International conference on Advanced Computing and Communication, May 3-4, 2010, Kanjirapally, Kerala, Page number: 206-210

DESIGN OF AN APPLICATION SPECIFIC INSTRUCTION SET PROCESSOR USING LISA

Umakanta Nanda
Electronics and Communication Engineering
National Institute of Technology
Rourkela, India
uk_nanda@yahoo.co.in

Kamalakanta Mahapatra
Electronics and Communication Engineering
National Institute of Technology
Rourkela, India
kmaha2@rediffmail.com

Abstract—**A Digital Signal Processor with specific instruction sets and meant for a specific application is called as Application Specific Instruction set Processor(ASIP). To design an ASIP many approaches are available. However optimization of an ASIP becomes handy if it is designed in a higher level of abstraction that is higher than Register Transfer Level (RTL). Application Description Languages (ADLs) are becoming popular recently because of its quick and optimal design convergence achievement capability during the design of ASIPs. Several stages are required to design a processor which are architecture design implementation, software development, instruction and system verification. Verification of such ASIPs at various design stages is a tedious job to do. This paper presents the architecture description of a simple DSP processor using ADL based instruction set description. The design process is more consistent after allowing maximum flexibility here. Furthermore, it enables the design process in both instruction and cycle accurate modes. The design process of a three stage pipelined FIR Filter processor is demonstrated as a case study. Further optimization can be done with respect to resources by changing the LISA code written in CoWare platform.**

***Keywords**- LISA, ASIP, RTL, Pipelining, FIR filter, HDL, CoWare, Profiling*

I. INTRODUCTION

Recently Application Specific Digital Signal Processors are considered to be an important member in the processor family because of its flexibility and portability. The flexibility of these processors can be achieved by many ADLs [1-2] like LISA, EXPRESSION, MIMOLA etc. Different phases of design of the processor are distributed among different designers in their respective fields.

The authors gratefully acknowledge the financial support from Ministry of Communication and Information Technology, Government Of India for conducting this research. We also acknowledge the technical support of the entire team of CoWare.

There should be some type of communication between the groups of design engineers or between the phases of the design. Out of the above languages, LISA [3-4] is preferred in most cases because of it’s software development and HDL generation capability.

VHDL and Verilog languages are widely used to design and simulate a processor keeping in mind that it is to be implemented finally for IC fabrication. But these models can not be used for architecture exploration and optimization especially to design cycle based or instruction level processor simulation as the hardware implementation details are very high which are not required for performance evaluation, cycle based simulation and software verification[5-6].

In this paper we have implemented the architecture of an Embedded DSP processor using LISA [7] where the description for each instruction of the instruction set (of that specific architecture) is described properly in CoWare platform. A brief description of LISA is presented in the next section.

II. LISA

Language for Instruction Set Architecture is very much helpful to reduce the gap between the traditional design of a processor using VHDL or Verilog and instruction set languages for architecture exploration. The syntax of this language is having so much flexibility to describe the processor (RISC, VLIW, DSP, ASIPs, Special purpose co processors) instruction set which includes complex pipelining.

Generally the processor model that include LISA consists of two sections. Those are Resource and Operation section [6]. Instruction resource is a register that is usually referred as the instruction register. But the instruction resource can be a memory location, an input pin array or a concatenation of a multiple storage elements. Operation section describe the complete transition function of the processor including pipelining stages such as fetch, decode, execute and write back.

This section generally consists of three sub sections. Those are behavior, syntax and coding. Behavior describes the transition functions of the processor. Coding section depicts the binary image of the instruction word and the syntax section indicates the syntax of that particular instruction in assembly programs.

This language is more suitable with the processor designer tool called CoWare [5] for its advanced and flexible features such as,

- Automatic generation of synthesizable RTL with both control and datapath.
- Accurate profiling capabilities for high speed instruction set simulator.
- Compatible with extensively used synthesis tool like SYNOPSYS and physical design tool like MAGMA.

