

Silicon Hive's Scalable and Modular Architecture Template for High-Performance Multi-Core Systems

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

Within an SoC, performance can only grow through scalability, which in turn can only be achieved through short wires and low fan-in and fan-out. The Silicon Hive ULIW¹ processor template makes use of these properties. It provides multiple threads of control (cell-level multi processing), each processor cell having a multitude of issue slots, localised register files, memories, and interconnects. The compiler controls all of these resources separately, eliminating hardware overhead for instruction decoding, pipeline control, hazard detection, and bypass networks. This paper describes the template and presents examples of processors in the Silicon Hive product offering.

Introduction

Today's Systems-on-Chip (SoCs) show an increasing gap between high performance and power budget [1].

Another such conflicting set of trends that can be observed in the SoC industry is increased complexity and integration versus decreasing time-to-market. Design teams have to decide on vast ranges of integrated functionality, implement that functionality in a very short time, whilst reducing market risks.

In order to combat such a wide set of conflicting requirements, the SoC industry needs to take some radical steps. Market risks can only be reduced by replacing fixed-function ASICs with programmable devices. Power consumption can be reduced through full exploitation of parallelism and reduction of speculative operations (large caches speculate on data being re-used, deep pipelines speculate on code being sequential, branch prediction keeps pipelines filled speculating on jump statistics tables, etc.). Programmable parallelism can grow through scalability at multiple levels: multi-processing, instruction-level parallelism, and vector processing. Scalability can only be achieved through locality of

reference, short wires, and low fan-in and fan-out. Time-to-market decreases through programmability, IP-based design, and IP design at high abstraction levels. In conclusion, future SoCs will depend on a flexible and versatile market for programmable, scalable, and highly parallel IPs and IP design methodologies.

An informal study, done in August 2000 (see Figure 1), shows how increased parallelism can lead to improvements in power consumption. If Intel would have switched from singular to multiprocessors in 1985, then in 2000, power consumption of a multi-386DX/33 would have been 22 times lower than a single-processor Pentium-III/600. The 28M-transistor Pentium-III die would have fitted 102 386DX cores. Implemented in a CMOS 0.18µm process and running at 33 MHz, these cores would consume 1.1W and achieve a combined SpecInt95 score of 23. On the other hand, the Pentium-III/600 consumes 25W, in order to achieve a SpecInt95 score of 22. The same analysis was done for AMD processors, ranging from AMD K5 to K7, with comparable results. Obviously, this would only have made sense if the software development community had simultaneously switched to multi-threaded programming...

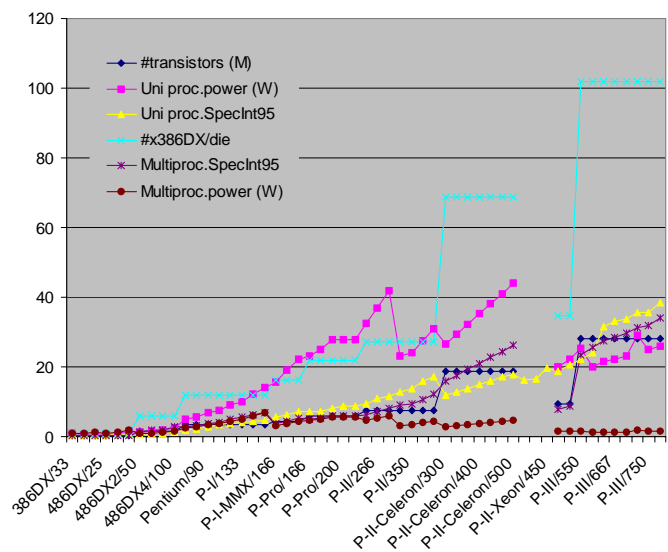


Figure 1: Multi- versus single-processor

¹ ULIW is a trademark of Philips Electronics .

