Low Power Embedded Software Optimization using Symbolic Algebra

Armita Peymandoust
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Tajana Simunic
HP Labs & Stanford University
1501 Page Mill Rd., MS 3U-4
Palo Alto, CA 94304

Giovanni De Micheli
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

{armita, tajana, nanni}@stanford.edu

Abstract

The market demand for portable multimedia applications has exploded in the recent years. Unfortunately, for such applications current compilers and software optimization methods often require of designers to do part of the optimization manually. Namely, the high-level arithmetic optimizations and the use of complex instructions are left to the designers' ingenuity. In this paper, we present a tool flow, OptAlg, that partially automates the optimization of power-intensive algorithmic constructs using symbolic algebra techniques combined with energy profiling. OptAlg is used to optimize and tune the algorithmic level description of an MPEG Layer III (MP3) audio decoder for the SmartBadge [2] portable embedded system. We show that our tool lowers the number of instructions and memory accesses necessary to run the MP3 code and thus lowers the system power consumption. The optimized MP3 audio decoder software meets real-time constraints on the SmartBadge system with low energy consumption. Performance increase of a factor of 7.27 and energy consumption decrease of a factor of 1.19 over the original executable specification has been achieved.

1. Introduction

Low cost with fast time to market is the top requirement in system-level design of embedded multimedia appliances. In embedded system design environment, the degrees of freedom in hardware are often very limited, whereas for software much more freedom is available. As a result, the primary requirement for embedded system-level design methodology is to effectively support code performance and energy consumption optimization. Automating as many steps in design of software from algorithmic-level specification is necessary to meet time to market requirements. Unfortunately, currently available compilers and software optimization tools cannot meet all designer needs. Typically, software engineers start with algorithmic level C code, often developed by standards groups, and manually optimize it to execute on the given hardware platform such that power and performance constraints are satisfied. Needless to say, this conversion is a time-consuming and often error-prone task, which introduces undesired delay in the overall development process. In addition, often compilers are unable to compile C code efficiently for embedded processors. Therefore, software engineers need to design key routines in assembly [1], which is extremely time consuming.

Our objective is to improve quality of the compiled code for embedded systems and facilitate the software design process. In this paper, we propose a new methodology based on symbolic manipulation of polynomials and energy profiling which reduces manual interventions. We apply a set of techniques previously used in algorithmic-level hardware synthesis [22] and combine them with energy profiling, floating-point to fixed-point data conversion, and polynomial approximation to achieve a new embedded software optimization methodology. The combination of these tools and standard compiler optimization techniques, allows for novel automatic code transformations.

As a motivating example, consider the code segment shown below:

```
  for i=1..3
       y = y + cos(i*x);
```

Using standard loop unrolling, the given code is transformed into the following:

```
  y = cos(x) + cos(2*x) + cos(3*x);
```

Now assume that for a given application cos(x) can be approximated into a Taylor series with three terms without noticeable degradation on the output. Many multimedia applications tolerate computational inaccuracy well, as long as the resulting effects (e.g. audio, video degradation) are limited. Therefore, y can be transformed into the following polynomial:

$$y = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + 1 - \frac{1}{2}2^2x^2 + \frac{1}{24}2^4x^4 + 1 - \frac{1}{2}3^2x^2 + \frac{1}{24}3^4x^4$$

This polynomial can be further simplified using the expand routine in symbolic algebra:

$$y = 3 - 7x^2 + \frac{49}{12}x^4$$

Assuming that the embedded processor used to execute this code has a MAC instruction, another symbolic routine called the Horner transform can be used on y:

$$y = 3 + (-7 + \frac{49}{12}x^2)x^2$$

The new equation can be mapped to one multiply instruction and two multiply-accumulates. Obviously, this mapping is much more efficient than three calls to the cosine library function. Unfortunately, to our knowledge, there is no tool currently available that can perform this simple optimization automatically. Thus it would be up to a designer to manually implement such optimizations.

