OPTIMIZING AREA ON THE GENERATION OF SPECIFIC CIRCUITS IN FPGAS FOR SIMD APPLICATIONS

Germán León, José M. Claver, Germán Fabregat
Dept. of Computer Science and Engineering, University Jaume I
Campus Riu Sec, 12071– Castellón, Spain
E-mail: leon,claver,fabregat@icc.uji.es

ABSTRACT
We describe the current state of development of an environment for the automatic generation of specific circuits in FPGA for SIMD applications from high level specifications. The methodology used is based on the transformation of the whole algorithm in a graph of LUTs (Look Up Tables), without the use of library components, that implements all the required operations. The quality of the obtained circuitry is guaranteed by the use of "type inference", a technique that is focus of current research on the design high level circuit generators, which is context dependent. Two sort of implementations of the iterative expressions are possible with these tools: with or without a complete unrolling. As a first case study, we have tested the efficiency of this environment in the implementation of an integer discrete wavelet transform (DWT).

KEYWORDS
High-level Description Languages, Configurable Computing, FPGA, Hardware Compilation, SIMD applications.

1. INTRODUCTION

Currently there is a rising interest in the design of heterogeneous architectures for applications that need intensive computations. These designs increase their performance by using, either partially or totally, generic hardware resources. Among this set of architectures, specially important are those focused on the transformation of continuous dataflows, i.e. multimedia applications, and embedded applications that work into quickly changing environments. One of these tendencies, so called reconfigurable architectures (RA), has been powered by the technological advances in high scale integration and the availability of hardware resources [Villasenor-97]. The architecture of these platforms [Kozyrakis-98] , and their programming [Lee-00], is object of many studies which try to find more efficient configurations, often composed by a set of processors, configurable circuitry, either for random logic or for accelerating intensive computation, networks of configurable elements and different kinds of memory.

Methods used to develop applications for reconfigurable architectures are dominated by the use of industrial CAD/EDA tools, which are generalized in the VLSI design field. Each new device launched to market has a set of libraries and tools for its planning, simulation, placement and routing usually offered by the manufacturer. These tools use as inputs hardware description languages (HDL) and other methods of low level specification of logic circuits. There are some recent initiatives in the specification of public architectures (Xilinx 6200) and open environments for the design of circuits (Xilinx Jbits API, used to configure the Virtex II). However, many important problems are still unsolved: portability, reusability and open access to architecture resources, that do not allow a more general use of RA, and the application, in this field, of the techniques commonly used in software engineering.

Multimedia applications are known examples of SIMD applications, characterized by fine grain operations and high dependency on the dataflow from memory. In order to increase the performance of this sort of applications, the use of RA is an alternative method to the SIMD extensions of current general-purpose superscalar processors (characterized by the flexibility of their use) and to the ASICs integrated in DSPs (less flexible and more specific, but more performance efficient). The use of RA can extract the best characteristics of these two approaches, allowing an easy way to update the system functionality and performance.

In multimedia applications, discrete wavelet transform (DWT) is widely used on storage systems and images and video transmission. Specially, there is a rising interest in reversible DWT (lossless) based on integer arithmetic. Since DWT is a paradigm of current SIMD applications, and due to their polynomic characteristics and data dependencies, we have chosen it as the first case study for our work.

The rest of this paper is organized as follows. Next section addresses code generation problems for reconfigurable architectures, showing some programming considerations. In section 3 solutions adopted in previous works are revised. In section 4 the execution model and compiling process on the reconfigurable logic used in our approach are described. and first implementation results are shown in section 5. Finally, section 6 discusses the results and presents future work.

2. GENERATING CIRCUITS ON RAs

Reconfigurable architectures can combine hardware resources with different granularity: buses, wires, swit-ches, memories, processors, etc. Fine grain components on RA are constituted by flip-flops and small tables of memory (LUT) addressed by a set of signals. LUTs have often a few bits (between 4 and 8), and constitute the basic resources for combinational logic generation. From the logic synthesis point of view, a LUT of n bits is the more general way to generate any logic function of n boolean variables. There are enough well-known algorithms that allow to adapt any logic table to a particular architecture with a given LUT size. Thus, a simple polynomic expression as

$$z = 3x + 4(y + 2t) \tag{1}$$

can be implemented generating an adequate LUT for each basic operator or using its associated library component, which will transform input data on output results. The set of generated operators are connected by the intercommunication resources following the data dependencies of expression (1). Fig. 1.a shows a possible representation of the network of LUT operators that performs this implementation, where the bubbles above the figure are the input data.

