Towards a Self-Reconfigurable Embedded Processor Architecture

Shady O. Agwa, Hany H. Ahmad, and Awad I. Saleh

***Abstract*—Embedded processors with fixed architecture have disadvantages: they are neither reusable nor are they flexible enough to match the specific needs of different application domains. The main technique employed to accelerate instruction execution in such processors is to add fixed hardware units, which may be useless for some applications yet insufficient for others. This additional hardware may affect area and power constraints badly. A self-reconfigurable architecture would be more flexible as the extended hardware execution units can be reconfigured or replaced at runtime to accelerate more than one algorithm avoiding area, power, and routing problems. Allowing the processor to replace at runtime, unnecessary acceleration execution units with others necessary depending on runtime profiling information, results in significant performance gains with only marginal increase in area and power. As proof of concept, in this paper we show that a significant execution time reduction of approximately 42.5% and a 37% reduction in executed instruction count are achievable for the "RGB to CMYK Conversion" benchmark, at the cost of a 20% increase in area, using the techniques described here. This preliminary investigation also indicates that, although power increases due to the additional hardware acceleration units employed, significant overall energy savings can be achieved (37.7% in our case study). These results were obtained using Tensilica's Technology Tools.**

I. Introduction

During the last decades a trend has emerged towards customization of the "general purpose" embedded processor to be more "application specific", in order to cope with the ever-increasing performance demands of embedded applications [1]. The addition of hardware acceleration units, however, has a negative effect on area – and often power – constraints. A better approach, hence, would be to determine precisely when and how the running application – or part of it – needs to be accelerated. This can be done by monitoring the instructions executed and measuring their weights; a process known as "profiling".

Runtime profiling allows the processor to collect results of monitoring and take the correct decision as to whether or not a particular part (or region) of the application needs to be accelerated [1] [2].

Runtime profiling only provides the necessary information; a so called "Runtime Reconfiguration Unit" is needed to actually add the required acceleration hardware dynamically. This approach should help overcome area problems since the processor will be able to replace unnecessary hardware acceleration units with others necessary, allowing the acceleration of more algorithms within the same area and without extra cost.

Some of the terminology related to this research is used rather loosely in the literature. We feel, at this point, that it is appropriate to clarify our interpretation and usage of the following somewhat overlapping terms:

- **Extensible:** the processor allows normal instructions to be extended into accelerated special instructions [2].
- **Adaptive:** the processor can tune itself to the running application according to its load [1].
- **Dynamic:** the processor's instruction set extension and execution unit configuration are done at runtime [1].
- **Self-reconfigurable:** One particular technique used to achieve "adaptability" through "dynamic extensibility".

The main objective of this research work – in its wider context – is to investigate the potential performance gains obtainable by utilizing processor adaptation and self-reconfigurability. The research goes through three major phases: Acceleration Techniques, Runtime Profiling, and Runtime Reconfiguration. **In this paper we focus on Acceleration Techniques and Runtime Profiling.** Runtime Reconfiguration will be the subject of our future work.

The remainder of this paper is organized as follows: Section II provides an overview of the processor core used and the proposed extensions for the purpose of this research. In Section III the "RGB to CMYK Conversion" benchmark is presented and analyzed as a stimulating case study. Next, a brief review of Tensilica's acceleration techniques as well as the hardware acceleration units used in the case study is introduced in Section IV. Runtime profiling is the subject of Section V, and Section VI concludes the paper with a summary of its major results.

II. General Overview Of Processor Core Architecture

A general purpose RISC processor core [3], as shown in Fig. 1 below, is used in this research:

Manuscript received July 31, 2009.

Shady O. Agwa is M.Sc. Student at Computer and Systems Section, Department of Electrical Engineering, Assiut University, Assiut, Egypt (corresponding author to provide phone: 002-012-4376775; e-mail: shady.agwa@yahoo.com).

Hany H. Ahmad is Assistant Professor of Computer Engineering at Computer and Systems Section, Department of Electrical Engineering, Assiut University, Assiut, Egypt (e-mail: hanya@aun.edu.eg).

Awad I. Saleh is Professor of Automatic Control at Computer and Systems Section, Department of Electrical Engineering, Assiut University, Assiut, Egypt(e-mail: awad.i.saleh@gmail.com).

