2011 14th Euromicro Conference on Digital System Design

Hardware Reuse in Modern Application-specific Processors and Accelerators

Alexandre S. Nery*‡, Lech Jozwiak‡, Menno Lindwer§, Mauro Cocco§, Nadia Nedjah¶, Felipe M.G. França*
*LAM – Computer Architecture and Microelectronics Laboratory
Systems Engineering and Computer Science Program, COPPE
Universidade Federal do Rio de Janeiro, Brazil
¶Department of Electronics Engineering and Telecommunications – Faculty of Engineering
Universidade do Estado do Rio de Janeiro, Brazil
‡Department of Electrical Engineering – Electronic Systems
Eindhoven University of Technology, The Netherlands
§Silicon Hive – Intel UMG, The Netherlands
E-mails: solon@cos.ufrj.br, L.Jozwiak@tue.nl, Menno.Lindwer@siliconhive.com,
Mauro.Cocco@siliconhive.com, nadia@eng.uerj.br, felipe@cos.ufrj.br

***Abstract*—Effective exploitation of the application-specific parallel patterns and computation operations through their direct implementation in hardware is the base for construction of high-quality application-specific (re-)configurable application specific instruction set processors (ASIPs) and hardware accelerators for modern highly-demanding applications. Although it receives a lot of attention from the researchers and practitioners, a very important problem of hardware reuse in ASIP and accelerator synthesis is clearly underestimated and does not get enough attention in the published research. This paper is an effect of an industry and academic collaborative research. It analyses the problem of hardware sharing, shows its high practical relevance, as well as a big influence of hardware sharing on the major circuit and system parameters, and its importance for the multi-objective optimization and tradeoff exploitation. It also demonstrates that the state-of-the-art synthesis tools do not sufficiently address this problem and gives several guidelines related to enhancement of the hardware reuse. *Index Terms*— Resource Sharing, Application-Specific Processors, Hardware Accelerator, Area Reduction, Power Reduction.**

I. Introduction

A large majority of the development effort of an application-specific processor or hardware accelerator is focused on achieving a required acceleration for highly-demanding application or a class of applications. In both cases, a detailed application analysis is required to identify and efficiently exploit the acceleration opportunities. It is well known that the performance gain of an Application-Specific Instruction set Processor (ASIP) or a hardware accelerator over a general-purpose processor is substantial [1].

However, in parallel to the computation speed or throughput, several other important aspects have to be addressed in the ASIP or accelerator design, such as circuit area and power consumption. A careful and adequate consideration of these important aspects is especially important in modern embedded system domains. Unfortunately, the system and circuit designers too often let the synthesis tools in charge of applying some design optimizations to lower the area and power consumptions of their designs. Although most commercially available synthesis tools are capable of performing some optimizations that may lower the area and power consumption, the result may often be unsatisfactory. It is important to understand that each synthesis tool implements a limited set of limited quality design transformations that are selected and applied using limited information on a particular a design. Using them, the synthesis tool struggles between timing, area and power constraints in order to find a good design balance. The synthesis tools are thus far from ideal and can still much be improved. It is however also up to the designer to assist the synthesis tool through a more precise and tool-targeted description of their designs and adequately controlling the tool. Some of the commercial tools include some guidelines and methods on how to specify a digital design or set some tool parameters to achieve better timing, area or power results.

Resource sharing is a well-know circuit and system optimization approach traditionally employed for saving the circuit area. However, in parallel to creation of new important opportunities, the recent spectacular progress in the semiconductor technology, and particularly the introduction of the nano-dimension CMOS technologies, has created many difficult to solve problems. The system and circuit complexity, energy and power issues, the interconnect scalability problems and on-chip communication issues are some of the most important. Moreover, the introduction of the nano-CMOS technologies changed the importance relationships among various design aspects. For instance, the static power negligible in the past is now comparable to its dynamic counterpart, and the interconnects instead of active elements tend to have a dominating influence on the major SoC characteristics (area, throughput, etc.). On the other hand, many modern mobile/autonomous applications in several fields like communications, multimedia, security, military and medical instruments impose extremely high throughput and/or energy related requirements.

Unfortunately, the contemporary circuit and system design methods and tools remain behind the actual needs of the modern applications and technologies. In particular, they do

978-0-7695-4494-6/11 $26.00 © 2011 IEEE
DOI 10.1109/DSD.2011.22

not well account for the changed importance relationships of different design aspects, and for the now necessary multi-objective decision modeling and careful tradeoff exploitation among various design objectives. This particularly relates to the resource sharing aspects that can be exploited to limit the system and circuit area, but equally well, to limit the system and circuit energy consumption, and in some specific cases, even to increase the speed. This way, hardware reuse is one of the major aspects of the multi-objective system and circuit optimization and tradeoff exploitation.