This paper presents a tool-flow, called OptAlg, that automates the algebraic manipulations such as the one shown in the previous example. First, the energy critical code sections are identified using the energy profiler. Then, a tool like Fridge [4] can be used to transform floating-point data types into fixed-point, if necessary. Next, complex nonlinear arithmetic functions are approximated as polynomials such that the final output is within the acceptable tolerance limits. Finally, symbolic algebra is used to map the polynomial representations of the critical basic blocks to the instruction set available so that performance and power consumption are optimized. Note that more complex instructions (such as those developed by Tensilica tools [5]) and hardware accelerators can also be used during the mapping step.

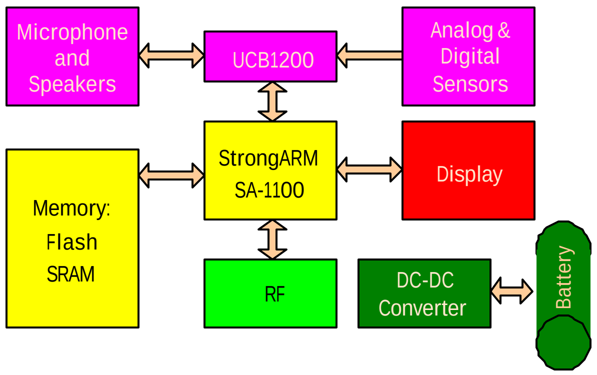


Figure 1. SmartBadge Architecture

We used OptAlg to optimize the implementation of MP3 software decoder so that it would meet real-time constraints on the SmartBadge [2]. The SmartBadge, as shown in Figure 1, is an embedded system consisting of Sharp's display, Lucent's WLAN link, StrongARM-1100 processor, RAM, FLASH, sensors, and modem/audio analog front-end on a PCB board powered by the batteries through a DC-DC converter. The outcome of this experiment is a higher performance MP3 decoder software on SmartBadge that uses less power. For an MP3 player, shorter than real-time execution time implies that lower voltage and frequency can still meet the real-time constraint. This in turn translates into longer battery life or lighter battery requirement for the embedded system.

The paper is organized as follows: Section 2 discusses previous work done in the area of software optimization for energy and performance. Section 3 presents the OptAlg flow, and gives an overview of each of its component. The results of MP3 decoder optimization for SmartBadge are presented in Section 4. Performance increase of a factor of 7.27 and energy consumption decrease of a factor of 1.19 over the original executable specification has been achieved. Finally, Section 5 summarizes the contributions of this work.

2. Related Work

Optimization of software performance and size has been utilized by designers for many years. Code optimization process translates a high level specification into optimized machine code for the target processor, often using compilers. There has been a lot of research on optimizing compilers in last few years [6]. Prototype research compilers have shown impressive results [7]. Most optimizing compilers target high-performance and/or general-purpose computers. Relatively little effort has been dedicated to create powerful optimizing compilers for embedded processors. Even though several researchers are studying automatic code optimization techniques for embedded processors [8,9], currently, most embedded processors (or DSPs) are programmed directly by expert programmers and code optimization is mostly based on human intuition and skill. In addition, due to recent growth in market demand for portable devices, optimization of software for power consumption is gaining in importance. As a result, one of the primary requirements for system-level design methodology of embedded devices is to effectively support code energy consumption optimization.

Several optimization techniques for lowering energy consumption have been presented in the past. A methodology that combines automated and manual software optimizations focused on optimizing memory accesses has been presented in [10]. Tiwari et al.[11,12] uses instruction-level energy models to develop compiler-driven energy optimizations at assembly level such as instruction reordering, reduction of memory operands, operand swapping in the Booth multiplier, efficient usage of memory banks, and series of processor specific optimizations. In addition, several other optimizations have been suggested, such as energy efficient register labeling during the compile phase [13], procedure inlining and loop unrolling [14] as well as instruction scheduling [15]. Work presented in [16] applies a set of compiler optimizations concurrently and evaluates the resulting energy consumption via simulation. All of these techniques focus on automated instruction-level optimizations driven by the compiler. Unfortunately, currently available commercial compilers have limited capabilities. Specifically, the arithmetic optimization, such as the example discussed in the Introduction, illustrates the limitations of current optimization tools.

