Vesa Lahtinen

Design and Analysis of Interconnection Architectures for On-Chip Digital Systems

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB104, at Tampere University of Technology, on the 21st of June 2004, at 12 noon.
Abstract

The past decades have seen the rise of integration density in integrated circuits making it possible to design whole systems on a single chip. The previously designed interconnection architectures for multiprocessor systems cannot directly be applied in on-chip systems since they require a different type of a cost-performance trade-off. This is why the interconnection architectures of system-on-chips (SoC) are such a problem. This Thesis presents a novel on-chip interconnection architecture scheme as a solution to this problem.

The Thesis begins with a theoretical comparison of interconnection architectures. These comparisons do not take the targeted application into account. The studies show that the choice for an architecture is almost always a trade-off between the connectivity and the implied cost.

Based on this initial research, some practical comparisons are made. A circuit-switched bus and crossbar as well as a packet-switched hierarchical bus and a 2-D mesh are implemented and simulated with different traffic patterns. These practical studies give more accurate information about the interconnection architectures in realistic test cases.

All this prior work leads to the development of a novel heterogeneous IP block interconnection (HIBI) scheme that is used to interconnect processing nodes in future SoCs. The scheme is based on a hierarchical bus structure, but allows the utilization of specialized structures when needed. This type of a heterogeneous interconnection architecture can be modified according to the requirements of the application. The implementation of the scheme requires a wrapper component. In this Thesis, the development and verification of this wrapper is presented. The wrapper is then used to implement test systems to verify the applicability of the scheme.

The scheme is found to be versatile due to the arbitration and architecture reconfiguration schemes, low in implementation complexity due to the local bus interconnections, and high in performance because of the utilization of a combination of hierarchical bus and specialized structures as global interconnect.
Preface

The work presented in this Thesis has been carried out in the Institute of Digital and Computer Systems, Department of Information Technology, Tampere University of Technology in 1999 - 2004.

All the conducted research has been supervised and guided by Prof. Hämäläinen for which I am very grateful. The preliminary assessment of the Thesis has been conducted by Dr. Risto Suoranta and Dr. Pasi Kolinummi. The opponents at the public defense of the dissertation are Prof. Manfred Glesner and Dr. Pasi Kolinummi. A very warm thank you to all of you for your insightful comments. In addition, I would like to thank Profs. Jukka Saarinen and Jarmo Takala for their guidance during the start of my academic career.

I would also like to thank the personnel of the Institute. In particular, Dr. Kimmo Kuusilinna has had a huge influence on the outcome of this work. The co-workers and co-authors of this work also deserve extra credit. Tero Kangas M.Sc, Jouni Riihimäki M.Sc, and Erno Salminen M.Sc: Thank you for the numerous discussions and debates about subjects related and unrelated to the work presented here. I would also like to acknowledge the work done by the newcomers of our group Mr. Kalle Holma and Mr. Ari Kulmala. Finally, I would like to thank Jari Nikara M.Sc, Tuomas Järvinen M.Sc, and Perttu Salmela M.Sc; co-workers who have helped me in many practical and personal issues during the whole five year period.

This work has been financially supported by Tampere Graduate School in Information Science and Engineering (TISE), Jenny and Antti Wihuri foundation, Ulla Tuominen foundation, Nokia foundation, and the Foundation of advancement of technology (TES).

I would like to dedicate this work to my family and friends. Now I am finally done and can answer your question that has been repeated numerous times: “What is it that you really do?”
List of publications

This Thesis is a monograph which contains some unpublished material but is mainly based on already published work. Copyright of the previously published material is owned by the copyright holders of the following publications.


Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-D</td>
<td>Two dimensional</td>
</tr>
<tr>
<td>3-D</td>
<td>Three dimensional</td>
</tr>
<tr>
<td>4-D</td>
<td>Four dimensional</td>
</tr>
<tr>
<td>A</td>
<td>Arbiter</td>
</tr>
<tr>
<td>Addr</td>
<td>Address</td>
</tr>
<tr>
<td>AMBA</td>
<td>Advanced microcontroller bus architecture</td>
</tr>
<tr>
<td>AMBA AHB</td>
<td>AMBA advanced high-performance bus</td>
</tr>
<tr>
<td>AMBA APB</td>
<td>AMBA advanced peripheral bus</td>
</tr>
<tr>
<td>AMBA ASB</td>
<td>AMBA advanced system bus</td>
</tr>
<tr>
<td>AMBA AXI</td>
<td>AMBA advanced extensible interface</td>
</tr>
<tr>
<td>API</td>
<td>Application programming interface</td>
</tr>
<tr>
<td>A/R</td>
<td>Aspect ratio</td>
</tr>
<tr>
<td>ARM</td>
<td>Advanced RISC machines Limited</td>
</tr>
<tr>
<td>AS</td>
<td>Application specific</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application specific integrated circuit</td>
</tr>
<tr>
<td>aSOC</td>
<td>Adaptive system-on-a-chip</td>
</tr>
<tr>
<td>Async</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>B</td>
<td>Bridge</td>
</tr>
<tr>
<td>Bi</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>BUF</td>
<td>Buffer component</td>
</tr>
<tr>
<td>Cen</td>
<td>Centralized</td>
</tr>
<tr>
<td>Clk</td>
<td>Clock signal</td>
</tr>
<tr>
<td>CMOS</td>
<td>Complementary metal oxide semiconductor</td>
</tr>
<tr>
<td>Comm</td>
<td>Command</td>
</tr>
<tr>
<td>CoreConnect DCR</td>
<td>CoreConnect device control register</td>
</tr>
<tr>
<td>CoreConnect OCP</td>
<td>CoreConnect on-chip peripheral bus</td>
</tr>
<tr>
<td>CoreConnect PLB</td>
<td>CoreConnect processor local bus</td>
</tr>
<tr>
<td>CPB</td>
<td>Configurable physical logic block in Celaro emulator</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>DCT</td>
<td>Discrete cosine transform</td>
</tr>
<tr>
<td>DES</td>
<td>Data encryption standard</td>
</tr>
<tr>
<td>Dist</td>
<td>Distributed</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct memory access</td>
</tr>
<tr>
<td>DPM</td>
<td>Dynamic power management</td>
</tr>
<tr>
<td>DRAM</td>
<td>Dynamic random access memory</td>
</tr>
<tr>
<td>DSP</td>
<td>Digital signal processor</td>
</tr>
<tr>
<td>eDRAM</td>
<td>Embedded dynamic random access memory</td>
</tr>
<tr>
<td>FIFO</td>
<td>First in, first out buffer memory</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field programmable gate array</td>
</tr>
<tr>
<td>FPS</td>
<td>Frames per second</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite state machine</td>
</tr>
<tr>
<td>GALS</td>
<td>Globally asynchronous, locally synchronous</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>GOPS</td>
<td>Giga operations per second</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware description language</td>
</tr>
<tr>
<td>HIBI</td>
<td>Heterogeneous IP block interconnection</td>
</tr>
<tr>
<td>IBM</td>
<td>International business machines Corporation</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated circuit</td>
</tr>
<tr>
<td>ID</td>
<td>Identification</td>
</tr>
<tr>
<td>IDCT</td>
<td>Inverse discrete cosine transform</td>
</tr>
<tr>
<td>ILD</td>
<td>Inter-level dielectric</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/output</td>
</tr>
<tr>
<td>IP</td>
<td>Intellectual property</td>
</tr>
<tr>
<td>ISS</td>
<td>Instruction set simulator</td>
</tr>
<tr>
<td>ITRS</td>
<td>International technology roadmap for semiconductors</td>
</tr>
<tr>
<td>LAN</td>
<td>Local area network</td>
</tr>
<tr>
<td>M</td>
<td>Master</td>
</tr>
<tr>
<td>Marble</td>
<td>Manchester asynchronous bus for low energy</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple input, multiple output</td>
</tr>
<tr>
<td>MIN</td>
<td>Multistage interconnection network</td>
</tr>
<tr>
<td>MUX</td>
<td>Multiplexer component</td>
</tr>
<tr>
<td>n/a</td>
<td>Not available</td>
</tr>
<tr>
<td>NoC</td>
<td>Network-on-chip</td>
</tr>
<tr>
<td>NRE</td>
<td>Nonrecurring engineering</td>
</tr>
<tr>
<td>NUMA</td>
<td>Non-uniform memory access</td>
</tr>
<tr>
<td>OCP</td>
<td>Open core protocol</td>
</tr>
<tr>
<td>OMI</td>
<td>Open microprocessor systems initiative</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>P2P</td>
<td>Point-to-point</td>
</tr>
<tr>
<td>PADDI</td>
<td>Programmable arithmetic devices for high-speed digital signal processing</td>
</tr>
<tr>
<td>PI</td>
<td>Peripheral interconnect</td>
</tr>
<tr>
<td>QCIF</td>
<td>Quarter common intermediate format</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-service</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>RE</td>
<td>Read enable</td>
</tr>
<tr>
<td>RF</td>
<td>Radio frequency</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced instruction set computer</td>
</tr>
<tr>
<td>RNS</td>
<td>Registered next state</td>
</tr>
<tr>
<td>ROM</td>
<td>Read only memory</td>
</tr>
<tr>
<td>Rr</td>
<td>Router</td>
</tr>
<tr>
<td>RstX</td>
<td>Active low reset signal</td>
</tr>
<tr>
<td>RTL</td>
<td>Register-transfer-level</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-time operating system</td>
</tr>
<tr>
<td>Rx</td>
<td>Receiver</td>
</tr>
<tr>
<td>S</td>
<td>Slave</td>
</tr>
<tr>
<td>Shr</td>
<td>Shared</td>
</tr>
<tr>
<td>SoC</td>
<td>System-on-chip</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SoCIN</td>
<td>SoC interconnection network</td>
</tr>
<tr>
<td>SRAM</td>
<td>Static random access memory</td>
</tr>
<tr>
<td>STBus</td>
<td>Split transaction bus</td>
</tr>
<tr>
<td>Sync</td>
<td>Synchronous</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time division multiple access</td>
</tr>
<tr>
<td>T-S-T</td>
<td>Time-space-time</td>
</tr>
<tr>
<td>Tx</td>
<td>Transmitter</td>
</tr>
<tr>
<td>UMA</td>
<td>Uniform memory access</td>
</tr>
<tr>
<td>Uni</td>
<td>Unidirectional</td>
</tr>
<tr>
<td>VC</td>
<td>Virtual component</td>
</tr>
<tr>
<td>VCI</td>
<td>Virtual component interface</td>
</tr>
<tr>
<td>VHDL</td>
<td>Very high speed integrated circuit hardware</td>
</tr>
<tr>
<td>VSI</td>
<td>Virtual socket interface</td>
</tr>
<tr>
<td>W</td>
<td>Wrapper</td>
</tr>
<tr>
<td>WE</td>
<td>Write enable</td>
</tr>
<tr>
<td>X</td>
<td>Switch</td>
</tr>
</tbody>
</table>
## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>Permittivity</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Resistivity</td>
</tr>
<tr>
<td>$0x$</td>
<td>Hexadecimal number</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND function or AND gate</td>
</tr>
<tr>
<td>BU</td>
<td>Number of burst commands in a transaction</td>
</tr>
<tr>
<td>C</td>
<td>Capacitance</td>
</tr>
<tr>
<td>CC</td>
<td>Number of clock cycles</td>
</tr>
<tr>
<td>CCR</td>
<td>Number of reserved clock cycles</td>
</tr>
<tr>
<td>CCTD</td>
<td>Number of clock cycles transmitting data</td>
</tr>
<tr>
<td>D</td>
<td>Delay</td>
</tr>
<tr>
<td>$d_{ij}$</td>
<td>Distance between two nodes (i,j) in a graph</td>
</tr>
<tr>
<td>DT</td>
<td>Number of clock cycles transmitting data</td>
</tr>
<tr>
<td>E</td>
<td>Energy</td>
</tr>
<tr>
<td>e</td>
<td>Number of parallel edges in an interconnection network</td>
</tr>
<tr>
<td>Err</td>
<td>The number of encountered errors in verification</td>
</tr>
<tr>
<td>F</td>
<td>Frequency</td>
</tr>
<tr>
<td>FO4</td>
<td>Delay of an inverter driving four identical copies of itself</td>
</tr>
<tr>
<td>i</td>
<td>Instantaneous current</td>
</tr>
<tr>
<td>K</td>
<td>Miller multiplication parameter</td>
</tr>
<tr>
<td>$&lt;k&gt;$</td>
<td>Average degree of a graph</td>
</tr>
<tr>
<td>L</td>
<td>Average path length of a graph</td>
</tr>
<tr>
<td>$L_B$</td>
<td>Length of the bursts</td>
</tr>
<tr>
<td>$L_{TF}$</td>
<td>Length of the time frame</td>
</tr>
<tr>
<td>N</td>
<td>Total number of nodes in a graph or interconnection network</td>
</tr>
<tr>
<td>$N_{Seg}$</td>
<td>Number of nodes in a bus segment</td>
</tr>
<tr>
<td>OR</td>
<td>Logical OR function or OR gate</td>
</tr>
<tr>
<td>P</td>
<td>Power</td>
</tr>
<tr>
<td>$P_{Leakage}$</td>
<td>Power consumption caused by leakage phenomena in CMOS</td>
</tr>
<tr>
<td>$P_{Short-circuit}$</td>
<td>Power consumption caused by short-circuit phenomena in CMOS</td>
</tr>
<tr>
<td>$P_{Switching}$</td>
<td>Power consumption caused by switching phenomena in CMOS</td>
</tr>
<tr>
<td>$P_{Total}$</td>
<td>Total power consumption</td>
</tr>
<tr>
<td>R</td>
<td>Resistance</td>
</tr>
<tr>
<td>RW</td>
<td>Number of read and write commands in a transaction</td>
</tr>
<tr>
<td>Seg</td>
<td>Number of bus segments</td>
</tr>
<tr>
<td>t</td>
<td>Time</td>
</tr>
</tbody>
</table>
\( t_p \)  
Processing time

