTB FPGAs netlist generation
  - Clock generator
  - Transactor instantiations
  - Trace memory logic
  - Trigger logic

ZeBu Commands Categories

- ZeBu commands can be found in the `bin` directory of the software installation
- They are organized in different categories:
  - System utilities
  - RTL Front-End
  - zFAST synthesizer
  - ZEMI-3
  - Compilation
  - Runtime
  - Debug
- Commands in green can be run by the user
- Commands in red are normally not of user interest, they are run automatically by other commands
- Most synthesis and compilation commands are run automatically by the `zCui` graphical user interface

System Utilities

- zInstall
  - Installs the ZeBu board driver
- zInitSystem
  - Sets up the ZeBu system
- zUtils
  - Runs the diagnostics and scans the configuration scanning
- zConfig
  - Creates the compiler's hardware target configuration file
- zXtorSetup
  - Sets up Linux for fast transactional accesses
- zUpdate
  - Updates the ZeBu PCIe board firmware
- ztclsh
  - ZeBu TCL shell
- zwish
  - ZeBu wish shell

System Utilities
RTL Front-End

- zRFEvlog
  - Simulator command replacement, Verilog analysis
- zRFEvhdl
  - Simulator command replacement, VHDL analysis
- zRFEelab
  - Simulator command replacement, design elaboration
- zFEui
  - Front-End graphical user interface
- zRtlFrontEnd
  - RTL Front-End analysis, elaboration and splitting
- PostFE
  - RTL Front-End post processor
- synthesis_launcher
  - Used to launch XST synthesizer

System Utilities
zFAST Synthesizer

- zFe
  - zFAST Front-End
- vhs
  - Verilog synthesis
- vhorder
  - Sort VHDL file order
- hcs_mixed
  - VHDL synthesis
- zDivMod
  - Divider/Multiplier macro generator
- zFastStatCollate
  - Creates a design statistics database
- zFastStatBrowse
  - Browses the design statistics database

System Utilities
ZEMI-3

- zEmiComp
  - ZEMI-3 compiler
- zEmiRun
  - ZEMI-3 runtime utility

System Utilities Compilation

- zCui
  - Graphical user interface for project management
- zNetgen
  - Netlist processing utility
- zMem
  - ZeBu memory model generator
- zTopBuild
  - Processes design netlists at system level (top level)
- zCoreBuild
  - Processes design netlists at core level
- zPar
  - System-level Place & Route
- zRTB_FE
  - Front-End firmware generator
- zRTB_FW
  - Back-End firmware generator
- zTime
  - Timing analysis utility
- zDB
  - Debugs database creation
- zDbPostProc
  - Debugs database post-processing
- zVisiPostProc
  - Simulated Combinational Signals database creation
- dbFlagger
  - Flags the RTL names in the debug database

System Utilities Runtime

- zRun
  - Graphical user interface providing a runtime debug environment
- zServer
  - ZeBu board server process
- zPinCheck
  - Checks used pins against a faulty pins list
- zPattern
  - Applies a test-pattern vector to a design
- bin2vcd
  - Converts ZeBu binary waveform format to a standard VCD file
- hex2bin
  - Converts hexadecimal format to ZeBu binary memory content file format
- zState
  - Allows converting, printing and comparing hardware and logic states of the emulator
- zsvaReport
  - SVA report post-processor
- ztdb2fsdb
  - Flexible Local Probes waveform format converter (to fsdb)
- ztdb2vcd
  - Flexible Local Probes waveform format converter (to VCD)