This paper addresses the important problem of hardware reuse in synthesis of ASIPs and hardware accelerators implemented in modern nano-CMOS technologies. It is an effect of an industry and academic collaborative research on analysis and resolution of the resource sharing problem which is of high practical relevance. It analyses the problem and presents important results of high practical relevance from our extensive experimental research regarding this issue. It demonstrates that the state-of-the-art automatic synthesis tools do not address this problem in a sufficient way, and shows how important an effective hardware reuse is for an adequate synthesis of ASIPs and hardware accelerators. It demonstrates a big influence of hardware reuse on all the major physical parameters: computation speed, circuit area and energy consumption. The resource sharing potential should be exploited along the whole system development, and using various approaches and means. In the part of our research related to this issue that is reported in this paper, we focused on resource sharing during the ASIP accelerator micro-architecture synthesis for execution of similar RTL-level operations that are mutually exclusive (are not going to be executed at the same time). We present and discuss results of our extensive experimental research on resource sharing performed with commercially available synthesis tools. The results show that the contemporary synthesis tools are still not able to produce satisfactory resource sharing results in many cases.

The rest of this paper is organized as follows: Section II gives a brief introduction on application-specific processors and synthesis design flow. Section III introduces the resource sharing problem, and Section IV exemplifies some guidelines to perform resource sharing. Finally, Section V presents experimental results, and Section VI draws the conclusions of this research.

II. Design of Hardware accelerators and application-specific processors

Application-specific processors and hardware accelerators have recently become a popular implementation technology for accelerating high-demanding modern embedded applications [1], [2], especially the ones that can efficiently be parallelized and executed by multiple processors. Multimedia and communications applications often fall into the category of highly parallel applications, as for example the LDPC decoding [2] and the Ray-Tracing algorithm [3]. Since the modern communications and multimedia applications are usually very demanding regarding throughput, energy consumption and other features, they often require a sophisticated application-specific hardware implementation [2], [4].

If a high flexibility is also required to accommodate late design changes or even some field-use changes, then a programmable application-specific processor can be a more attractive choice than a hardware accelerator. For instance, in a customizable Very Long Instruction Word processor (VLIW) [5], vector issue slots can be made to exploit the application data parallelism, as well as issue slots can be customized and additional issue slots can be inserted to increase the Instruction Level Parallelism (ILP). The application has of course to be recompiled, in order to actually exploit the ILP. Also, instead of changing the hardware, the compiler can perform some further optimizations of the application code so that it can be executed more efficiently at the existing hardware. In both cases, a deep application analysis must be performed to identify and efficiently exploit the acceleration opportunities.

In this paper, we focus on the hardware sharing aspect at the ASIP and accelerator RTL-level architecture synthesis. More specifically, we focus on the hardware sharing in the final part of the RTL-level architecture synthesis, when the architecture design of an ASIP or accelerator is expressed in an RTL-level hardware description language (HDL) (e.g. VHDL or Verilog) to be sub-sequently transferred to the (commercial) RTL-level, logic-level and physical-level hardware synthesis tools.

III. The resource sharing problem

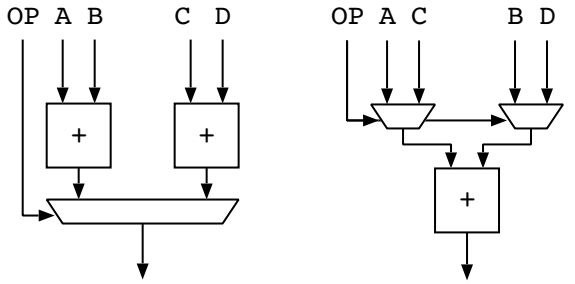
Resource sharing at the RTL-level is one of possible hardware optimization techniques traditionally used for saving circuit area [6], [7], [8]. It can also deliver substantial power consumption savings, what will be shown in the paper. It works by sharing one or more functional units (FUs) to implement several operations described in HDL, producing less hardware components in the final netlist representation. Therefore, resource sharing may also substantially reduce the power consumption or even increase the design performance.

RTL-level resource sharing can be performed at several stages of the ASIP and accelerator micro-architecture synthesis, exploiting a.o. the high-level synthesis scheduling and mapping, application-specific instruction creation [4] or RTL-level compiler optimizations. Since the high-level synthesis is very broadly discussed in literature, in our work we focused on hardware sharing for application-specific instructions and through the RTL-level compiler optimizations.

