PROCEEDINGS OF SPIE

Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks V

Harold H. Szu
Jack Agee
Editors

10–13 April 2007
Orlando, Florida, USA

Volume 6576

The International Society for Optical Engineering

FPGA Wavelet Processor Design using Language for Instruction-set Architectures (LISA)

Uwe Meyer-Bäse[a], Alonzo Vera[b], Suhasini Rao[a], Karl Lenk[a], and Marios Pattichis[b]

[a]FAMU-FSU, ECE Dept., 2525 Pottsdamer Street, Tallahassee, FL USA-32310;
[b]Department of Electrical and Computer Engineering The University of New Mexico Albuquerque, NM 87131

ABSTRACT

The design of an microprocessor is a long, tedious, and error-prone task consisting of typically three design phases: architecture exploration, software design (assembler, linker, loader, profiler), architecture implementation (RTL generation for FPGA or cell-based ASIC) and verification. The Language for instruction-set architectures (LISA) allows to model a microprocessor not only from instruction-set but also from architecture description including pipelining behavior that allows a design and development tool consistency over all levels of the design.

To explore the capability of the LISA processor design platform a.k.a. CoWare Processor Designer we present in this paper three microprocessor designs that implement a 8/8 wavelet transform processor that is typically used in today's FBI fingerprint compression scheme. We have designed a 3 stage pipelined 16 bit RISC processor (NanoBlaze). Although RISC μPs are usually considered "fast" processors due to design concept like constant instruction word size, deep pipelines and many general purpose registers, it turns out that DSP operations consume essential processing time in a RISC processor. In a second step we have used design principles from programmable digital signal processor (PDSP) to improve the throughput of the DWT processor. A multiply-accumulate operation along with indirect addressing operation were the key to achieve higher throughput. A further improvement is possible with today's FPGA technology. Today's FPGAs offer a large number of embedded array multipliers and it is now feasible to design a "true" vector processor (TVP). A multiplication of two vectors can be done in just one clock cycle with our TVP, a complete scalar product in two clock cycles. Code profiling and Xilinx FPGA ISE synthesis results are provided that demonstrate the essential improvement that a TVP has compared with traditional RISC or PDSP designs.

Keywords: Microprocessor, FPGA, Wavelets, LISA

1. INTRODUCTION

A microprocessor like a finite state machine (FSM) implements algorithms in a iterative way usually slower than a direct hardware implementation of the algorithms. However, this results in a much more efficient (in terms of area not power or speed) way of using FPGA resources than a direct hardware implementation of an algorithm. Microprocessors (μPs) have become one of the most important IP blocks for FPGA vendors in recent years. Altera for instance reported that they sold 10,000 systems of the NIOS microprocessor development systems in the first 3 years alone. Xilinx reported a even larger number of "downloads" of their PicoBlaze and MicroBlaze microprocessors.[1–3]

A new generation of design tools now enables software developers to take their algorithmic expressions straight into custom VLSI hardware without using the traditional HDL design flow. These tools and associated design methodologies are classified collectively as electronic system level (ESL) design, broadly referring to system design and verification methodologies that begin at a higher level of abstraction than the current mainstream hardware description language (HDL). The Language for Instruction Set Architecture (LISA)[4,5] for instance allows us to specify an instruction or cycle accurate μP using a few LISA operations, then make architecture exploration using tool generator and profiler (see Fig. 1) and finally determine speed/size/power parameter via automatically synthesized VHDL or Verilog code. ESL tools have been around for a while, and many perceive that these tools are predominantly focused on ASIC design flows. But with ASIC mask charges of $1.5 million in 90 nm and $4 million in 65 nm technology (according to J. Donovan Vice president at Gartner Dataquest)[6] the

Independent Component Analyses, Wavelets, Unsupervised Nano-Biomimetic Sensors, and Neural Networks V, edited by Harold H. Szu, Jack Agee, Proc. of SPIE Vol. 6576, 65760U, (2007) · 0277-786X/07/$18 · doi: 10.1117/12.719020

