Exploration of power reduction and performance enhancement in LEON3 processor with ESL reprogrammable eFPGA in processor pipeline and as a co-processor

Syed Zahid Ahmed[1,2], Julien Eydoux[1], Laurent Rougé[1], Jean-Baptiste Cuelle[1],
Gilles Sassatelli[2], Lionel Torres[2]
1 (Menta, France), 2 (University of Montpellier 2, UMR CNRS 5506, LIRMM, France)
1 (ahmed, eydoux, rouge, cuelle @menta.fr), 2 (ahmed, sassatelli, torres @lirmm.fr)

Abstract

We will explore how processing power of LEON3 processor can be enhanced by connecting small commercially available embedded FPGA (eFPGA) IP with the processor. We will analyze integration of eFPGA with LEON3 in two ways, inside the processor pipeline and as a co-processor. The enhanced processing power helps to reduce dynamic power consumption by Dynamic Frequency Scaling. More computational power at lower frequency helps fabrication of chip in LP (Low Power) process compared to GP (General Purpose) which helps to significantly reduce Static Power which has become a very crucial issue at and beyond 90nm technologies.

Use of reconfigurable accelerator raises the question of its programming complexity, HW/SW partitioning and silicon overhead. We will present that silicon overhead of eFPGA is small compared to the benefits which can be obtained with it. We will present a profiling tool which we created for our experiments. To analyze the issue of programming complexity we have explored state of the art Catapult™ ESL tool of Mentor Graphics®.

1. Introduction

The explosion in the demand of portable devices which are rich in multimedia and internet applications has brought several new challenges to semiconductor industry in recent years. In addition to the usual ever increasing processing power demands industry now also has to face the challenge of power consumption for increasing the battery life. As technologies have moved beyond 90nm the static power has become as much significant as the dynamic power. To reduce the static power Fabs have come up with Low power (LP) process technologies to reduce leakage of transistors. Starting from 45nm intel® has even moved to high-k transistors with metal gate (Hafnium) to reduce leakage. But reduction in static power comes with the trade off in speed [1] which is of fundamental importance to cope with ever increasing processing power demands.

Microprocessors are at the heart of the semiconductor designs. In this paper our target is to explore how we can improve the processing power and reduce power consumption using reconfigurable computing. The concept of using accelerators is not new and is widely used in industry. We got the inspiration from state of the art solutions of ARC® [2] and Tensilica® [3] which are widely used in consumer electronics products. They profile the application and create new instructions by custom hardware for critical computational intensive portions of the program and by doing so significantly improve computational power and hence reduce the power consumption.

Such solutions are very efficient for specific known applications. But in case of current mobile applications scenario for instance, the application processor is running several applications like PC and is running OS for mobiles like Symbian®, Windows CE®, Linux, Mac® etc. It is difficult to know even for the vendors each and every kind of acceleration hardware which they may need as now users also create several applications for their devices (iPhone™ is a very prominent example). To give advantage of acceleration and energy reduction for portable devices it will be helpful to have small reconfigurable accelerators in addition to standard ones, which can be reconfigured for the specific needs.

This research is done in close research collaboration between startup Menta® (www.efpga.com) and LIRMM (research center of University of Montpellier 2 and CNRS). In the research collaboration we are investigating new paradigms in which commercially available embedded FPGA IP of Menta® can help provide advantages of reconfigurable computing to industry. The main targets are the key problems of industry like product differentiation, future up gradation, fast time to market, enhancement of computational power and issue of power consumption which is most dominant challenge of the time and this paper is dedicated to this issue. We are exploring how to improve eFPGA architecture to achieve further benefits for wide range of applications.

As this research is industrial oriented we also analyzed several similar approaches of the past in this area both from academia and industry. It is widely known that although scientifically they had strong potential, but almost all of them commercially failed or found very limited acceptance in very specific segment of industry. A prominent example is XiRisc [4], several others e.g. GARPH [5], PipeRench [6] etc. can be analyzed on internet. In our opinion all of those approaches had a good scientific & theoretical potential but they all failed mainly because of commercial reasons. These approaches were quite restrictive for adoption in wider range. The programming of these solutions required additional efforts than standard languages (like ANSI C, VHDL, Verilog etc.)