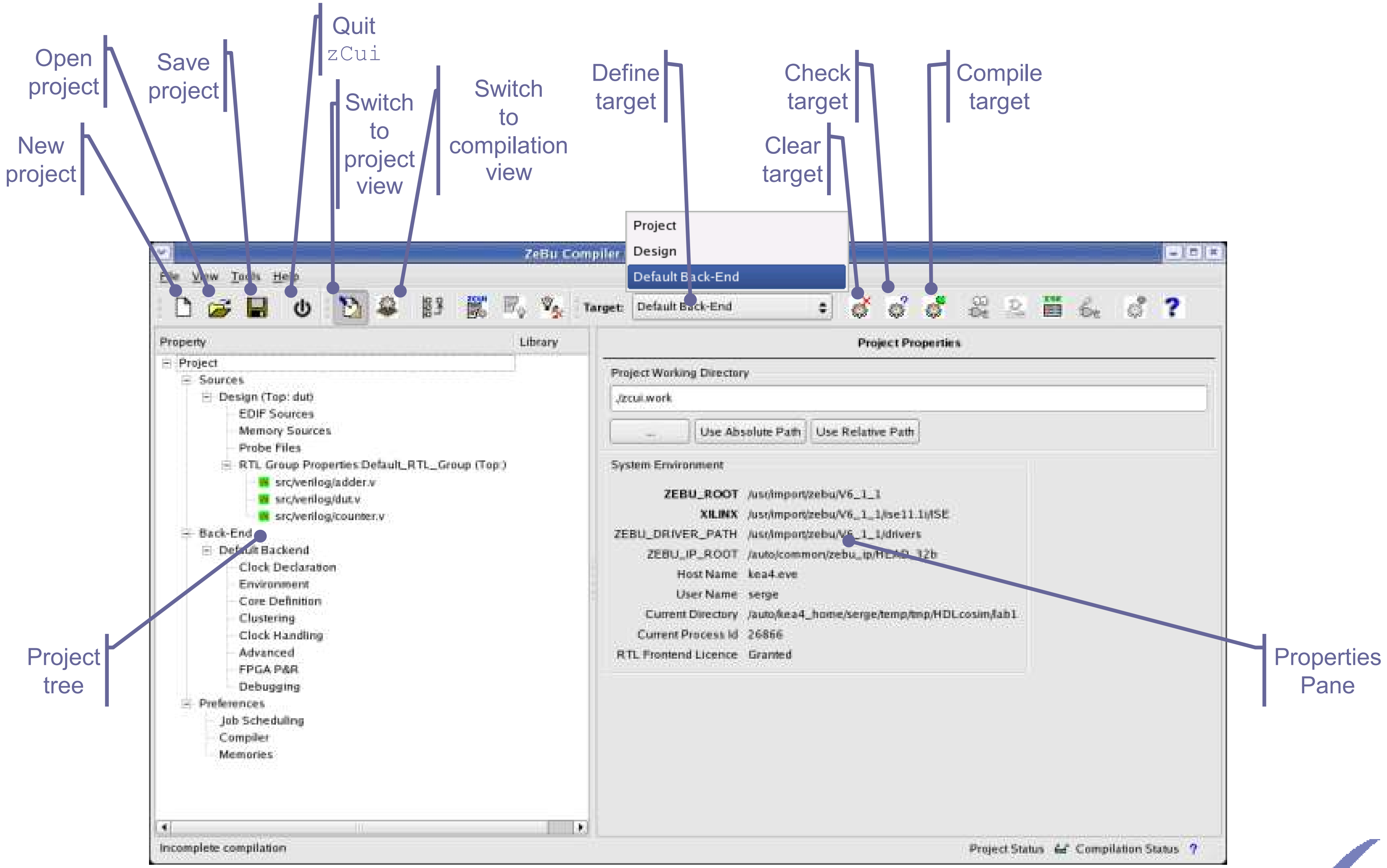
System Utilities
Debug

- zBrowser
  - Browses EDIF netlists
- zSelectProbes
  - Dynamic Probes selector
- zSelectSignals
  - Simulated Combinational Signals probes selector

Introduction to `zCui`

- ZeBu Compiler User Interface
  - Graphical user interface
  - Allows listing of all project source files
  - Allows setting of compilation parameters
  - Manages ZeBu compilation project
  - Automatically creates scripts and launches individual tools
    - Handles the Front-End and the Back-End
  - Supports parallel processing
    - Support for PC farm
  - Supports incremental processing

