

Catalyst: A DSIP Design Flow Development in Industry.

W. De Rammelaere, K. Eckert, E. Hilken, T. Lawell, R. McGarity, P. Le Moenner, F. Steininger
Motorola SPS

31000 Toulouse, FRANCE

Werner.De.rammelaere@motorola.com

Abstract

The Motorola System on Chip Design Technologies (SoCDT) team aims at providing a system design environment for its customers. The Toulouse branch concentrates on design efforts incorporating DSP functionality. This is referred to as the Catalyst methodology. We found that in current systems very often the software development cycle is longer than that of the silicon development. To ease the software burden, we have changed the silicon architecture and its flow to permit the DSP software to be written in the C language instead of assembler code, as is normally done. The resulting architecture is domain specific; it is smaller, has a reduced design cycle and is simpler to implement because it is tuned to the application software we are providing. This paper will describe the methodology which we are developing to create domain specific architectures, it shows one example architecture and aspects which are critical for industry acceptance.

1. Introduction

This paper describes the Catalyst methodology to design domain specific instruction set processors (DSIP). First we indicate the problems one typically encounters when designing a system on chip (SoC). Then we describe today's design methodology followed by our Catalyst approach and its advantages. We then become more specific, describing some implementation and verification aspects, and focus on some industry-specific requirements. Subsequently we indicate our results on an architecture we have implemented to test the new methodology and its related architectural ideas. We close with a list of hurdles and solutions for the acceptance in an industry environment.

2. The Problems to Solve

The problems we found by talking to our customers are common to many marketplaces. First, the market requirements tend to be vague and change during the project. With a long design cycle, a system is often outdated by the time it is completed. Second, the marketplace is tiered. The hardware and software requirements for the high end are different from those of the low end, so different design points are required. Third, there are tight cost and

power consumption constraints. Lastly, the software effort is greater than the hardware effort, especially because traditional methods use assembly language for the DSP algorithms.

To solve these problems, we saw the need for our design methodology and architecture to have these properties.

- The hardware design cycle must be reduced
- The software design cycle should be reduced
- Architectural flexibility is key: It should be easy to produce different versions of a basic hardware design; it should be possible to introduce design changes as late as possible in the design cycle to react to new market requirement or a tiered market.
- A compiler and simulator should be available in advance of silicon.
- We must provide efficient verification of software and hardware, as early as possible and as fast as possible.

3. Methodology Improvements

3.1 Today's Methodology

In industry the rather traditional approaches to design chips have not changed drastically when being confronted with the more complex, heterogeneous SoC designs requested today. The flow is depicted below.

First, the design team develops algorithms which meet the market requirements. Second, the algorithms are

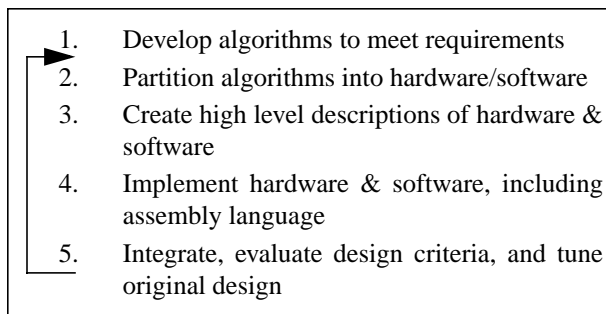


Figure 1 - Today's Methodology

partitioned into hardware and software parts. At one extreme, the partitioning could result in one or more general purpose CPU and/or DSP processors running software

written in both C and assembler. At the other extreme, it could result in specific hardware modules for specific application components with various degrees of programmability.

Third, the design team further specifies the software and hardware by writing high level descriptions of them. For the hardware this is typically documentation in the form of text and figures. For the high level protocol software it is the same, or perhaps a high level description written in the SDL[1] language. For the assembler level signal processing software, the high level description is typically C.

The fourth step is to, in parallel, implement the hardware and software. For hardware, the high level descriptions are typically converted to Verilog or VHDL, while the software is written in C (if possible), or assembler (if necessary to meet performance requirements).

The final step is to integrate the software and hardware, evaluate the important design criteria (performance, cost, power consumption, etc.), and tune the original design if the criteria are not met.

3.2 Disadvantages of Today's Methodology

We see several problems with today's methodology. First, after the initial partitioning, the hardware and software are designed in isolation. Second, the performance critical software which is targeted to a DSP or specialized hardware unit must be written in assembler.