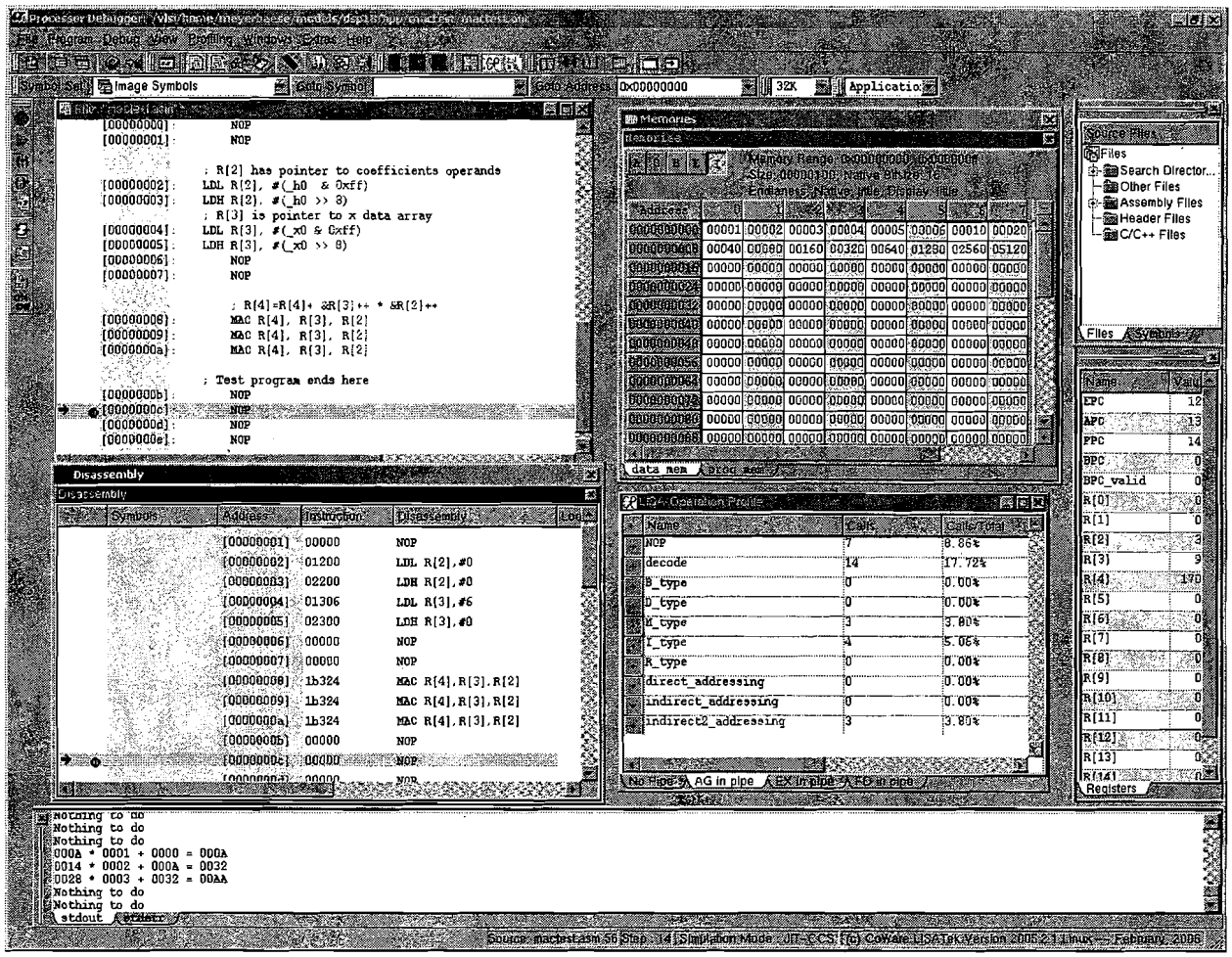

Figure 1. LISA Development tools. (left) Disassembler. (center) Memory monitor and Pipeline profiles. (right) Source files and register window (©2007 Springer[9]).

number of designs using FPGAs is rapidly increasing. In reality, an increasing number of ESL tool providers (e.g. Celoxica, Codetronix, CoWare, Binachip, Impulse Accelerated, Mimosys, etc.) are focusing on programmable logic.[7,8]

Today the majority of microprocessors are employed in embedded systems. This number is not surprising because a typical home today may have a laptop/PCs with a high performance microprocessor but probably dozens of embedded systems including electronic entertainment, household, and telecom devices, each of them equipped with one or more embedded processors. A modern car typically has more than 50 microprocessors. Embedded processors are most often developed by relatively small teams within short time-to-market requirements and the processor design automation is clearly a very important issue. Once a model of a new processor is available, existing hardware synthesis tools enable the path to custom VLSI implementation. However embedded processor designs typically begin at a much higher abstraction level, even far beyond an instruction set architecture (ISA) and involves several architecture exploration cycles before the optimum hardware/software partitioning is found. It turns out, that this requires a number of tools for software development and profiling. These are normally written manually - a major source of cost and inefficiency in embedded processor design so far. The CoWare Processor Designer formerly known as LISAtek processor design platform (LPDP) originally developed at RWTH Aachen[4,5] and now a product of CoWare Inc. addresses these issues in a highly innovative and satisfactory manner, see Fig. 1. The LISA language supports profiling-based stepwise refinement of processor models down

to cycle-accurate and even VHDL or Verilog RTL synthesis models for fast custom VLSI implementation. In an elegant way, it avoids model inconsistencies otherwise inevitable in traditional design flows. Microprocessor from simple RISC to highly complex VLIW processor have been described and successfully implemented using LPDP for FPGAs and cell-based ASICs.

CoWare provides 14 different example/starting-point models. This include 7 tutorial models that are used as part of CoWare training material. Some have multiple versions that have over 10 different designs as seen in the `QSIP_X` model. 4 starting point models are provided and used as skeletons for starting a new architecture. 3 different IP models for classic architectures are also included. All models are instruction accurate and most of the models are Harvard type RISC models that are also cycle accurate. Pipeline stages vary from 3 to 5. Provided are all types of modern processor from simple RISC (QSIP), over PDSP like `LT_DSP_32p3` to VLIW `LT_VLIW_32p4` to special processors like a 16 to 4096-point FFT processor `LT_FFT_48p3`.

2. LISA 18-BIT ISA RISC PROCESSOR

Xilinx offers a 32-bit MicroBlaze and a 8-bit PicoBlaze RISC processor but no processor with 16 or 24 bits typical for DSP algorithms is offered. In fact BDTI's "Pocket Guide to Processors for DSP"[10] shows that all commercial sucessful fixed-point PDSPs use 16 or 24 bits. Let us create in the following such a 16-bit RISC machine with the LPDP. Since a 16-bit processor fits in the middle between Micro- and PicoBlaze we will call our RISC processor NanoBlaze[†].

For a FPGA design we can start with the 3-pipeline RISC tutorial design of the LISA 2.0 `QSIP_12` model and extend the ISA to make it more useful for the FPGA design. The BlockRAM in Xilinx FPGAs are 18 bits wide and the instruction words should therefore also be a multiple of 18 bits. There is no benefit when instruction words less than 18 bits are used in the BlockRAM. A more careful choice of the instruction word width needs to be done for a cell-based ASIC design. The byte wide access in the `QSIP` model should be changed to a flat 18-bit for both, instructions and data. Changes should then be included in the instruction counter, memory configuration `*.cmd` file, `step_cycle`, and the data memory instruction `LDL`,`LDH` and `LDR`. The following listing shows the supported instruction of the designed NanoBlaze.

