# Designing tightly-coupled extension units for the STxP70 processor

Yves Janin, Valérie Bertin, Hervé Chauvet, Thomas Deruyter, Christophe Eichwald, Olivier-André Giraud,
Vincent Lorquet and Thomas Thery
STMicroelectronics, Grenoble, France
Email: firstname.lastname@st.com

***Abstract*—Designed by STMicroelectronics for the embedded market, the STxP70 processor is a small but extensible processor: designers have the possibility to define tightly-coupled extensions that can be reused in different designs. We explain why this modularity has a strong impact on the toolchain, detail the hardware/software flows and give results for two extensions.**

## I. Introduction

The STxP70 processor belongs to this class of application specific instruction set processors (*ASIPs*) that can be seen as a trade-off between general purpose processors and hardware accelerators. From the former ones, they inherit their programmability whereas from the latter they benefit from the sheer power of hardware acceleration. As such, these processors are already a real challenge for software code generation and simulation tools. It is however often taken for granted that each new configuration of the *ASIP* processor is independent from any other one, and that it is therefore enough to customize and rebuild each tool.

This traditional viewpoint is challenged by the STxP70 architecture. Indeed, the core configuration can be extended with up to eight tightly-coupled extensions, the complexity of which may range from a simple arithmetic operator to a complex unit with fully dedicated register files. The key point is that these extensions may be reused and shared between several designs, which leads to obvious productivity gains.

For extension reuse to become a reality in an industrial environment, important adaptations were required on software tools. In this paper we give an overview of the STxP70 architecture in Section II, detail the hardware/software flows in Section III and present performance results for two extensions in Section IV.

## II. An overview of the STxP70 architecture

The STxP70v4 processor implements a 32-bit RISC load store architecture, and its variable length instruction set (16, 32 or 48-bit) is fully predicated. The processor is highly configurable: hardware loops, hardware contexts, program and data caches, tightly coupled data memory, branch history buffer and integer multiplier are optional. The minimal configuration only implements 16 32-bit general purpose registers (GPR) and a fully interlocked single-issue pipeline. More powerful versions of the core support a dual-issue pipeline and 32 GPRs.

978-3-9815370-0-0/DATE13/©2013 EDAA

The most salient feature of the STxP70 comes from its extension model. Extension units communicate with the core through a well-defined hardware interface. Their instructions may read up to two GPRs and write at most one. They access the same memory space as the core and may have their own (multi-level) register files. Extensions can be single-issue or dual-issue, and in the latter configuration the two pipelines may share resources. The range of instructions that can be defined within an extension is large enough to address floating point computations, DSP oriented algorithms, or SIMD integer operators. The STxP70 has been integrated in several products and is also the processing element of Platform 2012 [1], a many-core computing accelerator for embedded systems-on-chips.

## III. Tools for an extensible and configurable processor

### A. Extension development flows and reconfiguration toolkit

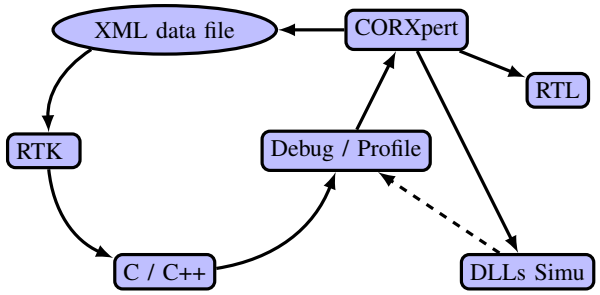


Fig. 1. Schematic Design Flow of an Extension

The definition and implementation of an extension is based on the semi-automated design flow described in Fig. 1. The CORXpert tool designed by Synopsys [2] and specifically tailored for the STxP70 architecture (following STMicroelectronics specifications) is used by designers to capture the definition of a new extension, its resources and instruction set. From a first definition, CORXpert automates the generation of an RTL implementation, and provides ways to gradually refine the structure of the extension pipeline. CORXpert also delivers two shared libraries that can be dynamically connected to the STxP70v4 core instruction accurate and cycle accurate simulators.

