A Self-adaptable Distributed DFS Scheme for NoC-based MPSoCs

Thiago Raupp da Rosa, Douglas Cardoso, Fernando Moraes
{thiago.raupp, douglas.cardoso}@acad.pucrs.br, fernando.moraes@pucrs.br

Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS

Abstract

A clear trend in high-end embedded systems is the use of multiprocessor systems on chip (MPSoCs). As processor count in these devices increases, the use of NoCs becomes relevant, if not mandatory. However, power and energy restrictions, especially in mobile applications may render the design of NoC-based MPSoCs over-constrained. The use of traditional dynamic voltage and frequency scaling (DVFS) techniques proved useful in several scenarios to save energy/power, but it presents scaling problems and slow response times. This work proposes a self-adaptable distributed dynamic frequency scaling (DFS) scheme for NoC-based MPSoCs. It takes into account the communication load and the utilization level of each processor to dynamically change its operating frequency. Frequency change decisions and clock generation are executed locally to each processor. Clock generation is simple, based on clock gating of a single global clock. The overhead of the scheme in terms of area is minimum, the range of generated clocks frequencies is wide and the response time of frequency switching is negligible. Experimental results in an actual MPSoC running a real application show that the proposed scheme has an average execution time overhead below 14%, and may lead to considerable power and energy savings, since it allows an average reduction of 64% on the total number of executed instructions.

1. Introduction

MPSoCs increase system performance by employing multiple processors to execute system tasks, which are interconnected by a communication infrastructure. NoC-based MPSoCs provide massive computing power on a single chip.

Energy consumption in CMOS circuits can be reduced by controlling two main variables: the supplied voltage and the operating frequency. Controlling these two variables at runtime is the basis of Dynamic Voltage and Frequency Scaling (DVFS) techniques.

This paper proposes and evaluates a new technique for Dynamic Frequency Scaling (DFS) with fixed system voltage. The main goal of this technique is to enable fast frequency switching according to each processor’s workload and communication load. The proposed DFS scheme is evaluated in a synthesizable NoC-based MPSoC.

The rest of this paper is organized as follows. Section 2 reviews the related work, comparing it to the proposed technique. Section 3 presents the clock generation module. Section 4 describes the MPSoC architecture and the required modifications to enable the DFS scheme. The proposed DFS controller is presented in Section 5. Section 6 presents the experimental setup and results. Finally, conclusions and future works are drawn in Section 7.

2. Related Work

Tab. 1 summarizes the state of the art according to three criteria: target architecture, monitoring parameter, and implementation. Just a few works address NoC-based MPSoCs [1], [10]-[13]. Most works target only one CPU or bus-based architectures, using a centralized controller. DVFS schemes may use hardware or software control parameters. Hardware parameters for controlling DVFS include temperature [7], [10], process variation [1], current [2] and load in communication buffers [4]-[6]. Software parameters include application profile [8], [13] and scheduling tasks [3], [9], [12]. In terms of implementation, most proposals employ software parameters, releasing to the hardware the monitoring process (when a hardware parameter is monitored). The approach proposed here aims to control DFS scheme through hardware and software mechanisms. The hardware mechanism obtains data from the Network Interface (NI) and from the processor to parameterize the clock generation module, setting the correct frequency for the NoC and PEs. Software mechanisms are responsible for monitoring a set of parameters, making them available to the hardware mechanism.

3. Clock Generation

This Section presents the principles and design of a clock generator to enable the proposed DFS scheme. This module uses as input a reference clock, which consists in the highest frequency usable in the system as a local clock. The principle of the clock generation process is to achieve clock division by simply omitting selected cycles of the reference clock, as Fig. 1 illustrates: initially, inputs num_i and den_i are natural numbers 2 and 5, respectively. This corresponds to set the frequency of the clock generator to two-fifths (40%) of the
reference clock. In other words, for each \( \text{den}_i \) reference clock cycles, \( \text{num}_i \) cycles are propagated to the output clock.

Any frequency obtained by changing the \( \text{num}_i \) and \( \text{den}_i \) values can be generated with the obvious exceptions (\( \text{den}_i=0 \) is not an acceptable value, \( \text{num}_i=0 \) corresponds to a clock gating action and the constraint \( \text{num}_i \leq \text{den}_i \) must be respected). Before changing the \( \text{num}_i \) and \( \text{den}_i \) values, the \( \text{restart}_i \) signal must be asserted to momentarily stop the output clock and reinitialize internal registers. After releasing \( \text{restart}_i \), the new frequency, defined by the modified values of \( \text{num}_i \) and \( \text{den}_i \) appears at the output.