Third, the validation loop is long since the hardware must be built, the software must be written, and the two must be integrated to obtain good feedback. So design errors made in the early stages are not found for a long time. Also, it is difficult to tune a design to improve performance, reduce power, or reduce cost, since the effects of changes are not fully seen until much later.

The validation loop is so long that the original market requirements may have changed by the time the feedback is received.

Because of the enormous effort to make (full-custom) hardware designs, industry often decides to stick with a multi-core implementation, using hard IP cores (for voice codec applications often a microcontroller and a DSP core) as components, since soft IP cores are mostly non-existent in the DSP arena. The design time gained doing so is only relative: an army of hardware designers is needed to adapt the cores to the system requirements and integrate them, as well as an army of software programmers to write hand assembly for all critical parts. It is one of the reasons for critical path of the software in the flow. In addition, system trade-offs are kept minimal, and non-core modules are avoided if possible since they complicate the integration process. Lacking efficient codesign tools it is a safe (and well known) but labor intensive route to choose.

3.3 The Catalyst Approach

We rethought this methodology and chose a different one which avoids or greatly reduces the problems above (see Figure 2). The first step is identical to that described in section 3.1.

In the second step, the software which is critical for the design criteria is implemented. It will allow us to measure (and adapt) the efficiency of the implemented hardware.

The third and fourth steps are identical to the second and third steps of 3.1.

By using a retargetable compiler, such as Chess[2], described in section 3.5, it is possible, in the fifth step, to measure the performance and gauge other design criteria. This enables a designer to, in the sixth step, improve the hardware and/or software, or repartition the algorithm between the hardware and software. He then repeats step 5, or perhaps returns to step 1 if the results are far from the requirements. When the team is satisfied with these results, it implements the hardware and writes the remainder of the software, all in C.

We provide a fast path to implementation by using an automated Verilog generation process, use of standard cell technology and advanced verification schemes for both software and hardware, with a maximum of verification prior to silicon processing

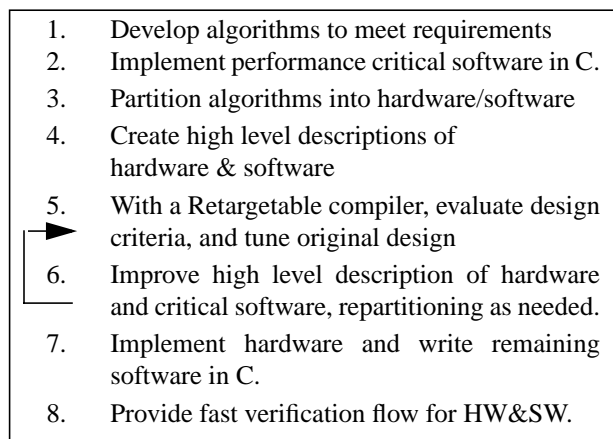


Figure 2 - Catalyst Methodology

3.4 Advantages of the Catalyst Approach

Several advantages result from this approach. First, the hardware and critical software are designed in tandem, not isolation. Second, virtually all software is written in a high level language (C). Third, the validation is short - typically hours instead of years. Fourth, a C compiler and ISS¹ are available quickly, as will be seen in section 3.5. Lastly, the

¹. Instruction Set Simulator

'back end' of the design itself is very automated, leading to a fast design cycle, and the necessary flexibility alluded to before. In short, this new methodology addresses many of the problems we noted with today's methodology in Section 2. We will discuss the flow in more detail below.

3.5 The Target Compiler Technologies Tool Suite

The Target Compiler Technologies tools suite forms the heart of our design environment. It is oriented to support DSP embedded processor design, and consists of three major components:

- A retargetable compiler (Chess[2]) that translates a C-based source code program into highly optimized machine code. The compiler copes well with the architectural peculiarities of fixed-point DSPs, including specialized instructions and heterogeneous register structures.
- A retargetable ISS (Checkers), which is able to simulate the execution of machine code in a cycle and bit accurate way. Simulation speed is comparable to a high level C simulation. It can either be executed in a stand-alone mode or be embedded in a co-simulation environment with existing simulators.
- A retargetable assembler and disassembler (Darts).