978-3-9810801-5-5/DATE09 © 2009 EDAA

and needed use of specialized compilers to program the reconfigurable portion. For instance XiRisc, the solution has a good potential but from commercial point of view its reconfigurable portion (PiCoGA) is unavailable and virtually useless for industry which for instance is highly ARM® & MIPS® dominant. We have considered all these issues of great importance in our research and want to remove the barriers and provide a programmable IP to the industry which is general, easy to program and usable for everyone for wide range of applications.

Therefore it is important to mention that, although we are following the same theoretical path of previous approaches but our target is much different and broader. The eFPGA IP is not only for reconfigurable accelerator but it is one of the prominent applications. We think that this will be a good contribution to industry and help them solve several of their biggest challenges like time to market, product differentiation and power consumption etc. eFPGA IP will bring advantages of FPGAs directly inside the SoC.

For the processor we chose LEON3 [7] because it is close in standard and quality like basic commercial processors of ARM® and MIPS®. We analyzed the integration of eFPGA in two ways, inside the processor pipeline and as a co-processor. Our results with LEON3 have showed that co-processor implementation has a small impact on performance compared to direct pipeline implementation. But the overall benefits of co-processor approach (most prominently no change in processor integrity) exceeds the small performance loss compared to pipeline integration due to additional cycles spent in co-processor interface.

Reconfigurable accelerators bring two major questions, first how much will be the silicon overhead and second how we will program them. We have considered both issues in our work. We will present that the overhead of Area and Power of eFPGAs is small compared to benefits which it brings. For programming eFPGA we have made investigations of automatically generating the VHDL or Verilog for eFPGA from ANSI C/C++ source using Mentor Graphics® Catapult™ (www.mentor.com). We have found that programming at ESL level is much faster and easier compared to manual HDL programming. Design space exploration with ESL is much faster and with the advances in ESL the gap between hand coded and ESL generated HDL is decreasing.

The rest of the paper is organized as follows. In section 2 we will describe our experimentation methodology. We will briefly explain our eFPGA architecture, the modifications which we made to LEON3 and a profiling tool which we have created to analyze applications for HW/SW partitioning. In section 3 we will present results of AES and DES cryptographic algorithms to illustrate the concept and compare the results for pipeline and co-processor implementation of LEON3+eFPGA. In section 4 we will analyze the trade offs in Area, Power & Speed by the use of eFPGA with LEON3 on 65nm Low power (LP) process technology. Finally in section 5 we will conclude our work and explain future ideas and plans.

2. Introduction to the hardware and experiment methodology

In this section we will briefly discuss the building elements of our hardware, the tools and experiments methodology. We will use these concepts and hardware to demonstrate practical examples in section 3 and in section 4 we will present their actual silicon implementations on 65nm to analyze area, power and speed statistics of our experiments.

2.1 The eFPGA

At the heart of our experiments in this work is the eFPGA which we have designed. The abstract view of the eFPGA is shown in fig. 1. We have completely designed it as a soft IP. It is completely written in VHDL so is technology independent. As it is soft IP there are no SRAM cells, Pass transistor or tri state buffer switches. The configuration element is a Flip Flop and switching element is a Multiplexer. The core is highly configurable. We can select all the fundamental parameters of the eFPGA like LUT size, Cluster size (number of LUT in a eCB), Routing channel size and array size (number of eCBs) etc. This highlights the advantage of its high flexibility. It is very easy to generate the IP of the user requirements. The details of eFPGA architecture are beyond the scope of this paper. We have discussed more details about it in [1]. In this paper we only use it as a reconfigurable accelerating element and generate the IP of our requirements.

