Optimized Reconfigurable MAC Processor Architecture

M. Iliopoulos and T. Antonakopoulos

The 8th International IEEE Conference on Electronics, Circuits, and Systems, ICECS 2001

MALTA, 2– 5 SEPTEMBER 2001

OPTIMISED RECONFIGURABLE MAC PROCESSOR ARCHITECTURE

Marios Iliopoulos and Theodore Antonakopoulos

Computers Technology Institute (CTI), Riga Fereou 61, 26221 Patras, Greece
Department of Electrical Engineering and Computers Technology,
University of Patras, 26500 Rio-Patras, Greece
Tel: +30-61-997346, e-mail: antonako@ee.upatras.gr

ABSTRACT: Inefficient resources utilization is met in various embedded communication devices, which are based on standard processor cores and custom hardware modules. This paper addresses the inefficient resources utilization problem in MAC processor designs and presents a solution that is based on reconfigurable processor architecture and on dynamic-static instruction partitioning, depending on medium access protocol requirements. The presented instruction partitioning is based on statistical and time critical functional analysis for minimizing the required hardware resources.

1. INTRODUCTION

Medium Access Control (MAC) chips use powerful RISC processor cores integrated with sophisticated hardware modules in order to support the complex and stringent timing requirements of the supported access protocol [1], [2]. As the time-to-market becomes shorter and various versions of the same protocol are issued for covering new market needs and trends, the MAC chips must be designed in order to be easily adapted to new protocol requirements. This desirable feature of MAC processors increases the cost and power consumption of the system, since the chip resources are not used efficiently, while a static design could not always meet the new protocol requirements. Therefore the designer has to trade-off between efficiency and flexibility for determining the final chip architecture.
A solution to this problem is to replace the dedicated hardware by programmable logic that can be adapted to the protocol requirements (and its newer versions) in a flexible and reliable way. The reconfigurable hardware is easily adapted to new protocol requirements and may offer solutions optimized for speed, area or power consumption according to system needs. The major advantage of a reconfigurable solution is that the same logic resources can be used for implementing different functions, depending on the specific protocol functionality and this can be done ‘on-the-fly’ by exploiting dynamic reconfiguration. The idea of dynamic reconfiguration is also applied to dynamic instruction set computers that alter their instruction set during the program execution.
Generally, there are two major disadvantages on using reconfigurable hardware. The reconfigurable hardware costs more than dedicated hardware for implementing the same set of functions. This problem can be solved by increasing the reusability of the hardware resources, that is to share the hardware resources to more than one functions and thus to increase the functional density of the device. The second disadvantage of reconfigurable hardware is the long reconfiguration time, which in some cases is unacceptable, especially in dynamic reconfiguration. The work presented in this paper deals with this problem by exploiting instruction caching and dynamic-static instruction partitioning, depending on the MAC protocol requirements.
Section 2 introduces the conventional network processor design flow and illustrates the processor usage in some MAC processors. Section 3 describes the architecture of a new reconfigurable processor, while Section 4 introduces a methodology that leads to instruction set optimization. Section 5 demonstrates the results obtained by applying the new methodology to convert a conventional MAC processor into a dynamic one.

2. MAC PROCESSOR USAGE

The design flow used for designing a MAC processor is based on formal protocol description and on hardware/software partitioning [3]. The partitioning is usually based on the assumption that critical functions are transferred into hardware, while functions that require more relaxed timing can be implemented using microcode.

An example of a MAC processor design based on this flow is described in [4]. In this example, an IEEE802.11 MAC processor was designed based on an ARM CPU, while the initial processor architecture was implemented on a development board using FPGAs for design verification. This development board was used to evaluate system performance and to test MAC functions, by downloading code in the ARM processor. This process allowed the identification of time critical functions and architectural bottlenecks, at initial design stages.