One solution for saving resources in ASIPs is to introduce datapath merging, in which graph (or sub-graph) representations of different instruction set extensions can be combined, or partially combined, into one datapath. In [9], application profiling and dataflow graphs (DFGs) are used to perform custom instruction selection and to automatically construct a corresponding pipelined (re-)configurable hardware extension, as well as in [10]. Furthermore, resource-efficient pipelined configuration is performed, achieving an average area reduction of 43.9%. In [11], [12], [13], similar resource sharing techniques are used together with a heuristic algorithm to

Listing 1: VHDL architecture example for two add operations.

```
architecture example of FU is begin
process(A,B,C,D,OP)
begin
  if(OP = '0')then
    R <= A + B;
  else
    R <= C + D;
  end if;
end process;
end example;
```



(a) No shared resources. (b) Shared resources.

Fig. 1: Resource sharing example.

control the degree of resource sharing between a set of custom instructions.

Despite various hardware sharing optimizations performed at the earlier micro-architecture synthesis stages, there is usually a substantial further hardware sharing potential in the input description for the RTL-level compiler. Moreover, simple RTL-level hardware description changes can greatly affect the way a given design is synthesized. For instance, changing conditional statements, such as *IF* or *CASE* statements, may produce a different circuit [6]. The mutually exclusive forms are one of the bases for successful resource sharing. In [14], independent of which conditional construction is used, the authors proposed to explicitly identify the mutually exclusive execution branches through transformations of the original conditional description. In the end, a larger set of mutual exclusive branches is detected, allowing a better reuse of resources. Consider first a simple example presented in Listing 1.

Without resource sharing optimizations, the synthesis tool would create separate circuits, one for each operation, as in Fig.1a. However, since both operations are never used at the same time, it is possible to allow for resource sharing transformations, and in consequence, create only one adder, as depicted in Fig.1b. When operations have to be executed at the same time, resource sharing may also be exceptionally applied through pipelining of operations[10], [9], although not as straightforward as for the mutually exclusive operations, due to possible data and control dependency hazards.

A. Automatic resource sharing

Most commercial synthesis tools are able to automatically detect and exploit some limited resource sharing possibilities within a digital design [8], as for instance shown in the example of Fig.1, based on Listing 1. Usually, automatic resource sharing is enabled whenever the synthesis tool is set to the area reduction effort and operations are never executed at the same time [14]. Furthermore, in Fig.1 it is possible to observe that applying resource sharing caused an exchange of a relatively complex functional unit (adder) for a simpler extra multiplexer. This is usually the case, because sharing of one functional unit among many operations implies the usage of multiplexers to select which input data to process, what also may introduce the cost of a possible additional latency [6]. If timing constraints cannot be met, the synthesis tool may disable resource sharing in advance, what is not always a clever decision.

B. Manual resource sharing

The synthesis tools are often not capable to automatically exploit the sharing opportunities, mainly because of the following three reasons:

1) The hardware is not specified in an appropriate style, so that the tool is unable to identify sharing opportunities;
2) The tool misses more global and extensive application analysis, and only exploits some local optimizations for resource sharing;
3) Resource sharing is not possible due to timing constraints issues.

The first issue can often be resolved by rewriting the hardware description, so that resource sharing opportunities are better visible to a particular tool. For instance, instead of performing the operation inside of each conditional statement, the input data is first selected. Only after the evaluation of the boolean expressions and input data selection, the operation is performed, as shown in Listing 2 for example. The second issue can only sometimes be solved by specification rewriting, but often requires the synthesis method and tool extensions. The third issue cannot easily be solved by only rewriting the specification. It requires a careful analysis of the digital design in order to perform complex algorithm optimizations that could reduce the digital system latency, such as the replacement of Ripple-Carry adders by faster Carry-Save adders [15], [16], [6].

IV. General guidelines

In this section, some general guidelines are briefly presented that make possible to achieve a high resource sharing. The VHDL hardware description in Listing 3 is used as an example. It should be clear by now that in essence a resource sharing for two or more operations is only possible if the operations are never going to be executed at the same time, as presented in Listing 3. In this example, operations (A+B) and (C+D) can share a single adder within the first process, but cannot share the same adder across with the second process,

Listing 2: Rewriting Listing 1 architecture to ensure resource sharing.

```
architecture example of FU is begin
process(A,B,C,D,OP)
var X,Y : std_logic_vector(N-1 downto 0);
begin
  if(OP = '0')then
    X <= A;
    Y <= B;
  else
    X <= C;
    Y <= D;
  end if;
  R <= X + Y;
end process;
end example;
```

because the second process can be executed concurrently with first one.

