1 Introduction

Modern telecommunication systems are rapidly increasing in design complexity. Elaborate network management is needed to support a wide variety of broadband multimedia services. The design of these systems is often a collaborative effort taking place at multiple sites. Such complex systems require a combination of both hardware and software components in order to deliver the required functionalities at the desired level of processing performance and programmability. These hardware and software components must be designed concurrently to minimize time to market.

Example telecom network applications include system components for ATM-based broadband networks, mobile network infrastructures to support GSM-based cellular communication, SONET and SDH based networks, and interactive video-on-demand servers. These and other emerging applications in similar areas are among the fastest growing segments of the systems industry today. These market sectors are being fueled by new advances in optical fiber transmission, broadband switching, wireless communication, video compression, and deep submicron process integration. At the same time, an increasingly open market world-wide is fueling global competition where considerable benefits can be attained by the first companies to bring a product to the market.

In current industrial design practice, formal specification formalisms are mainly used to model each hardware or software component separately, using a formalism largely dedicated to model either hardware or software, not both. System design, preceding hardware/software partitioning, proceeds still to a large degree in an informal manner. The system specification handed over to hardware and software designers is usually an informal description in the form of a report document, and is often susceptible to ambiguous interpretations. Significant time is spent in interpreting and understanding these specifications, and still a lot of mismatches occur between the expected and the actual behavior of designed components, leading to a lengthy system integration and test phase after component design. Currently, the system integration and test phase can account for nearly 50% of the design cycle for typical telecom network designs in system houses. There is a clear need for suitable system level specification formalism and validation techniques to overcome this problem.

Once the hardware and software parts are identified, more formal specification models and design techniques are employed. In the case of software, C or C++ is used. In the case of hardware, a register transfer language like VHDL or Verilog is used. The different hardware and software components of a complete system are specified and synthesized separately, which often introduces implementation mismatches, in addition to specification mismatches. Small changes at the system level often require very substantial changes to the hardware or software models of the components. While automated code generators exist for producing machine code from C or C++, and automated hardware synthesis tools exist for producing hardware implementations from VHDL or Verilog, there is a general lack of tools to bridge the gap from the global system level design specification to these traditional hardware and software design methods. Moreover, there is a general lack of coherent design methodologies for structuring the system design flow.

In this presentation, we outline a design methodology for the design of hybrid software/hardware systems that are typically found in telecom network applications. This methodology is based on the results of an investigation and evaluation of an actual industrial system application for ATM (Asynchronous Transfer Mode) based broadband networks at Alcatel Bell [14]. This case study is described in Section 2. As a result of this investigation, we have developed a system design methodology based on a concurrent object-oriented programming model as the system behavioral specification formalism. This model is described in Section 3.

For implementation, we provide automated design tools for transforming the system level model into the traditional levels of design entries for hardware and software implementation. In particular, conventional C++ [12] and VHDL [8] models are generated for the parts of the system to be implemented in software and hardware, respectively. New system-level synthesis functionalities are required to achieve these automations, which are outlined in Section 4. It is important to note that traditional hardware and software design methods, mostly commercially available, are used in our design flow, thereby not reinventing new solutions where adequate ones already exist. The work outlined in this presentation is embodied in a system design compiler called Matisse.

2 Case Study: a Connectionless Router for ATM

As a case study, we have investigated an actual industrial ATM (Asynchronous Transfer Mode) based broadband network application developed by Alcatel Bell. The application is a user transparent connectionless
router called the ACTS (Alcatel Connectionless Transport Server) [14] that provides the necessary functions for the direct provision and support of data communication between geographically distributed computers or local area networks (LANs) over a broadband ATM-cell based transport network.

ATM [4] is a fast packet switching transfer mode that supports high-speed integrated services by splitting all communications into equal 53-byte cells. These cells can be used to carry every kind of information, be it computer data, video, or voice. By using small cells to transfer data, the technology enables networks to provide the flexible multiplexing needed to support a wide variety of traffic, ranging from high to low bandwidths, and from bursty to steady bit rates. In addition, ATM is characterized by a connection oriented mode of operation.

Since local area networks are connectionless oriented, the ATM network needs to be augmented with special user transparent connectionless routers to provide connectionless services. This is the role of the ACTS. This router can for example support Switched Multi-megabit Data Services (SMDS) [1] in a B-ISDN environment. The SMDS protocol features packets of variable length, which are transported by one or more fixed-size ATM cells. The communication service offered is connectionless, meaning that a sender does not have to set up a connection prior to sending data, but can start transmitting data immediately. The router is responsible for storing incoming data, finding out where to route them, and forward them. This situation is depicted in Figure 1.