TABLE I
VECL AND MP1X EXTENSIONS

| Extension | Software applications | Architecture | Technology Max frequency | Gate count | Number of instructions |
|---|---|---|---|---|---|
| VecL Video processing | Vectorial video applications | Dual-issue / load-store 32 256-bit registers | CMOS 65nm LP 430 MHz | 514 Kgates | 67 |
| MP1x Media processing unit | Fract. & saturating arith. support for 64-bit int. arith. | Dual-issue / load-store 16 64-bit registers | CMOS 65nm LP 400 MHz | 67 Kgates | 40 |

The reconfiguration toolkit (RTK) designed by STMicroelectronics automates the generation of the software development tools specific to an extension: C/C++ compiler, assembler, linker, debugger, profiler and binary utilities. More precisely, the RTK takes as input the XML extension database maintained and used by CORXpert, and produces a set of dynamic libraries allowing to reconfigure each of these tools. As this generation process is fully automated, it is then possible to rapidly check whether an extension definition meets its performance, power and area constraints.

### B. Extension programming model and code generation tools

The compiler *stxp70cc* is a critical component in the software evaluation loop. *stxp70cc* automatically maps extension instructions, provided that mapping information is available in the compiler extension library produced by the RTK. Complex instructions, that typically cannot be described by composition of basic C operators, are handled by intrinsics and require modifications in application source codes. In any case, the compiler performs register allocation[1] and instruction scheduling.

*stxp70cc* is based on the Open64 [3] framework and offers a wide range of optimizations including high level loop analysis and transformations, interprocedural analysis and global scalar optimizations. The compiler backend was tailored to support predication, hardware loops and other idiosyncrasies of the processor. Many internal data structures were adapted to the dynamic reconfiguration process, and the low level target information library (*targinfo*) that is part of the backend was enriched to become the single intermediate representation used by all STxP70 *GNU* binary utilities [4] (assembler, linker . . .).

### C. Software packages for extensions

There are a lot of cases where rapidly and easily updating tools on user's site is much needed: a new extension or a new version of an extension has been developed, a specific STxP70 configuration requires two extensions . . . Software extension packages simply wrap-up RTK dynamic libraries and checksum information into a single component that can be easily deployed by software developers in their toolsets.

## IV. RESULTS

Table I presents results for two extensions, VecL and MP1x. VecL is a vectorial video processing extension with a 256-bit register file. MP1x is a DSP-oriented extension that supports fractional and saturating arithmetic and provide support for 64-bit integer arithmetic. Both are fully synthesizable, integrate a fairly large number of instructions[2], and target a complete application domain.

TABLE II
MP1X SPEED-UPS - *stxp70cc* OPTIMIZATION LEVEL -O3

| Benchmarks | FFT | convolution | MP3 |
|---|---|---|---|
| Configuration *2)* | $\times 4.6$ | $\times 1.0$ | $\times 9.3$ |
| Configuration *3)* | – | $\times 3.7$ | $\times 11.6$ |

For VecL, speed-ups up to $\times 70$ were measured on video applications after rewriting code with intrinsics. For MP1x, Table II gives speed-ups for three typical benchmarks, an integer FFT, a convolution filter and a MP3 decoder. Three configurations were tested: *1)* out-of-the-box (OOB) code and STxP70v4 dual-issue core, *2)* OOB code and STxP70v4 dual-issue core + MP1x, *3)* code modified with MP1x intrinsics and STxP70v4 dual-issue core + MP1x. All ratios were calculated with respect to configuration *1)*.

## V. CONCLUSION

The STxP70 toolset offers a high level of integration and a very efficient software evaluation loop. It puts a strong focus on extension reusability at an industrial level, and was successfully used to design numerous extensions covering a wide range of applications (audio, video, DSP processing. . . ). Compared to other fully automated approaches, the STxP70 toolset requires human expertise in the design of an extension. To the best of our knowledge, many instructions defined in STxP70 extensions still exceed the possibilities of state of art algorithms for instruction set customization and compiler instruction selection [5]. We believe though that this kind of approach paves the way for further improvements in the STxP70 tools.

## REFERENCES


[1] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *DATE'12*, pp. 983–987.
[2] *Synopsys*. [Online]. Available: http://www.synopsys.com
[3] *Open64*. [Online]. Available: http://www.open64.net
[4] *GNU Binutils*. [Online]. Available: http://www.gnu.org/software/binutils
[5] C. Galluzzi and K. Bertels, "The instruction-set extension problem: A survey," *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 209 – 220, 2008.


[1]Technically, a set of *predefined* instructions available for every extension register file guarantees that spill/reload sequences can always be generated, and therefore that register allocation is always possible.

[2]The number of instructions reported in Table I corresponds to the number of instructions before sub-operator expansion. For instance a single comparison sub-operator expands into ten sub-instructions if one takes into account signed/unsigned comparisons for $(=, \neq, <, \leq, >, \geq)$.