After verifying the design, we used the development platform to evaluate processor usage for MAC functions execution. The evaluation was performed by calculating how frequently different instructions were used in order to implement the specific protocol functions. The same procedure was used to implement and to evaluate an IEEE802.11-to-Ethernet bridge. Both systems were based on the ARM processor core and additional custom hardware for implementing the any additional MAC protocols [5], [6]. The evaluation results are shown in Figure 1 (a) and (b). The instructions used in both protocols are less then 50% of the instruction set of the ARM processor and only six of them (ADD, MOV, CMP, LDR, STR, B) take up to 95% of the total instruction count, while the rest instructions take less then 5%. These results mean that using the 'off-the-shelf' ARM core a lot of hardware resources of the final chip are used infrequently or are not used at all.

A first approach that would lead to hardware optimisation is to design a processing unit that is optimised for executing only the instructions contained in the specific code. This approach is commonly used to implement Application Specific Instruction Set Processors (ASIPs). The disadvantage of this approach is that the implementation is of limited use. For example, we could implement a version of the ARM processor that would be optimised to execute the previous instructions efficiently, but it would have limited performance for other protocol implementations or for future updates of the same protocol that would require a different instruction mix.

The approach that has more potential in implementing flexible medium access processing systems is based on processors with configurable instruction set. These processors can load their instruction sets either dynamically, while the program is executing (run-time reconfigurable), or statically, before program execution. Both solutions are easily adaptable to the requirements of each protocol, and they have the capability to implement more complex and specialised instructions (especially when run-time reconfigurability is used). Moreover, they reduce the protocol development time, due to their adaptability and ease the implementation of the sensitive parts of the protocol by transferring functions into hardware.

The dynamic approach is the most promising one, since it offers more functional density as it replaces instructions at run-time. A dynamic instruction set processor with instruction caching ([7], [8]) downloads instructions into the reconfigurable processor, like conventional processors download instructions into a cache memory. According to the traditional dynamic instruction set processors, instruction replacement takes place each time there is a cache miss. In this paper we present the application of a dynamic approach in order to implement medium access processors, which have dynamic instruction set, compatible with a known processor (e.g. ARM), and a methodology for partitioning the instruction set to dynamic and static subsets, based on statistical analysis and the specific MAC protocol needs.

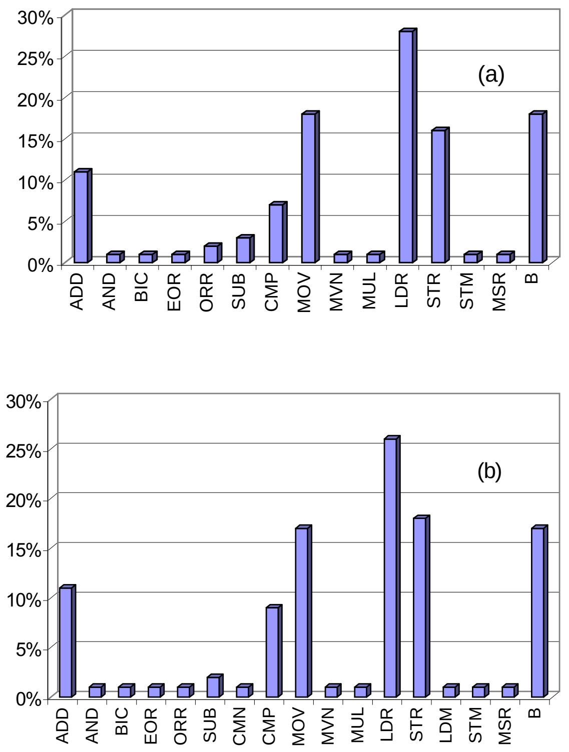

Fig. 1: Instruction usage for the IEEE802.11 (a) and the IEEE802.3 (b) MAC protocols.

3. RECONFIGURABLE MEDIUM ACCESS PROCESSING UNIT

In this paragraph we define the architecture on which the instruction set optimization methodology is based. The proposed reconfigurable architecture is illustrated in Figure 2 and is based on the principles of dynamic instruction set processors and characteristics such as: dynamic instructions based on configuration tables, fixed operand coding, re-locatable instruction loading and partial reconfiguration of the hardware resources.