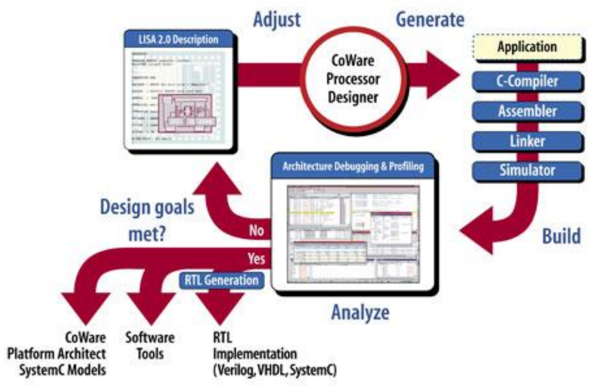

Adjust
Generate
LISA 2.0 Description
CoWare Processor Designer
Application
C-Compiler
Assembler
Linker
Simulator
Build
Architecture Debugging & Profiling
Design goals met?
No
Yes
RTL Generation
Analyze
CoWare Platform Architect SystemC Models
Software Tools
RTL Implementation (Verilog, VHDL, SystemC)

Figure 1. Design tool flow of CoWare

- Software development tool generation like assembler, linker, debugger, C- compiler.
- Integrated profiling [5,6] helps to optimize instructions for the target architecture.
- Enables the design team to develop flexible and reusable ASIPs rapidly.

The design flow of an ASIP is shown in the figure-1.

III. ARCHITECTURE DESIGN

Two fields are used for the architecture design. A high level language first describes the architecture. However for implementation purpose hardware description languages [3] are used to model the underlying hardware as shown in the fig.2. It is an advantage to combine both the development process and the HDL description. Here the LISA compiler can generate both the of these.

After design exploration and application design the target architecture needs to be implemented which is discussed in next section of this paper.

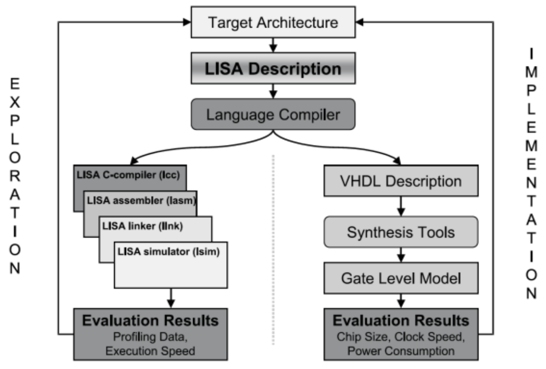


Figure 2. Exploration and implementation

IV. ARCHITECTURE IMPLEMENTATION

The LISA compiler should derive all the necessary information from the given LISA description since the generated HDL model does not have any predefined components. Then the generated HDL model can be compared to the LISA model components as shown in the figure below.

- LISA memory model derives the memory configuration which summarizes the registers and the memory sets.

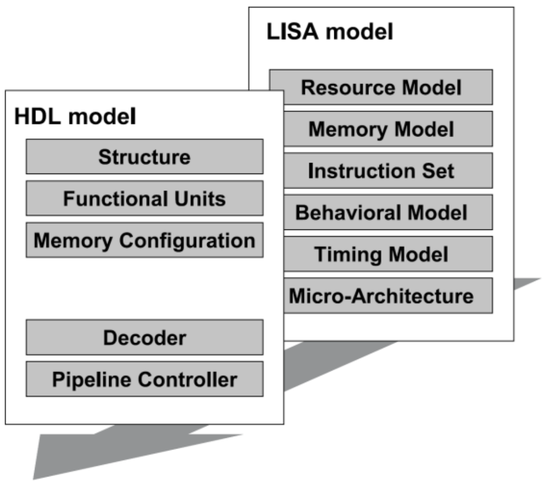


Figure 3. Comparision of HDL And LISA model

- Resource models [3] gives the idea about the structure of the architecture such as pipeline stages and pipeline registers.
- Functional units are either generated as empty frames or with fully functionality depending on the HDL language used.
- Coding information in the instruction set model and the timing model results the decoders.
- Pipeline controller is also generated from the above.

The designer will have full control over the generated HDL model with all its components. The generated HDL model can be analyzed with respect to power, area and time constraints and the optimized HDL model can be replaced with the handwritten HDL code written by the experienced designers which will be done in future work.

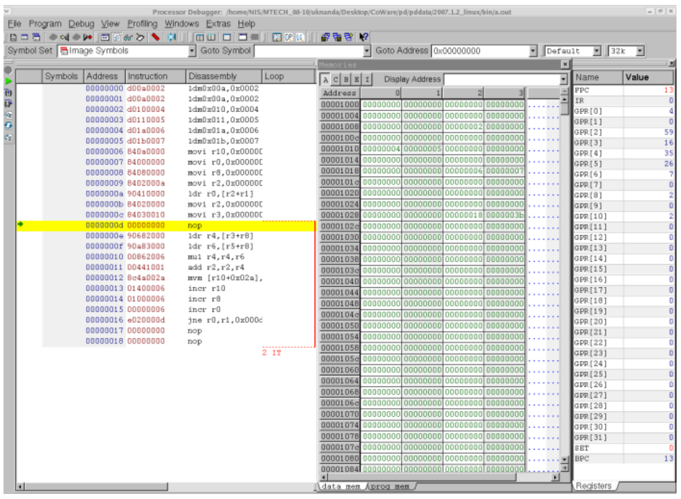
V. IMPLEMENTATION RESULT

A simple FIR filter with three stage pipelining is implemented here with the help of LISA in Coware platform. Then the resource section of this model has been optimized. A major decrease in total architecture design time can be seen, as the LISA model results from the design exploration phase.

The software development tool suit includes assembler, linker and simulator as well as a graphical debugger frontend. The tools are the enhanced version of those tools used for architecture exploration. The enhancements for the software simulate the ability to graphically visualize the debugging process of the application under test. The LISA debugger frontend is a generic graphical user interface for the generated LISA simulator as shown in the figure.

It visualizes the internal state of simulation process. Here the C source code, the disassembly of the application as well as all the configured memories and registers (pipeline) are displayed. In this frontend all contents can be changed at the run time of the application. Tools like assembler and linker can be enhanced in functionality as well. More than 30 assembler directives, labels and symbols are supported by the assembler.

The processor debugger provides extensive hardware and software profiling capabilities. Operation profiling gives us the information about Calls/Total which shows the proportion of operation executions for a specific operation to all executed operations.