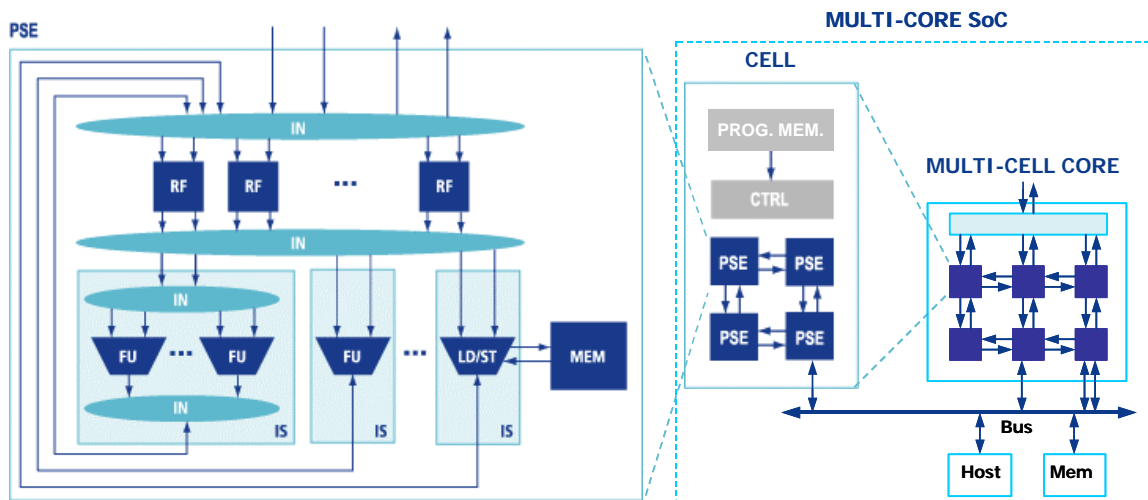


Figure 2: Hierarchy of Processor Cell Template

The Silicon Hive ULIW processor template supports all requirements to achieve exactly this goal; fast development of ASIPs, featuring scalability at all levels of parallelism, while giving the compiler full control, in order to actually benefit from available parallelism. Conversely, the template can be seen as an approach to designing programmable custom logic. Comparative analysis shows that domain specific programmable ULIW processors are comparable in size and power consumption to ASICs designed for the same set of functions. The next chapters discuss the features of the ULIW processor template: processor, processor cells, processing and storage elements (PSEs), interconnect networks, issue slots, function units, and I/O blocks. After that, as an example, a fully programmable Digital Television receiver chip, based on the ULIW template, is being discussed. The last chapter gives some conclusions.

Template Components

The ULIW template is fully hierarchical. A core consists of multiple cells, each of which having its own thread of control. Cells consist of Processing and Storage Elements (PSEs), a CoreIO module (not drawn in Figure 2), and interconnect networks. PSEs consist of register files and issue slots. Issue slots consist of interconnect networks, and function units. This hierarchy is depicted in Figure 2. The template is supported by a library of specialised PSEs (control, DSP, media, OFDM) and function units (load/store, branch, arithmetic, shift, MAC, etc.).

As an example, Figure 2 shows how single- and multi-cell cores may be incorporated in an SoC. Figure 2 also shows that cells can have multiple slave- and master interfaces connecting to SoC buses. The CoreIO template, to be discussed

below, offers ample flexibility to incorporate ULIW cores in any SoC environment.

It is worth noting that the control points of all template components (including interconnect networks) are visible to (i.e. under control of) the HiveCC² compiler.

Processor Core

A processor core consists of multiple cells. Cells can have streaming interfaces, which allow the cells to be interconnected. For scalability reasons, usually a nearest-neighbour interconnect strategy is chosen, leading to a mesh structure (comparable to that of the transputer network [2]). In order to reduce the number of external streaming interfaces, the core can be equipped with so-called stream switches, which act as run-time configurable concentrators.

