SoftSONIC: A Customisable Modular Platform for Video Applications

Tero Rissa[1], Peter Y.K. Cheung[2], and Wayne Luk[1]

[1] Department of Computing, Imperial College London
[2] Department of Electrical and Electronic Engineering, Imperial College London
{tero.rissa,w.luk,p.cheung}@imperial.ac.uk

Abstract. This paper presents the Customisable Modular Platform (CMP) approach. The aim is to accelerate FPGA application development by raising the level of abstraction and facilitating design reuse. The solution is based on network of Nodes, communicating using packet-based protocol. The approach is illustrated using SoftSONIC, a CMP for video applications. Our approach promotes modularity and design reuse by having multiple interoperable layers of design abstraction, while supporting advanced development and verification methods such as mixed-abstraction execution and efficient system-level simulation based on Transaction Level Modelling. The platform provides domain-specific abstractions and customisations of various elements such as communication protocols and topology, enabling exploitation of data locality and fine- and coarse-grain parallelism. The benefits of our approach is demonstrated using SoftSONIC for development of several real-time HDTV video processing applications.

1 Introduction

Design cost has become a critical issue as a result of the exponential rise in silicon and design complexity. This paper presents the Customisable Modular Platform (CMP) approach, which aims to lower the design cost of complex digital systems. Reconfigurable platforms are utilised to alleviate some of the problems caused by *manufacturing* Non-Recurring Engineering (NRE) and *silicon* complexity, such as mask cost, probe card, signal integrity, power and clock management, manufacturing and process variability [1]. The novel aspects of the CMP approach aims to solve *system complexity* issues and those *silicon complexity* issues that are not directly solved by using reconfigurable platforms.

This area has attracted much attention during the past few years. Examples include platform-based system-level design [2,3], automated communication refinement [4] and network-on-chip paradigm [5]. The SystemC community has been the main contributor to the work on Transaction Level Modelling (TLM) [6], alongside with SpecC [7].

Our work differs from these previously proposed approaches in the following ways. (1) Exploitation of reconfigurable platforms: automated mixed simulation and hardware execution, rapid prototyping and run-time reconfigurability. (2)

J. Becker, M. Platzner, S. Vernalde (Eds.): FPL 2004, LNCS 3203, pp. 54–63, 2004.

Domain specific design approach, which entails domain-specific information for design optimisation. (3) Platform customisation to minimise overhead. (4) Combining verification and reuse of Intellectual Property (IP) within a single framework. In addition, while previous work on TLM has been focused on bus-based embedded mircroprocessor systems. Our work offers a hardware oriented approach with a selection of communication topologies. Finally, CMP can be used for abstracting and generalising the Sonic-on-a-Chip approach [8].

The rest of the paper is organised as follows. Section 2 introduces the CMP approach. Section 3 presents SoftSONIC. Section 4 and 5 cover functional and design abstractions. Section 6 presents implementation results. Section 7 looks at current and future work, and concludes the paper.

2 Customisable Modular Platform Approach

Advanced multi-million gate FPGAs such as Xilinx Virtex II Pro and Altera Stratix and can significantly reduce the manufacturing NRE cost. However, the design NRE cost dominates the overall cost and time [9]. The CMP approach is developed to address this issue.

The CMP approach originates from the platform-based design concept [1,2, 10]. A CMP is a domain specific design methodology, capturing the underlying abstractions and design rules in the form of a virtual platform. It contains a set of design rules that characterise the target architecture. These rules are determined partly by the physical architecture and implementation technology, for instance hardware/software or FPGA/microprocessor. Many of the rules are 'virtual', enabling higher-level abstraction of architectural and physical issues. This higher-level abstraction accelerates the mapping of the application to the target architecture, and facilitates the design and verification process.

The CMP defines different *abstraction layers* for the functional abstractions. Each abstraction layer is defined in terms of information that they entail, for example numerical precision, timing behaviour, and resource usage. The CMP adopts four distinct abstractions: Algorithmic, Virtual TLM, Physical TLM, and Register Transfer Level (RTL).

