Integration, the VLSI Journal xxx (2018) 1–10

Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

Accelerating cycle-accurate system-level simulations through behavioral templates

Anushree Mahapatra [a], Yidi Liu [a], Benjamin Carrion Schafer [b,*]

[a] Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong
[b] Department of Electrical and Computer Engineering, The University of Texas at Dallas, USA

ARTICLE INFO

Keywords:
Heterogeneous SoCs
Hardware accelerators
High-level synthesis
Templates
Acceleration
Simulation

ABSTRACT

This work presents a method to accelerate the running time of cycle-accurate system-level simulations. The proposed method substitutes the computational units specified at the RT-level or behavioral level in the system with fast templates of the exact latency of its original design and thus, preserving the performance measurement accuracy. The method has been extended to deal with control dependencies inside loops in order to maintain high modelling accuracies under any condition. Experimental results show that our proposed method works well speeding up individual accelerator kernels by up to 8 and 15×. Moreover when used to explore entire SoC configurations it achieves similar result as using the exact models while achieving an average speedup of 4.7×.

1. Introduction

VLSI circuits are reaching complexities never seen before. Most circuits are now heterogeneous Multi-Processor System-on-Chips (MPSoCs), which typically include embedded micro-processors, memory controllers, memories and dedicated hardware accelerators (HWAccs), all interconnected through a single bus or bus-hierarchies.

The Problem that arises while designing these complex Integrated Circuits (ICs) for the system designer is to determine the overall system architecture. For this purpose, they tend to use fast transaction level models (TLM) based on high-level languages such as SystemC, which allows to model concurrent processes. Once the overall system structure has been fixed, the model needs to be refined by using more accurate models, typically cycle-accurate. These models allow to fine-tune the system's performance by e.g. matching the Data Initiation Interval (DII) of certain dedicated hardware accelerators (HWaccs) with the memories' DII and/or the bus bandwidth.

This is particularly important when using C-based VLSI design as High-Level Synthesis (HLS) has one additional advantage over traditional RT-level VLSI design: The ability to generate multiple micro-architecture with unique area and performance trade-offs from the same behavioral description. This implies that a trade-off curve of Pareto-optimal designs can be generated from a given behavioral description. The Problem for the system designer is to determine which micro-architecture to use to meet a given set of constraints (e.g. area, performance and power). Fig. 1 shows an example of the choices that the system designer faces when especially using C-based VLSI design. The SoC depicted contains one master, and four hardware accelerators which compute dedicated tasks. Three of these accelerators are specified in C and thus, given as behavioral IPs (BIPs) and one in RTL, where $HWacc_1$,$HWacc_2$ and $HWacc_3$, are loosely coupled accelerators (any master in the SoC can access them), while $HWacc_4$, is a tightly coupled accelerator that only $Master_1$ can access. As shown, for each BIP a trade-off curve of Pareto-optimal designs is generated (not for the RTL IP as the micro-architecture is fixed). The system designer thus, needs cycle-accurate timing information to determine which micro-architecture to choose for each accelerator, the bus type, arbitration policy and bus bandwidth in order to study their impact on area, performance and power of the complete system. This can only be done with detailed cycle-accurate models, but for large systems this can prove inviable or take extremely long runtimes. Thus, new methods to speed-up these cycle-accurate simulations are needed. The main contributions of this work can be summarized as follows:

- Introduce the concept of behavioral template to accelerate cycle-accurate simulations by abstracting away the functionality of dedicated hardware accelerators, while maintaining their timing accuracy.
- Propose different types of templates based on the internal structure of the accelerator, by investigating data dependencies in order to increase the accuracy of the simulations.

* Corresponding author.
E-mail addresses: anushree.mahapatra@connect.polyu.hk (A. Mahapatra), dylan@connect.polyu.hk (Y. Liu), schaferb@utdallas.edu (B. Carrion Schafer).

https://doi.org/10.1016/j.vlsi.2018.03.014
Received 24 April 2017; Received in revised form 20 November 2017; Accepted 18 March 2018
Available online XXX

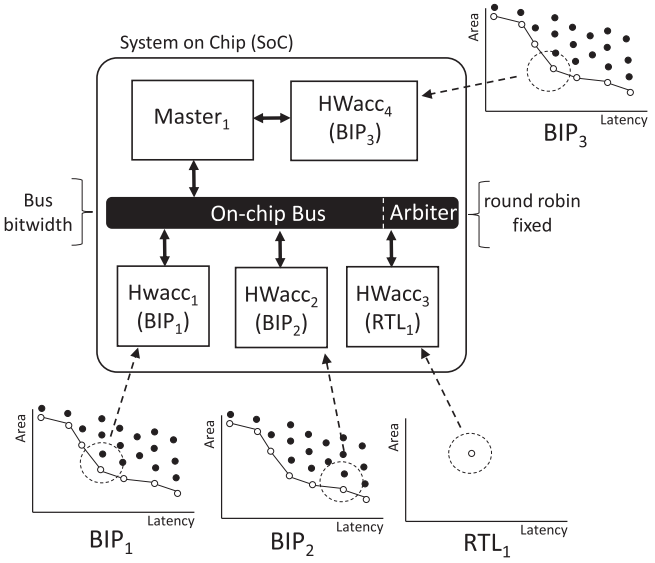


Fig. 1. Heterogenous SoC overview with multiple micro-architectures for each behavioral hardware accelerator.

