2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Development of a Core-Monitor for HW/SW Co-Debugging targetting Emulation Platform

Shreya Morgansgate
Dr. techn. Johannes Grinschgl
Dr. Ing. Djones Lettnin

infineon

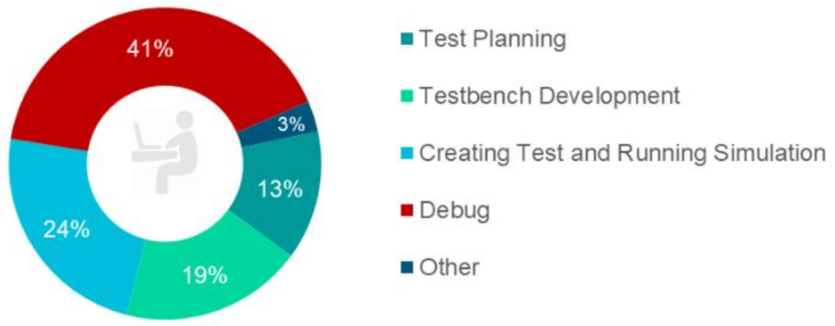
accellera SYSTEMS INITIATIVE

Outline

Outline

Motivation

- The SoC architecture that combines multiple heterogeneous cores and IPs are prevalent in a wide range of applications
- HW/SW co-debugging requires high effort:
  - Complex HW/SW system integration
  - More time spent in debugging
  - Shortens the time to market



Source: Wilson Research Group and Mentor, A Siemens Business, 2020 Functional Verification Study [1]

Outline

1. Motivation
2. Problem Statement
3. Technical Background
4. Methodology
5. Implementation
6. Results
7. Conclusion

What is a Core-Monitor?

- Core-Monitor is required to have a general HW/SW co-debugging environment for an embedded processor
- It generates a trace file that monitors the run-time behavior of the CPU
- The trace file is a human understandable log file with executed core instructions
- This trace file will be input to the debugging tools like Indago [2] for further processing

Problem Statement(1)

- There is a need to trace interactions across multiple and heterogeneous processing elements
- There is no universal and reusable solution to trace the processor state of multiple cores in a processor
- There is no reusable platform to enable switching between simulation and emulation environments for HW/SW co-debugging

Problem Statement(2)

- The Core Monitor aims to solve this problem:
  - It is standard for trace file generation in both the simulation and emulation environments
  - Reusable approach for processors with multiple cores
  - Reduces the HW/SW co-debugging time and shortens the time to market

Outline

Why Emulation?

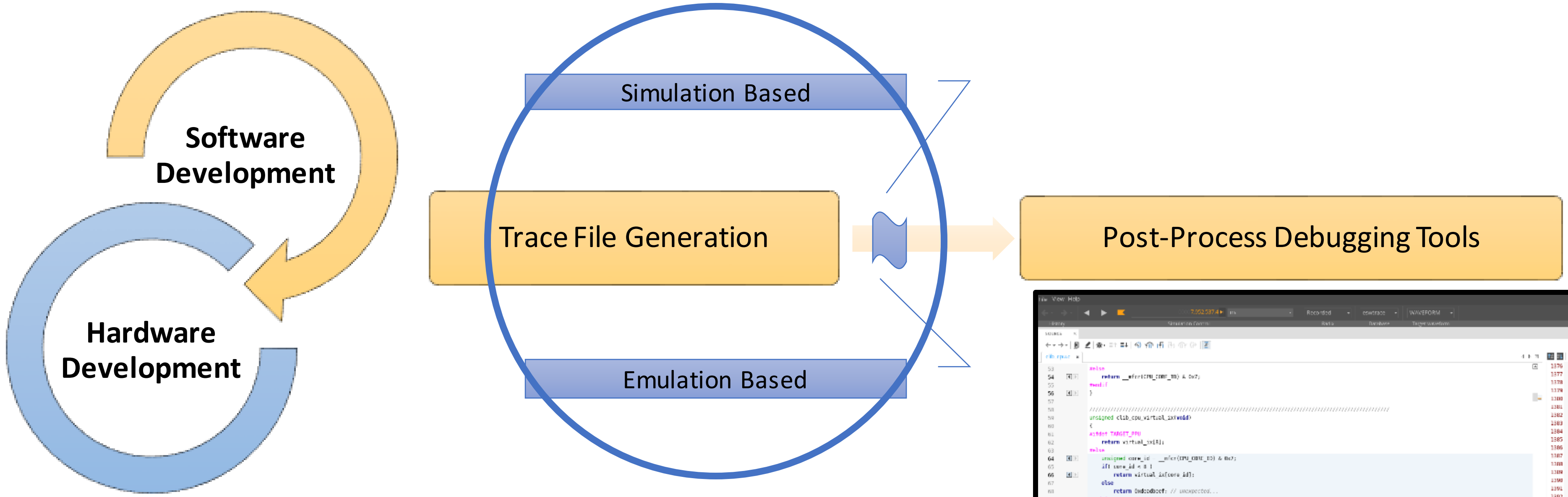
- Simulation-based techniques are unable to keep-up with today’s growing size and complexity of designs with software and embedded processor core models
- Large design projects have moved on to emulation platform for SoC integration and verification