In its current implementation, the ACTS consists of several custom ASICs and a programmable processor for executive control. In future implementations of similar systems, the functionalities of several of these processors will be integrated into a single VLSI chip, and some functionalities previously implemented in hardware will migrate to software to exploit the increasing processing performance of emerging embedded microprocessors.

3 The Programming Model

In this section, we outline the essential concepts and rationale behind the programming model that we use in Matisse. Detailed language syntax is beyond the scope of this presentation.

From our application experience on a number of industrial broadband telecom network applications, we can conclude that behavior of these applications are best characterized by control-flow dominated data processing algorithms. Expressiveness in terms of control flow is essential. The algorithms and the data structures that they operate on are usually tightly coupled. In many cases, the data represents the real conceptual core of the application rather than the algorithms. Support for concurrency is essential, but parallelism tends to be at the task-level and is usually coarse to medium scale. Although the implementation target is often a mixture of software and hardware, telecom network applications are often conceived at the top-level from a software perspective.

Figure 1: A user transparent connectionless router interconnecting LANs to the ATM network.

Given the above considerations, we have followed an object-oriented approach to parallelism as the basis of our programming model, as we believe it provides a natural model of concurrency for the applications considered: objects encapsulate processes and (remote) member function calls encapsulate interprocess communication. In addition, we believe object-oriented features like encapsulation, inheritance, and polymorphism are invaluable in any large scale development. Specifically, we use an active object semantic model that combines parallelism with object oriented principles. We have implemented the programming model on top of the widely used object-oriented programming language C++ [12] by introducing minimal syntactic extensions to it.

An active object differs from a passive object (as found in standard C++) in that it additionally encapsulates a process, as well as state and operations. The process allows an active object to execute its member functions in parallel with the activity of the rest of the program. In a typical program, only a small number of objects will be active, providing the parallel structure of the program. The majority of objects will be standard passive objects, existing as members of active objects or in data structures managed by active objects. This provides support for coarse to medium scale parallelism.

Communication between active objects is via the use of (remote) member function calls. A member function call to an active object behaves in the same way as a standard C++ member function call, in that the caller is suspended until the called member function terminates. This rendezvous semantics provides synchronization between active objects that are operating concurrently. Argument passing (in both directions) is the principle mechanism of data transfers.
Since we have implemented our model on top of C++, the language also supports mechanisms for expressing (abstract) data types, assignment and arithmetic operations, and control-flow statements like if-then-else, switch-case, and for-while loops. Also, we can leverage upon the wide corpus of existing software compilation and runtime support tools for the software implementation path, and on existing execution environments [5, 11] and debugging tools [7] for early conceptual validation of the system specification.

4 Design Methodology

To implement the proposed programming model, several key problems must be solved. They are outlined in this section.

4.1 Software/hardware partitioning

Software/hardware partitioning is driven by the designer by means of directives. System partitioning decisions are driven by factors that are often not easily quantifiable. These factors often cannot be easily formulated into simple cost equations that an automated tool can optimize. Therefore, we believe that such decisions should be best left to the designer who is in a better position to make such judgements. We instead try to automate the refinement steps to lower level hardware and software tools as these steps tend to be the most problematic for designers.

4.2 Software/hardware communication

After the user-driven partitioning step, different parts of the specification will be assigned to software or hardware. Since we are building an application-specific solution, different software programmable processors from different vendors may be used, thus resulting in a heterogeneous hardware/software architecture. The implementation of such heterogeneous embedded architectures, while ensuring that the different system components are correctly integrated together, is a surprisingly difficult task. Designers spend an enormous amount of time on this task, partly in understanding how to interface to the different processors being used, how to get the hardware and software parts to communicate correctly, and how to synchronize between different components operating on different clocks. This is a highly error-prone task, often responsible for many low-level implementation mismatches, leading to a lengthy test phase after implementation.

Our approach to this problem is based on an orchestrated combination of architectural strategies, parameterized libraries, and CAD tools for automating low-level design tasks that are error prone and time consuming. More specifically, tools are provided to implement the communication structure between the software and hardware components, thus solving the system integration and architecture co-implementation problems. The communication structure will support point-to-point communication, shared bus based communication, and shared memory based communication. In addition, the communication and architecture synthesis tools will also support the embedding of real-time kernels for multi-tasking support on a software processor. Although there are many existing commercial real-time kernels available, they are not readily portable to application-specific embedded architectures. To solve this problem, tools are also provided to automate the low-level configuration of real-time kernels so that they can be used with the communication structures and embedded architecture that we generate. These tools are embodied in a system called Symphony. Some of the problems have been addressed in [10].

