Using transactors

ZeBu-Server Training

*THE **FASTEST** VERIFICATION*

Agenda

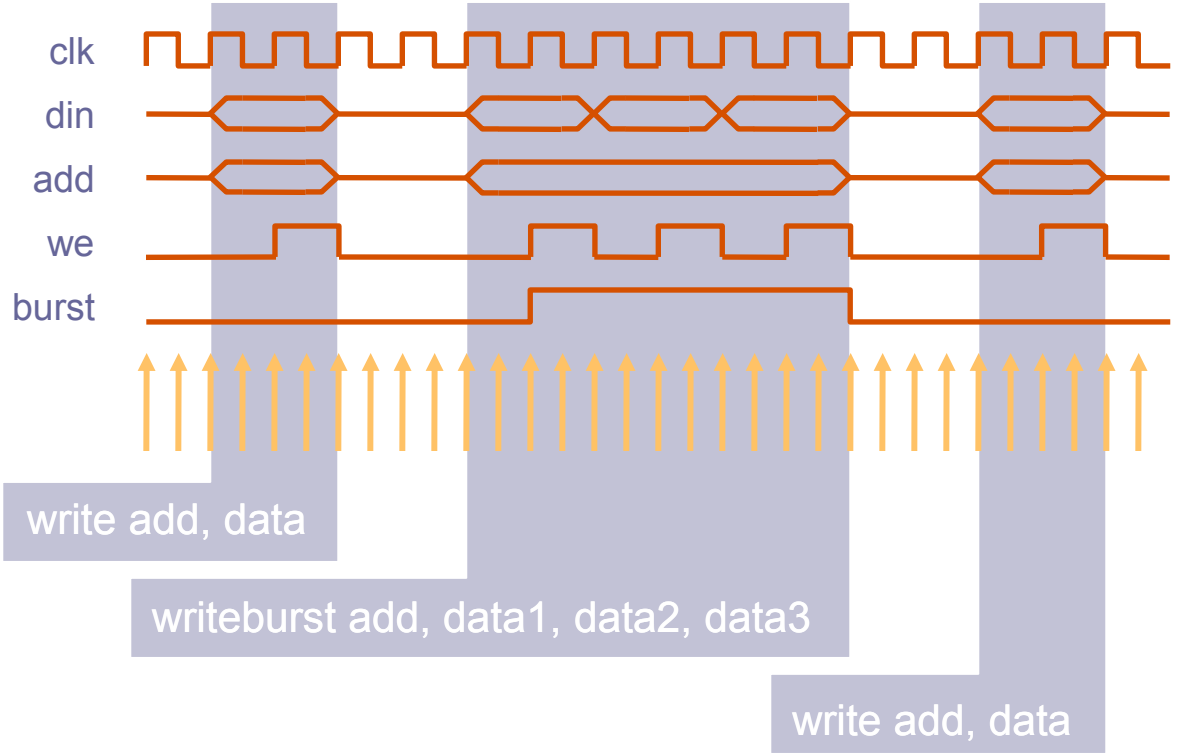
- Introduction to transactional emulation
- Transactors from EVE Vertical Solutions
- Case studies
- Compilation

Introduction
Benefits

- Instead of exchanging low-level data at bit/cycle level, software and hardware exchange high-level messages
- Only essential information is exchanged, emulation speed can be much faster than traditional emulation

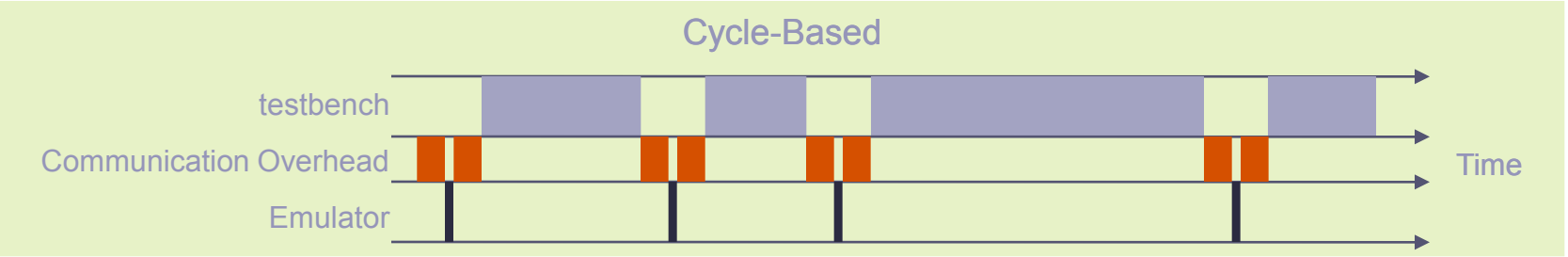
Introduction
Example: memory write burst

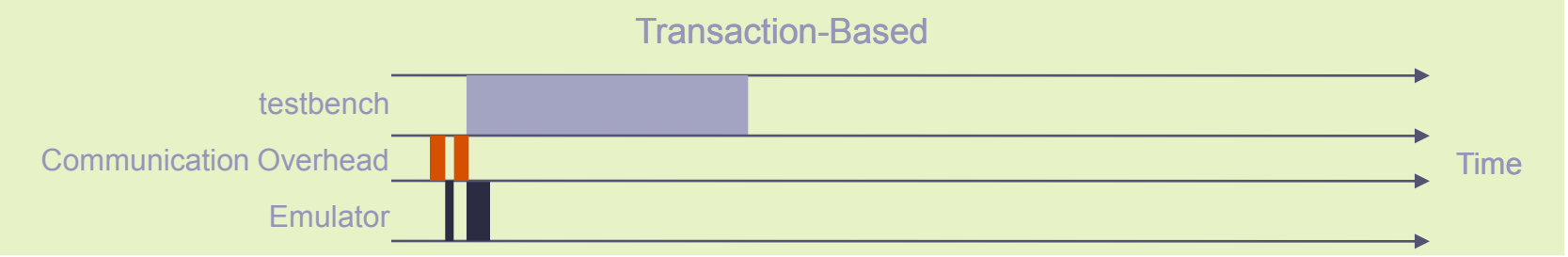
- In non-transactional emulation an exchange of information would occur at each signal change or each cycle
- In transactional emulation, exchange of information occurs only when needed



Introduction
Speed gain

- In cycle-based interfaces, the typical sequence of operations is:
  - Force DUT input signals
  - Run one cycle
  - Read DUT output signals
- Testbench and DUT are synchronized on a cycle basis, and communication overhead occurs at each and every cycle





- With Transaction-Based Emulation, synchronization is done only when required
- Testbench and DUT can run in parallel and transactions can be queued
- The speed improvement over cycle-based can reach several tenths of MHz

