Handling Dynamic Frequency Changes in Statically Scheduled Cycle-Accurate Simulation

Marius Gligor and Frédéric Pétrot

TIMA Laboratory - SLS Group, INP-UJF-CNRS
Grenoble, France

January 27, 2011
Outline

- Introduction
- Multiple clocks approach
- Frequency division approach
- Experimental results
- Conclusion and Perspectives
Context

System simulation

- Architecture validation and early software coding
- Optimize hardware/software performances and consumption through architecture exploration
- Simulation speed is an issue
  - High level simulation models increasingly used

Cycle-accurate simulation

- Hardware protocol validation and accurate power/energy estimation
- Components modeled at the finite state machine (FSM) level
  - Many events during the simulation
- Low simulation speed
Motivation

Static scheduling approaches

- Sequential part of the components executed only once per clock cycle
- 3X to 5X faster than dynamic scheduling
- Static scheduling computed off-line
  - Impose constraints
  - Architectures containing components working at different frequencies not supported
  - Dynamic frequencies change not supported

Nowadays IPs

- Designed to belong to voltage-frequency islands
- DVFS used for power efficiency

Contribution: 2 static scheduling strategies for such architectures
FSM Based Simulators / SystemCASS

FSMs
▶ Connected to the component ports
▶ Moore: outputs depend only on the current state
▶ Mealy: outputs depend on the current state and the inputs
▶ Transition, Moore and Mealy functions

SystemCASS
▶ Modeling constraints
  • FSMs synchronous on an unique clock
  • Transition functions: positive clock edge
  • Moore functions: negative clock edge
  • Mealy functions: negative clock edge and input ports
▶ Implementation
  • No event notification when a signal value changes
▶ Optimizations
  • Signals have only the current value
  • Calling order of the Mealy functions computed using a graph of dependencies

```c
void simulate_a_cycle ()
{
    //positive clock edge
    transition_functions ();
    update_registers ();

    //negative clock edge
    moore_functions ();
    stab_all_mealy_functions ();
}
```
Outline

- Introduction
- **Multiple clocks approach**
- Frequency division approach
- Experimental results
- Conclusion and Perspectives
Multiple clocks Approach

Characteristics

- Uses multiple clock components
  - Dependency on non clock signals not allowed
- Main idea: scheduling pattern with the period equal to the least common multiple (LCM) of the clocks periods
- Simulation point: contains at least a clock edge

Scheduling pattern example
Details

Implementation & execution

- Processes are sorted
  - A list of Transition and a list of Moore functions for each clock
  - These lists don’t change during the simulation

- A list of simulation points
  - A list of positive and a list of negative clock edges
  - Time offset to be added to pass to next simulation point

- Execution of a simulation point

```c
void simulation_point ()
{
    transition_functions_simpoint ();
    moore_functions_simpoint ();
    update_registers ();
    stab_all_mealy_functions ();
}
```
Multiple clocks approach -

Architecture example

Modeling

▸ The Interconnect, RAM, TTY etc. use the frequency of the system clock
▸ The frequency of each processor and its caches generate by a clock component
Run-time Change of the Frequencies

Characteristics

- A new API added to the clock component: `change_period`
- Scheduling pattern recomputed
- Frequency changes at the next edge of the target clock

Frequency change example
Details

Run-time frequency change

- Scheduling pattern recomputed
- Takes time

- Number of simulation points in the new scheduling pattern

$T_{\text{pattern}} = 12$

$T'_{\text{pattern}} = 4$
Outline

- Introduction
- Multiple clocks approach
- **Frequency division approach**
- Experimental results
- Conclusion and Perspectives
Frequency Division Approach

Characteristics

- Uses a single clock component
- Frequency equal to the LCM of all possible frequencies
- The components frequencies obtained by division
  - Dependency on non clock signals allowed
- The simulation cycle changes
  - Call of the functions dependent on non clock signals

```c
void simulate_a_cycle ()
{
    //positive clock edge
    transition_functions ();
    update_registers ();

    //negative clock edge
    moore_functions ();

    transition_functions_non_clock_signal();
    moore_functions_non_clock_signal ();
    update_registers ();

    stab_all_mealy_functions ();
}
```
Architecture example

