DOUZIEME COLLOQUE GRETSI - JUAN-LES-PINS 12 AU 16 JUIN 1989

ASIC ARCHITECTURES FOR DIGITAL SIGNAL PROCESSING IMPLEMENTATION

Stewart G. Smith, Ralph W. Morgan
and Julian G. Payne

VLSI Technology e.u.r.l., Les Taissounières, Route des Dolines,
Sophia Antipolis, 06560 Valbonne, FRANCE

RESUME

Nous décrivons une architecture générique de type "pipe-line" appliqueé aux circuits de traitement numérique du signal à haute performance. La méthode présente de nombreux avantages dans le cadre de la réalisation d'un ASIC. Cette approche se prête bien à l'automatisation de la conception et résulte en une realisation rapide de machines performants dedieés au traitement numérique du signal. implémentant une grande variété de fonctions et de débits.

SUMMARY

We describe a generic pipeline architecture for high-performance DSP, which has important advantages when coupled with ASIC technology. This approach is highly amenable to automation, and allows the rapid implementation of efficient, dedicated DSP machines over a wide spectrum of functional and throughput requirements.

Background

In recent years, IC implementation capability has greatly increased. This is due to advances on three main fronts: IC process technology, computer-aided design (CAD) and architecture. The ASIC industry has emerged to take advantage of this situation, and the possibility now exists to integrate, in rapid timescales, impressively complex algorithms in silicon. DSP stands to benefit enormously from these developments, as the complexity of DSP applications always seems to push the limits of current technology [1]. Yet, for mostly historic reasons, DSP has been associated with the high-performance sector of the computer industry. We now find that most CAD for DSP is targetted not at hardware, but at software which runs on DSP microcomputer chips.

These parts are flexible, cheap and well-supported, and each new generation has improved processing power and design aid. Often programmability is the key to a product's success, for instance modem hardware which must conform to the present and future communications standards of many different countries. But the perception of DSP design as a software problem has grown out of proportion. With the emergence of ASIC technology, designers have more power than ever to implement ideas directly in silicon. Thus we expect to see a growing requirement for dedicated DSP machines, in particular to address the processing bottlenecks at the heart of many DSP algorithms, which make them impossible to exploit by conventional means. Although parallelism and pipelining are well-known ways to overcome these problems, the inclusion of programmability severely compromises these techniques.

Admirable implementations of DSP algorithms in silicon, which use special-purpose architectures, are often reported. However the CAD tools used to realise these devices are, by modern standards, fairly low-level. The final results may be impressive, but in the absence of architectural assistance from the tools, design cycles are long. It must be recognised that DSP has its own requirements, and although it is able to benefit from the enormous investment in computer technology, DSP differs fundamentally from mainstream computer design. Before the common view of DSP machines as high-performance microcomputers can be seriously challenged, DSP-specific CAD must evolve to as high a level as computer-industry CAD, offering a complete algorithm development environment and an effective architectural means to exploit the parallelism inherent in the application. We have some ideas as to how this might be done.

Exploiting Parallelism

Parallelism exists at the heart of all DSP applications. Even the most unstructured, data-dependent algorithm can exploit parallelism at bit-level (and we would probably recommend the data-path approach for many of these). However most DSP algorithms support concurrent execution of multiple operations, and our methods are most effective when this is the case. We construct processors in the form of a dedicated network of arithmetic operators [2,3], not an ALU.

Our approach relies on decomposing computation and communication structures down to a fundamental space-time building-block, which we call a 'grain'. The grain is a gated full-adder, which processes one bit per clock cycle. Bit-level pipelining allows high clock rates and fine quantisation of the parameter space, so that resources may be closely matched to requirements. An example of this is in tuning for throughput, where two essentially unrelated quantities - task rate and max. internal clock rate - must be reconciled. The former is application-dependent and the latter is technology-dependent. Our matching mechanism involves coarse (hence easy) decisions on task- and word-parallelism, and fine decisions on bit-parallelism.

We require the overall problem to be partitioned so as to isolate a 'core' processing task, and the structure, in terms of data flow, of this task must be clear. Then a time budget (in clock cycles) may be assigned to the task. The amount of bit-parallelism is chosen so that the appropriate number of bits (known as the *working word*) are processed in the task period.

