MOVAL: A FRAMEWORK FOR TURNING DIGITAL SIGNAL PROCESSING ALGORITHMS INTO CUSTOM CHIPS

A. LIGTENBERG (Member EURASIP) and J. H. O'NEILL
AT&T Bell Laboratories, Holmdel, NJ 07733, U.S.A.

M. VETTERLI (Member EURASIP)
Laboratoire d'Informatique Technique, Ecole Polytechnique Fédérale de Lausanne, 16 Chemin de Bellerive, CH-1007 Lausanne, Switzerland

Received 10 July 1985
Revised 4 February 1986

Abstract. A framework supported by a set of tools for turning digital signal processing algorithms into custom chips is proposed. The framework, called MOVAL, integrates analysis, layout synthesis and validation and is based on a structured top-down design methodology covering seven succinct abstraction levels: the behavioral, data representation, space/time, hardware, symbolic and geometric (mask) descriptions, and the chip. The top four levels efficiently cope with the implementation trade-offs and are all written in the same high level language, currently "C". As a result, the design can be modeled using a mixture of components defined at different abstraction levels. This allows an efficient mixed-mode multi-level validation. The hardware description is unambiguous and is the key to (semi-) automatic synthesis of the layout. Furthermore, it generates test vectors for the symbolic layout, the mask and, finally, the chip. The validation is done automatically, based on back substitution. As an example of the proposed design methodology, the crucial steps of the implementation of a fast Fourier cosine transform algorithm are described.

Zusammenfassung. Ein Entwicklungskonzept mit Werkzeugen wird vorgeschlagen um Signalverarbeitungsalgorithmen in spezialisierte Chips umzuwandeln. Das Konzept, MOVAL genannt, integriert die Analyse, die Synthese und die Verifikation, und ist eine strukturierte 'top-down' Methodologie mit sieben Abstraktionsniveaus: Algorithmus, Datenpräzision, Raum/Zeit, Hardware, symbolisches Layout, Masken, und Chip. Die vier oberen Niveaus erlauben es, die verschiedenen Kompromisse einer Implementation zu erforschen, und sie sind in der gleichen hohen Sprache, "C", geschrieben. Deshalb kann der Design mit Komponenten modelisiert werden die auf verschiedenen Niveaus geschrieben sind. Dies erlaubt eine effiziente, 'mixed-mode, multi-level' Verifikation. Die hardware Beschreibung ist eindeutig und erlaubt eine (semi-) automatische Layout-Synthese. Damit können auch Testvektoren generiert werden, und dies sowohl für das symbolische Layout wie für die Masken und das Chip. Die Verifikation ist automatisch und basiert auf die Rückwärts-Substitution. Als Beispiel für diese Design Methologie werden die wichtigsten Etappen des Designs eines 'fast cosine transform' Chips gezeigt.

Résumé. Dans cette contribution, on propose un contexte de développement et certains outils afin de transformer des algorithmes de traitement du signal en circuits intégrés spécialisés. Ce contexte de développement, appelé MOVAL, intègre l'analyse, la synthèse du schéma ainsi que la validation. Il est basé sur une méthodologie 'top-down' structurée et faisant appel à sept niveaux d'abstraction: le niveau algorithmique, de précision finie, d'espace/temps, de matériel, de schéma symbolique, de masque, et finalement le niveau physique (circuit). Les quatre niveaux supérieurs permettent d'explorer efficacement les compromis possibles lors de l'implantation et sont tous quatre écrits dans un langage commun de haut niveau, le "C". Ainsi, on peut simuler un circuit en utilisant des éléments décrits à des niveaux d'abstraction différents. Ceci permet une validation en modes et niveaux mélangés. La description matérielle est non ambiguë et permet une génération (semi-) automatique du schéma symbolique. De surcroît, elle permet la génération de vecteurs de test, tant pour le schéma symbolique que pour le plan de masques ou le circuit physique. La validation est faite automatiquement par voie de substitution. Comme exemple, on montre les étapes principales du développement d'un circuit réalisant une transformation en cosinus rapide.

Keywords. Digital signal processing hardware, design methodologies, algorithm transformation, VLSI.

Preface

MOVAL, which is neither a magic work station concept nor an expert system, is still under construction. It is a framework or discipline supported by tools to translate digital signal processing (DSP) algorithms into specific purpose chips. The proposed method is based on the data-flow concept. Operation and memory elements are built in the hardware where they are required: This is in sharp contrast to the often-employed approach wherein a general digital signal processor architecture is programmed for the algorithm.

MOVAL allows the human designer to interact at all levels, in order to analyze the design trade-offs or to improve the synthesis of the layout. More precisely, MOVAL starts by helping to define, to refine, and to verify the initial concept. Then, for each following step in the abstraction hierarchy, the correctness of that description can be verified automatically. The layout is synthesized with the help of generators from MOVAL's hardware level description of the algorithm. The use of MOVAL ends only when it helps to check if the fabricated chip is working correctly. Thus, MOVAL can be regarded as a backbone, covering in a top-down way, the stages needed to translate DSP algorithms into working custom chips.

1. Introduction

Increased control over the fabrication process and the availability of powerful software tools make special purpose chip design feasible and economical. However, even with the help of recent advances in software tools, the translation of a concept into a chip is not simple: it remains a time-consuming, tedious, and error-prone process, which involves the mastery of all kinds of skills.

A review of existing software tools reveals that the principal components of a design system consist of a powerful layout editor combined with verification/simulation programs. MULGA [14] is an example of a symbolic layout tool, which plays an important role in the proposed framework. It eliminates the need for the designer to know the process design rules thereby increasing his productivity. In brief, cell design is reduced to placing symbols (transistors, wires, and pins). Once a cell has been finished, its function can be simulated and verified by applying proper test signals (data, control and clocks). Finally, cells are put together in a hierarchy to obtain the complete symbolic description of the chip. A mask is generated from the symbolic description and sent to one of the foundries. One crucial question is "**Will the chip work?**". The answer depends on how good the chip is and whether or not it can be tested. Faults and design changes may require repeating the whole design process to obtain a 'working' chip. This iterative manual design process is schematized in Fig. 1.