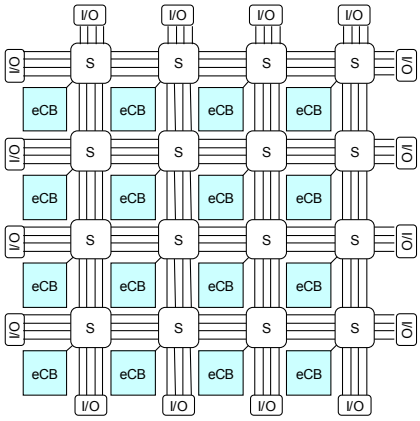

Fig. 1 : Abstract view of eFPGA

2.2 LEON3 processor modifications

The LEON3 [7] is a configurable processor written in VHDL. The advantage of the availability of its source helps to make modifications and explore new concepts. The LEON3 processor itself needs no introduction; we will only discuss our modifications which we made in LEON3. Figure 2 shows an abstract block diagram of our modifications. To explore the concept of reconfigurable acceleration we integrated our eFPGA soft IP in two ways with the processor.

On the left hand side of figure 2 the integration of eFPGA inside the processor pipeline is shown. We integrated it like the Multiply/Divide unit. We created a new custom

instruction for eFPGA which allows executing data in reconfigurable instruction which is created depending on running application inside the eFPGA.

On the right hand side of figure 2 eFPGA integration as a coprocessor is shown. In LEON3 the co-processor interface is only partially implemented. We created the entire interface based on the SPARC V8 manual [8] based on which LEON processor is designed. According to SPARC manual the co-processor interface is similar to FPU (Floating Point Unit) but is flexible and custom dependent, so we only created subset of interface which was needed by us. This also avoids unnecessary silicon overhead and requires very few clock cycles to send and receive data. We only implemented 8 registers in the register bank and only used basic Load, Store and CPOP instructions of SPARC V8 Manual.

This setup provides us the flexibility to analyze the pros & cons of eFPGA integration as pipeline vs co-processor from both theoretical and commercial point of view. The configuration portion of eFPGAs is not discussed in this paper for simplicity reasons. There is a small configuration hardware attached with the eFPGA which loads the configuration bitstreams of eFPGA from the main memory through the AMBA bus. The configuration hardware can be controlled by the software so is very flexible and easy to use.

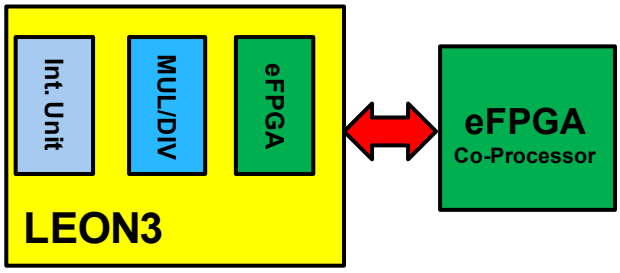

Fig. 2 : Integration of eFPGA in LEON3 processor as pipeline and as a co-processor

2.3 The profiling tool and experiment flow

Our experimentation flow is shown in fig. 3. It starts from the application written in C/C++. We profile the application with our profiling tool which we have created with the help of SimpleScalar modeling tool which analyzes the application on a MIPS like model [9]. We created several tools to analyze the outputs of SimpleScalar and display them in a user friendly HTML GUI. With this whole tools set (SimpleScalar + our custom tools) we can analyze applications in detail and extract information about each and every function, every single line of the code, execution trees and much more. The operation and results of our profiling tool will be further illustrated in next section when we will analyze applications with the complete flow of fig. 3.

With profiling we can make HW/SW partitioning (currently done manually). At this step comes the importance of programming complexity issue of reconfigurable accelerator (eFPGA in our case). For this issue we created the VHDL source for programming eFPGA both with hand coded optimal VHDL and also using Catapult™ for obtaining VHDL/Verilog directly from the ANSI C source [10]. We will demonstrate these results in next section. The HDL for eFPGA is given as a source to proprietary eFPGA CAD tool Niagara™ of Menta® to perform synthesis (using Synopsys® Design Compiler™), mapping and place & route to obtain the programming file and simulation models of eFPGA.