The penalty in speed, area and power consumption is reduced by two factors. (1) The platforms are *application domain* specific. By narrowing the scope, platform abstractions can be chosen in accordance to known efficient implementations. Examples of such application domains are video processing [15], wireless networks [11] and communications [12]. (2) The platforms are *customisable* within the domain for a particular application and implementation, to support efficient architectural exploration and realisation. In general, architecture-level design decision have potentially much greater impact on the final performance than low level implementation optimisations.

Often, the first step in our approach is the selection of a domain specific CMP, characterised by the communication and computation. The specific scope of the chosen CMP enables efficient modelling and optimisation. Within SoftSONIC, our CMP for video applications, the customisable aspects include the level of

coarse and fine grained parallelism, communication packet structures, communication protocols and topologies, packet data sequencers, and hardware/sofware partitioning. For example, the achievable parallelism can be dictated by the inherent parallelism of the algorithm or can be bound by communication bandwidth. The platform can be customised to meet the maximum performance in either case, for example with the selection of inter or intra device communication.

The main goal of a CMP is the separation of design concerns. In addition to the traditional separation of computation from communication and the functionality from architecture, the CMP approach provides the separation of verification of functionality from verification of performance. When applicable, this separation reduces the overall verification effort. The purpose is to avoid functional re-verification after design exploration and customisation for performance. High-level functional verification is also simplified, as implementation details need not be considered.

The CMP approach reduces the *observed complexity* to alleviate the overall design and verification effort in the following ways. **IP Reuse.** The platform concept enhances IP reusability by specifying an unambiguous interface between communication and computation. Furthermore, the modularity of the approach makes IP reuse possible in several levels and abstractions. **Restricted Design Space.** Application developers can focus on using the optimised facilities offered by a particular CMP, while CMP developers can focus on creating efficient CMPs by optimising and generalising designs. **High-Performance System Simulation.** The CMP approach uses TLM for the high-level block models. Simulation speed at transaction level can be 100 times faster than RTL simulation [13]. **Mixed-Abstraction Execution.** When a new block is developed, the behaviour of the rest of the system can be obtained by the fastest available model, reducing the simulation time. In addition, when using reconfigurable platforms, it is possible to use the actual hardware of available blocks in real-time execution. This can be several orders of magnitude faster than software simulation. **Co-Simulation.** Co-simulation between different design tools is difficult. The CMP approach alleviates the co-simulation problem by specifying a restricted, unambiguous platform communication mechanism. **Intrinsic Support for Run-Time Reconfiguration.** The possibility to update a product after deployment enables: (a) support for new features, (b) post-delivery design optimisation, and (c) adaptation to run-time conditions.

3 SoftSONIC CMP

SoftSONIC is a CMP for video processing applications. It is inspired by the UltraSONIC board-level architecture [14]. The main difference between SoftSONIC and UltraSONIC is that UltraSONIC was developed to support easy integration of hardware modules, whereas the aim of SoftSONIC is to reduce the observed design complexity of the system in order to speed up the design and verification.

In UltraSONIC the system is constructed from discrete PIPE hardware modules. In SoftSONIC these modules, called Nodes, are virtual, and are separated from an actual implementation. For example, one FPGA can contain several

Nodes, or one Node can span several FPGAs. UltraSONIC has a specific set of protocols to support a particular board architecture. SoftSONIC, in contrast, can be customised to target different board architectures and applications.

From the functional perspective, SoftSONIC is similar to UltraSONIC; both cover high-bandwidth video image processing applications [14,15]. SoftSONIC also adds or improves the following functionalities. **Multi-cycle pixel processing.** The processing of a pixel can take an arbitrary number of cycles. SoftSONIC supports **pixel-level parallelism** in three ways. (1) At high level, the number of Nodes that run in parallel can be varied. (2) At Node level, the data packets can be processed in parallel. In FPGA implementations of the Nodes, the packets are stored in block RAM, which supports parallel accessing. (3) At Node level, the clock frequency of each Node can be customised to meet design requirements. **Fork and join of data streams.** Forking of a stream is important, when new data are extracted from a stream while the original stream is needed later in the processing pipeline. The joining of streams is used, for example, in creating composite images, like blue/green-screen composition, logo insertion, and gradual stream transition. **Support for non-image data.** In addition to pure image data such as RGB 4:4:4[:4] and YUV (4:2:2, 4:2:0) in lines, windows and scattered pixels, SoftSONIC supports image-related information such as image metadata, compressed image data, and audio. **Shared memory random access.** Memory random access is enhanced with the use of memory-server Nodes and random-access data packets.