The reconfigurable architecture consists of the following basic blocks: the instruction fetch and decoding units and the instruction execution unit, which consists of the static and the dynamic sections. The static section consists of configurable static instructions that are loaded at power up, while the dynamic section contains a cache of dynamic instructions that are loaded on demand. The architecture is completed by an instruction scheduler, which takes care of the caching and the dynamic instruction loading. The basic blocks of the architecture are analyzed below:

The instruction fetch unit is responsible for the generation of the addresses issued to the memory for program execution. The instruction fetching unit of the dynamic processor is similar to the fetching unit of a conventional processor and consists of a program counter that points to the position of the next instruction, an interrupt handler that issues the appropriate command in case of an exception and a branch mechanism that calculates the next program position, when a branch instruction is executed. The fetched instruction is latched in the instruction register in order to be used by the decoding unit.

The decoding unit uses the instruction register to decode the instructions and to generate the appropriate signals to the execution unit (for static and dynamic sections), the fetching unit (branch instructions) and the scheduler unit. As mentioned above, the decoding unit uses fixed operand decoding, which means that the selection of the operands is not changed at run-time, because it is encoded in the instructions at compile time. However, in order to increase flexibility, the

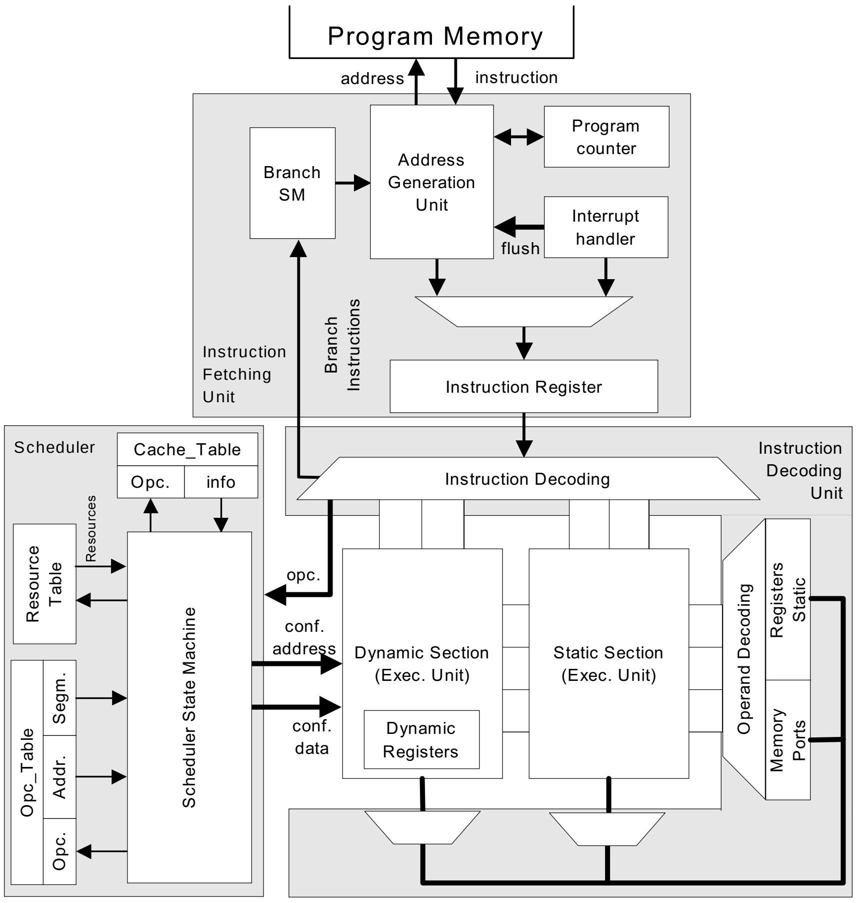

Fig. 2: The Dynamically Reconfigurable Processor Architecture

selected dynamic instruction is able to select any of the registers or memory ports, by encoding the selection in the dynamic function. The dynamic instructions can also use virtual (reconfigurable) registers that can be backed up/restored from the memory, each time the dynamic instruction that uses them is swapped in or out the reconfigurable array.