For systolic array type programming, homogeneous sets of cells are usually chosen. In the current Silicon Hive product offering, Bresca² is such a core. It is meant for front-end filtering. FIR filters are typically spread out over several cells, each cell computing one or more taps. Conversely, Silicon Hive also markets single-cell template instances. Obviously, a single-cell core does not require stream switches. The Multi Standard Digital Television IP is a multi-core design, which consists of a Bresca, Avispa-CH1², and Moustique-FEC². Next to streaming interfaces, cells can have both master and slave bus interfaces for several different IP interconnect standards.

² HiveCC, Bresca, Avispa, and Moustique are trademarks of Philips Electronics.

Processor Cell

Each processor cell has its own thread of control. The control PSE (indicated as 'CTRL' in Figure 2) is standardised and has two functions: stepping through program code, and guaranteeing ANSI C compliance. The control function of the control PSE is much less complex than in traditional RISC or VLIW processors. Because all datapath features are visible to and controlled by the compiler, there is no need for hazard detection, bypass networks, branch prediction, or pipeline control.

Next to the control PSE, a cell normally contains a set of DSP PSEs. For simplicity's sake Figure 2 shows PSEs as having nearest-neighbour interconnect. In reality, this is an interconnect network, as described below.

Processing and Storage Element (PSE)

A PSE comprises multiple issue slots (ISs). In order to achieve the objectives of having short wires, low fan-in and low fan-out, the input data for the ISs is distributed over multiple register files (RFs). Interconnect networks (INs) route the data from the outputs of ISs into the RFs and from the RFs into the inputs of ISs. The designs of the INs are such that the RFs are connected to FUs on a need-to-have basis. This reduces the input/output ports of the RFs to a minimum.

Interconnect Network

As discussed above, in order to minimise wirelength and RF ports, the INs are specifically set up to obtain a sparsely interconnected design. There are two strategies to arrive at a meaningful IN configuration: design space exploration and stripping tools. Design space exploration usually starts from a minimum configuration and builds additional interconnect, as required. When using the stripping tool, the starting point is a more or less fully connected core. The stripping tool will analyse the software schedules for the applications and strip the core from all interconnect that is not being used. Obviously, the first strategy does not guarantee a minimal result. The latter strategy may preclude future applications.

Issue Slots and Function Units

Issue slots basically consist of multiple function units (FU). Issue slots are also logical elements, in the sense that the program word contains exactly one operation per IS. This means that in every subsequent cycle one operation per issue slot can be fired. Maximum parallelism of a cell depends on the number of issue slots.

The INs within ISs are often more or less fully connected, as they are either present for distribution or concentration.

FUs determine the semantics of the core. FUs generally do not contain state. However, this is not a template requirement. FUs can have multiple cycles latency. FUs generally can perform several operations (a load/store unit can load and store values of different sizes, either with or without auto-increment, etc.). With every FU instantiation, the core designer determines which of the FU-supported operations are actually implemented. This saves valuable program memory bits, for example, when instantiating an ALU which only needs to do additions.

Load/store units (LSU) are special kinds of FUs. Besides being connected to RFs within the core, they are also connected to CoreIO. As far as an LSU is concerned, it does not make a difference whether CoreIO connects it to a streaming interface, a bus interface, or to a local memory.

FUs have parameterised latency. If a certain FU appears to be a bottleneck, its latency can be increased. The compiler will take this into account when scheduling the availability of the FU's results.

