General-purpose and digital signal processors using fixed instruction sets are failing to meet compute requirements of today’s complex embedded systems. Introducing hardware acceleration provides the necessary performance increase, but at the cost of increasing complexity and extending the overall design cycle.

Complex compute-intense functions can be converted into a single instruction through software-configurable processors, which have resources that can be extended easily under the control of the software programmer through multiple configurations. This article will describe the architecture of a software-configurable processor and the mechanisms for loading individual configuration contexts.

Software-configurable processors
Software-configurable processors integrate programmable logic as part of the processors execution pipeline, which is accessed in the same way as any functionality in the processor: via software instructions. Complex algorithms can be reduced to a handful of optimized custom instructions that each represent hundreds of lines of C code executed in a highly parallelized pipeline. Such calculations execute in tens of cycles, down from hundreds or thousands of cycles, increasing the computational capacity of the processor for this specific application. Unique software instructions loaded into the programmable logic effectively configure the processor to the software application. As the developer writes new software, custom software instructions can be created to greatly reduce the number of processor cycles required. As the software developer continues to improve the algorithm(s), the processor continues to be configured (customized) to meet the new requirements.

The S5 software-configurable processor architecture from Stretch (Figure 1) simplifies the configuration of programmable logic through a tightly integrated software development environment. The software-configurable processor architecture is the first off-the-shelf implementation of the Xtensa Instruction Set Architecture (ISA), a configurable and extensible processor developed by Tensilica. The Xtensa ISA provides the base mechanisms for supporting custom instructions implemented in hardware as part of the execution pipeline. Stretch extended the flexibility of the Xtensa ISA to a higher dynamic with the introduction of an Instruction Set Extension Fabric (ISEF) and a 128-bit Wide Register (WR) file. The ISEF provides the programmable logic resources capable of holding multiple custom instructions, and is run-time configurable and reloadable. The WR serves as an efficient data-passing unit between the Xtensa ISA, the ISEFs, and memory.

Developing applications for software-configurable architectures follows the same process as the traditional software development cycle. An integrated development environment manages the project and acts as a front end to the development tool chain, including compiler, debugger, and profiler. When it comes time
to improve application performance, however, rather than hand-coding assembly language or, for FPGA coprocessor architectures, passing the software algorithm to a second hardware development team to implement the function in hardware, developers instead identify “hot spots” within the program. This enables the compiler to accelerate algorithmic code by creating an extension instruction. Without requiring additional programming from the developer, the compiler creates an optimized configuration to be implemented in a programmable fabric and schedules the instruction as it would any other instruction. Developers can then profile the performance of the extension instruction. If required, the function can be characterized to process multiple data words in parallel.

Accessing this configurable functionality as software instructions has a tremendous impact on the way system developers approach embedded design. Even though the custom instructions are implemented in hardware, that is, programmable logic, developers create and use them in an entirely software context. This keeps the design of the system in a single development environment, which is both conventional and familiar to software developers. If a change in application code results in a change in the custom instructions, the compiler handles the details, rather than requiring two teams to rearchitect their hand-optimized logic based on the new partitioning.

Since both hardware and software functionality is captured in the same application code, the compiler plays an essential role in the abstraction of custom instructions and their partitioning into hardware and software resources. The Stretch C compiler manages the abstraction of custom instructions in application code, optimizes allocation of ISEF configurable resources, and tracks dependencies across the ISEF and the Xtensa processor. By leveraging key aspects, such as operator fusion and vectorization, the software developer can realize 10x-100x software performance through hardware acceleration directly from C/C++ software.

**Operator fusion**

Operator fusion combines multiple computation operations into a single instruction. As a result, an entire function is encapsulated as a single extension instruction. This operation in effect transforms a generic instruction set architecture into a highly specialized set of operations specific to the application. In the software-configurable processor, the resources of the ISEF limit the number of operations. The ISEF has proven to be large enough to hold multiple instructions in a single configuration. Since multiple instructions are often the desired goal for optimal software performance, operator fusion should be designed from the ground up with vectorization in mind.

**Vectorization**

Vectorization is a traditional stage of hardware acceleration. The ability to process multiple words of data with a single instruction (Single Instruction Multiple Data or SIMD architecture) is critical for improving performance without having to clock processors at higher frequencies, ultimately leading to greater manufacturing cost and increased power consumption.

One capability important to overall performance efficiency is the ability to work with different data sizes and data formats. Limited choice of data width and format constrains the use of fixed instructions. For example, depending upon the application and task at hand, the ideal data size may not be byte (8 bits), half-word (16 bits), or word (32 bits) oriented. The ideal size may be better specified in bits. The advantage of a software-configurable architecture is that data size and format can be determined on an instruction-by-instruction basis. It is unnecessary to convolute data to fit the size of the instruction; extension instructions can be designed specifically to match data to reduce parsing overhead, maximize resource usage, and achieve maximum performance. In doing so, multiple data objects can be passed to the extension instruction through one or more of the 128-bit WRs. Through
simple extraction and concatenation operators, the Stretch C compiler arranges the data easily for the compute operations.

**ISEF configuration**

In some applications, a single ISEF configuration satisfies the compute requirements. However, as the compute requirements increase, so does the need for more specialized instructions. At some point the capacity of the ISEF will be exceeded and the instructions will need to be organized into different configurations. Various factors should be considered in supporting multiple ISEF configurations, including the number of extension instructions held in a single ISEF; mechanisms for loading different instructions, and how long it takes to load.

