Design of heterogeneous ICs for mobile and personal communication systems

Gert Goossens Ivo Bolsens Bill Lin Francky Catthoor

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

Abstract – *Mobile and personal communication systems form key market areas for the electronics industry of the nineties. Stringent requirements in terms of flexibility, performance and power dissipation, are driving the development of integrated circuits into the direction of heterogeneous single-chip solutions. New IC architectures are emerging which contain the core of a powerful programmable processor, complemented with dedicated hardware, memory and interface structures. In this tutorial we will discuss the real-life design of a heterogeneous IC for an industrial telecom application : a reconfigurable mobile terminal for satellite communication. Based on this practical design experience, we will subsequently discuss a methodology for the design of heterogeneous ICs. Design steps that will be addressed include : system specification and refinement, data path and communication synthesis, and code generation for embedded processor cores.*

1 Introduction

The mobile and personal communication systems market imposes a number of severe requirements on the IC design process. First of all, mobile and personal communicators are compact portable devices. To make this possible, the functionality of a complete system (e.g. a complete mobile terminal) has to be integrated on as few as possible ICs. Secondly, time-to-market is a critical factor in the design. On the one hand this means that solutions need to be flexible enough to include late specification changes or customer-specific features. This can be realised by making part of the chip *field* or *mask programmable*. On the other hand, it imposes a *modular design style*, in which hardware or software components can be reused successfully.

Last but not least, power consumption and silicon area are important cost functions. This may necessitate the design of *specialised data paths and memory organisations* for critical functions.

A good compromise between these different requirements can be made by designing *heterogeneous architectures*, which combine different architectural design styles on a single IC. In order to integrate such complex systems on a single ASIC, the realisation of powerful design technology in terms of design methods, macro-cell libraries and CAD tools is required. In this tutorial we describe a practical design and outline the problems and directions of a design methodology for the envisaged domain of applications.

The demonstrator application and its characteristics are described in Section 2. This leads to the definition of a heterogeneous IC architecture as explained in Section 3. The concept of a design environment for heterogeneous ICs is proposed in Section 4. The requirements for specific CAD components are elaborated in Sections 5 and 6 respectively.

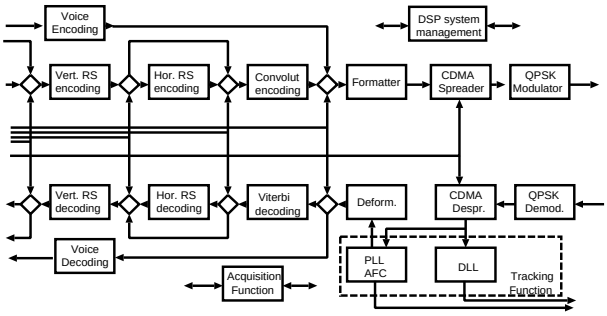

Figure 1: *Block diagram of the mobile terminal system.*

2 A reconfigurable mobile terminal for satellite communication

The demonstrator application discussed in this tutorial is a *reconfigurable mobile terminal* for satellite communication, that forms a key component in the *Mobile Services Business Network* (MSBN) and the *Micro LEO Messaging System* (MLMS), initiated by the European Space Agency [53].

MSBN will provide continental voice and data transmission between fixed and mobile earth stations, using a GEO stationary satellite. MLMS is a world wide bi-directional, store and forward messaging system, using LEO satellites.

The terminal is able to operate in a wide range of modulation and demodulation schemes : CDMA (code division multiple access) or non-CDMA, synchronous/asynchronous, QPSK, BPSK (quadrature or binary phase-shift keying).

The functional block diagram of the complete system is shown in Figure 1. It can be divided into three main parts, discussed next.

2.1 Low-throughput signal processing

This part includes the *voice encoding*, *channel encoding* and *formatting* functions in the transmission path, and the corresponding *de-formatting* and *decoding* functions in the receiver path. The functionality of this part is programmable, as indicated by the switches in Figure 1.

The mobile terminal can process both voice and data signals. In the former case, the voice coder compresses the input signal rate (64 kbit/s) up to a factor of 40. In the latter case, channel coding is applied to recover from transmission errors. Convolutional coding is used, optionally preceded by block coding (vertical and/or horizontal Reed-Solomon algorithms). The formatter organises the signal stream by inserting headers, synchronisation sequences and closing frames. The formatting of the signal stream is strongly influenced by the transmission mode. The typical output rate of the formatter is 10 kbit/s.

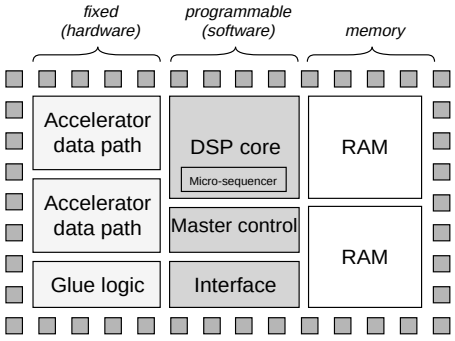

Figure 2: *Components of a heterogeneous IC architecture.*

2.2 High-throughput signal processing

This part includes the *spread-spectrum modulator* and *demodulator* and the *up conversion* to an intermediate frequency (IF). These functions operate at high throughputs up to 160 Mbit/s. However, the required flexibility within the envisaged application domain is low.

A direct-sequence *CDMA spreading* technique is used based on Gold codes, with a code length varying between 1 and 1023. *Modulation/demodulation* uses a QPSK or BPSK scheme. This part contains various functions like chip matched filtering, correlation and noise estimation. *Up conversion* is performed through a complex multiplication; the sine and cosine waveforms are generated with the Cordic algorithm. The IF is application dependent.

2.3 Control processing

In addition to the signal processing functions, which operate on signal streams with predefined rates, the mobile terminal contains several control functions that have to obey less stringent real-time constraints. Examples are : *DSP system management*, *synchronisation*, and the *man-machine interface*.

The DSP system management function forms a shell around the signal processing part, determining the mode of execution of the signal processing blocks. The synchronisation loop includes acquisition and tracking functions and carrier, chip and frame synchronisation algorithms.

All control functions can be subject to late specification changes, e.g. to use the system in different contexts, so that a flexible implementation is desired.

3 Heterogeneous IC architectures

The above characteristics of the functional components indicate the need for a heterogeneous IC architecture as shown in Figure 2 [21]. It combines : a *programmable component* in the form of a DSP- or microprocessor core that runs an application programme; one or more *hardware components* in the form of accelerator data paths and/or glue logic; and *memory, communication, and peripheral components*.

3.1 Processor core

This term refers to the data path and the micro-sequencer of a *programmable instruction-set processor*, integrated on the chip. In some cases, local data memories are included. The content of a core is predefined and cannot be changed.

In the case of control dominated systems, a *RISC microprocessor core* [2, 24] may be a good solution. For systems with extensive real-time signal processing, *fixed-point DSP cores* [33] are however a better choice. Compared to RISC microprocessors, DSPs have more arithmetic power and more specialised instructions.

A DSP core can have two possible origins. A first option is to extract it from a *commercial DSP processor*. Commercial DSP cores are becoming available from most DSP vendors. In this case, C-compilers are available to support the design process. However, for embedded applications, commercial DSP cores often consume too much power and silicon area. For these reasons, semiconductor groups in system industries prefer to develop more specialised DSP cores for specific application domains. These in-house cores are referred to as *application-specific instruction-set processors* (ASIPs). ASIPs result in less hardware overhead and power dissipation, and reduce the dependency from external suppliers. The disadvantage is that there is no commercial tool support for ASIPs.

3.2 Accelerator data paths

These data paths are added to speed up the execution of time critical functions of the system. This is important as we consider real-time applications. Accelerator data paths are constructed by selecting functional building blocks (like adders, shifters, multipliers) from a hardware library, and connecting them in a function-specific way. Examples are : dedicated data paths for Viterbi decoding, shaping filters and Cordic rotation for trigonometric functions.

Accelerator data paths can be equipped with a local controller, so that they act as accelerator processors that execute an algorithm independently from the control flow of the core processor. In this case, a run-time synchronisation of the different control threads (of the core and the accelerators) will be required. Alternatively, the accelerators may be included in the same control thread as the core processor. In our application, the first approach has been applied.

3.3 Memory, communication, and peripherals

In between the different data path and core processor components, hardware blocks are required to facilitate data communication and synchronisation.

Data communication between the different components occurs via bus or memory based channels [24]. The communication band-width is often limited by the available data ports of the core processor. Depending on its architecture, the core may communicate via parallel or serial I/O channels or according to a memory mapped scheme. Communication can be scalar based, as in audio, voice or simple data processing, or employ more complex data structures like video frames or speech frames. In the scalar case, simple buffering by means of registers or small FIFOs is usually sufficient [44, 31]. For more complex cases, communication may require large memory based buffers [12].

The different components in the heterogeneous IC architecture may use different *clocking schemes* with respect to each other and with respect to the external environment. Also, the programmable processor core(s) used and the external environment may already have a *predefined protocol scheme*. Therefore, synchronous and asynchronous I/O interfaces may be required to implement handshaking, pro-

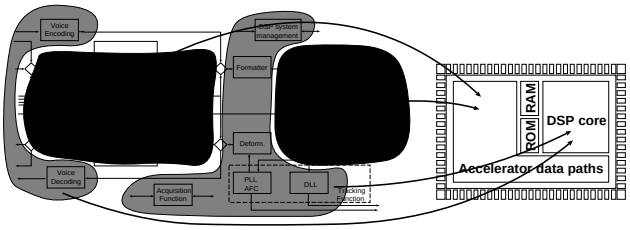

Figure 3: *Mapping of the mobile terminal system (cf. Figure 1) on a heterogeneous architecture.*

tocol control, and synchronisation functionalities [35, 51].

Finally the chip may contain special peripheral blocks such as A/D or D/A converters, JTAG interfaces, clock synthesisers and timers.

3.4 Architecture of the mobile terminal

Our *final goal* is to realise the digital functions of the mobile terminal as one single heterogeneous IC. This design will be used as an illustration of the heterogeneous architecture concept. The *partitioning* of the block diagram into *software and hardware parts*, to be implemented on a DSP core processor and on accelerator data paths respectively, is shown in Figure 3.

This partitioning is mainly governed by criteria of throughput and flexibility requirements. The *DSP core* will be used to implement most of the low-throughput and control functions : voice encoding and decoding, acquisition and tracking, as well as formatting and DSP system management. The channel encoding and decoding functions, as well as all high-throughput parameterised signal processing functions will be implemented as *accelerator processors* with their own local control. The latter include (de)spreading, chip matched filtering, correlation, frequency up/down conversion, noise estimation and clock synthesis.