Retargetability means that the compiler, ISS, assembler and disassembler can be easily reconfigured for a new processor architecture. A processor architecture description is written by the designer in a high level language called nML[4]. The Target Compiler Technologies tools read the nML description, and use it to reconfigure the compiler, ISS, assembler and disassembler. So the software development tools are ready as soon as the designer changes the high level nML description. The level of abstraction of nML is very high, making it possible for the designer to make significant architectural changes quickly. For example, a description of a DSP architecture is typically just a few thousand lines of commented nML.

The Chess tool also provides outputs which show the designer where the program is spending its time, allowing him to alter either the program or the nML processor description to improve performance. Figure 3 shows how this works. The designer modifies the architectural description or source code, and reruns Chess. For example, he may add a special instruction or address unit to eliminate a bottleneck. Chess reads the architectural description and the source code, and applies numerous optimizations to efficiently map the C code to the architecture. This optimization step typically takes a few minutes, but may take a few hours for more complicated algorithms. The

designer receives statistics which show how many cycles the software algorithm requires, and where the bottlenecks are. He then alters the hardware and/or the software description and tries again..

In the end, he can create a DSP-like architecture which has very good performance even though the source language is C. This design loop would take months or even years with the traditional approach.

While the Chess tool gives the designer a great deal of freedom in defining an architecture, there are some restrictions. First, the architecture must be load/store. Second, for the compiler to optimize most effectively, a pipelined architecture should be replaced with a simpler horizontal architecture. In a sense, the pipeline is modeled in software, leading to additional simplicity in the hardware architecture. As will show in section 4.1, these proved to not be significant restrictions in practice.

In summary, the Chess tool gives the designer immediate feedback, it permits DSP software, even the critical inner loops, to be written in C, and it makes new, specialized instructions readily accessible to the C code, often without having to make any changes to that C code.

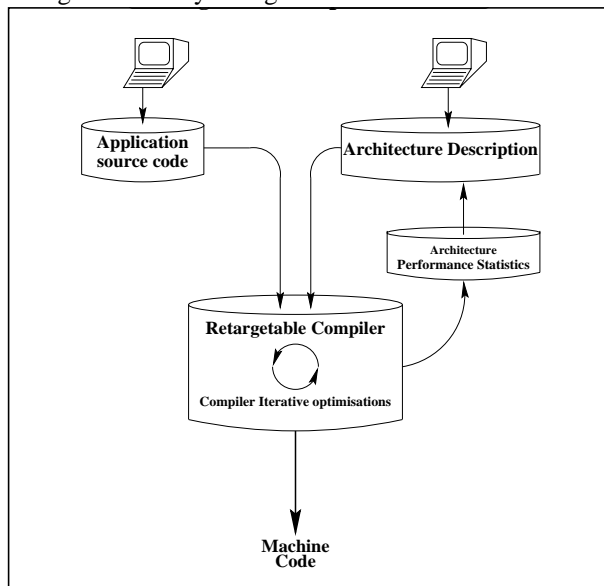


Figure 3 - Trade-off flow

3.6 An Efficient, Automated and Verifiable Flow down to Silicon for HW, Assembly for SW.

Though the high-end part of our flow is made possible by using the Target tools, Motorola needed to add several key features to make it an industry-usable methodology:

- Customers impose the availability of a debugger environment for the cores. This necessitates a hardware debug unit, the use of software layers to talk to the debug pins on the chip, and Elf/Dwarf2 compliant high-level

tools. Motorola India has written a GUI and the necessary software to communicate with the hardware debug unit.

- Motorola Australia has written a nML to Verilog generator, which generated RTL level verilog out of the final nML description. The software is written in such a way that advanced scripts can be used to manipulate the verilog for more optimal hardware.
- Standard cell design is a well established design methodology, but with area and timing limitations compared to full custom design. Motorola SPS, known for its silicon quality, has key IP that enables us to minimize the effects of these limitations.
- We have added a complex verification scheme to the design flow: we use RIS (Random Instruction Sequences) verification to compare Checkers results to Verilog simulations, PRPG (Pseudo Random Pattern Generation) to compare instruction primitives at the C and verilog level, formal verification between RTL level and gate-level netlist, a TCP/IP based environment to verify the interaction of debugger action and functional core simulations, etc.

We believe these features make the distinction between a research environment and adoption of this research in industry. Our customers impose fully functional silicon on a meaningful testchip example before they are willing to adopt the methodology for their application domain.

