Hardware-Software Co-Simulation

Ki young Choi

School of EECS
Seoul National University
Outline

1. Introduction
2. Co-Simulation Systems
3. Case Study
4. Timed Co-Simulation
5. Examples
6. Conclusion
1. Introduction

• What is HW/SW co-simulation?
  – Simulation of heterogeneous systems whose HW and SW components are interacting

• Roles of co-simulation
  – Verification of system description before system synthesis
  – Verification of mixed system after system synthesis and integration
  – System performance estimation for system partitioning
1. Introduction (cont’d)

- Issues of HW/SW co-simulation
  - Time-accuracy
  - Processor model
  - Performance
  - Interface transparency
  - Transition to co-synthesis
2. Co-Simulation Systems

- Becker 1992
  - Co-simulation of Network Interface Unit (NIU) using PLI of Verilog-XL simulator, C++, and Unix Socket
  - Synchronized handshake + cycle accurate processor interface model

System under Development
Co-simulation Environment

NIU Firmware in C++
  - Interface functions
  - Co-simulation support library
  - Unix

NIU HW description in Verilog
  - SPARC model
  - Port model
  - other device model
  - IPC tasks
  - Builtin tasks
  - Verilog-XL
  - Unix

NIU Monitor Program in C++
  - Interface functions
  - Co-simulation support library
  - Unix

Network
2. Co-Simulation Systems (cont’d)

- Thomas 1993
  - Co-simulation using PLI of Verilog-XL simulator, and Unix Socket
  - Synchronized handshake with no processor model
2. Co-Simulation Systems (cont’d)

- **Poseidon (Gupta 1992)**
  - Pin-level (cycle accurate) model of processors
  - Software in assembly code of the target processor (DLX)
2. Co-Simulation Systems (cont’d)

- Ptolemy (Kalavade 1993)
  - Pin-level (cycle accurate) model of processors
  - Software in assembly code of the target processor (DSP56000)
  - Multiparadigm simulation by supporting different design styles encapsulated in object called domains
    - SDF domain: algorithm simulation
    - Thor domain: digital hardware simulation
      - Socket connection with Sim56000 (Motorola’s DSP56000 simulator)
2. Co-Simulation Systems (cont’d)

- Lee 1993
  - Ptolemy is used
  - Simulation intermixed with physical implementation

![Diagram of co-simulation systems](image)

- UNIX
  - sockPort
  - Wrapper
  - Simulation
  - Workstation

- VxWorks
  - Interface Library
  - msgSend()
  - msgReceive()
  - Remote Process
  - Server Process
  - Single-Board Computer

- Remote Process
  - Interface Library
  - msgSend()
  - msgReceive()
  - Custom Board
2. Co-Simulation Systems (cont’d)

- **POLIS** (Balari n 1997)
  - Model of computation: network of CFSMs
  - Ptolemy is used for co-simulation
    - Each CFSM is a single star
    - HW stars run concurrently and terminate in a single clock cycle
    - SW stars are mutually exclusive and use run time estimation
    - DE (Discrete Event) domain of Ptolemy is used
    - Software scheduler is used to schedule mutually exclusive SW stars
  - VHDL simulators can also be used
2. Co-Simulation Systems (cont’d)

- **Petrellis 1996**
  - *Generic processor description + front end specific processor adaptor*

*Diagram showing a flowchart with interconnections between components labeled as GPD, FESPA, INTER, LATCH, FL2BITV, BITV2FL, data bus, I/O, address, read, write, sen_act, strb, sen_val, act_val, error, and time.*
2. Co-Simulation Systems (cont’d)

- Borgatti 1996
  - Virtual Emulation = Virtual System (software simulation) + Prototype System (hardware emulation)
2. Co-Simulation Systems (cont’d)

- Simulator
- Simulator
- Simulator

network

software interface

Scheduler

SCSI bus

hardware interface

Pod (xilinx)

Aptix FPCB prototyping board

hardware interface
2. Co-Simulation Systems (cont’d)

  - Memory image server
3. Case Study

- **Simulation levels**
  - Co-simulation at any abstraction levels
  - Abstract level co-simulation
    - VHDL and C for HW and SW components
    - Simulation models of IF are generated and inserted
  - Detailed level co-simulation
    - After target architecture is determined and IF synthesized
    - More detailed IF models are generated and inserted
3. Case Study (cont’d)

- **Software process**
  - Executing a C program (SW component) on a workstation
    - No need of processor models
    - Run at workstation speed with speed dominated by HW simulation performance
  - Synchronized handshake
    - Implemented using Unix socket

- **Hardware simulation process**
  - VHDL simulator (Inspire) exercising a model of HW
  - Use foreign attribute (VHDL-93)
    - HW in VHDL + IPC in C
    - Enables simulation of HW communicating with SW
3. Case Study (cont'd)

