RISPP: Rotating Instruction Set Processing Platform

Lars Bauer, Muhammad Shafique, Simon Kramer and Jörg Henkel
University of Karlsruhe, Chair for Embedded Systems, Karlsruhe, Germany
{lars.bauer, shafique, henkel} @ informatik.uni-karlsruhe.de

ABSTRACT

Adaptation in embedded processing is key in order to address efficiency. The concept of extensible embedded processors works well if a few a-priori known hot spots exist. However, they are far less efficient if many and possible at-design-time-unknown hot spots need to be dealt with. Our RISPP approach advances the extensible processor concept by providing flexibility through runtime adaptation by what we call “instruction rotation”. It allows sharing resources in a highly flexible scheme of compatible components (called Atoms and Molecules). As a result, we achieve high speed-ups at moderate additional hardware. Furthermore, we can dynamically trade-off between area and speed-up through runtime adaptation. We present the main components of our platform and discuss by means of an H.264 video codec.

Categories and Subject Descriptors

C.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems

General Terms: Performance, Design

Keywords: ASIP, extensible embedded processors, run-time adaptation, reconfigurable computing

1. INTRODUCTION AND RELATED WORK

Embedded processors are key for rapidly growing application fields ranging from automotive to personal mobile communication/computation/entertainment etc. In the early 1990s, the term ASIP has emerged denoting processors with an application specific instruction set. They present a far better efficiency in terms of performance/area, MIPS/mW compared to main stream processors and eventually make today’s embedded (and often mobile) devices possible. The term ASIP comprises nowadays a far larger variety of embedded processors allowing for customization in various ways including a) instruction set extensions, b) parameterization and c) inclusion/exclusion of predefined blocks tailored to specific applications (like, for example, an MPEG-4 decoder) [14]. Tool suites and architectural Molecules are used for embedded customizable processors with different flavors are available from major vendors like Tensilica [8], CoWare/LisaTek [2], ARC [1], 3DSP [5] and Target [4] as well as academic approaches like ASAP Meister (PEAS-III) [11], LIISA [3], [12] and Expression [13]. A generic design flow of an embedded processor can be described as follows: 1) an application is analyzed/profiled, ii) an extensible instruction set is defined, iii) extensible instruction set is synthesized together with the core instruction set, iv) retargetable tools for compilation, instruction set simulation etc. are (often automatically) created and application characteristics are analyzed, v) the process might be iterated several times until design constraints comply.

These approaches assume that customizations are undertaken during design-time with little or no adaptation possible during run-time. Our project combines the paradigms of extensible processor design with the paradigm of dynamic reconfiguration in order to address the following concerns in embedded processing: a) an application might have many hot spots that are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

An overview for the benefits and challenges of ASIPs is given in [14], [15]. The architecture [16] uses a library of reusable instructions which are manually designed, but therefore of high quality. In [17], [18] the authors describe methods to generate special instructions from operation patterns matching. [19] introduces an estimation model for area, overhead, latency, and power consumption under a wide range of customization parameters. In [20] an approach for a instruction set description on architecture level is proposed, which avoids inconsistencies between compiler and instruction set simulator. The authors in [21] investigate local memories in the functional units which are then exploited by special instructions.

On the other side, we utilize the techniques of reconfigurable computing for our approach. Overviews can be found in [22], [23]. In the scope of this paper we focus on extensions of CPUs with reconfigurable fabrics which are then made available to the programmer. An overview for this specific area of reconfigurable computing is given in [24]. Still, most of the CPU extensions are coarse-grained. The Molen Processor couples a reconfigurable processor to a core processor via a dual port register file and an arbiter for shared memory [25]. The run-time reconfiguration is explicit controlled by additional instructions in the application. The OneChip project [26] and its successor OneChip98 [27] use a Reconfigurable Functional Unit (RFU) to utilize reconfigurable computing in a CPU. This RFU is coupled to the host processor and obtains its speed-up mainly from memory streaming applications [27]. A first commercial product is presented in [6] although it concentrates on startup- instead of run-time reconfigurability.

The rest of the paper is organized as follows: In section 2, we present the motivation for our RISPP project. The Atom Model and Special Instruction Composition are presented in section 3. Section 4 describes how to add the Special Instructions Forecasts at compile-time. The run-time architecture is explained in section 5. We show a case study and results in section 6 and conclude our paper in section 7.