- Arithemtic/Logic unit (ALU) instructions:
  - `ADD` Three operands add operation with 2 source operands and a third destination operand.
  - `MUL` Three operands multiply operation with 2 source operands and a third destination operand. Only the lower 16-bit of the product are preserved.
- Data move instructions:
  - `LDL` Load the lower 8-bit of the data word with a constant value
  - `LDH` Load the upper 8-bit of the data word with a constant value
  - `LDR` Load register from memory. The memory location can be specified explicitly as constant or indirect via a general purpose register.
  - `STR` Store register content to memory. The memory location can be specified explicit or indirect via a general purpose register.
- Program control instructions:
  - `BC` The condition branch checks a (loop) register for zero and not zero
  - `B` Is an unconditional branch
  - `BDS` The delay branch is a condition `BC` except that the next instruction after the `BDS` instruction is also executed.

[†]Micro- and PicoBlaze are (non-registered) trademarks from Xilinx according to Xilinx legal information at `http://www.xilinx.com/legal.htm` but no claim is made for NanoBlaze at time of writing this paper.

Table 1. (a) NanoBlaze and (b) DSP18 synthesis result for the Xilinx device XC3S1000-4ft256 according to Xilinx ISE mapping and timing analysis.

(a)

Parameter	NanoBlaze with CLB-based RAM	NanoBlaze with BlockRAM
Slices	1896	1893
4-input LUT	3443	3602
Multiplier	1	1
BlockRAMs	0	2
Total gates	32,986	162,471
Clock period	13.293 ns	13.538 ns
$F_{\max}$	75.2 MHz	73.9 MHz

(b)

Parameter	DSP18 with CLB-based RAM	DSP18 with BlockRAM
Slices	3145	2679
4-input LUT	6053	5183
Multiplier	2	2
BlockRAMs	0	2
Total gates	81,509	177,203
Clock period	25.542 ns	19.565 ns
$F_{\max}$	39.15 MHz	51.11 MHz

The basic instruction set of the DWT RISC processor consists of 9 instructions that were designed using 28 LISA operations. The instruction coding of the instruction in the "execution" pipeline stage can be found in the literature.[9]

The NanoBlaze processor can now be synthesized and implemented in an FPGA. The synthesis results are shown in Table 1 for both CLB- and BlockRAM-based memory.

If we now use the RISC processor to implement a length-8 DWT processor as shown in Fig. 2, we need two length-8 filter $g[n]$ and $h[n]$ and for each output sample pair 16 multiply and 14 add operations are necessary. For 100 sample with a output downsampling by 2 the arithmetic requirements for the DWT filter band would therefore $8 \times 100 = 800$ multiplications and $7 \times 100 = 700$ additions. From the instruction profile shown in the second column of Fig. 6(a) we see that the number of multiplication is in fact 800, however the number of add instructions was more than 4 times higher as expected. This is due to the fact that the register updates for the memory access are also computed with the general purpose ALU.

In addition to the large number of add operations to update the memory register pointer the 1600 `LDR` load operations were performed. This can be substantially improved by using for PDSP[10,12,13] typical MAC operation with indirect memory access and auto-increment, discussed next.

3. LISA PROGRAMMABLE DIGITAL SIGNAL PROCESSOR (PDSP)

From the DWT processor discussed in the last section we have seen that large arithmetic count is required for updating memory pointer and the memory access itself. A single multiply accumulate instruction in NanoBlaze requires the following operations:

```
; use pointer R[2] and R[3] to load operands
LDR R[8], R[2]
LDR R[9], R[3]
; increment register pointer using R[1]=1
; multiply and add result in R[4] and avoid data hazards
ADD R[2], R[2], R[1]
MUL R[7], R[8], R[9]
ADD R[3], R[3], R[1]
ADD R[4], R[4], R[7]
```

DSP algorithm (e.g. convolution, correlation, FIR, IIR filter or fast DFTs[14,15]) typically operate on linear data arrays (i.e., vectors) and post-auto-increments or decrements in the memory pointer are therefore frequently used. In addition a fused add and multiply usually called `MAC` allows the previous 6 instructions to be combined into one single instruction, i.e.,

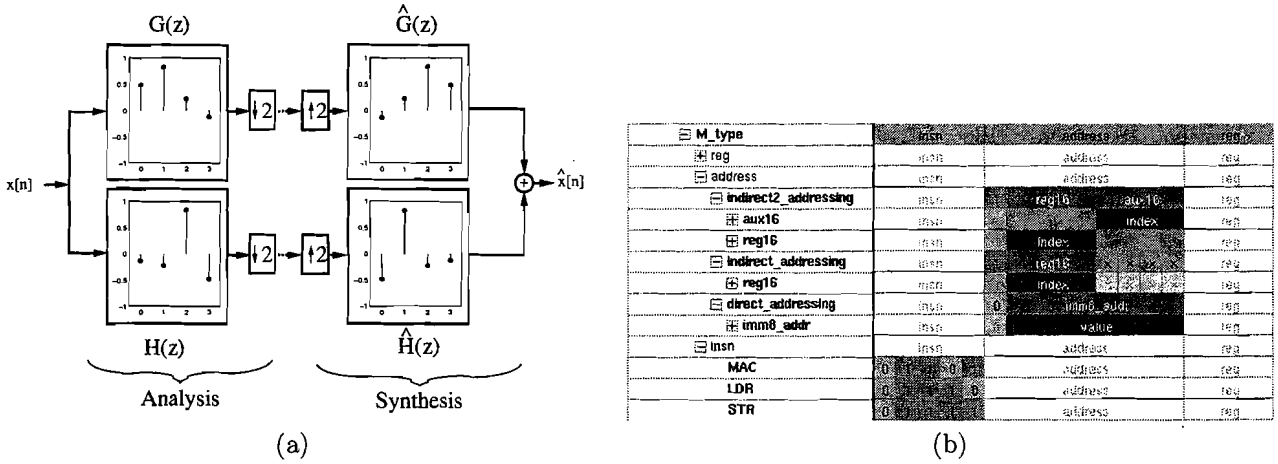

Figure 2. (a) Two-channel filter bank using Daubechies filter of length-4. (b) Programmable Digital Signal Processor (DSP18) instruction set additions (©2007 Springer[9]).

```
; load and multiply the values from pointer R[2] and R[3],
; and add the product to register R[4]
MAC R[4],R[3],R[2]
```

The additional ISA instructions added to the NanoBlaze are shown in Fig. 2(b). The addition of such a MAC operation to the instruction set requires two major modifications. First we need to provide a LISA operation that allows two indirect memory accesses. In hardware this results in a more complex address generation unit and a dual output port data memory that supports 2 reads in one clock cycle. Secondly, we need to add the LISA operation for the MAC instructions.