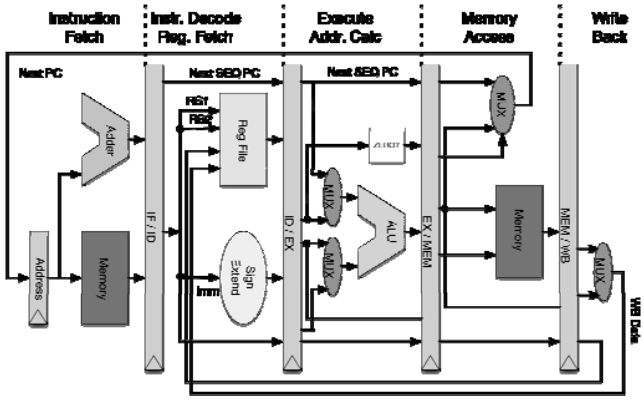

Fig. 1. General overview of a simple RISC processor core.

This simple core will be extended by adding new acceleration units built on a reconfigurable fabric, a reconfiguration unit, and configuration memory. Since traditional processors have been extremely difficult and time consuming to design and modify; and since our main objective at this stage is to study the characteristics of different profiling and acceleration techniques; we decided to use Tensilica's Technology Tools [4] as our basic development platform. In a later stage the design will be ported to a commercially available reconfigurable platform.

Tensilica's Tools:

Tensilica provides tools and techniques for designing processors with extended acceleration units. The algorithm to be executed will be written in C/C++ code, and the extended acceleration units are specified using TIE (Tensilica Instruction Extension) language. This will allow us to map our architecture to hardware to meet different speed, area, and power constraints.

Xtensa LX2 Core:

Tensilica's Xtensa LX2 is a processor core that borrows the best features of established RISC architectures and adds new Instruction Set Architecture (ISA) developments of its own as shown in Fig. 2.

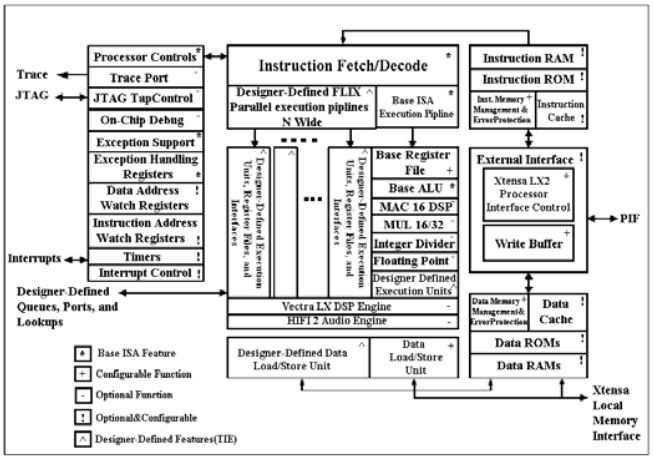

Fig. 2. The Xtensa LX2 Architecture: Designed for Configurability and Extensibility.

The Xtensa core ISA is implemented as a set of 24-bit instructions that perform 32-bit operations. Instructions can be extended to 32-bit, 64-bit, and 128-bit [4] [5].