2. MOTIVATIONAL CASE STUDY

Extensible/Configurable Processors have played a significant role to target the multifaceted applications for low power. But one of the strict shortcomings of this domain is the fixed hardware for hot spots along the whole application span. The special instructions’ (SIs) hardware is fixed at design-time irrespective of the consideration of its utilization during the application’s run-time. To address this problem, another approach is coarse-grained dynamic reconfiguration which offers the adaptation at runtime. But in case of intricate applications like Multimedia TV where 30 frames are encoded and decoded at high video quality with Audio, Multiplexing, Pre- and Post- Processing the time schedule is very tight not allowing for time consuming reconfigurations.
Within this paper we are presenting a novel platform which gives origin to an innovative building blocks process. It is based on what we call the Rotating Instruction Set Processing Platform. It provides rotations of hardware for SIs at run-time. It eliminates the need of dedicated hardware for hot spots and presents a framework for a time-multiplexed utilization of rotating hardware resources. We will demonstrate our concept with the help of an ITU-T H.264 Video Encoder [9].

Figure 1 shows the comparison of an Extensible Processor and the RISPP hardware requirements (Gate Equivalents (GE)) while showing the performance maintenance using RISPP’s rotating concept. The distribution of processing complexity is shown in percentage for Motion Estimation (ME), Motion Compensation (MC), Transform and Quantization (TQ), and Loop Filter (LF), which are the major functional blocks, containing various hot spots in the H.264 Video Encoder. An Extensible Processor provides dedicated hardware resources in form of SIs which are constructed at design-time. It is obvious from Figure 1, that these resources are not used in the complete run-time span of the application. The hardware for LF, TQ, and MC is not used while processing ME, resulting in power/energy loss and overhead of silicon area.

One of the benefits of our approach is that we can support SIs using slower hardware executions. And it is apparent from Figure 1 that during the time for ME execution the partial hardware for MC is nearly completely loaded. When ME execution is finished, then the MC hardware is not completely loaded. Nevertheless, the available hardware resources can be used to immediately start executing MC in hardware, although not with the same performance of the full hardware. This is due to our fine-grained SI composition. When the faster hardware version will be loaded, the MC execution will gradually upgrade to a faster implementation. We call this as “Rotation in Advance” which is the analogon of pre-fetching or forecasting in the context of succeeding SIs.

Our Novel Contributions:
We present a novel platform for Dynamically Extensible Processors. Our RISPP approach features:

a) Special Instruction forecasting algorithm/scheme.

b) Composition and formal model of Special Instructions in form of Atoms and Molecules.

c) Run-time adaptation to different Molecules, depending upon the available Atoms and application requirements.

d) Optimized software Molecule for each Special Instruction.

Our RISPP approach offers a high degree of freedom for switching between software components, tasks, functions, and operations (from top to bottom level) while reducing the number of rotations and also forecasting the expected rotations well in advance. Furthermore, our forecast updating scheme maximizes the expectation / probability of the prediction.

3. SPECIAL INSTRUCTION COMPOSITION

We present a new vision of Special Instructions (SIs) in our Rotating Instruction Set Processing Platform. The SIs are realized as complex combinational atomic operations which are key for low power, high speed data paths with high reusability. We call a basic data path an Atom and the combinations of Atoms that implement an SI a Molecule. Comprehensive analysis of a set of SIs, sharing similar functional capabilities, leads to the conclusion that underlying hardware has a certain degree of similarity, which leads to the formulation of Molecules in terms of connecting Atoms. With enhanced reusability an Atom can be used for different Molecules, even of different SIs. Figure 2 shows a visualization using an ITU-T H.264 Video Encoder [9]. It is noteworthy that three different SIs can be implemented while sharing the same set of Atoms.

Figure 2: Molecule implementations of HT_4x4, DCT_4x4, and SATD_4x4 using different number of available Atoms

Let us take the example of the Hadamard Transformation for a 4x4 Macroblock (HT_4x4 in Figure 2), as it is used in the ITU-T H.264 Video Codec. For our designed SIs each HT_4x4 requires 4 Transformations and 4 Pack LSB MSB-executions to complete the computation. To implement this SI we have the choice to perform these atomic operations in parallel, sequentially or a combination of both1 giving the best trade-off for...
performance vs. area or power. Additionally, our RISPP approach moves the freedom from design-time to run-time, by offering different SI implementations (i.e. Molecules).

Hence, in this particular case, Hadamard Transformation is performed using a single call of the HT 4×4 SI in the application binary and the run-time system has multiple Molecule options to execute considering the available hardware, relative speed-up gain, required rotation effort and power savings.