The LISA operation to implement the MAC instruction can be implemented as follows:

```
OPERATION MAC IN pipe.EX {  /* This LISA operation implements the instruction MAC.  */
  DECLARE                   /* It accumulates the product of two register and stores*/
    { REFERENCE address;    /* the result in a destination register.               */
      REFERENCE reg; }
  CODING { 0b01101 }
  SYNTAX { "MAC" }
  BEHAVIOR
    {
      short tmp1, tmp2, s1, s2; /* Temporary */
      short tmp_reg, res;
      tmp_reg = reg;
      s1 = (data_mem[EX.IN.ar] & (char)0xffff);
      s2 = (data_mem[EX.IN.ar1] & (char)0xffff);
      res = tmp_reg + s1 * s2;
#pragma analyze(off)
      printf ("%04X * %04X + %04X = %04X\n", s1, s2, tmp_reg, res);
#pragma analyze(on)
      reg = res;}}
```

The MAC LISA operation start with the DECLARE section that references to elements that are defined in other LISA operations. The CODING section that describes the OP code follows. The assembler syntax would be MAC and finally in the BEHAVIOR section the implementation of the instruction is shown. For 16 bits product the operands have been reduced to 8 bits multiplier and multiplicand. A more efficient way would be to use the "all fractional"

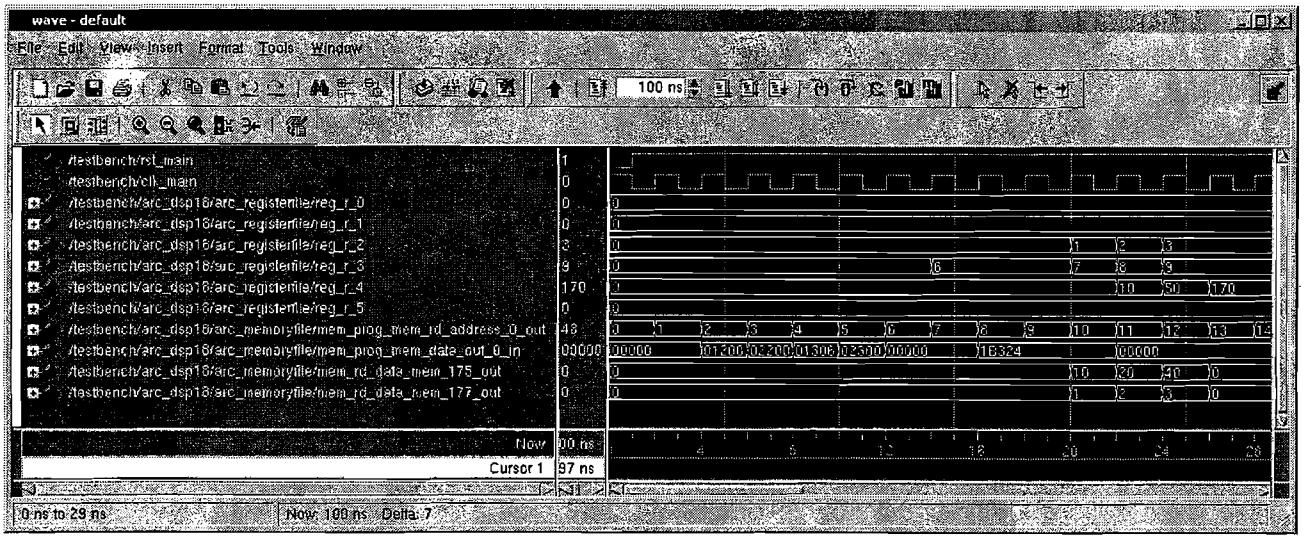

Figure 3. DSP18 testbench for MAC operation (©2007 Springer[9]).

1.15 format typically used in PDSP and using only the 16 MSBs of the product, but the testbench/simulation results would then be less intuitive. We have used an additional `printf` inside the operation to monitor the progress. While this does not change the hardware, we can monitor the output of our `MAC` instruction directly in the debugger window, see lower window in Fig. 1.

We can then go ahead and synthesize the new processor that we like to call DSP18 due to the PDSP like added features in the instruction set and perform a testbench simulation in ModelSim simulator.

To verify the functionality of the generated VHDL code we use the ModelSim simulator. LPDP generated all required HDL code (for VHDL or Verilog) and all required simulation script (i.e., ModelTech `*.do`-files). As test values we use $x = [1, 2, 3]; g = [10, 20, 40]$; and the MAC operation should progress as follows:

```
1. MAC = 1*10=10
2. MAC = 2*20=40 => 40+10=50
3. MAC = 3*40=120 => 120+50=170
```

The correct function can be seen from the ModelSim simulation from `reg_r_4` shown in Fig 3 which shows the content of register $r[4]$.

If we now write the program for the same 100-point length-8 DWT as in the last section, we will see the large impact the MAC operation has on the overall instruction count. Now the `MAC` operation with 800 is the dominant operation and much less explicit add or memory operation are required. The total instruction count improves from 5870 for the NanoBlaze to 1968 for the DSP18. The operation profile for the DWT example is shown in the third column of Figure 6(a). Since the DSP18 is larger and the addressing modes are more sophisticated than in the NanoBlaze, the overall registered performance decrease to 39 MHz when using CLB-based RAM and 51 MHz when using BlockRAM. Table 1 shows the implementation results for the two different external memory configuration.