- Extend a previously developed method to abstract RTL IPs with data dependencies having different execution latencies, into fast behavioral templates proposed in this work.

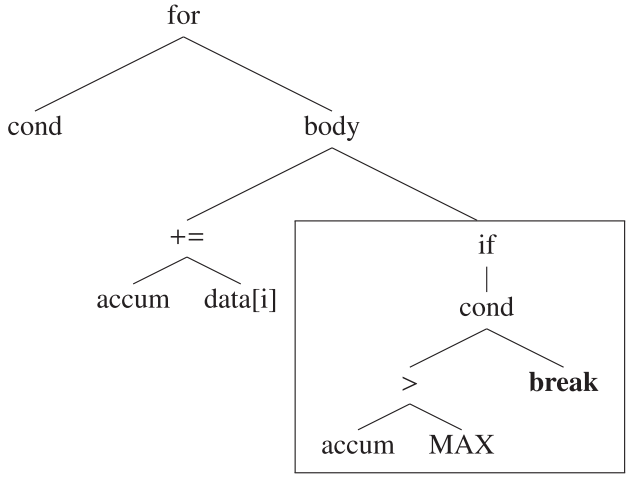
2. Motivational example

One of the problems when exploring complex SoCs, is that although a behavioral-level (e.g. SystemC) cycle-accurate simulation is faster than an RT-level simulation, it is still often too slow, especially when dealing with very large systems. For this purpose, this work introduces the concept of Behavioral IP (BIP) templates to substitute the HWaccs mapped as slaves in the system.[1] The idea behind these templates is to substitute each HWacc (either specified as a RTL IP or behavioral IP) with a template which mimics the IPs' IOs behavior, but is *empty* inside. This implies that it only reads data from the module it is connected to (i.e. a master when mapped as a slave in an SoC) and returns data after X cycles similar to the original IP, where X is the latency of the IP. Although the results returned are functionally incorrect, the timing behavior is preserved. This implies that the system only works if control dependencies within different components in the system do not impact the execution order. Typically, this is the case as these accelerators often perform data intensive applications like image processing or digital signal processing functions. Examples where this work will either not work or loose accuracy are e.g. in medical diagnostic systems, where certain features need to be extracted from the image to be further analyzed, or some scientific applications, e.g. accelerating particle assembly simulations where the coordinates of each particle is updated every simulation cycle and is required in subsequent cycles to compute the inter-particle forces.

This strategy has several key advantages: Firstly, the workload pattern of the entire system is preserved (considering that the master is not using the returned data for control actions). Secondly, the compile time of the entire model is accelerated, as the complexity of these behavioral

```
int accum_until(int *data){
int accum=0;
for(i=0;i<N;i++){
    accum+=data[i];
    if(accum>MAX)
        break;
}
return accum;
}
```

(a) Code example with data dependent break.



(b) Parse tree snippet of conditional accumulator example.

Fig. 2. Accumulator example with data dependent loop execution example.

templates (BT) is much lower than that of the actual IPs (it should be noted that for larger circuits the compilation time can be significant). Thirdly, the cycle-accurate simulation is much faster as each template does not require to perform any actual computation. Lastly, it allows the exploration of configuration of any latencies, hence it is very easy to generate different *what-if* scenarios.

The main Problem is that many IPs often have variable execution latency. Fig. 2a shows an example of an accumulator, which computes the sum (*accum*) of data stored in in array and stops if all the data has been added or a maximum value is reached. In this case, if *accum* reaches a maximum saturation value *MAX*. The latency of the synthesized design is therefore data dependent. In the worst case, the loop is iterated N times, while in the best case, it only executes the loop a single time, i.e, $L = [1, N]$. If this IP is substituted by a behavioral template, in order to maintain the accuracy, the data dependent (*DD*) constructs that affect the latency need to be preserved.

As described in detail later, if a *DD* operation is found, the method builds an abstract syntax tree (*AST*) shown in Fig. 2b and keeps all the code required to resolve this data dependency. In the case of Fig. 2b all the code is kept in the same tree branch (in the enclosed box). In the worst case, this could imply that the BIP can not be abstracted away and thus the BT would be exactly the same as the BIP. The Problem can be formally defined as:

Problem Definition: Given N computationally intensive kernels $K_1, K_2, \ldots, K_N$ to be mapped onto an SoC as hardware accelerators (HWaccs), find an exact cycle-accurate Behavioral Template (BT) for each kernel $K_1 \leftarrow BT_1, K_2 \leftarrow BT_2, \ldots, K_N \leftarrow BT_N$, such that the running

[1] This work makes indistinguishable use of the terms hardware accelerator (HWacc), IP, kernel or slave, for a design module in an SoC that accelerates the computation of a dedicated task.