3.1. Model for Molecule Assembly, Definitions

We will now explain our fine-grained SI composition on a formal basis to present the Molecule/Atom interdependencies and to simplify expressions in later used pseudo code. We define a data structure \( \mathbb{N}^n \) where \( \mathbb{N}^n \) is the set of all Molecules and \( n \) is the number of different available Atoms. For expediency we consider \( \vec{m}, \vec{o}, \vec{p} \in \mathbb{N}^n \) as Molecules with \( \vec{m} = (m_1, \ldots, m_n) \) where \( m_i \) describes the desired number of instances of Atom \( i \) to implement the Molecule (similarly for \( \vec{o} \) and \( \vec{p} \)). The operator \( \cup : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n \) with \( \vec{m} \cup \vec{o} = \vec{p}; \ p_i = \max \{ m_i, o_i \} \) describes a Meta-Molecule which contains the Atoms required to implement both \( \vec{m} \) and \( \vec{o} \). We name the resulting Molecule \( \vec{p} \) as a Meta-Molecule to distinguish it from the elementary Molecules which are dedicated to implement a specific SI. Since the operator \( \cup \) is commutative and associative with the neutral element \( (0, \ldots, 0) \), therefore \( \mathbb{N}^n \) is an Abelian semigroup.

The same is true for \( \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n \) being defined as \( \vec{m} \cap \vec{o} = \vec{p}; \ p_i = \min \{ m_i, o_i \} \). The relation \( \vec{m} \leq \vec{o} \) is defined to be true iff \( \forall i \in [1, n]: m_i \leq o_i \), As \( \leq \) is reflexive, anti-symmetric, and transitive, \( (\mathbb{N}^n, \leq) \) is a partially ordered set. For a set of Molecules \( M \subseteq \mathbb{N}^n \), the supremum is defined as \( \sup M = \bigcup_{m \in M} m \). The supremum from \( M \) is a Meta-Molecule with the meaning of declaring all Atoms that are needed to implement any of the Molecules in \( M \) such that \( \forall \vec{m} \in M: \vec{m} \leq \sup M \). The infimum is correspondingly defined as \( \inf M = \bigcap_{m \in M} m \) with the meaning of containing those Atoms that are collectively needed for all Molecules of \( M \). As any subset \( \mathcal{D} \subseteq M \subseteq \mathbb{N}^n \) has a well-defined supremum and infimum, \( (\mathbb{N}^n, \leq) \) is a complete lattice.

3.2. Molecule and SI Compatibility

Given these definitions, we can now combine multiple Molecules which are chosen to implement different SIs. To calculate the cost for constituting Molecules in hardware, we define the following two functions: The determinant of a Molecule is defined as \( |\vec{m}| = \sum_{i=1}^n i m_i \), i.e. the total number of Atoms that are required to implement \( \vec{m} \). Further, we consider already configured Atoms. We define the function

\[
\triangleright : \mathbb{N}^n \times \mathbb{N}^n \rightarrow \mathbb{N}^n; \quad \vec{m} \triangleright \vec{o} = \vec{p}; \quad p_i = \begin{cases} 0, & m_i, if \ 0, & m_i \geq 0, \ \text{else.} \end{cases}
\]

The created Meta-Molecule \( \vec{p} \) contains the minimum set of Atoms that additionally have to be offered to implement \( \vec{o} \), assuming that the Atoms in \( \vec{m} \) are already available.

To find a metric for the compatibility of SIs we have to consider that an SI in general consists of multiple Molecules with potentially different compatibilities. To handle this, we can compute statistical indicators like the average compatibility of multiple SIs. As we need to compute the compatibility of SIs at run-time (it depends on the currently available Atoms), we decided to represent each SI by a Meta-Molecule for the average Atom usage of its Molecules. By doing so we reduce the incompatibilities of the SIs to the incompatibilities of the representing Meta-Molecules. We define this representing Meta-Molecule for an SI \( S \) as \( \text{Rep}(S) = m, \vec{m} = \left( \left[ \sum_{i=1}^n i \right] \right) \), where \( |S|, S \subseteq \mathbb{N}^n \) is the number of hardware Molecules\(^2\) for the SI \( S \). These definitions will be used in section 4.2 to select forecast instruction out of a set of candidates and it is also used at run-time to select SIs that should be supported in hardware out of a set of requested SIs.