Our proposed methodology and tools automates the process of identifying the code sections that would profit from algebraic optimizations, and then performs the optimization using symbolic techniques. Such symbolic techniques have been previously used in algorithmic level synthesis of data intensive circuits [22]. OptAlg uses the same principles previously used for high-level component mapping of hardware and applies them to the software optimization problem.

3. OptAlg Flow

Here we present a tool flow, OptAlg, which aims to automate most of parts of embedded system software optimization for the given embedded processor. Ideally, the software designer would code an algorithmic-level description of the software and have a compiler-like tool optimize it for the given platform. However, optimum implementation of calculation heavy routines for the particular hardware design is not possible with traditional compiler optimizations alone. Commonly, the designer has to do most of such optimizations by hand. Automating even a portion of the optimization process can save much design time.

OptAlg embodies a set of tools that enable the optimization process. Figure 2 shows the OptAlg flow. The first step in the optimization is to check if software data representation matches the hardware implementation. Most embedded processors support only fixed point computation, but many multimedia algorithms that are currently being implemented on portable systems utilize floating point operations. The profiler, described in Section 3.2, detects if data representation is an issue in a matter of seconds. Then, if needed, floating point operations can be replaced with fixed point using a tool such as Fridge [4]. The next step is to profile the code using the energy profiler. The output of profiling identifies target routines for optimization. Next, basic blocks of the critical routines are identified, and if needed, reformulated using polynomial approximation techniques. Accuracy of optimization has to be checked against the original code, as both during the data representation conversion and during the polynomial

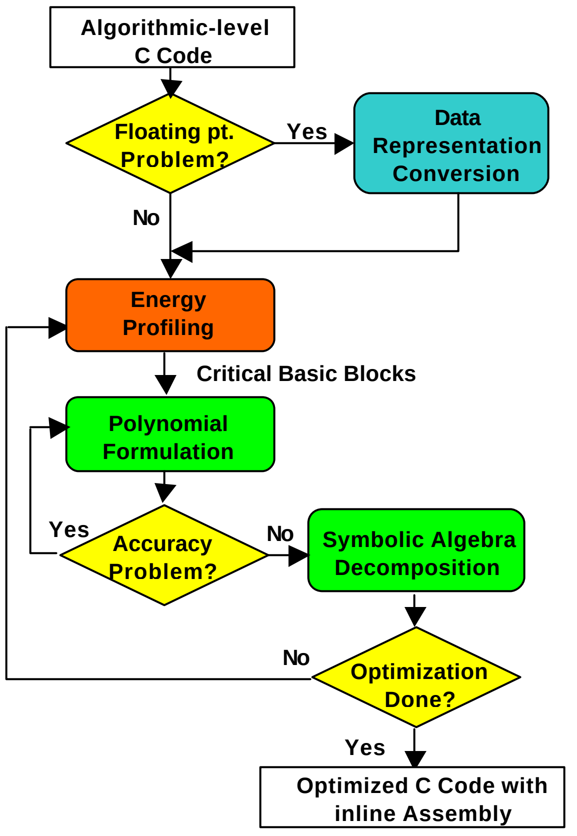


Figure 2. OptAlg Tool Flow

formulation, some rounding occurs. Once accuracy is satisfactory, resulting polynomials are decomposed into a sequence of instructions available on the particular hardware by novel symbolic techniques discussed in Section 3.4. Finally, another check is done using the profiler to see if the code has been sufficiently improved in terms of energy consumption and performance. Typically, it takes at most a few iterations to obtain the optimized code.