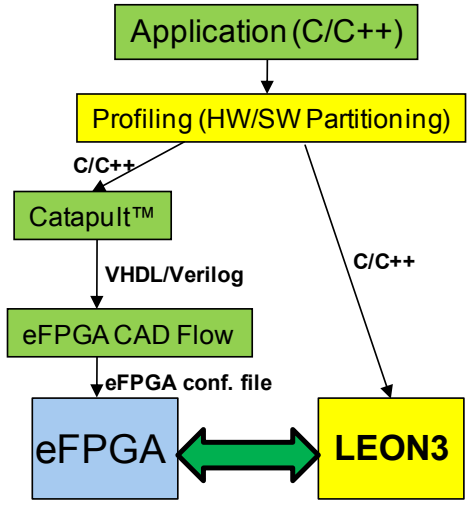

Fig. 3 : Application flow for LEON3+eFPGA implementation

3. Experiment Results

To illustrate the flow and explore the advantages in terms of overall performance and power consumption we present the results by the analysis of two cryptographic algorithms. We took them as an example application because they are practical and big enough to analyze with profiling for HW/SW partitioning. They are mostly composed of bit level operations so are suitable to analyze the benefits of bit level random logic implementation capabilities of eFPGA. We will present AES in detail with profiling illustrations to demonstrate complete flow and for DES we will directly show our achieved results.

3.1 AES

The results of profiling of AES are shown in Table 1 and Table 2. Table 1 shows statistics of all the functions in the application. For every function we can explore information like how many times it is called, how many cycles it takes on average, on total and finally how much it contributes in percentage of execution to the total application. Table 2 describes the similar statistics for the four top most computationally expensive lines. With the help of table 1 and table 2 it is very easy to analyze the whole application and see where are the computationally expensive critical areas in the application. From table 2 it is observed that line 356 of the application is consuming 23.68% of the computation time of whole application. We analyzed this line and found that it is in MixColumns function which according to table 1 is almost half (54.6%) of the total execution of AES algorithm, so by implementation of this function in eFPGA we can approximately double the computational power.

Table 1 : Profiling of AES application

Function name	Called	Avg. Cycles	Total Cycles	% of execution
main	1	179838	179838	100.0%
AES	1	147224	147224	81.9%
MixColumns	9	10904	98140	54.6%
Product	576	53	30549	17.0%
KeyExpansion	1	30396	30396	16.9%
SubByte	10	1642	16422	9.1%
ShiftRows	10	1633	16333	9.1%
AddRoundKey	11	1453	15990	8.9%
SubWord	10	396	3960	2.2%
RotWord	10	146	1466	0.8%

Table 2 : Profiling of all C instructions in AES Code

Line	CODE	CYCLES	%	Executed times	Avg. Cycles
356	C[i][j]=(Product(T[k][j],Matrix[i][k]))^ (C[i][j]);	42194	23.68%	576	73.25
442	SousKey[j][k][i]=SousKey[j][k1][i]^SousKey[j][k][i-1];	13352	7.49%	120	111.27
247	T[j][i]=sbox[(T[j][i]/16)][(T[j][i]%16);	13007	7.30%	160	81.29
388	T[j][i]=(T[j][i])^(K[j][i][Round]);	12190	6.84%	176	69.26

```
void MixColumns(short int T[4][4])
{
    short int C[4][4];
    int i,j,k;                                          STEP-4

    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            C[i][j]=0;

    for(i=0;i<4;i++) {
        for(j=0;j<4;j++)
        {
            for(k=0;k<4;k++)                            STEP-3
            {
                C[i][j]=
                (   Product(T[k][j],Matrix[i][k])   )^ (C[i][j]);
            }                       STEP-1      STEP-2
        }
    }
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            T[i][j]=C[i][j];
}
```

Fig. 4 : Implementation of complete MixColumns function in hardware in gradual steps

The C code of MixColumns function is shown in fig. 4. It can be seen that the most computational expensive line (356) is at the heart of the function and is repeatedly executed in several nested loops. We implemented this function in hardware in four gradual steps to see the overall gain in terms of area, power and speed. The four implementation steps are shown in fig. 4 and the implementation results are shown in table 3. These four steps are as follows.