Listing 3: Architecture description with mutually exclusive statements and separate processes.

```
architecture example of FU is begin
P1: process(A,B,C,D,OP1)
begin
  if(OP1 = '0')then
    R1 <= A + B;
  else
    R1 <= C + D;
  end if;
end process;
P2: process(X,Y,Z,W,OP2)
begin
  if(OP2 = '0')then
    R2 <= X + Y;
  else
    R2 <= Z - W;
  end if;
end process;
end example;
```

Also, note that operations (X+Y) and (Z-W) can be shared within the second process, because they are similar: a subtraction can be implemented as an addition with the second operand inverted and with incoming carry equal to one in the lowest position. Thus, the synthesis software should create in this situation a single adder functional unit and inverter for the second operand to enable subtraction. This kind of resource sharing is also referred as *Functionality Sharing* and is more difficult to be achieved automatically, because it requires a deeper knowledge of the relationships between the operations (or functionalities) that can be merged into a single hardware. Further examples can be found in [6], [7].

Nevertheless, in general, placing operations considered for possible sharing more explicitly at mutually exclusive branches in the RTL-level hardware specification may facilitate the identification of the resource sharing potential. For this reason, the use of conditional *CASE* statements should be preferred, because the case branches have to be mutually exclusive. A similar principle applies for operations described in separate processes: inside the process, statements are executed sequentially, but two or more processes are executed in parallel.

V. Discussion of Experimental Results

In this section, some selected results are discussed of our experiments with ASIP and accelerator hardware sharing at the RTL-level. The experimental results are based on the synthesis of a functional unit which performs four similar operations, that are described in Table I. Four hardware architecture description styles of the same functional unit entity are presented, so that resource sharing potential identification and exploitation can be analyzed for each case. The first description style is a mixed structural and behavioral, while the others are purely behavioral. Also, the selected operations are similar to each other, so that they may be merged into single operators, such as an adder/subtractor.

TABLE I: Table of operations for the functional unit.

OPERATION	OPERANDS
Addition	$R <= A + B$
Subtraction	$R <= A - B$
Multiplication	$R <= A \times B$
Multiply–Accumulate	$R <= A \times B + C$

The operations cannot be executed at the same time. Since they are mutually exclusive, they can be shared. A black-box diagram of the functional unit is presented in Fig.2. Its computation result is produced based on the selected operation and input data. The synthesis results of the functional unit from a state-of-the-art commercial synthesis tool regarding the area and delay are presented in Table VI, for each hardware description style and area reduction effort (medium or high), with the high area reduction effort enabling automatic resource sharing. First, each architectural description style of the functional unit is presented and discussed together with the corresponding synthesis results. At the end of the section, the results are further discussed, compared and summarized. All the presented results were collected after logic synthesis.

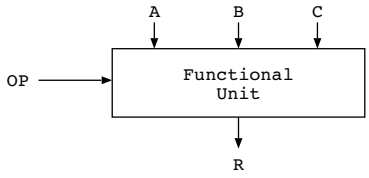


Fig. 2: Black-box diagram of the functional unit.

A. Structural RTL description

The first architecture description style is a structural description of the functional unit, and it corresponds to a particular optimal circuit that is expected to be created. However, each sub-component of this structural description was described in a behavioral style, so that the synthesis software is in charge of creating or selecting an appropriate final hardware.

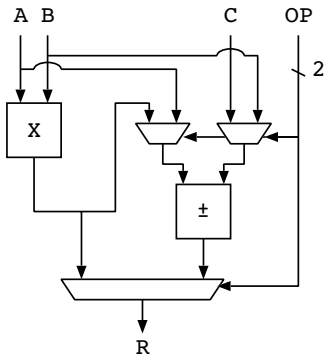


Fig. 3: Structural design.

Listing 4: Architecture description using case statements.

```
architecture example of FU is begin
process(A,B,C,OP)
begin
  case OP is
    when MUL =>
      R <= A * B;
    when MAC =>
      R <= A * B + C;
    when SUB =>
      R <= A - B;
    when others =>
      R <= A + C;
  end case;
end process;
end example;
```

In this structural description, the synthesizer was able to precisely identify each component as they were described, as depicted in Fig.3. Also, a high area reduction effort exploited some resource sharing by merging the adder and the subtractor into a single hardware. The results are presented in Table II, for both area reduction efforts. As we can observe from it, the overall area is greatly reduce thanks to resource sharing. However, the resource sharing optimizations can increase the worst path delay, because of extra multiplexors that are included to select between different input data, as depicted in Fig.3.