\( t_t \)  
Transaction time

\( u \)  
Instantaneous voltage
# Table of contents

Abstract ............................................................................................................... 1  
Preface ............................................................................................................. 2  
List of publications .......................................................................................... 3  
Abbreviations .................................................................................................. 5  
Symbols ............................................................................................................ 8  
Table of contents ............................................................................................ 10  
1. Introduction ............................................................................................ 13  
   1.1. Background ....................................................................................... 13  
   1.2. Objectives and methodology ........................................................... 14  
   1.3. Outline of Thesis ........................................................................... 15  
   1.4. Summary of publications and the Author’s contribution ............... 15  
2. Digital system design and interconnects ................................................ 18  
   2.1. Abstraction-levels and description styles ....................................... 18  
   2.2. Digital system design flow ............................................................... 20  
   2.3. Design and fabrication cost ............................................................... 22  
   2.4. Power consumption ....................................................................... 23  
   2.5. Interconnection networks in heterogeneous systems .................... 24  
   2.6. Effect of technology trends to digital system design ....................... 25  
3. Interconnection networks ....................................................................... 27  
   3.1. Basic terminology .......................................................................... 27  
   3.2. Basic topologies ............................................................................. 28  
   3.2.1 Bus ............................................................................................. 28  
   3.2.2 Crossbar ...................................................................................... 30  
   3.2.3 Tree ............................................................................................ 31  
   3.2.4 Ring ........................................................................................... 33  
   3.2.5 Cube .......................................................................................... 33  
   3.2.6 Fully connected, point-to-point network ...................................... 36  
   3.2.7 Multistage interconnection networks ........................................... 36  
   3.2.8 Conclusions of the networks ....................................................... 37  
   3.3. Switching and routing schemes ....................................................... 40  
   3.3.1 Circuit-switching vs. packet-switching ....................................... 40  
   3.3.2 Switching techniques .................................................................. 41  
   3.4. Arbitration ...................................................................................... 42  
   3.5. Physical interconnections ............................................................... 43  
   3.6. Network-on-Chips ....................................................................... 46  
   3.6.1 NoC architectures ..................................................................... 46  
   3.6.2 Example NoC schemes .............................................................. 47  
4. Buses in on-chip digital systems ............................................................ 52  
   4.1. Structure ....................................................................................... 53  
   4.2. Transfers ....................................................................................... 55  
   4.3. Bus arbitration ............................................................................... 57  
   4.4. Support for complex structures ..................................................... 58
9.1. Results of Thesis ................................................................. 149
9.2. Future trends ................................................................. 150
10. References ................................................................. 152
1. Introduction

1.1. Background

The introduction of digital electronic systems at the latter part of the 20th century has made a considerable impact to the everyday lives of the population living in industrialized countries. The rapid adoption of new design and fabrication techniques along with the development of industrial process automation enabled devices that had previously been unattainable because of their complexity and price. These systems included personal computers, mobile communication equipment, and other home electronic devices. This development was made possible by the invention of integrated circuits (IC) that enabled the fabrication of reliable and cheap devices with high computing performance.

The integration density of ICs has been increasing steadily for the last decades. More and more transistors can be fitted into smaller and smaller area, and one billion transistor ICs are only a few years away. Whereas the feature sizes of transistors keep shrinking, many electrical phenomena make the reduction of the interconnect area more problematic. In the past, the speed and area of the functional units has always been the limiting factor in IC design. It has been predicted that the interconnection architecture will become the limiting factor.

The increased capacity of ICs has made it possible to construct whole systems on a single chip containing for example application specific logic, analogue and radio frequency (RF) parts, and programmable processor cores. These heterogeneous system ICs are usually referred to as system-on-chips (SoC) or if network design principles are applied network-on-chips (NoC). Many new design methods are needed in their design, for example the utilization of hierarchical design in the form of pre-designed and pre-verified components, usually called intellectual property (IP) blocks.

The reuse of IP blocks is a good way to make SoC design more efficient. However, interconnecting the heterogeneous IP blocks can become a problem. The IP blocks may have varying demands for the interconnection architecture. In addition, interconnecting IP blocks from different sources may require additional ‘glue’ logic to fit the blocks together. It is usually not practical to design new interconnections for every application. A preferred way is to design a standard interconnection scheme or a standard interface.

This design approach is taken in a design methodology called platform-based design. It requires that the principal components of the hardware should remain the same with only a small degree of variation being allowed. This usually limits the application area of the platform.
1.2. Objectives and methodology

This Thesis presents the design, implementation, and verification of a novel interconnection scheme developed for system-on-chips. The choices made are justified with theoretical comparisons and practical studies. The significance of the Thesis is in helping digital system designers to choose the interconnection architecture for their own cases. The selection process is almost always a trade-off between complexity and performance.

The initial analyses are based on theoretical comparisons between the different interconnection architectures. They do not take the targeted applications into account. These analyses provide rough theoretical connectivity and complexity estimates. For more accurate results, studies that take into account also the practical limitations of the architectures are required.

The practical studies are conducted with the aid of a standard simulation and synthesis procedure that will be presented in this Thesis. These analyses give more accurate results about the presented architectures with different transfer patterns. The results of these studies give the complexity and performance of the architectures with physical area and throughput measures.

The conducted studies led to the design of an on-chip architecture called heterogeneous IP block interconnection (HIBI). The architecture is implemented and verified with simulations and emulations with realistic test cases and real applications.

The thesis of this work is that an interconnection approach that utilizes a hierarchy of bus structures and augments this strategy with more specialized interconnection networks on case to case basis is an excellent candidate for the architecture of future SoCs. In this type of architecture, the local interconnections are buses making use of their advantageous properties that will be discussed in this Thesis. These bus segments can be clustered together by bridge components to form a hierarchy of bus structures. In addition, some of the global interconnections can use a more complex network structure if the required connectivity or bandwidth cannot be provided by a bus. This type of a heterogeneous architecture can be modified according to the application requirements.

Since the field of SoC design is vast, all its aspects cannot be studied in a single thesis. Issues that are left out of this Thesis, or discussed only very briefly, include the software aspects of SoC design, analogue issues, and SoC memory hierarchies.
1.3. Outline of Thesis

Chapter two of this Thesis gives the background to the design of complex digital systems. It presents a classical digital design methodology and flow with emphasis on interconnection networks. Some of the material in this Chapter has been previously published in [P3], [P6], [P8], and [P9].

Chapter three presents interconnection networks and switching schemes that have formed the basis of implemented digital system architectures. In addition, the Chapter describes a set of published NoC schemes as well as some general design principles for NoCs. Some of the material in this Chapter has been previously published in [P1], [P2], and [P4].