4. LISA TRUE VECTOR PROCESSOR

General purpose CPUs could be improved in previous years by exploring instruction level parallelism (ILP), adding on-chip cache and floating-point units, speculative branch execution and improved speed etc. One particular problem that occurs now is that the logic to track dependencies between all in-flight instructions grows quadratically in the number of instructions.[16] As a result these improvements have considerably solved down

since 2002 and the use of multiple CPUs on the same die is now favored instead of increasing clock speed. This requires to write code for parallel processors, which maybe less efficient than using a vector processor to start with. Vector processors were successfully commercialized long before ILP machines and use an alternative approach to controlling multiple function units with deep pipelines. Vector processors like Cray, NEC, or Fujitsu VP100 provide high level instructions that work on vectors, i.e., a linear array of numbers. Usually vector processor are characterized by using

- a vector array with dedicated load/store unit
- functional unit that are highly pipelined
- Hazard control is minimized.
- support of vector instruction, that replace a complete loop by a single instruction

The second DSPstone[17,18] benchmark: $d[k] = a[k] \times b[k]; 0 \leq k \leq N$ for instance would be implemented in VMIPS by

```
MULV.D    V1,V2,V3,
```

i.e., multiply elements of `V2` and `V3` and put each result in `V1`. Fig. 4 shows an experimental vector processor that is a vector extension of the popular MIPS machine called VMIPS. VMIPS was introduced in 2001 with 8 vector registers each with 64 elements, 1 load/store, 5 arithmetic units, 1 lane and runs with 500 MHz. Details on VMIPS can be found in the literature.[16]

However as can be seen from Fig. 4(a), the typically implemented vector processor architecture, only looks for a programmer as a vector machine. Inside the vector processor we may typically find 8 vector registers where each vector has 32 to 1024 (Fujitsu VP100) elements but usually only one floating-point arithmetic unit for each operation. The vector multiply or add like in DSPstone benchmark 2 still requires N clock cycles (not counting the initialization). Multiple "lanes" that allow more than one floating-point operation per clock cycle are limited. In previous 30 years of vector processor history only two machines (NEC SX/5 from 1998 and Fujitsu VPP5000 introduced in year 1999) have over 10 lanes, but the quotient of register elements to lanes is still only 3% for the 512 elements per vector in the NEC SX/5 with 16 lanes.[16] The reason that typically VP do not have more than one lane comes from the fact that the floating-point units in 64 bit need large die area.

Another weakness of current vector processors is the limited use for DSP operations. In DSP we not only need a vector multiply, but instead more often an inner product computation is needed, i.e.,

$$\mathbf{X} \times \mathbf{Y} = \sum_{k=0}^{N-1} X[k] \times Y[k]. \tag{1}$$

While the multiplication can be done in a vector element-by-element parallel fashion the summation requires the addition of all products in an adder tree. This usually is not supported with vector instructions. A third operation that is not supported in most vector processors is the (cyclic) shift of the vector register elements. For instance, if an FIR application requires the vector elements $x[0] \ldots x[N-1]$ then in the next step the elements $x[1] \ldots x[N]$ are needed. A PDSP uses cyclic addressing to address this issue. In a vector processor it is usually necessary to reload the complete vector.

In an custom VLSI Application Specific Integrated Processor (ASIP) design we can therefore improve the processing by

- Adding the vector shift instructions `VSXY` and `VSXY` to our instruction set that loads two words from data memory, shift the two vector registers of the data or coefficients, and place the two new values in the first location.