4. ADDING FORECASTS AT COMPILE-TIME

As the rotation time is in the range of milliseconds\(^3\) it is of paramount importance to forecast which SIs will be needed next. This allows starting the rotation earlier and thus the chance is increased that the hardware implementation can actually be used when the SI is required. Adding the Forecast Points (FCs) is done at compile-time on the Base-Block (BB) level of the application. Figure 3 shows the BB-graph from the AES application as it is automatically generated from our toolchain. The coloring visualizes profiling information for the execution time. To add FCs to this BB-graph we use the following scheme:

1. For each SI-Type determine a set of BBs as FC Candidates
2. For each BB remove those FC Candidates that are incompatible to the other FCs in the same BB
3. Choose FCs out of the FC Candidates and combine them to FC Blocks, which will ease the run-time computation effort

4.1. Determining FC Candidates

To determine whether a BB \( B \) is a good candidate to forecast a specific SI \( S \) we have to consider the following points:

- The probability to reach an execution of \( S \)
- The minimal, typical, and maximal temporal distance between \( B \) and any usage of \( S \)
- The expected number of executions when \( S \) is reached

If the probability at location \( B \) for executing \( S \) is high but the actual execution of \( S \) occurs only a few cycles after \( B \), then \( B \) is an inappropriate candidate for starting to initiate a rotation, since there is too little time left to finish rotation. In the case that \( B \) is temporally too far from the execution of \( S \) then a rotation would block the according Atom-Containers (ACs) for a too long period resulting in inefficiency.

Figure 3 shows an example BB graph. Outlined is the usage of an SI-type (circles). From profiling information we obtain the before-mentioned measurements distance, probability, and number of executions. These values are used to evaluate whether a BB represents a good FC Candidate or not. The probability is computed by a recursive algorithm that segments the BB graph into a tree of strongly connected components\((\text{SCC})\)\(^2\), recursively calls itself to compute the probability values of the SCCs and finally executes the algorithm proposed by Li/Hauck\(^2\) to compute the probability in the resulting tree. Our recursive addition to Li/Hauck is needed for our more fine-grained approach.

Next, the Forecast Decision Function FDF is executed. Inputs are the probability and the temporal distance. The output of the FDF is the minimal number of expected executions of the SIs that are needed for the current BB to turn it into an FC Candidate. This output is then compared with the computed number of expected executions. The FDF for the probability \( p \) and the temporal distance \( t \) is computed as\(^5\):

\[
\text{FDF}(p,t) := \text{offset} + \max \left\{ \begin{array}{ll} \frac{T_{\text{Rot}} - t}{T_{\text{TW}}} \cdot p, & \text{if } T_{\text{Rot}} > t, \\ \frac{T_{\text{Rot}} \cdot t}{p}, & \text{else} \end{array} \right\}
\]

\(^2\) as it is shown in the results in Section 6
\(^3\) e.g. loops or subroutine calls
\(^4\) for clarity some additional adjustment parameters are omitted
where $T_{Rot}$ is the average rotation time for $S$, $T_{SW}$ is the time that $S$ needs for software execution and offset is the minimum number of executions that are needed to make the rotation process energy efficient. The offset is computed as the energy cost for the rotation divided by the difference of the execution of $S$ in software and in hardware. The offset is then multiplied by a parameter with which a trade-off between energy efficiency and speed-up can be achieved:

$$\text{offset} = \alpha \left( \frac{E_{Rot}}{T_{SW} - T_{HW}} \right)$$

where

$$E_{Rot} = T_{Rot} \times P_{Rot} \times E_{Rot}$$

Output: Number of minimal SI usages to issue a Forecast Candidate

(number of SI usages to cast Candidate)

Temporal distance until usage of SI (relative to rotation time of SI with logarithmic scale) $[t / T_{Rot}]$

**Figure 4: Forecast Decision Function (FDF)**

An instance of the FDF is shown in Figure 4, where the temporal distance is shown relative to $T_{Rot}$ (logarithmic scale). In the range where the BB is closer than $1 T_{Rot}$ to the SI, a high number of expected executions are needed to become FC Candidate. In a range where the BB is farther than $10 T_{Rot}$, the number of needed executions is increasing to avoid an unnecessary long blocking of the Atom-Containers.

### 4.2. Trimming the FC Candidates and Choosing FCs

At run-time, we need to regard every single FC to trigger rotation decisions. But as every FC invokes the run-time system to re-evaluate, we need to reduce the number of FC Candidates in the first place. Good candidates for elimination are those which would contribute little to the overall speed-up or efficiency. The main observation is that one BB can contain FCs which are definitely not going to fit collectively into the according ACs. Then, those FCs whose SIs are providing the worst relation of speed-up and additional needed hardware resources are truncated. The pseudo-code is shown in Figure 5.