4. An example: The LSE TestChip

4.1 A DSIP for Voice Codecs such as G723, G729, EFR and HR.

To evaluate the flow we have decided to design a LSE (Load Store Engine) testchip for the application domain of voice coders: the ETSI and ITU standard bodies provide ANSI-C code for a number of popular voice codec algorithms such as G723, G729, EFR¹ and HR², along with testbenches to verify the code. Since the given algorithms have been implemented on various popular DSP engines, they provide a nice test case and allow for comparison with real products. Limited features were added to the architecture to support equalization and channel decoding.

The Catalyst team explored several architectures, using the methodology described earlier in section 3.3. We

¹. Enhanced Full Rate GSM

². Half rate GSM

searched for ones which best met our anticipated market requirements. The design was done in close interaction with the WSSG³ group in Toulouse, to guarantee useful and meaningful results. Working at a high level of design abstraction, we were able to quickly make such trade-offs as:

- addition and removal of instructions
- variation of the number of registers (data registers, base address registers, offset registers, and hardware loop registers.)
- the implementation of the MAC⁴ unit
- data width of the ALUs
- the total number of ALUs
- number of read / write accesses to the data memory
- bundling of instructions. A dedicated instruction can be added to the architecture that groups consecutive instructions.

After about 5 months of C profiling and architectural trade-offs, we finalized on an architecture with the following features:

- An instruction width of 16 bits to provide competitive instruction code density for both DSP and control code. Almost all instructions execute in 1 cycle, the other ones (e.g. jump) in 2.
- Two single cycle MACs in parallel with two loads from data memory and two address computations to guarantee competitive performance for classic DSP functions.
- Additional hardware in the address computation unit to support close loop searches in speech coders, to reduce the number of cycles in this critical function.
- Dedicated hardware in the datapath and the address generation unit to guarantee support for equalization and channel decoding (a Viterbi processor)
- A BMU⁵ to reduce number of cycles for CRC⁶s, interleaving, firecodes, etc. Running these kind of codes on a classic controller CPU could result in an overhead of up to 400:1.

³. Wireless Subscriber Systems group

⁴. Multiply-Accumulate

⁵. Bit Manipulation Unit

⁶. Cyclic Redundancy Code

- Support of all basic control instructions, to provide good code density and support for the controller-like protocol software. Most of these instructions are capable of executing in parallel a memory access.
- An instruction set which is mainly orthogonally encoded, and has headroom for future instructions. This gives us the ability to react to the inevitable late changes in, and additions to, the requirements.

We have an architectural description of this LSE architecture written in the nML language which has most of the features described above. It consists of about 2000 lines of commented nML.

We have generated RTL level verilog code from this nML in a semi automated way, we have added data and program memory, a debug unit, and some peripherals.

4.2 Results

After several months of work in the back end flow, we have produced fully functional silicon, running at 100Mhz at 2.7V, and 65Mhz at 1.8V. The core size is less than 5mm² in 0.35um technology. Table1 shows the performance of the LSE and the Motorola Onyx DSP. The results for the LSE are for compiled code taken almost directly from the standard body on EFR. Onyx numbers are benchmarks for optimized hand-assembly code. Both results can still be

Arch	Mips	Code Size
LSE	24	20KB
Onyx	17	16KB

Table 1 -Performance

improved by adapting the C code more towards the architecture.

4.3 Advantages of the Catalyst Approach

We see many advantages to this architecture approach, which we list below.

- **Reduced Design Time**

We believe the design time is greatly reduced, for several reasons. First, the LSE architecture reduces the complexity and size of the hardware, especially the control logic. The net effect is that almost all of the control logic is combinatorial rather than sequential. This simplifies the implementation, and greatly simplifies the verification task.

So the total effort required to create a functionally correct RTL¹ level model is greatly reduced. Second, as we explained in Section 3.6, we have written smart translators from nML to Verilog, and we have set up an advanced verification methodology to cover the design flow itself.

Third, the layout is entirely synthesized from the RTL model, greatly reducing the time required to convert the RTL model to layout.

Fourth, the simplified architecture permits the efficient use of Chess, so an ISS is readily available for verifying the design.

Finally, the single-cycle execution combined with memory mapped peripherals simplifies the design of the processor. No special instructions are required for use when the processor communicates with a peripheral.