The main advantages of this clock generation module are the low area overhead and a large set of generated frequencies. In addition, the clock output is always stable, contrary to what happens in standard DFS methods, where the time required to stabilize a new frequency can be significant. The proposed module is also glitch free by construction. Such features make the use of the proposed clock generator module appropriate for methods, where the time required to stabilize a new frequency can be significant. The proposed module is also generated frequencies. In addition, the clock output is always stable, contrary to what happens in standard DFS

### Tab. 1 - DVFS state-of-the-art comparison.

<table>
<thead>
<tr>
<th>Author</th>
<th>Architecture</th>
<th>Monitoring Parameter</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Herbert[1] 2009</td>
<td>NoC-Based MPSoC</td>
<td>Process Variation</td>
<td>Off-line Calibration (Design Variability), Algorithms in Software</td>
</tr>
<tr>
<td>Chabloz [3] 2010</td>
<td>Synchronous Islands</td>
<td>Tasks Deadlines</td>
<td>GRLS Scheme, Local clock generation</td>
</tr>
<tr>
<td>Alimonda [5] 2009</td>
<td>Bus-Based MPSoC</td>
<td>Queues Load</td>
<td>Central Controller Hardware</td>
</tr>
<tr>
<td>Liu [9] 2009</td>
<td>2 CPUs, Bus-based Interconnect</td>
<td>Tasks Slacks</td>
<td>Task Graph Unrolling Software</td>
</tr>
<tr>
<td>Goossens [12] 2010</td>
<td>NoC-Based MPSoC</td>
<td>Tasks Slacks</td>
<td>Voltage and Frequency Scaling Hardware, Software to adjust the Controller</td>
</tr>
<tr>
<td>Kong [13] 2010</td>
<td>Bus-Based MPSoC</td>
<td>Application Profile</td>
<td>Software computes Suitable DVFS Level and Informs Controller Hardware</td>
</tr>
<tr>
<td>This Work</td>
<td>NoC-Based MPSoC</td>
<td>Communication and CPU Load</td>
<td>Local Clock Generation, Controller sets Correct Frequency Level, Software updates Controller with Current CPU state</td>
</tr>
</tbody>
</table>

![Fig. 1 - Example of the proposed clock generation process. Signal clock_i is the reference clock and clock_o is the output of some clock generator.](attachment:image1)

### 4. System Architecture

The reference MPSoC [14] is a homogeneous multiprocessing NoC-Based MPSoC. Fig. 2 shows an instance of this MPSoC. The 2-D mesh NoC used in the reference MPSoC has the following features: wormhole packet switching, flit width equal to 16 bits, XY routing algorithm, round-robin arbitration, input buffers with 8-flits depth.

![Fig. 2 - Block diagram of the reference MPSoC architecture.](attachment:image2)

Each PE includes the following modules: (i) a 32-bit Plasma processor (a MIPS-like architecture); (ii) a local memory (RAM); (iii) a DMA module, responsible for transferring the task object code to the memory and messages to/from the NoC to the local memory; (iv) a network interface (NI). Two types of PEs are used: slave and master. Slave-PEs are responsible for executing application tasks, while the Master-PE is responsible for managing task mapping and system debug. The task repository is an external memory, responsible to store all object codes of applications that will eventually be executed. Each slave processor runs a multitask microkernel...
that enables the communication between tasks through send and receive primitives, respectively called WritePipe() and ReadPipe(). Each microkernel contains a vector, named pipe, which contains messages to be exchanged between tasks. The most relevant features of the system for the DFS controller are task scheduling and the inter-task communication process. Monitoring the scheduler it is possible to evaluate the CPU utilization and monitoring the pipe occupation it is possible to evaluate the communication load.

4.1. Router-PE GALS interface

The present work assumes the NoC operates at a fixed frequency (the reference frequency divided by two) and only processors change their frequency. Therefore, the router-PE interface needs to be modified to adapt to the GALS paradigm, since processor and router may operate at different frequencies due to the DFS scheme. This is achieved by substituting buffers in the NoC and network interface by bisynchronous FIFOs, and introducing two-flip synchronizers in control signals. The router-PE interface contains two buffers (GALS FIFOs), one at the router to receive data from the PE, and the other at the NI, to receive data from the NoC. Besides these hardware modifications, the microkernel was changed to monitor CPU utilization and communication pipe occupancy, storing them in new memory-mapped registers. Based on this information, the controller can take decisions and dynamically change the processor frequency.