Today, a *first prototype* of the reconfigurable mobile terminal is available, which essentially still is a *two-chips* design. The printed circuit board contains a TMS320C30 *DSP processor* and a *dedicated ASIC* [44], as well as some additional peripheral hardware. The ASIC implements all high-throughput parameterised signal processing functions of the mobile terminal. It interfaces with the DSP processor using memory mapped I/O. The ASIC has been synthesised using the CATHEDRAL-3 high-level synthesis tools [39]. It contains 400k transistors, operates at 40 MHz clock speed, and has an area of 186 mm^2 in a 1.2 μm technology. The layout of this chip is shown in Figure 4.

Starting from the existing solution, a further integration is currently taking place in two steps :

- In a first step, the ASIC of Figure 4 is being extended with an Advanced Risc Machines microprocessor core [2], to execute all control functions of the mobile terminal.
- The final version will include a DSP core, and corresponds to the solution of Figure 3.

4 Design methodology

The complexity of the complete mobile terminal chip will be between 0.5 and 1 million transistors. Industrial design

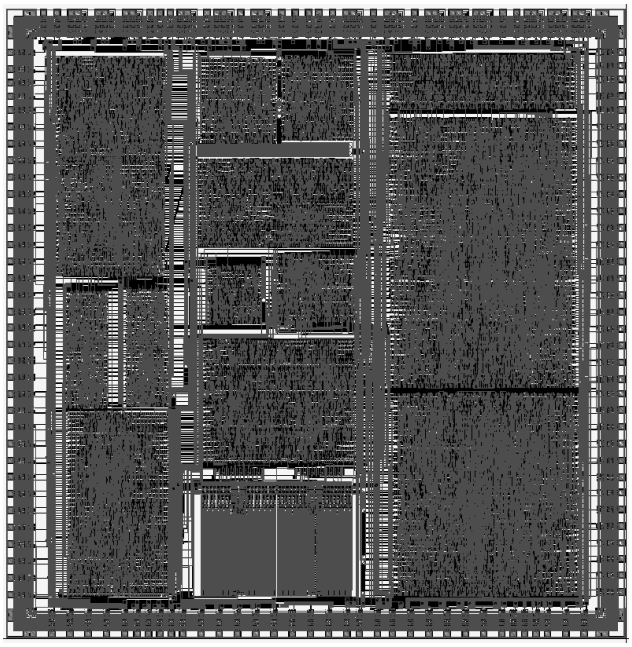

Figure 4: *Layout of the spread-spectrum modulation/demodulation chip.*

times allowed for this kind of applications are typically less than one year. It is clear that this complexity can only be mastered by means of adequate design technology.

Based on previous observations and on the experience with the mobile terminal design, a design environment for heterogeneous ICs is currently under development at IMEC. The basic flow is shown in Figure 5 [9]. The concepts are outlined next.

4.1 Integration of component compilers

Different tools can be used to design individual components of a heterogeneous IC (e.g. glue logic, accelerator data paths, machine code for a core processor, asynchronous interfaces). The design environment should be conceived in such a way that these *component compilers* can be plugged in. This reduces the complexity of the design problem (divide and conquer), and allows to adapt the overall environment in a dynamic way to an evolving "market" of tools.

Component compilers *can* be commercial tools. A number of synthesis technologies are becoming commercially available and should be used wherever possible. However, in many cases these tools still lack flexibility and optimality with respect to relevant cost functions. Moreover, the abstraction level of their input specification is often too low to allow for exploration of system alternatives. Due to these limitations, *new research* is needed to develop more powerful component compilers. This will be explained in Section 6.

4.2 System specification front-end

Component compilers start from a local description of the component's behaviour. If a designer would only rely on component compilers, (s)he would have to provide component specifications without being able to guarantee the overall consistency of the design. Therefore, a *global system*

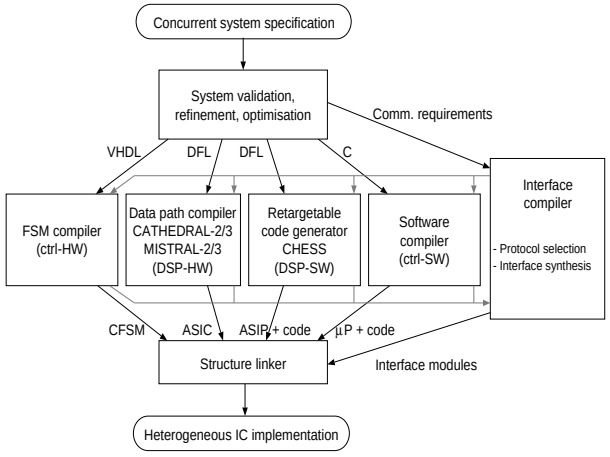

Figure 5: *Outline of an IC design environment for mobile and personal communication systems.*

specification front-end is needed that defines the interaction between the components.

This front-end should be based on *formal models* at a sufficiently high abstraction level, be suited for *verification*, and form the basis for *refinement and partitioning* over the components. More details will be provided in Section 5.

4.3 Communication synthesis and linkage

Data and control communication between components have to be taken into account. Based on the system specification, specific communication protocols between components may be determined, which result in design constraints for component compilers. After the components have been synthesised, buffers and communication paths can be synthesised. Finally, a linkage of the different hardware components into a global netlist, and a linkage of the software components into a global executable, are required.

In the next sections, we will illustrate the above concepts in the design of the mobile terminal application. At the same time we will derive more specific requirements for the different design tools.