Introduction to `zCui` Project view



Property | Library

- Project
  - Sources
    - Design (Top: dut)
      - EDIF Sources
      - Memory Sources
      - Probe Files
      - RTL Group Properties:Default_RTL_Group (Top:)
        - src/verilog/adder.v
        - src/verilog/dut.v
        - src/verilog/counter.v
  - Back-End
    - Default Backend
      - Clock Declaration
      - Environment
      - Core Definition
      - Clustering
      - Clock Handling
      - Advanced
      - FPGA P&R
      - Debugging
  - Preferences
    - Job Scheduling
    - Compiler
    - Memories

Project Properties

Project Working Directory

./zcui.work

... | Use Absolute Path | Use Relative Path

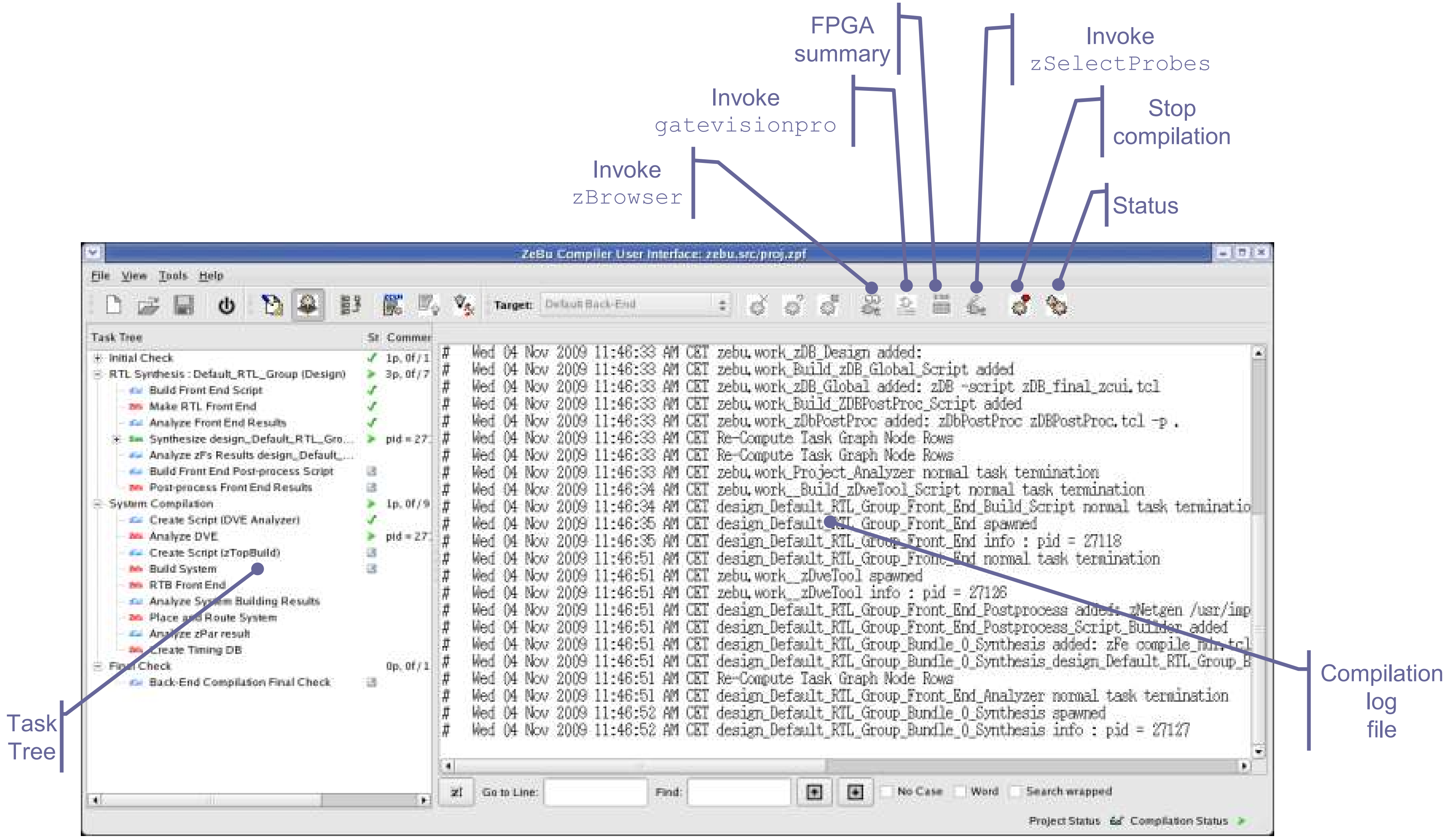
System Environment

ZEBU_ROOT	/usr/import/zebu/V6_1_1
XILINX	/usr/import/zebu/V6_1_1/ise11.1/ISE
ZEBU_DRIVER_PATH	/usr/import/zebu/V6_1_1/drivers
ZEBU_IP_ROOT	/auto/common/zebu_ip/HEAD_32b
Host Name	kea4.eve
User Name	serge
Current Directory	/auto/kea4_home/serge/temp/tmp/HDLcosim/lab1
Current Process Id	26866
RTL Frontend Licence	Granted

Incomplete compilation

Project Status | Compilation Status

Introduction to zCui Compilation view



Task Tree	St	Commer
Initial Check	✓	1p, 0f / 1
RTL Synthesis : Default_RTL_Group (Design)	➤	3p, 0f / 7
Build Front End Script	✓	
Make RTL Front End	✓	
Analyze Front End Results	✓	
Synthesize design_Default_RTL_Gro...	➤	pid = 27
Analyze zFs Results design_Default_...		
Build Front End Post-process Script		
Post-process Front End Results		
System Compilation	➤	1p, 0f / 9
Create Script (DVE Analyzer)	✓	
Analyze DVE	➤	pid = 27
Create Script (zTopBuild)		
Build System		
RTB Front End		
Analyze System Building Results		
Place and Route System		
Analyze zPar result		
Create Timing DB		
Final Check		0p, 0f / 1
Back-End Compilation Final Check		

```
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_zDB_Design added:
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_Build_zDB_Global_Script added
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_zDB_Global added: zDB -script zDB_final_zcui.tcl
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_Build_ZDBPostProc_Script added
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_zDbPostProc added: zDbPostProc zDBPostProc.tcl -p .
#   Wed 04 Nov 2009 11:46:33 AM CET Re-Compute Task Graph Node Rows
#   Wed 04 Nov 2009 11:46:33 AM CET Re-Compute Task Graph Node Rows
#   Wed 04 Nov 2009 11:46:33 AM CET zebu.work_Project_Analyzer normal task termination
#   Wed 04 Nov 2009 11:46:34 AM CET zebu.work__Build_zDveTool_Script normal task termination
#   Wed 04 Nov 2009 11:46:34 AM CET design_Default_RTL_Group_Front_End_Build_Script normal task terminatio
#   Wed 04 Nov 2009 11:46:35 AM CET design_Default_RTL_Group_Front_End spawned
#   Wed 04 Nov 2009 11:46:35 AM CET design_Default_RTL_Group_Front_End info : pid = 27118
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Front_End normal task termination
#   Wed 04 Nov 2009 11:46:51 AM CET zebu.work__zDveTool spawned
#   Wed 04 Nov 2009 11:46:51 AM CET zebu.work__zDveTool info : pid = 27126
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Front_End_Postprocess added: zNetgen /usr/imp
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Front_End_Postprocess_Script_Builder added
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Bundle_0_Synthesis added: zFe compile_run.tcl
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Bundle_0_Synthesis_design_Default_RTL_Group_B
#   Wed 04 Nov 2009 11:46:51 AM CET Re-Compute Task Graph Node Rows
#   Wed 04 Nov 2009 11:46:51 AM CET design_Default_RTL_Group_Front_End_Analyzer normal task termination
#   Wed 04 Nov 2009 11:46:52 AM CET design_Default_RTL_Group_Bundle_0_Synthesis spawned
#   Wed 04 Nov 2009 11:46:52 AM CET design_Default_RTL_Group_Bundle_0_Synthesis info : pid = 27127
```