In step 1 we implemented only Product function in hardware. Table 3 shows the results of this implementation for number of clock cycles if eFPGA in pipeline is used vs if eFPGA as a co-processor is used along with the eFPGA hardware resources needed (which are off course same for both cases). The AES algorithm takes 40358 clock cycles for purely software execution on LEON3 processor, by moving Product function to eFPGA, number of required clock cycles came down to 30430 for pipeline implementation and 30447 for co-processor and gave a speedup of almost 1.3X. The eFPGA took only 41 LUT6 for hand coded VHDL and 44 LUT6 for Catapult™ generated VHDL (the eFPGA which we used has LUT size of 6). In step 2 we implemented the "xor" in addition to Product function, in step 3 we implemented the inner most "for" loop and finally in step 4 we implemented the complete function. Results for all these implementations are shown in table 3 with the speedup which we achieved in all these steps and the hardware resources of eFPGA for both hand coded VHDL and Catapult™ generated VHDL.

Table 3 : Different implementations of AES with LEON3+eFPGA processing, pure soft (40358 cycles)

	Pipeline eFPGA (Cycles)	Gain X Times	Co-Processor eFPGA (Cycles)	Gain X Times
STEP-1	30430	1.326	30447	1.325
STEP-2	30722	1.313	30735	1.313
STEP-3	23752	1.699	23767	1.698
STEP-4	16244	2.484	16265	2.481

	STEP-1	STEP-2	STEP-3	STEP-4
VHDL: Hand	41-LUT6, 0FF	39-LUT6, 0FF	67-LUT6, 8FF	506-LUT6, 324FF
VHDL: Catapult	44-LUT6, 0FF	72-LUT6, 33FF	130LUT6, 78FF	819-LUT6, 604FF

From table 3 it can be observed how the gain in performance gradually increases by transferring more and more computation to eFPGA. However it is very important to note the relative increase in hardware resources. We can see that step 3 is giving a very good speedup with very small amount of eFPGA resources (only 67 LUT6 for hand coded VHDL). It can be seen that the state of the art Catapult™ is giving close results to hand coded VHDL. It is widely known that with ESL the final RTL depends a lot on the way source C/C++ is written. We have found the same issues, it can be seen that in some cases the differences between hand coded VHDL and ESL is larger than others (specially last 2 steps) because of the style of implementation. However we have found that programming through ESL is much faster and easier to verify. In case of Catapult™ we have built-in support for different levels of verification and integrated support of

ModelSim™ which made it very easy for us to write the code and quickly verify compared to our hand coded VHDL. We can check several implementation options for target HDL at a higher level, like trade offs in area for decreasing latency and increasing throughput, pipelining etc. That is relatively difficult and time consuming to do at HDL level.

Another very interesting observation in table 3 is that the speed up achieved with integrating eFPGA inside the processor pipeline and using it as a co-processor is almost same on overall application level analysis. Individually for execution there is off course a difference because with co-processor interface we have to spend some additional cycles to load data into co-processor registers and there is some further delay which is caused by co-processor controller state machines. For our case the difference is much less also because as mentioned in section 2.2 we created a very compact and fast co-processor interface custom to our needs which requires very few clock cycles for performing the data transactions between processor and eFPGA.

But overall we see a great advantage with co-processor implementation from practical and commercial point of view. Firstly with this interface type we do not need to modify the integrity of the processor (which is very critical for commercial products due to reasons of testing mainly). Most of the commercial processors already have a co-processor interface and can easily take advantage of connecting a reconfigurable accelerator with that. Secondly with co-processor interface it is more flexible and convenient to program and control the accelerator and also allows the possibility to execute things in parallel for large applications.

3.2 DES

We conducted experiments on DES algorithm in similar fashion like AES and found incredible gain of almost 10X by implementing the critical function in eFPGA. This high value of gain was achieved by only spending 95 LUT6 (hand coded VHDL). The gain in DES is much higher than AES due to the nature of the algorithm. This also highlights how advantageous it can be to have small reconfigurable accelerators in our design for applications which require use of multiple kinds of algorithms.