5 System specification front-end

5.1 Specification model

The system specification model should allow to express *concurrency*, to *mix signal processing with reactive control*, and to express *timing constraints*.

5.1.1 Concurrency

Existing high-level synthesis systems and software compilers for the signal processing domain usually schedule the application statically into one *single thread of control*. This paradigm is reflected in the choice of the specification language or model, which will only allow to express a restricted form of concurrency.

For example, *Silage* (used by the CATHEDRAL and HYPER synthesis systems), is a *synchronous* language in which all operations are synchronised to basic time frames that correspond to a "sample period" [19]. PHIDEO uses a synchronous stream-based model [56]. *DFL*, used by MISTRAL, in addition allows to synchronise operations to loop iterations [14]. Software compilers usually start from a *sequential language* like *C*, in which concurrency is either not allowed or restricted to the internals of a "basic block".

Conceptually, the mobile terminal is partitioned by the designer in coarse functional entities that can operate concurrently and autonomously. The behaviour of these entities is very complex; some parts are elegantly described using a data flow paradigm whereas others are typically control dominated. We call the various entities of the system "*processes*". Processes can be implemented using different component targets varying from software to highly pipelined data paths.

When permitted by timing constraints, concurrent processes can *sequentialised* during the design process. Alternatively, they can also be implemented as *concurrent threads of control*. In the case of the mobile terminal we mostly used the latter option, because of the asynchronous nature of some processes (e.g. the keyboard monitoring) and because of unrelated signal processing rates (e.g. of the correlators and the chip matched filters).

These examples show that the system specification model should support *concurrent processes*, which are asynchronous with respect to each other, but which may communicate data or control information.

5.1.2 Mixed signal processing and reactive control

High-level synthesis has traditionally focussed either on signal processing (using synchronous data flow languages) or on reactive control (using sequential languages), but not on the *interaction* between both. This interaction is however present in any system.

For example, the tracking and acquisition algorithms in the mobile terminal are modelled as complex finite-state machines that monitor the results of the correlator functions, which in turn are most naturally expressed in a data flow language.

5.1.3 Timing and ordering constraints

A final requirement is that timing constraints may have to be specified. Examples are the specification of a *relative delay* between two operations, of a *rate constraint* for a process that is executed periodically, or of a *response time* to an external event in case of a reactive system. Timing constraints are often specified as *intervals*. Also *ordering constraints* between the production and consumption of complex signals (e.g. video frames) have to be specified to reduce buffer costs.

With the mobile terminal design, stringent timing constraints occured in the specification. These prevented us from sharing hardware between the chip matched filters in the transmitter and receiver paths.

5.1.4 Existing models

Many system specification languages have originated from the software community. Asynchronous imperative languages such as *CSP* and *Occam* [25, 29] are well-suited for describing distributed processes that are loosely coupled. *Esterel* and *Statecharts* [6, 23] are synchronous imperative languages, dedicated to the description of reactive systems. Processes are then tightly coupled and deterministic. Each of the above classes is unable to handle the problems for which the other class is intended.

Data flow models have also been used for system specification, as for example in PTOLEMY [11], GRAPE-2 [8], SPW, COSSAP, and the DSP-STATION [14].

The usage of hardware description languages, such as VHDL, for system description has been examined by many researchers. The main conclusions from these experiments seem to indicate that VHDL has many syntactical and semantical obstacles to elegantly and unambiguously define complex systems [50, 41].

Recent work has tried to unify several of the above characteristics [7, 9, 41, 28]. However, further research is required on system specification in order to arrive at a working environment for automatic synthesis of complex systems in software and hardware.

5.2 Validation

Validation of the global system specification is essential as it is the starting point for synthesis. Two approaches are of interest : *model checking* and *simulation*.

In the case of model checking, a number of properties can be formally defined to check the correctness of the design (e.g. liveness, maximum buffer size requirements, etc.). These properties can then be checked for the given specification, independent of input stimuli.

With simulation, the functional behaviour is checked for specific input stimuli only. Efficiency of simulation is an important concern. In the PTOLEMY environment [11], different simulators, each optimised for a specific type of behaviour, can be combined to gain efficiency. PTOLEMY has the ability to simulate a heterogeneous specification that is composed from different "domain-specific" specification models. The PTOLEMY framework has facilities to permit the interaction between different domain-specific simulators.

The behaviour of the ASIC of Figure 4 has been specified and simulated using PTOLEMY. The composing functions were described using the *synchronous data flow* (SDF) domain. In order to model timing concepts such as latency, the SDF functions were embedded in a *discrete event* (DE) domain.

5.3 Architectural refinement

The purpose of architectural refinement is to derive the input specification for each component compiler. This is a non-trivial process that is not yet well formalised. Below, two specific aspects of the refinement process are discussed.

5.3.1 Process refinement

As mentioned in Section 5.1, designers naturally think of a system in terms of communicating processes, containing operations that logically belong together. However, the grouping of operations into processes specified by a designer, is not necessarily efficient or even feasible from an implementation view point. Process refinement is the task of *reorganising the process structure*, in such a way that each resulting process can be fed separately to a *component compiler* (e.g. a high-level synthesis or a software compilation tool).

We assume that a component compiler is able to *statically schedule* the internals of a process at compilation-time. This implies that the processes fed to a component compiler must be fully deterministic. Therefore, during process refinement all sources of non-determinism (i.e. reactive behaviour like wait-loops for external events, global conditions, preemption of processes) must be isolated at the interface between processes. The relative scheduling of the processes with respect to each other is determined at run-time, depending on external events. Run-time scheduling can be implemented in various ways, ranging from a hard-wired handshake circuit between components to a run-time kernel using interrupt mechanisms implemented in the processor core.