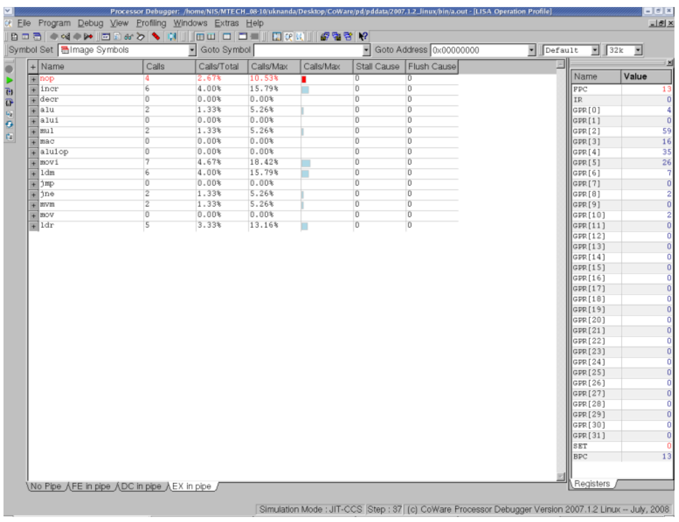

Processor Debugger: /home/NIS/MTECH_08-10/uknanda/Desktop/CoWare/pd/pddata/2007.1.2_linux/bin/a.out
File Program Debug View Profiling Windows Extras Help
Symbol Set: Image Symbols
Goto Symbol
Goto Address: 0x00000000
Default
32k
Memories
A C B E I
Display Address
2 IT
data mem
prog mem
Registers

```
Symbols  Address   Instruction  Disassembly              Loop
         00000000  d00a0002     ldm0x00a,0x0002
         00000001  d00a0002     ldm0x00a,0x0002
         00000002  d0100004     ldm0x010,0x0004
         00000003  d0110005     ldm0x011,0x0005
         00000004  d01a0006     ldm0x01a,0x0006
         00000005  d01b0007     ldm0x01b,0x0007
         00000006  840a0000     movi r10,0x0000C
         00000007  84000000     movi r0,0x00000C
         00000008  84080000     movi r8,0x00000C
         00000009  8402000a     movi r2,0x00000C
         0000000a  90410000     ldr r0,[r2+r1]
         0000000b  84020000     movi r2,0x00000C
         0000000c  84030010     movi r3,0x00000C
         0000000d  00000000     nop
         0000000e  90682000     ldr r4,[r3+r8]
         0000000f  90a83000     ldr r6,[r5+r8]
         00000010  00862006     mul r4,r4,r6
         00000011  00441001     add r2,r2,r4
         00000012  8c4a002a     mvm [r10+0x02a],
         00000013  01400006     incr r10
         00000014  01000006     incr r8
         00000015  00000006     incr r0
         00000016  e020000d     jne r0,r1,0x000d
         00000017  00000000     nop
         00000018  00000000     nop
```

```
Address   0         1         2         3
00001000  00000000  00000000  00000000  00000000
00001004  00000000  00000000  00000000  00000000
00001008  00000000  00000000  00000002  00000000
0000100c  00000000  00000000  00000000  00000000
00001010  00000004  00000005  00000000  00000000
00001014  00000000  00000000  00000000  00000000
00001018  00000000  00000000  00000006  00000007
0000101c  00000000  00000000  00000000  00000000
00001020  00000000  00000000  00000000  00000000
00001024  00000000  00000000  00000000  00000000
00001028  00000000  00000000  00000018  0000003b
0000102c  00000000  00000000  00000000  00000000
00001030  00000000  00000000  00000000  00000000
00001034  00000000  00000000  00000000  00000000
00001038  00000000  00000000  00000000  00000000
0000103c  00000000  00000000  00000000  00000000
00001040  00000000  00000000  00000000  00000000
00001044  00000000  00000000  00000000  00000000
00001048  00000000  00000000  00000000  00000000
0000104c  00000000  00000000  00000000  00000000
00001050  00000000  00000000  00000000  00000000
00001054  00000000  00000000  00000000  00000000
00001058  00000000  00000000  00000000  00000000
0000105c  00000000  00000000  00000000  00000000
00001060  00000000  00000000  00000000  00000000
00001064  00000000  00000000  00000000  00000000
00001068  00000000  00000000  00000000  00000000
0000106c  00000000  00000000  00000000  00000000
00001070  00000000  00000000  00000000  00000000
00001074  00000000  00000000  00000000  00000000
00001078  00000000  00000000  00000000  00000000
0000107c  00000000  00000000  00000000  00000000
00001080  00000000  00000000  00000000  00000000
00001084  00000000  00000000  00000000  00000000
```

Name	Value
FPC	13
IR	0
GPR[0]	4
GPR[1]	0
GPR[2]	59
GPR[3]	16
GPR[4]	35
GPR[5]	26
GPR[6]	7
GPR[7]	0
GPR[8]	2
GPR[9]	0
GPR[10]	2
GPR[11]	0
GPR[12]	0
GPR[13]	0
GPR[14]	0
GPR[15]	0
GPR[16]	0
GPR[17]	0
GPR[18]	0
GPR[19]	0
GPR[20]	0
GPR[21]	0
GPR[22]	0
GPR[23]	0
GPR[24]	0
GPR[25]	0
GPR[26]	0
GPR[27]	0
GPR[28]	0
GPR[29]	0
GPR[30]	0
GPR[31]	0
SET	0
BPC	13

Figure 4. LISA debugger window


Processor Debugger: /home/NIS/MTECH_08-10/uknanda/Desktop/CoWare/pd/pddata/2007.1.2_linux/bin/a.out - [LISA Operation Profile]
File Program Debug View Profiling Windows Extras Help
Symbol Set: Image Symbols
Goto Symbol
Goto Address: 0x00000000
Default
32k
No Pipe
FE in pipe
DC in pipe
EX in pipe
Registers

Name	Calls	Calls/Total	Calls/Max	Calls/Max	Stall Cause	Flush Cause
nop	4	2.67%	10.53%	🟥	0	0
incr	6	4.00%	15.79%	▮	0	0
decr	0	0.00%	0.00%		0	0
alu	2	1.33%	5.26%	▮	0	0
alui	0	0.00%	0.00%		0	0
mul	2	1.33%	5.26%	▮	0	0
mac	0	0.00%	0.00%		0	0
alu1op	0	0.00%	0.00%		0	0
movi	7	4.67%	18.42%	▮	0	0
ldm	6	4.00%	15.79%	▮	0	0
jmp	0	0.00%	0.00%		0	0
jne	2	1.33%	5.26%	▮	0	0
mvm	2	1.33%	5.26%	▮	0	0
mov	0	0.00%	0.00%		0	0
ldr	5	3.33%	13.16%	▮	0	0