ZeBu software installation tree

- Once installed, the ZeBu software can be found in a directory called `/zebu/<version>`, e.g.

  ```
  /usr/local/zebu/V6_2_0
  ```

- Multiple versions can co-exist but the same version must be used during compilation, runtime and ZeBu board kernel module installation
- Preferably, use the Xilinx ISE package provided in the installation package, it has been qualified to work with ZeBu software

ZeBu software installation tree

- E.g.: `/usr/local/zebu/V6_2_0`

– `bin`	Executable commands
– `include`	`.h` and `.hh` header files for C/C++ code
– `lib`	`.so` binary libraries for C/C++ code
– `doc`	User documentation
– `Tutorial`	Tutorial labs files
– `license`	License management software
– `version`	Installed version and information on patches
– `etc/configurations`	Sample HW target configuration files
– `ise`*`X.Y`*	Xilinx ISE FPGA Place & Route package
– `Systemverilog`	SystemVerilog DPI related files
– `cevision_`*`X.Y.Z`*	Concept Engineering software

  – Setup scripts

ZeBu Software Setup Scripts

- Several Linux environment variables need to be set up in order to use the ZeBu software
- Use one of these setup scripts, according to Linux shell:
  - `source <path_to_tools>/zebu/<version>/zebu_env.bash`
  - `source <path_to_tools>/zebu/<version>/zebu_env.csh`
  - `source <path_to_tools>/zebu/<version>/zebu_env.sh`
  - `source <path_to_tools>/zebu/<version>/zebu_env.tcsh`
- It will set up several environment variables such as:
  - `ZEBU_ROOT` (points to `<path_to_tools>/zebu/<version>`)
  - `XILINX`
  - `PATH`
  - `LD_LIBRARY_PATH`
  - `ZEBU_DRIVER_PATH`

ZeBu System Directory

- In addition to the previous variables, a variable named `ZEBU_SYSTEM_DIR` is mandatory for runtime
- It points to the `$ZEBU_SYSTEM_DIR` system directory
  - It is specific to each ZeBu system
  - It contains calibration and diagnostics data
  - This directory needs to be regenerated each time there is a HW configuration change
  - It contains the `zCui` HW configuration file describing the HW configuration to the ZeBu compiler flow

System Utilities `zInstall`

- `zInstall` performs the following operations:
  - Compiles the `zKernel` loadable module
  - Loads `zKernel` into the Linux kernel
  - Initializes the PCIe board
  - Launches the `zKernel` as a daemon
- Needs to be run on each PC connected to a ZeBu system
- To run this command, you need to be logged as `root` on the Linux PC which hosts the ZeBu-Server unit:

```
<path_to_tools>/zebu/<version>/bin/zInstall
```

- Never run this command if a design is already running on the ZeBu system
  - It may hang the PC
- No other ZeBu runtime commands will work if `zInstall` was not launched
- You can launch several `zInstall` commands, for each version you plan to use
- You must launch `zInstall` if:
  - The PC is rebooted
  - `zKernel` process was killed
  - You use a new version of the tools
- To launch it automatically at PC boots, add the following line to the `/etc/inittab` file:

```
zK:2345:wait:<path_to_tools>/zebu/<version>/bin/zInstall
```

ZeBu system directory generation

- Prepare a `setup.zini` file
  - Copy template from `$ZEBU_ROOT/etc/configurations/setup_template.zini`
  - Edit the file and modify the following line:
    `$ZEBU_SYSTEM_DIR = "my_system_dir";`
  - If you have installed in the ZeBu software installation tree only one diagnostics patch, it will be used automatically
  - If you have multiple diagnostics patches (multiple Zebu system configurations) you must edit the following line:
    `$patchFile = "/zebu/release/etc/firmwares/ZSE/dt/V2.0/C00-00-512-161-161-161-161-161.diag";`