Introduction
Benefits

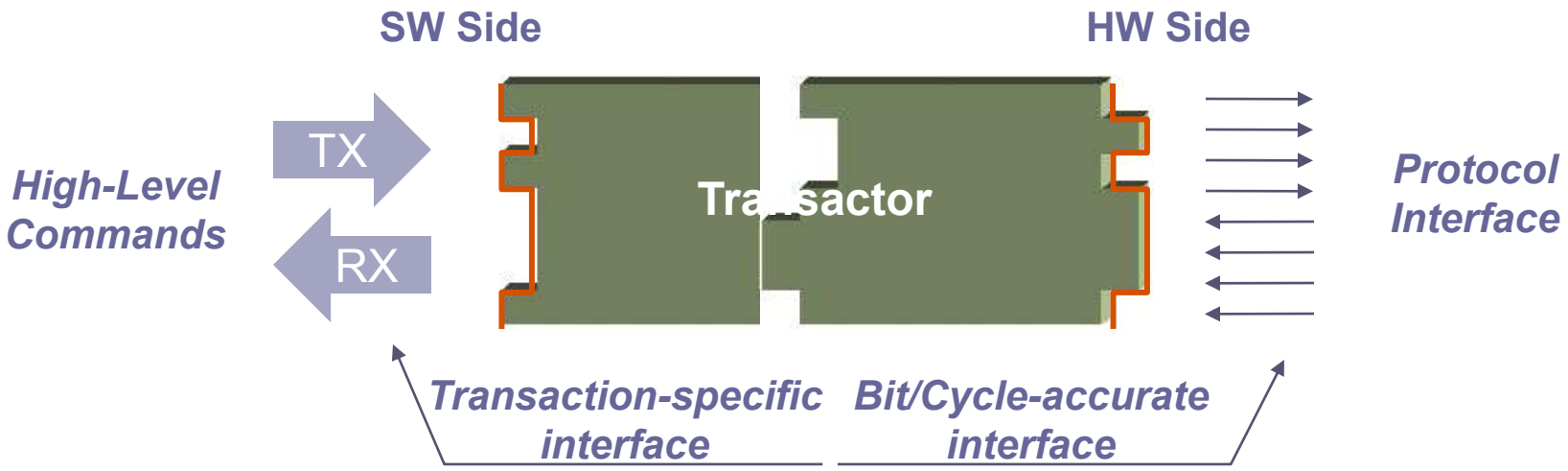
- High-level specific API knowledge is sufficient
- Initial investment to develop a transactor, but lower investment for further testbench development
- High-level reusability possible: transactors are written once and used many times
- Better speed:
  - Activity scheduled on a transaction basis, only when events happen
  - Parallel run of testbench and emulator (especially for data streaming application)

Introduction
testbench Abstraction Levels

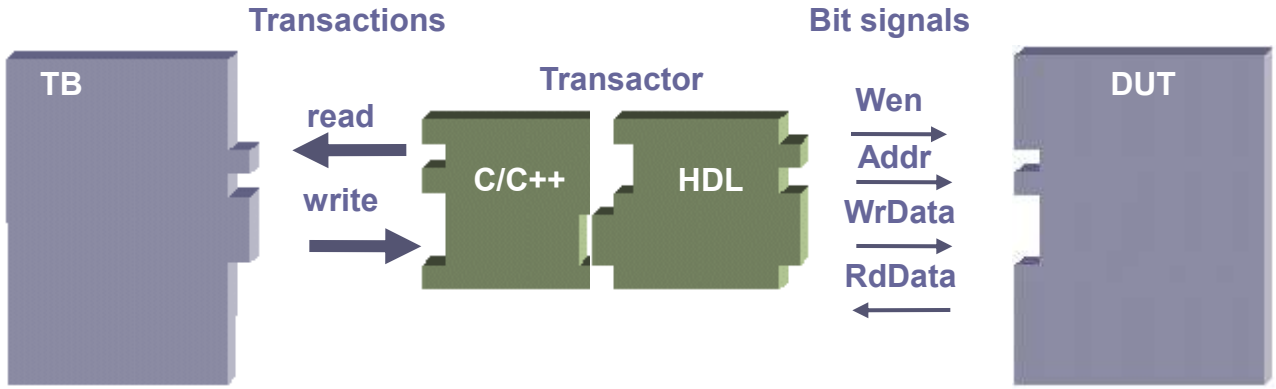
- Events
  - Involves individual signals, pins or nets. Transitions of these signals are generated or monitored
- Cycles
  - Focuses on behavior at each chip, bus or interface clock cycle. Creates or monitors the activity at each clock edge
- Transactions
  - A complete interchange of data or status: a transaction may span several clock cycles, which involves many signals and events
  - Examples: a bus read or write cycle, a bus burst, a DMA transfer, an interrupt, a packet or cell or frame of data

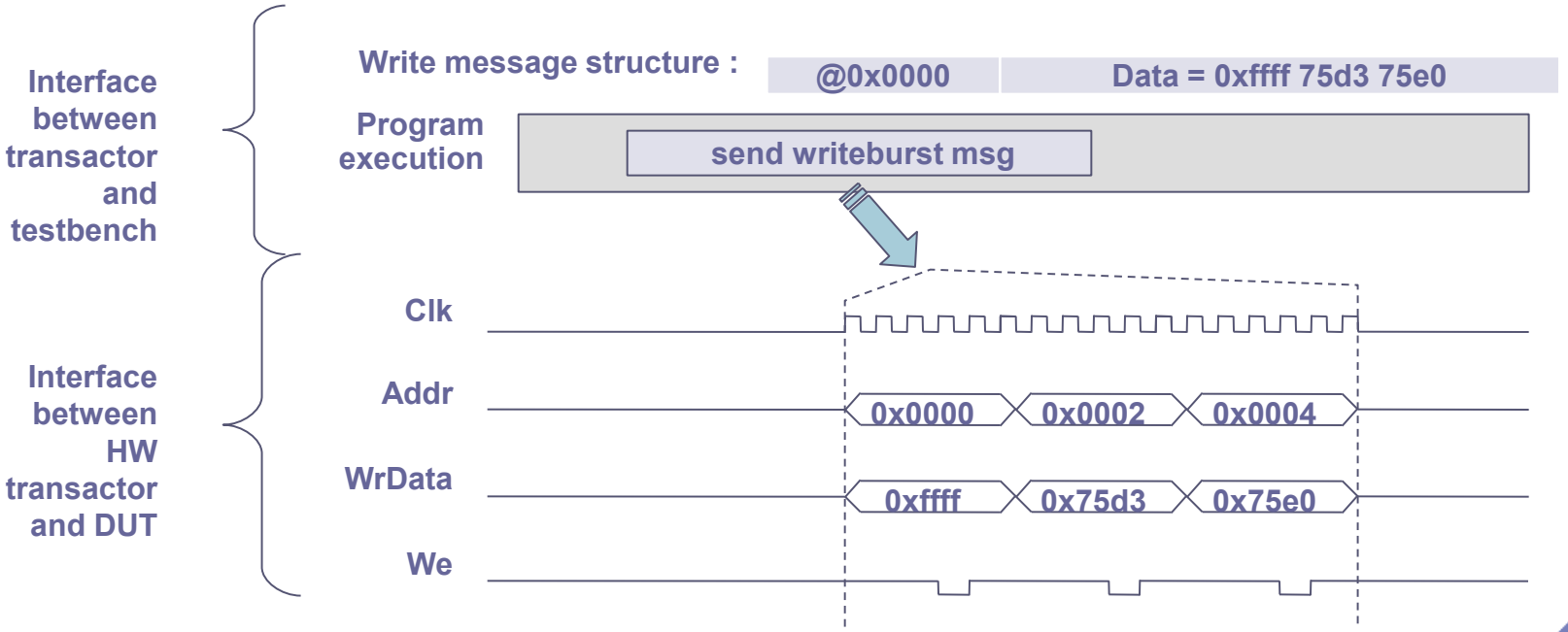
Introduction
Basics

- A transactor provides the abstraction for a given protocol
- A transactor converts the data between two interfaces:
  - On the software side, it provides high-level commands to perform protocol-specific actions
  - On the hardware side, it matches the specific protocol interface at cycle/bit level