5. The DFS Controller Structure

The DFS controller computes the communication load and CPU utilization level according to values provided by the microkernel. Such values are used by the controller to define each PE frequency. The controller always operates at the reference frequency (the highest frequency in the system, used as input to the clock generation module). As the DFS controller works at the reference frequency and the processor at a different frequency, a synchronization scheme between them is necessary. The controller uses the clock generation module, detailed in Section 3, to provide the output clocks. The role of the DFS controller is to choose the correct PE frequency, by evaluating the following parameters:

- Pending message requests from other tasks. This situation takes place when the processor is not producing data to the consumer task.
- Occupancy of the pipe. If the pipe has a high occupancy, the processor is producing messages at a higher rate than the consumer tasks can consume, while the inverse scenario means a lack of produced messages. Upper and lower parameterizable thresholds define the high and low occupancy states, respectively. Occupancy between these values defines an operational state.
- CPU utilization. When the utilization is low the CPU is not executing any task or tasks are blocked, e.g., waiting message(s) from other tasks. When the utilization is high, tasks are using the processor at the maximum rate. Two parameterizable thresholds define high, low and operational CPU utilization states.

Frequency decreases in three situations: (i) the pipe is almost full; (ii) the pipe occupation is increasing, i.e. in the previous evaluation its state was low and the present state is operational; (iii) the pipe occupation is almost empty and the CPU usage is low, meaning that even at a lower frequency the data in the pipe is being consumed. Frequency increases in three situations: (i) existence of pending messages with operational or high CPU utilization; (ii) the pipe is almost empty and the CPU has high utilization; (iii) the pipe occupation is dropping, i.e. in the previous evaluation its state was high and the present state it is operational.

Lastly, when a given PE receives a message request, and it has data to transmit, this PE goes to the reference frequency during the message transmission. This action avoids stalling consumer PEs operating at higher frequencies than the producer PEs.

6. Experimental Results

This section employs an instance of the reference MPSoC with 6 processors (1 Master-PE and 5 Slave-PEs) and a 3x2 NoC to demonstrate the characteristics and advantages of the proposed DFS scheme. A Partial MPEG filter application was used to evaluate the performance of the proposed DFS controller. The partial MPEG filter is composed by five tasks, modeled as a pipeline. The DFS controller was parameterized to generate 9 different frequencies: 5, 10, 25, 40, 50, 60, 75, 90 and 100% of the reference frequency. In the graphic presented in this Section, these frequencies are plotted in the y axis, with values ranging from 0 to 8.

The result for the partial MPEG decoder is shown in Fig. 3. In this application iVLC is a CPU-intensive task, Tasks iQuant and IDCT are simpler than iVLC. Tasks Start and Print are used to initialize the system and to print the results, respectively. In this test case, 200 frames were transmitted.

The graphic shows that only the task executing a high amount of computation had its frequency increased to the reference frequency, while Print and Start tasks had their frequency decreased to the lowest frequency level. The execution time overhead, compared to the execution with the whole system operating at reference frequency was 13%. The number of executed instruction is reduced in 64%.

When the whole system executes with no DFS scheme, the six processors and the NoC operate at the reference frequency. On the other hand, using the proposed DFS scheme, only one processor operates at the
reference frequency, while three other processors and the NoC operate, in average, at half of the reference frequency (including the Master-PE) and two processors operate at the lowest frequency level.

![Partial MPEG filter execution for 200 frames.](image)

Fig. 3 - Partial MPEG filter execution for 200 frames.

7. Conclusion

This work proposes a new DFS scheme and evaluates it in a real MPSoC platform. The frequency scaling scheme is based on the communication load and CPU utilization of each MPSoC PE. The DFS controller uses information provided by the microkernel. A clock generation module was designed to enable frequency changing. This module presents a low-area overhead and requires no stabilization time at each frequency switching. Results show that the DFS scheme adjust the processor frequency according to the load injected into the network. As shown in the MPEG benchmark, the CPU-intensive task has its frequency increased to generate more data to the other tasks. Once the tasks with lower injection rate reach the reference frequency, the system stabilizes, reducing the frequency of other tasks. The proposed DFS method has a small impact in the total execution time. Therefore, an important energy reduction is expected, since few processors of the MPSoC operate at the reference frequency, drastically reducing the number of executed instructions.

8. References