ZeBu system directory generation

- Generate the ZeBu system directory:
  - From any PC connected to Unit 0 (just one PC)
  - Launch:
    `zInitSystem setup.zini`
  - This will run diagnostics, calibration and will initialize the entire ZeBu-Server system
- `$ZEBU_SYSTEM_DIR/config/zse_configuration.tcl` file will contain the hardware target configuration file for `zCui`

ZeBu system initialization

- You must initialize the ZeBu-Server system with zUtils each time a power OFF/ON occurred (with all units ON and no configuration change).
- Run these commands from any PC connected to **Unit 0**:

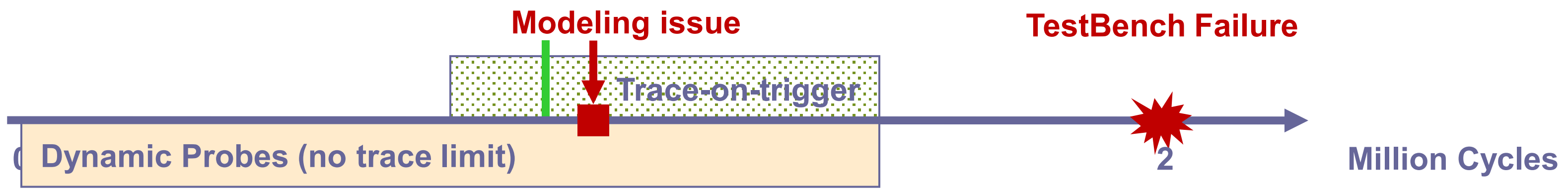
```
$ export ZEBU_SYSTEM_DIR=<my_system_dir>
$ zUtils -initSystem
```

Agenda

- Overview
  - Company overview
  - ZeBu-Server hardware overview
  - ZeBu-Server software overview
  - Smart Debug methodology

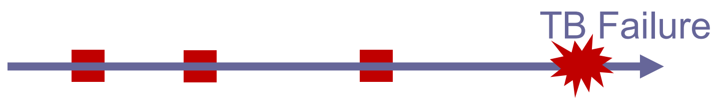
Debugging Bring-up/Initialization Bugs

- Typically memory/clocking/reset issues
  - Often fail just after reset
- Debug using “Traditional” Emulation/Simulation techniques
  - Logic Analyzer Triggers and Trace Memory
  - Dynamic Probes from Trigger, or Time 0