An optimized implementation of this net of LUTs can make use of non standard operators, i.e., operators $4 * B$, $A + B$, and $2 * B$ can be implemented on a single LUT that works with two input variables $(4 * (y + 2t) = 4 * (A + 2B) = 4 * B|(A + 2B) = (4 * B|A + B|2 * B)$, where A, B are operator inputs and " | " denotes the operator fusion (see Fig. 1.b). This technique, known as *operator fusion*, removes unary operators by fusing them with their producer/consumer operators. That is due to the fact that LUTs give the necessary flexibility to generate a combinational circuit more adequate for each problem.

In order to design this circuit, it remains necessary to know the type of input data, which is directly related to the complexity of LUTs and their interconnections, and therefore to the complexity of the resulting implementation. Fig. 1 also shows the input data range of each LUT operator in the decomposition of LUTs for this circuit, if initial input data (x, y, z) are 2-bit positive binary numbers. If this information is taken into account, when an implementation of this circuit is performed, the total amount of gates used to build it can be optimized. Thus, if library components are used to implement operators, they must be chosen to accordingly to the bitwidth of input data range [Constantinides-03, Stephenson-00]. Otherwise, this information can be used to optimize the synthesis of non standard operators in some specific problems [Lagadec-02].

Many current systems that automatically generate logic circuits for RA from high level specifications use functions in programming languages like C, C++, or Java. These languages need an *a priori* declaration of the type and bitwidth of variables. These types are related to scalar types that can be processed by arithmetic units of current processors. However, the bitwidth of these data types is either not adequately adapted for their implementation on RA or overdimensioned. Therefore, resulting circuits are inefficient in resource allocation and performance due to the semantic gap between the specification language used and the hardware resources.

Another problem is provided by the sequential execution model of the traditional programming languages, which is completely different to the execution model of RA and the execution model of hardware in general. A well known outcome of this problem is the inefficient use of SIMD extensions of current processors. These extensions are only efficiently used in some specific problems. A pure object oriented language like SmallTalk provide many advantages when used as starting point for the logic generation process on RA. SmallTalk uses a delayed type binding until messages, which are equivalent to the operators in the previous example, are sent to an object previously instantiated. Furthermore, SmallTalk is an

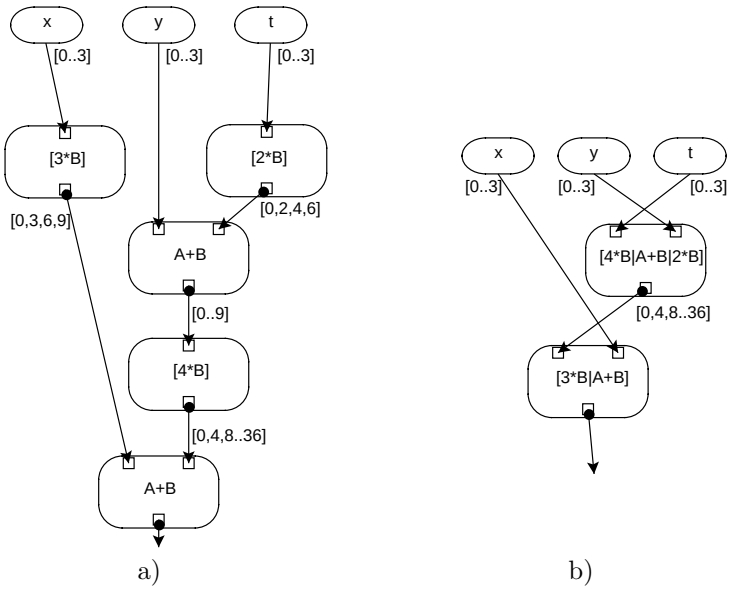

Figure 1. Decomposition on a network of LUTs of $z = 3x + 4(y + 2t)$: a) standard, b) with operator fusion. We show, into brakets, the input data set for each operator.

interpreted language running on a virtual machine. So the high level description of operations is maintained in a symbolic form until execution time. These characteristics allow a delayed assignment of data types adequate for optimal generation of operators. SmallTalk also provides, in a similar way, compiling from high level specifications on an heterogeneous embedded system, as described in [Fabregat-99]. Thus, the programming and simulation of applications and their compiling on an heterogeneous architecture can be integrated in the same environment without any additional development cost.

3. PREVIOUS WORK

There are two different ways of describing hardware, structurally and behaviorally. In the structural description, the designer indicates what components should be used and how they should be connected. This designing method can be rather tedious and time consuming. Behavioral descriptions allow to design a circuit by describing their behavior at a higher level of abstraction which can be automatically compiled down to structural hardware. The popular industrial description languages VHDL and Verilog allow both sorts of descriptions. However, these languages are very specific and far form traditional high-level programming languages used by software developers, programmers and, in particular, embedded sytem engineers.

There are several recent tools that allow high level hardware description for languages equal or similar to the traditional programming languages. Thus, languages like HandelC [Sullivan-02], SystemC [Swan-01] and Ocapi [Vanmeerbeeck-01] are derived from languages C and C++. Forge is a Java compiler tool developed by Xilinx that allows automatic loop unrolling and pipelining. These compilers have some common characteristics in order to reduce the semantic gap between the original programming execution model and the hardware execution model. Thus, SystemC and Ocapi are C++ class libraries, allowing their use as modeler and Co-design tool. Moreover, as object oriented languages, these use combined behavioral and structural ways to describe hardware. In both systems, synchronization process is provided by specific C++ classes and parallelism between operations is automatically extracted. Thus, HandelC is a behavioral C based hardware description system developed by Celoxica that allows Co-simulation. Parallelism of process and synchronization are taken from the CSP model, in particular, from the Occam language, already studied for this purpose several years ago [Page-91] . Since these compilers are derived from strongly typed languages, data types must be declared at beginning. SystemC, Ocapi and Forge use some of the standard data types with standard bitwidths (char = 8 bit, integer = 32 bit, etc.), while HandelC uses standard data types with user defined bitwidths. So, HandelC provides a more efficient use of hardware resources. In both cases, data types must be established by programmers

that will take into account the bitwidth of intermediate values needed to avoid overflow and underflow situations in the design process.

In this sense, recently published work describes methods to automaticaly adjust the bitwidth of internal variables in the generation of digital circuits from high level specifications or in order to translate float-point formulations into fixed-point formulations, specially for specific DSP applications (see [Constantinides-20003] and therein). These approaches use fixed-point data types and are specified from C/C++ like languages. Some of these approaches use analytic techniques to scaling and/or error stimation [Constantinides-2003, Stephenson-2000], some use simulation [Cantin-2002], and other use a hybrid of the two. The analytic approach is the most intersting since it does not require representative simulation stimulus, and can be faster, but it tends to be more pesimistic. These methodologies are based on type inference techniques and proposes propagation of integer/fixed-point variable ranges forward and/or backward through dataflow graph. All these approach use library components in order to generate the digital cicuitry.

Recently, there are some attempts to use embedded languages [Hudak-96] to design hardware description languages. An embedded description language is realized by means of a library in an already existing programming language (the *host language*). Thus, Lava [Claessen-02] is an structural hardware description language embedded in the funtional programming language Haskell. Another example of this approach, called MADEO, is described in [Lagadec-02]. In this system, a behavioral hardware description dialect is embedded in the object oriented programming language SmallTalk. The environment, designed for the generation of combinatorial circuits, analyses the dataflow graph of the expressions to automatically extract and exploit the existent parallelism among operations. This framework does not use libraries to generate the final hardware configurations, but generates the required operators using *type inference*. As a result, while the compiling process is quite time consuming, the size and performance of the obtained circuit is near optimal when the input data set allows simpler operators implementation than with standard library components. Moreover, the bitwidth of the inner operators is automatically tuned to avoid overflows due to the forward propagation of integer variables though the dataflow graph.