4.3 Software implementation

For software implementation, we use existing C++ compilers to produce machine code for the embedded processor target. However, these compilers are aimed at compiling conventional C++ code, which is sequential. To manage concurrent tasks running on the same processor, we make use of ultra-lightweight real-time microkernels, which provide bare minimum services for task scheduling and interprocess communication. These services are provided at the level of library function calls. In contrast to distributed runtime environments for workstations, which are many orders of magnitude larger, these microkernels can be less than 2K in code size. Currently, we make use of a solution based on a commercial real-time kernel called Virtuoso [15].

To make use of such real-time microkernels and conventional C++ compilers, automated compilation tools are provided in our design flow to transform our concurrent specification model to a level of design entry suitable for these tools.

4.4 Hardware implementation

For the hardware implementation path, designers will be able to start from the same system specification model instead of having to re-specify and re-interpret the system specification into lower level hardware description languages like VHDL. This will help to eliminate time and effort spent in understanding and interpreting an informal specification, which is again often susceptible to ambiguity and mismatches.

Specifically, we use behavioral VHDL as the level of design entry to current commercial hardware synthesis tools. This VHDL is automatically generated from the parts of the system specification that are assigned to custom hardware. Several first generation commercial behavioral synthesis tools are appearing on the market. Examples include the Behavioral Compiler from Synopsys and Mistral-2 DSP Compiler from Mentor. These compilers incorporate many of the key results from the last decade of high-level synthesis research. We are currently targeting the Synopsys Behavioral Compiler [13] as the backend in our design flow, which already provides basic functionalities like dependency graph construction, scheduling, resource and register assignment, and datapath and controller synthesis.
Despite the availability of these high-level synthesis functionalities, there is still a significant gap from our system level model to the behavioral VHDL model. The object model has to be translated into VHDL's package and entity models. Also, provisions for dynamic data structures inherent in the object-oriented programming model must be supported. Behavioral synthesis tools today only support simple data types like static records and arrays. Dynamic data structures are not supported. Instead, the designer has to explicitly map them in terms of memory locations and memory operations. Hardware behavior must be explicitly provided to manage the runtime behavior of the memory. For applications in the telecom network domain, this can be a significant limitation. Moreover, the abstract notion of data structures and having algorithms operate on them, the basic premise of object-oriented programming, is partially lost. In our refinement step to behavioral VHDL, we automate the allocation of physical memory, the mapping of dynamic data structures to memory locations, the refinement of data structure accesses to primitive memory operations, and the synthesis of dynamic memory management behavior for the runtime management of the memory. Some of these issues have been addressed in [6].

4.5 Heterogeneous co-design and co-simulation

In this presentation, we have proposed a common system specification model for capturing control-flow dominated data processing applications, such as those found in the telecom network domains. Based on this specification, lower level hardware and software specification models are generated for further synthesis. This is complementary to hardware/software co-design flows from system data flow models for DSP applications (e.g., [9]), which are also based on a common top-level representation. However, there are design situations where it would be best to mix different formal system design models. This is one manifestation of heterogeneity.

Even for co-design flows where a common formal system design model is used, such as the one described here, heterogeneity will naturally manifest itself by the virtue of the fact that different implementation technologies will be used. For example, C or C++ models will most likely be used for software targets, and VHDL or Verilog models will most likely be used as a lower level design entry for hardware targets. In general, there are many intermediate steps from system-level specification to the final implementation, and different computational models are required at different levels of the refinement trajectory.

Therefore, a heterogeneous co-design environment is needed that can integrate multiple computational domains at different levels of abstraction within a single environment. We are currently developing one such environment called CoWare [3]. Ptolemy [2] is for example another heterogeneous co-design environment.

A key problem that must be solved by any heterogeneous co-design environment is co-simulation. Simulation remains the primary workhorse for functional verification. Co-simulation has two distinct manifestations of heterogeneity: different models of computation must be made to operate together for designs that mix models of computation; different models of designs must interact for designs that mix implementation technologies. For example, designers must be able to co-simulate ASIC hardware blocks together with software running on programmable components. In the case of processor models, co-simulation may make use of timing accurate or only instruction-set accurate simulation models, depending on the stage of refinement.

Another key problem is the encapsulation of different design tools. For example, the Matisse system described in this presentation, the Symphony system for hardware/software communication, the Synopsys compiler, and C++ compilers for different processors used, are all being integrated into our CoWare environment.

Acknowledgments.

This work has been supported in part by the Belgian Flemish funding agency IWT under the project HASTE-C. This is a joint project between IMEC and Alcatel-Bell. The author would like to thank M. Genoe, G. Van Wanau, E. Huykens, and L. Cloeters from Alcatel-Bell for their contributions and support in this project. The work outlined in this presentation is the result of ongoing research work at IMEC, to which many people are contributing. The author is grateful to all their contributions.

References