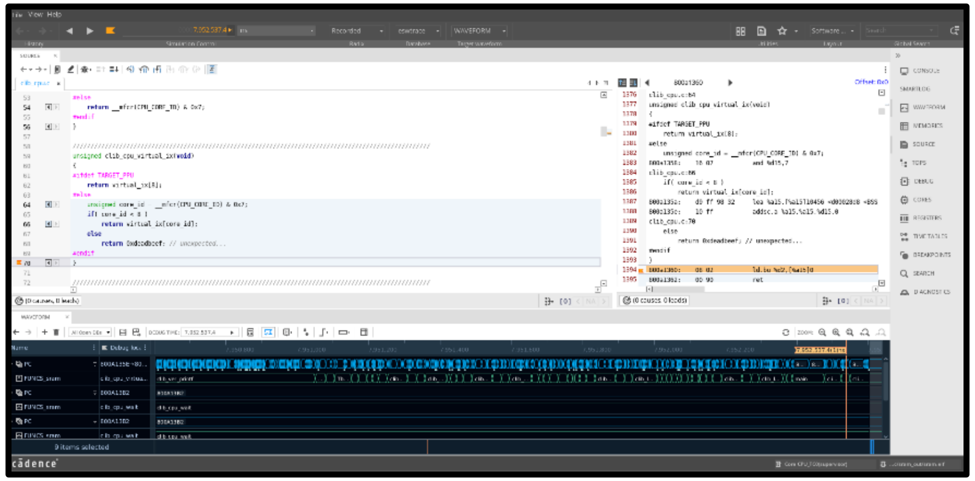
Emulation Setup

- Zebu Server [3]
  - It is a Synopsys emulation system
  - In ZeBu emulation, design files and project files are compiled using VCS simulator
- Direct Programming Interface (DPI) [4]
  - System Verilog DPI is an interface for SV to interact with other languages like C, C++, SystemC
  - Zebu supports SV-DPI import calls to the host
  - Maximizes runtime efficiency

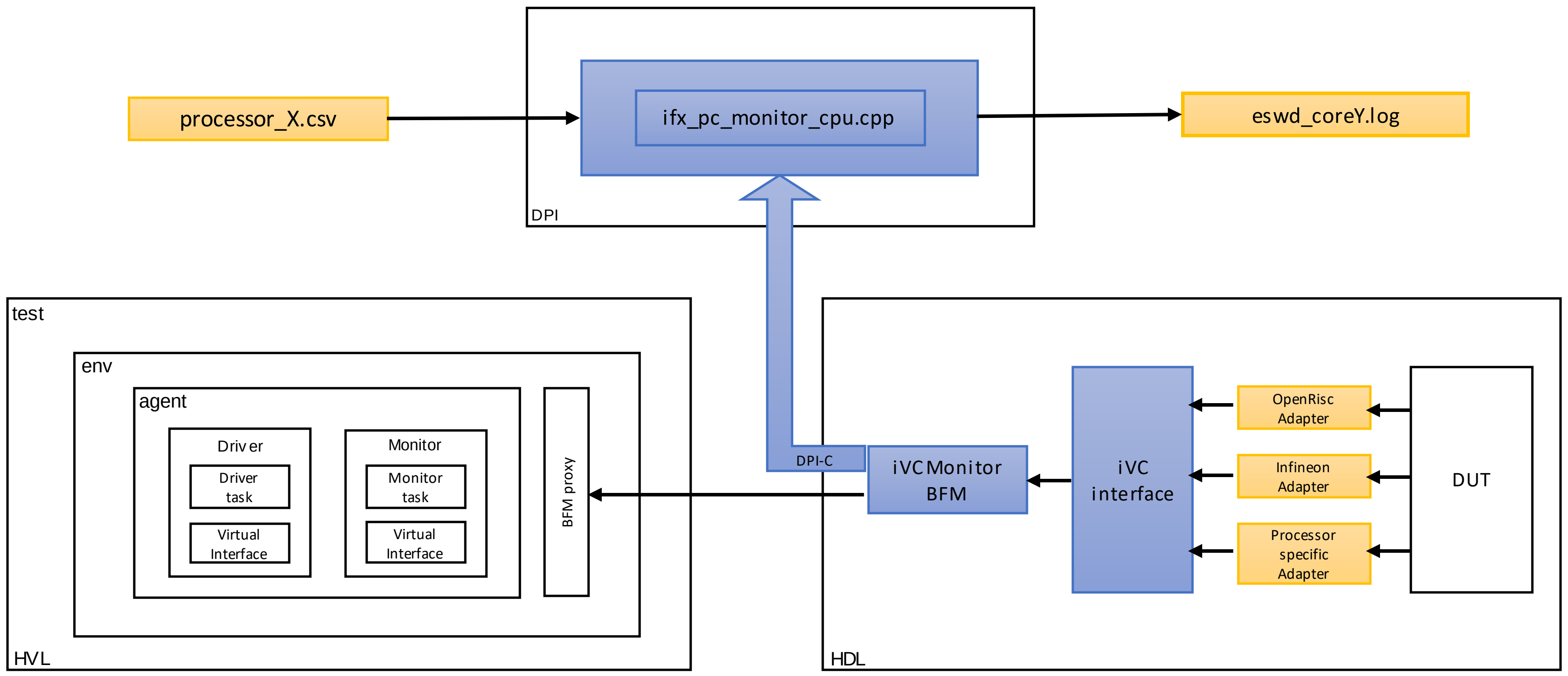
Outline

Typical System Flow



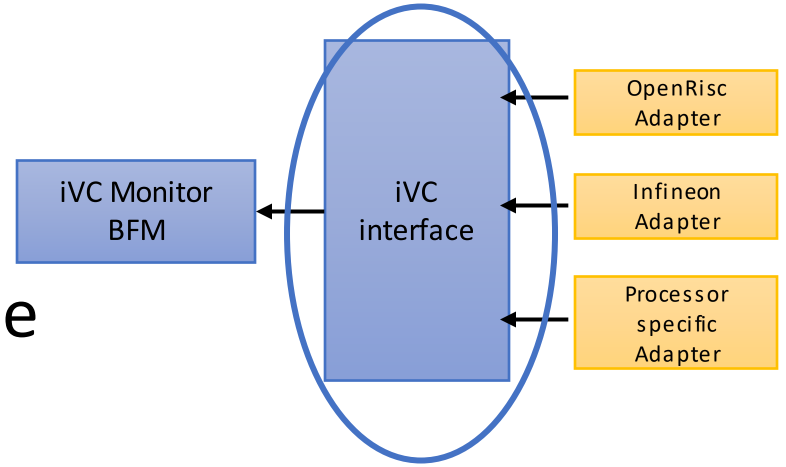


Architecture: Core-Monitor



Outline

Implementation - iVC Interface



- Connects the output from different adapters to the monitor BFM.
- The signals in the interface are CPU independent and can be used by all the adapters.

```
logic cpu_rst_n;                                              // low active reset
logic cpu_clk;                                                // clock
logic [ifx_pc_monitor_pkg::MAX_PC_LENGTH-1:0] cpu_pc[ifx_pc_monitor_pkg::PC_NUM];      // program counter
logic [ifx_pc_monitor_pkg::MAX_INSTRUCTION_LENGTH-1:0] cpu_instruction[ifx_pc_monitor_pkg::PC_NUM];
logic cpu_pc_valid[ifx_pc_monitor_pkg::PC_NUM];                                 // particular PC valid
logic [ifx_pc_monitor_pkg::MAX_MEM_ADDR_LENGTH-1:0] cpu_mem_addr;               // memory store/load address
logic [ifx_pc_monitor_pkg::MAX_MEM_DATA_LENGTH-1:0] cpu_data;                   // memory load/store data
logic [ifx_pc_monitor_pkg::MAX_MEM_SIZE:0] cpu_mem_size;
logic cpu_mem_ack;     // acknowledgement from the memory subsystem so that the memory write/read operation can proceed
logic cpu_mem_str_enable;     // memory write enable
logic [ifx_pc_monitor_pkg::MAX_REG_DATA_LENGTH-1:0] reg_data[ifx_pc_monitor_pkg::MAX_REG_NUM];  // accessed register values
logic [ifx_pc_monitor_pkg::MAX_CPU_ID_NUM:0] cpu_id;        // id for processor config file
```