Whereas deterministic processes can be sent at once to a component compiler, the designer may also choose to partition them further into smaller processes which are scheduled with respect to each other at run-time. This may be advantageous in case of large processes, to reduce the complexity of the component compilation process, or to make the design process more modular.

5.3.2 Component selection and partitioning

A selection of architectural components has to be made out of a library. The library may contain predefined macros (e.g. DSP cores, memories, interface modules), but also abstract macros of which the content will be synthesised by a component compiler (e.g. data paths, logic). Furthermore, the system functions have to be partitioned into parts to be implemented in hardware and in software.

How to formally represent a *library of macros* with their *behaviour*, is an open research issue. Initial work has focussed on restricted types of macros, such as specific hardware blocks [1] or ASIPs [17, 54]. First approaches to *hardware/software partitioning* have been presented in literature (e.g. [16, 22, 26]). These papers are concerned with a generic partitioning problem and use abstract library models and cost functions. So far, it has been difficult to evaluate their practical use.

On the other hand, designers usually have a good insight in partitioning strategies for particular designs. In the case of the mobile terminal application, programmability requirements and functional characteristics of the different functions naturally lead to the hardware/software partitioning shown in Figure 3. The hardware functions were then further partitioned into several accelerator processors, synthesised by CATHEDRAL-3. This partitioning was mainly driven by area and power optimisations. For example, the correlation calculations in the receiver and the chip matched filter require completely different data paths in terms of complexity and throughput. Therefore, the hardware of every accelerator processor is tuned towards the digital functions it has to execute and the throughput it has to reach. The use of these distributed processors allows for the reduction of their clock frequencies by matching them to the throughput requirements of each partition. As a consequence, this reduces the power consumption.

This experience suggests that partitioning be considered as an *interactive task*. In such an approach the compilation environment should support *interactive specification* of partitioning alternatives, combined with capabilities for *fast evaluation* of the quality of a solution [30]. The latter can be achieved by extending the component compilers in such a way that they can be called in a "fast" mode, in which only first order optimisations are carried out. This results in an iterative partitioning strategy with the designer in the loop [28, 41].

6 Component compilers

6.1 High-level synthesis

A number of hardware synthesis technologies are commercially available today. Commercial tools do not address the system design problem as a whole but rather concentrate on specific sub-problems.

The introduction of *logic synthesis* [10, 48] has been very successful. In the context of heterogeneous architectures, logic synthesis is very useful to design custom logic blocks. However, the technology provides insufficient hardware sharing and timing optimisation capabilities to successfully design accelerator data paths, and the abstraction level of the input description is still low. *High level synthesis* [37] is an emerging technology that is being introduced commercially, especially for signal processing applications [40]. It has raised the abstraction level from the register-transfer to the architectural level. However, these environments need further extensions in order to include sophisticated architectural specialisations required for *high throughput* signal processing functions.

High throughput signal processing functions can be represented by regular computation-intensive signal flow graphs, and often contain multi-dimensional signals of large size and dimensions. The number of operations that have to be performed per sample period is much larger than the available number of clock cycles. These characteristics of the application domain heavily influence the choice of an architectural style and synthesis approach [39, 46, 56].

The following tasks are crucial to efficiently synthesise dedicated accelerator data paths, that meet throughput constraints with minimal hardware cost :

1. *Data path optimising transformations :* The initial signal flow graph can be transformed to improve the eventual synthesis result [59, 43, 27]. Typical transformations are : the selection of multiple-precision computation schemes, algebraic laws, control flow transformations, etc.
2. *Regularity extraction and assignment :* In order to exploit the inherent regularity of the signal flow graph, operations can be grouped into clusters. Different clusters with a high degree of similarity can subsequently be assigned to the same multi-functional data path, with a minimal control overhead [20, 47]. This assignment process typically utilises compatibility measures between clusters, and is subject to timing and memory constraints.
3. *Data path definition :* Based on the functionality of the assigned clusters, the final data path composition is determined, including the realisation of local controllers [39, 46]. During this optimisation, area cost is the main objective function. In order to meet the throughput constraints, bit-level pipelining and retiming of the resulting data paths, as well as buffer tree optimisation are important optimisation tasks.

The CATHEDRAL-3 high-level synthesis system, developed in our lab, is based on the above concepts. It is targeted to high-speed real-time signal processing functions, which have a low potential for time multiplexing [39, 57], and extends the older CATHEDRAL-2 methodology [52] which is embedded in the commercial MISTRAL-2 environment of EDC/Mentor [40]. CATHEDRAL 3 has been used intensively in the mobile terminal design. Eight different accelerators have been synthesised that deal with the high-speed signal processing functions as defined in the previous sections.

6.2 Software compilation

A DSP or microprocessor core is a basic component of a heterogeneous architecture. For high-performance applications, a DSP (in particular an ASIP) is preferred over a microprocessor core. However, software compilers for DSPs are less developed than for microprocessors. In this section we will focus on software compilation for ASIP cores.