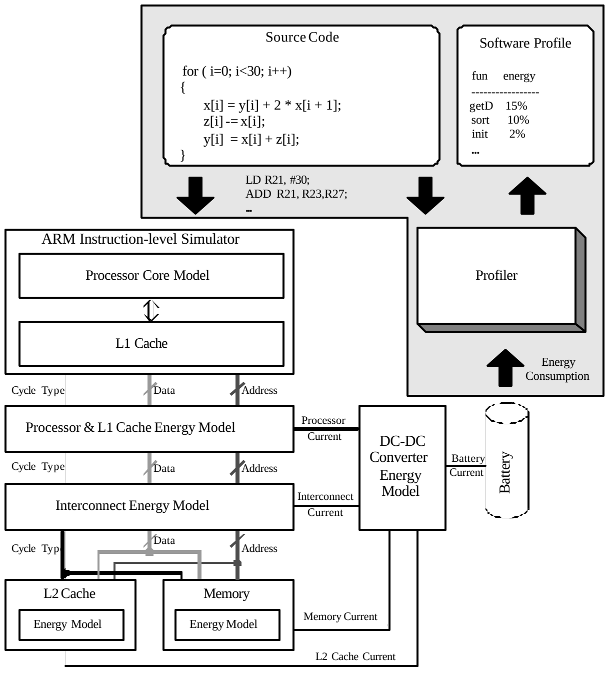
Our key contribution in OptAlg is a new method for basic block optimization using symbolic polynomial manipulation algorithms. We should also note that OptAlg is compliant with other software optimization techniques. Additional benefits are gained by combining traditional complier optimization algorithms with symbolic polynomial decomposition. The next sections described each portion of OptAlg in detail.

3.1 Data Representation Conversion

Signal processing algorithms are generally developed used ANSI-C with IEEE floating-point data types. However, these algorithms are often implemented in embedded systems using fixed-point data types in order to meet the power, cost, and performance requirements. Converting a floating-point algorithm to a fixed-point algorithm is a time consuming and error prone task. Facilitating and semiautomating this conversion has been targeted by tools such as Fridge (a.k.a. CoCentric fixed-point designer) [4]. Such tools use interpolative analysis to convert floating point C code into appropriate fixed-point code to reduce the manual work and the number of simulations required. The designer annotates the critical variables of the design with the desired bit width and uses Fridge to automate the rest of the conversion through simulation and numerical analysis.

3.2 Energy Profiling

Code optimization requires extensive program execution analysis to identify energy-critical bottlenecks and to provide feedback on the impact of transformations. Profiling is typically used to relate performance to the source code for CPU and L1 cache [17]. Energy profiler enables easy identification of the most energy-critical procedures, as well as analysis of the impact optimization transformations might have not only on the processor energy consumption, but also on the memory hierarchy and the system busses.



```
for ( i=0; i<30; i++)
{
      x[i] = y[i] + 2 * x[i + 1];
      z[i] -= x[i];
      y[i]  = x[i] + z[i];
}
```

```
LD R21, #30;
ADD  R21, R23,R27;
...
```

```
fun    energy
-----------------
getD   15%
sort   10%
init    2%
...
```

Figure 3. Profiler Architecture

The profiler exploits a cycle-accurate energy consumption simulator [18] to relate the embedded system energy consumption and performance to the source code. Thus, it can be used for analysis (i.e., to find energy-critical sections of the code), and for validation (i.e., to assess the impact of each code optimization). Estimation results were shown to be within 5% of measured energy consumption on the SmartBadge hardware.

The profiler architecture is shown in Figure 3. Source code is compiled using a compiler for a target processor. The output of the compiler is the executable that the cycle-accurate simulator executes (represented as assembly code that is input into the simulator) and a map of locations of each procedure in the executable that a profiler uses to gather statistics (the map is correspondence of assembly code blocks to procedures in 'C' source code). The profiler works concurrently with the cycle-accurate simulator. It periodically samples the simulation results (using sample interval specified by the user) and maps the energy and performance to the function executed using information gathered at the compile time. Once the simulation is complete, the results of profiling can be printed out by the total energy or time spent in each function.

With the profiler, OptAlg can obtain energy consumption breakdown by procedures in the source code and thus can quickly identify the areas in the source code whose optimization can provide largest overall energy savings. In addition, with the cycle-accurate simulator that is at the heart of the profiler, OptAlg can get detailed information about performance and energy consumption of smaller subsections of code. Therefore, in this step, the critical basic blocks of the power hungry procedures are identified. These basic blocks are then passed on as inputs to polynomial approximation and symbolic mapping tools which can optimally map that section of code to assembly instructions in matter of minutes.