Name	Value
FPC	13
IR	0
GPR[0]	4
GPR[1]	0
GPR[2]	59
GPR[3]	16
GPR[4]	35
GPR[5]	26
GPR[6]	7
GPR[7]	0
GPR[8]	2
GPR[9]	0
GPR[10]	2
GPR[11]	0
GPR[12]	0
GPR[13]	0
GPR[14]	0
GPR[15]	0
GPR[16]	0
GPR[17]	0
GPR[18]	0
GPR[19]	0
GPR[20]	0
GPR[21]	0
GPR[22]	0
GPR[23]	0
GPR[24]	0
GPR[25]	0
GPR[26]	0
GPR[27]	0
GPR[28]	0
GPR[29]	0
GPR[30]	0
GPR[31]	0
SET	0
BPC	13

```
Simulation Mode : JIT-CCS  Step : 37  (c) CoWare Processor Debugger Version 2007.1.2 Linux -- July, 2008
```

Figure 5. Operation profiling window

$$\frac{Calls}{Total} = \frac{Number\ \ of\ specific\ \ operation\ \ executions}{Number\ \ of\ all\ operation\ \ executions}$$

Where all operations include Fetch, Decode and main also which has not been shown in fig. 5.

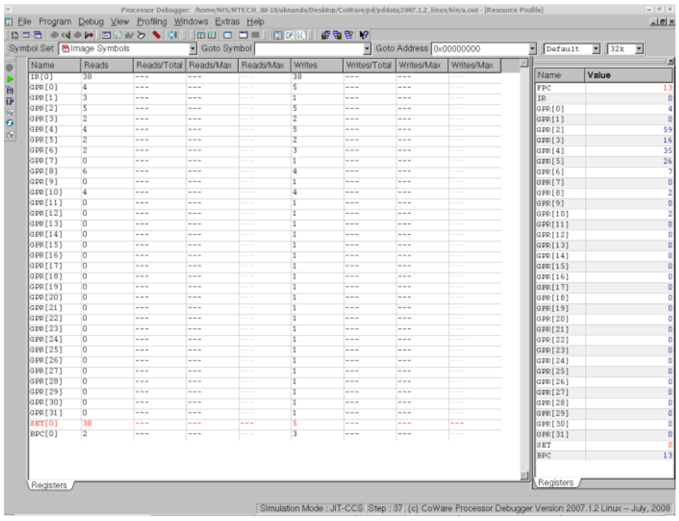
Calls/Max contains information containing the proportion of the execution of a specific operation to the execution of the LISA operation which has been executed the highest number of times.

$$\frac{Calls}{Max} = \frac{Number\ of\ specific\ operation\ executions}{Max\ Number\ of\ specific\ operation\ executions}$$

These information can be shown graphically also.

Resource profiling shows the access statistics for all resources modeled with the resource specifier as one of register, program counter and control register in the LISA model as shown in the figure 6.

Similarly memory profiling tells about the access statistics for the memories contained in the processor model. This model has the program memory range 0x0000 to 0x1111 and data memory range 0x1111 to 0Xffff.


Processor Debugger: /home/NIS/MTECH_08-10/uknanda/Desktop/CoWare/pd/pddata/2007.1.2_linux/bin/a.out - [Resource Profile]
File Program Debug View Profiling Windows Extras Help
Symbol Set
Goto Symbol
Goto Address
Registers