After Chapter three, Thesis concentrates on one traditional interconnection architecture, the bus. Chapter four gives a definition of a bus architecture and presents some widely used on-chip buses. Different implementations of systems based on bus architectures are also described. Some of the material in this Chapter has been previously published in [P1].

Comparisons based on simulation and synthesis are presented in Chapter five. In addition, different metrics frequently used in interconnection comparisons are discussed. Some of the material in this Chapter has been previously published in [P2].

Chapter six presents a new interconnection architecture that is based on the bus architecture. The design of this architecture is described in detail. Most of the material in this Chapter has been previously published in [P4].

The implementation and verification of the scheme presented in Chapter six, is described in Chapter seven. The implementation is done using the methodology presented in Chapter two and the verification process utilizes simulations and emulations. Some of the material in this Chapter has been previously published in [P4], [P5], [P7] and [P8].

In Chapter eight, the presented interconnection scheme is analyzed with synthesis results and test cases. The analyzed properties include performance metrics and physical properties of the interconnection architecture. Some of the material in this Chapter has been previously published in [P3] and [P4].

The conclusions of the work are given in Chapter nine.

1.4. Summary of publications and the Author's contribution

Publication [P1] describes the basic properties of bus-based interconnections. In addition, some bus enhancements schemes for large on-chip networks are
presented. A number of bus interconnections and bus-based systems are given as examples. The author of this Thesis is the main contributor. The studies presented in [Sal02] formed the basis of the on-chip bus comparisons. All other studies were conducted by the author. Salminen M.Sc, Dr. Kuusilinna, and Prof. Hämäläinen assisted in writing of this publication.

Publication [P2] introduces a comparison between generic bus and crossbar interconnections. The interconnections are implemented with VHDL (Very high-speed integrated circuit hardware description language) code and the comparison is based on synthesis results. The authors of the publication are Lahtinen M.Sc, Salminen M.Sc, Dr. Kuusilinna, and Prof. Hämäläinen of which the author of this Thesis is the main contributor.

Publication [P3] examines the power consumption of the presented interconnection scheme [P4] and speculates with different power consumption reduction methods that can be utilized in the scheme. The author of this Thesis is the main author and contributor to the publication. Dr. Kuusilinna, Kangas M.Sc, Salminen M.Sc, and Prof. Hämäläinen assisted in the writing of the publication.

Publication [P4] introduces a heterogeneous on-chip interconnection scheme. The design of the scheme along with implementation and analysis is presented. The author of this Thesis is the main author and contributor. Kangas M.Sc provided the video encoder test case and Dr. Kuusilinna, Kangas M.Sc, and Prof. Hämäläinen assisted in writing of the publication. The scheme was originally presented in [Kuu98].

Publication [P5] introduces a case study of finite state machine (FSM) optimization based on the VHDL encoding methods presented in [P6] and the interconnection scheme presented in [P4]. The author of this Thesis is the main author and contributor to the publication. Dr. Kuusilinna and Prof. Hämäläinen assisted in the writing of the publication.

Publication [P6] presents different FSM encodings in VHDL. The coded FSMs are synthesized to an application specific integrated circuit (ASIC) and a field programmable gate array (FPGA) technology and the results are compared. An automatic way to generate different FSMs is presented. The main author of this publication was Dr. Kuusilinna who also conceived the idea to this publication. The author of this Thesis assisted in the writing of the publication, coded the ‘leaving edge’ FSMs, and carried out the ASIC syntheses.

Publication [P7] describes the wrapper component utilized in composing systems based on the presented interconnection scheme [P4] in detail. This description includes the internal structure and the layout-level implementation of the component. The authors of the publication are Lahtinen M.Sc, Dr. Kuusilinna, Prof. Hämäläinen, and Prof. Saarinen of which the author of this Thesis is the main contributor.
Publication [P8] presents the verification of the interconnection scheme presented in [P4]. The verification is done using register-transfer-level and gate-level simulations along with emulations. The publication’s main contributor is the author of this Thesis. Assistance for the writing process was received from Dr. Kuusilinna, Prof. Hämäläinen, and Prof. Saarinen.

Publication [P9] presents an architecture-level system design test case that utilizes hardware/software co-simulation. The author of this Thesis is the main contributor of the publication. Dr. Kuusilinna, Prof. Hämäläinen, and Prof. Saarinen assisted in the writing of the publication.
2. Digital system design and interconnects

The digital abstraction has been a key enabler in the development of modern electronic devices. Abstraction-levels are used to hide lower level details by a higher level model that contains only the information relevant to the current phase of the design process. In addition, the utilization of modular design styles has helped in managing the ever increasing design sizes by building systems out of smaller components. Furthermore, the digital representation of data is quite insensitive to interferences and easy to store and modify with mathematical algorithms and programmable systems.

Digital system design has gone through a considerable change due to the rising integration density and design complexity. This development has been made possible by the steady rise of the abstraction-level [San03]. What started out as physical low-level design process dealing with geometrical layout-level structures has evolved through block-level design dealing with logic gates to the now dominant design method of using hardware description languages (HDL).

Traditionally, there have been three different description styles for digital systems. The behavioural style describes the system functionality, the structural style describes the system as a set of components and their interconnections, and the physical style binds the structure in space as geometrical objects.

The Y-chart [Gaj83] of Figure 2.1 depicts the different digital system description styles and their abstraction-levels. The lines forming the Y represent the different description styles and the circles the different levels of abstraction. The lowest level of abstraction is located nearest to the centre of the chart. The abstraction-levels starting from the centre are circuit-level, logic-level, register-transfer-level (RTL), and system-level.

2.1. Abstraction-levels and description styles

The modern HDL-based methods utilize a design-synthesize-analyze methodology where the behaviour, structure, or physical description of a system is first captured or designed at a high abstraction-level after which it is transformed to a lower level description or to a different description style. The transformation to a lower abstraction-level is called synthesis and it can be an automatic, semi-automatic, or a manual process. After the synthesis, the result is analyzed in order to verify the correctness and study the performance. The possible transformations of a system description are synthesis, mapping, and refinement [Rie99].
As stated, synthesis is a transformation from a high-level description to a low-level one. In Figure 2.1, this could mean, for example, a transformation from a register-transfer-level behavioural description to Boolean expressions at the behavioural-level. Since this transformation is the only one that changes the abstraction-level, its importance has been paramount in the development of design methodologies.

Mapping, on the other hand, is a transformation from a description style to another while maintaining the same abstraction-level. An example in Figure 2.1 is a transformation from Boolean expressions in the behavioural style to gates in the structural style description. This transformation is also sometimes erroneously called synthesis.

Finally, refinement is a transformation within a description style and an abstraction-level to improve the system description in some respect. An example in Figure 2.1 could be a transformation within Boolean expressions described in behavioural style. This example transformation is usually done to minimize the logic complexity.