**Figure 5: Removing FC Points with worst Expected-Speed-up per hardware resources**

The $\text{ExpectedSpeedup}(m)$ execution in the algorithm reflects the difference in execution speed between the $\text{Molecules}$ and the software execution. It may happen that no single SI exists that would reduce the number of needed $\text{Atom-Containers}$, e.g. the three $\text{Molecules}$ $(1,0), (0,1), (1,1)$. In such a case the condition in line 11/12 aborts the algorithm because we do not want to remove a complete cluster of SIs out of the FCs as this would be a major reduction in the search space for the run-time decision system. The algorithm is executed for each SI-type individually and running on the transposed BB-graph\(^7\). For each not-yet-visited FC Candidate a Depth First Search is performed. Each node’s suitability as an FC Candidate is re-evaluated. If there is no suitability plus another FC Candidate is far\(^8\), then the preceding FC Candidate is turned into an actual FC. The FCs are finally annotated with the profiled probability, temporal distance, and the expected number of executions as initial values for the online phase.

### 5. RUN-TIME ARCHITECTURE

Besides the off-line phase where FCs (forecast points) are inserted (see section 4), the run-time phase actually rotates instructions depending on compile-time obtained information plus run-time updated information. Whether a rotation actually takes place depends on various parameters including the state of the dynamic hardware, on updated profiling information etc. The run-time architecture is described more detailed in [30].

The main tasks during the run-time phase are as follows:

a) Monitoring FCs and SIs in order to fine-tune the profiling information to reflect varying run-time situations.

b) Selecting and composing $\text{Molecules}$ from $\text{Atoms}$ to implement a subset of the forecasted SIs.

c) Scheduling the rotations and replacing $\text{Atoms}$ to accommodate new rotations.

Figure 6 shows an exemplary scenario to illustrate the run-time architecture by means of an ITU-T H.264 Video Codec [9]. An excerpt consisting of two Tasks A and B is seen sharing six $\text{Atom Containers}$ (ACs) to implement various $\text{Molecules}$ of their respective SIs. Among others it shows the video codec Task A employs the ‘4x4 Sum of Absolute Transformed Differences’ (SATD_4x4)-SIs that are implemented with varying numbers of instances out of the four $\text{Atoms}$ namely $\text{QuadSub}$, $\text{Pack}$, $\text{Transform}$, and SATD (more details in section 6). Task B instead is using two different SIs called $S_{I_{1}}$ and $S_{I_{2}}$ for brevity. We now discuss the run-time scheme at selected points of time $T$ in Figure 6.

$T_0$: Both tasks are supposed to be running in steady state. The ACs 0 to 3 comprise the $\text{Atoms}$ that are needed to implement the smallest $\text{Molecule}$ implementing SATD_4x4. The ACs 4 and 5 belong to B and are used to implement $S_{I_{1}}$.

$T_1$: $S_{I_{1}}$ in Task B is forecasted. That triggers a re-allocation of AC 3 and a subsequent rotation to implement this SI. Task A then executes SATD_4x4 in software.

$T_2$: $S_{I_{1}}$ has already been executed several times deploying AC 3 and reusing ACs 1 and 2. Then, the forecast states that $S_{I_{1}}$ is no longer needed. This triggers a re-allocation of ACs 3 to 5 of Task A which initiates rotation to achieve a hardware implementation of SATD_4x4.

$T_3$: $S_{I_{1}}$ of Task B is executed in HW utilizing the ACs 4 and 5 although they now ‘belong’ to Task A. This is possible, as they still contain the $\text{Atom}$ needed to implement that SI and they share the available HW resource. $T_5$: AC 3 is rotated and the execution of SATD_4x4 immediately switches from SW to HW implementation.

$T_6$: AC 4 is rotated and SATD_4x4 is now executed in an even faster $\text{Molecule}$ implementation.

This scenario shows that our run-time architecture does not follow a fixed rotation schedule. Instead, adaptation takes place during run-time according to the state of the HW and SW as well as to run-time updated forecasts. This scenario also shows that our approach is suitable for Multi-Mode systems with their changing demands on quasi-parallel executed tasks. By means of a video codec we show the advantages of RISPP.