TABLE II: Structural description synthesis results.

Effort	Cells	Cell Area	Net Area	Total Area	Delay*
Medium	2119	3717	8944	12661	4025.7
High	1657	3151	8198	11349	4126.0
Rate	-21.80%	-15.23%	-8.34%	-10.36%	+2.5%

*Delay time in pico-seconds.

B. Conditional case statements

In the second architecture description style, as well as in the following architecture descriptions presented in Sections V-C and V-D, the hardware is purely described in behavioral style. In the architecture speculation style of Section V-B, conditional case statements are used to select between one of the four operations, according to the architecture description in Listing 4.

As expected, with the high area effort, the resource sharing was also performed for both adders and the subtractor operation, merging them into a single hardware unit. Also, only one multiplier was created, instead of two, as depicted in Fig.4, and the multiplexors were replaced by an optimized logic in the logic synthesis stage. On the other hand, disabling resource sharing possibility through not selecting the high area effort produced more hardware blocks, as shown in Table III. Additionally, carry-save adders were included into the design, that greatly contributed to increase the Net Area. Also, timing constraints were not met for a medium area reduction effort, possibly because of the longer paths due to the higher Net and Total Area. Although the overall area for this design style with a high area reduction effort has been reduced, it is still larger than the area of the structural design of the previous section under the same reduction effort. This shows that the synthesis tools did not realize an adequate resource sharing as expected from this design style. Regarding the worst path delay, a small improvement can be observed, as a result of using optimized logic instead of multiplexors in the inputs, as in Fig.4.

TABLE III: Case statements description synthesis results.

Effort	Cells	Cell Area	Net Area	Total Area	Delay*
Medium	1985	3850	33120	36970	4161.7
High	1754	3254	8368	11622	4067.3
Rate	-11.63%	-15.48%	-74.73%	-68.56%	-2.26%

*Delay time in pico-seconds.

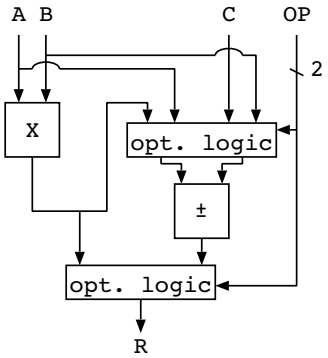


Fig. 4: Case statement design style.

C. Conditional if statements

Since the usage of conditional *case* statements produced worse than expected results, the architecture description style was changed from *case* to *if* conditional statements. Also, following the example presented in Listing 1 to enhance the possibility of resource sharing, the inputs are first selected, so that the operation is executed afterwards, as shown in Listing

5. Such behavioral description is an attempt to achieve a more substantial resource sharing, aiming at producing a design as close as possible to the structural one depicted in Fig.3.

Listing 5: Architecture description using if statements.

```
architecture example of FU is begin
process (A,B,C,OP)
  variable var_mlt:std_logic_vector(2*N-1 downto 0);
  variable v_A,v_B:std_logic_vector(N-1 downto 0);
  variable output :std_logic_vector(N-1 downto 0);
begin
  var_mlt := A * B;
  if(OP = MAC) then
    v_A := var_mlt(N-1 downto 0);
    v_B := C;
  else
    v_A := A;
    v_B := B;
  end if;
  if(OP  = MUL) then
    output := var_mlt(N-1 downto 0);
  elsif(OP = SUB) then
    output := v_A - v_B;
  else
    output := v_A + v_B;
  end if;
end process;
end example;
```

From the results presented in Table IV, it is possible to observe that even-though a high area reduction effort produced a slightly lower Cell Area compared to the medium effort, the Net Area increased substantially, in contrast to the results using the *case statements* in Section V-B. Thus, a high area reduction effort created a worse design regarding the Net Area and could not find resource sharing between any of the operations. One possible explanation is that the design is already described in a style aware of resource sharing. Therefore, applying a high reduction effort introduced further optimizations that worsen the design. For instance, carry-save adders were included by the synthesis tool, which could contribute to increase of the Net Area, as depicted in Fig.5. As we can see, a separate carry-save adder was created for almost each operation and the multiplexors were also exchanged for optimized logic. The interconnections between them are simplified in Fig.5. On the other hand, a medium area reduction effort yielded a better Net Area result, because no carry-save adders were included with this style, and no functional merging of the adder and the subtractor was performed at all.