These improvements illustrate the additional functionalities that SoftSONIC supports. However, the main goal of SoftSONIC is to reduce application development time, which is achieved by exploiting the advances in Section 2.

4 Functional Abstraction

The purpose of functional abstraction is to specify the platform in a concise and easily adoptable manner. Functional abstractions include the model of communication and the model of computation. These functional abstractions separate the timing inside a Node from its external interfaces. This is an important factor in separating the communication from computation to obtain the benefits listed in Section 2.

Model of Communication. Communication in SoftSONIC is based on packets that contain a header and a payload. The packet payload contains *structured* data: in addition to explicit information, the type of packet implicitly indicates how the data should be interpreted. In SoftSONIC, basic packet types are pixel-based image data, structured to lines, windows or scattered pixel and address-data pairs for random memory access.

The information structure between Nodes is restricted to packets. The communication is unidirectional: once the packet is sent by the producing Node it cannot be altered, and after consumption the packet is deleted. Packet scheduling follows bounded First-In-First-Out (FIFO) buffers in a process network. Both attempts to read an empty buffer and to write to a full buffer are *blocking* operations.

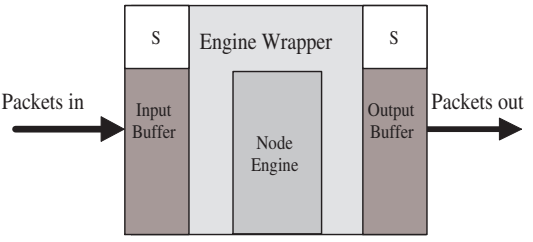


Fig. 1. A SoftSONIC node

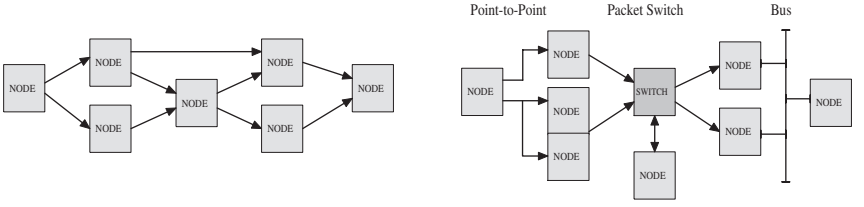
One exception to packet communication is Configuration Register (CReg) communication. An element outside the communication network, for example the host computer or on-board user interface, can access CRegs inside the Nodes without using packet-based communication. The CReg interface is intended for low-bandwidth, time-independent sporadic communication involving, for example, information about Node parameters and user control. The CReg interface implementation is application and environment specific, but must follow CReg update rules, which due to space restrictions are not explained here.

Model of Computation. User-defined functions take place inside the Nodes. The general structure of a SoftSONIC Node is depicted in Fig. 1. A Node consists of: the Input and Output Buffers, the Buffer status information 'S-box', the Node Engine Wrapper, and the Node Engine.

The Input and Output Buffers contain FIFOs for packet communication. The S-box contains the status information of the buffer and handles the arbitration of the buffer. The S-box interface handles the transition between abstractions layers, development environments, and clock domains. The Node Engine Wrapper is a customisable data sequencer for input and output buffers. It can be customised, for example, to access packets in parallel sequential streams or to have application specific scan pattern. It starts consuming the input packet when all required input buffer(s), output buffer(s) and the Node Engine(s) are available. The CReg interface is implemented within the Node Engine Wrapper and the actual registers locate inside the Node Engine.

User-defined computation is performed inside the Node Engine. The Node Engine can use only the information included in the input buffer(s), the packet header, internally stored data and the CReg data. A Node can contain one or more engines wrapped by a single Node Engine Wrapper.

The Nodes are connected to each others with channels. The channels have an identical interface to the input and output buffers, and have the same firing rules as computation Nodes. The only difference is that a channel does not change the contents of the packet in any way; it just transports a packet from one buffer to another. It is also possible to connect Nodes together in a point-to-point manner. In this case one Node's output buffer becomes another's input buffer.