Modeling

- The frequency of the Interconnect, RAM, TTY etc. given by the frequency divisor
- The frequency of each processor and its caches generated by the DVFS component
Details

Implementation

- Sort the Transition and Moore functions depending on the unique clock edges (SystemCASS part)
- Create a list of Transition and a list of Moore functions depending on non clock signal edges
- Generate a function for these 2 lists
  - Call the Transition and Moore functions on their required signal edge
  - A static variable (v1) for each generated clock signal

```c
void out_signal_clock_transition_processes ()
{
    register fct p;

    static unsigned char v1 = 0;
    if (*(unsigned char *) 0x934cb84UL != v1)
    {
        v1 = *(unsigned char *) 0x934cb84UL;
        if (v1)
        {
            // ARM1->transition()
            p.integer = 0x80a7160UL;
            p.pf ((void *) 0x86d1fa8UL);
            ...
        }
    }
    ...
```
Limitations

Static scheduling simulator based on multiple clocks

- The clock periods must be in a harmonic ratio to allow their LCM computation
- Can not be used for architectures containing generated clock signals that are not periodical

Static scheduling simulator based on frequency division

- Can not be used if there is at least one frequency unknown when the architecture is modeled
- A single level of signal dependency is permitted
Outline

- Introduction
- Multiple clocks approach
- Frequency division approach
- Experimental results
- Conclusion and Perspectives
Motion-JPEG Application Case Study

Software stack

- Motion-JPEG decoding application
- Mutek operating system
  - POSIX compliant
  - SMP version
- Energy saving algorithm - 1.5 frequency changes / simulated ms

Hardware platform

- Processors
- Caches
- Interconnect
- Memories
- DVFSes
- …
Experimental Results

Hardware architecture

- SoCLib components
- Compiled and linked with several simulators
  - SystemC - reference
  - SystemCASS
  - Multiple clock based static scheduling simulator
  - Frequency division based static scheduling simulator

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Simulation speedup vs. SystemC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 frequency</td>
</tr>
<tr>
<td>SystemCASS</td>
<td>3.50 X</td>
</tr>
<tr>
<td>Frequency division</td>
<td>3.50 X</td>
</tr>
<tr>
<td>Multiple clocks</td>
<td>3.53 X</td>
</tr>
</tbody>
</table>
Synthetic Architecture Case Study

Configurable hardware architecture

- Number of modules working at the system frequency
  - Some generates the clock signal for other modules
- Input and output ports, registers
- Number of frequency changes / simulated ms
- Pure hardware platform (no software stack)
Multiple clock simulator

- Simulation speedup
  - Number of simulation points in scheduling pattern
  - Number of frequency changes

![Graph showing simulation speedup vs. number of frequency changes. The x-axis represents the number of frequency changes per millisecond, ranging from 0 to 500. The y-axis represents speedup vs. SystemC, ranging from 0 to 4. The graph includes multiple lines, each representing a different number of frequency changes, marked as 20, 184, 2196, 11818, and 65918. The speedup decreases as the number of frequency changes increases.](image-url)
Experimental Results (2)

Frequency division simulator

- Simulation speedup
  - Ratio between the unique clock frequency and the system frequency
  - Does not depend on the number of frequency changes

![Graph showing simulation speedup vs. number of frequency changes](image)
Outline

- Introduction
- Multiple clocks approach
- Frequency division approach
- Experimental results
- Conclusion and Perspectives
Simulations strategies implemented at the CA abstraction level

- Static scheduling
- Support multiple frequencies that can change runtime
  - Multiple clock components - runtime change of the their period
  - Division of a single frequency
- Simulation speedup of 3.5 compared to a dynamic scheduling simulator

Future works

- Improve the simulation speed when the frequencies change often
  - Memorize the scheduling patterns already computed
    - Equivalence scheme between the scheduling patterns
Thank you!

{marius.gligor@imag.fr}