TABLE IV: If statements description synthesis results.

Effort	Cells	Cell Area	Net Area	Total Area	Delay*
Medium	1952	3740	9739	13479	4062.5
High	1912	3689	10497	14186	3939.4
Rate	-2.05%	-1.36%	+7.78%	+5.25%	-3.03%

*Delay time in pico-seconds.

As we can observe from Table IV, the worst path delay in this design style is better than the worst path delay in the previous design examples. As expected, the carry-save adders were able to increase the design's speed, while the Net Area also increased.

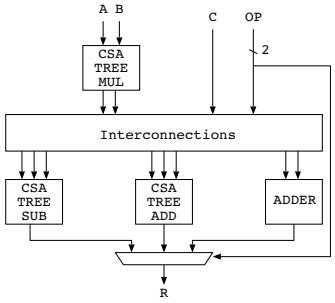


Fig. 5: If statement design style.

D. Cascaded conditional if statements

Finally, the architecture description from Listing 5 was slightly modified to the one presented in Listing 6, with the only difference being the cascaded (or nested) if statement for selecting between the addition and subtraction operations. The idea is to keep these two operations close to each other in one internal *if-else* block of the nested if structure, so the synthesizer can more easily identify the possible resource sharing between them, because in the previous example it clearly could not identify the resource sharing potential.

Listing 6: Architecture description using cascaded if statements.

```
architecture example of FU is begin
process (A,B,C,OP)
  variable var_mlt:std_logic_vector(2*N-1 downto 0);
  variable v_A,v_B:std_logic_vector(N-1 downto 0);
  variable output :std_logic_vector(N-1 downto 0);
begin
  var_mlt := A * B;
  if(OP = MAC) then
    v_A := var_mlt(N-1 downto 0);
    v_B := C;
  else
    v_A := A;
    v_B := B;
  end if;
  if(OP  = MUL) then
    output := var_mlt(N-1 downto 0);
  else
    if(OP = SUB) then
      output := v_A - v_B;
    else
      output := v_A + v_B;
    end if;
  end if;
end process;
end example;
```

Once again the results were unsatisfactory, especially regarded the Net Area, as shown in Table V. Carry-save adders were also included at the cost of an additional Net Area requirement. In contrast to the *if statement* design example, this design style actually helped the synthesis tool to identify

the resource sharing between the adder and the subtractor operators, again merging them into a single hardware. The worst path delay is the best among all the others, even-though the Net Area has increased substantially and no carry-save adders were introduced. The circuit schematic is very similar to the schematic of the *case* design in Fig.4, with a few additional optimizations in the multiplexors.

TABLE V: Cascaded if statements description synthesis results.

Effort	Cells	Cell Area	Net Area	Total Area	Delay*
Medium	1955	3525	8509	12034	4148.8
High	1783	3576	16071	19647	3883.4
Rate	-8.8%	+1.45%	+88.9%	+63.26%	-6.40%

*Delay time in pico-seconds.

E. Results overview

The experimental results are summarized in Table VI and in Fig.6, for a process of 40nm and a clock period of 5.0 ns. First of all, it is clear that the structural architecture description, with the implementation structure imposed by a human designer to a high degree, is the one that produced the best results with high area reduction effort, as shown in Fig.6b. To get the high-quality results, a structural description style is usually preferred over a behavioral one, because a human designer or a higher-level architecture synthesis tool can more directly and precisely specify the required design features and such description can be easier translated by the synthesis tool to a corresponding high-quality netlist. Also, enabling the resource sharing via a high area reduction effort clearly proved to be useful for saving resources, and resulted in substantial area improvements, with a Cell reduction area ratio of 15.23% and Net reduction area ratio of 8.34%, for such a structural design. The hardware sharing also reduced the overall area in different design styles. Furthermore, if only the medium area reduction effort is considered, then the best area result is achieved by the last set of experiments: the *cascaded if statements*. Such description style was able to improve the area cost, because the architecture was described in the style easy for resource sharing. Further optimizations performed with a high area reduction effort mainly included additional circuitry to the design and increased the Net Area, but reduced the worst path delay, as shown in Fig.7.