- **Complexity Moved to the Compiler**

The horizontal instruction set moves complexity from the hardware design to the compiler, reducing the size and complexity of the hardware. For example, a traditional way to achieve high performance is to create an instruction which effectively launches a series of operations, each in a succeeding pipeline stage. An example is a DSP multiply-accumulate instruction. A typical implementation of this might require the following pipeline stages, with associated control and state:

- Calculate and update effective addresses
- Fetch operands
- Multiply
- Add multiply result to accumulator

With the LSE architecture, each of these is a separate field in the instruction. The compiler coordinates this activity by properly setting up these fields in subsequent instructions. Effectively, the pipeline state is embedded in the instructions themselves, significantly simplifying the LSE hardware.

- **Smaller Implementation**

By reducing design complexity, the hardware simplifications mentioned above also reduce the total amount of hardware in the processor, especially that of control.

- **Can More Easily Modify**

Since LSE is designed with Chess in mind, since the instruction set is as orthogonal as possible, since the LSE design is inherently simpler, and since the layout is completely synthesized with no custom blocks, it is much easier to make changes. So when the market requirements change after a project is launched, the hardware can be more easily altered to meet these new requirements.

¹. Register Transfer Level

- **Quickly Map to New Process Technologies**

Since the design is entirely synthesized with no custom layout, it can be moved to a new process technology as soon as the underlying standard cell and RAM libraries are available in the process.

- **Good Performance**

We believe that an improved LSE architecture running compiled C code should require only slightly more cycles than a traditional DSP running hand crafted assembler code: The specialized instructions and datapaths which are added improve performance. We believe that this improved performance will offset any inefficiencies which might be introduced because the software is written in C.

5. Acceptance in the industry.

Since we have demonstrated the technical and business related advantages of the domain-specific approach, one may wonder why the flow has not yet found wide adoption within major semiconductor players such as Motorola. The major hurdles we need to tackle to gain acceptance in the industry are twofold:

- A mentality shift within the organizations of the major semiconductor players towards a more open, reusable and flexible design environment. While the shift from a closed full-custom based design flow into a more open and reusable flow adopting standard cell design methods is difficult and time consuming. Motorola is taking it very seriously to continue to be a key player in the SoC arena. The release of the MSRS¹ is Motorola's recent answer to SoC design.
- Offer a solution to the questions of support for such architectures which embed a risk of creating too domain specific cores. A majority of the questions concerning our approach are related to this issue, and it is a valid concern. We think we can offer valid solutions to these concerns; both technical and non-technical:

1. At the hardware side, we have designed the debugger environment as a flexible, core independent module.
2. The software system of the source-level debug environment is developed in such a way that different DSIPs have the same look-and-feel GUI and an identical debug environment. This greatly simplifies the job of support engineers.
3. All tools in the flow of the DSIP core itself are developed such that they are/will be

¹. Motorola Semiconductor Re-use Standard

retargetable and of industry quality. This is vital for success, since it allows support engineers to be relatively independent of the target architecture. We are currently putting more effort in this area.

4. Discipline is needed: in the same way standards for good quality ASIC design are developed and followed, one needs to adhere to certain rules in DSIP design. They key is to define the target domain before designing the DSIP.

6. Conclusions

We believe that, for the application area of embedded systems on silicon, these Catalyst methodology and architecture improvements reduce total system costs, reduce hardware design time, ease the effort to port to a new process technology, virtually eliminate the need for writing DSP software in assembler, and enable Motorola to react more quickly to market changes. As the Catalyst project progresses, we will further demonstrate this new approach, and convert it into a Motorola strategic asset.

7. References

- [1] R. Braek and O. Haugen: *Engineering real time systems: an object-oriented methodology using SDL*, Prentice Hall International, 1993.
- [2] G. Goossens, I. Bolsens, B. Lin, F. Catthoor, "Design of heterogeneous ICs for mobile and personal communication systems, *Proc. IEEE ICCAD-94*, November 94, pp. 524-531.
- [3] Michel Mouly, Marie-Bernadette Pautet, *The GSM System for Mobile Communications*, Cell & Sys, 1992.
- [4] A. Fauth, J. Van Praet, M. Freerick, "Describing Instruction Set Processors Using nML". *Proc. European Design and Test Conference*, Paris, March, 1995.