Nowadays, SmallTalk is not a programming language common to the system developpers and embedded system engineers, but these ideas can be moved to other object oriented languages as Java and C++. The tool we present in the following section extends this framework adding new data types and control flow constructs for SIMD applications.

4. EXTENDING *MADEO* TO SIMD APPLICATIONS

In this section, the compiler used to generate logic from an algorithmic specification in SmallTalk is briefly presented in subsection 4.1. In subsection 4.2, a more detailed description of the compiling process is shown. We focus mainly on the handling of iterative control structures.

4.1 Execution Model

The execution model followed by our compiler is adapted to the implementation of operations on LUTs, as it is shown in section 2. After a dataflow analysis of expressions, a Hierarchical dataflow Graph (HDFG) is generated, where nodes are operations that will be implemented on LUTs, and arcs are the data dependencies. At each level of the hierarchy, nodes are evaluated over the set of inputs, in order to obtain the set of possible outputs. This evaluation is carried out for all nodes, taking into account possible data feedbacks. The process finalizes giving a tabular description of its behavior for each node, which directly allows its implementation on LUTs.

This method presents total independence respect to input and output data types for each operator, and only actual values (set of input/output values for each LUT) that will participate in the design of operations are considered. Thus, input values for any LUT can be coded in order to obtain an adequate use of physical resources (logic circuitry, communication buses between LUTs, etc). The evaluation of nodes and their new coding of input/output values is carried out in a simple way by using a weakly typed language like SmallTalk.

4.2 Compiling and Logic Synthesis

As we have described before, the compiler generates an HDFG, whose nodes are operations to be done and arcs are the data dependencies between them. In a second step, this graph suffers a first set of

transformations in order to propagate constants, remove common expressions and release non accessible code. In this stage, with well defined HDFG inputs, a type assignment process is carried out for all input values. This process is followed by the evaluation of the input data set and the type inference for output and intermediate arcs. In order to complete this process, truth tables for all graph nodes are generated. These tables are sent to a set of standard tools for optimization, partition and adaptation to the characteristics of an specific RA.

The described method can be immediately applied to expressions of any complexity, but requires different considerations when it is applied to iterative algorithmic structures, i.e. treatment of vectors. Loop unrolling is the basic problem, but it will be implemented on an RA with limited resources. Since the number of iterations executed in parallel is bounded to the available resources, and complete unrolling cannot be always obtained, a mechanism to allow iterative execution must be provided.

```
a) new:=0.
   (1 to:2) collect:[:i|new:=new+((A at:i)*(B at:i)).].
b) new:=0.
   (1 to:2) do:[:i|new:=new+((A at:i)*(B at:i)).].
```

Figure 2. Messages **collect:**(a) and **do:**(b) applied to the scalar product.

With this aim we have developed two constructs, implemented as SmallTalk messages, that allow to consider both cases:

- **collect:** when this message is sent over a range with a code block as argument (see Fig. 2.a), it generates the unrolling of the block for the entire range (see Fig. 3.a). Once the loop is unrolled, the previously described analysis and processing of the generated code can be performed.
- **do:** when this message is sent over a range with a code block as argument (see Fig. 2.b) it generates the iterative execution of the operations specified in the block as many times as the range imposes. The result is a synchronous sequential circuit, which is generated taking into account values passed among different iterations (see Fig. 3.b). Thus the set of input values required by the compiling process described previously is completed, and intermediate registers and multiplexors are added as required to propagate values for future iterations.

5. EXPERIMENTAL RESULTS

Nowadays the wavelet transform (WT) is widely used in the treatment and compression of signals, images and video. However, until recently, its use has been limited to lossy applications. This is due to the fact that WT produce floating point coefficients which are not well-suited for lossless coding applications. The introduction of reversible wavelet transforms that transform integers to integers, allow perfect reconstruction of the original image. Thus there is an increasing interest in using integer WT (IWT) for lossless image coding [Dewite-97]. This sort of algorithms is a good example of algorithms in which the input and output data sets have bounded ranges. In this paper, only the Said and Pearlman transform (S+P) [Said-96] is considered. This is one of the most well-known IWT due to its good performance and use in the SPIHT compression algorithm. Fig. 4 shows this transform expressed in the lifting scheme, where x is a integer input vector, d and s are the resulting high and low pass output vectors (see [Said-96] for more datails).