This manual design process has been proven to be successful for low-to-medium complexity chips,

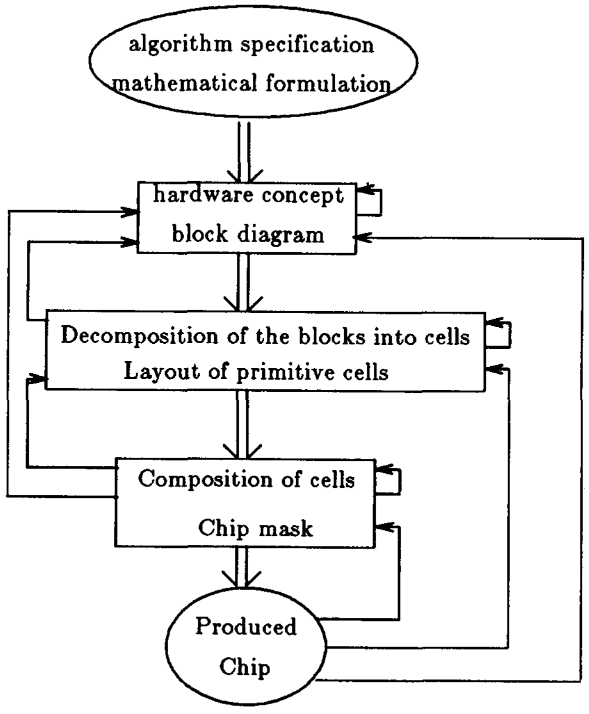


Fig. 1. Chip design process.

but problems occur as designs approach the MIT[1] level. There are two major bottlenecks in current custom chip design: the verification of the design steps (the hardware architecture, the layout and the chip) and the manual layout process (a potential for a large number of human errors). An even more important factor for a succesful chip design is the analysis and adoption of the algorithm for the specific-purpose custom chip implementation. Until now, algorithm analysis, within the context of custom chip implementation, has not had much attention. One reason for this is that, on the one hand, a 'hardware specialist', without detailed knowledge of signal processing techniques, is doing the design given a specific algorithm. On the other hand, signal processing oriented engineers have not had the knowledge necessary for chip design. In the future, it will be beneficial if both hardware and digital signal processing oriented knowledge are merged. This will allow a single person to understand the complete job, from analyzing the algorithm to designing a chip.

Some of these bottlenecks are addressed in [13], where custom chips containing up to MITs need to be produced and fabricated within one month using a team of specialists. These challenges are well recognized, and reflected in the steadily growing literature on silicon compilers [9, 10], CAD tools [7], and VLSI expert systems [1].

One approach to implementing digital signal processing algorithms is to make use of a fixed architecture found, e.g., in a general purpose digital signal processor chip. The performance (in terms of computational power) of this kind of implementation can be improved by generating a specific purpose processor or processors [2, 8]. In both cases, the algorithm is implemented in software. Depending on the algorithm, specialized hardware modules and/or a reduced instruction set can be used. All these implementations suffer from the Von Neumann bottleneck. This bottleneck is not present in a data-flow architecture, where the memory and the control are distributed in the architecture. A special type of data-flow is a systolic architecture [4]. These architectures are characterized by synchrone execution of identical and simple processing elements. This regularity has the advantage that it simplifies the design task. Broadening of the class of systolic architectures can be obtained by allowing all kinds of processing elements in one design. However, the implementation of an algorithm using the most effective processing element considerably increases the complexity of the design task, and forces the use of high level design tools.

This article presents a design methodology for efficiently mapping digital signal processing algorithms into custom chips. In Section 2, the proposed concept, called MOVAL, is described. Section 3 deals in more detail with the abstraction levels and the trade-offs introduced by the concept. Some aspects of analysis, verification, and synthesis are described in Sections 4 and 5. In Section 6, the design methodology is demonstrated with a case study based on the design of a fast Fourier cosine transform (FFCT) chip [11], which has been designed following the lines proposed in this paper.

2. The MOVAL concept

The MOVAL framework is based on six major concepts. First, the human designer must have the possibility to interact with the system at every level, because his insight is essential to obtain high performance chips. Second, the system integrates analysis, layout synthesis and verification. This integration simplifies but also unifies the mapping of algorithms into chips, and is necessary to reduce design time. Third, use is made of the generator (parameterizable super-cell) concept which provides a layout correct by construction. Human layout, which is one of the time-consuming and error-prone design steps, is replaced where possible by automatic generation. Fourth, hierarchical

[1] Instead of the term VLSI, a more precise definition involving the number of integrated transistors is used in this paper. KIT and MIT stand for chips containing Kilo Integrated Transistors and Mega Integrated Transistors, respectively.

abstraction levels are used to separate trade-offs and to simplify the implementation task. Fifth, the same high level language, currently "C", is used for the first four abstraction levels. The advantage is that the correctness of the implementation can be verified using a mixture of abstractions levels. Sixth, the correctness of successive levels is provided by automatic cross-checking. The advantage is that errors are detected in the earlier stages of the implementation.

Based on these concepts, a structured design methodology is provided. It consists of seven abstraction levels, each of which adds more detailed information:

(1) the behavioral (functional description written in the programming language "C"),

(2) the data representation (adding precision and accuracy information implemented by C-modules),

(3) the space/time (adding bit serial/parallel and concurrent/sequential information),

(4) the hardware (adding timing and control information),

(5) the symbolic layout (adding geometric information, using MULGA layout system),

(6) the mask (adding technology information),

(7) the chip.