The scheduler is a special unit that loads and replaces instructions from the reconfigurable logic. The scheduler has a port connected to the memory that contains the configuration information and a port connected to the SRAM based reconfigurable array. The scheduler contains also: a table (Opc_Table) which contains the opcodes, the corresponding addresses in the configuration memory and resources required by each dynamic instruction, a table with the available resources in the reconfigurable logic (Resource_Table) and a table (Cache_Table) with the dynamic instructions that are loaded into the reconfigurable array. The scheduler state machine works as follows:

1. The opcode of the decoded instruction is compared to the opcodes of the dynamic instructions that exist in the Opc_Table. If the instruction is static, the scheduler goes to the idle state waiting for the next decoded opcode, otherwise proceeds to step 2.
2. The opcode of the instruction is compared to the opcodes contained in the Cache_Table to determine if the instruction is in the reconfigurable array. If not, the scheduler stops normal execution by issuing a HOLD signal and proceeds to step 3, otherwise it goes to idle state.
3. In this step the scheduler checks the Resource_Table to find out if there are available resources in the reconfigurable array to load the new instruction. If there are no available resources, an existing instruction is replaced according to a replacement algorithm such as the ones used in caches (random, First-in-First-out, Least Recently Used - LRU). The load of a new instruction or the replacement of an existing instruction causes the Cache and Resource tables to be updated. As soon as the reconfigurable array is loaded, the scheduler deasserts the HOLD signal allowing normal code execution.

It is obvious that the selection of the static and dynamic subsets has to be performed in such a way that minimizes the cache missing effect.

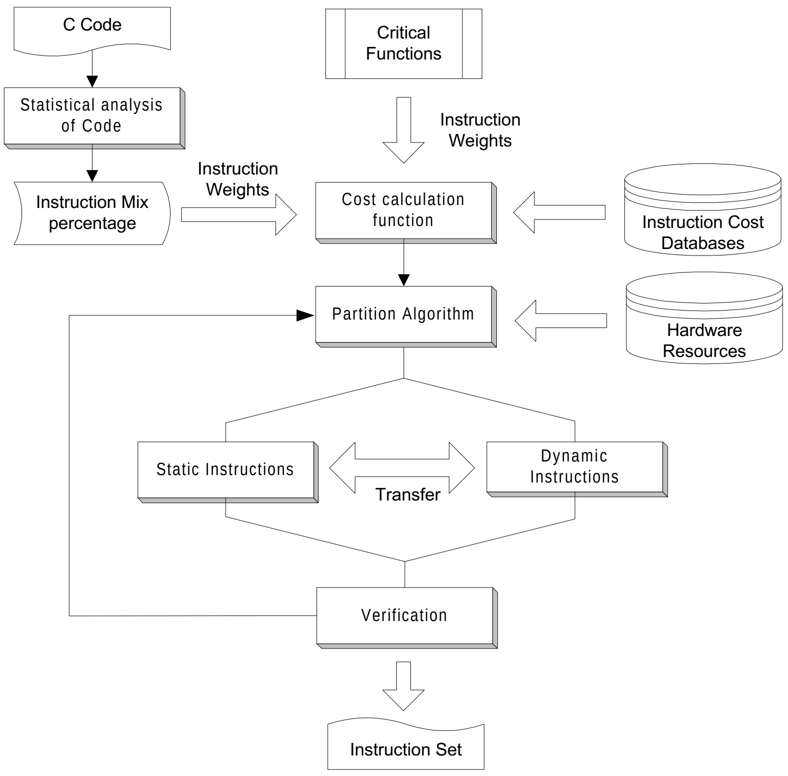

Fig. 3: Methodology for instruction set optimisation

4. INSTRUCTION SET OPTIMISATION

Using the dynamic processor presented in the previous section, we can proceed to instruction set optimisation methodology that is illustrated in Figure 3. There are two key points in this methodology: the function that evaluates the costs of the various instructions and the partitioning algorithm that divides the instructions into dynamic and static. The cost calculation function takes as input the instruction weights as extracted:

(a) by the C code of the protocol after statistical analysis and
(b) by the protocol critical functions as specified in the first step of the MAC processor development.

