

Exploiting Parallelism, while Managing Complexity using Silicon Hive Programming Tools

Geoffrey F. Burns, Marco Jacobs, Menno Lindwer, Bertrand Vandewiele
 Silicon Hive, High Tech Campus 45, 5656 AE Eindhoven, The Netherlands
 {G.Burns,Marco.Jacobs,Menno.Lindwer,Bertrand.Vandewiele}@philips.com, <http://www.siliconhive.com>

Abstract

A comprehensive development environment, including the HiveCC¹ spatial compiler, supports the Silicon Hive portfolio of processors. HiveCC is specifically targeted for each processor cell, and solves the problem of exploiting parallelism in Silicon Hive ULIW¹ processors. An integrated development environment manages the project source files, builds makefiles, and drives the HiveCC ANSI C compiler, debugger, and multi-core simulator. In this paper an overview of the programming environment, with a focus on the unique compiler technology is presented.

I. Introduction

Short time-to-market, programmability, flexibility, risk management, complexity, and cost reduction, enabled by the Silicon Hive technology, are only possible because the “triangle” system architecture, processor architecture design, and compiler is taken into consideration [7]. In [1], an overview of the technology is given and in [2], the processor template is described. This paper describes the software programming tools and explains how the HiveCC compiler exploits the high level of parallelism present in the processor template.

II. Software Development Flow

The entire Silicon Hive tool set is integrated into an Integrated Development Environment (IDE) shown in Figure 1. The HiveCC compiler is at the heart of the IDE. HiveCC is the first commercially available compiler, based on the principle of constraint driven instruction scheduling [6].

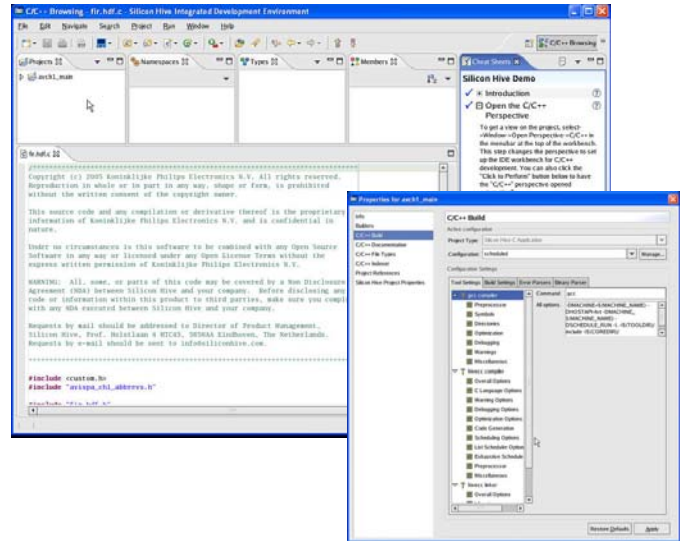


Figure 1: Silicon Hive Integrated Development Environment.

The toolchain offers different levels of simulation and verification. Abstraction levels differ in timing accuracy and simulation/execution speed. Simulation/execution levels are shown in Table 1.

Table 1 Different abstraction levels for compilation and simulation/execution

Abstr. Level	Hive code compiled to	Exec. method	Exec. speed	Acc-uracy
c-run	Sim. Host	Native	Native	Func
tm-run	Hive ops.	Native	>3 Mops/s	Bit
sched	Sched,ops.	Native	3 Mops/s	Cycle
binary	Assembled	ISS	~100 Kops/s	Cycle
RTL	Assembled	RTL	~100 ops/s	Signal

At all but the last level, the controller code runs on the simulation host and gets linked with the Hive application code. For c-run, the Hive code gets directly compiled (through gcc) into native simulation host code. This results in a native simulation host executable for the combination of embedded host code and Hive processor code. It serves merely to verify functional correctness of the algorithm implementation. For tm-run, the

¹ Moustique, Avispa, Bresca, HiveCC, ULIW, and SHAPE are trademarks of Philips Electronics NV.

- Green blocks indicate the utilization of a given issue-slot (10 first lines) for a given program-counter,
- Red blocks indicate the utilization of an interconnect line,
- The utilization of each register is shown for every program-counter position by a color code indicating whether a given register contains a live value (blue), whether it is written to (red), read from (green), or both written and read (yellow) in a given instruction.