An inseparable part of all the transformations is verification, which is used to make sure that the result is what was intended. This can be done with simulations, prototyping, or formal methods. Simulation is here defined as exercising test patterns on a software model of the actual hardware. It is utilized in all the description styles and at all the abstraction-levels of the design space.
digital system design. Prototyping [Ros99], on the other hand, is a verification technique where a physical hardware representation of the verified system is used. In the prototyping approach called emulation, the RTL description is synthesized to gate-level and then mapped onto a specialized configurable emulation hardware.

Formal methods utilize precise mathematical rules for the verification process and, therefore, require a precise mathematical system description. They can, for example, be used to verify that the behaviour of the synthesis result is the same as in the original higher level description. Choosing the proper verification method is usually a trade-off between verification time and accuracy since lower level descriptions contain more information and are slower to verify.

2.2. Digital system design flow

Figure 2.2 depicts an example of a digital system design flow. The depicted design process starts from a behavioural description at the algorithm-level. This first, possibly executable, specification is preferably implemented with some general programming language, like C, or a HDL. This specification is then executed or simulated until it has been confirmed that it specifies the desired functionality. The result of the simulation gives a golden reference to which the later design stages are compared, automatically if possible.

The first synthesis process, usually called behavioural synthesis or high-level synthesis, transforms the initial algorithmic description into register-transfer-level description given in a HDL format. Both these descriptions are still in the behavioural domain. It should be noted that frequently the register-transfer-level description is the first one to be implemented with a programming language. In these cases, the specification is given as a written document. In fact, although an executable specification is easier to verify and use as a later reference, the written specification - RTL code - logic synthesis flow is still dominant in many digital design areas.

At the register-transfer-level where the system is modelled as registers and data transfers between them, the description is again simulated with an RTL simulator and the results of the simulation are compared to the golden reference. In addition, the correctness of the synthesis process can be verified with formal methods. After the RTL implementation has been verified, the flow can proceed into two distinct directions: towards the final implementation or towards rapid prototyping with emulation. The same type of synthesis and mapping procedure is applied in both of these branches.

The RTL description is first synthesized to Boolean expressions in the behavioural domain, after which it is transformed into gates in the structural domain. These gates come from a library that contains the components of the utilized implementation or emulation technology. The result of this phase is a
gate-level netlist which models the system as physical components and their interconnections.

During hardware emulation, the emulation netlist is downloaded into a hardware emulator. The same test patterns that were used with simulations can be applied here. With emulation, implementation related logical errors can be observed and located, although, it is also used to speed-up RTL simulations.

Because the emulation technology differs from the actual implementation technology, it cannot be used for verifying all implementation details. For this purpose, gate-level simulations are used. Timing information, such as the maximum operating frequency of the implementation, can be deduced from the gate-level synthesis results and simulations. The correctness of the gate-level and the emulator netlist can again be verified with automatically comparing to the results of the original specification simulations and with formal methods, for example comparing the netlist to the RTL description.

The gate-level netlist is further transformed by a place-and-route process which is a mapping transformation followed by synthesis. This process starts from mapping the gates into cells from a standard cell library after which the cells are placed and the physical routes needed in their interconnection are formed.
After this, the implementation of the system can be verified. This step also produces accurate area and timing information which should be *back-annotated* to the previous phases of the design flow. The implementation can utilize an ASIC or an FPGA technology. At all the levels of abstraction, the results are analyzed in order to evaluate the performance of the implementations.

As a final note about the traditional digital system design, the separation between *control* and *dataflow* structures should be pointed out. This separation is present at all the abstraction-levels and description styles, but it is most visible in the behavioural style descriptions. Dataflow structures are frequently used to implement algorithms with processing nodes connected together [Lee95]. These structures can be *pipelined* by adding storage units, like registers, between the phases of the algorithm computation. In digital systems, control is usually implemented with programmable processors or application specific logic, namely *finite state machines* (FSM) [Sgr00]. As opposed to the predictable transfer patterns of the dataflow structures, control initiated transfers are usually very unpredictable.

### 2.3. Design and fabrication cost

The development of new products for the consumer electronic market with ever increasing capabilities and complex implementations is becoming more and more costly. The costs can be divided into nonrecurring engineering (NRE) costs and manufacturing costs. The design (NRE) costs can be lowered by applying design re-use of previously designed and verified IP blocks and by capturing the behaviour of a system at a higher abstraction-level [McF90].

In addition, the manufacturing costs will become a difficult problem. Particularly the mask costs are stated to rise to the million dollar range at the 0.15 µm technology node [Keu00]. This problem can be made easier by fabricating integrated circuits (IC) that can be used in a wide variety of systems implying the utilization of programmable and reconfigurable devices.

One solution to the rising costs of digital systems is applying a platform-based design methodology [Keu00][Cha99, pp. 51-182]. The idea is to have a mostly fixed hardware architecture that can be modified according to a specific implementation need to support a wide range of applications and the future development of a particular application. A platform consists of a hardware part, an application program interface (API) abstracting the hardware to the application software, and software containing real-time operating systems (RTOS) and device drivers. A platform is usually designed for a set of applications requiring similar services from the architecture.
2.4. Power consumption

There are two aspects of power consumption that are typically considered in relation to system design: average power and maximum power [Ben01]. Reducing the average power dissipation affects favourably the battery lifetimes and heat dissipation related costs in mobile devices. In addition, the maximum instantaneous power needs to be minimized in some applications because it affects the design of the power distribution subsystem. However, aggressive methods for reducing the maximum power in mobile systems are typically not as important as average power optimization.

In electronic systems, power is defined as

\[ P(t) = u(t) \cdot i(t) \]  

(2.1)

where \( u(t) \) is the supply voltage and \( i(t) \) is the instantaneous current. In complementary metal oxide semiconductor (CMOS) circuits, the total power dissipation (\( P_{\text{Total}} \)) can be divided into three main sources:

\[ P_{\text{Total}} = P_{\text{Switching}} + P_{\text{Short-circuit}} + P_{\text{Leakage}} \]  

(2.2)

Traditionally, the switching power (\( P_{\text{Switching}} \)) has been the dominant one. It is caused by the charging and discharging of capacitors during transient switching. Therefore, the switching activity, representing the frequency of switching, affects the switching power. The short-circuit power (\( P_{\text{Short-circuit}} \)) occurs when both N and P type transistors of CMOS are conducting at the same time during transient switching. The leakage power (\( P_{\text{Leakage}} \)) is becoming more important as the gate oxide thickness, channel length, and threshold voltage of transistors keep shrinking.

Although only power consumption is usually discussed, it must be stressed that actually energy consumption is the important issue. It rarely makes sense to optimize power consumption if at the same time the system execution time (\( \Delta t \)) increases to violate the design constraints. In practise, however, power minimization is usually only obtained at the price of increased area or decreased circuit speed [Ben01]. The energy consumption is defined as

\[ E = P \cdot \Delta t \]  

(2.3)

It is no longer possible to leave the power related optimizations to the final phases of the design flow due to the growing design complexity and increasing power consumption [Ben01]. In addition, usually both the power consumption and the performance need to be considered. In order to solve this trade-off problem, several power-performance metrics have been developed. A frequently used metric is the power-delay product [Ben01]. One alternative is
to use a performance limit where power minimization is acceptable if performance does not drop below an acceptable limit.