- **Interface transparency**
  - Through
    - **Automatic interface model generation**
    - **Automatic interface model call/instantiation**
  - **Regardless of target arch. and comm. protocol, user can concentrate on the functionality simulation of HW and SW cores**

```
C program
...
write_socket()
...
```

```
write_socket()
{
..............
}
```

```
read_socket()
{
..............
}
```

```
channel unit
IPC handler
read_socket()
............
```

```
HW core
decoder/signal register
```

```
HW prototype
standard device or HW prototype
```

```
IPC routines
Socket IPC
```

```
from generator
from library
user-given
```
3. Case Study (cont’d)

- **Performance improvement thru HW simulation acceleration**
  - HW sim time dominates overall co-simulation time
  - HW function added/refined --> sim time gets longer
  - Sim. acceleration through incremental HW prototyping

![Diagram showing Sparc CPU, HW (Inspire), device driver, SW, CAD tools, custom board, HW (FPGA), SBUS interface, and SBUS connections.]

- Sparc CPU
- HW (Inspire)
- SW
- CAD tools
- device driver
- custom board
- HW (FPGA)
- SBUS interface
- SBUS
3. Case Study (cont’d)

- Migration from VHDL sim. to HW prototype

- VHDL-simulated (incremental part)
- HW-prototyped

HW function is completely defined and prototyped
3. Case Study (cont’d)

- Transition to co-synthesis after co-simulation
  - IPC routine calls --> device driver/IO function calls
  - Top-level VHDL entity stripped
  - VHDL model --> refined VHDL model with decoder/signal register
  - IF HW model inserted

co-simulation  →  co-synthesis
4. Timed Co-Simulation

- **Untimed co-simulation**
  - Hardware and software time scales are not synchronized
  - Good for verifying functional completeness
  - Correct results for handshaking protocol

- **Timed co-simulation**
  - Hardware and software time scales are synchronized at each data exchange
  - Useful for overall performance assessment
  - Can be used for both handshaking and non-handshaking protocol
4. Timed Co-Simulation (cont’d)

- **How to Implement a timed co-simulator**
  - **Nano-second accurate co-simulation**
    - Need expensive processor model
    - Very slow
  - **Cycle-based co-simulation**
    - Need cycle-based processor model
    - Slow
  - **Synchronization thru software timing estimation**
    - No need of processor simulation model
    - Fast
4. Timed Co-Simulation (cont’d)

- Software timing estimation

Modify the intermediate C program for profiling
4. Timed Co-Simulation (cont’d)

- **Synchronization**
  - IPC (Inter-Process Communication)
  - Non-IPC
  - Lockstep
    - synchronize at every step
  - Optimistic
    - simulate to optimistic next synchronization point
    - in the event of inconsistency, rollback
  - Conventional synchronization
    - lockstep + IPC
    - large synchronization overhead
4. 4. Timed Co-Simulation (cont’d)

- Non-IPC-based implementation
  - SW instructions in the event list

```
bcnd _loopstart, LT
lacl var
```
4. Timed Co-Simulation (cont’d)

Implementation

HW Model (VHDL)

procedure sw_code
attribute foreign of sw_code is “C”
timing : postponed process
begin
  sw_code(address, data, t_time);
  wait for t_time - now;
end postponed process;

SW Processor Model

void sw_code(int*, int*, int*)
{
  ...
  ISR();
  asm[PC].func();
}

* Good performance
  with a limitation of
  single-processor co-simulation
4. Timed Co-Simulation (cont’d)

• Optimistic Co-simulation
  1. SW starts to run six clk’s optimistically.
  2. At checkpoint $t_c$, the snapshot of SW simulation is stored.
  3. HW starts to run.
  4. If HW -> SW event occurs before time six, then
     SW rolls back to $t_c$ and restarts from the stored state.

---

![Diagram showing SW, HW, and IPC overhead with time and checkpoint annotations.](diagram.png)
4. Timed Co-Simulation (cont’d)

1. At a synchronization point, set checkpoint ($t_c$) and target simulation time ($t_o$).

2. Run SW until $t_o$. If time reaches $t_c$, then store the state of SW processor.
4. Timed Co-Simulation (cont'd)

3. If SW>HW event occurs at $t_{a_{sw}}$ (before $t_0$), then
   stop SW at $t_{a_{sw}}$.
   Else, stop SW at $t_0$.

4. Run HW
   until $t_{a_{sw}}$ or $t_0$. 

![Diagram showing SW and HW timelines with conditions and actions]
4. Timed Co-Simulation (cont’d)

5. If HW->SW event occurs at $t_a^{\text{hw}}$ (before $t_0$ or $t_a^{\text{sw}}$), then, stop HW at $t_a^{\text{hw}}$, roll back SW to $t_c$ and re-execute until $t_a^{\text{hw}}$.