### 6. CASE STUDY AND RESULTS

ITU-T H.264 [9] is one of the latest Video Coding Standards and also the most complex one (10x computation power increase relative to MPEG-4 simple profile encoding, 2x for decoding [10]).
The minimum requirement for this SI is 4 AToms so that all three Transform SIs can be executed using different control signals DCT and HT. Therefore, by just adding the shift elements multiplexed with two control signals DCT and HT we can make this Atom reusable so that all three Transform SIs can be executed using different control signals DCT and HT. The more important ATom will be used, the less important ATom will be used, and the more important ATom will be seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.

As the coefficients are not exceeding the 16-bit range we can use 16-bit data values packed into one 32-bit register. In case of one ATom Container there is an additional one 2x2 Hadamard Transform operation on the 16 DC coefficients. Therefore, one SI implements the functionality. For our case study in Figure 7, the SATD_4x4 can be executed using different control signals DCT and HT. The more important ATom is chosen as the best candidate and forwarded to SATD, the less important ATom is used, and the more important ATom is seldom needed.
embedded BlockRAM row, the corresponding bitstream (and thus rotation time) is significantly bigger, although its logic utilization is moderate. The rotation time generally corresponds to the memory transfer rate (e.g. 66 MB/s for Virtex-II) and the bitstream size and our concept would directly profit from faster rotation time, due to e.g. faster memory bandwidth.

Figure 11: SI execution time for a different amount of RISPP resources

The execution time of three SIs for different Molecule options and an optimized software implementation are presented in Figure 11 (logarithmic scale). It is noteworthy that the SIs with min. Atom requirements are more than 22 times faster than the optimized software implementation. When hardware resources during certain points at run-time are unavailable, this software routine is executed as an alternative. Once the minimum number of Atoms is loaded the special instruction starts using the hardware resources. Figure 12 shows the relative speed up for different Molecule options for the encoder application. The implementation with minimum Atom requirements is more than 300% faster than the one with optimized software implementation. Amdahl’s law prevents significant further speed-up when offering more Atoms. To overcome this we will consider additional SIs focusing on different hot spots in future work.

Figure 12: Allower performance for a different amount of RISPP resources

Table 2: Molecule composition of different SIs

<table>
<thead>
<tr>
<th>Molecule</th>
<th>HT_4x4</th>
<th>DCT_4x4</th>
<th>SATD_4x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qual/BU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pack</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trans</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sat</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. CONCLUSION

It can be seen that our approach of instruction rotation provides a high flexibility during run-time to adapt to varying application constraints. It is able to exploit the area-performance design space as it can choose between various Pareto-optimal solutions and switches between them. This is a feature that extensible processors (with predefined special instructions) cannot deliver. As a result, we achieve a very high efficiency as can be seen in Figure 12. It comes at the cost of a thorough and therefore computational extensive design-time analysis that carefully trades-off the boundary between design-time decisions and run-time flexibility.

8. REFERENCES

[1] Arc tangent processor. ARC Int’l (www.arc.com/configurablecores/)
[2] CoWare Inc, LISA Tec (www.coware.com/)
[3] LISA (http://servus.ert.rwth-aachen.de/lisa/)
[5] SP-5Flex DSP Core, 3DSP (www.3dsp.com/sp5_flex.shtml)
[8] Xtena processor, Tensilica Inc. (www.tensilica.com)
[12] A Hoffmann et al. “A novel methodology for the design of application-specific instruction-set processors (ASIPs) using a machine description language”, IEEE Trans. on CAD of Int. Circ. and Syst. 01
[27] JE Carrillo, P Chow “The effect of reconfigurable units in superscalar processors”, IJPP Symp. on FPGAs, 2001
[29] Z Li, S Hauck: “Configuration prefetching techniques for partial reconfigurable coprocessors with relocation and defragmentation”, Int'l Symp. on FPGAs, 2002
[31] P Brisk, A Kaplan, M Sarrafzadeh “Area-efficient instruction set synthesis for reconfigurable system-on-chip designs”, DAC 2004

Figure 13 explains the role of the number of Atoms to speed up the SI execution. Each entry in this figure corresponds to a Molecule for the specific SI where the detailed Molecule formulation is given in Table 2. Our RISPP architecture can dynamically adapt the trade between performance and resource requirements, by selecting the implementations of the SIs. This corresponds to a movement on the highlighted lines of Pareto-optimal Molecules, whereas an ASIP has to choose fixed SI implementations at design-time. These results excel any results that can be achieved with today’s extensible processors. It is due to a carefully selected boundary of design-time and run-time decisions.