Implementation - Register Tracing(1)

```
always@(posedge  cpu_clk or negedge cpu_reset_n)
   if(!cpu_reset_n) begin
      vif.reg_data[0:15]  <='{default:0};
   end
   else begin
      if((cpu_rf_we != cpu_rf_we_reg) && cpu_rf_we) begin
         vif.reg_data[cpu_rf_wr_addr] <= cpu_rf_wr_data;
      end
   end

always@(posedge  cpu_clk or negedge cpu_reset_n)
   …
   if((cpu_except_started != cpu_except_started_reg) &&cpu_except_started) begin
      vif.reg_data[EPCR0] <= exception_reg[0];    //cpu_epcr;
      vif.reg_data[ESR0]  <= exception_reg[1];    //cpu_esr;
      vif.reg_data[EEAR0] <= exception_reg[2];   //cpu_eear;
   end
end

always@(posedge  cpu_clk or negedge cpu_reset_n)
   …
   if((cpu_supervision_reg_we  != cpu_supervision_reg_we_reg)  && cpu_supervision_reg_we)  begin
      vif.reg_data[SR]    <= cpu_supervision_reg_wr_data;
    end
end
```

Since the registers R0..R15 of the OpenRisc processor are driven in different clock cycles, they are put in their corresponding addresses of the interface array

The exception and supervision registers may be driven in the same cycle and hence they take in the corresponding addresses of the interface signal array.

Implementation - Register Tracing(2)

- In the case of the Infineon's processor the register is accessed as a complete array in the respective adapter.

```
assign vif.reg_data    = register_data;
```

- Here reg_data is an array with all the register values of the processor.
- It can be updated to each array location where the address is parameterized in the adapter.

Implementation - Memory Tracing

- The interface signals for memory tracing are as below:

```
logic [ifx_pc_monitor_pkg::MAX_MEM_ADDR_LENGTH-1:0] cpu_mem_addr;   // memory store/load adder
logic [ifx_pc_monitor_pkg::MAX_MEM_DATA_LENGTH-1:0] cpu_mem_data;   // memory load/store data
logic [ifx_pc_monitor_pkg::MAX_MEM_SIZE-1:0] cpu_mem_size;                // memory size
logic cpu_mem_ack;     //acknowledgement from the memory subsystem so that the memory write/read operation can proceed
logic cpu_mem_str_enable; // memory write enable
```

- The adapter for Infineon's processor included a FIFO to initially store the memory signals.
- The OpenRisc processor required an additional if condition.
- The BFM then calls the C functions to write the relevant information into the trace file.

Implementation - PC Tracing

- PC interface signals

```
logic [ifx_pc_monitor_pkg::MAX_PC_LENGTH-1:0] cpu_pc[CPU_PC_NUM];             // program counter array
logic [ifx_pc_monitor_pkg::MAX_INSTRUCTION_LENGTH-1:0] cpu_instruction[CPU_PC_NUM];      // executed instruction array
logic cpu_pc_valid[CPU_PC_NUM];                                               // undefined instruction (particular
```

- PC values from the DUT are assigned to the interface signals in the adapter using the simple assign statements.
- The signals from the interface are processed in the monitor BFM in every clock cycle and the PC value is driven in the generate block.
- Once the PC value is processed, it will be used by the C function call via DPI-C interface for further PC tracing