The resource sharing result very much depends on the hardware description style, as also observed in Table VI. For some design styles, a substantial increase in Net Area can be observed, especially due to further timing optimizations. The inclusion of carry-saver adders in several cases greatly increased the Net Area requirements, in an effort to overcome the extra delays that might arise from hardware sharing. Observe that when considering the functional unit sharing, the designers tend mainly to think on reduction of the functional unit (Cell) area, and not on the reduction of the interconnection (Net) area. However, due to the interconnect scalability problems and their dominating influence on the design in modern nano-CMOS technologies, we could observe larger changes in the interconnect area than in the cell area.

Also, observe that the RTL-level compiler optimization techniques involving resource sharing may cause very big circuit area changes (as high as -68% and +63% in the performed experiments), and involve substantial area/delay tradeoffs. Since for the modern nano-CMOS circuit implementation technologies the circuit power consumption of various circuits implementing a given computation is roughly proportional to the circuit area [17], these optimization techniques also may cause very big circuit power consumption changes and substantial power/delay tradeoffs.

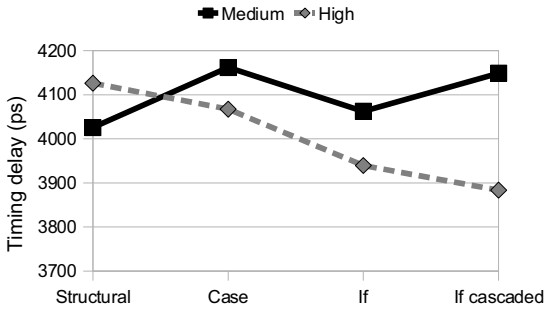


Fig. 7: Worst path delay of each design style.

VI. Conclusions

In this paper we analyzed the problem of hardware sharing in ASIP and accelerator synthesis. We focused on the final part of the micro-architecture synthesis, where the architecture design is expressed in an RTL-level HDL to be sub-sequently transferred to a RTL-level synthesis tool. Despite various hardware sharing optimizations at the earlier synthesis stages, there is usually a substantial hardware sharing potential in this RTL-level description. In the part of our research presented in this paper, we focused on experimental analysis of hardware sharing when applying commercial RTL-level synthesis tools. Our main conclusions from this research are as follows.

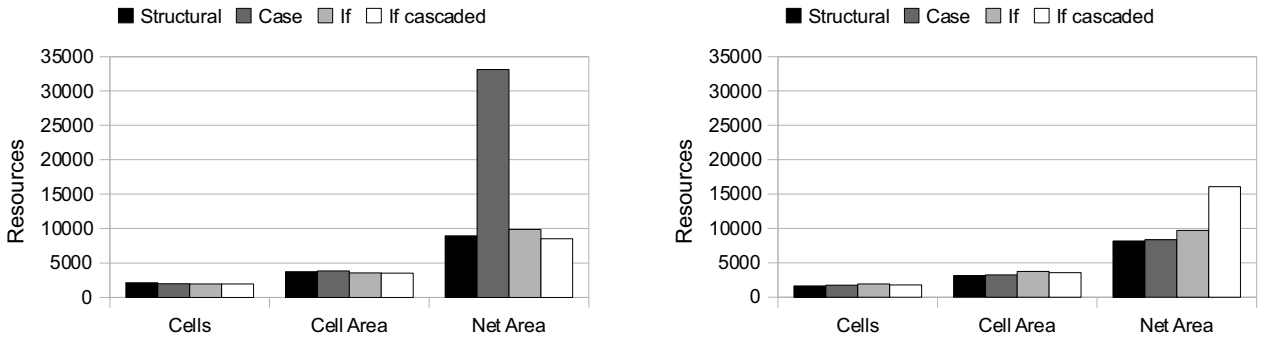
Exploitation of hardware sharing at the RTL-level can influence the area and power consumption of ASIP and accelerator designs to a very high degree and may cause substantial area/power/delay tradeoffs. Thus, adequate RTL-level optimization techniques involving resource sharing are of primary importance for the multi-objective ASIP and accelerator optimization and adequate tradeoff exploitation. Moreover, when considering functional unit sharing, the designers tend mainly to think on the reduction of functional unit area, while we observed larger changes in the interconnect area than in the active area. This is due to the interconnect scalability problems and their dominating influence on the designs in the modern nano-CMOS technologies.

The results produced by the RTL-level compiler optimization techniques involving resource sharing very much depend on the RTL-level design description style and optimization objectives specified for the used commercial synthesis tool.

TABLE VI: Synthesis results, for different design styles and area reduction efforts.