III. HiveCC compiler

From the prior section, it is clear that the HiveCC compiler is at the heart of the software development flow.

The HiveCC compiler is used for all cores (run time re-targetable). The instruction level parallelism (ILP) in the application is extracted with the operation scheduling and resource allocation.

Multi-core applications with different level of parallelism can be easily addressed, because all cores are designed based on the same architecture template and are accessed through the same API.

A/ ULIW processor template

High performance at low power is enabled by the parallel architecture of the ULIW processor template and the low clock frequency [2]. It provides multiple threads of control (cell-level multi processing), each processor cell having a multitude of issue slots, localized register files, memories, and interconnects.

As depicted in Figure 4, unlike conventional processor architecture, Silicon Hive processors do not have hardware overhead for instruction decoding, pipeline control and hazard detection. Scalability is possible thanks to the limited hardware overhead (limited interconnect and locality of references with distributed local memories) and advanced compiler that controls all the resources separately.

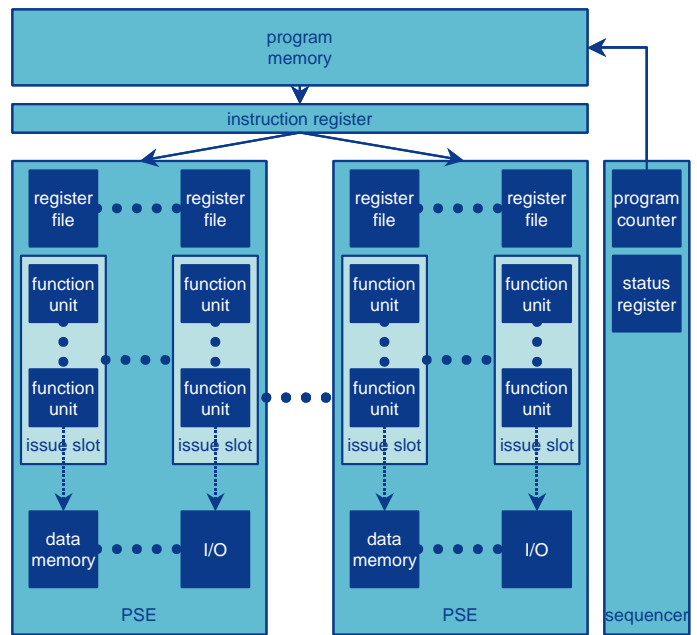


Figure 4: Processor Architecture Template

Figure 5 shows an actual portion of an Avispa-CH1 processor used for (de-)modulation of OFDM-based communication standards.

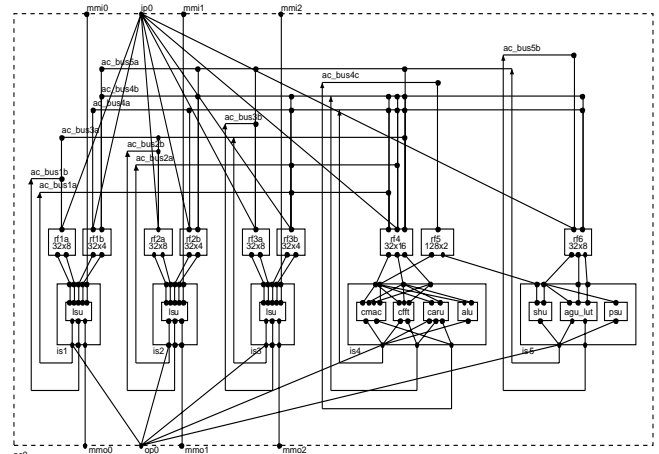


Figure 5: Avispa-CH1 complex DSP Processing Storage Element architecture (5 issue slots, 10 function units, 9 register files)

B/ HiveCC key features

- **Scheduler**

HiveCC uses deterministic constraint solving scheduling techniques that are capable of dealing with massive parallelism (dozens of issue slots), distributed register files (over hundred), and partial, constrained interconnect.

Given sufficient compilation time (~ few minutes), the generated code is guaranteed optimal.