4. Power Consumption, Area & speed trade offs

In this section we will compare our results of section 3 of LEON3+eFPGA processing for area and power trade offs. Table 4 presents our synthesis results of LEON3 processor core (no FPU) on 65nm LPLVT (Low Power Low Voltage Threshold) process libraries of ST Microelectronics® provided by CMP [11]. For Cache memory we used 32K Instruction and 32K Data cache. We chose this value because this value of cache is usually found in processors of ARM® and MIPS®. For SRAM memory blocks for Cache memory we used 65nm High Density low leakage memory blocks of STMicroelectronics® provided by CMP [11]. From the datasheet of memory blocks we found the Static power (at 25°C) and Dynamic power (at 100MHz, with normal activity rate of 50%).

Table 4 : Area and Power consumption of LEON3 Processor with Cache memory at 100MHz

65nm LP	Area (mm^2)	Static Power 25°C (uW)	Dynamic Power 100MHz (mW)
Core	0.191	85.3	5.75
32K/32K Cache	0.4	25.63	14.9
Total	0.591	110.93	20.65

Table 5 : Power statistics of 484 LUT-6 eFPGA at 25°C, 100MHz, different toggle rates and static probabilities

65nm LP process	LVT	SVT	HVT
Static Power (mW)	1.27	0.105	0.011
DP(mW) @ (Tr-0.25,Stp-0.25)	23	22	25
DP(mW) @ (Tr-0.50,Stp-0.5)	47.6	46.6	53.8
DP(mW) @ (Tr-1.0,Stp-0.50)	95.36	93.36	107.8

Table 6 : Area and Power consumption of eFPGA

	Speedup Gain X	Area (mm^2)	Stat. Power 25°C (uW)	Dyn. Power 100MHz(mW)
STEP-1	1.3	0.128	0.00861	1.804
STEP-2	1.3	0.122	0.0082	1.716
STEP-3	1.7	0.209	0.014	2.948
STEP-4	2.48	1.578	0.106	22.264
DES	10	0.296	0.0199	4.18

Table 6 shows the area overhead of eFPGA (for hand coded VHDL) and its power consumption figures for the four steps of our AES implementation in table 3 and DES based on results of table5. Static power for SVT (Standard Voltage Threshold) and Dynamic power is estimated at normal toggle rate and static probability (Tr-0.25, Stp-0.25).

If we analyze as an example DES, we achieved almost 10X speed up by spending only 95 LUT6. From table 6 we see that we achieved it by just spending 0.296mm^2 of additional silicon due to eFPGA which only consumes approx. 4.18mW of total power. So the LEON3 processor which we found has maximum frequency of almost 185MHz at 65nm LP, with eFPGA it can be possible to get 10 times more DMIPS for DES in same frequency. If we apply Dynamic frequency scaling while executing DES we can decrease the dynamic power of LEON3 almost 10 times by only spending around 4mW of additional power overhead of eFPGA.

A very important point to consider here is that eFPGA at the moment has no power management and is completely soft core written in VHDL and is under continuous research to greatly enhance the architecture. Even at soft implementation level (which we are using for fast exploration and technology independence) and pessimistic power comparison, we can observe the promising advantages that can be achieved by adding a small eFPGA IP to the designs. Also we can observe in table 5 & 6 that our eFPGA has a very low static power.

5. Conclusions & Future works

We explored the advantages of having small embedded FPGAs (eFPGAs) as a reconfigurable accelerator in LEON3 processor as an example to improve performance and reduction in power consumption. We presented a very brief overview of some of the prominent similar approaches from academics and industry done in the past in the introduction section. We provided our viewpoint to these approaches, their potentials and possible reasons of not finding wide acceptance in the industry on the bases of our industrial experiences and study. We proposed our differentiation from those approaches which led to the motivation of this work.

Use of reconfigurable accelerators raises major questions like silicon overheads and programming complexity. We considered both of these issues in our work. We presented with our experiments that overhead of eFPGA is small because we only use it for accelerating small portions of code which can give us overall benefits in terms of performance enhancement and power consumption compared to silicon overhead. We used state of the art Catapult® to explore the use of ESL for programming the eFPGA directly in ANSI C/C++ compared to hand coded HDL and discovered the advantages of ESL which allow us to develop and verify our task much faster than HDL. We found that ESL tools are improving fast and their results are getting closer to hand coded HDL.