We use two forms of bit-parallelism, one of which is global and affects maximum clock rate, and the other of which may vary locally and has no effect on maximum clock rate [2,3]. This widens the space of possible solutions. Whereas a bit-serial word [4] is transmitted on a wire, a digit-serial *subword* is transmitted on a *pipe* (i.e. a cluster of wires, one per bit). As digits [5] consist of several bits, the processing budget of bits/task may be increased almost linearly by increasing digit size. The combination of bits (separated in space) and digits (separated in time) described above forms a subword, and multiple subwords (separated in both space and time) form the full working word.

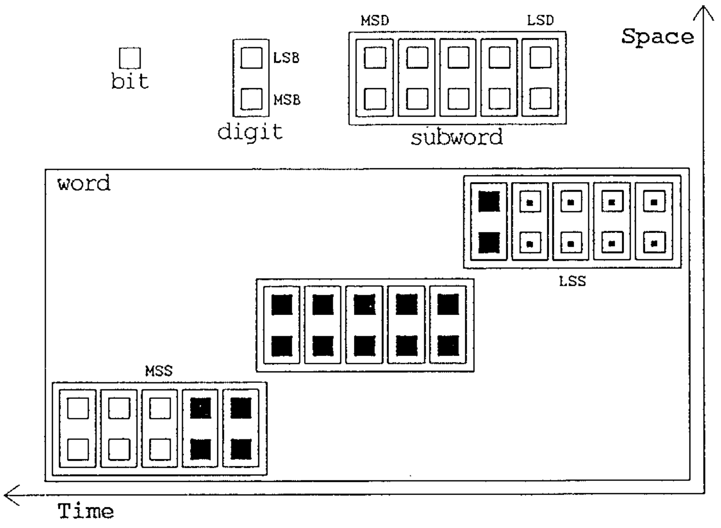

Figure 1: word structure

These *word structure* attributes are illustrated in Fig. 1. Here a 30-bit working word propagates on 6 wires in 5 clock ticks, where the 6 wires carry three 2-bit digits. The data resident in the working word may be manipulated, for instance to make an arbitrary define corner-turning as the mapping of a time index on an external bus to a space index in the processor schematic. Given a correct icon schematic, an interface schematic and sensible guidelines, we are able to synthesise a close-to-optimal realisation of the circuit.

This is achieved by propagating all word structure and format attributes through the network, checking and maintaining consistency throughout [2]. Each *actor* (the functional equivalent of an icon) has a default action on every attribute. Synthesis procedures attempt to maximise the use of the dynamic range inherent in the specified working word.

Estimation

We allow the user to alter various aspects of the design, either globally (by altering global attributes) or locally (by overriding default actions). We provide physical and numerical estimators [2,3] to provide informational feedback on the effects of any changes. Estimates are static checks, and are therefore instantaneous. This obviates trial layouts and long functional simulations in the early, exploratory phase of the design process.

Estimators are of two types: physical and numerical. By reading word attributes and counting the number of grains implied by the icon schematic, we may accurately estimate area costs. Power consumption may be estimated using known clock rates and activity factors. Numerical estimation attempts to highlight problems at both ends of the working word. At the low end, the intrusion of quantisation noise is estimated using simple statistical assumptions, and at the high end word growth and hence the possibility of overflow is estimated using statistical assumptions along with a user-supplied 'growth factor'.

Conclusions

We have shown that the automated design of parallel/pipeline machines is greatly simplified in the fixed-function case, and have argued that a sizeable proportion of dedicated DSP applications need no programmability in the high-performance, 'bottleneck' section of the system. The acceptance of the flexibility limitations of pipeline machines is also the key to the rapid and successful synthesis of efficient fixed-function machines from a simple blueprint, with minimal control overhead.

The small grain sizes yielded by deep pipelining allow design parameters to be finely-quantised, which in turn leads to the efficient tailoring of resources. At the same time, the ability to allocate resources efficiently over a wide parameter space allows synthesis techniques to be applied at all but the highest levels of abstraction, freeing the DSP systems designer to concentrate on the things he does best. Thus efficient architectural solutions to DSP problems may be produced in rapid timescales.