These levels provide a hierarchy, which is thought to be necessary to handle the design of complex custom chips. Furthermore, it allows efficient separation of the design trade-offs. MOVAL, which is the proposed framework, supports these abstractions levels and is written in a single language. It merges analysis, layout-synthesis and verification. Analysis is an interactive process, the results of which are reflected in the structure of the descriptions at every level. The layout is (semi-) automatically synthesized from the MOVAL hardware description. Verification is applied to each of the levels and is performed automatically. The function of the different domains is described next.

3. Levels of abstraction and analysis

From design specification to chip implementation is a difficult problem, throughout which different kinds of trade-offs must be made and where one has to manage voluminous amounts of complex data. To realize the transition of idea to chip, a partition of the problems can be useful. The proposed division of the problem and subsequently the hierarchy of abstraction levels differs from what is generally applied (from block diagram to logic to switch level). MOVAL is an attempt to realize this by introduction of hierarchical implementation levels. Successive levels contain all the information of the previous levels adding further specifications, obtained by the analysis of a single aspect. The concept is that at every level the designer will refine the idea and analyze a single trade-off. This approach is admittedly suboptimal, because it tries to find an almost optimal solution for one aspect instead of for all the aspects at the same time. However, most of the aspects can be considered as orthogonal and can be optimized independently. Furthermore, none of the existing methods uses a structured way to handle custom chip trade-offs. For a structured approach, it is important to break the design process into the basic elements (abstraction levels) and to analyze the role of each element (trade-offs).

3.1. The behavioral domain

In the behavioral domain, the initial description of the algorithm is found, as well as the refinements to adapt the algorithm for custom chip integration. The major characteristics of the behavioral domain are the following:

(1) The algorithm is written at a high level of abstraction, thus allowing a simple and compact form.

(2) The precision of the data can be regarded as infinite, thus making rounding and truncation errors negligible.

On this level, different ways to express the idea need to be studied with respect to its signal processing characteristics and implementation suitability. Mostly the designer concentrates on the reformulation of the algorithm to tailor it to custom chip implementation. Depending on the application, insight must be gained in the signal processing features of an algorithm, for example, bandwidth, computational aspects, noise immunities and so on. Once an algorithm has been chosen, it is important to reduce the number of operations and the communication costs. A data flow diagram is appropriate for reflecting both aspects.

3.2. *The data representation domain*

At the second abstraction level, one specifies:

(1) the data representations (fixed point unsigned, two's complement, floating point, etc.), and

(2) the number of bits of each data path.

This means that the resolution and accuracy of the implementation will be fixed. The effects of the choices can easily be computed by comparing the results of the data representation description with those obtained with the behavioral description. Results can be shown in error histograms or, when human senses (image, sound) are at the end of the chain, directly presented to a human being.

The data representation description is obtained by replacing the arithmetic operations of the behavioral domain by procedures describing these operations. Floating point variables are replaced by variables with a data structure. The data structure contains the data representation, the number of bits and the actual bit values. Fig. 2 illustrates this for the simple example of a multiplication performed with two's complement coding. The multiplier structure is derived by modifying the Baugh–Wooley algorithm as shown in Appendix A.

A mask function is used to truncate or to round the output variables. This can be required to avoid data length explosion, for example, multiplying an N-bit word with an M-bit word results in an $(N+M)$-bit answer.

3.3. *Space/time domain*

The space/time trade-off is an always reoccurring and important phase of the design. It applies to every operation one wants to implement. One chooses to process the data concurrently or sequentially. Concurrent processing is limited by chip area (for example, if a single chip can hold four 16×16 multipliers, this gives an upper limit on the number of parallel operations). Sequential data processing implies that operators are shared in time. Furthermore operations can be performed bit serial or bit parallel. The choice is partly determined by the throughput specifications. If the specifications allow latency, then pipelining can be appropriate to increase throughput.

Unfortunately, there exists no generally accepted criterion to express chip performance and to optimize the space/time trade-off. Therefore, a new chip performance measure named FAMMS (full adder operation per squared mm second) is proposed. It is a similar type of measure as the ones used to express computer performance (number of FLOPs, floating point operations per second). FAMMS relates the number of operations per time unit to the silicon area needed for it and expresses the computational power of the silicon area. It reflects the trade-off between bit serial and parallel arithmetic as well as concurrent and sequential processing. For digital signal processing it turns out to be a very useful measure, because digital signal processing can be thought of as performed with two basic elements only: operations (full adders) and memory (registers).

3.4. *The hardware domain*

It is at this level that clocks and control signals must be analyzed and specified. The resulting description establishes a one-to-one logical relation with the chip. It allows a (semi-) automatic synthesis of the layout. Furthermore, the hardware description allows the possibility of accessing every node in the circuit. It is used for the generation of vectors to verify the correctness of the symbolic layout, the mask and the chip and couples analysis, layout-synthesis and verification.

