

ASIP design methodology with Target's Chess/Checkers retargetable tools

Hervé Maréchal
Texas Instruments

BP5 – 821 Avenue Jack Kilby
06271 Villeneuve-Loubet Cedex
France
Tel : +33 4 93 22 14 20
h-marechal@ti.com

Introduction

Hardware accelerators are commonly found in complex platforms. They assist in computing intensive tasks such as modem physical layer or multimedia codecs, which a central, general purpose processor could not perform quickly enough on its own. Such accelerators are typically hand-designed, hard-wired, and their very dedicated nature and instruction set usually doesn't allow them to be programmed in a high-level programming language.

SoC designers face the constant evolution of standards and algorithm choices. Software implementation decisions are not completely settled until after SoC design is frozen.

This situation has naturally led to the emergence of ASIPs: Application Specific Instruction set Processors, i.e. of dedicated-yet-programmable coprocessors.

The Target Chess/Checkers tool suite

Accompanying the ASIP emergence, a number of ASIP design tool have appeared on the market. Research works based on high-level synthesis and retargetable compilers technology, have made it into commercially available products.

After evaluating several of them, TI has selected and is deploying the Chess/Checkers tool suite, from the Belgium-based Target Compilers Technology Company.

This tool suite is based on Chess, a retargetable C compiler, and Checkers, a retargetable instruction set simulator (ISS). Other tools, such as retargetable assembler/disassembler, linker, and an automatic RTL code generator and a random test pattern generator, complete the package.

Using an architecture description language (ADL) named nML, one captures a model of an instruction set architecture, including hardware resources description (registers, memories, pipeline registers, ...), pipeline behavior, and instruction set, assembly syntax, and binary encoding.

“Retargetable” means that the tools (including the C compiler) are targeted towards the architecture so described in nML. There is no restriction to the type of ISA that can be modeled: RISC, CISC, SIMD, VLIW, integer and floating-point architectures are supported. All tools take the nML model into account, and the retargeting process is very fast (a couple of seconds suffices to retarget the compiler, simulator and all other tools). Based on a number of patented register allocation mechanisms, the Chess compiler not only adapts to many different types of ISA, but also impresses by its superior performances. Complex C patterns can be turned into hand-designed level assembly code, with and even without using intrinsics.

The possibility of concisely capturing ISA models and generating in seconds a

powerful compiler and a cycle accurate ISS opens up new perspectives in the exploration of architectures. Multiple alternatives and high-level choices can be tested out and factually compared, allowing designing well-suited, well-crafted C programmable processors. Not only this answers the accelerator flexibility issue, but this also changes the design flow, and the resulting accelerator quality.

Finally, this tool opens the door to system level simulations, by having a System C wrapper be automatically generated around the ISS, making it possible to glue the ISS to other modules (caches, memories, buses ...) and performing system level analysis.

The retargetable tool design flow

The traditional approach to developing accelerator was essentially hand-made RTL based: suppose additional or even only slightly varying requirements arriving in the middle of the design, modifying hand-written RTL proves to be both painful and risky, as the conceptual gap between specifications and RTL code is high. Similarly, tuning and optimizing the architecture for more performances (i.e. in this case less cycle counts) is equally risky. Thus, it often leads to less-than-optimal designs, possibly compromising requirements versus development time.

In contrast, the new design flow offered by the Chess/Checkers suite is more flexible and allows focusing on the real issues at the architecture level. For instance, adding a pipeline stage or a register file is extremely easy and fast. The cycle time between modeling and simulation is far shorter. This tool thus truly offers design space exploration and hardware-software co-design. The ASIP architect now has multiple degrees of freedom to experiment with.

The first step of the flow consists of writing kernel functions of the algorithm to accelerate in C language. By analyzing requirements (e.g. area constraints, memory

environment, etc ...), the architect figures out an initial accelerator version which he then captures in nML language.