Symbol Set: Image Symbols | Goto Address: 0x00000000 | Default | 32k

Name	Reads	Reads/Total	Reads/Max	Reads/Max	Writes	Writes/Total	Writes/Max	Writes/Max
IR[0]	38	---	---	---	38	---	---	---
GPR[0]	4	---	---	---	5	---	---	---
GPR[1]	3	---	---	---	1	---	---	---
GPR[2]	5	---	---	---	5	---	---	---
GPR[3]	2	---	---	---	2	---	---	---
GPR[4]	4	---	---	---	5	---	---	---
GPR[5]	2	---	---	---	2	---	---	---
GPR[6]	2	---	---	---	3	---	---	---
GPR[7]	0	---	---	---	1	---	---	---
GPR[8]	6	---	---	---	4	---	---	---
GPR[9]	0	---	---	---	1	---	---	---
GPR[10]	4	---	---	---	4	---	---	---
GPR[11]	0	---	---	---	1	---	---	---
GPR[12]	0	---	---	---	1	---	---	---
GPR[13]	0	---	---	---	1	---	---	---
GPR[14]	0	---	---	---	1	---	---	---
GPR[15]	0	---	---	---	1	---	---	---
GPR[16]	0	---	---	---	1	---	---	---
GPR[17]	0	---	---	---	1	---	---	---
GPR[18]	0	---	---	---	1	---	---	---
GPR[19]	0	---	---	---	1	---	---	---
GPR[20]	0	---	---	---	1	---	---	---
GPR[21]	0	---	---	---	1	---	---	---
GPR[22]	0	---	---	---	1	---	---	---
GPR[23]	0	---	---	---	1	---	---	---
GPR[24]	0	---	---	---	1	---	---	---
GPR[25]	0	---	---	---	1	---	---	---
GPR[26]	0	---	---	---	1	---	---	---
GPR[27]	0	---	---	---	1	---	---	---
GPR[28]	0	---	---	---	1	---	---	---
GPR[29]	0	---	---	---	1	---	---	---
GPR[30]	0	---	---	---	1	---	---	---
GPR[31]	0	---	---	---	1	---	---	---
SET[0]	38	---	---	---	5	---	---	---
BPC[0]	2	---	---	---	3	---	---	---

Name	Value
FPC	13
IR	0
GPR[0]	4
GPR[1]	0
GPR[2]	59
GPR[3]	16
GPR[4]	35
GPR[5]	26
GPR[6]	7
GPR[7]	0
GPR[8]	2
GPR[9]	0
GPR[10]	2
GPR[11]	0
GPR[12]	0
GPR[13]	0
GPR[14]	0
GPR[15]	0
GPR[16]	0
GPR[17]	0
GPR[18]	0
GPR[19]	0
GPR[20]	0
GPR[21]	0
GPR[22]	0
GPR[23]	0
GPR[24]	0
GPR[25]	0
GPR[26]	0
GPR[27]	0
GPR[28]	0
GPR[29]	0
GPR[30]	0
GPR[31]	0
SET	0
BPC	13

Simulation Mode : JIT-CCS | Step : 37 | (c) CoWare Processor Debugger Version 2007.1.2 Linux – July, 2008

Figure 6. General Purpose Register window

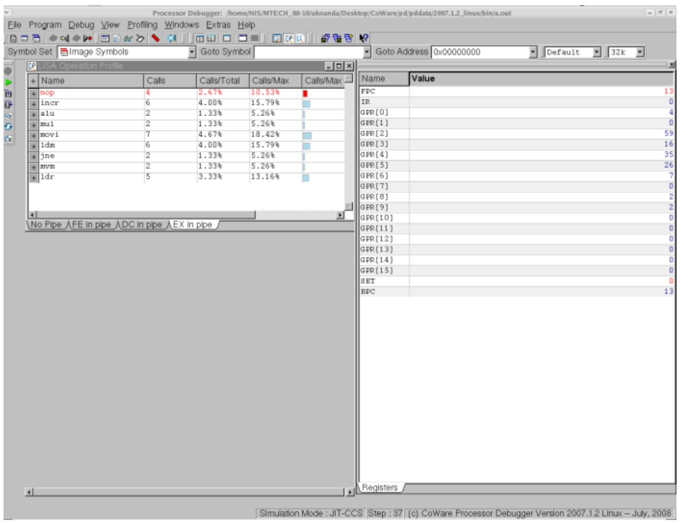
These profiling information is very much required to optimize our design. This architecture was designed on the respective abstraction level with LISA and software development tools were generated successfully.

VI. OPTIMIZED IMPLEMENTATION RESULT

Now we have optimized the FIR filter with respect to the resources we used like,

- Data and program memory
- Instruction set
- Number of general purpose registers

In the operation profiling we can see that the instructions or operations like decr, alui, mac, alu1op, jmp, sub, and, or and mov have not been called in our specific application. So writing the behavioral code for these operations is not required and the result will be unchanged.