The CoreIO Input/Output Concept

Every cell has a CoreIO component. It connects the cell's LSUs to a configurable number of so-called CoreIO devices, being: FIFOs, external (bus) interfaces, and local memories. Caches may be inserted between any LSUs and the devices to which it connects. The library contains a cache module, which has parameters, such as number of cache lines and associativity. However, within the Silicon Hive cores, caches are used very sparingly. The reason is that caches reduce predictability of

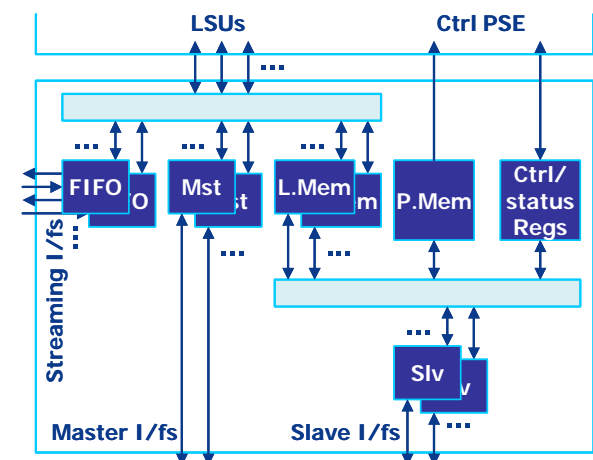


Figure 3: CoreIO Template

software schedules. Silicon Hive used caches on those infrequent occasions where pseudo multi-port memories were explicitly required and were, in terms of timing, accounted for in the code schedules.

Local memories have parameters such as width and depth.

CoreIO contains an IN for interconnect between LSUs and external master interfaces/memories and an IN for interconnect between external slave interfaces and memories. CoreIO also takes care of arbitration between accesses from the cell itself and external accesses.

CoreIO offers a high level of flexibility, needed to easily integrate ULIW cores in SoCs. CoreIO is completely scalable in terms of the number of external bus interfaces (both master and slave), and streaming interfaces. The streaming interfaces can be parameterised in terms of width and FIFO depth. CoreIO's external interfaces have parameters such as protocol (proprietary CoreIO protocol, AHB, etc.) and width.

The second loop iterates over such architecture issues, such as function unit latency, in order to balance cycle budgets within the design and remove timing bottlenecks. The processor generator takes only seconds to generate a new instance of the HDL code. Subsequently, standard EDA tools are used to derive metrics, such as area, clock speed, power consumption. Iterations times in this loop depend mainly on the available EDA tools. They are generally in the order of hours to days.

Example: Digital Television Prototype Receiver Chip for Mobile Appliances

This project started in February 2004. The chip taped out to a 0.13µm processing facility in September 2004 [8]. On December 2004, first-time-right working silicon was announced. Wireless DVB-T (Digital Video Broadcast – Terrestrial) reception was demonstrated during the GSPx TV On Mobile event of May 2005 [3]. In the mean time, DVB-H (DVB for handheld), DAB (Digital Audio Broadcast) and ISDB-T (Integrated Services Digital Broadcasting – Terrestrial) receivers have been programmed on this chip.

This is a prime example of an instance of the ULIW template. As part of the ULIW instance, the chip contains a Silicon Hive Bresca core, an Avispa-OFDM core ([4], [5]), and a Moustique-FEC core. As mentioned above, Bresca is a collection of cells with 24 separate threads of control. In total, the ULIW instance runs 29 separate threads of control. The instruction-level parallelism ranges from 9 to 42 issue slots per thread, totalling 353 issue slots for the whole ULIW instance. In order to save power, the different cores can be set to run at different clock speeds. When all cores run at their nominal clock speeds, the chip has a maximum throughput of about 43 GOPS.

When receiving DVB-T, the sustained throughput is about 25 GOPS. The embedded ARM processor is only used for configuration of the system. This can also be done by an external controller. Combined with a reduction of compute headroom and a transition to 90nm technology, a product version of this chip consumes less than 300mW.

During steady-state operation, the ARM is not active at all. This shows that the ULIW template can deal with most steady-state control tasks.