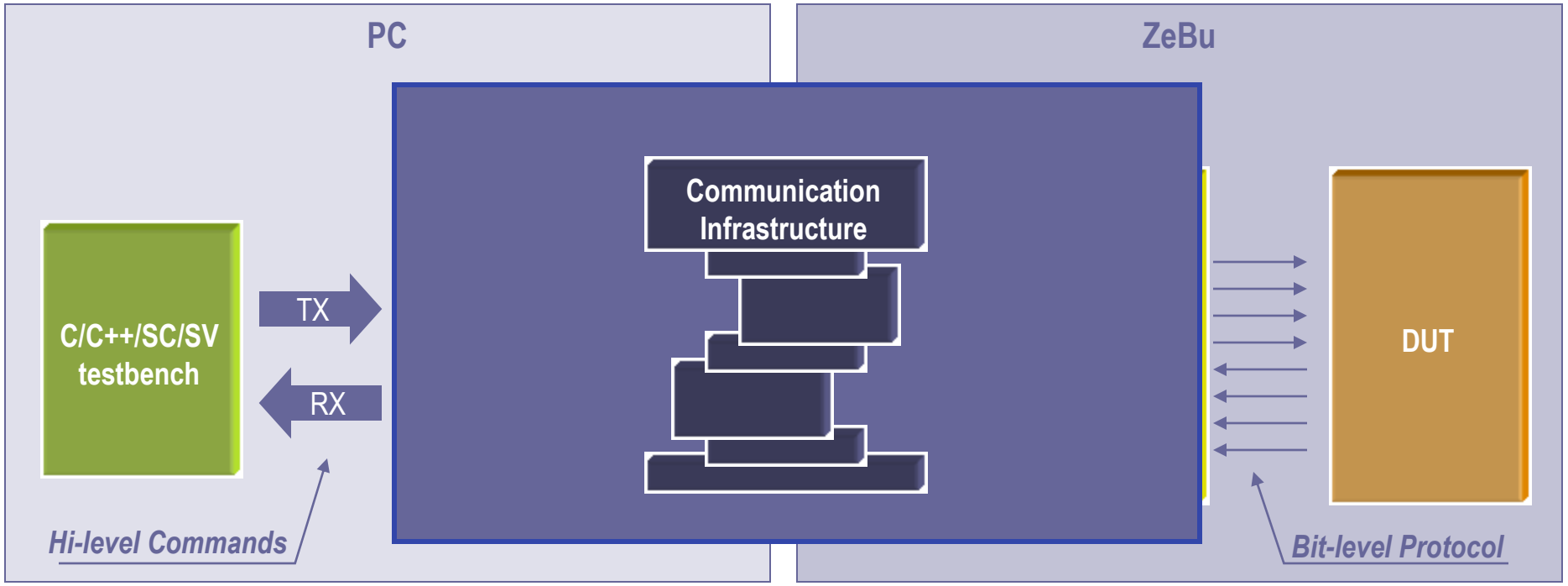
Introduction
Example: Memory Write at Transaction Level





Introduction
ZeBu transactors

- A transactor consists of two parts:
  - Front-end: C/C++/SystemC/SystemVerilog model to send/receive high-level commands (transactions) to/from the testbench - Not compute-intensive
  - Back-end: HDL BFM model to convert high-level commands into bit-level protocol - Compute-intensive



Introduction
Example : Simple Memory Transactor

- Goal: implement a memory model in SW, because address space exceeds hardware capacity (for instance, 8GB memory requirement)
- Transactor watches memory bus of design and services memory accesses
  - DUT Signals: RD/WR, ADDR, DATA_IN, DATA_OUT, CLK
- Algorithm:
  - When a READ is detected on the bus:
    - Send the ADDR in a message to the SW to query for the value at that address
    - Freeze the design clock so it appears to the design as if the response would happen in one clock cycle
    - Wait for a message back from the SW with the data
    - Put the data on the DATA_OUT bus
    - Unfreeze the design clock
  - When a WRITE is detected on the bus:
    - Send the ADDR/DATA pair to the SW
- Requirements:
  - At least one message port from HW to SW (to send ADDR on a READ and ADDR/DATA on a WRITE)
  - One message port from SW to HW (to return DATA on a READ)
  - A mechanism to stop the design clocks

Introduction
Message encoding

- **Command Port Format:**
  - **Read (ADDR):**
  - **Write (ADDR,DATA):**

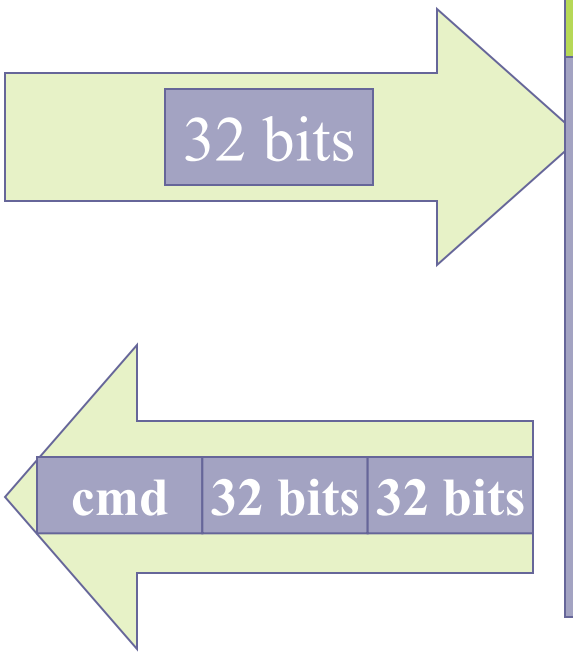
cmd	32 bits	32 bits
1	ADDR	n/a
0	ADDR	DATA

- **Reply Data Port:**

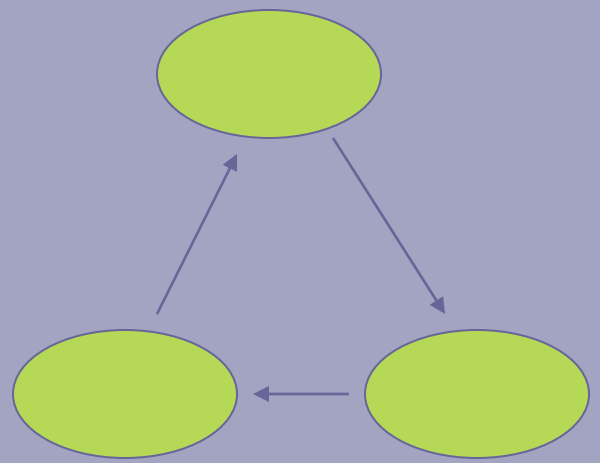
DATA

SW Slave Testbench

```
waitforMessage
case Read:
  // look up date
  addr = msg.field(0);
  sendMessage(mem[addr])
case Write:
  // update memory
  addr = msg.field(0);
  data = msg.field(1);
  mem[addr] = data;
```