3.3 Polynomial Formulation

The energy profiler helps detect the critical basic blocks of the code. The next step is to map the critical basic blocks to assembly instructions so that optimum power consumption and performance are achieved. The mapping algorithm, described in Section 3.4, uses the principles of symbolic algebra and Gröbner basis. The inputs to the mapping algorithm are the polynomial representations of the critical basic blocks and the polynomial equivalence of the arithmetic assembly instructions. The polynomial formulation step prepares the input to the symbolic mapping algorithm by calculating a polynomial representation for the critical basic blocks.

The polynomial representation of a basic block can be directly extracted from the C code if the basic block calculates a linear function. If the basic block performs a series of bit manipulations or Boolean functions previously developed algorithms based on interpolation [24] can be used to formulate the equivalent polynomial representation. When the basic block implements a nonlinear function, we use an approximation, such as the Taylor or Chebyshev series expansion, as the polynomial representation for that basic block.

The chosen polynomial approximation has to be verified by simulation to make sure that the software constraints, such as audio quality, are satisfied. A good approximation can result in even greater performance and power improvements for multimedia applications, since these applications can tolerate a slight degradation in the resulting output. For example, to verify the accuracy of the MP3 decoder we have used the compliance test provided by the MPEG standard [19]. The range of RMS error between the samples defines the compliance level. If the approximation is not sufficient to satisfy the accuracy constraints, the quality of approximation is changed and verified again through simulation.

3.4 Symbolic Optimization

At this step, the polynomial representations for critical basic blocks of the code are available. Arithmetic assembly instructions of the target embedded processor are also represented as polynomials. The goal of the symbolic optimization step is to decompose the polynomial representations of the basic blocks into the polynomial representations of available assembly instruction so that power consumption and performance are optimized.

Symbolic computer algebra is a set of algorithms capable of algebraic manipulation of expression containing undetermined values (symbols), such as variable `x` in `(x+1)*(x-1)`. Several commercial symbolic computer algebra systems are available on the market; Maple [20] and Mathematica [21] are most widely used. The algebraic object to be symbolically manipulated is a multivariate polynomial that represents a critical basic block identified in the profiling step. Most interesting symbolic polynomial manipulations are based on Gröbner bases [23]. Gröbner bases also solve variable elimination in a set of polynomials and ideal membership problems, which is the core of simplification modulo set of polynomials [23].

$$d = \cos\left(\frac{\pi}{72}\left(2p+1+\frac{N}{2}\right)(2m+1)\right) \quad \textbf{(1)}$$

In order to show the power of symbolic algebra, let us consider a simple example. Consider a basic block implementing Equation **1** and an instruction set that includes add, multiply, subtract, MAC, and square. The basic block has been approximated to the polynomial shown in Equation **2** in the previous step of the OptAlg flow.

$$x = \frac{\pi}{72}\left(2p+1+\frac{N}{2}\right)(2m+1)$$
$$d = \frac{1-\frac{3665}{7788}x^2+\frac{711}{25960}x^4-\frac{2923}{7850304}x^6}{1+\frac{229}{7788}x^2+\frac{1}{2360}x^4+\frac{127}{39251520}x^6} \quad \textbf{(2)}$$

The *simplification modulo set of polynomials* routine can be used to map the numerator and denominator of Equation **2** to the available instruction set. In order to comply with Maple terminology, we call the routine *simplify* and the set of polynomials *side relations* as shown below.

```
> dn:=1-a*x^2+b*x^4-c*x^6:
   siderels:={y=b-c*x^2, z=-a+y*x^2}
> simplify(dn, siderels,[x,y,z]);
  1+z*x^2
```

Note that side relations are a subset of our instruction set. If the side relation set is changed, other possible solutions for the specification may be found. The result indicates that:

```
dn:=1-a*x^2+b*x^4-c*x^6:=1+z*x^2
      :=1+(-a+(b-c*x^2)*x^2)*x^2
```

Therefore, the numerator of Equation **2** can be mapped to one square and three MACs instructions. Assuming `R1`, `R2`, `R3`, `R4`, and `R5` hold `1`, `-a`, `b`, `-c`, and `x`, respectively, the resulting assembly code is:

```
SQUARE R6, R5
MAC R7, R3, R2, R6
MAC R8, R2, R7, R6
MAC R7, R1, R8, R6
```

The original basic block shown in Equation **1**, takes 2384 cycles to run on the StrongARM-1100 processor. The optimized version of this basic block executes in only 1257 cycles. Thus we have achieved an improvement of 47% for this simple example.