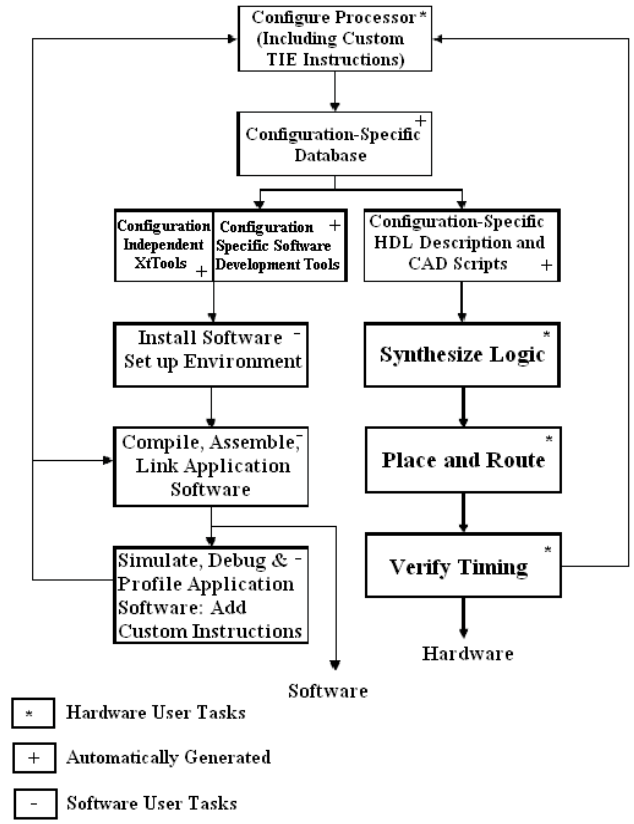

Fig. 3. Xtensa design flow.

As shown in Fig. 3, designing and creating the desired processor will pass through simple steps using Tensilica Tools [4] [5].

III. Motivating Case Study

We chose to use the "RGB to CMYK Conversion" benchmark from "EEMBC Consumer Bench™ Version 1.1" [6]. This benchmark provides opportunities for full-fury benchmark optimization. It is used for digital image processing in color printers and other digital imaging products.

- R, G, and B are 8-bit pixel color image inputs.
- Compute complementary colors c,m,y:
  1- c = 255 – R ;
  2- m = 255 – G ;
  3- y = 255 – B ;
- Find black level K:
  K = Minimum (c , m , y) ;
- Correct complementary colors:
  1- C = c – K ;
  2- M = m – K ;
  3- Y = y – K ;
- C, M, Y, K are 8-bit pixel outputs.
- All input and output values are in the range of

[0:255] as inputs will be [6]: R[0], G[0], B[0], R[1], G[1], B[1], … and outputs will be C[0], M[0], Y[0], K[0], C[1], M[1], Y[1], K[1], …

The algorithm is next divided into basic regions of code that will be written in C; TIE will be used to specify the acceleration hardware:

BEGIN ALGORITHM
Region # 1
Initializing R, G, B values.
End Region # 1

Region # 2
Calculate complementary colors c, m, y values.
End Region # 2

Region # 3

Region # 4
Find black level K value.
End Region # 4

Calculate correction of complementary colors C, M, Y, K values.
End Region # 3
END ALGORITHM

This is a very expensive routine that can be repeated many times for the same number of input pixels, and its processing time is almost linearly proportional to the number of input pixels [6]. In our case study, the number of input pixels was chosen to be 320x240 leading to 3x320x240 input values and 4x320x240 output values.

IV. Hardware Acceleration Units

To accelerate this algorithm we studied each of the code regions to determine whether it is a critical region or not.

Critical region: A set of instructions executed by the processor that consumes a large amount of time and contributes significantly to the total execution time of the algorithm [1]. Critical regions represent the best candidates for acceleration if processor performance is to be enhanced effectively.

Tensilica's acceleration techniques [4] [7]**:**

1- **Fusion:** Combines multiple computations into one multiple cycle operation. Many software instructions can be executed in one operation.
2- **SIMD:** Single Instruction Multiple Data deals with multiple data items in the same instruction to reduce processor memory traffic. A large amount of data can be loaded, stored, and computed in the same operation.
3- **FLIX:** Flexible Length Instruction Xtension is a technology by which we can make instructions with different lengths. The instruction length can be extended to 32-bit, 64-bit, or 128-bit. This user-defined instruction will be able to execute many operations at the same time.

The work described in this paper uses Fusion and SIMD acceleration only, FLIX was not used here.

Traditional acceleration techniques:

1- **Loop Unrolling:** Reduces predictions and branches with their conditional jumps. The number of instructions and computations will be reduced since many unnecessary loads, stores, conditional jumps, increments, or decrements are removed.
2- **Reducing Processor-Memory Traffic:** Fetching data from memory and storing data back to memory imposes a bottleneck on processor performance. Reducing memory traffic will improve overall system performance.

These traditional acceleration techniques are built in Tensilica's SIMD, and Fusion.

Coarse Grained Acceleration vs. Fine Grained Acceleration [8] [9]**:**

Fine grained granularity offers great flexibility when implementing algorithms in hardware. Fine grained acceleration units can be reused for more than one critical region, but their use will usually cause large penalties in terms of increasing area, power and time delay due to the associated routing problems.