Ideally, a CAD environment for ASIP architectures consists of three components [42] : an *ASIP synthesiser*, a *retargetable code generator*, and an *instruction-set simulator*. The ASIP synthesiser reads the high-level specifications of a *set of representative functions* for the application domain (e.g. in *C*, *Silage*, ...), extracts common features, and produces an ASIP description in a processor description language (e.g. ISPS [5], nML [17], ...). The retargetable code generator reads the high-level specification of *one application* as well as the full *ASIP description* using the processor description language mentioned previously. It subsequently maps the application onto the ASIP, producing low-level *assembly or microcode*. Finally, the instruction-set simulator allows to *simulate a given machine code description* on a *DSP specified in the processor description language*.

The *retargetable code generator* is a key component of the system. The user can retarget the code generation process to a new architecture by simply changing the ASIP description. In this way the system designer can quickly explore the use of different available DSPs for a given application. Moreover, a retargetable code generator can also be called iteratively during the ASIP synthesis process, to evaluate the impact of a modification of the data path or instruction set on the code quality for representative functions.

The problem of code generation has been addressed extensively by the compiler community during the early eighties [18, 38, 3]. Although several *basic techniques* like code optimisation, code selection, register allocation, and compaction are well understood today, the problem of *retargetability* has not been fundamentally solved.

Moreover, since the mid-eighties, a drastic evolution occurred in terms of processor architectures. On the one hand, RISC and VLIW processors have developed at a rapid pace, and are reasonably well supported by compilers today [15, 24]. On the other hand, (fixed-point) DSP processors have become increasingly popular in the embedded systems community. However, due to the initial small market share of DSPs, compiler technology did not support the latter evolution very well. As a result, industrial telecom design groups are today still spending a lot of time in manual assembly coding. With the advent of mobile/personal communication and multi-media systems, an important growth of the DSP market can be predicted, and high-quality code generators are needed. The following research issues need to be addressed :

1. *Code quality :* The main architectural difference between a DSP processor and a microprocessor is in the *register structure* : whereas most microprocessor cores have a large central *general-purpose register file*, a DSP typically contains a number of distributed *special-purpose registers* (or very small register files).

Recent code generators for ASIPs use sophisticated register allocation methods which aim at exploiting these specialised register structures [32, 34]. These methods are essential to produce high code quality.

2. *Retargetability :* The key issue in the development of a *retargetable* code generator is the *definition of a single, formal processor model* that can cover a wide range of different targets, contains all relevant programmer's information (e.g. instruction-set, register structure, pipelining, etc.), and can be used in every phase of the code generation process. Promising approaches to ASIP modelling have been described in [36, 17, 54].
3. *Multi-tasking :* The DSP or microprocessor core can be used to implement a run-time kernel taking care of run-time processes scheduling (see Section 5.3.1). A run-time kernel can perform multi-tasking, without the overhead of a full operating system. Several run-time kernels for DSP cores are commercially available (e.g. [49, 58]). Limitations of these tools are : their inability to guarantee that hard real-time constraints will be met, and the lack of automatic retargetability to different processor cores. Additional research is needed in this area.

In our lab, a retargetable code generator called CHESS is currently under development, based on the above concepts [54, 32].

6.3 Storage and communication synthesis

The different hardware and software components in the heterogeneous IC architecture may interact and communicate with each other and the outside world in complex ways by transferring information and synchronising on their execution. A communication architecture composing of storage components and bus structures must be synthesised [31]. Several key problems must be solved.

One problem is the allocation of the necessary *storage components* needed for data communication between processes and for internal data storage within processes. Simple audio, voice or control algorithms often use scalar signal types, which can be stored in small register files or FIFOs [44, 31]. In more complex algorithms, including video applications, data exchange requires large memory-based storage. In this case the memory size and number of memory transfers can often be reduced by transforming the original specification, which reduces the area and power dissipation [55, 12, 4]. In addition to synthesising the storage architecture, the mapping of the data transfers on to the actual storage locations and busses must be performed.

In the CATHEDRAL environment, memory synthesis techniques have been developed. These techniques have been applied to the mobile terminal design, which resulted in a drastic reduction of the memory usage for the voice coding/decoding and Viterbi decoding functions.

Another problem is the synthesis of (concurrent) *control behaviours for coordinating the communications.* When programmable processor cores are used, they usually have already a "built-in" communication protocol and clocking scheme that may be incompatible with the other components in the heterogeneous IC architecture. Moreover, the outside world may also impose a different communication protocol and clocking discipline. To implement handshaking, protocol control, and synchronisation functionalities, synchronous and asynchronous I/O interfaces may be required [35, 51], or use can be made of a run-time kernel (see Section 6.2). The correct design of interfaces that satisfy complex timing requirements constitutes a major design bottleneck in complex systems.

7 Conclusion

In this tutorial we introduced a methodology for the design of heterogeneous ICs for mobile and personal communication systems. The methodology relies on *powerful compilers* for different components of the IC, complemented with an *interactive front-end* for system specification and refinement. These concepts have been illustrated with an ongoing real-life design in the area of satellite communication.

With abstraction levels increasing, design methodologies will by nature become more and more *application-driven*. This observation, together with the necessity to encapsulate existing low-level tools, will lead to an increased requirement for *open design environments*.

Acknowledgements The ideas presented in this paper are the result of ongoing research work at IMEC, to which many people are contributing. We acknowledge contributions from Lieven Philips, Jan Vanhoof, Stefan De Troch, Karl Van Rompaey, Dirk Lanneer, Werner Geurts, Filip Thoen, Lode Nachtergaele, and Chantal Ykman. This work is sponsored by the European Space Agency through the project Scades-II and by the European Commission through the projects Esprit 2260, 3280, 6143, and 9138. Finally, we acknowledge the interaction with Carl Van Himbeeck of SAIT Systems.