Choosing the side relation set is a non-trivial task. In previous work [22], an algorithm was introduced to select the side relation set such that the hardware implementation of a polynomial representing a (portion of) data path with a given component library has minimal critical path delay. In this paper, we use the algorithm to optimize mapping of the critical basic blocks of software to assembly instructions. This method performs even more effectively when a rich instruction set (e.g. ASIP or hardware accelerator) is available.

Figure 4 gives a brief overview of the mapping algorithm. Inputs to the algorithm are the polynomial representations of the critical basic blocks and the polynomial representations of the given instruction set. The goal of the symbolic mapping algorithm is to decompose the polynomial representation of the critical basic block (*CBB*) into polynomial representations of the instruction set (*IS*) such that power consumption and delay are minimized. The power and number of cycles it takes to execute each instruction in the *IS* are given to the mapping algorithm as constants.

Decomposing *CBB* into elements of *IS* is synonymous to simplifying *CBB* modulo elements of the set *IS* as side relations. Thus, the symbolic algebra routine used for this decomposition is *simplification modulo set of polynomials*. Since different side relation sets result in different mappings of a basic block, the algorithm uses branch-and-bound method to reduce the search space. The bounding function is the best execution time or power dissipation seen so far. Expression manipulation techniques available in symbolic algebra are used as heuristic guidelines for choosing the side relation set. Initially, tree-height reduction, factorization, expansion, and Horner-based transform are applied to *CBB* resulting in several polynomial representations of the same basic block. Each of these representations can result in the desirable mapping based on the available instruction set. Starting with the inputs of the basic block, we try covering the expression tree with the available instructions. We choose all instructions that cover the inputs and a portion of the expression tree as a side relation. If the CBB is not yet fully mapped, we decompose the result without further guidance from the expression tree. This algorithm was implemented in C with calls to Maple V for the symbolic manipulations used.

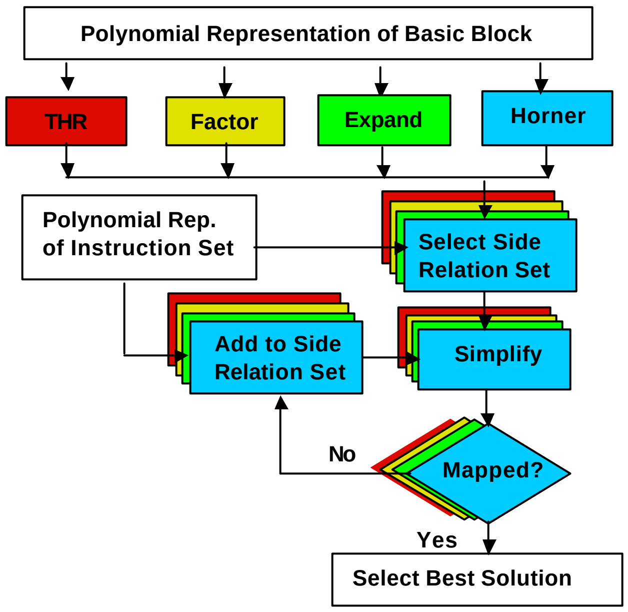


Figure 4. Overview of the mapping algorithm

4. Results

We have optimized several portions of the algorithmic level C program of MP3 decoder using OptAlg in order to run it on the SmartBadge embedded system [1] shown in Figure 1. We obtained the original MP3 audio decoder software from the International Organization for Standardization [3]. Our design goal was to obtain real-time performance with low energy consumption while keeping full compliance with the MPEG standard.

The first step in decoding MP3 stream is synchronizing the incoming bitstream and the decoder. Huffman decoding of the subband coefficients is performed before requantization. Stereo processing, if applicable, occurs before the inverse mapping which consists of an inverse modified cosine transform (IMDCT) followed by a polyphase synthesis filterbank.