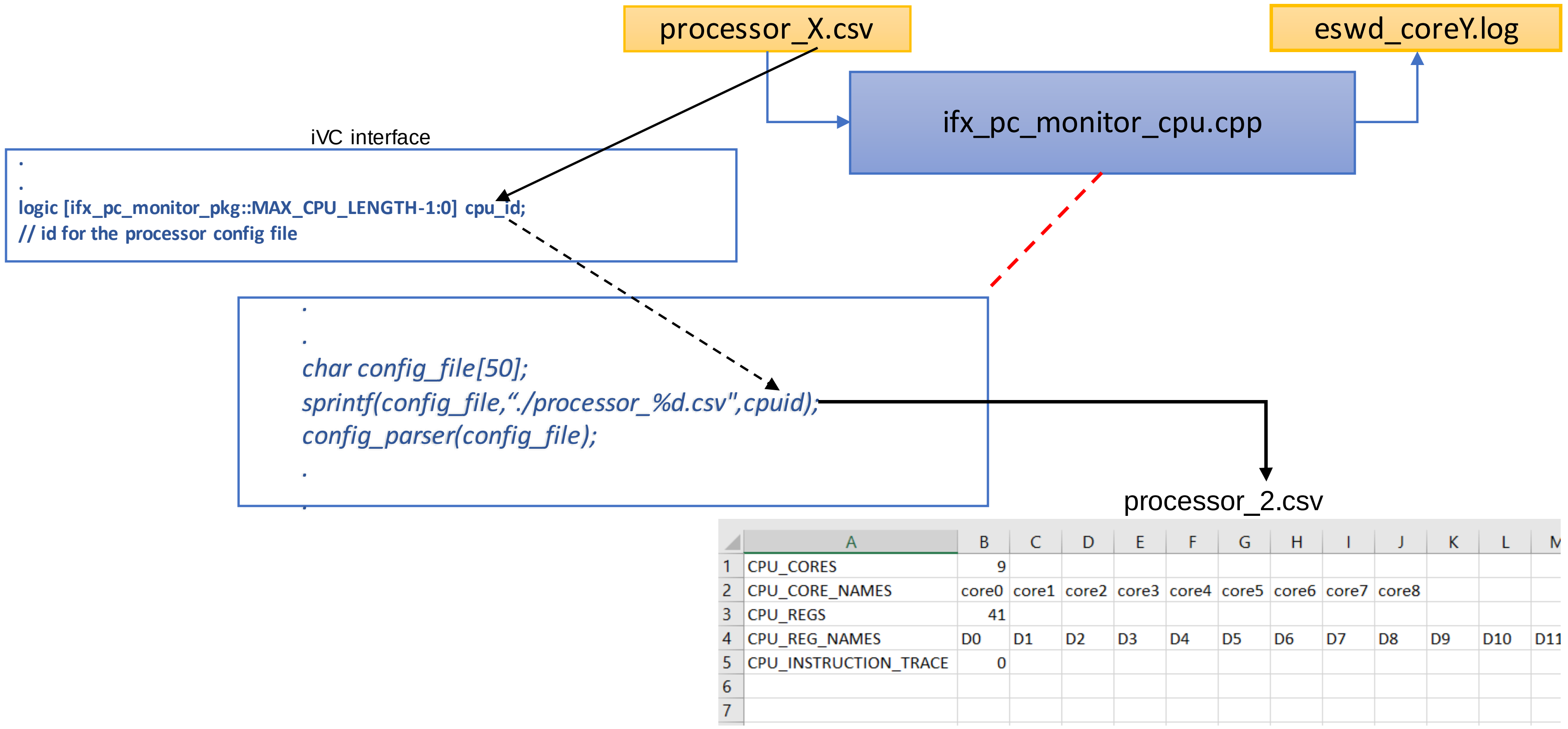
Implementation – Parameterization

- Interface and BFM definitions are parameterized for PC count and register count signals

```
interface ifx_pc_monitor_if#(parameter CPU_PC_NUM = 16, parameter CPU_REG_NUM = 100)(input logic [3:0] cpu_num);
interface ifx_pc_monitor_monitor_bfm#(parameter CPU_PC_NUM = 16, parameter CPU_REG_NUM = 100)(ifx_pc_monitor_if ifc);
```

- This provides flexibility for heterogeneous cores with varying PC and register count

Implementation – Processor specific config file



```
.
.
logic [ifx_pc_monitor_pkg::MAX_CPU_LENGTH-1:0] cpu_id;
// id for the processor config file
```

```
.
.
char config_file[50];
sprintf(config_file,"./processor_%d.csv",cpuid);
config_parser(config_file);
.
.
```

