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

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

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

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

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

- Software Development
- Hardware Development
- Trace File Generation
  - Simulation Based
  - Emulation Based
- Post-Process Debugging Tools
Architecture: Core-Monitor
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.

```verilog
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
```
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_epr;       //cpu_epr;
  vif.reg_data[ESR0]  <= exception_reg[1]; //cpu_esr;
  vif.reg_data[EEAR0] <= exception_reg[2]; //cpu_ear;
  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.

\[
\text{assign vif.reg\_data} = \text{register\_data};
\]

• 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 address
  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

```plaintext
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

```c
// id for the processor config file

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

// iVC interface

logic [ifx_pc_monitor_pkg::MAX_CPU_LENGTH-1:0] cpu_id;

// id for the processor config file

ifx_pc_monitor_cpu.cpp

processor_X.csv

eswd_coreY.log

processor_2.csv

```

![Diagram of iVC interface and file connections]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU_CORES</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CPU_CORE_NAMES</td>
<td>core0</td>
<td>core1</td>
<td>core2</td>
<td>core3</td>
<td>core4</td>
<td>core5</td>
<td>core6</td>
<td>core7</td>
<td>core8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CPU_REGS</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CPU_REG_NAMES</td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
</tr>
<tr>
<td>5</td>
<td>CPU_INSTRUCTION_TRACE</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

accellera SYSTEMS INITIATIVE
Outline

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

<table>
<thead>
<tr>
<th></th>
<th>Emulation Runtime</th>
<th>Core-Monitor</th>
<th>Existing-Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testcase 1</td>
<td></td>
<td>3197.3s (39053.3 Hz)</td>
<td>3635.9s (34344.6 Hz)</td>
</tr>
<tr>
<td>Testcase 2</td>
<td></td>
<td>889.2s (32186.3 Hz)</td>
<td>921.3s (31063.8 Hz)</td>
</tr>
</tbody>
</table>
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


Questions?
THANK YOU