Other examples of ULIW-based cores that may be co-operating in a multi-processor design are Moustique-IC1, which can handle image sensor

Design Methodology

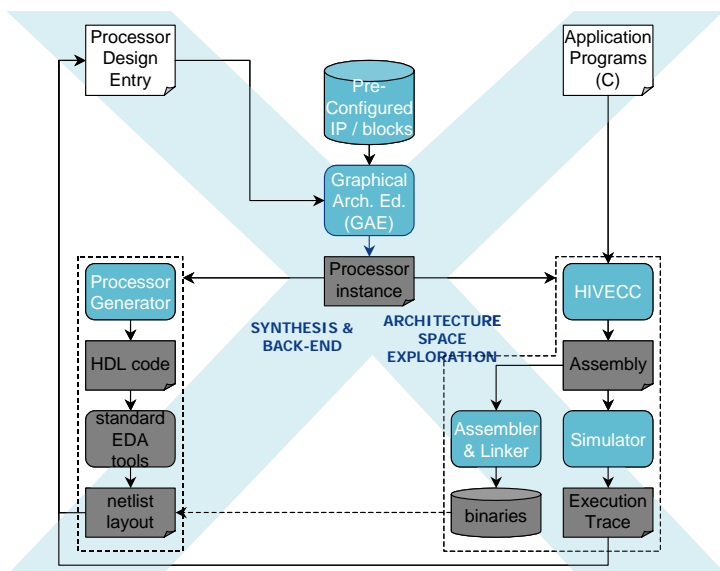


Figure 4 X-chart Design Methodology

The design methodology for generating ULIW cores consists of two main iterative loops. The initial loop iterates over the main features of the processor instance (processor design entry in Figure 3), compiling the application programs (HiveCC), and simulating them, until the execution trace indicates that schedule within a pre-set cycle budget. Each iteration in this loop typically takes in the order of one hour.

capture and encoding, and Avispa-IM1 [6] (or Avispa-IM2 [7]) for image enhancement.

Conclusions

The example illustrates almost all features of the template. It substantiates the claim that the template can be used to address highly complex programmable designs, while reducing time-to-market and market risks.

As far as we know, the discussed example illustrates the first working silicon implementation of a fully programmable digital television receiver. At the same time, it is one of the most power efficient DVB-T receivers. This ASIC-level power efficiency is a result of the template actually resulting in an ASIC-like design, with a comparable level of parallelism and locality of reference. The core designer determines which compute and memory facilities are being laid out. The HiveCC compiler schedules operations, in order to optimally utilise these facilities.

The example shows that the ULIW template achieves its goal of reducing time-to-market, because comparable IC development projects require at least two years and a silicon re-spin, as opposed to 9 months until tape-out for this project (and no re-spin).

References

[1] John L. Hennessy, *Directions and challenges in microprocessor architecture*, Holst Memorial Lecture 2002, <http://www.holstmemorial.nl/hennessy.ppt>

[2] <http://en.wikipedia.org/wiki/Transputer>

[3] *Silicon Hive Demonstrates World's First Fully Programmable Digital TV Demodulator IP Core*, <http://www.siliconhive.com/t.php?assetname=text&id=79>

[4] Tom R. Halfhill, *Silicon Hive Breaks Out; Philips Startup Unveils Configurable Parallel-processing Architecture*, http://www.siliconhive.com/uploads/m48_siliconhive_rprnt.pdf, January 2003.

[5] Jim Turley; *Avispa+ Buzzes With Innovation; High-End Core Combines Decades of Competing Architectural Ideas*, http://www.siliconhive.com/uploads/siliconhive_bestipcore.pdf, February 2004

[6] Tom R. Halfhill, *Busy Bees At Silicon Hive*;

New Processor Cores Target Pixel Processing and Communications, http://www.siliconhive.com/uploads/mpr_spring2005_avispa.pdf, June 2005

[7] <http://www.siliconhive.com>

[8] Geoff Burns, Marco Jacobs, Menno Lindwer, Bertrand Vandewiele, *Multi-Core Systems for Digital Video Reception and Video Processing*, GSPx 2005, October 2005