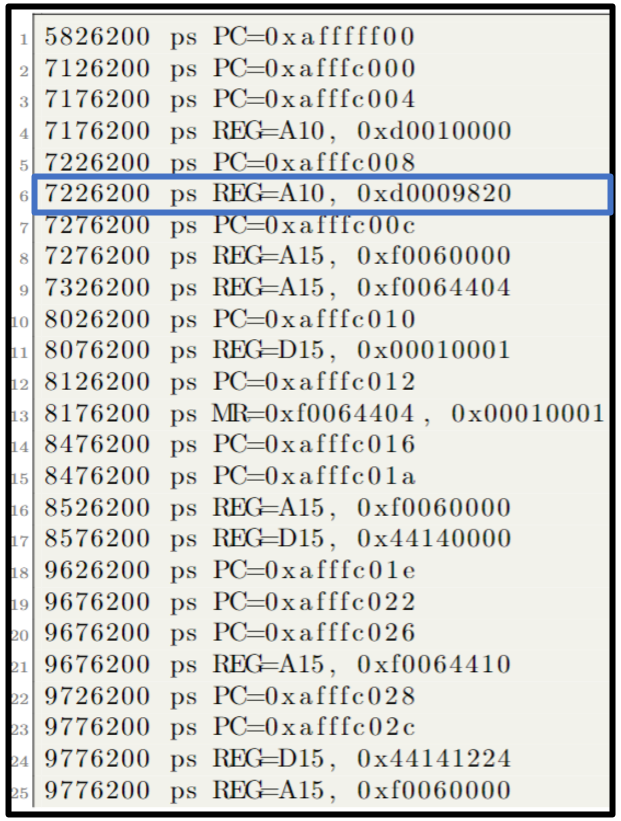
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	CPU_CORES	9											
2	CPU_CORE_NAMES	core0	core1	core2	core3	core4	core5	core6	core7	core8			
3	CPU_REGS	41											
4	CPU_REG_NAMES	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11
5	CPU_INSTRUCTION_TRACE	0											
6													
7													

Outline

Core-Monitor Setup for Emulation

- Core-Monitor setup for the Emulation environment with ZeBu emulator.
- Designed an adapter for the Infineon’s processor.
- Successfully generated the trace files.
- The generated trace file is an input to post-processing HW/SW co-debugging tools such as Indago.

CPU trace of core0 of Infineon processor,eswd_core0.log



```
5826200 ps PC=0xaffff00
7126200 ps PC=0xafffc000
7176200 ps PC=0xafffc004
7176200 ps REG=A10, 0xd0010000
7226200 ps PC=0xafffc008
7226200 ps REG=A10, 0xd0009820
7276200 ps PC=0xafffc00c
7276200 ps REG=A15, 0xf0060000
7326200 ps REG=A15, 0xf0064404
8026200 ps PC=0xafffc010
8076200 ps REG=D15, 0x00010001
8126200 ps PC=0xafffc012
8176200 ps MR=0xf0064404, 0x00010001
8476200 ps PC=0xafffc016
8476200 ps PC=0xafffc01a
8526200 ps REG=A15, 0xf0060000
8576200 ps REG=D15, 0x44140000
9626200 ps PC=0xafffc01e
9676200 ps PC=0xafffc022
9676200 ps PC=0xafffc026
9676200 ps REG=A15, 0xf0064410
9726200 ps PC=0xafffc028
9776200 ps PC=0xafffc02c
9776200 ps REG=D15, 0x44141224
9776200 ps REG=A15, 0xf0060000
```

Performance Evaluation – Emulation

- The eswd option is necessary for trace file generation.
- Emulation Runtime comparison.
- Results are based on two testcases
- There is no deterioration in performance.
- The Core-Monitor provides a high degree of reusability.
- Substantially decreases the coding-effort.

Emulation Runtime	Core-Monitor	Existing-Method
Testcase 1	3197.3s (39053.3 Hz)	3635.9s (34344.6 Hz)
Testcase 2	889.2s (32186.3 Hz)	921.3s (31063.8 Hz)

Outline

Conclusion

- A novel methodology that targets a reusable Core-Monitor
- Compatible with simulation and emulation platforms
- Designed processor specific adapters for Infineon processor and OpenRisc
- Verified the trace files generated
- No deterioration in performance
- Reduces the coding effort in large SoC designs

References

[1] https://blogs.sw.siemens.com/verificationhorizons/2021/01/06/part-8-the-2020-wilson-research-group-functional-verification-study/

[2] Cadence Design Systems, Indago Embedded Software Debug App User Guide.

[3] Synopsis, Zebu Server User Guide https://www.synopsys.com/verification/emulation/zebu-server.html

[4] https://www.doulos.com/knowhow/systemverilog/systemverilog-tutorials/systemverilog-dpi-tutorial/

Questions?

THANK YOU