Coarse grained acceleration units can be faster and more power efficient, but lack the flexibility offered by the fine grained approach.

We use coarse grained acceleration in this research. The rationale behind this decision is twofold. First, the lack of flexibility can be compensated for by the flexibility offered by the reconfiguration capability. Second, we believe that the fine grained approach would complicate the dynamic reconfiguration process due to the extra routing needed, a problem we are not willing to address at this stage.

Bank of Acceleration Units:

A bank consisting of 8 general and special-purpose (i.e. associated with a specific region) acceleration units was built for our case-study algorithm as shown in Table 1.

TABLE 1
Bank of Acceleration Units (8 units)

General Units	Special Units
Load 128-bit	Region #1 Acceleration (Initialze_RGB)
Store 128-bit	Region #2 Acceleration (Calculate_cmy)
Load 32-bit	Region #3 Acceleration (Calculate_CMYK_1,2)
Store 32-bit	

Acceleration results:

The following results are obtained by using Tensilica profiling tool.

In Table 2, the execution time reduction obtained for each region of the algorithm is given together with number of clock cycles with and without acceleration.

TABLE 2
FULL ACCELERATION RESULTS FOR CLOCK CYCLES

Region No.	Normal Execution (Clock cycles)	Accelerated Execution (Clock cycles)	Reduction (%)
Region #1	7,109,317	374,431	94.733
Region #2	5,990,424	446,442	92.547
Region #3	7,756,829	6,528,032	15.841
Region #4	---	---	No acceleration

Table 3 compares the total energy spent on each region with and without acceleration.

TABLE 3
FULL ACCELERATION RESULTS FOR TOTAL ENERGY

Region No.	Normal Execution (PJ)	Accelerated Execution (PJ)	Reduction (%)
Region #1	767,637,284	57,130,152	92.558
Region #2	828,812,553	69,904,509	91.566
Region #3	1,097,541,807	937,891,649	14.546
Region #4	---	---	No acceleration

Table 4 shows the savings in instruction count, the number of clock cycles, and for the number of loads and stores for the complete algorithm.

TABLE 4
FULL ACCELERATION RESULTS FOR WHOLE ALGORITHM

	Normal Execution	Accelerated Execution	Reduction (%)
No. Instructions	16,745,813	8,365,073	50.047
No. Clock Cycles	24,317,849	10,810,259	55.546
No. Loads	8,064,805	3,975,220	50.709
No. Stores	1,843,703	1,171,709	36.448

From the Tensilica profiling tests we found that Region # 4 will need no acceleration as the results will be nearly the same and from the previous results we noticed that Region# 3 acceleration is useful for energy and speed but the acceleration gain is not sufficient, so it will be recommended with many times of execution in the case of more than 320x240 input pixels, to reduce area consumed by hardware addition.

V. Runtime Profiling

Runtime profiling allows the processor to take important decisions about accelerating specific regions of the running algorithm. In this section we discuss briefly different runtime profiling approaches.

Coarse Grained Profiling vs. Fine Grained Profiling:

Fine grained profiling means that every instruction is monitored to measure its execution weight. If its weight exceeds the predetermined threshold, the instruction will be considered part of a critical region and it needs to be accelerated.

On the other hand, coarse grained profiling means monitoring a block of instructions as a whole and then determining if this block of instructions is a critical region or not depending on the predetermined threshold.

The threshold in both cases determines if acceleration is needed or not. We determine a suitable value for the threshold by experimentation at design time.

We use coarse grained profiling here, as it leads to coarse grained acceleration, and also to avoid the complexity of monitoring each individual instruction. Fine grained profiling would nearly double the number of executed instructions, since there will be an additional instruction for each one executed in the algorithm, in addition to the same number of extra storage elements.

Runtime Profiling Importance:

Runtime profiling plays a key role in our approach to self-reconfiguration. It provides the processor with the necessary flexibility to determine, at runtime, which regions of the running algorithm need to be accelerated and which regions do not. Without runtime profiling there would be no means to use the hardware acceleration units judiciously, as the information it provides is used by the reconfiguration unit to replace unnecessary acceleration units with others necessary.

The following paragraphs describe two different approaches to applying runtime profiling.

