

A scalable parallel computational core for embedded processing

Refik Shadich, Ian V McLoughlin, *Senior Member, IEEE*

Abstract—Embedded computational hardware has become prevalent in recent years for communications signal processing for reasons including size and cost. The availability of competing single processor solutions from traditional vendors gives system designers a degree of choice. Some recent market entrants have even embraced parallel concepts within their architectures. However the fact remains that while one particular computational device or parallel configuration may suit a given application, it seldom suits a broad range of other applications. This promotes design inefficiency: either developers familiar with one solution from a previous project choose to use it for the next project despite some probable degree of mismatch, or they are faced with a costly learning curve implied in the adoption of a different, but possibly better matched, architecture.

A preferable approach is to allow computational hardware to be adapted at a micro- and macro-architectural level to fit requirements on a project-to-project basis, but maintaining common instruction set and development tools. This gives designers the flexibility to choose the degree of parallelism and type of parallel arrangement required for their application, but without requiring a new tool and hardware learning curve.

This paper describes the 2ke, a flexible and modular computational system that allows developers to standardise on one processor, instruction set, software architecture and toolchain for many projects. Architectural enhancements to its forerunner, the 2k2, are presented to permit micro-architectural parallelism to be chosen along a continuum from SISD at one extreme to full SIMD at the other, whilst the very nature of the 2ke permits extension to MIMD along an orthogonal development direction. Results in terms of logic cell usage, current consumption and memory usage will be presented for each arrangement for example application code.

Index Terms—DSP, parallel processing, embedded, soft core

I. INTRODUCTION

THE attitude of traditional signal processor vendors has predominantly been based on a 'one-size-fits-all' approach until the widespread adoption of field programmable and mask programmable gate arrays (FPGAs and MPGAs). Since this time, improvements in design choice have come about both through the advent of soft cores and through system-on-chip silicon for volume OEMs. These approaches have undoubtedly allowed a better degree of matching between what a vendor can offer and what the developer requires through customisation. However even when customisation is available or is cost-effective, it may only apply to on-chip peripheral components and generally not to the microarchitecture of a processor core itself.

Manuscript received May 10th, 2005

R. Shadich is with the Custom Logic Group, Tait Electronics Ltd, PO Box 1645, Christchurch, New Zealand, email: refik.shadich@tait.co.nz

I. V. McLoughlin is with Group Research, Tait Electronics Ltd, PO Box 1645, Christchurch, New Zealand, email: ian.mcloughlin@tait.co.nz

FPGA and MPGA technology has become popular due to its flexibility: characteristics of which allow rapid prototyping of digital systems. In recent years, vendors have promoted processor core use within their FPGAs. Prominent examples are the Nios core from Altera [1], PowerPC and MicroBlaze cores from Xilinx [2]. The SPARC-based LEON core [3] and many cores available from opencores.org [4] also appear to be popular. None of the common soft cores are inherently parallel in architecture: however in a 50,000 logic element (LE) FPGA, parallelism can be achieved through repeated instantiation of a core such as the 1100LE Nios [5].

Cores allow developers to choose and in some cases customise a CPU solution to match their embedded requirements, and can add custom processing hardware around this. From one project to the next, basic hardware, including the FPGA and peripherals may remain unchanged. Software running on the core may be in C or assembly language and single instruction multiple data (SIMD) processing has been demonstrated with such cores [6].

Whilst cores appear to meet customers requirements for flexibility and ease-of-use, in reality, the development effort is mostly translated to software domain effort: when a new CPU core is chosen, it may require adopting a different toolchain to other cores, and certainly a degree of knowledge is not directly transferable to another core. Where advances in compiler technology have lagged behind hardware developments, the learning curve duration and cost when adopting a new processor will be extended. This is exacerbated where hand coding rather than high level coding remains prevalent such as for many embedded applications.