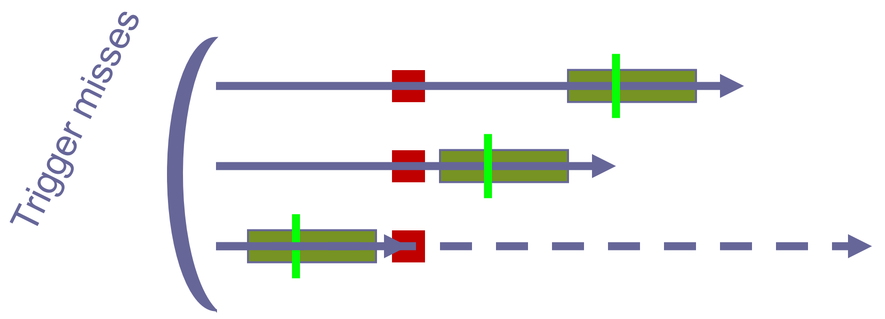


Limitations to the "Traditional" Techniques

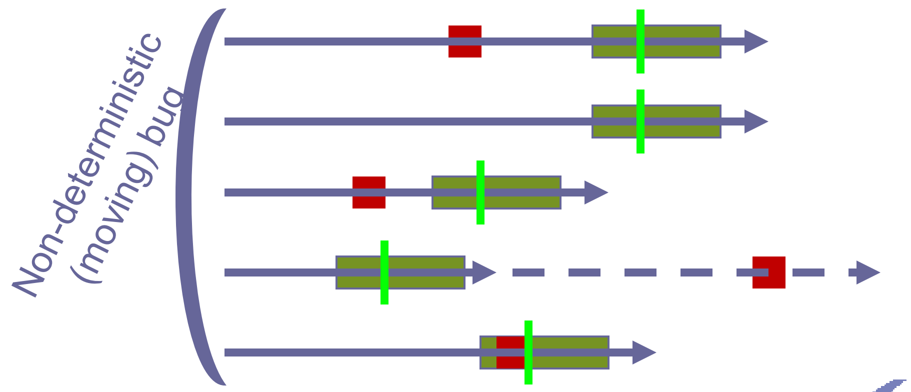
- When a self-checking test fails, the exact location of the failure may be unknown



- Using trace memory limits the number of signals & cycles that can be captured
  - Accurate triggering is critical to capture the real issue within the trace window