Power consumption can be minimized at many levels of design hierarchy. The most straightforward ways are the low-level (technology and logic-level) optimization methods. These methods include the use of low-power processes, low-power cell libraries, and low-voltage components. Clock gating is another low-level method that can be utilized. In clock gating, the clock signal of a component is disabled if it is not active or needed at the moment. This technique practically nullifies the dynamic power of the circuit when the clock is gated but leakage power is still dissipated. All the low-level techniques are usually applied at the later phases of the design flow.

If the run-time behaviour profile of the system is known or can be deduced run-time, a method called dynamic power management (DPM) [Ben98, pp. 85-214] can be used. Using DPM, the clock frequency or operating voltage of the system or a particular component can be adjusted so that power dissipation is minimized but the required performance is still achieved. In addition, the clock signal for the unused components can be disabled with a technique similar to low-level clock gating. System-level clock gating utilizes gathered information about the run-time profile of the system or is controlled at run-time by a high-level controller.

The unused components could also be switched off completely with power supply shutdown [Ben00]. This technique practically eliminates the power dissipation of an idle unit but has also severe drawbacks like the loss of all data in volatile memories which leads to extra power-up time and power dissipation. The power-up time and power dissipation can be reduced if the power supply is not shut down totally but only lowered to a value that still lets the memory components retain state information. System-level power optimization methods usually require that power-aware techniques are used throughout the system design flow.

The presented system-level power reduction methods mainly affect the switching and the short-circuit power. Low-leakage and low-performance cells are frequently used to cut down leakage power outside the critical path of the system. In addition, several technology and circuit-level optimization methods for reducing leakage current are presented in [Roy03].

2.5. Interconnection networks in heterogeneous systems

Modern large scale digital systems are often heterogeneous meaning that the computational units vary and their communication requirements are very different from each other [Rab98]. This is a clear distinction from the
homogeneous multiprocessor systems having identical computational units dealing with symmetrical communication streams.

The heterogeneous communication requirements imply that the transaction data amounts and transaction frequencies of the components vary significantly and that not all the components are necessarily logically connected. The heterogeneousness of the computation comes from the utilization of a variety of programmable processors and dedicated hardware blocks. This diversity in the used computation components calls for the use of hardware/software co-design methods [Wol94].

As stated, the design complexity of integrated circuits is increasing very rapidly and this complexity is managed by the use of modular system design methodologies. It is very likely that the complexity of these modules will not significantly increase since local interconnect wiring problems can only be dealt with if small modules are used [Ho01][Syl01]. Because the size of an applicable module will not grow, the number of these modules will grow instead. This will lead to a problem with the global interconnect wiring between the modules [Syl01].

Increasing system complexity and physical interconnect issues lead to a need for a common interconnection architecture. The topologies, switching schemes, physical wiring, and protocols of these interconnections need to be studied very carefully. In this work, the old studies of computer backplanes, local area networks (LAN), and multiprocessor architectures become relevant again. In these studies, systems having approximately 2-16 processing nodes are called small-scale systems and systems having over 64 processing nodes are called large-scale systems [Len95, p. 3]. This is a distinction that is also applied in this Thesis.

Until recently, SoCs have been based on simple buses. Simple bus networks have been applicable because the present SoCs usually contain only a modest number of processing nodes. When the systems become larger, more advanced methods are needed in their interconnection. These methods are used to form network-on-chips. Their implementation details are under extensive research.

2.6. Effect of technology trends to digital system design

The International Technology Roadmap for Semiconductors (ITRS) [ITRS03] is an assessment of the future development of semiconductor technology by manufacturers, suppliers, government organizations, consortia, and universities. Table 2.1 presents the roadmap for the requirements of future low-power, handheld wireless devices. These devices contain central processing units (CPUs), digital signal processors (DSPs), application specific processing engines, static random access memories (SRAMs), and embedded Dynamic
RAMs (eDRAMs). The year span of the roadmap is from 2003 to 2018, going through the technology generations from 100 nm to 22 nm (minimum metal pitch) for CMOS technologies. The last line of Table 2.1 shows the projected application starting from still image processing, going through real-time video coding, and finally to the real-time data interpretation.

Table 2.1 System drivers for low-power applications [ITRS03].

<table>
<thead>
<tr>
<th>Year</th>
<th>2003</th>
<th>2006</th>
<th>2009</th>
<th>2012</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology [nm]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supply voltage [V]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clock [MHz]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Performance [GOPS]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power [W]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores [Mtrans.]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td>Still image</td>
<td>Realtime video codec</td>
<td>Real time interpretation</td>
<td>?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The projected growth in the maximum on-chip frequency of the low-power applications is quite modest being only 5-10 % of the highest possible processor clock frequencies [ITRS03] at each technology node. The peak power consumption of these devices is limited to 0.1 W due to battery lifetimes, although the required processing power in Giga operations per second (GOPS) grows quite rapidly. This very low allowed power consumption is partly achieved through the lowering of the supply voltage.

A troublesome trend is that the amount of core processor logic (CPU and DSP) increases four-fold in each technology generation. This complexity increase and the required constant power consumption are the most difficult demands for the global interconnection architecture. These are also the issues that should be emphasized in the design process of digital systems.
3. Interconnection networks

Of the presented computer architecture [DeC89, pp. 1-296][Hwa89, pp. 32-40] classifications the one by Flynn [Fly72] is the most famous. The classifications deal with the different ways of utilizing multiple processing units to compute a given task. All these architectures require a network for interconnecting the processing nodes together. In this Chapter, some of these interconnection networks and their basic properties are described.

3.1. Basic terminology

In this Thesis, the term transaction refers to the whole process of sending one or several data items over an interconnection network including the setup of the connection and the process of returning the network into an idle state. A transfer, on the other hand, is the operation for sending a single data element.

When data is transmitted over an interconnection network, a certain time period elapses from the start of the process to the time when data reaches the end destination. This time is composed of the time it takes to get the network ownership and the actual transfer time and is called latency. Latency is a very good performance metric for systems having small data transactions.

Throughput is a measure of the transmitted data in a certain time interval [Bhu89]. It is the most commonly used performance measure but it is very sensitive to manipulation. Sometimes also the term bandwidth is used to measure the same property. In this Thesis, however, bandwidth is used to denote the maximum throughput the interconnection can support [Gus95]. Very high throughputs can be achieved with extremely long lasting data transactions that are seldom utilized in real applications.

In addition to latency and throughput, two other performance metrics are used: setup time and delay. Setup time refers to the time it takes to form the path in the network between the sender and the recipient of data. Delay is the time it takes from the transfer to go through the network from the sender to the receiver. In addition, physical measures of interconnection networks can be used to compare them. Examples of these are area, operating frequency, and power consumption.