Else, stop HW at $t_0$ or $t_a^{\text{sw}}$.

Go to step 1.
4. Timed Co-Simulation (cont’d)

**HW Model (VHDL)**

```vhdl
procedure socket_read;
attribute foreign of socket_read is “C”;
procedure socket_write;
attribute foreign of socket_write is “C”;

timing : postponed process
begin
  if now = t_time or hw2sw_intr then
    -- send intr or ready msg to SW
    socket_write(address, data, now);
    -- receive a next t_time from SW;
    socket_read(address, data, t_time);
  end if;
  wait for t_{GCD};
end postponed process;
```

**SW Processor Model**

```c
void main()
{
  while(1) {
    if (mode == SYNC)
      // set mode = ASYNC
      // in socket_read()
      socket_read();
      // roll back if needed
      ...
      ISR();
      asm[PC].func();
    
    if (mode == SYNC)
      socket_write();
  }
}
```
4. Timed Co-Simulation (cont’d)

- Experiments
  - Types of Co-simulation
    - IPC-based (multiple-process) v.s. non-IPC (single-process)
    - Synchronous v.s. optimistic
  - Simulators
    - HW: an event-driven compiled-code VHDL simulator (Inspire)
    - SW: a compiled model of TMS320C50 processor or ARM processor + optimistic simulation features
Example 1: A CNC machine

<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC-based</td>
<td>3685</td>
<td>1097</td>
</tr>
<tr>
<td>Non-IPC</td>
<td>1073</td>
<td>1095</td>
</tr>
</tbody>
</table>

- IPC overhead is dominant: 70%
- Rollback overhead: < 2%
4. Timed Co-Simulation (cont’d)

- Example 2: LZ-77 algorithm

<table>
<thead>
<tr>
<th></th>
<th>synchronous</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC-based</td>
<td>1165</td>
<td>768</td>
</tr>
<tr>
<td>Non-IPC</td>
<td>732</td>
<td>758</td>
</tr>
</tbody>
</table>

- HW simulation is dominant: 63%
- Rollback overhead: < 3%
4. Timed Co-Simulation (cont’d)

- Task model for reducing state saving overhead
4. Timed Co-Simulation (cont’d)

- Task-based state saving for SW

<table>
<thead>
<tr>
<th>Time</th>
<th>Task A</th>
<th>Task B</th>
<th>Stored States</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_c(k)$</td>
<td>$I(A,i)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_c(k+1)$</td>
<td></td>
<td>$I(B,j)$</td>
<td>B1</td>
</tr>
<tr>
<td>$T_c(k+2)$</td>
<td>A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_c(k+3)$</td>
<td></td>
<td>B3</td>
<td></td>
</tr>
</tbody>
</table>

$I(A,i)$ and $I(B,j)$ represent the states of tasks A and B, respectively.
4. Timed Co-Simulation (cont’d)

- **Task-based state saving for HW**
  - Non-interruptible
  - Single checkpoint at the end of execution of each instance of a task
  - A straggler message to a HW task represents
    - Cancellation of initiation signal
    - Modification of data previously read by the task
    - Both cases require rollback to the start point of the instance execution
4. Timed Co-Simulation (cont’d)

- Example: CNC machine

Optimistic cosim.  Synchronous cosim. based on GVT

HW simulator  SW (C50) simulator  Motor simulator

init

straggler
rollback

30 us
### Timed Co-Simulation (cont’d)

#### Speedup

<table>
<thead>
<tr>
<th></th>
<th>SW</th>
<th>Motor</th>
<th>IPC</th>
<th>HW</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sync.</td>
<td>135</td>
<td>50</td>
<td>4551</td>
<td>211</td>
<td>4947</td>
</tr>
<tr>
<td>Opt.</td>
<td>166</td>
<td>50</td>
<td>107(SW)</td>
<td>80(HW)</td>
<td>245</td>
</tr>
</tbody>
</table>

#### State Saving

- **Overhead**

![Graph showing State Size vs. Run-time](image-url)
5. Examples

- **MPEG2 Video Decoder**
  - IDCT: software implementation is inefficient and hardware can exploit the inherent parallelism

- **H.263 Video Encoder**
  - Large amount of same computation for motion estimation (hardware)
  - Flexibility required for some functions (software)

- **CNC Controller**
  - Real-time hardware-software-mechanical system simulation
5. Examples (cont'd)

- **MPEG2 Video Decoder**
  - Block Diagram

- VLD: Variable Length Decoder
- IQ: Inverse Quantizer
- Zigzag Buffer
- IDCT
- Frame Store
- Forward Predictor
- Backward Predictor
- Merger