A dichotomy remains when moving on to a new embedded project: re-use a known core from a previous project or choose a new core. The old core may not be a good match for current needs, but will involve substantially less acclimatisation at a software and tools level than a new core that may be a better computational fit. More recent approaches allow adaptation of processor architecture to the program being executed [7], [8]. This either requires complex software or substantial runtime management hardware within the processor core for dynamic adaptation. However soft core adaptation can be made not only in terms of degree of unit parallelism, but also in internal parallelism. This flexibility both within units and with unit instantiation makes such a system ideal for exploitation of parallel structures for multiple applications. In this paper the adaptation of an embedded core, the 2k2, is explored in terms of parallel structures related to implementation and power efficiency.

The 2k2 system allows a single assembly language and

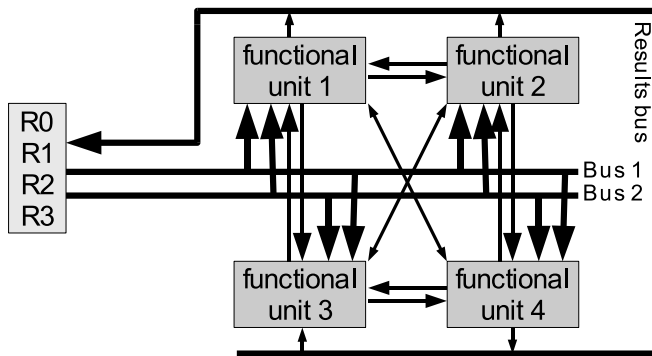


Fig. 1. Internal bus structure of the 2k2 showing dual interconnection buses and results bus connecting to all functional units in addition to hardwired data forwarding paths

toolchain to be adopted and employed for a wide range of projects requiring different processor arrangements. A development team can own both the software and the firmware for their processing solution. The concept of a modular processor core allows CPU core capabilities to be expanded at will with little or no software change. This includes adopting a single instruction multiple data (SIMD) model, or a multiple instruction multiple data (MIMD) model or simply using the core in a single instruction single data (SISD) arrangement, all named according to Flynn's Classification [9]. The core can thus be changed to suit many target applications and the multi-dimensional trade off between processing capability, code density and cost can be optimised at will at a fine-grained level. The extra dimension of development cost due to tool familiarisation can be largely removed from the equation once an initial design is complete. Since the team can adapt the same core at will, there may be no need to move to another processing solution.

II. BASIC DESIGN

The original 2k2 comprises VHDL core, functional unit library and development suite. The core contains modules for instruction decode unit, register file, bus framework, and so on as described in IIA below. Functional units provide operations, memory and input/output interfacing.

While an SISD machine uses a single core but multiple functional units, section IIC extends this to SIMD/MIMD. Where LE figures are given, these are taken from compilation reports from the Altera QuartusII version 4.0 tool. Indications at the time of writing are that the version 5.0 tool can reduce the size of the optimised 2k2 by around 30% in many cases (to 850 LEs for an SISD instantiation)

A. Internal Structure

In each bus framework, two main data buses are connected to a register file. These buses are operated similar to those found in standard dual-bus microprocessors as shown in fig. 1 where it can be seen that two buses deliver operands from a register file simultaneously. A result bus normally carries the output of functional units back to the register file. The original 2k2 [7] made extensive use of data forwarding paths

[10] to feed certain outputs to the input of certain other various functional units. Fig. 2 shows the method of driving the dual buses from the register file and the logical multiplexer to results bus. This is clearly a well-proven arrangement.

In implementation terms the connections to each functional unit are simply a function of the size and connectivity of the multiplexer on the output and input as fig.3 shows. Note that functional units are not named, rather the underlying machine code simply moves data to and from different endpoints, where the endpoints are identified through a mapping between assembler mnemonics and the hardware configuration of a particular instantiation. There is evidently no need in hardware, or software to require all functional units to be interconnected although limited interconnection based on code structure may be desirable, and this is described in section IIC.

Functional units provide all processor functions except for data move, instruction decode and register file. ALU, multiplier (either complex or real), divider, shifter, ECC generator and other specialised functions. Since the primary use of the processor to date has been in telecommunications products, a number of hardware signal processing and control extensions have been defined. The nature of these is that they do not all need to complete in a single cycle, but that the cycle timing (if not single cycle) must be taken into account by the assembler when it tracks resource usage.