References

1. J. Allen, "Computer Architecture for Digital Signal Processing," *Proc. IEEE* **73** pp. 852 - 873 (May 1985)

2. S. G. Smith and R. W. Morgan, "High-level DSP ASIC Design Tool," *Proc. EURO ASIC '89* pp. 117 - 129 (Grenoble, France, January 1989)

3. S. G. Smith and R. W. Morgan, "Generic ASIC Architecture and Synthesis Scheme for Digital Signal Processing," *Proc. IEEE ICASSP'89, Paper 10V1.9* (Glasgow, Scotland, May 1989)

4. P. B. Denyer and D. Renshaw, *VLSI Signal Processing - A Bit-Serial Approach*, Addison-Wesley (1985)

5. R. I. Hartley and P. F. Corbett, "A Digit-Serial Silicon Compiler," *Proc. 25th ACM/IEEE DA Conf.* pp. 646 - 649 (Anaheim, CA, June 1988)

6. S. G. Smith and P. B. Denyer, *Serial-Data Computation,* Kluwer Academic Publishers (1988)

7. R. Jain et al., "Custom Design of a VLSI PCM-FDM Transmultiplexer from System Specifications to Circuit Layout Using a Computer-Aided Design System," *J. IEEE* **SC-21** pp. 73 - 85 (February 1986)

amount of product bits visible at the output of a multiplier. Local accuracy requirements may be met in this way. Fig. 1 shows a 16-bit data word (black) with 8 trailing zeros (dotted), resident in the 30-bit working word.

The acceptance of the limitations of pipelining allows arbitrary word-decompositions, in contrast to the conventional datapath approach, with its large grain-size and inflexible data format. Further flexibility may be introduced by multiplexing (feeding one processor from several sources) and/or duplicating (feeding several processors from one source), thus effectively compressing and/or expanding the task period in relation to the processor period.

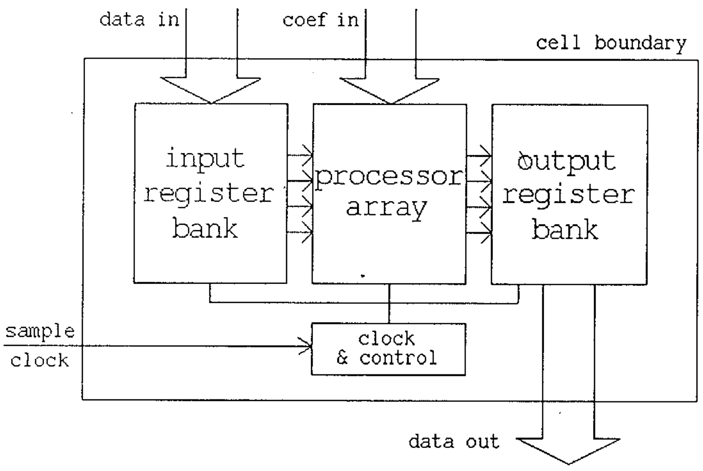

Figure 2: architectural template

Architectural Template

These decompositions occur inside the cell; from the outside it appears like a datapath, communicating along conventional bus-based lines. We call the external part the *parallel domain*. All compiled processors obey the architectural 'template' or 'blueprint' illustrated in Fig. 2. Input data arrive in sequence on a bit-parallel bus, and load up the register bank. When this process is complete and the register bank is full, the data block is transferred into the *serial domain* and transmitted, using the appropriate word-structure, through the processor. The register bank immediately starts filling up with the operands for the next computation. Results are captured in the output register bank, are transferred back into the parallel domain, and depart the register bank in sequential fashion. We refer to the process of transferring between domains as *corner-turning*. In the (usual) case where multipliers are programmable, coefficients arrive in sequence on dedicated buses. All multipliers are 'serial/parallel' [6], so no forward transmission of coefficients is required once loaded. As algorithm structure is naturally reflected in the data flow, control requirements in both parallel and serial domains are minimal.

Design Synthesis

One of the cornerstones of our approach is the conceptual separation of structure, throughput and accuracy in the design process. We intend the user to be able to concentrate on these issues one at a time, carrying the minimum of low-level mental baggage. This separation illustrates one of the fundamental differences mentioned earlier; throughput and accuracy requirements are inherent in DSP applications, and their specification is as important a part of the design process as that of processor function. While computer software function can be well-defined, throughput and accuracy are invariably a by-product of the design process, not an integral part of specification.

The user requires to have identified a core 'task' in the application, which needs dedicated hardware for execution. The structure of the processor is explicit in the icon schematic; we believe that, despite some limited success to date in DSP behavioural synthesis [7], a high-level structural interface is adequate for most users.

By filling in a simple menu sheet, the user assigns values to some important global 'guideline' attributes. These include the task rate (the number of times the machine must perform its task per second) input precision, and workspace (the expected working precision). Interface specifications are captured via menus, the main purpose here being to