The compiler automatically not only schedules all operations in time, (i.e. temporal assignment task that is classical in compiler technology) but also in space with an aim to maximize locality of reference. HiveCC allocates all resources of the architecture. For example, HiveCC takes into account the distributed register files and memory, the high level of parallelism provided by the numerous issue slots. (See Figure 4 and Figure 5). This “space” compilation is analogous to what hardware synthesis tools do and it is unique for a processor compiler.

When required, HiveCC offers full control over its resource utilization to the programmer. The programmer can control allocation of data structures to memories, of operations to issue slots, or the instructions themselves. It also offers the use of application specific operations present in the architecture in the form of intrinsic functions at source-code level (e.g. FFT butterfly).

Figure 6, depicts the different levels of programming, simulation, and validation as explained earlier.

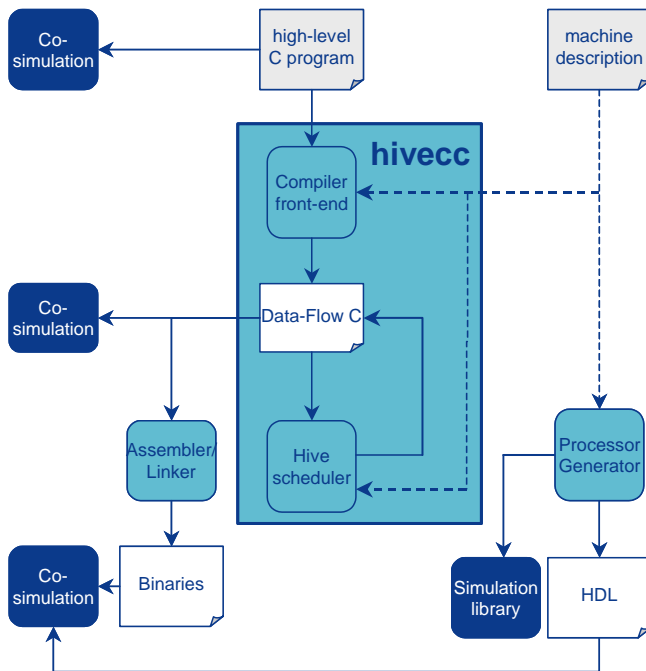


Figure 6: The HiveCC spatial compiler

For different levels of optimizations, two different schedulers can be used. The “Hivesched” list

scheduler is a fast scheduler that can do software pipelining. It is fast but not optimal. For optimal scheduling, the “Manifold” scheduler has to be used. The dark boxes in Figure 6 show the various feedbacks and outputs given to the programmer.

- **Level of parallelism**

The combination of HiveCC and Silicon Hive processor, supports multiple levels of parallelism:

- 1- SIMD operations: Like MMX: 1 instruction operates on multiple data packets (4/8 bytes, 2/4 half words, etc.)
- 2- Instruction Level Parallelism (ILP): Multiple instructions execute in 1 cycle (E.g. up to 5 instructions inside 1 Avispa-CH1 complex DSP PSE, Figure 5)
- 3- Multiple PSEs: E.g. 1 Base processor PSE and up to 4 complex DSP PSE in Avispa-CH1
- 4- Multiple cells / processors: E.g. Bresca [4,5]
- 5- Multiple cores: E.g. TV on Mobile chips has 6 cores, from 3 different families [4]

IV. Application Examples

A/ Running an application

HiveCC automatically takes updates to the processors into account and HDL code for the processors is automatically generated. Proposed design changes can almost immediately be implemented and tested. Thus, the emphasis can be laid on specifying and analyzing optimization scenarios. The actual implementation of the concepts almost comes for free.

Let’s re-visit the feedback of the scheduler as presented in Figure 2. The example application is now a 2k-FFT used in OFDM receiver for DVB-T [4]. In Figure 7, one can see the FFT kernel makes full use of the ULIW architecture of Avispa. The shading indicates the loop-nesting level. Darker blocks around program-counter positions 16 and 40 indicate 2 inner loops. The application spends 98% of its execution time on those inner loops. This shows how optimally compiled and scheduled code, running on a massively parallel ULIW processor at moderate clock speed, can have as low power as an ASIC equivalent.