The manual optimization for MP3 decode on the SmartBadge, as implemented in [18], required the designer first to implement a fixed point library and replace all floating point operations with fixed point, and then to fully understand the details of the SmartBadge's design, so that the critical arithmetic operations could be manually optimized, often with inline assembly code. The full optimization process took a number of days when done manually. In contrast, OptAlg partially automates the same process and its iterations take only a few minutes.

The first step in the optimization flow is to check if floating-point data types are suitable for the given platform. Since SmartBadge's processor, StrongARM 1100, can only emulate the floating-point operation, there is a need for data representation transformation. The code was converted to use fixed-point arithmetic. It was verified through simulation that 27-bit precision fixed-point data-types are sufficient to meet the compliance test provided by MPEG standard [19]. The range of RMS error between the samples defines the compliance level.

Table 1. Profiling the Original MP3 Code

Function	% Power
SubBand	49%
IMDCT	26%
Dequant	5%
Antialias	0.74%
Hufman	0.47%
SynFilter	0.26%

Energy profile of the original source code highlights the critical procedures of the code and their critical basic blocks. Table 1 shows a list of critical procedures and their impact on the final power consumption. These sections of the code are selected for further optimization. In the next step, we use polynomial approximations for the non-linear calculations in the critical basic blocks. This approximations satisfy the MPEG compliance test. The polynomial representations of the critical basic blocks are next mapped into the assembly instructions by algorithm described in Section 3.4. It is important to note that StrongARM compiler was not capable of using the MAC instruction effectively. Therefore, the result of the decomposing algorithm was inserted as inline assembly in the C code.

The results of optimizing critical functions of the MP3 code by OptAlg are compared with the original results from straightforward compilation in Table 2. As we can see 17-56% improvement has been achieved using the OptAlg methodology. This much improvement would previously be possible only thorough manual software optimization. The automation introduced by OptAlg drastically reduces the embedded software optimization cycle.

Table 2. MP3 Results by Optimized Function

	Performance (#cycles)			Energy Consumption (mJ)		
Function	original	optimized	%imp	original	optimized	%imp
MDCTCoeff	1454550	957051	34.2	1.051	0.922	12.2
FilterS	5263831	4196853	20.3	3.630	3.319	8.6
Power3/4	14135	5380	61.9	0.040	0.009	76.6
Dequant	650894	421976	35.2	0.940	0.747	20.5
SubBandSyn	155204	70633	54.5	1.015	0.306	69.8
MDCT	63583	31954	49.7	0.101	0.051	49.6

Table 3 compares the power consumption and performance of three versions of the MP3 decoder running on the SmartBadge. The first column corresponds to the original MP3 code obtained from the standards organization. The second column corresponds to the optimized code using the OptAlg flow. The third column is the hand optimized code. It can be seen that OptAlg flow can achieve results very close to manually optimized software. However, manual optimization of the code was done in number days while OptAlg optimization took only matter of hours.

Table 3. MP3 Combined Optimization Results

Comparison	Original	OptAlg	Manual
Energy (mWhr)	0.446	0.375	0.36
improvement factor	1.0	1.189	1.24
Performance (s)	68.5	9.42	8.20
improvement factor	1.0	7.27	8.353

5. Conclusion

The contribution of this paper is a tool flow, OptAlg, that automates energy and performance optimization of arithmetic sections of code for implementation on a given embedded processor. Our tool combines energy profiling, automated data representation conversion, derivation of polynomial representation and symbolic algebra. Energy profiling is necessary to identify critical sections of code that need to be optimized. For more complex arithmetic functions, the conversion into a polynomial representation is needed in order to enable symbolic algebra techniques. Symbolic computer algebra decomposes the polynomial representation of the basic blocks into a set of instructions available on the embedded processor.

We gave an example of application of our tool, OptAlg, to the optimization of MP3 audio decoding for the SmartBadge [2] portable embedded system. The final MP3 audio decoder is fully compliant with the MPEG standard and runs in real time with low energy consumption. Using OptAlg for source code optimization we have been able to increase performance by a factor of 7.27 while decreasing energy consumption by a factor of 1.19. This improvement is primary achieved by reducing the number of instructions and memory accesses executed in critical basic blocks. The technique presented in this paper can be easily used in conjunction with other compiler optimization techniques [6].