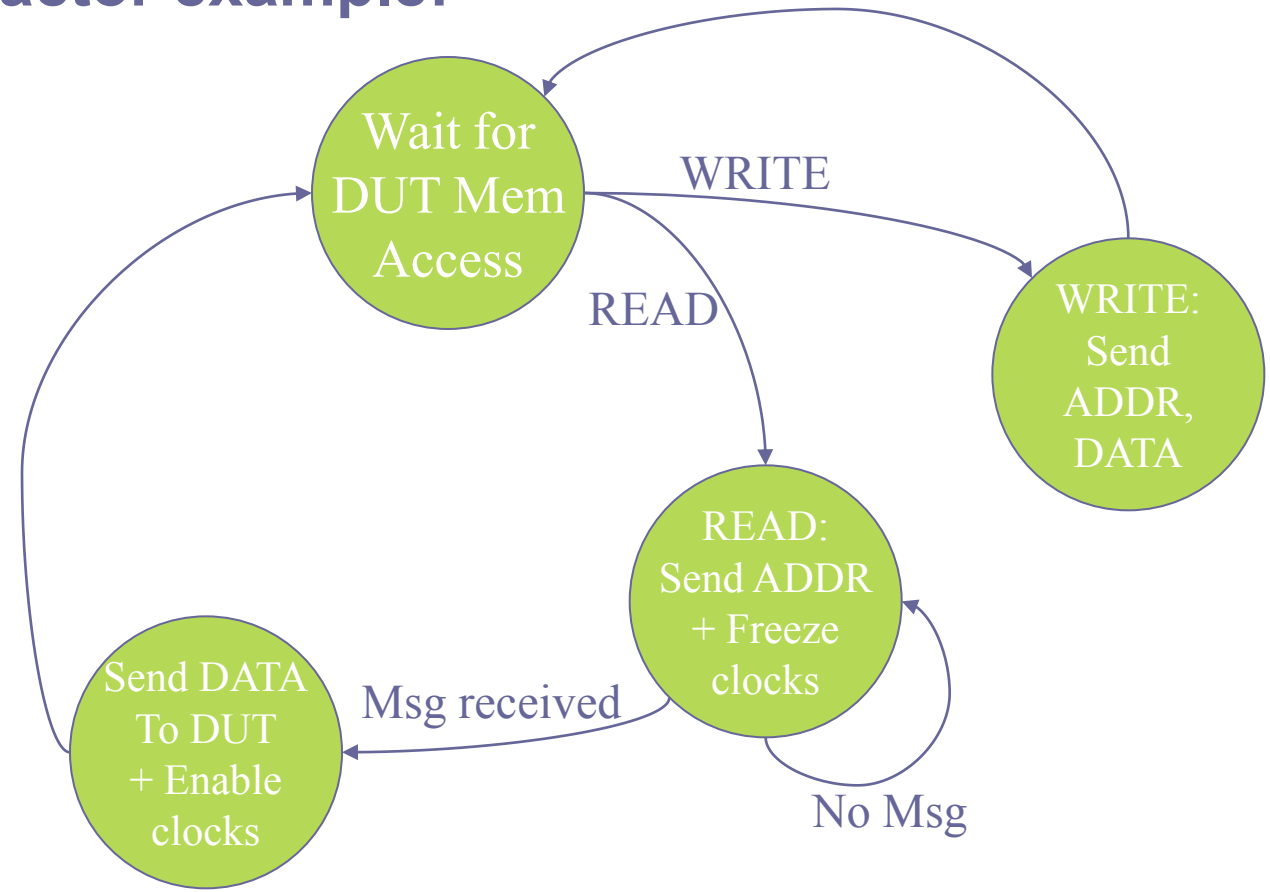


HW BFM



Introduction
Simplified state machine

- **High-Level state machine that implements the memory transactor example:**



Introduction
How to obtain transactors?

- Design your own transactor
  - zcei transactor
    - Offers best level of performance (RTL code)
  - ZEMI-3 transactor
    - Offers higher level of abstraction (behavioral code)
- Acquire an off-the-shelf transactor
  - From third-party vendors
  - From EVE Vertical Solutions catalog

Agenda

- Introduction to transactional emulation
- Transactors from EVE Vertical Solutions
- Case studies
- Compilation

EVE Vertical Solutions transactors Catalog

- Large (Sparse) SRAM Memory
- RS232 (UART)
- Keypad
- Input/Output Stream
- LCD RGB TFT Display
- Digital Video/ YUV 420 Display
- I2S Digital Audio (Master, Slave)
- JTAG (Xtensa from Tensilica)
- JTAG (SeeCode Debugger from Virage Logic)
- JTAG TAP Controller
- 10/100/1G ETHERNET MII/GMII
- PCI Express Gen1 (1/4/8/16 lanes)
- MMC/SD Card Host
- MMC/SD Device Mode
- FireWire (Isochronous only)
- Digital Video Input stream (YUV,656,DVB)
- USB 2.0: Host
- USB 2.0: Device
- AMBA AHB Master
- AMBA AXI Master
- AMBA AXI Slave
- I2C Master
- I2C Slave
- Ethernet Transactor SW bridge
- HDMI Sink (Video Only)
- CoWare Platform Architect PV_Adapter (ESL Emulation)
- ETHERNET LAN Controller
- PCI Express Gen2 (1/4VC , 1/4/8/16 lanes)
- Digital Video : YUV/656/RGB: Multistandard

EVE Vertical Solutions transactors
Transactor package

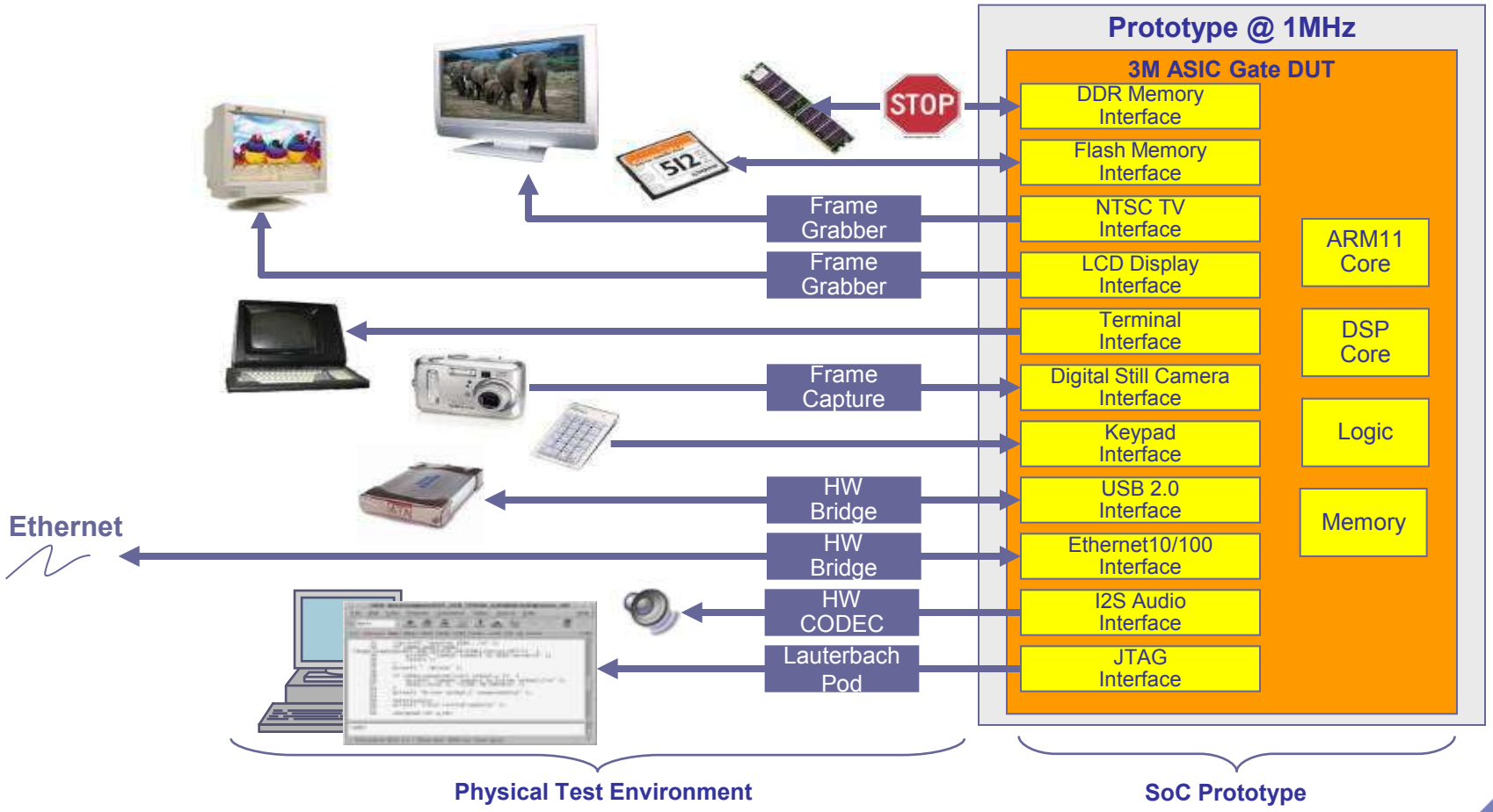
- Hardware part of the transactor
  - an EDIF netlist
  - a simulation model
- Software part of the transactor
  - a dynamic library (.so)
  - a header file (.h)
- Link between the hardware and the software parts
  - an `.install` file
- Documentation
- Examples