B. Instruction Set Architecture

The 16-bit instruction set is two thirds occupied by predefined instructions for basic common operations such as data moves, ALU operations, shifts and multiplies, leaving space for customisation and composite instructions. This mechanism provides support for parallel operation.

Since there is a level of abstraction between the functional unit position and the operation that the unit is capable of, the hardware configuration has to be determined prior to compile time, and effort has been made to define tools that preprocess code to determine efficient arrangements of functional units - since data forwarding paths may not exist between each of these. In fact certain units (such as ALU) are able to perform a variety of operations whilst some units (such as I/O port and multiplier) are hard wired in function. For this reason, all functional units must be provided with sub-function-configuration signals by the instruction decode unit irrespective of their position and current status. This is not inefficient since unused signals will be optimised out of an FPGA during the synthesis process.

Custom instructions are software definable at synthesis time. This is possible since the processor itself is synthesised with its program into a single monolithic block of FPGA code. Custom instructions support the custom functional units identified by bus decoder/multiplexer location or can be composite instructions allowing simultaneous actions to occur, where physically possible. A major reason for allowing customised instructions is to allow the user a higher degree of optimisation control especially when implementing the processor in a parallel form.

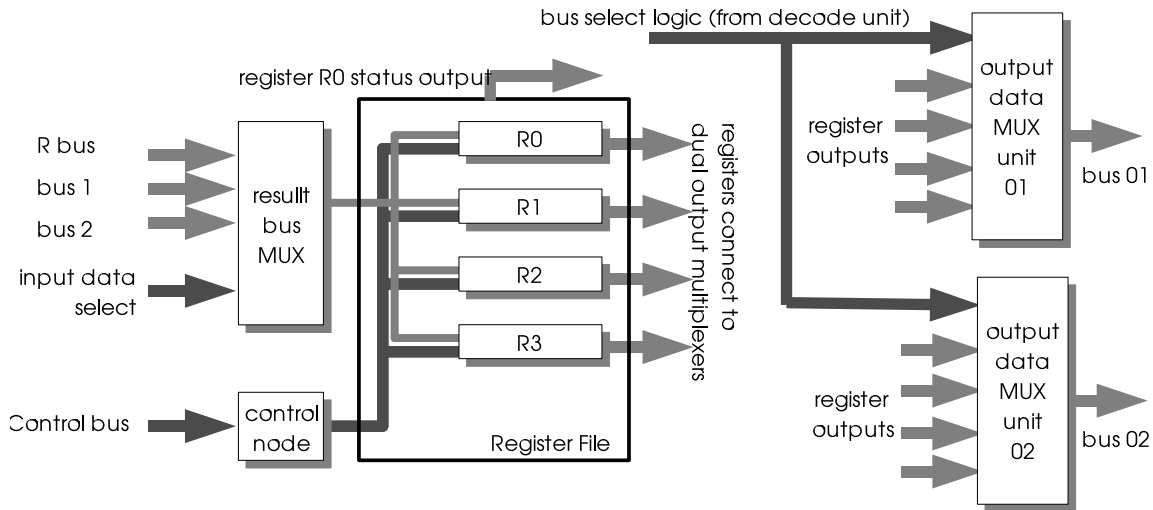


Fig. 2. Register file multiplexer connection to dual buses within the 2k2 architecture

C. Implementing Parallelism

The 2k2 is by nature not a parallel machine. This is deliberate since to make it parallel using, say a fine-grained MIMD approach, would likely rule out a course-grained SIMD arrangement. In this section, soft cores in general are analysed in terms of implementation in a parallel structure. Section IV will later benchmark the 2k2 with example code.