(a) Virtual TLM (b) Physical TLM

Fig. 2. Transaction Level Model Layers: Virtual and Physical

5 Design Abstraction Layers

SoftSONIC is divided into four distinct design abstraction layers. Each layer is defined in terms of timing, implementation and data accuracy, the incoming and resulting system model and the target design questions the layer aims to answer.

Algorithmic: Executable Specification. The algorithmic layer does not use SoftSONIC functional abstractions. The purpose of this layer is to provide an executable specification for the system under development. In this layer, the whole system functionality is described using high level software languages such as C/C++ or Matlab. The executable specification does not contain information about the timing and performance of the system. However, this layer must contain information about limited bitwidth effects and the maximum error bound.

Virtual TLM: Parallel Functionality. Layer 2, Virtual TLM, is the first SoftSONIC specific layer. The purpose of this layer is to re-specify the contents of the Algorithmic layer using the SoftSONIC functional abstraction. The main design effort is in mapping the functionality to individual Nodes and extraction of parallelism in the application. Node communication is modelled using unlimited communication resources in bandwidth and the number of connections, but with bounded input and output buffers. In other words, the packet always takes zero time to arrive from producer to the consumer. Bounded input and output buffers enable mixed-abstraction execution. The bounding of the FIFOs can be made arbitrarily at this point, as it does not affect the functionality of the system. An example of a Virtual TLM is depicted in Fig. 2(a).

In this layer, sending and receiving a packet is modelled as a single transaction in a queueing process network. Coarse grain parallelism at Node level can be explored, but there is limited feedback of resource usage as unbounded resources are assumed. In other words, it is possible to explore whether the algorithm can be parallelised, but it is impossible to determine whether such a system is feasible for real-life implementation. The verification task is system-to-system: the functionality of the Algorithmic layer must correspond to that of Virtual

TLM. In this layer, both communication channels and computation Nodes are described using untimed TLM models.

Physical TLM: Design Exploration. In the Physical TLM layer, the functional description of the Nodes remains unchanged. The focus is on resource allocation, design exploration, and customisation. As the functionality does not change, the verification task of this layer involves purely the *validation of the performance requirements* in speed, area, power, etc. The separation of functional and performance verification simplifies the task, and makes it easy to support a systematic approach to the problem. This can guarantee 'correct by construction' functionality when carrying out architectural design exploration.

Design exploration can be divided into two parts: communication and Node implementation analysis. Communication exploration examines the choice of different communication protocols, media and throughput versus resource usage. Choices can be made between on-chip and off-chip resources, packet sizes and types. A crucial decision is the size of the input and output buffers, as these can have significant impact on performance and resource usage. In Node implementation analysis, different levels of coarse and fine grain parallelism are explored. Also, if the decision has not been made earlier, Node-level software/hardware partitioning is carried out.

In this layer, the final packet sizes and types, communication and implementation media are allocated and buffer sizes re-examined. Thus performance analysis in terms of throughput can be estimated by modelling the individual throughput of the Nodes and communication system. These figures can be obtained from the lower RTL layer, or estimated. The communication is described using timed TLM models. As mentioned before, the computation Nodes remain unchanged; they are still modelled with untimed TLM models.

From the Virtual TLM layer description in Fig. 2(a), one possible outcome of design exploration and customisation is illustrated in Fig. 2(b). The communication of the first two Nodes is handled by point-to-point communication, such that the output buffer of the producing Node is the input buffer of the consuming Node. One of the Nodes is replicated to enhance Node-level parallelism. The second communication choice is to utilise a packet switch for the communication and finally, a bus is utilised.

RTL: Path to Implementation. The RTL layer implementation begins by generating the communication description from the Physical TLM layer. The communication acts as a wrapper for the RTL implementation. For simulation, the higher-level description of the communication can be used. This layer is focused on to the computation in the Nodes and the communication remains unchanged. The RTL description of the Node engines can be done by using a synthesisable language like RTL SystemC, HandelC, VHDL or Verilog. The verification of the Nodes is made easier by the independence of the Nodes and the ability to re-use the verification environment. The verification task is Node-to-Node when we move from Physical TLM layer to RTL layer. This is in contrast to system-to-system verification, which is the case when we move from Algorithmic layer to Virtual TLM layer. This also leads to opportunity for the use of advanced verification methods like assertion-based verification and formal