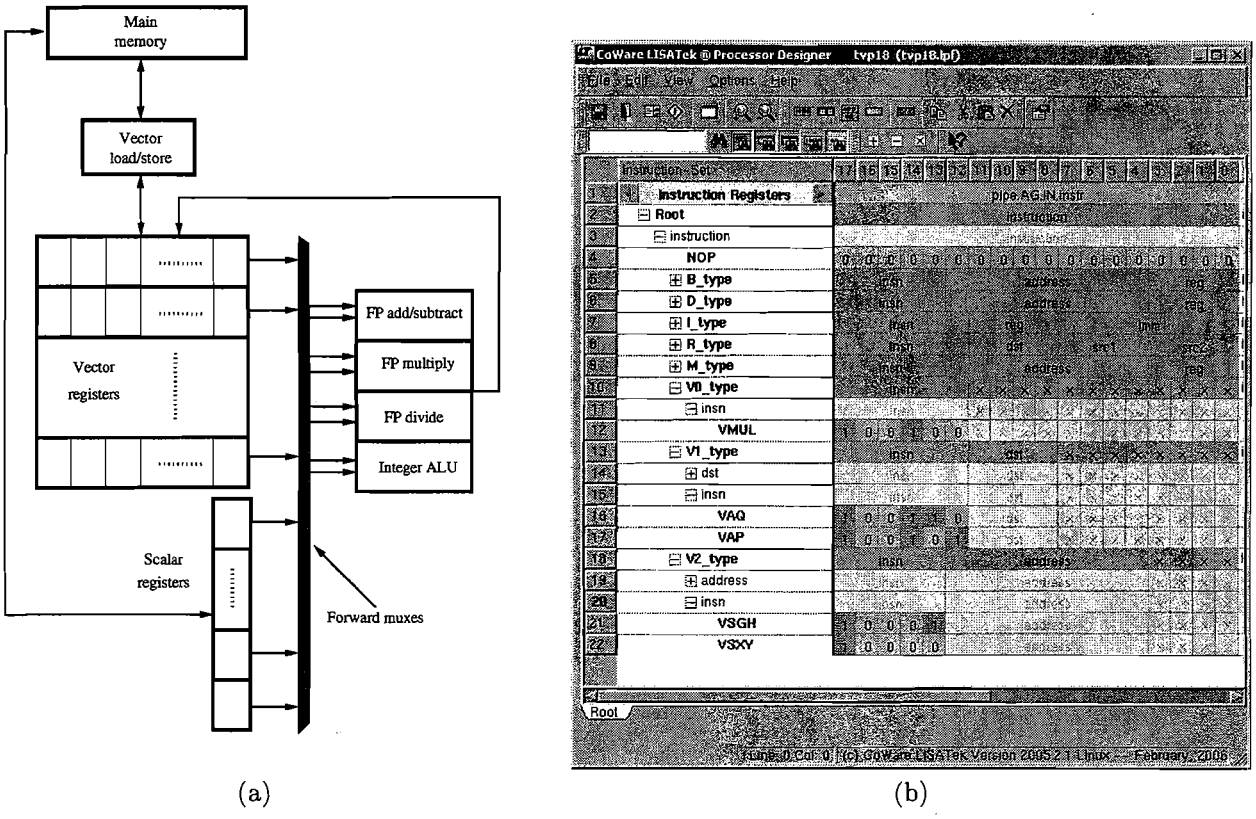

Figure 4. (a) The VMIPS vector processor. (b) The true vector processor (TVP) instruction set additions (©2007 Springer[9]).

- Since modern FPGA can have more than 512 embedded multipliers, we can implement as many multipliers as the number of vector elements contained in a vector. A `VMUL` instruction will perform 2×8 multiplications and place the products in the two product vector registers `P` and `Q`.
- Implementing (inner product) vector sum instructions `VAP` and `VAQ` that adds up all elements in a (product) register vectors.

We like to call such a machine a true vector processor (TVP) since vector operations like vector multiply are no longer translated in a sequence of single multiplies – all operations are done in parallel. For a two channel length-8 wavelet processor we would therefore require 16 embedded multipliers. A Spartan-3 device XC3S1000-4ft256 that is used in the $119 low cost Nexys Digilent University boards for instance, has 24 embedded 18×18-bit multipliers available, more than enough for our TVP.

The inner product sum maybe a concern in terms of speed since here $N-1$ "horizontal" additions are needed to be performed for a vector register with N elements. But we can perform the additions on a binary adder tree as the following LISA code examples shows for the `VAP` instructions

```
OPERATION VAP IN pipe.EX { /* Vector scalar add of all P register */
DECLARE { REFERENCE dst; }
CODING  { 0b100101 }
SYNTAX  { "VAP" }
BEHAVIOR
```

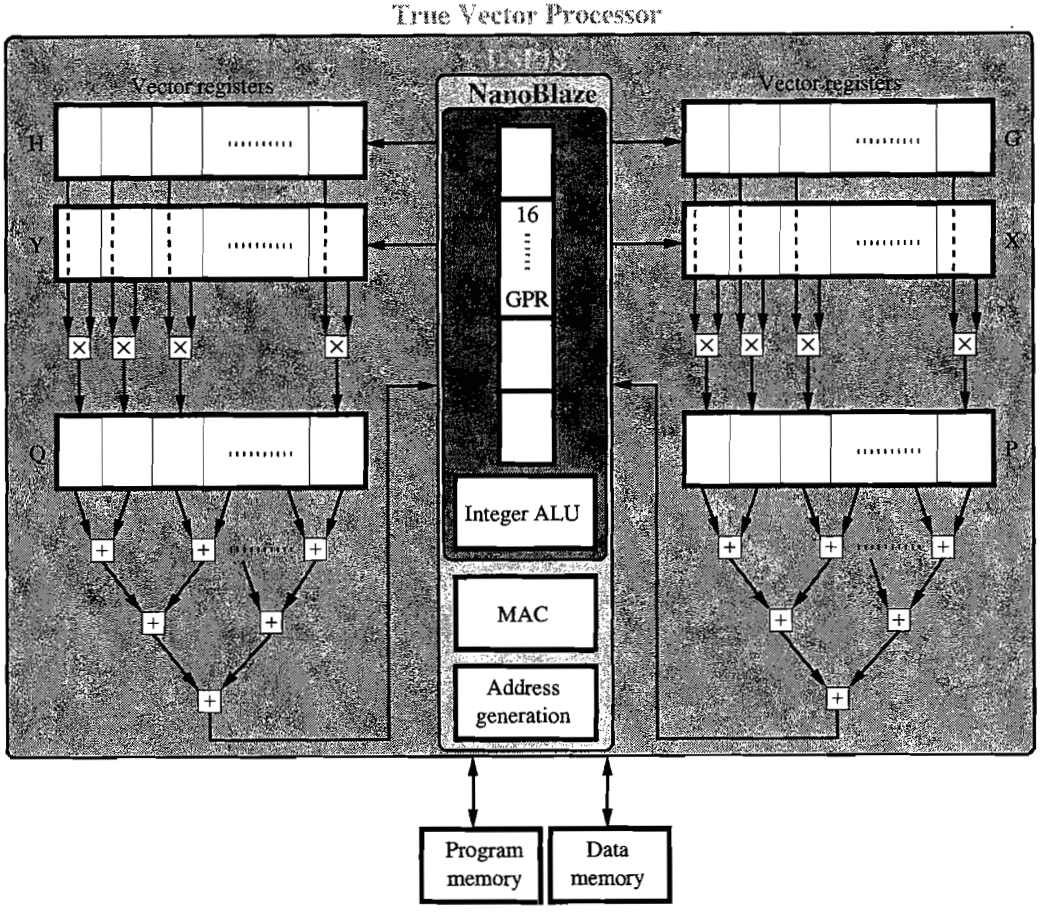

Figure 5. NanoBlaze, DSP18, and true vector processor (TVP) architecture (©2007 Springer[9]).

```
  {short t1, t2, t3, t4, t5, t6, t7;
    t1 = P[0] + P[1]; t2 = P[2] + P[3]; t3 = P[4] + P[5];
    t4 = P[6] + P[7]; t5 = t1 + t2; t6 = t3 + t4; t7 = t5 + t6;
    dst = t7;
  }
}
```

which reduces the worst case delay from 7 additions to 3.