The integer S+P wavelet transform has been programmed in SmallTalk and has been treated following the proposed method for its implementation on a RA. Fig. 5. shows the SmallTalk code obtained for the S+P wavelet transform, applied to a vector of four elements, using the message **collect:/do:** (we denote as **spcint/spint**). The codes obtained can be either executed and evaluated on the SmallTalk environment or transformed on a hardware description and implemented on a RA.

In order to evaluate the results of this approach, we have used the Celoxica RC1000 PCI based FPGA board . The RC1000 is a PCI bus plug-in card for PC. It has one large Xilinx FPGA (in our case a Virtex 2000E) with four banks of memory for data processing operations and two PCI Mezzanine

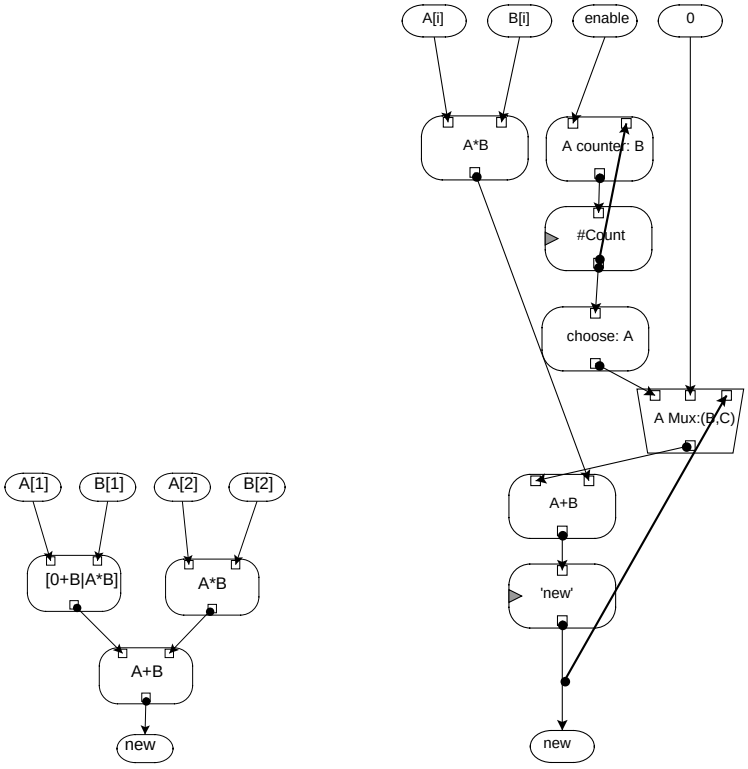

Figure 3. Dataflow graph when the message **collect:** is applied to the expression in Fig. 2.a and when the message **do:** is applied to the expression in Fig. 2.b.

$$
\begin{array}{|l|}
\hline
d^{(1)}[n] = x[2n+1] - x[2n] \\
s[n] = x[2n] + \lfloor \frac{d^{(1)}[n]}{2} \rfloor \\
d[n] = d^{(1)}[n] + \lfloor \frac{2}{8}(s[n-1] - s[n]) + \\
\frac{3}{8}(s[n] - s[n+1]) + \frac{2}{8}d^{(1)}[n+1] + \frac{1}{2} \rfloor \\
\hline
\end{array}
$$

Figure 4. The S+P wavelet transform.

Cards. The FPGA Virtex 2000E is based on slices that contain two 4-bit LUTs each one. Therefore, the LUTs generated in a first step are simplified and adjusted to this granularity. To do this, the circuit optimization tool SIS and the command line tool called Xflow, that allows to automate the Xilinx implementation, simulation, and synthesis flows, are used. Finally, a programmable specification (EDIF or Bitstream) is obtained. In this process, we have generated different implementations on the FPGA to show the advantages obtained using a type inference based optimization. In all cases, the final circuits are synthesized to optimize their size and the type of input data used on implementations are integers from 4 to 7 bitwidths. Due to the very time consuming synthesis process, we have not used bitwidths larger than 7 in our implementations.