4. Layout-synthesis

Efficient fully automatic layout of digital processing algorithms has yet to be achieved. Our layout strategy [5] relies on human assistance to

```
mult2c(x, y, out)

 * Input:           x - multiplier                y - multiplicand
 * Output:          out - result

DATA     *x, *y, *out;
begin
         BIT      b, *sum, *carry, xcarry, xsum;
         int      row, col, xmax, ymax;

         xmax = x->n;                                              * number of x bits
         ymax = y->n;                                              * number of y bits

         row = 0;                                                  * initialization
         for (col=0; col<xmax-1; ++col)
                  and(&sum[col], y->bit[row], x->bit[col]);
         nand(&sum[col], y->bit[row], x->bit[col]);
         out->bit[row] = sum[0];

         row = 1;                                                  * multiplication steps
         for (col=0; col<xmax-1; ++col) {
                  if (xmax != ymax && (col == xmax-2 || col == ymax-2))
                           andha1(x->bit[col], y->bit[row], sum[col+1],
                              &sum[col], &carry[col]);
                  else
                           andha0(x->bit[col], y->bit[row], sum[col+1],
                              &sum[col], &carry[col]);
         }
         nand(&sum[col], y->bit[row], x->bit[col]);
         out->bit[row] = sum[0];
         for (row=2; row<ymax-1; ++row) {
                  for (col=0; col<xmax-1; ++col) {
                           andfa(x->bit[col], y->bit[row], sum[col+1], carry[col],
                              &sum[col], &carry[col]);
                  }
                  nand(&sum[col], y->bit[row], x->bit[col]);
                  out->bit[row] = sum[0];
         }
         for (col=0; col<xmax-1; ++col) {
                  nandfa(x->bit[col], y->bit[row], sum[col+1], carry[col],
                     &sum[col], &carry[col]);
         }
         and(&sum[col], y->bit[row], x->bit[col]);
         out->bit[row] = sum[0];

         if (x->n == y->n)                                         * final ripple carry adder
                  halfadd1(carry[0], sum[1], &xcarry, &xsum);
         else
                  halfadd0(carry[0], sum[1], &xcarry, &xsum);
         out->bit[ymax] = xsum;
         for (col=1; col<xmax-1; ++col) {
                  fa(carry[col], sum[col+1], xcarry, &xcarry, &xsum);
                  out->bit[col+ymax] = xsum;
         }

         out->n = xmax + ymax;
         out->represent = x->represent;
end
```

Fig. 2. Software description for a two's complement parallel multiplication.

improve the performance of automatically generated layout. The assistance of powerful design tools plays an important role in obtaining a satisfactory floorplan, in laying out specific cells or in improving the routing. The approach used for automatic layout is illustrated in Fig. 3.

The fundamental tools are generators and routers. Parameters for generators are defined depending on their function. Specifications for arithmetic operations or mathematical functions include the functional parameters (data representation), the speed requirements (space/time), and the geometry of the silicon area (floorplan). Routers must be capable of routing at the symbolic as well as at the mask level. They obtain a connection list from the hardware description.

5. Verification

The ultimate design goal is to produce a working chip in a timely manner. Therefore, it is important to detect errors as early as possible. To do so, the descriptions at the successive abstraction levels are verified for their functional correctness as soon as they are accomplished by comparing the results of the current abstraction level with a higher level. This comparison is done automatically by using the same input data and checking the results.

There are different methods of generating input test data. Interactive generation by the human, where an interface is used, allows the input of decimal number representation. Other possibilities are the use of waveform generators or real world data. To verify the correctness of the layout, one can use a switch level simulator for the functionality and more accurate simulators for timing. The input of all of these is the same as the input to the first four MOVAL domains. The hardware domain gives us the possibility of finding the correct results for every node. Thus, in theory, all nodes can be checked for correctness. Once the chip is back, the same procedure is valid, as above. The verification approach is illustrated in Fig. 4. Unmatched verification results indicate errors at the abstraction level under test and need to be corrected before proceeding with the implementation.

Another important point is that MOVAL also allows a multi-level mode simulation. An algorithm can be described at different levels. For example, a portion exists only in the behavioral description, another has been laid out, and another part already exists as a chip. The MOVAL concept allows the integral verification of all these parts at the same time by mixing the different levels.

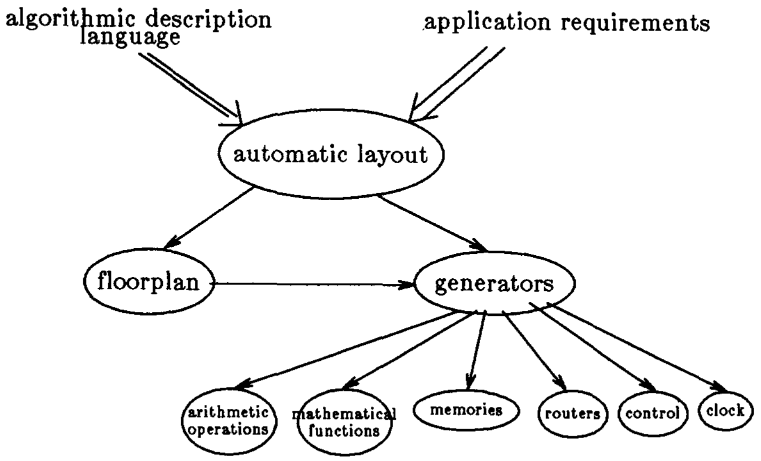


Fig. 3. Schematic of automatic layout tools.

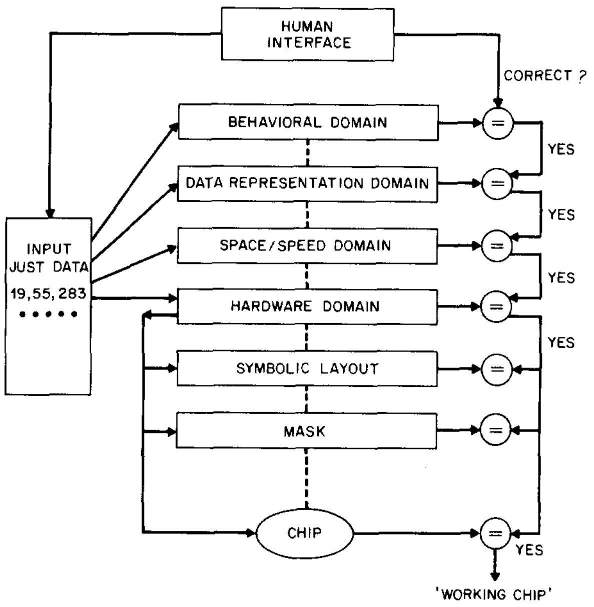


Fig. 4. Verification methodology of MOVAL's abstraction levels.

6. An example

As an example of the presented methodology (MOVAL), the crucial steps in the implementation of an eight-point discrete cosine transform algorithm for image data compression are described.