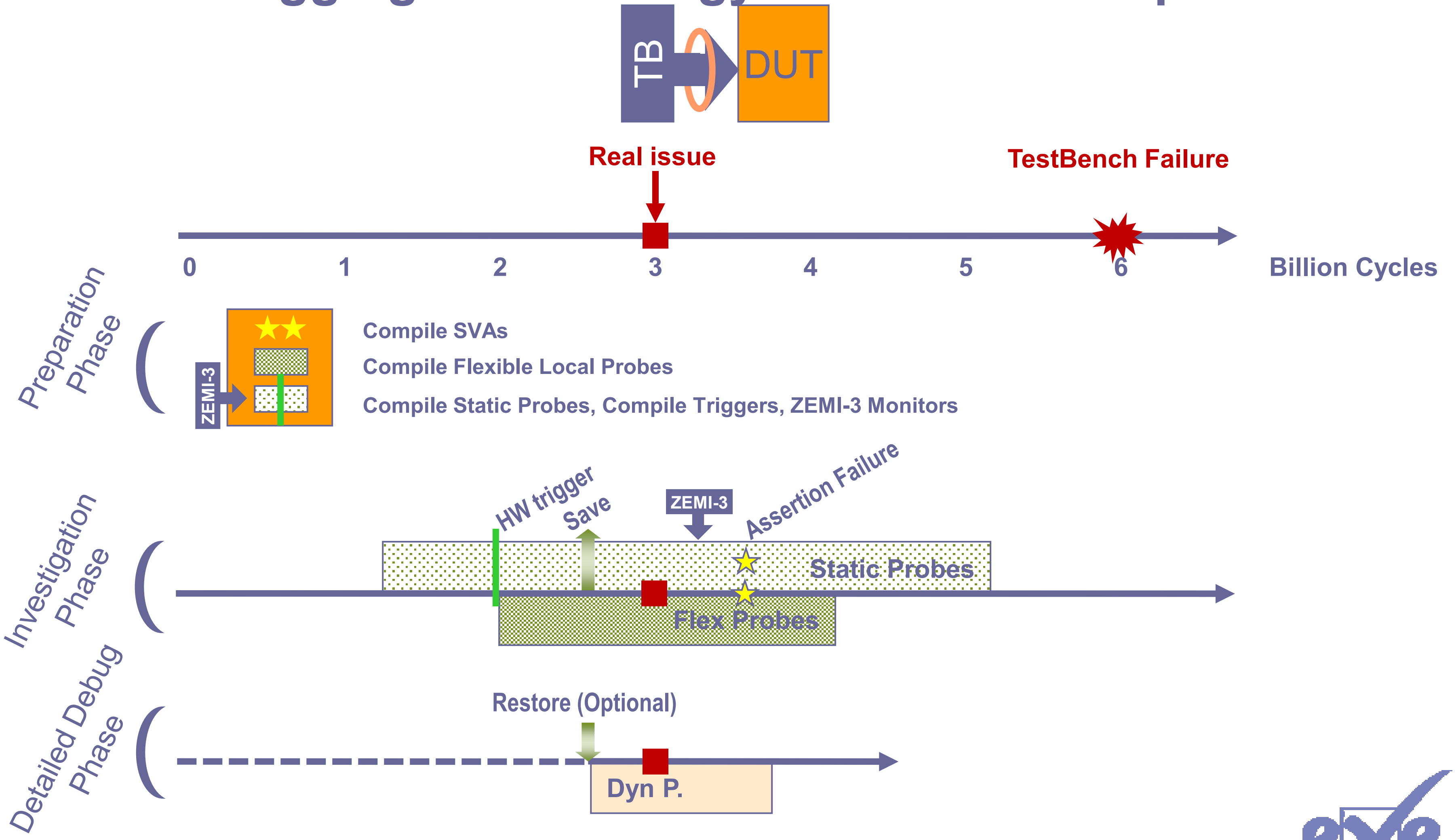


- Traditional In-circuit Emulation (ICE) test environments may not be deterministic
  - Same bug might not occur at the same point in subsequent runs, if at all



ZeBu's "Smart Debug" Methodology

- Debugging methodology based on three phases



“Smart Debug” Toolkit

- In a ZeBu system, by default, only the top level ports of the design are visible
- To gain accessibility to the design’s internal signals one of the following tools must be used:
  - Static probes
  - Dynamic probes
  - Flexible local probes
  - Simulated combinational signals
- In addition, an embedded logic analyzer provides the following capabilities:
  - Triggers
  - Trace memory

Static Probes

- Underlying mechanism
  - Signals are routed to the top of the design so they become visible
- Advantages
  - Very high speed
- Disadvantages
  - Adding a probe requires recompilation
  - Limited number of probes

Dynamic Probes

- Underlying mechanism
  - Use Xilinx internal scan path to gain access to registers
- Advantages
  - Unlimited probe number
  - Adding a probe does not require recompilation
- Disadvantages
  - Slows down the emulation
  - Only shows registers
    - A combinational signal can be read by adding a register to it
    - In this case, recompilation is required

Flexible Local Probes

- Underlying mechanism
  - Embedded tracers capture signal events and transfer waveforms to PC using high-speed system bus
- Advantages
  - High speed
  - High number of probes
- Disadvantages
  - Adding a probe requires recompilation
  - Some resources are consumed within the design FPGA

Simulated Combinational Signals

- Underlying mechanism
  - Register values are read though dynamic probes and intermediate combinational signals are re-computed by simulation on the PC
- Advantages
  - Any signal is visible without recompilation
- Disadvantages
  - Slows down the emulation
  - Not designed to handle full designs, prefers block-by-block approach
- Requires a design synthesized with `zFAST`

Probes Summary Table

Probe Type	Recompi-lation	Emulation Speed	Registers	Combina-tional	Controlla-bility	Number Limit	Testbench Accessible	Waveform Generation	Requires zFAST
Static	✓	10MHz	✓	✓		4K	✓	✓	
Dynamic		kHz	✓	Recompi-lation	✓	no	✓	✓	
Flexible	✓	MHz	✓	✓	✓	30K / FPGA		✓	
Simulated		kHz	✓	✓	✓	no	✓	✓	✓

Logic Analyzer Triggers

- A trigger can detect a Boolean condition and raises when the condition becomes true
- 16 triggers can be defined
- A trigger can stop the clocks or start the trace memory capture, it can be read from a C/C++ testbench
- Two types of triggers:
  - Static trigger: complex Boolean condition which can be specified at compilation time
  - Dynamic trigger: simple bus values which can be specified at runtime; bus is specified at compilation time
- Can be controlled from a `zRun` or C/C++ testbench

Logic Analyzer
Trace memory

- An embedded system memory is used to capture signals
- Can capture signals (static probes) at very high speeds (up to 60MHz)
- List of probed signals must be defined at compilation time
- Can be controlled by `zRun` or C/C++ testbench
- Capture can be controlled by a trigger
- A pre-trigger ratio can be defined to reserve a part of the memory for pre-trigger waveform