For both algorithms, *spcint* and *spint*, two sort of optimizations are used to obtain the synthesized circuit. In the first version (MADEO (OF)), operator fusion is applied to LUT description of the algorithm before the optimization of SIS, and type inference technique is not used. In the second one (MADEO (TI)), operator fusion is avoided, and type inference techniques are applied. In order to compare our methodology with another current hardware description language, we have implemented a HandelC version of *spcint* and *spint* algorithms. Each description of these algorithms has been compiled to obtain its VHDL representation and the bitstream has been generated by using the Xflow tool.

Table 1 shows that optimized areas obtained with our tools are in general better. In particuar, type inference versions of *spint* and *spcint* algorithms from 4 to 7 bits have reduction in area size respet to the HandelC version. Note that using type inference is more efficient than operator fusion respect to the

```
spcint/spint
| ddh ddh2d3 s_low_d0 sp_diff_d0 s_high_d0
v_high_d1 vsp_diff_d0 v_low_d1|
      v_low_d1:=0.  v_high_d1:=0.  vsp_diff_d0:=0.
      (1 to: 4 by: 2) collect/do: [:i |
            s_low_d0:=((X at: i)+(X at: (i+1)))/2.
            s_high_d0:=(X at: i)-(X at: (i+1)).
            sp_diff_d0:=v_low_d1-s_low_d0.
            ddh:=(vsp_diff_d0+sp_diff_d0)-s_high_d0.
            ddh2d3:=((ddh*2)+sp_diff_d0)+3.
            high at: (i quo: 2) put:
            (v_high_d1-(ddh2d3/8)).
            low at: (i quo: 2) put: v_low_d1.
            v_low_d1:=s_low_d0.
            v_high_d1:=s_high_d0.
            vsp_diff_d0:=sp_diff_d0. ].
```

Figure 5. S+P WT code in SmallTalk using the message **collect/do:** denoted as **spcint/spint**.

Table 1. Results of area (in LUTs) obtained, using signed integers from 4 to 7 bits, for HandelC and our implementation with operator fusion (OF) and type inference (TI).

	spint				**spcint**			
Version	4 bits	5 bits	6 bits	7 bits	4 bits	5 bits	6 bits	7 bits
Handel-C	105	124	143	161	203	234	268	300
MADEO (OF)	106	116	141	184	148	167	213	268
MADEO (TI)	77	106	139	160	113	150	200	269

number of LUTs needed to implement these circuits, but when the bitwidth is increased the difference between the two implementations is reduced. The obtained results shown that area reductions are better for the type inference *spcint* circuits. That is because *spint* is a sequential fed-back circuit and the input set is increased with the results of previous iterations. So, type inference can obtain less optimization due to an increasing of infered input data set of operator LUTs and the resulting reduction of *"don't care"* entries in their definition.

Type inference is generally better than operator fusion for all data bitwidths. This behavior is due to the fact that operator fusion technique reduces the number of operator LUTs of the circuit, but the resultant operator LUTs are more complex and difficult to simplify (SIS spent less time to synthesize the circuit when type inference is applied). Otherwise, type inference maintain the number of operator LUTs, but reduce their complexity by introducing *"don't care"* entries. Thus, in general, the SIS tool (used to optimize the circuit in the final step) can obtain a more simplified circuit when type inference is used. But the improvement of this optimization also depends on the input data bitwidth.

Table 2 shows similar delay results for the HandelC and our *type inference* implementations. However, the HandelC version have better delays when the data bitwidth is increased. That is due to the fact that the FPGA Virtex 2000E architecture in not completly regular and have fast lines to propagate the *carry*

Table 2. Results of Delay (in ns) obtained for the *spcint* circuit, using signed integers from 4 to 7 bits, for HandelC and our implementation with operator fusion (OF) and type inference (TI).

	spcint			
Version	4 bits	5 bits	6 bits	7 bits
Handel-C	49.16	49.28	50.05	52.13
MADEO (OF)	52.56	55.23	59.69	62.23
MADEO (TI)	44.46	46.69	52.21	56.18

bit when standard adder operators are used. These fast *carry* lines can not be used on our non standard operator LUTs. We will try to solve this problem in a near future by using more advanced optimizing techniques.