VLD : Variable Length Decoder
IQ : Inverse Quantizer
+ : video output
5. Examples (cont’d)

- Implementation
  - HW (IDCT) : FPGA
  - SW: executable codes on IBM PC
5. Examples (cont’d)

- Co-simulation/prototyping results

<table>
<thead>
<tr>
<th>codes</th>
<th>time (usec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send data</td>
<td>154</td>
</tr>
<tr>
<td>IDCT_start</td>
<td>3</td>
</tr>
<tr>
<td>IDCT</td>
<td>181</td>
</tr>
<tr>
<td>IDCT_done</td>
<td>3</td>
</tr>
<tr>
<td>receive result</td>
<td>149</td>
</tr>
</tbody>
</table>

interface overhead
5. Examples (cont'd)

- H.263 Video Encoder
  - Block Diagram of Video Codec

  - a) Video encoder
    - Source coder
    - Video multiplexer coder
    - Transmission buffer
    - Transmission coder

  - b) Video decoder
    - Source decoder
    - Video multiplexer decoder
    - Receiving buffer
    - Receiving decoder

External control

Coded bit stream

Video signal

Coded signal
5. Examples (cont’d)

- Block Diagram of Source Coder

Video in → Coding control → DCT → Q → IQ → IDCT → + → motion vector

- ME/MC

- +
### Analysis of Telenor’s H.263 software (encoder program)

- **Analysis result by GNU’s gprof for encoding 160 QCIF frames**

<table>
<thead>
<tr>
<th>% Time</th>
<th>Run time (second)</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>39.5</td>
<td>83.6</td>
<td>SAD_Macroblock</td>
</tr>
<tr>
<td>25.0</td>
<td>53.0</td>
<td>mcount</td>
</tr>
<tr>
<td>9.2</td>
<td>19.5</td>
<td>umul</td>
</tr>
<tr>
<td>6.6</td>
<td>13.8</td>
<td>umul_8bit</td>
</tr>
<tr>
<td>6.2</td>
<td>13.1</td>
<td>FindHalfPel</td>
</tr>
<tr>
<td>3.7</td>
<td>7.7</td>
<td>MotionEstimation</td>
</tr>
<tr>
<td>3.3</td>
<td>7.0</td>
<td>Dct</td>
</tr>
<tr>
<td>2.4</td>
<td>5.2</td>
<td>idctref</td>
</tr>
<tr>
<td>2.0</td>
<td>4.1</td>
<td>Quant</td>
</tr>
<tr>
<td>1.7</td>
<td>3.8</td>
<td>umul_4bit</td>
</tr>
</tbody>
</table>

Subprograms for Motion Estimation 49.4% of total run time!!!
5. Examples (cont’d)

- Decoder program
  - decode 30 QCIF frames/sec @ Pentium 150MHz
  - software implementation meets performance requirements

- Partition of HW/SW
  - manual partition by intuition and convenience
  - HW: integer motion estimation
  - SW: other functions
5. Examples (cont’d)

- **Co-simulation of H.263 video encoder**
  - **Co-simulation environment model**
    - synchronized handshake (untimed)
    - using socket
  - **Co-simulation results**
    - co-simulation time: 31 hours/frame
    - most of the time is taken by hardware simulation
5. Examples (cont’d)

- **CNC Controller**
  - 7 tasks

- **Partitioning**
  - 15%–33% cost reduction through HW resource sharing
  - Maintain maximum error under 1%

![Diagram of contour error between desired trajectory and cutter trajectory, with 10mm, 15mm, and 10mm annotations]
5. Examples (cont’d)

- Timed Co-simulation

CNC Controller

VHDL simulator

```
procedure sw_code
attribute foreign of sw_code is “C”
postponed process
sw_code(address,data,t_time)
wait for t_time-now;
```

Multi-func module 1

Hardware Engines

C50 compiled model

```
void sw_code(int*,int*,int*) {
    if TIME % 300 == 0
        socket_rw();
    ISR();
    asm[PC].func();
}
```

Multi-func module 3

Motor Simulator

```
void main() {
    do {
        if TIME % 300 == 0 {
            socket_wr();
            plant_x();
            plant_y();
        }
        TIME++;
    } while(1);
}
```
6. Conclusion

- Co-simulation is becoming popular
  - Mentor Graphics: Seamless CVE
  - VIEWlogic: Eagle Tools
  - Synopsys: COSSAP + VHDL
  - Cadence: SPW+ VHDL

- Performance and time-accuracy are still an issue
  - cycle-based simulation
  - synchronization thru SW timing estimation
References

References (cont’d)


References (cont’d)