Agenda

- Introduction to transactional emulation
- Transactors from EVE Vertical Solutions
- Case studies
- Compilation

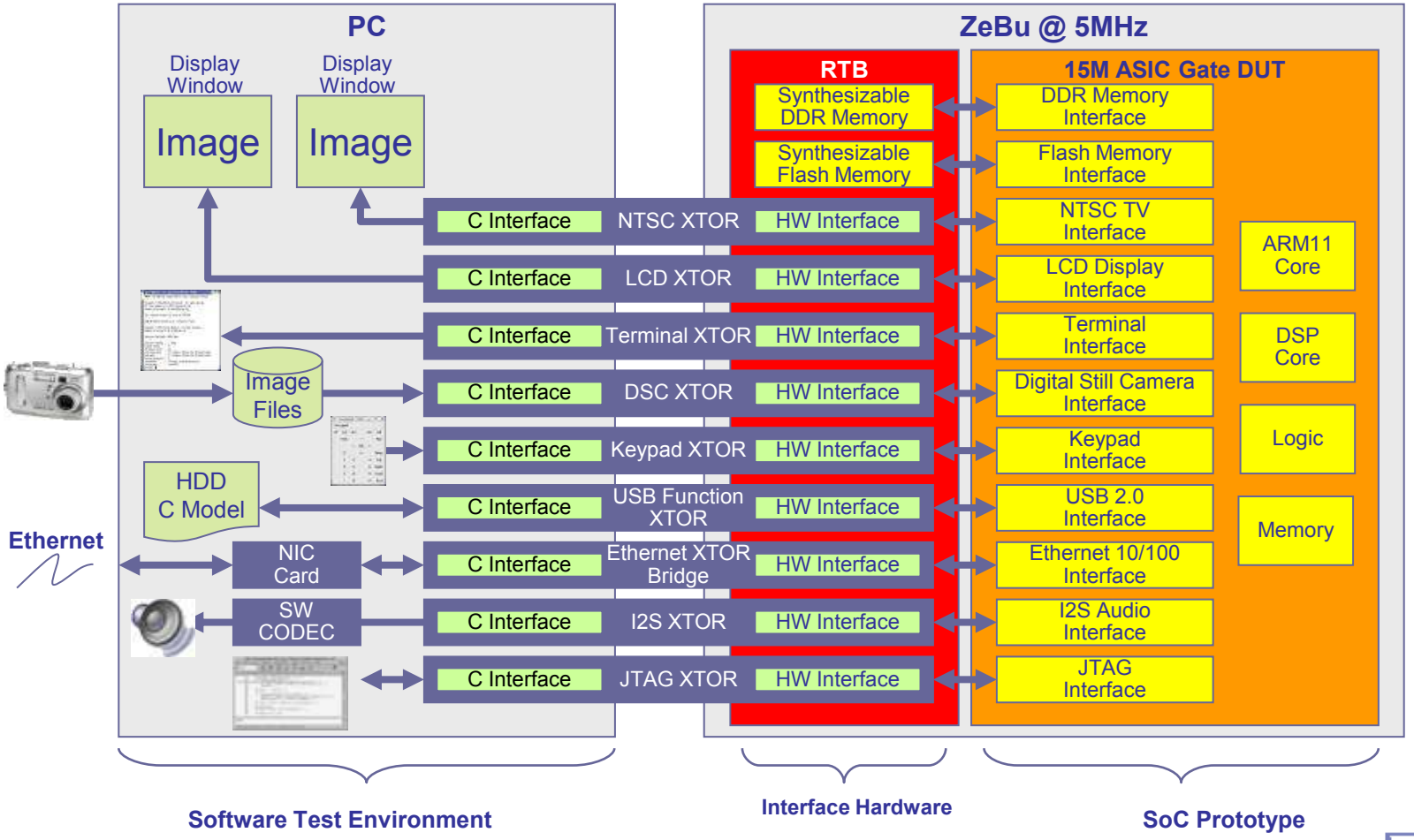
Case studies
Prototyping of a wireless processor

Physical In-Circuit Prototype



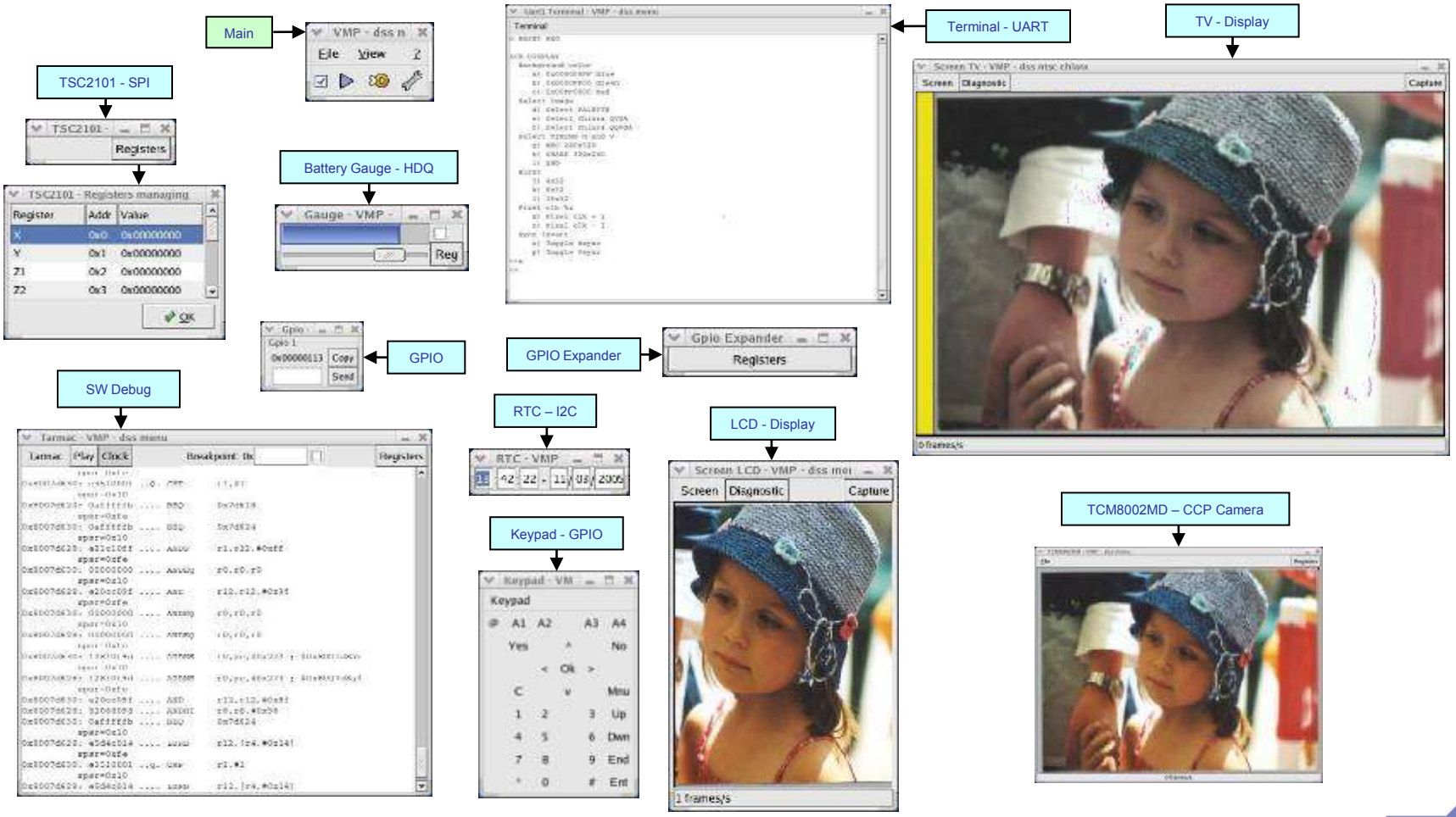
Case studies
Prototyping of a wireless processor