6.1. Behavioral level

The discrete cosine transform of length N for a real vector $x(0), x(1), \ldots, x(N-1)$ is defined as [3]

$$\mathrm{DCT}(k, N, x) := \sum_{n=0}^{N-1} x(n) \cos\left(\frac{2\pi(2n+1)k}{4N}\right),$$
$$k = 1, 2, \ldots, N-1. \tag{1}$$

For $k=0$ there is an additional scaling constant of $\frac{1}{2}\sqrt{2}$.

Direct implementation of equation (1) requires $O(N^2)$ operations. Therefore, fast algorithms requiring $O(N \log N)$ operations are better suited for implementation. In the VLSI context, for fixed point arithmetic, the area needed for an adder is negligible in comparison to the area occupied by a multiplier. Thus, an algorithm requiring a minimum number of operations was developed [12]. Its software image is written and checked with the DCT definition. To prove the correctness of the implementation, it is sufficient (due to the linearity property) to compare the results for the basis vectors.

Further refinements of the algorithm are made by inspecting the data flow diagram. A rough area estimate shows that some of the operators must be shared. A full concurrent implementation would require twelve multipliers and thirty adders. To share operators, regularity needs to be improved by adapting the data flow, as shown in Fig. 5.

6.2. Data representation trade-off

The next step is to select a proper data representation and fix the data width. In our case, a two's complement representation is chosen for all the variables. This is because the input data is only eight bits (image), which eliminates the need for floating point arithmetic. Furthermore, two's complement coding provides an efficient implementation of arithmetic operations. These choices are translated into a new software image, adapting the behavioral abstraction level into the data representation abstraction level, where explicitly two's complement code and a fixed number of bits are used. This is illustrated in Fig. 6 for a rotation (major part of the algorithm).

The correctness of this new description is compared with the behavioral model. Having the behavioral and data representation description of the algorithm, an analysis can be made of the effects of rounding and truncation.

6.3. Space/time level

Once data representation and width have been fixed, one has to analyze the space/time trade-off. This includes two major choices: first, the number of concurrent processing elements must be determined and, second, one has to decide between bit parallel or bit serial arithmetic. Throughput specifications aiming at least 100 Mbit per second lead to a choice in favor of bit parallel processing

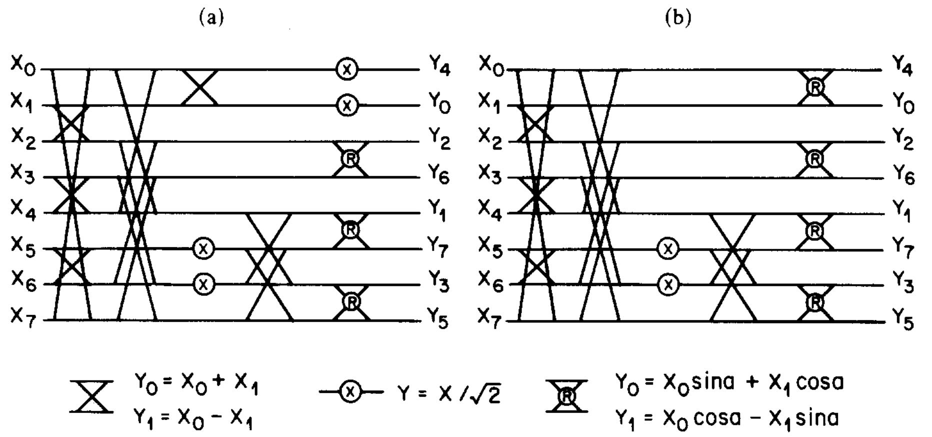


Fig. 5. Data flow diagrams before (a) and after (b) the transformation.

and heavy pipelining. It is obvious by inspection of the data flow diagram that pipelining can be achieved by separating the algorithm into six different stages. Each stage consists of registers, a permutation network and arithmetic operations, as illustrated by Fig. 7.

Given the maximum chip size (36 mm^2) and the size of a bit parallel multiplier (10×12 bits 1.2×0.8 mm^2), it was decided to operate on two samples in parallel. This allows a rotator (the most complex operator implemented with three multiplications and two additions) to fit in the width of the chip. Next, the software image is arranged so that the bit parallel operations are shared (Fig. 8). The correctness is checked against the data-representation description.

```
rotator
in: in1, in2, radians;
out: out1, out2;
begin
        DATA   tsum, m1, m2, m3;
        BIT    ci;
        DATA   c0, c1, c2;

        ci = 0;
        lookup(radians, &c0, &c1, &c2);
        add(in1, in2, ci, &tsum);
        mask(&tsum, &tsum, 1, NBITS);
        mult2c(in1, &c0, &m1);
        mask(&m1, &m1, NBITS, 2*NBITS-1);
        mult2c(&tsum, &c1, &m2);
        mask(&m2, &m2, NBITS-1, 2*NBITS-2);
        mult2c(in2, &c2, &m3);
        mask(&m3, &m3, NBITS, 2*NBITS-1);
        add(&m2, &m3, ci, out1);
        add(&m1, &m2, ci, out2);
        mask(out1, out1, 0, NBITS-1);
        mask(out2, out2, 0, NBITS-1);
end
```

Fig. 6. Description of a rotator at the data-representation level.

6.4. Hardware level

At this stage, the software image is completed with memory elements, buffers, clock and control logic as well as clock and control signals. A graphical representation, and the code for the first stage of the algorithm are shown in Fig. 9.

The created software image specifies every node and is unique. A two phase clocking scheme is used. The maximal clock rate is determined by the path delay in the multiplier, simulated as 150 ns. From the hardware description, the semi (-automatic) layout can be produced. Software procedure-based generators are used for layout synthesis of operators, such as multipliers and adders. A connection list, derived from the hardware description, forms the input for routers. A set of test vectors is generated, specifying each bit at every observable node in the circuit (Fig. 10).