Table 1. Comparison of performance in Thermal Camouflage effect

Implementation	Clock Seed [MHz]	Throughput [Frames/s]	Speed-Up Factor (compared to SW)
Software (PC)	2400	1.6	1.00
VHDL simulation (PC)	2400	0.0006	0.0004
Mixed VHDL and SoftSONIC	2400/133	0.02	0.01
SoftSONIC	133	64	37.85
Parallel SoftSONIC	133	128	75.70
UltraSONIC	66	32	18.88
Normalised SoftSONIC	66	32	18.78

methods. The reduced scope of the verification problem increases the feasibility of these methods.

6 Results

This section compares the performance of various implementations of SoftSONIC against UltraSONIC and software simulations. We have developed an experimental hardware implementation which involves a real-time HDTV 1920 by 1080 RGB 4:4:4 10 bit/channel (SMTPE 372M – "Super2K") video effect application. The system creates a 'thermal camouflage' effect used to visualise semi-transparent objects. Similar effect has been used, for example, in the movies Predator (1987) and Hollow Man (2000). This application consists of 6 Nodes: Packet Source and Sink, 3x3 Blur Filter, 2D 3x3 Sobel Filter, Image Differentiator, and finally the 'Lens Effect' Node that produces a special effect to areas where the foreground image is different from the background image. The lens effect is created by varying the refraction according to the intensity of the edges.

One implementation of SoftSONIC is customised to the application and to the Xilinx Virtex-II Pro 50 FPGA by selecting packets of one line with buffer size of 2. This way, eight Block RAMs in 512x36 bit mode can form one buffer. With this setup, it is possible to have eight parallel pixel reads and writes. In this application, maximum throughput is determined by external memory access. Buffer size of 2 offers optimal performance/area tradeoff, as one buffer can be read while the other is being written, without significant pauses in processing. As all the functions in the application involve stream processing without non-deterministic components, point-to-point communication is the optimal selection for communication. Sometimes, the performance could be further optimised by maximising the individual clock speed of the Nodes, but in this case the maximum clock speed of 133MHz is dictated by the ZBT SRAM interface. Clock rates could also have an effect on the power consumption, but this is not currently being evaluated.

We consider seven different implementations, and the performance results are summarised in Table 1. The software implementation is the C++ Algorithmic level description of the system as a DirectShow filter running on a dual Athlon PC at 2.4GHz with 2G bytes of memory. Software simulation is based on Mod-

Table 2. SoftSONIC Node performances

Kernel	BRam	1 engine			2 engines			4 engines			8 engines		
		Slice	MHz	fps	Slice	MHz	fps	Slice	MHz	fps	Slice	MHz	fps
Invert Colours	8	160	289	139	183	295	282	228	315	604	405	296	1137
Image Diff	16	264	277	132	313	270	259	445	279	535	809	267	1025
Alfa Blend	16	364	175	83	554	167	160	951	160	307	1868	146	560
3x3 Noise Filter	24	1162	201	95	1773	202	191	3056	200	380	5525	220	836
2D 3x3 Sobel	24	1578	200	94	2736	202	191	4999	185	351	9389	140	532

elSim SE Plus 5.7f running on the above PC. In the mixed VHDL and SoftSONIC implementation, only the Lens Effect Node is simulated, and the other parts of the design is running on the FPGA. The Lens Effect Node would most likely be the only one that needs to be built from scratch, as the other Nodes are common image processing kernels. In the Parallel SoftSONIC implementation, the Packet Source and Sink produce/consume packets in double rate, and there are two Node Engines in each processing Node to enhance parallelism. The application does not have inter-line dependencies, so the upper limit of the parallelisation is bound only by the available memory or memory bandwidth. The UltraSONIC implementation contains Xilinx Virtex 1000E FPGAs using UltraSONIC protocols. Finally, in order to enable fair comparison between SoftSONIC and UltraSONIC, the Normalised SoftSONIC is an implementation on UltraSONIC hardware.

From the results it can be seen that the SoftSONIC implementations provide significant speedup compared to the software implementation. The Normalised SoftSONIC implementation is only slightly less efficient than the UltraSONIC implementation, indicating that the performance penalty is not significant. The main overhead is the higher usage of Block RAMs for input and output buffers.