Virtual Multimedia Platform



Case studies
Prototyping of a wireless processor

Sample Display

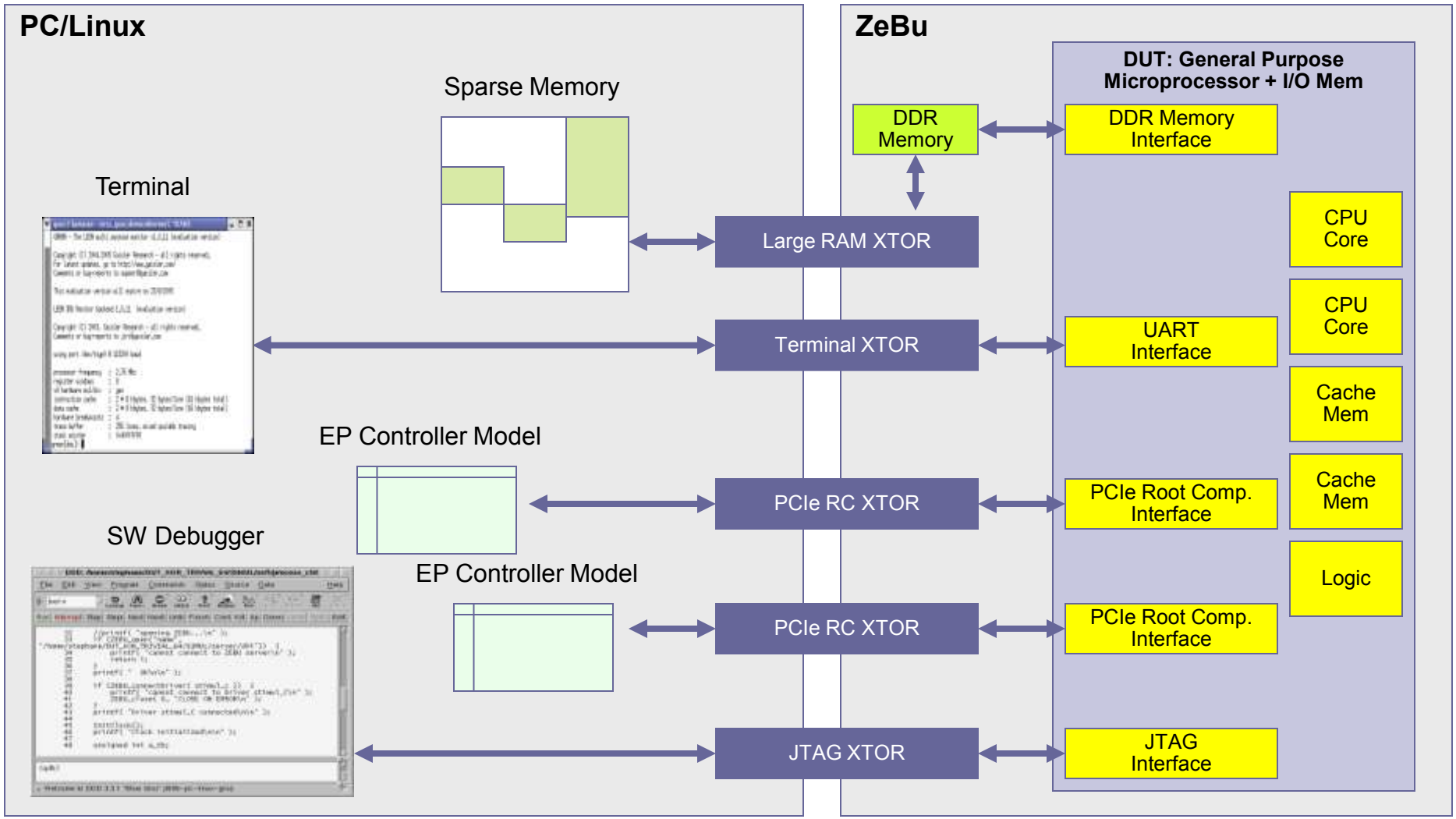


Case studies
Block-level verification of a FPU

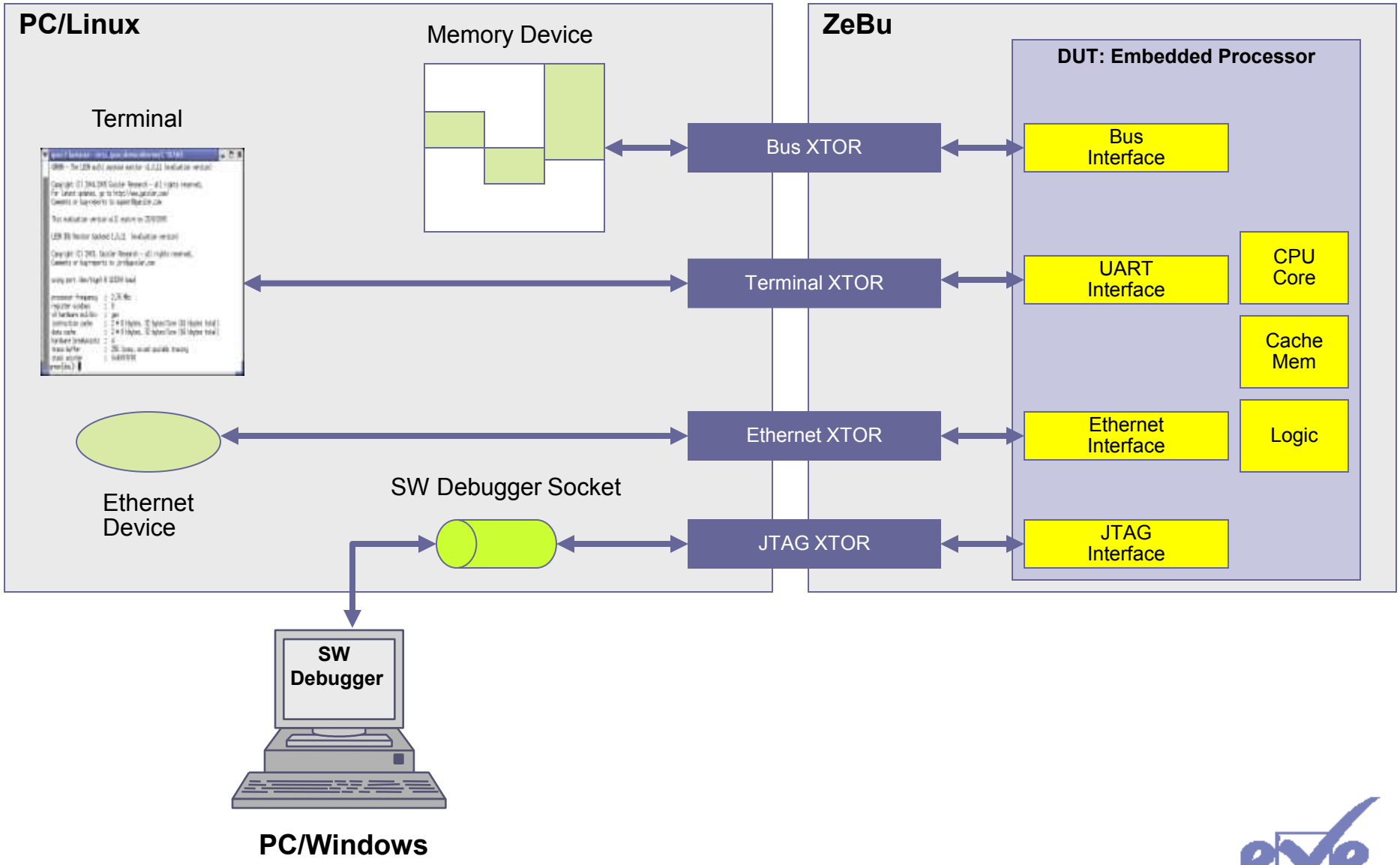
- A processor company is verifying its pipelined Floating Point Unit using custom transactors streaming opcodes and operands to the DUT at every cycle, reading back the results and comparing them to a C reference model