Single CPU cores can always be configured into a MIMD arrangement through multiple instantiation. This is particularly true of FPGA-based soft cores such as the 2k2 [7] and 2ke. For SIMD-type processing, many cores can simplistically be wired to the same control memory as in fig.4a or more efficient would be to modify the internal structure of the core to remove the replicated instruction fetch and decode units as in fig. 4b. Whether a particular soft core can be modified in this way is a function of licensing terms and access to source rather than basic design.

The 2ke has the flexibility to add and remove functional units to suit a particular application. For some applications an implementation short of a full SIMD machine, but with additional functional units over and above the SISD machine, is more efficient. We term this 'virtual SIMD'. By contrast, other applications may suit an MIMD implementation. These questions deal with multiples of the 2ke whereas the question of the structure within each 2ke can be separately optimised. We term these dimensions stackability and scalability. The ability to add or delete hard wired data forwarding paths between blocks bypassing crowded shared bus resource, ensures that such resource conflicts are far less likely to impact performance.

Composite instructions are those which combine a number of operations simultaneously. These are user-definable at compile-time and aim to improve code density. This may not be true for a general-purpose loadable software processor, but is of small embedded-program processors in which all code is known and fixed at design time. In the 2ke this simply impacts the instruction decode unit look-up-tables.

III. APPLICATION

In order to analyse an MIMD, optimised MIMD and virtual SIMD arrangement of 2ke and compare these to straight SISD, a baseline application is applied [11]. This is in a telecommunications product where it implements various low-level control and baseband signal processing functions. These include operations on dedicated control lines, ADC (analogue-to-digital) and DAC (digital-to-analogue converter) handling at 50kHz, automatic gain control (AGC) computation and implementation of dual 16-bit, 20-tap FIR filters, also at 50kHz.

The baseline program, hand-crafted in assembler, resides in considerably less than 1kbyte of memory, and utilises less than 800 Altera logic elements (LEs) [1]. Note that the 2ke program occupies embedded RAM but no LEs. The low memory requirement is a major consequence of flexibility in the instruction set and judicious use of composite instructions.

When implemented in an Altera EP1C20F400C7 cyclone device, alongside a 115200 baud UART, hardware interfaces to ADC and DACs, two sample buffers and debug support, the design uses 1359 LEs (Logic Elements - out of a total of 20060) and 24576 memory bits (out of a total available of 294912 bits). In this configuration 6.7% of LE resource and 8.3% of memory resource are utilised.

The implementation performs much of the signal-processing requirements of an analogue radio communication system with software control as part of a commercial product. Elements of the software were implemented in TMS320C5xx and ARM assembler for comparison with 2k2 in terms of code density and operating efficiency with these processors in [11]. The 2ke is an extensible version of the 2k2 where one configuration of the 2ke is identical to the 2k2.

IV. EVALUATION

In this section, the parallel 2ke is evaluated in terms of price and flexibility.

TABLE I

TABLE 1: HOW AN EXAMPLE SAMPLE SCALING PROGRAM WOULD OPERATE ON AN SISD 2K2 MACHINE.

Instruction	Resources used							
	ALU	MEM	MUL	IO	REG	Bus1	Bus2	Rbus
Using fig. 5a architecture								
if R0=0 then io1=0 else io1=10	N	N	N	Y	Y	Y	N	N
R0=R4 x *ptr0 + R0	Y	Y	Y	N	Y	Y	Y	Y
R1=R5 x *ptr0 + R1	Y	Y	Y	N	Y	Y	Y	Y
ptr0++	Y	N	Y	N	N	Y	N	Y
out io0, R0	N	N	N	Y	Y	Y	N	N
out io1, R1	N	N	N	Y	Y	Y	N	N
Total 6 cycles, at hardware cost of approx 1400LEs								