Table 2 illustrates individual SoftSONIC Node performances, without concerning external memory access. The results are obtained after place and route, but without I/O buffers, as reported by Xilinx ISE 6.2.01i. For synthesis, Synplify Pro 7.2.2 is used. For each kernel, there are four separate implementations, which indicate 1, 2, 4, and 8 parallel engines inside a Node. Naturally, the faster implementations consuming more area, has more parallelism. All the implementations are automatically generated according to the level of engine parallelism. Because of the used abstractions, the engines can simply be replicated in order to accommodate the parallelism. However, as all these implementations use before mentioned 8 BRAM I/O buffers size, the data sequencers are different in each implementation in order to handle serialisation and de-serialisation of the data. Although, in the case of window processing this is non-trivial task as parallel engines have overlapping data, also these are automatically generated.

The table shows that very high frame rates can be achieved by using the SoftSONIC Nodes, and it is likely that the performance limitation will come from external memory access. In addition high performance of the kernels, the integration of kernel-Nodes to applications is greatly facilitated, as the Node interface guarantees interoperability.

7 Summary

We have described the Customisable Modular Platform approach for rapid application development and optimisation of reconfigurable designs. Our approach is illustrated using the SoftSONIC platform. Opportunities for customising SoftSONIC are discussed, and it is shown that SoftSONIC can produce many implementations rapidly without significant overheads. Current and future work includes refining the SoftSONIC model, automating the customisation process, and evaluating our approach using complex applications.

References

1. Lysaght, P.: FPGAs as meta-platforms for embedded systems. In: Proc. IEEE Int. Conf. on Field-Programmable Technology. (2002)
2. Keutzer, K., Newton, A.R., Rabaey, J.M., Sangiovanni-Vincentelli, A.: System-level design: orthogonalization of concerns and platform-based design. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems **19** (2000)
3. Ferrari, A., Sangiovanni-Vincentelli, A.: System design: traditional concepts and new paradigms. In: Proc. Int. Conf. on Computer Design: VLSI in Computer and Processors. (1999)
4. Abdi, S., Shin, D., Gajski, D.D.: Automatic communication refinement for system level design. In: Proc. Design Automation Conference. (2003)
5. Benini, L., Micheli, G.D.: Networks on chips: A new SoC paradigm. IEEE Computer (2002)
6. Grötker, T., Liao, S., Marting, G., Swan, S.: System Design with SystemC. Kluwer Academic Publishers (2002)
7. Gajski, D.D., Zhu, J., Domer, R., Gerstlauer, A., Zhao, S.: SpecC: Specification Language and Methodology. Kluwer Academic Publishers (2000)
8. Sedcole, P., Cheung, P.Y.K., Constantinides, G.A., Luk, W.: A reconfigurable platform for real-time embedded video image processing. In: Proc. Int. Conf. on Field-Programmable Logic and Applications. (2003)
9. Semiconductor Industry Association: International Technology Roadmap for Semiconductors. (1999, 2001, 2003)
10. Lysaght, P.: Future design tools for platform FPGAs. In: Proc. Symp. on Integrated Circuits and Systems Design. (2003)
11. Tuan, T., Li, S., Rabaey, J.: Reconfigurable platform design for wireless protocol processors. In: International Conf. on Acoustics, Speech, and Signal Processing. (2001)
12. Spivey, G., Bhattacharyya, S.S., Nakajima, K.: A component architecture for FPGA-based, DSP system design. In: IEEE Int. Conf. on Application-specific Systems, Architectures and Processors. (2002)
13. Kogel, T. et. al.,: Virtual architecture mapping: A systemc based methodology for architectural exploration of system-on-chip designs. In: Proc. Int. Conf. on Systems, Architectures, Modeling, and Simulation. (2003)
14. Haynes, S.D., Epsom, H.G., Cooper, R.J., McAlpine, P.L.: UltraSONIC: A reconfigurable architecture for video image processing. In: Proc. Int. Conf. on Field-Programmable Logic and Applications. (2002)
15. Haynes, S.D., Stone, J., Cheung, P.Y.K., Luk, W.: Video image processing with the SONIC architecture. IEEE Computer **33** (2000)