- Running for several days at 2 MHz, they process many billions of operations with random values to unearth potentially deeply seated bugs

Case studies
Verifying a general-purpose microprocessor



Case Studies
Verifying an embedded processor



Agenda

- Introduction to transactional emulation
- Transactors from EVE Vertical Solutions
- Case studies
- Compilation

Compiling transactors for ZeBu
Writing the DVE file

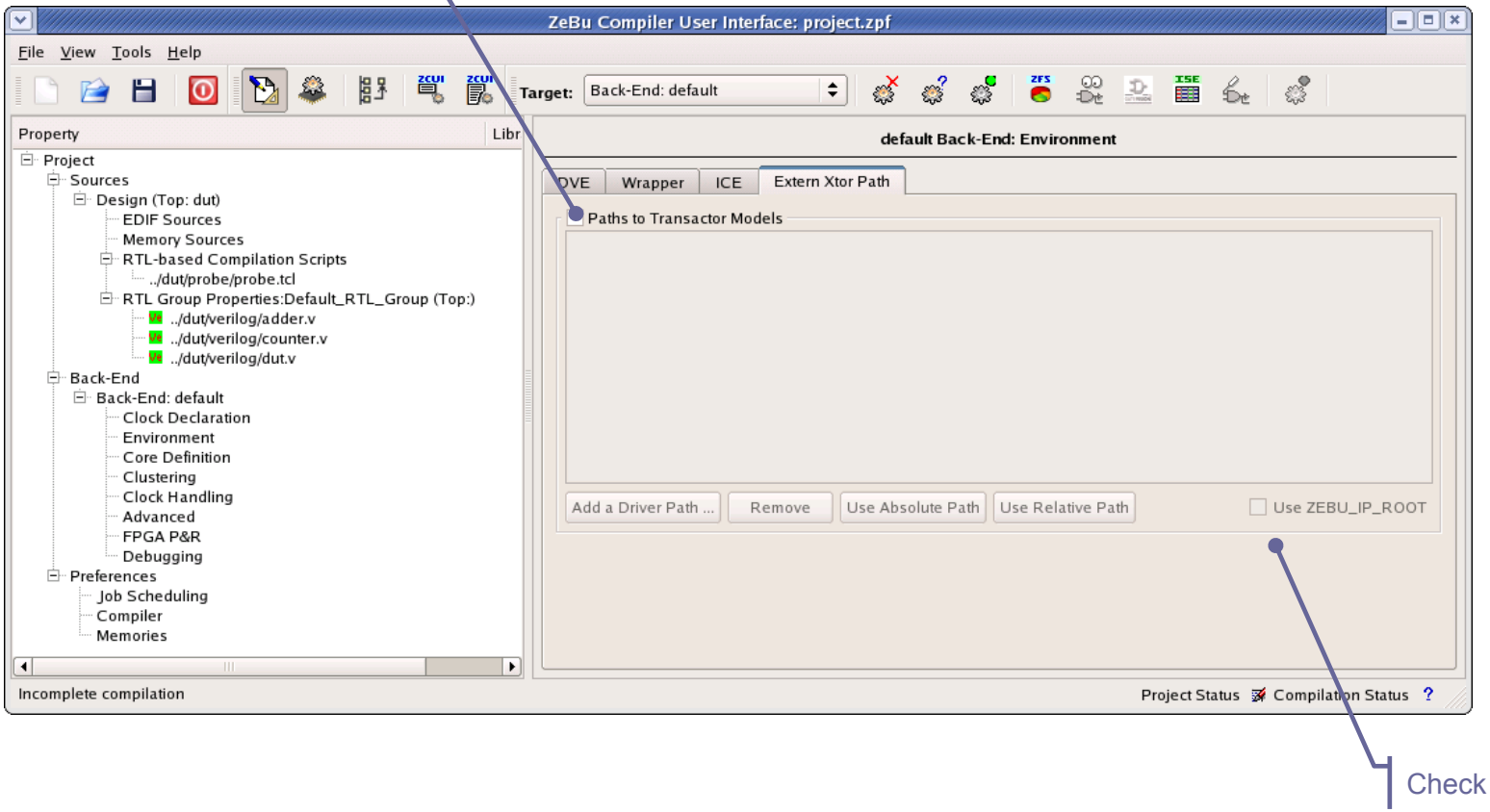
- The Design Verification Environment file
  - connects transactors to the DUT
  - attaches clocks to transactors

```
jtag_driver jtag_driver_0 (
        .tms( dut_tms ),
        .tdi( dut_tdi ),
        .tdo( dut_tdo )
);
defparam jtag_driver_0.cclock = dut_tck;
zceiClockPort clock_ClockPort0 (
    .cclock( dut_tck )
);
```

Compiling transactors for ZeBu
zCui

- The Vertical Solutions transactors are fully integrated with zCui
- Just set the ZEBU_IP_ROOT environment variable point to the location where EVE transactors are installed
- In zCui ("Environment">"Extern Xtor Path" tab), check the "Paths to Transactor Models" then the "Use ZEBU_IP_ROOT" checkboxes.
  - zCui will automatically find the correct install path