6.5. Layout

The MULGA design system is used to manipulate the symbolic layout as well as the mask. The

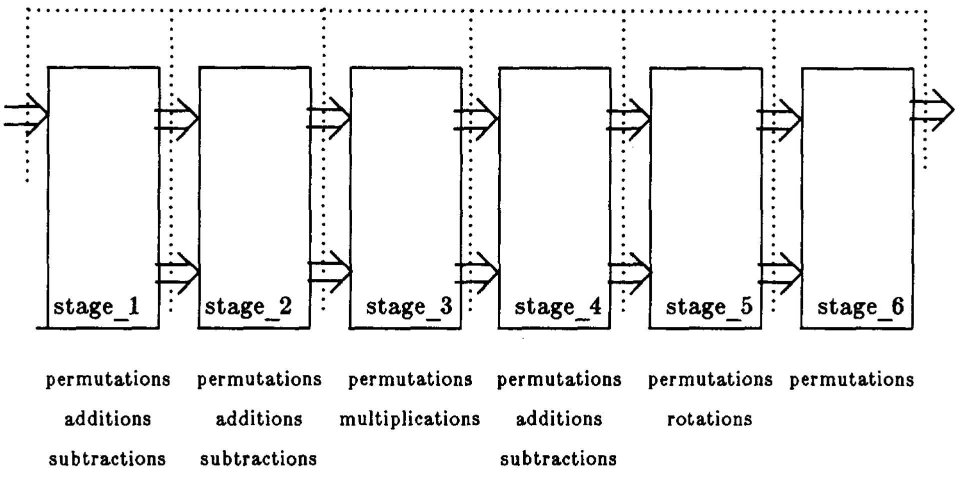
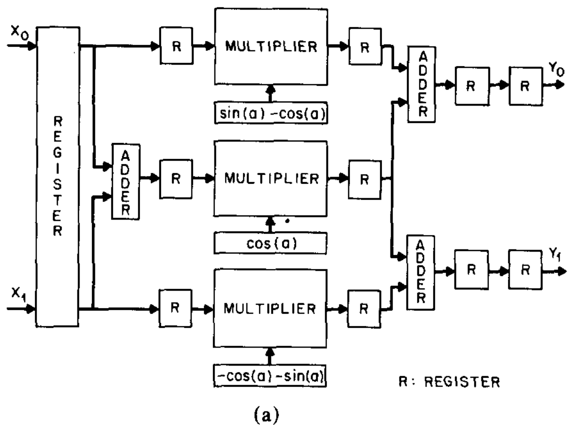

stage_1
stage_2
stage_3
stage_4
stage_5
stage_6
permutations additions subtractions
permutations additions subtractions
permutations multiplications
permutations additions subtractions
permutations rotations
permutations

Fig. 7. Block diagram of the FFCT chip.



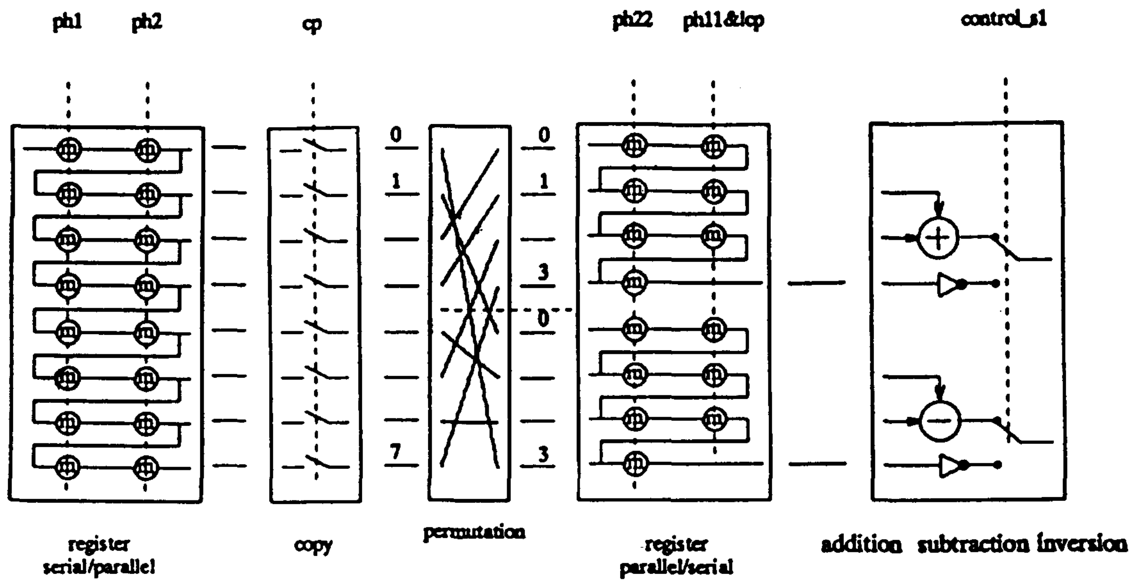
```
rotator
in: in1, in2;
out: radians, out1, out2;
begin
        DATA   tsum, m1, m2, m3;
        BIT    ci;
        DATA   c0, c1, c2;
        DATA   d0, d1, d2;
        int    i;

        ci = 0;
        for (i=0; i<NCYCLES; ++i) {
                rom_d(radians, &c0, &c1, &c2, phi1[i], phi2[i]);
                add(in1, in2, ci, &tsum);
                mask(&tsum, &tsum, 1, NBITS);
                reg_2phase(state1, in1, in1, phi1[i], phi2[i]);
                reg_2phase(state2, &tsum, &tsum, phi1[i], phi2[i]);
                reg_2phase(state3, in2, in2, phi1[i], phi2[i]);
                mult2c(in1, &c0, &m1);
                mask(&m1, &m1, NBITS, 2*NBITS-1);
                mult2c(&tsum, &c1, &m2);
                mask(&m2, &m2, NBITS-1, 2*NBITS-2);
                mult2c(in2, &c2, &m3);
                mask(&m3, &m3, NBITS, 2*NBITS-1);
                reg_2phase(state4, &m1, &m1, phi1[i], phi2[i]);
                reg_2phase(state5, &m2, &m2, phi1[i], phi2[i]);
                reg_2phase(state6, &m3, &m3, phi1[i], phi2[i]);
                add(&m2, &m3, ci, out1);
                add(&m1, &m2, ci, out2);
                mask(out1, out1, 0, NBITS-1);
                mask(out2, out2, 0, NBITS-1);
        }
end
```