To keep the practical scenarios completely in consideration we analyzed the use of eFPGA as an accelerator in LEON3 in two ways, inside the processor pipeline and as a co-processor. We found that in overall co-processor interface provides greater benefits because of its flexibility and removing the need of any change in processor design and the compiler. Most of the commercial processors already have a co-processor interface and they can easily take advantage of connecting an eFPGA IP to their design without any modification to their main architecture which is in many cases very expensive and unfeasible mainly because of the fabrication and verification issues. Such issues also have been a reason of commercial failures of many reconfigurable solutions in the past.

We presented the complete idea with the help of a practical example of AES algorithm. We analyzed the application for HW/SW partitioning with the profiling tool which we have created for our research. We investigated the speed up advantages by transferring the computation to eFPGA in gradual steps along with the hardware resources needed for that. We analyzed our results in terms of overall gains in terms of area, power and speed up.

Looking towards the future we see great advantages that can be achieved using eFPGAs in SoCs which gives us inspiration about the challenges for our future research. The advantages of eFPGAs will only be meaningful if they provide high logic density with very low power consumption. This will allow us to put relatively more logic in eFPGA which is essential to give more overall benefit if there is a longer communication delay between processor and eFPGA in case of buses (like AMBA). We have analyzed the eFPGA as an accelerator in two ways in this work. We will investigate it now connected with a bus, like AMBA to provide flexibility for wide range of applications like also I/Os etc. The balancing figure between overall gain and loss is very challenging. We saw in table 3 that the logic resources can dramatically change depending on what we want to implement. AES was one example; the results can be different for different applications. Sometimes they can be much better and sometimes may be less like we saw in case of DES there was enormous speedup with very low logic resources. So a high logic density of eFPGA will improve the chances of overall gain for wider range of applications.

For increasing the logic density we need to make innovations both at architectural level and also implementation level like ARM® (www.arm.com) which uses special custom cells to improve performance. Some nice guidelines are also obtained from academic research for importance of hard macro blocks [12] and use of custom cells to improve logic density of eFPGAs compared to pure soft core [13]. We will explore the use of custom cells for eFPGA for transforming it from purely soft core to more custom and targeted to special technology to achieve optimal results which are closer to industrial requirements and also investigate how to make it technology independent with custom cells also for further flexibility. We will explore the addition of hard macro blocks like Memories, DSP blocks, shift registers and even small processors. Having coarse grained eFPGA will increase the benefits compared to fine grained, the range of applications in which eFPGA can give advantages will significantly increase. By having more coarse grained eFPGA will help reduce the configuration resources and hence configuration time, the cost and most importantly the gap between ASIC & FPGA.

6. References

[1] Syed Zahid Ahmed, Michael Fernandez, Gilles Sassatelli, Lionel Torres, “eFPGA architecture explorations: CAD & Silicon analysis of beyond 90nm technologies to investigate new dimensions of future innovations”, ReCoSoC’08, July 9-11, Barcelona, pp. 17-24

[2] www.arc.com

[3] www.tensilica.com

[4] F. Campi et al, “A Reconfigurable Processor for Embedded Applications”, ISSCC 2003

[5] J. Hauser et al, “Garp: a MIPS processor with a reconfigurable coprocessor”, FCCM97

[6] S.C. Goldstein et al, “A reconfigurable Architecture and Compiler”, IEEE Computer, 2000

[7] LEON3, Gaisler research www.gaisler.com

[8] Sparc V8 manual : www. sparc.com/standards/V8.pdf

[9] SimpleScaler : www.simplescalar.com

[10] www.mentor.com/products/esl/

[11] CMP (Circuits Multi-Projects) http://cmp.imag.fr

[12] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs" IEEE Transactions on CAD of Integrated Circuits and Systems, Vol. 26, NO. 2, FEBRUARY 2007, pp. 203 - 215

[13] V. Aken'Ova, G. Lemieux, R. Saleh, “An Improved Soft eFPGA Design and Implementation Strategy”, Custom Integrated Circuits Conference, San Jose, California, pp. 179-182, September 2005.