References

[1] R.P. Ang, N.K. Dutt, "A representation for the binding of RT-component functionality to HDL behavior", *Proc. IFIP Conf. Hardw. Descr. Lang.*, pp. 251–266, Ottawa,April 1993.

[2] "*ARM7DM data sheet*", *Doc. No. ARM-DDI-0010-F*, Advanced Risc Machines Ltd., Cambridge,May 1994.

[3] T. Baba, H. Hagiwara, "The MPG system : a machine-independent efficient microprogram generator", *IEEE Trans. Computers*, Vol. C-30, No. 6, pp. 373–395, 1981.

[4] F. Balasa et al., "Dataflow-driven memory allocation for multi-dimensional signal processing systems", *Proc. IEEE Int. Conf. Comp.-Aided Design*, Santa Clara,Nov. 1994.

[5] M.R. Barbacci, "Instruction Set Processor Specifications (ISPS) : the notation and its applications", *IEEE Tr. Comp.*, Vol. C-30, No. 1, pp. 24–40, Jan. 1981.

[6] G. Berry, G. Gonthier, "The Esterel synchronous programming language : design, semantics, implementation", *Science of Comp. Prog.*, Vol. 19, No. 2, pp. 87–152, 1992

[7] G. Berry et al., "Communicating reactive processes", *Proc. 20th ACM Principles of Prog. Lang.*, 1993.

[8] G. Bilsen et al., "Static scheduling of multi-rate and cyclo-static DSP-applications", *IEEE Workshop VLSI Signal Proc.*, La Jolla,Oct. 1994.

[9] I. Bolsens et al., "User requirements for designing complex systems on silicon", *IEEE Workshop VLSI Signal Proc.*, La Jolla,Oct. 1994.

[10] R.K. Brayton et al., "Multi-level logic synthesis", *Proc. IEEE*, Vol. 72, No. 2, pp. 264–300, Feb. 1990.

[11] J.T. Buck et al., "PTOLEMY : a framework for simulating and prototyping heterogeneous systems", *Int. J. Computer Simulation*, January 1994.

[12] F. Catthoor et al., "Global communication and memory optimizing transformations for low power signal processing systems", *IEEE Workshop VLSI Signal Proc.*, La Jolla,Oct. 1994.

[13] G. de Jong, B. Lin, "A communicating Petri net model for the design of concurrent asynchronous modules", *Proc. 31st ACM Design Autom. Conf.*, pp. 49–55, San Diego,June 1994.

[14] "DSP Architect – DFL – User's and Reference Manual", EDC/Mentor Graphics Corp., Leuven,1993.

[15] J.R. Ellis, "BULLDOG : *a compiler for VLIW architectures*", MIT Press, Cambridge,1986.

[16] R. Ernst et al., "Hardware-software co-synthesis for microcontrollers", *IEEE Design & Test of Computers*, Vol. 10, No. 4, December 1993.

[17] A. Fauth et al., "Generation of hardware machine models from instruction set descriptions", *VLSI Signal Processing VI*, pp. 242–250, 1993.

[18] M. Ganapathi et al., "Retargetable compiler code generation", *Computing Surveys*, Vol. 14, No. 4, pp. 573–593, 1982.

[19] D. Genin et al., "DSP specification using the Silage language", *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Proc.*, pp. 1057–1060, Albuquerque,April 1990.

[20] W. Geurts et al., "Memory and data-path mapping for image and video applications", *Application-driven architecture synthesis*, pp. 143–166, Kluwer, Boston,1993.

[21] G. Goossens et al., "Integration of signal processing systems on heterogeneous IC architectures", *Pres. at 6th IEEE Int. Workshop High Level Synth.*, Dana Point,Nov. 1992.

[22] R.K. Gupta, G. De Micheli, "Hardware-software cosynthesis for digital systems", *IEEE Design & Test of Computers*, Vol. 10, No. 3, pp. 29–41, Sept. 1993.

[23] D. Harel et al., "Statemate : a working environment for the development of complex reactive systems" *IEEE Tr. Software Eng.*, Vol. 16, No. 4, April 1990.

[24] J.L. Hennessy, D.A. Patterson, "*Computer architecture : a quantitative approach*", Morgan Kaufmann Publ. 1990.

[25] C.A.R. Hoare, "*Communicating Sequential Processes*", Prentice Hall, 1985.

[26] T.B. Ismail et al., "Interactive system-level partitioning with Partif", *Proc. European Design & Test Conf.*, pp. 464–468, Paris, Feb. 1994.

[27] M. Janssen et al., "A specification invariant technique for operation cost minimisation in flow-graphs", *Proc. 7th IEEE Int. Symp. High-Level Synth.*, pp. 146–151, Niagara-on-the-Lake,May 1994.

[28] A. Jerraya, K. O'Brien, "SOLAR : an intermediate format for system level design and specification", *Pres. at Int. Workshop Hardw./Softw. Co-Des.*, Grassau,May 1992.

[29] G. Jones, M. Goldsmith, "Programming in Occam 2", *C.A.R. Hoare Series in Computer Science*, Prentice Hall.

[30] A. Kalavade, E.A. Lee, "A hardware/software codesign methodology for DSP applications", *IEEE Design & Test of Computers*, pp. 16–28, Sept. 1993.

[31] T. Kolks et al., "Sizing of communication buffers for communicating signal processors", *VLSI Signal Processing VI*, pp. 426–434, 1993.