(b)

Fig. 8. Block diagram (a) and corresponding software image for the rotator (b).

advantage is that in writing generators one does not need to worry about design rules. Thus, our generators are technology-independent. The synthesis of the layout is based on module generators. According to a specific floor plan, cells from a cell library are tiled to obtain the layout. Cells can be defined in symbolics or in mask. If use is made of a symbolic layout, the MULGA system takes care of converting them to mask. The input parameters of the generators are: area, aspect ratio, maximal delay time, and the data representation requirements (code and number of bits). Fig. 11 illustrates the automatically generated layout of a 10×12 bit parallel multiplier.



```
BIT ph1[RT] = { 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0 };
BIT ph2[RT] = { 0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0 };
BIT cp[RT] = { 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 };
BIT ph11[RT] = { 1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0 };
BIT ph22[RT] = { 0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0 };
BIT p1cp[RT] = { 0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0 };

int     clk;

stage1
in: in
out: out1, out2

begin
        DATA    regp[NSAMP];    /* output conversion from serial in parallel */
        DATA    per1[NSAMP/2], per2[NSAMP/2];
        DATA    s1, s2;


        for(clk=0; clk<RT; ++clk)
        {
                reg_2phase_shift(state1, regp, &in, ph1[clk], ph2[clk], NSAMP);

                permuts1(regp, per1, per2, cp[clk]);

                reg_2phase_ps(state2, &s1, per1, ph22[clk], p1cp[clk], cp[clk]);
                reg_2phase_ps(state3, &s2, per2, ph22[clk], p1cp[clk], cp[clk]);

                add1(&s1, &s2, 0, &out1);
                sub1(&s1, &s2, 1, &out2);
                mask(&out1, &out1, 0, NBITS-1);
                mask(&out2, &out2, 0, NBITS-1);
        }
end
```

Fig. 9. Graphical representation as well as the software code for stage 1.

Example of human generated input vectors (sample1, sample2, degrees)

```
100 100 0
100 100 0.5
100 100 45
100 100 90
100 100 180
```

Automatic generated test vectors for a functional simulator

```
rotator
5
5 phi1 1 0 phi2 1 0 in1 10 2 in2 10 2 alpha 10 2
2 out1 10 2 out2 10 2
0 1 0010011000 0100110000 0000000000 0000000000 0000000000
0 0 0010011000 0100110000 0000000000 0000000000 0000000000
1 0 0010011000 0100110000 0000000000 0000000000 0000000000
0 0 0010011000 0100110000 0000000000 0000000000 0000000000
0 1 0010011000 0100110000 0000000000 0001100000 0011000000
0 1 0010011000 0010011000 0000000100 1001100000 1001100000
0 0 0010011000 0010011000 0000000100 1001100000 1001100000
1 0 0010011000 0010011000 0000000100 1001100000 1001100000
0 0 0010011000 0010011000 0000000100 1001100000 1001100000
0 1 0010011000 0010011000 0000000100 1111111111 1100010000
0 1 0010011000 0010011000 0000000001 1111111111 1100010000
0 0 0010011000 0010011000 0000000001 1111111111 1100010000
1 0 0010011000 0010011000 0000000001 1111111111 1100010000
0 0 0010011000 0010011000 0000000001 1111111111 1100010000
```

Fig. 10. Set of automatically generated test vectors for the rotator.

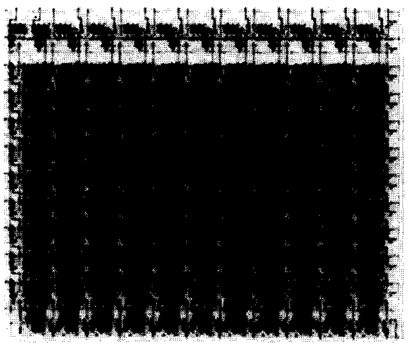


Fig. 11. Example of an automatically generated multiplier module.

Once the complete layout has been created, one verifies it automatically using the test vectors generated with the hardware description. Multi-level verification is achieved by substituting a software module with a model obtained from the layout. However, errors are quite unlikely, if the layout is automatically created with generators.

7. Conclusions

A discipline, MOVAL, for dedicated digital signal processing custom chip design is proposed. MOVAL integrates analysis, (semi-) automatic layout synthesis and automatic verification for digital signal processing designs. This is achieved by the definition of seven abstraction levels: the behavioral, the data representation, the space/time, the hardware, symbolic layout, the mask and the chip description. The unambiguous hardware domain description allows a (semi-) automatic layout, based on generators. Beside design time reduction, the advantage is that automatic layout is correct by construction. The verification is done fully automatically in a sequential fashion, starting at the behavioral level and going all the way down to the chip. All the levels are written in the same language (C), which has the advantage that it allows multi-level mixed mode testing. Different from other methods, MOVAL emphasises on analysis during the design phase to produce high-performance chips.

In the present authors' opinion, the only way to handle 'complex' (KIT or MIT) custom chips is by a structured method such as MOVAL. It provides the designer with a top-down design discipline and allows him or her to handle efficiently the trade-offs required for high performance custom chips.

Only a small part of MOVAL is constructed yet. Much work remains to be done to bring MOVAL to maturity.

Acknowledgment

We thank Bill Ninke for his continuous support and Path Hayes whose comments on an earlier draft have improved this paper.