Design style	Structural		Case statements		If statements		Cascade if statements	
	Total Area	Delay	Total Area	Delay	Total Area	Delay	Total Area	Delay
Medium	12661	4025.7	36970	4161.7	13479	4062.5	12034	4148.8
High	11349	4126.0	11622	4067.3	14186	3939.4	19647	3883.4
Rate*	-10.36%	+2.5%	-68.56%	-2.26%	+5.25%	-3.03%	+63.26%	-6.40%

*A plus signal before **rate** data actually indicates an increase in area or delay. Delay time is in pico-seconds (ps).



(a) Medium area reduction effort.

(b) High area reduction effort.

Fig. 6: Area synthesis results.

The structural specification style, in which the implementation structure is imposed in a high degree by a human designer, is the most preferred to achieve the best results regarding the area and energy savings. However, also the behavioral case-based specification resulted in a high area reduction and a very low delay. Regarding delay reduction, the best results come from cascaded-if style, but unfortunately on the high cost of area.

Using different specification styles and optimization efforts, very different results can be produced that involves substantial tradeoffs between area and power from one side and delay from the other. The commercial RTL-level synthesis tools are very sensitive to the input specification style and optimization effort. Using them, it is very difficult to handle the resource sharing appropriately. Clearly, further research and development of those tools is necessary. Currently, the tools may much improve, but also much destroy the architecture designs, and therefore the fully automatic use is difficult. Nevertheless, our experiments delivered some useful guidelines and hints on how to control the tools.

REFERENCES

[1] L. Józwiak and N. Nedjah, "Modern architectures for embedded reconfigurable systems - a survey," *Journal of Circuits, Systems, and Computers*, vol. 18, no. 2, pp. 209–254, 2009.

[2] L. Jozwiak and Y. Jan, "Quality-driven methodology for demanding accelerator design," in *Quality Electronic Design (ISQED), 2010 11th International Symposium on*, march 2010, pp. 380 –389.

[3] J. Hurley, "Ray tracing goes mainstream," *Intel Technology Journal*, vol. 9, no. 2, pp. 99–107, May 2005.

[4] L. Jówiak, N. Nedjah, and M. Figueroa, "Modern development methods and tools for embedded reconfigurable systems: A survey," *Integr. VLSI J.*, vol. 43, pp. 1–33, January 2010. [Online]. Available: http://portal.acm.org/citation.cfm?id=1663659.1663983

[5] J. A. Fisher, P. Faraboschi, and C. Young, *Embedded Computing : A VLIW Approach to Architecture, Compilers and Tools*. Morgan Kaufmann, 2004.

[6] P. P. Chu, *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. Newark, NJ: Wiley-IEEE Press, 2006.

[7] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*. Wiley-IEEE Press, 2007.

[8] Xilinx, *Xilinx Synthesis Technology: User Guide*, 2008, january, 2009. [Online]. Available: http://www.xilinx.com/itp/xilinx5/pdf/docs/xst/xst.pdf

[9] H. Lin and Y. Fei, "Resource sharing of pipelined custom hardware extension for energy-efficient application-specific instruction set processor design," in *Proceedings of the 2009 IEEE international conference on Computer design*, ser. ICCD'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 158–165.

[10] C. Ulmer and A. Javelo, "Floating-point unit reuse in an fpga implementation of a ray-triangle intersection algorithm." in *ERSA'06*, 2006, pp. 93–102.

[11] M. Zuluaga and N. Topham, "Resource sharing in custom instruction set extensions," *Application Specific Processors, Symposium on*, vol. 0, pp. 7–13, 2008.

[12] M. Zuluaga, T. Kluter, P. Brisk, N. Topham, and P. Ienne, "Introducing control-flow inclusion to support pipelining in custom instruction set extensions," in *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, july 2009, pp. 114 –121.

[13] M. Zuluaga and N. Topham, "Exploring the unified design-space of custom-instruction selection and resource sharing," in *Embedded Computer Systems (SAMOS), 2010 International Conference on*, july 2010, pp. 282 –291.

[14] O. Peñalba, J. M. Mendías, and R. Hermida, "A global approach to improve conditional hardware reuse in high-level synthesis," *Journal of Systems Architecture*, vol. 47, pp. 959–975, June 2002.

[15] I. Koren, *Computer arithmetic algorithms*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2001.

[16] B. Parhami, *Computer arithmetic: algorithms and hardware designs*. Oxford, UK: Oxford University Press, 2009.

[17] L. Jozwiak, D. Gawlowski, A. Ślusarczyk, and A. Chojnacki, "Static power reduction in nano cmos circuits through an adequate circuit synthesis," June 2007, pp. 172 – 177.