TABLE II

TABLE 2: HOW THE SAME EXAMPLE PROGRAM AS IN TABLE 1 WOULD OPERATE IF RECODED FOR AN SIMD MACHINE AS SHOWN IN FIG.5B

Instruction	Resources used							
	ALU	MEM	MUL	IO	REG	Bus1	Bus2	Rbus
Using fig. 5b architecture								
if R0=0 then io1=0 else io1=10	N	N	N	1	1	1	N	N
	N	N	N	2	2	2	N	N
R0=R4 x *ptr0 + R0	1	1	1	N	1	1	1	1
	2	2	2	N	2	2	2	2
ptr0++	1	N	1	N	N	1	N	1
	2	N	2	N	N	2	N	2
out io0, R0	N	N	N	1	1	1	N	N
	N	N	N	2	2	2	N	N
Total 4 cycles, at hardware cost of approx 2800LEs								

TABLE III

TABLE 3: THE EXAMPLE PROGRAM FROM TABLE 1 IMPLEMENTED ON AN OPTIMISED VIRTUAL SIMD MACHINE USING DEDICATED DATA-FORWARDING INTERFACES TO REDUCE DEPENDANCE ON SHARED BUS RESOURCE.

Instruction	Resources used							
	ALU	MEM	MUL	IO	REG	Bus1	Bus2	Rbus
Using fig. 5a architecture								
if R0=0 then io1=0 else io1=10	N	N	N	Y	Y	Y	N	N
R0=R4 x *ptr0 + R0	2	2	2	N	Y	Y	Y	Y
R1=R5 x *ptr0 + R1								
ptr1++	1	N	1	N	N	Y	N	Y
out io0, R0	N	N	N	Y	Y	Y	N	N
out io1, R1								
Total 4 cycles, at hardware cost of approx 2000LEs								

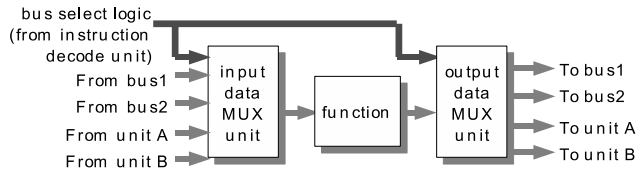


Fig. 3. Multiplexer size limits number of custom interconnects in the 2k2 processor

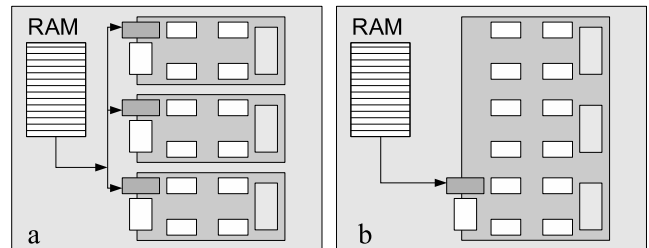


Fig. 4. Multiple instantiations of core in (a) giving an SIMD arrangement, or an optimised version (b) without replicated instruction fetch, decode and control units.

The basic core from section III utilises between 14% and 18% of the LE and memory resources in an Altera Cyclone EP1C6 (depending on optimisation level). Since an EP1C6 currently costs around €7 in volume It would therefore be reasonable to price a single instance of the core at about €1. One EP1C6 device can host 6 or 7 individual cores in a true MIMD arrangement, while tests show that using the optimised

MIMD structure of fig.4b at least 8 cores can be operated in parallel.

The 2ke can clock at up to 50MHz in the Cyclone, and

can execute all built-in instructions apart from divide in a single cycle. Since the functional units can be extended at will, it can perform 50MMACS (million multiply-accumulates per second) per multiplier. If an embedded application requires more than this, the solution is to either instantiate a second 2ke and partition the processing (MIMD), or to add one or more multipliers to the processor. Fig. 5 shows this process graphically. Table 1 introduces an example code fragment that applies fixed scaling to data obtained from one 16-bit input port, and outputs the result from another port. There are two such independent streams handled here: an obvious example in which parallelism is advantageous. The 2ke can perform this function in a number of ways. Table 1 analyses the resource usage when implementing this in a single instance of SISD 2ke.

In table 2, dual 2ke cores have been used fed from a single program in a full parallel SIMD arrangement. This evidently suits the code fragment although the analysis of resources used shows considerable unused resource. The hardware arrangement for this is shown in fig. 5b.