Appendix A

For the CMOS implementation of a two's complement multiplier, the derivation of a regular structure, which minimizes communication is important. Such a structure can be achieved by reordering and merging different terms. Let X and Y be the values of the multiplicand and the multiplier respectively; then the value of the product is

$$XY = x_{n-1}y_{m-1}2^{m+n-2} + \sum_{i=0}^{i=n-2}\sum_{j=0}^{j=m-2} x_i y_j 2^{i+j} - \left(\sum_{i=0}^{i=m-2} x_{n-1}y_i 2^{n-1+i} + \sum_{j=0}^{j=n-2} y_{m-1}x_j 2^{m-1+j} \right). \quad \text{(A.1)}$$

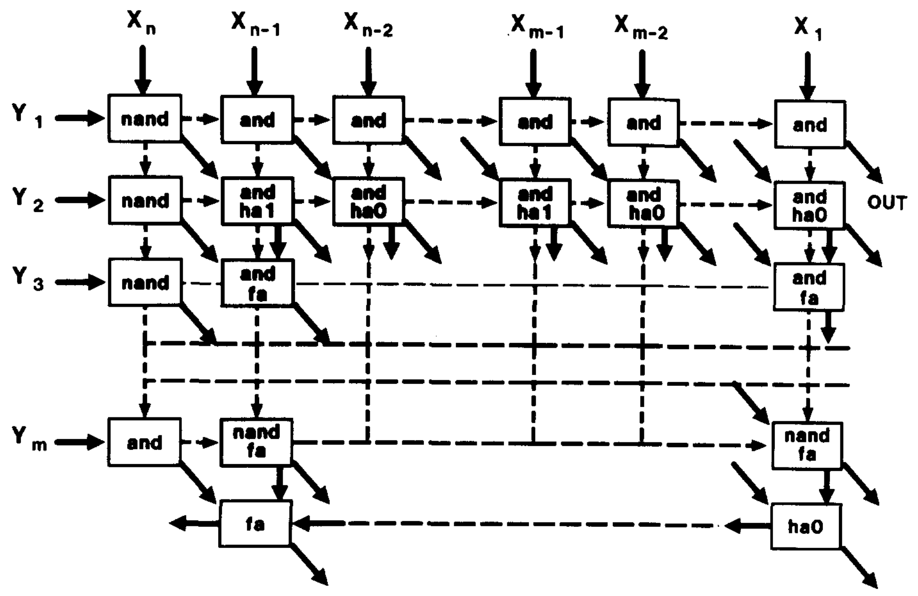


Fig. A.1. Block diagram of the multiplier structure.

Replacement of the second term by its negation yields

$$XY = x_{n-1}y_{m-1}2^{m+n-2} + \sum_{i=0}^{i=n-2}\sum_{j=0}^{j=m-2} x_i y_j 2^{i+j}$$

$$+\left(\sum_{i=0}^{i=m-2}\overline{x_{n-1}y_i}\,2^{n-1+i} + \sum_{j=0}^{j=n-2}\overline{y_{m-1}x_j}\,2^{m-1+j} + 2^{m-2} + 2^{n-1} + 2^{(n-1)m}\right). \tag{A.2}$$

The product is obtained with summations only. Furthermore, one's have to be added to the 2^{n-2}, 2^{m-1}, and $2^{(n-1)m}$ positions. Implementation is achieved with and, nand, full adder (fa) and halfadder (ha) gates. Two types of halfadder gates can be distinguished, with a carry set to one (ha1) and with the carry set to zero (ha0). The use of these half-adders permits the incorporation of the addition of one's into the structure. The structure of the resulted multiplier is illustrated in Fig. A.1.

References

[1] B. Ackland et al., "CADRE—a system of cooperating VLSI design experts", submitted to *IEEE ICCD*, 1985.

[2] M.R. Buric, P. Christensen and T.G. Matheson, "Plex: Automatically generated microprocessor layouts", *IEEE Design and Test*, Vol. 1, August 1984, pp. 52-65.

[3] W.H. Chen et al., "A fast computational algorithm for the discrete cosine transform", *IEEE Trans. Commun.*, Vol. COM-25, September 1977, pp. 1004-1009.

[4] H.T. Kung, "Why systolic architectures?", *IEEE Comput.*, Vol. 15, No. 1, January 1982, pp. 37-46.

[5] A. Ligtenberg and J.H. O'Neill, "Towards an automatic layout for digital signal processing algorithms", to be published.

[6] B. Liu, "Effect of finite word length on the accuracy of digital filters—a review", *IEEE Trans. Circuit Theory*, Vol. CT-18, November 1971, pp. 670-677.

[7] J.K. Ousterhout et al., "The magic VLSI layout system", *IEEE Design and Test*, Vol. 2, February 1985, pp. 19-29.

[8] J.M. Rabaey, S.P. Pope and R.W. Brodersen, "An integrated automated layout generation system for digital signal processing circuits", *Proc. Custom Integrated Circuits Conf.*, May 1985.

[9] "Silicon compilers", *VLSI Design*, September 1984, pp. 44-69.

[10] J.M. Siskind, J.R. Southard and K.W. Crouch, "Generating custom high performance VLSI designs from succinct algorithmic descriptions", *MIT Conf. on Advanced Research in VLSI*, Cambridge, MA, January 1982.

[11] M. Vetterli and A. Ligtenberg, "A discrete Fourier-cosine transform chip", *IEEE J. Selected Areas in Commun.*, Vol. SAC-4, No. 1, January 1986, pp. 49-61.

[12] M. Vetterli and H.J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations", *Signal Processing*, Vol. 6, No. 4, August 1984, pp. 267-278.

[13] P. Wallich, "The one-month chip: Design", *IEEE Spectrum*, Vol. 21, September 1984, pp. 30-34.

[14] N. Weste and B. Ackland, "A pragmatic approach to topological symbolic IC design", *Proc. 1st Internat. Conf. on VLSI*, Edinburgh, August 1981, pp. 117-129.