6. CONCLUSIONS AND FUTURE WORK

In this paper, new possibilities of a development environment to generate hardware on a RA from high level specifications are described. The use of SmallTalk is essential due to their characteristics that allow to delay the type assignment, and to evaluate the input and output data sets during the compiling process. This work has been focused on the development of an efficient method to manage loops and other iterative structures appropriately. The first results obtained in the management of loops on a case study, the S+P integer wavelet transform, have been described. In these results, has been possible to see that type inference is a powerful technique that allows reduction of area on a SIMD algorithm as the S+P wavelet transform without significant impact on the delay. Currently, we are developing new high level specifications that combines the benefits of **collect:** and **do:** and extending our study to other SIMD algorithms. This message will allow to unroll a loop a given number of times and iterate as required to complete all the loop iterations. An interesting aspect we are studying is the automatization of this process for an specific architecture.

ACKNOWLEDEMENT

This work is founded by the Spanish CICYT project under grant No. TIC2003-08154-C06-06 and the EUROPEAN FEDER programme.

REFERENCES

Cantin, M., *et al.*, 2002. A comparison of automatic word length optimization procedures. *IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, Vol. 2, pp. 612-615

Claessen, K., Pace, G., 2002. An embedded language framework for hardware compilation. *4^{th} International Workshop on Design Connect Circuit (DCC'02)*, Grenoble, France.

Constantinides, G., *et al.*, 2003. Word-length Optimization for Linear Digital Signal Processing. *IEEE Transactions on CAD of Integreted Circuits ans Systems*, Vol. 22, No. 10, pp. 1432-1442.

Dewitte, S., Cornelis, J., 1997. Lossless integer wavelet transform. *IEEE Transactions on Image Processing*, Vol. 4., pp. 158-160.

Fabregat, G., *et al.*, 1999. Embedded system modeling and synthesis in OO environments: a smart-sensor case study. *Compiler and Architecture Support for Embedded Systems (CASES'99)*, Washington D.C., USA.

Hudak, P., 1996. Building domain-specific embeddd languages. *ACM Computing Surveys*, Vol. 28, No. 4, pp. 196-200.

Kozyrakis,C., Patterson, D., 1998. A new direction for computer architecture research. *IEEE Computer*, Vol. 33, No. 11, pp. 24-32.

Lagadec, L., Pottier, B., 2002. A LUT based high level synthesis framework for reconfigurable architectures. *11^{th} IEEE/ACM International Workshop on Logic & Synthesis*, pp. 167-172.

Lee, E., 2000. What's ahead for embedded software. *IEEE Computer*, Vol. 33, No. 9, pp. 18-26.

Page, I., Luk, W., 1991. Compiling Occam into field-programmable gate arrays. W. Luk and W. Moore, editors. *FPGAs*, Abingdon EE&CS books, pp. 271-283.

Said, A., Pearlman, W., 1996. An image multiresolution representation for lossless and lossy compression. *IEEE Transactions on Image Processing*, Vol. 5, pp. 1303-1310.

Stephenson, M., *et al.*, 2000. Bitwidth analysis with application to Silicon Compilation. *Proceedings of the ACM, SIGPLAN PLSI*, Vancouver, Australia, pp. 108-120.

Sullivan, C., Chapell, S., 2002. Handel-C for co-processing an co-design of field programmable systems on chip. *JCRA 2002*, pp. 65-70.

Swan, S., 2001. An introduction to system level modeling in SystemC 2.0. Available from `http://www.systemc.org`.

Vanmeerbeeck, G., *et al.*, 2001. Hardware/software partitioning of embedded systems in OCAPI-xl. *9^{th} International Symposium Hw/Sw Codesign (CODES 2001)*, pp. 30-35.

Villasenor, J., Mangione-Smith, W., 1997. Configurable computing. *Scientific American*, pp. 66-71.