If we now implement the DWT length-8 processor with the TVP ISA we find that the inner loop is much shorter, i.e., only 9 instructions are needed. To implement the downsampling by 2 of the DWT two initial shifts of vector `X` and `Y` are required.

```
_loop:
  VSXY  R[2],R[3]     ; Vector shift X,Y and load R[2] and R[3] in first element
  VSXY  R[2],R[3]     ; second shift since downsampling by 2 is performed in DWT
  VMUL                ; Perform all 2x8 multiplications and store results in P/Q vectors
  VAP   R[4]          ; Add vector P elements and store sum in register R[4]
  VAQ   R[5]          ; Add vector Q elements and store sum in register R[5]
  STR R[4], R[6]      ; Store R[4] using R[6] as pointer with post-autoincrement
```

Table 2. (a) TVP synthesis result for the Xilinx device XC3S1000-4ft256 according to Xilinx ISE mapping and timing analysis. (b) BlockRAM synthesis results of three different DWT length-8 processor designs using LISA for Xilinx device: XC3S1000-4ft256

(a)

Parameter	μP only	with BlockRAM
Slices	4907	4993
4-input LUT	8850	9226
Multiplier	18	18
BlockRAMs	0	2
Total gates	141,158	274,463
Clock Period	22.082 ns	20.799
$F_{\max}$	45.3 MHz	48.1 MHz

(b)

Parameter	NanoBlaze	DSP18	TVP
LISA operations	28	32	40
Prog. memory	$2^7 \times 18$	$2^7 \times 18$	$2^7 \times 18$
Data memory	$2^8 \times 16$	$2^8 \times 16$	$2^8 \times 16$
BRAMs	2	2	2
Equiv. Gates	162,471	177,203	274463
MHz	73.9	51.11	48.1

```
STR R[5], R[6]      ; Store R[5] using R[6] as pointer with post-autoincrement
BDS @_loop, R[7]    ; Check if R[7] > 0 and jump to begin of loop
SUB R[7],R[7],R[1] ; this instruction is in the branch delay slot
```

The overall instruction count when compared with DSP18 is further decreased as can be seen from the profile shown in the 4^{th} column of Figure 6(a). The total instruction count for TVP is only 479 for the 100-point two 8/8 channel DWT. From Table 2(a) we notice large resource requirements and lower maximum operation frequency.

5. LISA PROCESSOR DESIGN COMPARISON

Let us finally compare all three designs in terms of size, speed and overall throughput in MSPS for a DWT length-8 example. The key synthesis properties are summarized in Table 2(b). Because the Xilinx logic FPGAs have embedded multipliers, BlockRAMs, and CLBs with one 3-input and two 4-input tables, the design area used is the equivalent number of gates from the Xilinx "Mapping Report File." To have reliable timing data we use the "Post Place&Route Static Timing Report" rather than the map time estimations.

The device used is a Spartan-3 XC3S1000-4ft256 that is used in the $119 low cost Nexys Digilent University boards (see `http://www.digilentinc.com/`), with 7680 slices, 15360 4-input LUTs, 24 embedded multiplier, and 24 BlockRams with 18 Kbit each. When data or program memory are implemented with CLB-based RAM (called distributed RAM by Xilinx) then about 800 4-input LUTs are required for a $2^8 \times 16$ and about 120 4-input LUTs for a $2^7 \times 18$ memory.

The overall performance of the three processors is measured by the throughput (MSPS) when implementing a length-8 DWT as shown in Fig. 2(a). We need two length-8 filter $g[n]$ and $h[n]$ and for each output sample pair 16 multiply and 14 add operations are computed. For 100 samples with an output downsampling by 2 the arithmetic requirements for the DWT filter band would therefore be $8 \times 100 = 800$ multiplications and $7 \times 100 = 700$ additions, or 800 MAC calls. From the NanoBlaze instruction profile however we see that many addition cycles for `LDR` and `ADD` are required, due to the fact that the register updates for the memory access are also computed with the general purpose ALU. The DSP18 processor reduces the `LDR` and `ADD` essential, and although the maximum clock frequency is decreased the overall throughput is improved by a factor of 2. If we use a true vector processor, then a inner product can be computed in two clock cycles and the overall throughput is improved by a factor 8 compared with NanoBlaze and a factor 4 when compared with a single core MAC DSP18 design.

Finally, the performance data of the 3 LISA based processors and a direct RNS polyphase implementation on an Altera FPGA (4217 LEs, 155 MSPS[19]) are compared in Fig. 6(b). We see the large improvement from NanoBlaze to TVP, that closes the performance gap between a direct mapping of the algorithm into FPGA hardware and the microprocessor solution. The polyphase hardware architecture however can just implement only one configuration, while the TVP software architecture can implement many different algorithms, e.g., adaptive filters, 2D filter etc.