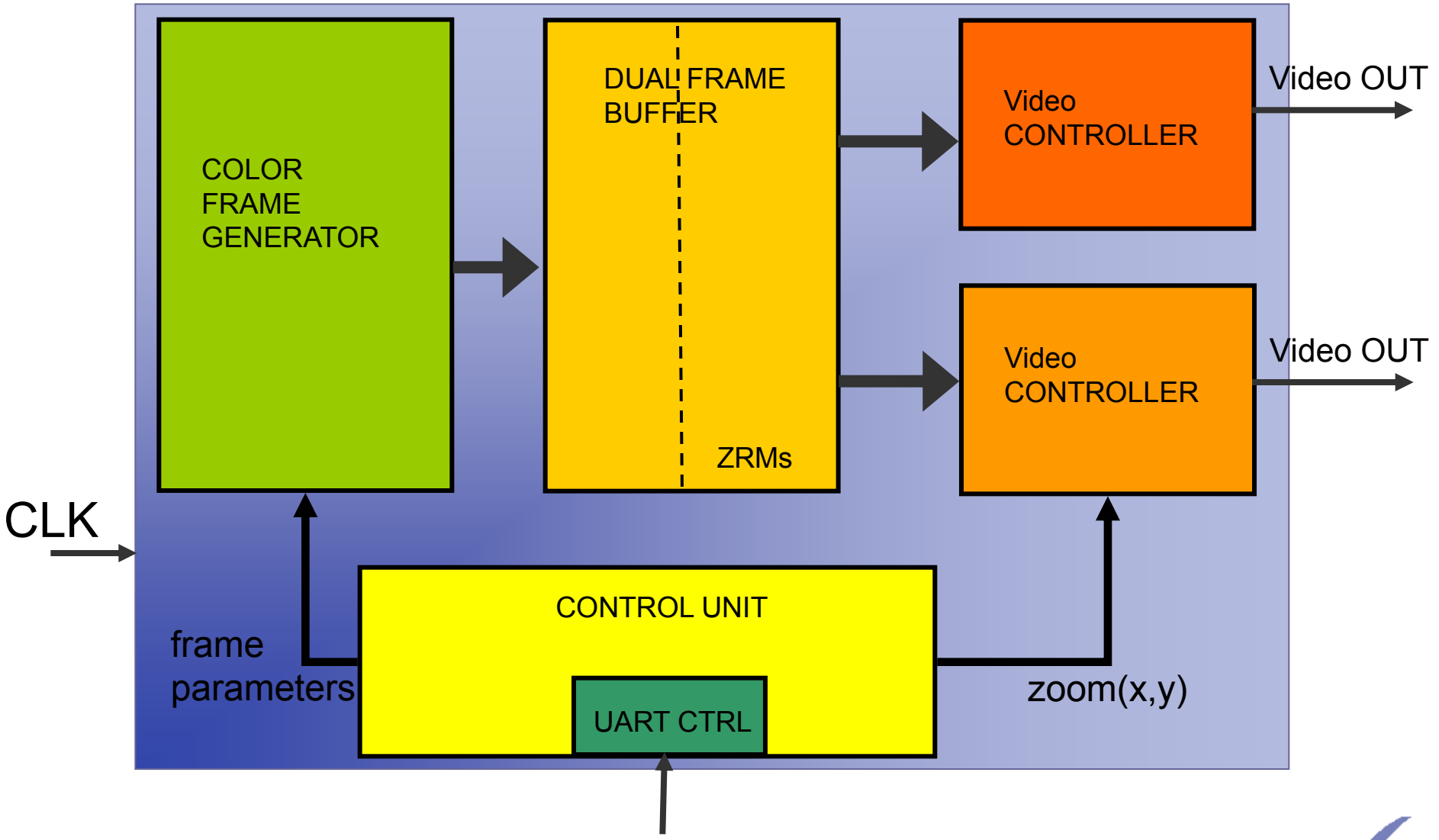
Compiling transactors for ZeBu
zCui



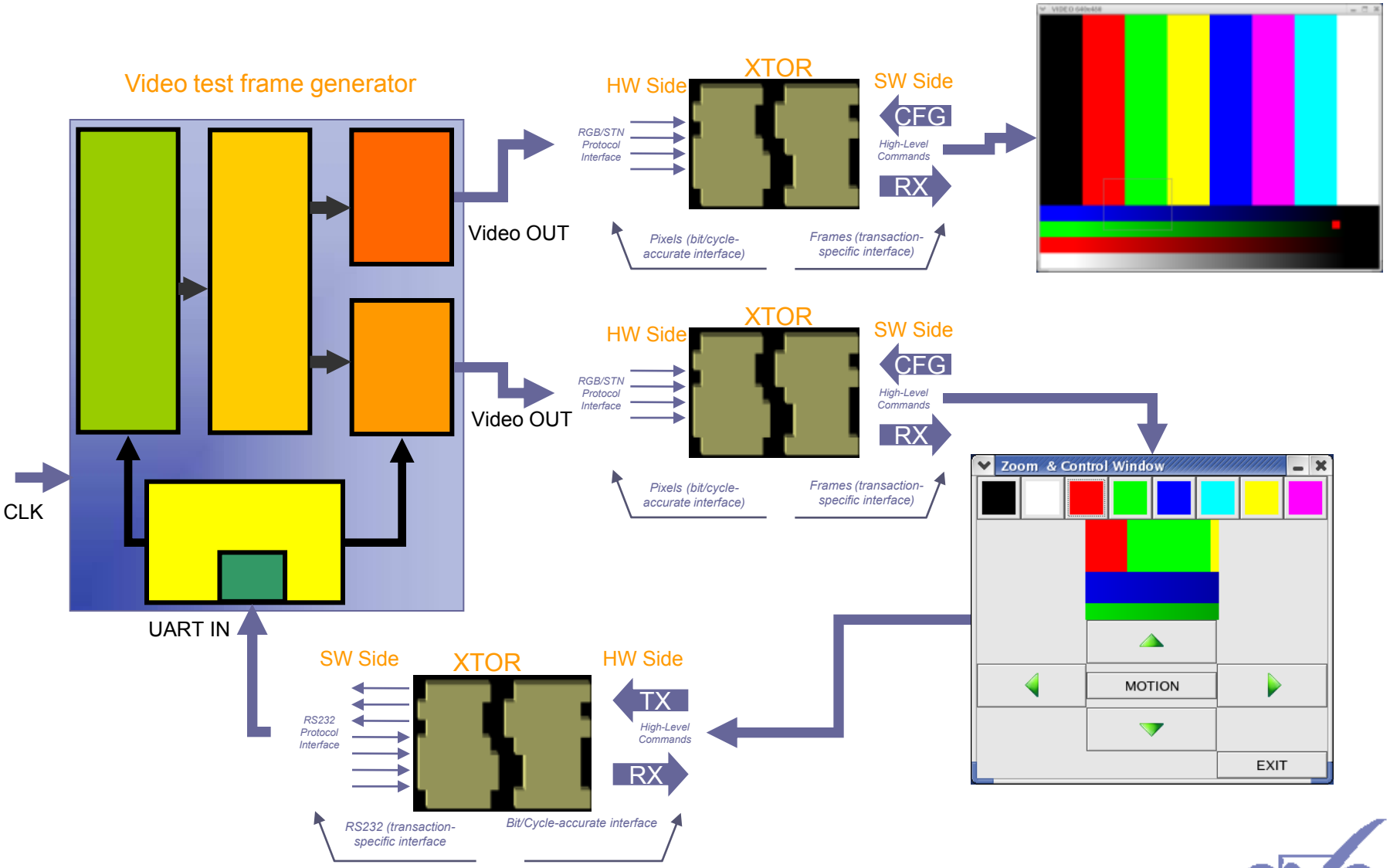
Using transactors
Lab 4

- In this lab you will learn how to use off-the-shelf transactors
- We will use a video pattern generator DUT driving 2 LCD screens and controlled by an UART
- Both LCD screens will be displayed on a virtual GTK window using the EVE video transactor
- Control characters will be sent to the DUT's UART through EVE UART transactor
- You will first compile the design
- Then you run the emulation

Using transactors
Lab 4



Using transactors
Lab 4



Using transactors
Lab 4

```
Trainee/lab4
|-- C-TRANS
|   |-- Makefile
|   `-- src
|       `-- zrm_sram_2d_2x32.tcl
|-- common_compil
|   |-- demo_lcd.dve
|   `-- dut.edf
|-- common_run
|   `-- designFeatures
`-- static
    |-- c
    |   `-- testbench.cc
    `-- interface
        `-- images
```

Using transactors
Lab 4

- Prepare a `zCui` project for compilation
  - Go into the `C-TRANS` directory
  - Open `zCui`
  - Add `../common_compil/dut.edf` as EDIF source file
  - Right-click the EDIF file and select “Extract Toplevel”
  - Set the target hardware configuration file according to your ZeBu system
  - Left-click on the “Environment” property
  - Select “User Defined” as DVE file “Generation Mode”
  - Select the file `../common_compil/demo_lcd.dve` as DVE file
  - Check the “Paths to Transactor Models” then the “Use ZEBU_IP_ROOT” checkboxes
  - Set the job scheduling preferences according to your network
  - Save the project

Using transactors
Lab 4

- Launch the compilation
- While the compilation is running have a look at:
  - the DVE file
  - the C++ testbench
- When the compilation is finished run the emulation:
  - Type:
    ```
    make run
    ```
  - Play with the graphical menu!