6. References

[1] P. G. Paulin, C. Liem, M. Cornero, F. Nacabal, and G. Goossens, "*Embedded software in real-time signal processing systems: application and architecture trends*," Proc. IEEE, vol. 85, no. 3, pp. 419-435, Mar. 1997.

[2] G. Q. Maguire, M. Smith, H. W. Peter Beadle, "SmartBadges: a wearable computer and communication system", *6th International Workshop on Hardware/Software Codesign*, Invited talk, 1998.

[3] "Coded representation of audio, picture, multimedia and hypermedia information", *ISO/IEC JTC/SC 29/WG 11*, Part 3., May 1993.

[4] M. Willems, H. Keding, T. Grötket, and H. Meyr, "Fridge: An interactive Fixed-Point Code Generation Environment for HW/SW CoDesign", *Proceedings of Int. Conf. On Acoustics, Speech, and Signal Processing*, 1997.

[5] Albert Wang, Earl Killian, Dror Maydan, Chris Rowen, "Hardware/Software Instruction Set Configurability for System-on-Chip Processors", *Design Automation Conference*, pp. 184-190, 2001.

[6] S. Muchnick, *Advanced Compiler Design and Implementation*, Morgan Kaufmann, 1997.

[7] M. Hall, J. Anderson, S. Amarasinghe, B. Murphy, S. Liao, E. Bugnion, M. Lam, "Maximizing multiprocessor performance with the SUIF compiler", *IEEE Computer* vol. 29, no. 12, pp. 84—89, Dec. 1996.

[8] P. Marwedel and G. Goossens. *Code Generation for Embedded Processors*. Kluwer Academic Publishers, 1995.

[9] R. Leupers, *Retargetable Code Generation for Digital Signal Processors*, Kluwer Academic Publishers, 1997

[10] F. Catthoor, S. Wuytack, E. De Greef, F. Balasa, L. Nachtergaele, A. Vanduoppelle, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*, 1998, Kluwer Academic Pub.

[11] V. Tiwari, S. Malik, A. Wolfe, M. Lee, "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing Systems*, vol 13, no.2—3, pp.223—2383, 1996.

[12] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization", *IEEE Transactions on VLSI Systems*, vol. 2, no.4, pp.437—445, December 1994.

[13] H. Mehta, R.M. Owens, M.J. Irvin, R. Chen, D. Ghosh, "Techniques for Low Energy Software", *International Symposium on Low Power Electronics and Design*, pp. 72—75, 1997.

[14] Y. Li and J. Henkel, "A Framework for Estimating and Minimizing Energy Dissipation of Embedded HW/SW Systems", *Design Automation Conference*, pp.188—193, 1998.

[15] H. Tomyiama, H., T. Ishihara, A. Inoue, H. Yasuura, “Instruction scheduling for power reduction in processor-based system design”, *Design, Automation and Test in Europ*e, pp. 23—26, February 1998.

[16] M. Kandemir, N. Vijaykrishnan, M. Irwin, W. Ye, “Influence of Compiler Optimizations on System Power”, *The 27th International Symposium on Computer Architecture*, pp.35—41, 2000.

[17] Advanced RISC Machines Ltd (ARM), *ARM Software Development Toolkit Version 2.11*, 1996.

[18] T. Simunic, L. Benini, G. De Micheli, “Energy-Efficient Design of Battery-Powered Embedded Systems”, *Special Issue of IEEE Transactions on VLSI*, pp. 18-28, May 2001.

[19] ISO/IEC JTC 1/SC 29/WG 11 13818-4, “Information Technology, Generic Coding of Moving Pictures and Associated Audio: Conformance”, International Organization for Standardization, 1996.

[20] Maple, Waterloo Maple Inc., www.maplesoft.com, 1988.

[21] Mathematica, Wolfram Research Inc., www.wri.com, 1987.

[22] Omitted for blind review.

[23] T. Becker and V. Weispfenning, *Gröbner Bases*, Springer-Verlag, New York, NY, 1993.

[24] J. Smith and G. De Micheli, “Polynomial Methods for Component Matching and Verification”, *Proceedings of the International Conference on Computer Aided Design*, 1998.