Profiling techniques:

- **Preparation mode profiling:**

In this technique, a "preparation run" precedes normal execution. In preparation mode, the processor starts profiling the algorithm to identify critical regions that need acceleration. The processor will then generate the required acceleration units and switch to normal

execution mode. Actual execution, thus, has to wait for results, decisions, and for hardware reconfiguration. In the literature, the terms "training mode" and "monitoring mode" have been used to describe the same concept [1] [2].

- **Pessimistic Runtime profiling:**

We propose a new runtime profiling technique which we call "Pessimistic Runtime Profiling". The processor will start at "full throttle" and consider all blocks of instructions to be critical regions that need to be accelerated (i.e. full acceleration). This fully accelerated run will be done only once and runtime profiling will provide the processor with sufficient data to identify the real critical regions that truly need acceleration. In the next phase of execution, all unnecessary acceleration units will be removed, and profiling will continue collecting data about the new critical regions during the full execution life time of the algorithm.

If the processor is pessimistic, it will not have to wait for results and decisions from the preparation phase, leading to execution-time and often power savings.

Runtime profiling results:

We assume that the algorithm will be repeated five times with the same number of input pixels.

We have five cases in our runtime profiling results:

1- **Case # 1:** Normal execution without any acceleration and without runtime profiling.
2- **Case # 2:** Fully accelerated execution without runtime profiling.
3- **Case # 3:** Accelerated execution with preparation mode profiling.
4- **Case # 4:** Accelerated execution with pessimistic runtime profiling.
5- **Case # 5:** Accelerated execution with pessimistic runtime profiling, with a change in threshold values to take into consideration the additional area cost.

"Table 5" below gives the number of clock cycles and the number of instructions executed for the five cases mentioned above, and Table 6 compares the results relative to case # 1.

TABLE 5
PROFILING RESULTS

	No. Clock Cycles	No. Instructions
Case # 1	121,579,398	83,723,913
Case # 2	54,041,448	41,820,213
Case # 3	94,495,104	69,422,774
Case # 4	65,034,008	49,068,578
Case # 5	69,949,063	52,754,908

TABLE 6
PROFILING SPEED UP RESULTS RELATED TO Case # 1

	Clock Cycles Reduction (%)	Instructions Reduction (%)
Case # 2	55.551	50.050
Case # 3	22.277	17.081
Case # 4	46.509	41.392
Case # 5	42.466	36.989

The following remarks can be made in view of the above results:

1- In Case # 3, using preparation mode profiling, only a modest execution time reduction was achieved (approx. 22%). This can be attributed to the fact that the preparation phase represents an overhead that is not effectively compensated for by the acceleration used in the following execution phase.
2- Pessimistic Runtime Profiling (Case # 4) achieves a significant execution time reduction of just over 46% which is twice that of Case # 3. This is a result of employing all the acceleration units in the first phase so that profiling data can be obtained and the associated decisions to remove unnecessary acceleration units can be made faster.
3- Case # 5 also uses Pessimistic Runtime Profiling but attempts to strike the balance between the performance gain in terms of speedup and the penalty inflicted in terms of the extra area consumed. Noting that acceleration of Region # 3 is area consuming yet its speed gain is comparatively low, we set the threshold for this region to be accelerated only in cases with greater than 320x240 input pixels.

Case # 5 is our target case, it saved about 34.6% of total area consumed by the total hardware acceleration units, and it also saved about 17.5% of the total special registers added for acceleration. Despite its area advantage, this case still gives us a significant execution time reduction of approximately 42.5% and an approximate 37% reduction in the number of instructions executed.

VI. Conclusion

The results presented in this paper support our view that combining coarse grained acceleration with runtime profiling is a valid approach to achieve significant speedups at moderate area expense. We are able to avoid the "slow startup" associated with preparation mode profiling by making the processor act pessimistically in the first execution phase by turning on all the acceleration units. Runtime profiling can be done by a completely separate hardware unit to avoid increasing the number of instructions executed by the normal execution units of the processor. This is one area currently under investigation and will be presented in a future paper. Building and evaluating the reconfiguration unit is our next major step towards our goal and will be carried out by porting

our designs to a state-of-the-art commercially available reconfigurable platform. A general overview of our new self-reconfigurable processor core architecture is shown in Fig. 4.