Table 3 on the other hand analyses the resource requirements of the re-coded program for the architecture of fig. 5c, the virtual SIMD arrangement in which only the most used functional blocks are replicated. The 2ke data forwarding feature is used in this case to remove the over-dependence on shared resource (namely the three global buses).

This underlines the principal that since not all functional units are used at capacity, it may be more efficient to replicate almost all functional units instead into a virtual SIMD machine that replicates sufficient for data processing needs but does not replicate slow control. Within the 2ke the concept of a virtual machine can also be enabled through a combination of composite instructions (controlling multiple functional units simultaneously) and hard wired data forwarding paths between units (controlled through input/output multiplexers). In the code example given, a simple composite instruction could perform the pointer increment and I/O output function since, with the exception of bus1 which can be replaced by bus2, both instructions utilise orthogonal resource sets.

Since every instruction in the example of table 1 to table 3 completes in a single cycle, performance analysis in terms of time to completion is trivial. More important for an embedded system is the overall hardware cost of the implementation. Fig. 6 plots the current consumption, memory utilisation and resource usage in logic elements for between 1 and 6 instances of a 2ke, or equivalent in MIMD, SIMD (fig. 5b) and virtual SIMD arrangements implemented in an Altera Cyclone EP1C20F400C7 device running at 13MHz. The effect of different clock rates on single 2ke instance current consumption was explored in [11], and is here expected to scale linearly with logic element usage due to the synchronous nature of the design. In the test system, devices were running a test program that contained a single parallel task of the nature of that in tables 1-3, and all non-essential on-FPGA devices removed apart from a 10 LE control interface. This has been subtracted from the utilisation figures plotted in fig. 6 although the figures include a single instance of divide hardware for each unit.

Fig. 6 clearly shows the expected minimal savings in

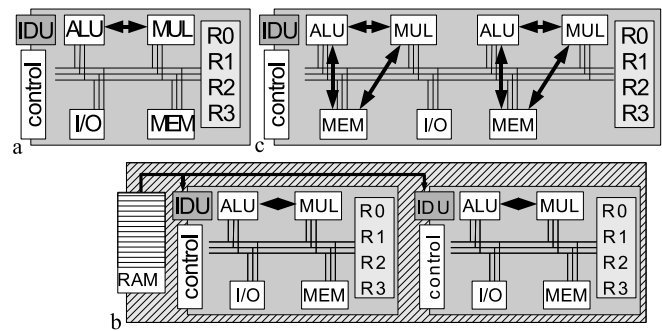


Fig. 5. Single 2ke (a) extended with extra functional units (c) into a true SIMD machine (b) where IDU is instruction decode unit, R0-R3 is the register bank and MEM is a memory interface.

hardware on a per-instance basis with multiple instantiations due to compiler optimisations, and the more significant savings using a virtual SIMD arrangement. Due to the single-cycle nature of the 2ke all methods compute the program at the same rate, the difference lies in the efficient usage of programmable logic resources and the power consumption. The graph shows that power consumption is linear with number of 2kes. The slope of course, reduces in the SIMD and virtual SIMD cases although this is not shown.

V. CONCLUSION

For certain applications standard parallel or non-parallel CPUs may be an ideal solution, whilst for other applications a customised solution is preferable. The 2ke soft core allows users to perform these customisations within the existing instruction set architecture and bus framework. This paper has shown that this can be extended to providing a degree of parallelism in a continuum from SISD to SIMD, all without imposing significant changes in the development toolchain and assembly language experienced by the developer. The concepts of stackability and scalability have been demonstrated as applied to the core. Examples have been given of 2ke source code which indicates the effectiveness of the architectural modifications possible to the 2ke, all of which are accomplished at a user level with no change in development toolchain. An example has been investigated of different arrangements from SIMD to MIMD and the virtual SIMD arrangement made possible through internal reorganisation of the arrayed processor functional units. Results show the current consumption and hardware utilisation of the cores as small numbers are paralleled together using the different arrangements, with the virtual SIMD arrangement scaling more advantageously than other arrangements.