Parameter	NanoBlaze	DSP18	TVP
LDL	261	260	7
LDH	309	308	7
LDR	1600	0	0
VSXY	–	–	107
VSGH	–	–	8
VMUL	–	–	50
VAP	–	–	50
VAQ	–	–	50
MAC	–	800	0
STR	100	100	100
ADD	2750	400	0
SUB	–	50	50
MUL	800	0	0
BC	0	0	0
BC	0	0	0
BDS	50	50	50
Total	5870	1968	479
Clock	73.9	51.1	48.1
MSPS	1.26	2.60	10.0

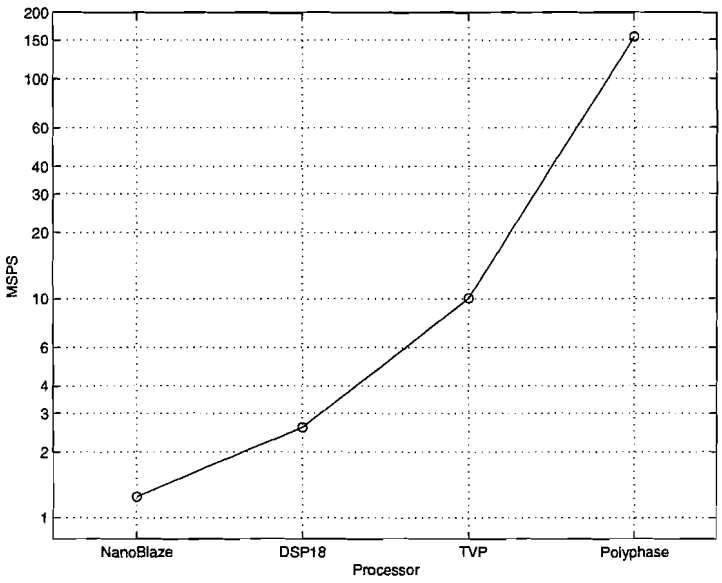

Figure 6. (a) Comparison of three different DWT length-8 processor designs using LISA for Xilinx device: XC3S1000-4ft256 (b)Comparison of LISA-based processor and direct RNS polyphase implementation.

6. SUMMARY AND FUTURE STUDY

We have presented two channel 8/8 DWT LISA processor designs for a reduced instruction set computer (RISC), a programmable digital signal processor (PDSP) and true vector processor (TVP). The following conclusion can be made:

- Consistency between RTL and architecture of the models is guaranteed by designing the processor in LISA.
- The error prone and difficult task of writing assembler, linker, simulator, debugger is accomplished by LPDP in a few seconds.
- HDL and synthesis script generation by LPDP allows an immediate verification in silicon and make size, speed, and power data available in the design process.
- The throughput for an 100-point DWT example shows a factor 8 improvement for a TVP by adding a few LISA operations.
- The required clock speed of the processor decreased from 73 MHz (RISC) to 48 MHz (TVP) with a 8 times increased throughput that reduce the required bus speeds and power consumption.
- Low cost FPGAs like Xilinx Spartan 3 or Altera Cyclone II allow to build a TVP for $100 with todays devices.

Future goals of further investigation maybe:

- Use a more balanced pipeline to avoid the throughput decrease in TVP compared with NanoBlaze and take better advantage of the fast synchronous memory in the FPGA.
- Detail power estimation using XPower estimator and analyzer available for Xilinx ISE tools that allow a energy by task computation rather than a simple power analysis.
- Use a more realistic memory hierarchy model design and other PDSP arithmetic like block floating point or custom floating-point.

ACKNOWLEDGMENTS

U. Meyer-Baese acknowledge the support of the Humboldt Foundation. We thank M. Witte and H. Meyr from ISS RWTH Aachen for the review of an early draft of the paper. Products and company name used in this article may be trademarks of their respective owners. The authors would like to thank Xilinx Inc. and CoWare Inc. for their support under the University programs. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

1. Altera Corporation, "Netseminar NIOS processor." http://www.altera.com, 2004.
2. Altera Corporation, "Delivering RISC processors in an FPGA for $2.00." White Paper, 2002.
3. Xilinx Inc. Online, "Microblaze – the low-cost and flexible Processing Solution." www.xilinx.com, 2005.
4. A. Hoffmann, H. Meyr, and R. Leupers, *Architecture Exploration for Embedded Processors with LISA*, Kluwer Academic Publishers, Boston, 1 ed., 2002.
5. P. Ienne and R. Leupers, *Customizable Embedded Processors*, Morgan Kaufmann, Boston, 1 ed., 2006.
6. J. Donovan, "The Truth about 300 mm." EETimes http://www.eet.com, 2002.
7. C. Rowen, *Engineering the Complex SOC*, Prentice Hall, Upper Saddle River, NJ, 1 ed., 2004. Tensilica Founder.
8. Xilinx Inc. Online, "Electronic system level design ecosystem." http://www.xilinx.com/products/design_tools/logic_design/advanced/esl/index.htm, 2007.
9. U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, Springer-Verlag, Berlin, 3 ed., 2007. 631 pages.
10. BDTI, "Pocket Guide to Processors for DSP." http://www.bdti.com/pocket/pocket.htm, 2007.
11. U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, Springer-Verlag, Berlin, 2 ed., 2004. 527 pages.
12. Analog Device, "ADSP-2103." 3-Volt DSP Microcomputer, Apr. 1993.
13. Texas Instruments, "TMS320C5x User's Guide." Digital Singal Processing Products, 1993.
14. U. Meyer-Baese, H. Natarajan, E. Castillo, and A. Garcia, "Faster than the FFT: The chirp-z RAG-n Discrete Fast Fourier Transform," *Frequenz* **60**, pp. 147–151, July 2006.
15. U. Meyer-Baese, J. Chen, C. Chang, and A. Dempster, "A Comparison of pipelined RAG-n and DA FPGA-based multiplierless Filters," in *Proceeding APCCS conference*, in press, Dec. 2006.
16. J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman Publishers, Inc., San Mateo, CA, 3 ed., 2003. Appendix F: Vector Processor.
17. V. Zivojnovic, J. Velarde, C. Schläger, and H. Meyr, "DSPstone: A DSP-oriented Benchmarking Methodology," in *Proc. of ICSPAT*, pp. 1–6, Oct. 1994.
18. ISS RWTH Aachen, "DSPstone." Institue for Integrated Systems for Signal Processing, Final Report, Aug. 1994.
19. J. Ramirez, U. Meyer-Baese, and A. Garcia, "Efficient Wavelet Architectures using Field-Programmable Logic and Residue Number System Arithmetic," in *Proc. SPIE Int. Soc. Opt. Eng.*, pp. 222–232, (Orlando), Apr. 2004.