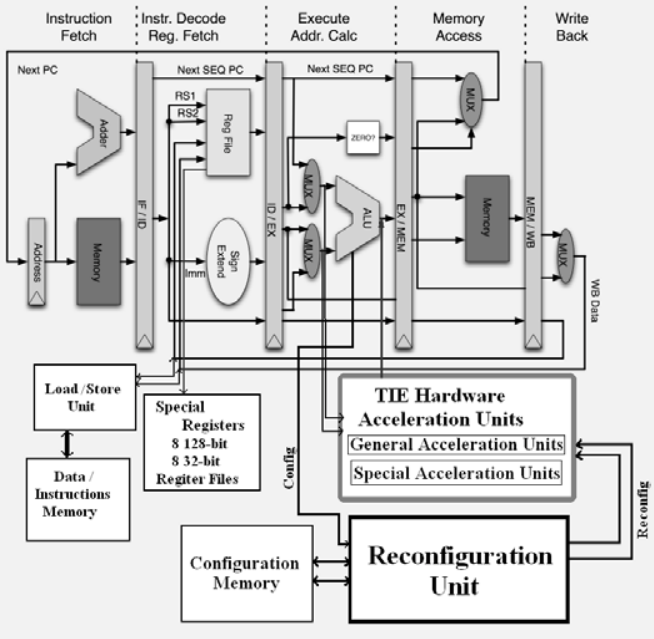

Fig. 4. General overview of our processor architecture

In addition to general core of RISC processor, hardware acceleration units will be added in our new core to accelerate critical regions detected by our pessimistic runtime profiling. A separated configurable load/store unit will be responsible of transferring data from and to Memory. As shown in Fig. 4, a special registers unit is added for holding 128-bit and 32-bit register files. There will also be a reconfiguration unit that will reconfigure hardware acceleration units, replacing unnecessary units with others necessary. Reconfiguration unit will get acceleration unit configurations from a special configuration memory.

Increasing the number of acceleration unit configurations will make the processor more general and more applications can be executed and accelerated by it. On the other hand, this will increase the size of the configuration storing unit. These topics also represent directions for future investigation.

REFERENCES

[1] Hamid Noori, Kazuaki Murakami, and Koji Inoue "An Adaptive Dynamic Extensible Processor," in Conf. Rec. 2005 IEICE General Conference, CPSY2005-29(2005-12).

[2] Lars Bauer, Muhammad shafique, Dirk Teufel and Jorg Henkel, "A Self-Adaptive Extensible Embedded Processor," in Conf. Rec. 2007 IEEE Int. Conf. SASO '07. First International Conference, pp. 344-350.

[3] MIPS Architecture, http://en.wikipedia.org/wiki/MIPS_architecture.

[4] Tensilica Tools, Tensilica Inc: http://www.tensilica.com/.

[5] Xia, Chen and Zhang, Qi, "H.264 Decoder Tensilica Configurable Embedded Processor Implementation", December 19, 2008. Web document found at: http://www.ee.hawaii.edu/~xrzhou/ee693e/slides/final_project.pdf.

[6] EEMBC Software Benchmark Data Book, http://www.eembc.org/.

[7] Steven Leibson, Tensilica Inc, "Customizable Processors and Processor Customization," in Processor Design: System-on-Chip Computing for ASICs and FPGAs, J.Nurmi (ed.), *©2007 Springer.* Ch. 8, pp. 149-175.

[8] Fabio Campi and Claudio Mucci, STMicroelectronics, ARCES, University of Bologna, "Run-Time Reconfigurable Processors," in Processor Design: System-on-Chip Computing for ASICs and FPGAs, J.Nurmi (ed.), *©2007 Springer.* Ch. 9, pp. 177-208.

[9] Koen De Bosschere, Wayne Luk, Xavier Martorell, Nacho Navarro, Mike O'Boyle, Dionisios Pnevmatikatos, Alex Ramirez, Pascal Sainrat, Andre Seznec, Per Stenstrom, and Olivier Temam. "High-Performance Embedded Architecture and Compilation Roadmap," HiPEAC Network of Excellence, in Transactions on HiPEAC I, P. Stenstrom (Ed.), LNCS 4050, 2007. *© Springer-Verlag Berlin Heidelberg 200,* pp. 5-29.