REFERENCES

- [1] (2005) Altera webpages. [Online]. Available: <http://www.altera.com>
- [2] (2005) Xilinx webpages. [Online]. Available: <http://www.xilinx.com>
- [3] (2005) Gaisler research webpages. [Online]. Available: <http://www.gaisler.com>
- [4] (2003, May) Opencores webpages. [Online]. Available: <http://www.opencores.org>
- [5] (2005) Altera nios 3.0 datasheet, ds-nioscpu-2.1. [Online]. Available: <http://www.altera.com>

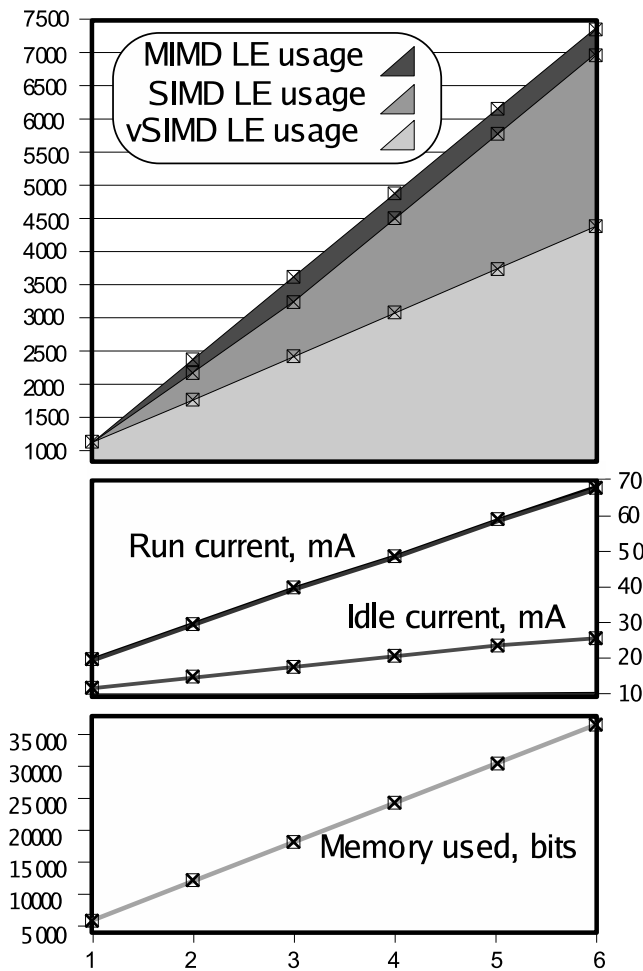


Fig. 6. Showing the results for 1 to 6-way parallel implementations of the 2ke in MIMD, SIMD and virtual SIMD form capable of running the example code from tables 1-3. The SIMD arrangement shares instruction fetch/decode unit as in fig.4b. LE usage, measured current consumption for MIMD case and memory usage is given for Altera Cyclone 1C20 device.

- [6] X. Wang and S. Zivarras, "Parallel direct solution of linear equations on FPGA-based machines," in *11th International Conference on Parallel and Distributed Real-Time Systems*, Nice, France, Apr. 2003.
- [7] R. Shadich and I. V. McLoughlin, "The 2k2: a modular computational toolkit for embedded signal processing," in *2003 Int. Conf. on Integrated Circuits and Systems, and IEEE Pacific Conference on Multimedia*, Singapore, Dec. 2003.
- [8] P. M. Athanas and H. F. Silverman, "An adaptive hardware machine architecture and compiler for dynamic processor reconfiguration," in *Int. Conf. on Computer Design: VLSI in Computers and Processors*, Oct. 1991, pp. 397-400.
- [9] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. C-21, pp. 948-960, 1972.
- [10] A. Tanenbaum, *Structured Computer Organization*, 4th ed. Prentice-Hall, 1999.
- [11] R. Shadich and I. V. McLoughlin, "A modular computational engine for communications processing," in *Australian Telecomms. and Networking Apps. Conference*, Melbourne, Australia, Dec. 2003.