Interconnection networks can be divided into static and dynamic networks [Hwa89, p. 334]. Static networks utilize only point-to-point or shared connection lines whereas the dynamic counterparts use also switch components to form the required paths. Static networks have been utilized particularly in the computation of fixed, dedicated algorithms.
Another possible interconnection network division is to divide them into direct and indirect networks. In a direct network, a switch is always connected to a processing node. In an indirect network, the switches can also be connected to other switches as well as processing nodes.

Interconnection networks can be modelled with graphs that have nodes representing processing units connected with edges representing data streams between the nodes [Cor90, pp. 86-90]. The total number of nodes, denoted by $N$, is one basic property of a network. The distance between two nodes $i$ and $j$, denoted by $d_{ij}$, is defined as the number of edges in the path between them. Another figure to describe a network is the average path length, denoted by $L$, which stands for the average of all the $d_{ij}$ components in a network. Large values of $L$ can cause large average latencies. In addition, the average degree, denoted by $<k>$, is sometimes used to compare networks. It stands for the average number of edges in each of the switches in a network. Networks with large values of $<k>$ can be complex to route and implement.

### 3.2. Basic topologies

In this Section, various basic network topologies are presented with examples. The topology of a network includes those properties that do not change when their graph representations are transformed by stretching or bending the graph. The used symbols are depicted in Figure 3.1. Unless otherwise stated, the communication edges are bidirectional.

![Network graph symbols](image)

**Figure 3.1** The utilized network graph symbols.

#### 3.2.1 Bus

In bus-based systems, there are one or several shared communication edges between the processing nodes [Gus95][Var94, pp. 178-184]. In this Thesis, the case with one edge is referred to as a single bus and the case with several parallel edges a multiple bus. These two are static interconnection networks. A third basic bus network is the hierarchical bus. It has a switch, usually called a bridge, between single or multiple bus segments, and is therefore a dynamic network. In theory, the number of edges in a multiple bus is not restricted. On the other hand, the number of segments in a hierarchical bus cannot exceed the number of nodes unless the system contains empty segments with no nodes.
In Figure 3.2, the single and multiple bus topologies are presented in a sixteen node case with average path lengths of one. Since the single and multiple bus networks do not have switches, their degree cannot be determined.

![Single bus and Multiple bus diagrams]

**Single bus**
- \( N = 16 \)
- \( L = 1 \)
- \(<k> = -\)

**Multiple bus**
- \( N = 16 \)
- \( L = 1 \)
- \(<k> = -\)

Figure 3.2 Single and multiple bus.

Hierarchical buses can be composed in many ways. A simple implementation is the *chain* in which the bridges are used to connect only two bus segments. This is depicted in the upper part of Figure 3.3 for sixteen nodes. The degree of this hierarchical bus is two. Because the processing nodes are connected in a chain, the average path length, which is 2.3, can become a problem. This problem can be helped if the bus segments are connected at least partly in a tree structure like depicted in the lower part of Figure 3.3. The depicted hierarchical bus is a combination of tree and chain structure. In this sixteen node case, the average path length is 2.1 and the average degree is 2.5.
Hierarchical bus (chain + tree)
\[ N = 16 \]
\[ L = 2.1 \]
\[ \langle k \rangle = 2.5 \]

Hierarchical bus (chain)
\[ N = 16 \]
\[ L = 2.3 \]
\[ \langle k \rangle = 2 \]

Figure 3.3 Hierarchical buses with four bus segments.

3.2.2 Crossbar

A crossbar resembles quite closely the multiple bus network [Var94, pp. 178-184]. Also in the crossbar, there are multiple parallel edges between the processing nodes. The difference is that a crossbar is a dynamic indirect network and has communication switches that direct the transmitted data to the correct receiver. Note that the classifications here do not separate circuit-switching and packet-switching.

The upper part of Figure 3.4 depicts a ‘classical’ crossbar that can be used in systems applying the uniform memory access (UMA) model [Hen03, p. 533]. In Figure, this could mean that the processors are located on the left and the memories on the bottom. In this scheme, the memory units do not communicate with each other. Because this network separates inputs and outputs, they are usually also separated in the node count, for example 8x8 in Figure 3.4 meaning eight inputs and eight outputs. In the rest of this Section, however, we treat the crossbar connections as bidirectional.
In a system applying the *non-uniform memory access (NUMA) model*, where all the nodes communicate with each other, a one-sided crossbar is preferred [Hen03, p. 533]. In the lower part of Figure 3.4, this would mean that the nodes contain both a processor and memory. In this case, all the nodes need to be able to communicate with each other. Both the example crossbars of Figure 3.4 have a node count of sixteen. The 8x8 crossbar has an average path length of 9 and degree of 3.75. The one-sided crossbar, on the other hand, has an average path length of 14.7 and a degree of 3.75.

![Figure 3.4 Crossbar and one-sided crossbar network.](image)

### 3.2.3 Tree

There are a multitude of indirect tree networks with different properties but when digital system architecture is targeted, only regular tree topologies are usually applicable [Var94, pp. 8-18][Cor90, pp. 91-96]. A *rooted* tree has one node that is distinguished from all the others like the highest node of the binary tree in Figure 3.5. The example tree has sixteen nodes, average path length of 6.5, and average degree of 2.9.
The height of the tree is counted from the root node downwards (4 in Figure 3.5). Child nodes are nodes that are connected to a higher node in the tree. The number of child nodes is called the degree of a node. In a complete binary tree, all leaf nodes that are lowest in the hierarchy, have the same depth ($\log_2 N$) and all the internal nodes have the same degree. Trees are well suited for recursive and divide-and-conquer algorithms.

Another example of a regular tree structure is the fat tree [Lei85] depicted at the bottom of Figure 3.5 with sixteen nodes, average path length of 6.5, and average degree of 3.5. A fat tree has a constant number of edges at each level of the hierarchy. Because the fat tree does not have a root, a single node does not become a performance bottleneck like in the rooted binary tree. One way to connect the levels of switches is to use butterfly elements which connect each pair of edges together. Fat trees are identified by the number of edges in the switches that is called the fanout. The fat trees with fanout of $n$ are also called $n$-ary fat trees.

Figure 3.5 Binary and fat tree networks.
3.2.4 Ring

*Ring* is a direct network consisting of a variable number of nodes that are connected to their nearest neighbours with uni- or bidirectional links [Var94, pp. 8-18]. Rings are useful in small systems but they are relatively unreliable since a single failure can severely limit the performance of the network and two failures can disconnect the network. This is a problem in other networks also; for example buses. This problem can be helped by adding new edges between nodes. In Figure 3.6, a bidirectional ring with sixteen nodes is presented which has an average path length of 6.3 and an average degree of 3.