The number and the types of instructions that are used in the various timing critical paths of the protocol implementation determine the minimum size of the dynamic instruction cash and the static instruction subset. The cost calculation function extracts instruction weights depending on how often each instruction is used and how critical are the functions that contain the specific instruction. The calculated weights are used by the partitioning algorithm to divide the instructions into dynamic and static. The static instructions are configured at the system power-up and are not replaceable by other instructions. Dynamic instructions are replaced at run-time using the scheme described in the previous section.

The partitioning algorithm takes as input the hardware resources available and the cost of each instruction set for hardware implementation. The outcome of this process is a mix of dynamic and static instructions that fits into the available hardware resources and is capable to perform the protocol functions by minimizing the performance penalties. The instructions' cost is calculated using databases produced after implementation in a specific technology. The databases contain information like area, power consumption and performance (maximum frequency) and are used by the partitioning algorithm to evaluate different implementations.

In case a specific static-dynamic partitioning does not satisfy the critical functions, the partitioning algorithm performs new instruction set optimisation using the feedback produced to recalculate the instruction weights. The process repeats until all requirements are met. The final step of the proposed methodology performs the verification of the system, running the code at the reconfigurable processor.

5. EXPERIMENTAL RESULTS

In order to demonstrate the use of the new methodology, we emulated a dynamic version of the ARM processor used in the medium access processing systems mentioned in Section 2. The dynamic ARM processor contains the same instruction set as its static version, but implements part of them as dynamic instructions (the instructions indicated in the statistical analysis) and contains also a scheduler that executes the state machine described in Section 3. Each of the instructions was assigned a resource value that corresponds to the complexity of the instruction and an address to the configuration memory that is connected to the scheduler and stores the configuration data for each instruction. The emulation program monitored the processor performance by calculating the instruction cache misses in two different cases. In the first case the partitioning between static and dynamic instructions changes, while the total resources remain the same. In the second case, the partitioning remains the same but the cache size varies for supporting a different number of dynamic instructions.

In the first case, the partitioning algorithm divided the instructions to static and dynamic, initially using no weights for the instructions (all dynamic) and then by transferring the instructions with the higher weights in the static instructions subset. The weights of the instructions using the static analysis of code are shown in Table 1.

Partition 1 is based on static analysis results, while partition 2 uses also protocol execution information, e.g. how frequently a path or a subroutine is used. The results for these two scenarios are shown in Figure 4. Partition 1 gives its best results for the 8th run (7 static and 10 dynamic instructions). Partition 2, which approaches the optimum partition, since it uses information containing the dynamic use of each instruction, produces better results compared to partition 1 for almost all runs.

Figure 5 depicts the results produced for different cache sizes, when 6 static instructions and 11 dynamic are used. We performed 6 runs using total configurable resources that were 35, 45, 53, 60, 70 and 80% of the static configuration resources. As shown in this Figure, for cache sizes over 60% of