The configuration time for the S5 architecture is approximately 100 µsec (32 K processor cycles). Considering this, the decision to use multiple configurations requires that at least this many cycles will be saved. Using the software profiling tools provided in the Stretch Integrated Development Environment (IDE) determines the number of cycles consumed by the processor. These tools report the number of times the function is called and the number of cycles consumed. Once the functions are identified, the organizing into extension instructions configuration requires understanding the algorithm and the resource usage inside the ISEF.

The ISEF unit in the S5 architecture can implement 16 extension instructions in a single configuration. To the first order, ISEF resource utilization can be estimated by looking at the number of multiplications and additions inside the function. Figure 2 shows the resource utilization by operation. The S5 ISEF contains 8 K Multiplier Units (MU) and 4 K Arithmetic Units (AU), and 4 K state registers supported by the necessary logic and routing resources. When multiplying A x B, the number of MUs required is the product of the number of bits for A and B, for example, a 16 x 16 multiplier requires 256 MUs. For addition and subtraction operations, the number of AUs consumed is the same as the size of the larger datum, for example, a 16-bit + 32-bit value requires 32 AUs. During development, resource utilization and the cycle counts are reported by the development tools.

The S5 architecture provides two independent ISEF blocks. For applications with one or two configurations, both configurations will be loaded during initialization and instructions from either ISEF can be used. For applications with two or more configurations, the BIOS will automatically manage the loading of instructions. When a user-defined instruction is issued, the S5 hardware checks to make sure the corresponding configuration is loaded into one of the two ISEFs. If the required configuration is not present in either ISEF, it is automatically loaded prior to the execution of the user-defined instruction.

The automatic system uses the simple Least Recently Used heuristic to determine which configuration to swap out in order to load a new configuration. In most applications, the developer overtiming control and loading the desired configuration in anticipation of its usage minimizes this delay. The S5 architecture supports the background loading through the use of a BIOS call. The BIOS call, sx_isef_load_by_name_async(), loads the requested configuration to the specified ISEF using a dedicated DMA engine. This function is nonblocking, that is, sets up the DMA and returns, and can be called again to configure the other ISEF.

While loading one of the ISEFs, the instructions in the other ISEF can continue to be used as well as the Xtensa’s processor pipeline. In effect, with proper scheduling the processor will continue to operate without stalling or looping to check for status and availability.

Another key benefit of this development flow is that it keeps design in a single development environment that is well established and familiar to software developers. FPGA-based architectures require the use of the second development team, and any repartitioning of application code requires a rearchitecting of hand-optimized logic to match the new partitioning. With a software-configurable processor, the compiler is responsible for rearchitecting. This means that even though extension instructions are implemented in programmable hardware, developers design, create, and use them entirely in a software context.

Together, all of these factors have a tremendous impact on the way developers approach application design. A single development team can create an application, and enabling concurrent software and hardware development can significantly reduce development time without time-consuming hand optimization.

**Figure 2**

<table>
<thead>
<tr>
<th>C Operators</th>
<th>AU</th>
<th>MU</th>
</tr>
</thead>
<tbody>
<tr>
<td>A*B</td>
<td>0</td>
<td>(</td>
</tr>
<tr>
<td>A(+,-)B</td>
<td>(\max(</td>
<td>A</td>
</tr>
<tr>
<td>A(&lt;,&gt;)B</td>
<td>0</td>
<td>(</td>
</tr>
<tr>
<td>A(&amp;,</td>
<td>)</td>
<td>B</td>
</tr>
</tbody>
</table>

**Conclusion**

Through the use of software-configurable processors, developers can implement hardware acceleration for compute-intensive algorithms by means of extension instructions coded in C/C++. Extension instructions provide the performance of hardware implementations with the flexibility of software design. Specialized computations on specialized application data sizes and formats increase flexibility and optimize the use of computational resources. By describing software and hardware functionality using a single programming language and development tool chain, a development team can design hardware and software concurrently, significantly reducing time-to-market.

The flexibility of dynamically loading multiple instruction configurations into software-configurable processors also enables developers to extend the number of custom instructions available for increasing the compute performance of the processor based on the application.

Joe Hanson is the director of business development for Stretch. Previously, Joe spent eight years with Altera, serving as director of marketing for system level tools and director of marketing and applications for the Excalibur business unit, Altera’s embedded processor solutions group. He has BS degrees in Electrical Engineering and Biological Sciences and holds three patent awards.

To learn more, contact Joe at:

---

Joe Hanson is the director of business development for Stretch. Previously, Joe spent eight years with Altera, serving as director of marketing for system level tools and director of marketing and applications for the Excalibur business unit, Altera’s embedded processor solutions group. He has BS degrees in Electrical Engineering and Biological Sciences and holds three patent awards.

To learn more, contact Joe at:

---

Joe Hanson is the director of business development for Stretch. Previously, Joe spent eight years with Altera, serving as director of marketing for system level tools and director of marketing and applications for the Excalibur business unit, Altera’s embedded processor solutions group. He has BS degrees in Electrical Engineering and Biological Sciences and holds three patent awards.

To learn more, contact Joe at:

---

Joe Hanson is the director of business development for Stretch. Previously, Joe spent eight years with Altera, serving as director of marketing for system level tools and director of marketing and applications for the Excalibur business unit, Altera’s embedded processor solutions group. He has BS degrees in Electrical Engineering and Biological Sciences and holds three patent awards.

To learn more, contact Joe at:

---

Joe Hanson is the director of business development for Stretch. Previously, Joe spent eight years with Altera, serving as director of marketing for system level tools and director of marketing and applications for the Excalibur business unit, Altera’s embedded processor solutions group. He has BS degrees in Electrical Engineering and Biological Sciences and holds three patent awards.

To learn more, contact Joe at: