Abstract— This work presents a SystemC-based design of custom SIMD instructions for accelerating media and telecom codes on a next-generation configurable, extensible processor. The SS_SPARC processing platform, incorporates a generic vector unit which can be extended with pipelined, SIMD computation units (datapaths) designed either with established (RTL-based) or in this case, hybrid (SystemC-RTL) methodologies. This work elaborates on a custom methodology for automatically encapsulating the data-parallel sections of the MPEG-4 XviD the G723.1 and G729A reference codes into a SystemC wrapper which is subsequently synthesized to RTL with a commercial SystemC-synthesis tool. The resulting RTL is then attached to the exposed vector unit of the SS_SPARC engine. We present results from a standard-cell RTL synthesis campaign and the VLSI implementation of a high-end (8-contexts, 256 bit) and a low-end (2-context, 128 bit) configuration of the vector engine for the workloads of interest.

I. INTRODUCTION

Parallelism is the fundamental performance leverage for the real-time execution of video coding workloads on SoC-based, battery-powered consumer products. New approaches based on ultra-long instruction word (ULIW) architectures show significant potential for the acceleration of telecom and media kernels [1]. More established configurable, extensible CPU vendors [2], [3] have been actively improving their microarchitectures for a number of years and there are now many examples of successful consumer products utilizing these CPUs. In addition, traditional 32-bit RISC processor vendors have acquired data-parallel technology which they make available to their licensees [4]. A common characteristic in all these successful architectures is the single CPU approach, with either automated or manual custom instruction set extensions for the exploitation of the inherent data-level parallelism (DLP) of consumer workloads. However, none of these major vendors has conclusively addressed a fundamental and substantial alternative form of parallelism, namely thread level parallelism (TLP).

This work advances the knowledge in this area in a number of ways. It proposes a unique CPU system architecture, incorporating mechanisms to exploit the predominant forms of parallelism in the consumer and telecom workloads domain. The CPU architecture and microarchitecture are parametric and the processing platform is capable of exploiting instruction level parallelism (ILP, per processor) to a moderate extent, as well as substantial amounts of both DLP and TLP, the former through a high-performance vector engine and the later via user-selectable simultaneous multithreading (SMT) and chip-multiprocessing (CMP). A further novel aspect of this work is the fusing of electronic system-level (ESL) design languages such as SystemC with the established RTL-based methodologies to automate the process of custom vector instruction design. This is achieved by a custom tool-flow which allows the user to define C-level vector macros for at-speed development of the target application on an X86 platform while automatically turning these vector instructions into combinatorial or pipelined SIMD datapaths which are then attached to the exposed vector unit of our SS_SPARC processor.

II. BACKGROUND

In our previous work [5], we quantified extensively the DLP for a number of video coding standards, including the open-source, integer-based MPEG-4 (XviD) implementation [6]. We take this work further by fully threading the encoder and presenting the combined DLP and TLP performance improvements obtained from a parallel-RAM (PRAM) model. From the same PRAM model we also compute the theoretical performance benefits of the optimized (vectorized) implementations of the ITU G723.1 [7] and G729.A [8] speech coders. Such results are essential as they clearly show the scalability of both algorithms along the axes of parallelism (data-level and thread-level).

There is a small number of articles available in the literature regarding the use of SystemC for the design of custom SIMD instructions. Azevedo et al. [9] developed ArchC, a SystemC-based language for describing CPU architectures, including a number of commercial devices incorporating SIMD extensions. However, the work focused
primarily at functional simulation level and no route to a silicon implementation was provided. Wieferink et al. [10] described a methodology for CPU design and optimization as part of an architecture utilizing SystemC transaction-level models. Using this technique, the SystemC models can be successively refined to meet the performance requirements of the workload. Oussorov et al. [11] have integrated SIMD extensions written in SystemC through a co-processor interface to a CPU simulator. The extensions required for a given application are developed through a process of gradual refinement using system-level simulation. Our approach takes this later route and, in addition to easily interfacing to our PRAM and performance modeling simulation environments, provides a clear route to a silicon implementation of the SIMD extensions for the workloads of interest.

III. THE SS_SPARC ASIC PROCESSING PLATFORM

This section discusses a new ASIC processor known as the SS_SPARC processing platform. The platform is uniquely architected to exploit all forms of parallelism namely ILP, DLP and TLP. This is accomplished via a three pronged approach, at the kernel, processor and vector unit levels.

![SS_SPARC Kernel Diagram](image1)

Fig. 1 depicts a high-level view of a generic SS_SPARC processing kernel which consists of a configurable number of superscalar, SMT processor cores, themselves highly parameterized (discussed later), a configurable number of loosely-coupled, streaming coprocessors (accelerators), a switch matrix and a multi-banked, level-2 data cache which communicates with the remainder of the SoC via the (embedded) DRAM interface. The CPUs follow a shared-memory programming model and full cache-coherency is supported across the internal, level 1 data caches of the processors and the banked level 2 data cache via a simple, 3-state (MEI) coherence protocol. The switch matrix is segmented into separate coherent and non-coherent ‘channels’, assigned to the processors and to the streaming accelerators respectively. At this level, the platform can exploit TLP via distinct software threads running in different CPUs. The core processor is a highly-parameterized, five-wide, in-order issue, out-of-order commit (with in-order exception resolution), multi-threaded (SMT), Sparc V8-compliant microarchitecture.

![SS_SPARC CPU Detailed Diagram](image2)

As shown in Fig. 2, the CPU is segmented into four major sections: the processor instruction front-end (IFE), the core datapaths for scalar operations (SCORE), the configurable, extensible vector unit (VCORE) and the high bandwidth load/store unit (LSU). VCORE is responsible for the extensive DSP capabilities of the CPU and is the primary mechanism provided for exploiting DLP. It consists of a single-issue, configurable vector datapath, supported by an architected vector register file. VCORE is uniquely designed to connect to arbitrary, ‘plug-in’ datapaths at RTL level, via exposing a highly consistent and simplified interface to the external system designer, as depicted in Fig. 3. This microarchitectural block constitutes the hardware target of our tool flow which parses the vectorized application code and builds the functionality of the plug-in SIMD datapath automatically.

![Vector Pipeline Interfaces](image3)

---

1 The latter can be removed and replaced by a controller for a typical on-chip bus standard such as AMBA

2 Although essentially single issue, the datapath will also allow for a vector result to return from the LSU and commit to the vector register file in parallel to a standard vector op

3 The architected vector register file is not windowed; it is an extension to the Sparc architecture.
Figure 4. G723.1 and G729A relative performance improvement over VLMAX for all ITU test vectors, for the Codebook search

The G723.1 codebook search shows performance saturation at a vector length of 8 (8x16-bit) while the G729.A shows relative performance improvement for up to a VLMAX of 16 (16x16-bit). Fig. 4. The XViD open-source MPEG-4 video coder implementation shows good DLP-scalability up to a vector length of 16 (16x8-bit); at the same time, the PRAM shows excellent scalability with regard to the number of CPU contexts (SMT/CMP) for each vector length and for both quality settings (Fig. 5). The results suggest that good performance compromise points are single context/128 bit for G723.1 and G729.A and dual-context 128-bit for MPEG-4.

V. ESL FLOW

The starting point is the target application C-reference implementation. This is systematically vectorized and parallelized (threaded) and during this stage, its theoretical performance is evaluated on a proprietary exclusive-read, exclusive-write (EREW) PRAM model with full vector state and instruction set extensibility support. This stage yields very important statistics as it determines the vector instruction set extensions required to exploit most of the DLP in the application and profiles these extensions over an ideal, parametric multiprocessor. The outputs of the stage are C-level macros that precisely define the vector instruction set extensions, the extra state required and the optimal number of processors (processor contexts) for efficient parallel execution. The custom flow parses these C-macros and creates a SystemC module that instantiates these SIMD instructions. At the same time, test vectors produced during running the vectorized algorithm are also applied to the SystemC model to ensure that this 'packing' of the SIMD ISA has not affected the functionality of the individual instructions. Next, a user-specified number of pipeline registers are added and taps to the bypass infrastructure are wired. The SystemC datapath is then synthesized to RTL VHDL using a commercial SystemC synthesizer. This is followed by a further validation campaign which applies the very same test vectors (applied to the SystemC model) to the RTL model to ensure that the SystemCRTL transformation was successful. Finally, the resulting RTL datapath is instantiated in the exposed vector unit of the SS_SPARC processor and further decoding logic is added to the core processor to enable these extensions. The overall flow is demonstrated in Fig. 6.

VI. VLSI RESULTS

We implemented the VCORE for 1, 2, 4 and 8 contexts, targeted at 166, 200, and 333 MHz, for 16 and 32-byte wide SIMD configurations, for both the MPEG-4 (XviD) code and the G723.1 and G729.A codes. Figs. 8 and 9 depict the area (μm²), and power (mW) usages of these VCORE instances. Results were collected post-synthesis (pre-route) over a period of three weeks, using a fully-scripted validation/synthesis campaign. Fig. 7 depicts the VLSI
layout of two configurations a high performance 1P8M 0.13 μm silicon process from UMC;

![Figure 6. Toolflow](image)

Figure 6. Toolflow

![Figure 7. 2-context, 128-bit ITU accelerator (a) and 8 context, 256-bit video accelerator VCORE VLSI layout](image)

Figure 7. 2-context, 128-bit ITU accelerator (a) and 8 context, 256-bit video accelerator VCORE VLSI layout

![Figure 8. VCORE (ITU accelerator) area, pre-route](image)

Figure 8. VCORE (ITU accelerator) area, pre-route,

![Figure 9. VCORE (ITU accelerator) power results (pre-route)](image)

Figure 9. VCORE (ITU accelerator) power results (pre-route)

**REFERENCES**


**VII. CONCLUSIONS**

This paper has discussed a novel methodology for the design of custom vector instructions extensions using a proprietary tool flow. The target is a high performance, configurable, extensible, ASIC processing platform which allows the use of arbitrary, vector datapath ‘plug-ins’. Vectorized reference code is passed through a proprietary tool to generate synthesizable SystemC which is then synthesized to RTL with a commercial-quality SystemC compiler. Finally, the resulting implementation is attached to the exposed vector unit of the core processor of the SS_SPARC ASIC platform.