[32] D. Lanneer et al., "Data routing : a paradigm for efficient data-path synthesis and code generation", *Proc. 7th IEEE Int. Symp. High-Level Synth.*, pp. 17–22, Niagara-on-the-Lake,May 1994.

[33] E.A. Lee, "Programmable DSP architectures : Part I & Part II", *IEEE ASSP Magazine*, Dec. 1988 and Jan. 1989.

[34] C. Liem et al., "Register assignment through resource classification for ASIP microcode generation", *Proc. ACM/IEEE Int. Conf. Comp.-Aided Design*, San Jose,Nov. 1994.

[35] B. Lin, S. Vercauteren, "Synthesis of concurrent system interface modules with automatic protocol conversion generation", *Proc. ACM/IEEE Int. Conf. Comp.-Aided Design*, San Jose,Nov. 1994.

[36] P. Marwedel, "Tree-based mapping of algorithms to predefined structures", *Proc. IEEE/ACM Int. Conf. Comp.-Aided Design*, pp. 586–593, Santa Clara,Nov. 1993.

[37] M.C. McFarland et al., "The high-level synthesis of digital systems", *Proc. of the IEEE*, Vol. 78, No. 2, pp. 301–318, Feb. 1990.

[38] R.A. Mueller et al., "Global methods in the flow graph approach to retargetable microcode generation", *Proc. 17th Microprog. Workshop*, pp. 275–284, 1984.

[39] S. Note et al., "Cathedral III : architecture driven high-level synthesis for high throughput DSP applications", *Proc. 28th ACM/IEEE Design Autom. Conf.*, pp. 597–602, San Francisco,June 1991.

[40] S. Note et al., "A low-power, low-voltage dedicated DSP implementation of a GSM baseband processor with DSP-Station", *IEEE Workshop VLSI Signal Proc.*, La Jolla,Oct. 1994.

[41] S. Narayan et al., "System specification and synthesis with the SpecCharts language", *Proc. ACM/IEEE Int. Conf. Comp.-Aided Design*, pp. 266–271, Santa Clara,Nov. 1991.

[42] P.G. Paulin et al., "DSP design tool requirements for embedded systems : a telecommunications industrial perspective", *To be publ. in J. VLSI Signal Proc.*, 1994.

[43] M. Potkonjak, J. Rabaey, "Optimizing resource utilization using transformations", *Proc. IEEE Int. Conf. Comp.Aided Design*, pp. 88–91, Santa Clara,Nov. 1991.

[44] L. Philips et al., "Silicon integration of digital user-end mobile communication systems", *Proc. Int. Conf. Communications*, pp. 212–216, Geneva,May 1993.

[45] L. Philips et al., "Silicon synthesis of a flexible CDMA/QPSK mobile communication modem", *DSP Applications*, Jan. 1994.

[46] J.M. Rabaey et al., "Fast prototyping of datapath-intensive architectures", *IEEE Design & Test of Computers*, pp. 40–51, June 1991.

[47] D.S. Rao, F.J. Kurdahi, "Partitioning by regularity extraction", *Proc. 29th ACM/IEEE Design Autom. Conf.*, pp. 235–238, Anaheim,June 1992.

[48] E.M. Sentovich et al., "Sequential circuit design using synthesis and optimization", *Proc. IEEE Int. Conf. Comp. Design*, pp. 328–333, Oct. 1992.

[49] "*SPOX – The DSP operating system*", Spectron Microsystems, Santa Barara,1992.

[50] M.B. Srivastava, R.W. Brodersen, "Using VHDL for high level mixed mode system simulation", *IEEE Design & Test of Computers*, pp. 31–41, Sept. 1992.

[51] P. Vanbekbergen et al., "A generalised state assignment theory for transformations on signal transition graphs", *J. VLSI Signal Proc.*, pp. 101–116, Feb. 1994.

[52] J. Vanhoof et al., "*High-level synthesis for real-time digital signal processing*" Kluwer Ac. Publ. Boston,1993.

[53] C. Van Himbeeck, "The use of CDMA in European mobile satellite communication systems", *Proc. IEEE Int. Symp. Spread Spectrum Techn. and Applic.*, Oulu,July 1994.

[54] J. Van Praet at al., "Instruction set definition and instruction selection for ASIPs", *Proc. 7th IEEE Int. Symp. on High-Level Synth.*, pp. 11–16, Niagara-on-the-Lake,May 1994.

[55] M. van Swaaij et al., "Automating high-level control flow transformations for DSP memory management", *VLSI Signal Processing V*, pp. 397–406, IEEE Press, New York,1992.

[56] W.F.J. Verhaegh et al., "Modeling periodicity by PHIDEO streams", *Pres. at 6th IEEE Int. Workshop High-Level Synth.*, Dana Point,Nov. 1992.

[57] S. Vernalde et al., "Synthesis of high throughput DSP ASICs using application specific datapaths", *DSP Applications*, June 1994.

[58] "*Virtuoso Classico user manual*", Intelligent Systems International, Linden,1993.

[59] R. Walker, D. Thomas, "Behavioral transformation for algorithmic level IC design", *IEEE Transactions on Comp.-Aided Design*, Vol. 8, No. 10, pp. 1115–1128, Oct. 1989.

[60] C. Ykman-Couvreur et al., "Concurrency reduction transformations on state graphs for asynchronous circuit synthesis", *Proc. Int. Workshop Logic Synth.*, Lake Tahoe,May 1993.