Table 1: Instruction Weights

Instr.	Static Weight	Critical Func. Weight	Tot.	Part.1	Part.2
ADD	0.056	0.02	0.076	8	12
AND	0.004	0.03	0.034	9	11
ORR	0.014	0.03	0.034	10	10
SUB	0.002	0	0.002	14	14
CMP	0.054	0.05	0.104	7	7
MOV	0.213	0.15	0.363	2	1
MVN	0.001	0	0.001	16	16
MUL	0.001	0	0.001	15	15
LDR	0.155	0.17	0.325	3	2
STR	0.212	0.19	0.402	1	6
STM	0.003	0	0.003	13	13
LSL	0.080	0.09	0.170	5	5
LSR	0.039	0.10	0.139	6	4
PUSH	0.010	0.02	0.030	11	8
POP	0.010	0.02	0.030	12	9
B	0.135	0.11	0.245	4	3
SWI	0.001	0	0.001	17	17

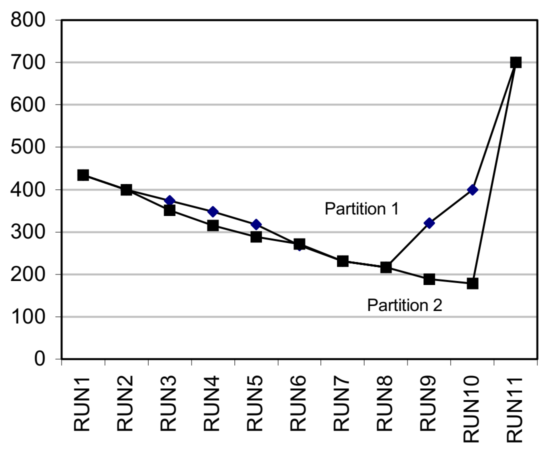

Fig. 4: Cache misses for different partitions

the total resources, the cache miss is very low (and practically does not affect the total execution time).

6. CONCLUSIONS

This paper addressed the problem of inefficient resources utilization of MAC processor designs and presented a reconfigurable processor architecture and a methodology for optimizing processor instruction sets by using the

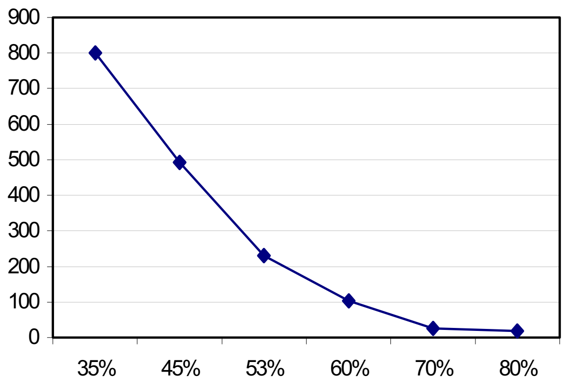

Fig. 5: Cache miss as a function of cache size (percentage of total static resources)

advantages of dynamic reconfiguration. The presented methodology produces a mix of dynamic and static instructions that implement the MAC protocol functions and evaluates different implementation options for minimizing the required system resources.

REFERENCES

[1] C-PORT, "C-5 Digital Communications Processor", Product Brief, Version 2.0, 1999.

[2] Level One, "IXP1200 Network Processor", Advance Datasheet, Revision 278298-001, 1999.

[3] Ivo Bolsens, Hugo J. De Man, Bill Lin, Karl Van Rompaey, Steven Vercauteren, and Diederik Verkest, *Hardware/Software CoDesign of Digital Telecommunication Systems*, IEEE Proceedings on Communications, Vol. 85, No. 3, March 1997, pp. 391-318.

[4] Iliopoulos, M., Maniatopoulos, A. and Antonakopoulos, T., "Design and Implementation of a MAC Controller for the IEEE802.11 Wireless LAN", *Journal of Electronics*, March 2001.

[5] IEEE Std 802.11-1997: *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification.*

[6] ANSI/IEEE Std 802.3-1996: *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access method and physical layer specifications.*

[7] Wirthlin, M., Hutchings, B., "A dynamic instruction set computer", *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, Napa, April 1995, pp. 99-107.

[8] J.R. Hauser, and J. Wawrzynek, *Garp: A MIPS Processor with a Reconfigurable Coprocessor*, Proc. IEEE Symposium. FCCM, April 1997, pp.12-21.