Next, he can build the kernel functions with the Chess compiler, and simulate them with the Checkers ISS. Bottlenecks and issues are thus discovered early. At this stage, any number of iterations is possible between the C kernels and the nML architecture model, so as to remove sub-optimality, until software performance and feature requirements are met.

In parallel with C and nML (i.e. architectural) optimizations, RTL code can be automatically generated. Using regular synthesis tools, hardware parameters can be quickly (or deeply) extracted: area, power consumption, timing violations, etc ... On the hardware side too, corrections can be brought to the architecture until requirements are fulfilled. Figure 1 below illustrates this flow.

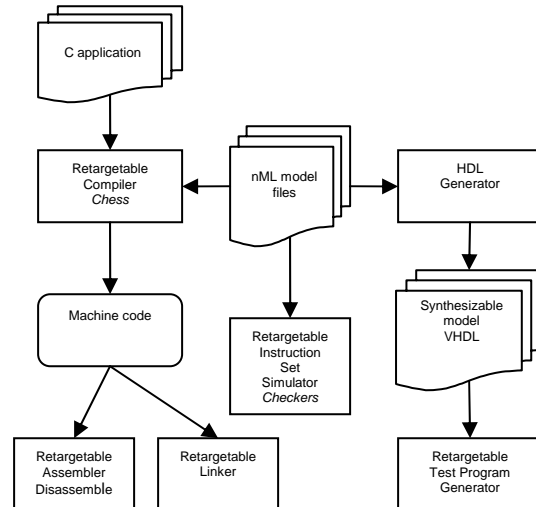


Figure 1: Chess/Checkers based design flow

In summary, the new design cycle time is considerably shortened, allows to take into account many parameters very early that could not be easily accessed before, and also allows better interaction between the ASIP architect and the future software developers,

resulting in a design which is well suited to its use cases in terms of features, performances, etc ... Lastly, shorter cycle time also implies savings of money and greater competitive value.

Exploration process

As mentioned before, the Target tool set allows capturing a coprocessor description quite simply. However, there remains the issue of figuring out the architecture in the first place. Unfortunately, a CAD/EDA tool that would assist the architect in understanding an application and would offer architecture hints or even dump a model of it don't seem to exist.

The assumption in our approach to exploration is that algorithms determine much of the architecture. This means that understanding the application to support gives a lot of clues about what the processor to run it should look like. This obviously particularly applies to the hardware accelerator field. In fact, our belief is that the process by which an architect works out an initial ISA could be, to an extent, automated, borrowing from known techniques in the compiler area.

Indeed, what architects do, after writing kernel functions code, is to see what operations could be done simultaneously. Adding to this picture memory interface and bandwidth constraints, one usually ends up with a coprocessor ISA that performs multiple operations in a clock cycle, e.g. computing one or several memory addresses, fetching data, calculating something out of those data, incrementing some pointers, and writing back values in memory.

We have thus developed a prototype tool which parses C code and creates an abstract syntax tree (AST) then builds up a control and data flow graph (CDFG). Then specific analysis is performed, within and across basic control blocks, and a sequential tree of simultaneous operations is produced.

Figure 2 below illustrates the exploration process.

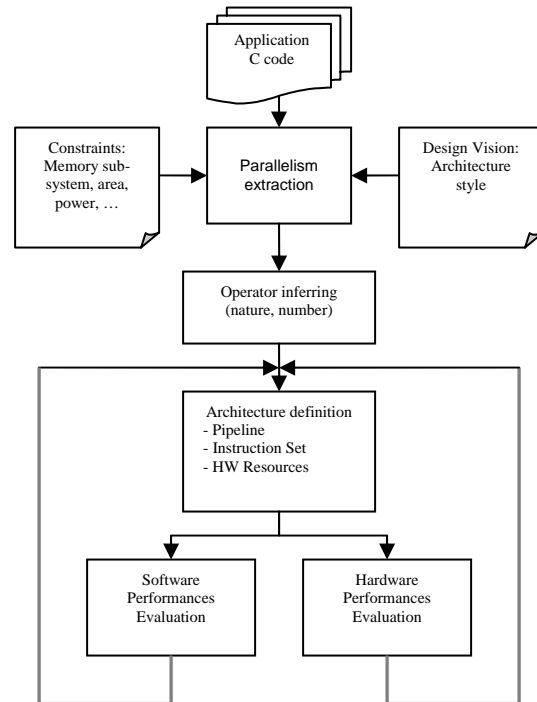


Figure 2: Exploration process

This tool helps out the architect in his creative job. In the future, the tool scope will be augmented and its performances will be improved, so that it also suggests operators and pipeline operation sequence.

TI coprocessor strategy

Developing coprocessors, whatever the flow and tool to support it, remains an expensive task. Besides, the skill set for an architect to use ASIP design tools such as Target tools is a mix of software and hardware competencies not commonly found among design engineers. Lastly, there is a trend towards more and more IP re-use in order to decrease development cost and time.

As multiple coprocessors need to be designed for next-generation wireless platforms, the new flow now in place at TI

allows reusing more elements from a coprocessor to another.

Compared to solutions of configurable and extensible coprocessors that exist on the market, this approach has several advantages. First, TI has full control of the ISA choices, and does not have to re-use a too generic, possibly too large, base architecture. Next, TI can develop and integrate it to a specific memory sub-system. Moreover, the Chess compiler features very good performances. Finally, extensible IP's business model is usually royalty based. As TI ships very large volumes, this can prove to be very expensive and would degrade profit margin.

Video accelerator architecture and modeling

Video codecs and standards are plentiful, making programmability a key feature. The advent of high definition resolutions puts an extremely high computing speed requirement.

Using the Target tools, a video coprocessor model is now being defined. Its purpose is to accelerate motion estimation and its use cases include video encoding and decoding.

This coprocessor features a rather generic base instruction set. In addition, dedicated resources, directly under control of the generic side, allow accessing a wide memory interface, and process many data in parallel through SIMD operators. Special focus is brought to parallelism, and the IPC ratio (Instruction Per Cycle) and control instructions latencies have been carefully studied for high performances.

This results in a small yet very powerful coprocessor well suited to its environment constraints (memory interface) and which can be tailored to specific needs by customizing the number and nature of dedicated SIMD operators.

Comparisons with a functionally equivalent earlier hand-made design show that area is

within 10% of overhead. This accounts for the immense added flexibility which the previous design didn't have. Moreover, as the design is more independent and can work out more things on its own, thanks to its generic base instruction set, it can actually also carry out a number of tasks on its own (such as control oriented decisions between two intensive algorithmic processes), thus reducing interactions with a master controlling processor. This way, the little area overhead is turned in an advantage at system level, because the sequencing processor can be chosen less powerful, and because overall processing is interrupted less often.

For software developers, there is the added bonus that they can program the whole codec in C at a very early stage of the coprocessor design. This design also comes with an extensive test suite, including specific tests for validation of dedicated operators.

Overall, design time is shortened so less money is spent in designing elements, and competitiveness is high as this coprocessor can arrive quickly in a product on the market.

Conclusion

Programmable ASIPs, and design tools such as Target's suite, have a full range of advantages over traditional accelerators: flexibility, powerful programming tools, and automated RTL generation.

TI is successfully putting in place a design flow centered on Chess/Checkers, together with an internal architecture exploration methodology.

Finally, a coprocessor strategy based on these tools offers important benefits: flexibility, validation, elegance, resulting in lowered development cost and design time, and better match to ever changing requirements.

References

1. *“Chess/Checkers: a retargetable tool suite for embedded processors”*, White paper, Target Compilers Technology.

“Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted, provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers, or to redistribute to lists, requires prior specific permission.

GSPx 2006. October 30-November 2, 2006. Santa Clara, CA. Copyright 2006 Global Technology Conferences, Inc.”