Processor Debugger: /home/NIS/MTECH_08-10/uknanda/Desktop/CoWare/pd/pddata/2007.1.2_linux/bin/a.out
File Program Debug View Profiling Windows Extras Help
Symbol Set
Goto Symbol
Goto Address
LISA Operation Profile
No Pipe
FE in pipe
DC in pipe
EX in pipe
Registers

Symbol Set: Image Symbols | Goto Address: 0x00000000 | Default | 32k

Name	Calls	Calls/Total	Calls/Max	Calls/Max
nop	4	2.67%	10.53%	▮
incr	6	4.00%	15.79%	▮
alu	2	1.33%	5.26%	▮
mul	2	1.33%	5.26%	▮
movi	7	4.67%	18.42%	▮
ldm	6	4.00%	15.79%	▮
jne	2	1.33%	5.26%	▮
mvm	2	1.33%	5.26%	▮
ldr	5	3.33%	13.16%	▮

Name	Value
FPC	13
IR	0
GPR[0]	4
GPR[1]	0
GPR[2]	59
GPR[3]	16
GPR[4]	35
GPR[5]	26
GPR[6]	7
GPR[7]	0
GPR[8]	2
GPR[9]	2
GPR[10]	0
GPR[11]	0
GPR[12]	0
GPR[13]	0
GPR[14]	0
GPR[15]	0
SET	0
BPC	13

Simulation Mode : JIT-CCS | Step : 37 | (c) CoWare Processor Debugger Version 2007.1.2 Linux – July, 2008

Figure 7. Optimized profiling window

In the optimized model we have less space allocated for data and program memory. Program memory starts from 0x0000 to 0x0015 and Data memory starts from 0x0016 to 0x0042 reducing the area further.

To reduce the resource section further we can take 16 general purpose registers (GPR) instead of 32 which will reduce the area of our model.

VII. CONCLUSION

This method of processor design facilitated changing the model according to the instruction set provided for the specific application and limited resources. Here we implemented FIR filter architecture using LISA. Then the same model was optimized with respect to resources like data memory, program memory, instruction set and number of general purpose registers.

Our future work will focus on the generation of the RTL file from which we can get the HDL model of the same architecture. The automatic generation of pipelined functional units of the ASIP with optimization in data path and resources will be another interesting research work. Furthermore we can analyze and compare the area, power and timing issues of our generated HDL model and the hand written model for the same architecture.

REFERENCES

[1] Anupam Chattopadhaya, Arnab Sinha, Dandian Zhang, Rainer Leupers, Gerd Ascheid, Henrich Meyr, ”Integrated Verification approach during ADL driven processor design”, Microelectronics journal 40(2009), page 1111-1123.

[2] Welhua Sheng, Jianjiang Ceng, Manuel Hohenauer, Hanno Scharwachter, Rainer Leupers, Henrich Meyr, Institute for Integrated systems, Achen, Germany, ”A novel approach for fexible and consistent ADL driven ASIP design”, DAC’04, June 7-11, 2004, San Diego, California, USA.

[3] Andreas Hoffman, Member IEEE, Tim Kogel, Achim Nohl, Gunnar Braun, Oliver Schliebush, Oliver Wahlen, Andreas Wieferink and Henrich Meyr, Fello, IEEE, “A novel methodology for the design of application specific instruction set processors (ASIPs) using a machine description language”. IEEE transaction on Computer Aided Design of integrated circuits and systems, vol-20, number 11, Nov.-2001.

[4] O. Schliebusch, A. Chattopadhayay, E M Witte, D Kammler, G. Ascheid, R Leupers, H Meyr, ”Optimization techniques for ADL driven RTL processor synthesis” in IEEE workshop on rapid system prototyping(RSP), Montreal, Canada, June 2005.

[5] CoWare, The ESL design Leader reference manuals, Product version V2007.1.2, June-2008.

[6] CoWare,inc,http://www.coware.com.

[7] U K Nanda, K K Mahapatra, ”Design of a pipelined FIR filter using Application Description Language” , National Conference on Wireless Communication and VLSI Design-2010, GEC, Gwalior, March 27-28, 2010.