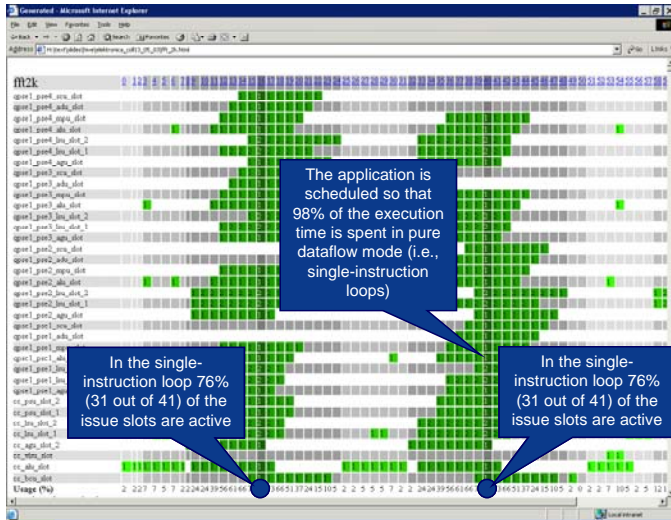


Figure 7: Avispa schedule example for a 2k complex FFT application

B/ Other examples, Multi-Core Systems

In [3], examples of Multi-Core Systems for Digital Video Reception and Video Processing have been designed and programmed, using the described software development flow.

In [4], a programmable OFDM receiver that can address most broadcast formats that are used for both digital television and digital radio in multiple geographic regions (DVB-T in Europe, ISDB-T in Japan, and DAB) is described. In this case, three main processors are developed. The first includes the Bresca stream processor for sample-by-sample processing, common to the time domain portion of each OFDM system. Another processor, the Avispa, is specialized in OFDM demodulation and frequency-domain processing. Finally, a forward error correction suite (Moustique iFEC, Avispa FEC, Moustique oFEC) handles metric processing, Viterbi decoding, Reed-Solomon decoding, and byte processing.

CONCLUSION

A methodology for programming processors for high-performance applications, embedded into SoCs, has been presented.

The combination of processor and HiveCC compiler enables reduced time-to market with gate count and power efficiency similar to hardwired ASIC circuit. Programmability reduces risks, lengthens product lifetime and allows re-use of hardware as well as software. HiveCC is a unique compiler that can fully exploit highly

parallel architectures efficiently taken into account all the resources. It schedules in seconds-minutes. It lets the user control code mapping, if needed. The integrated environment provides an easy-to-use development tool to program and simulate, as well as providing graphical feedback. Complex applications such as OFDM DVB-T receiver using 6 processors are programmed using the IDE.

For this task, the designer benefits from Silicon Hive's automated processor generation flow, which allows for a complete design iteration cycle in the order of hours.

HIVECC spatial compiler blurs the borders between compilers and hardware synthesis tools, while preserving a sequential ANSI-C input format. In this way, the compiler can ensure that the accelerator will run in flow-through mode most of the time, boosting the achievable computational efficiency (MOPS/W).

REFERENCES

[1]: *Enabling Software-Programmable Multi-Core Systems-on-Chip for Consumer Applications*, Marco Jacobs, GSPx, Santa Clara, CA-USA, Oct. 24-27, 2004

[2]: *Silicon Hive's Scalable and Modular Architecture Template for High-Performance Multi-Core Systems*, Menno Lindwer, GSPx, Santa Clara, CA-USA, Oct. 24-27, 2004

[3]: *Multi-Core Systems for Digital Video Reception and Video Processing*, Geoffrey Burns, GSPx, Santa Clara, CA-USA, Oct. 24-27, 2004

[4] *Flexible Embedded Processors for Developing Multi-Standard OFDM Broadcast Receivers*, Paul Gruijters, Klaus J. Koch, Geoffrey Burns, GSPx, Santa Clara, CA-USA, Sept. 27-30, 2004

[5] www.silicon-hive.com

[6] *Constraint Driven Operation Assignment for Retargetable VLIW compilers*, Marco Bekooij, Philips Electronics N.V., 2004, ISBN 90-74445060-8

[7] *Generating Instruction Sets and Microarchitectures from Applications*, I. Huang, A.M. Despain, ACM 0-89791-690-5, 1994

[8] <http://www.eclipse.org/>