![Ring network](image)

**Figure 3.6 Ring network.**

3.2.5 Cube

A general direct *cube* network is called a *k-ary n-cube*, where *n* is the dimension of the network and *k* represents the number of nodes (*N = k^n*) [Var94, pp. 8-18]. One example of such a network is the *n-dimensional hypercube* or, more accurately, the *binary n-cube*. In a hypercube, each switch has *n* external connections in addition to the one connection to the local processing node. The hypercube can be used to efficiently implement many parallel algorithms and it can also be utilized in the simulation of other networks, such as trees, rings, and meshes. A 4-dimensional example of this network is given in Figure 3.7 with sixteen nodes, average path length of 4.1, and average degree of 5.
Two frequently used 2-dimensional cube networks are the mesh and the torus [Var94, pp. 8-18] depicted in Figure 3.8 and Figure 3.9, respectively. For $N$ processing nodes, they both use $N$ communication switches. The difference is that torus has a connection from the outmost column and row to the other. The topologies based on 2-D arrays are well suited for applications dealing with matrix computations. Both examples have a node count of sixteen. The average path length of the mesh is 4.7 and the torus 4.1. The average degree of the 2-D mesh is 4 and the 2-D torus 5.
2-D mesh
\[ N = 16 \]
\[ L = 4.7 \]
\[ \langle k \rangle = 4 \]

Figure 3.8 2-D mesh network.

2-D torus
\[ N = 16 \]
\[ L = 4.1 \]
\[ \langle k \rangle = 5 \]

Figure 3.9 2-D torus network.
3.2.6 Fully connected, point-to-point network

With respect to connectivity, a fully connected point-to-point network is an extreme case [Hwa89, pp. 334-335]. It has a dedicated point-to-point link between every processing node. This interconnection architecture might not be sensible to implement because of the wire cost but a restricted amount of point-to-point links can help in data transfers in connections requiring high bandwidth.

An example fully connected, point-to-point network is presented in Figure 3.10 with 16 nodes and an average path length of one. As the architecture has no switches, the degree cannot be determined.

![Fully connected, point-to-point network](image)

**Fully connected, point-to-point network**

\[ N = 16 \]
\[ L = 1 \]
\[ \langle k \rangle = - \]

Figure 3.10 Fully connected, point-to-point network.

3.2.7 Multistage interconnection networks

Multistage interconnection networks (MIN) consist of stages of switches that can be used to form at least one path between every input and output, and are, therefore, also indirect networks [Var94, pp. 104-123]. They can be designed to match the communication patterns of many parallel algorithms.

Most of the multistage networks for \( N \) nodes, like the Omega network of Figure 3.11 with 16 nodes and path length of four, contain \( \log_2(N/2) \) stages each with...
$N/4$ switches that are basically 2x2 crossbars [Var94, pp. 104-123]. The difference between the multistage networks is how they are connected between the switch stages. In the Omega network, the connections between the stages utilize the *perfect shuffle* network. The average degree of the Omega network is four.

![Omega network diagram](image)

**Omega network**

$N = 16 \ (8\times8)$

$L = 4$

$<k> = 4$

Figure 3.11 An Omega multistage interconnection network.

### 3.2.8 Conclusions of the networks

The symbols used in the network conclusions of this Section are given in Table 3.1. Table 3.2 presents some selected performance metrics [Var94, p. 12][Len95, p. 30][Dun90][Hen03, p. 818] and Table 3.3 some implementation costs [Len95, p. 30] of the presented interconnection networks. The tabulated metrics are only applicable to the presented symmetric cases where $N$ has the limitations tabulated in Table 3.1.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Number of nodes ($N=i^2$, $i=0,1,2,...$ and $N=2^r$, $r=0,1,2,...$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e$</td>
<td>Number of parallel edges</td>
</tr>
<tr>
<td>Bi</td>
<td>Bidirectional</td>
</tr>
<tr>
<td>Uni</td>
<td>Unidirectional</td>
</tr>
</tbody>
</table>

The amount of parallel transactions tells how many simultaneous transactions are possible in the best case. The only limitation is that each transfer has to
have a sender and a receiver and a node cannot receive data from more than one place at a time. This means that the maximum value of parallel transactions in this comparison equals the number of nodes \( N \). The single bus is the only network not supporting parallel transactions. All the others have some amount of inherent parallelism. On the other hand, the longest path in a single and multiple bus is a fixed constant, whereas in the other topologies it grows as a function of the connected nodes. The longest path is here defined as the largest number of edges in any possible path in the network.

In this Thesis, the *bisection bandwidth* is calculated by dividing the network into two identical parts containing an equal amount of nodes and switches and then counting the edges crossing the division line [Len95, p. 24]. This is a valid definition since all the presented networks are symmetrical.

<table>
<thead>
<tr>
<th>Network</th>
<th>Parallel transactions</th>
<th>Longest path</th>
<th>Bisection bandwidth</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single bus</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Bi</td>
</tr>
<tr>
<td>Multiple bus</td>
<td>( e ) (( e \leq N ))</td>
<td>1</td>
<td>( e )</td>
<td>Bi</td>
</tr>
<tr>
<td>Hierarchical bus (chain)</td>
<td>( e ) (( e \leq N ))</td>
<td>( e ) (( e \leq N ))</td>
<td>1</td>
<td>Bi</td>
</tr>
<tr>
<td>Crossbar</td>
<td>( N )</td>
<td>( N )</td>
<td>( N-1 )</td>
<td>Bi</td>
</tr>
<tr>
<td>One-sided crossbar</td>
<td>( N )</td>
<td>2( N-1 )</td>
<td>( N/2 )</td>
<td>Bi</td>
</tr>
<tr>
<td>Binary tree</td>
<td>( N )</td>
<td>2( \log_2 N )</td>
<td>1</td>
<td>Bi</td>
</tr>
<tr>
<td>Fat tree (fanout 2)</td>
<td>( N )</td>
<td>2( \log_2 N )</td>
<td>( N )</td>
<td>Bi</td>
</tr>
<tr>
<td>Ring</td>
<td>( N )</td>
<td>( N/2+2 )</td>
<td>2</td>
<td>Bi</td>
</tr>
<tr>
<td>3-D hypercube</td>
<td>( N )</td>
<td>( \log_2 N+2 )</td>
<td>( N/2 )</td>
<td>Bi</td>
</tr>
<tr>
<td>2-D mesh</td>
<td>( N )</td>
<td>2( N^{1/2} )</td>
<td>( N^{1/2} )</td>
<td>Bi</td>
</tr>
<tr>
<td>2-D torus</td>
<td>( N )</td>
<td>( N^{1/2+2} )</td>
<td>2( N^{1/2} )</td>
<td>Bi</td>
</tr>
<tr>
<td>Point-to-point, fully connected</td>
<td>( N )</td>
<td>1</td>
<td>((N/2)*(N/2))</td>
<td>Bi</td>
</tr>
<tr>
<td>Omega network (MIN)</td>
<td>( N/2 )</td>
<td>( \log_2 N )</td>
<td>( N )</td>
<td>Uni</td>
</tr>
</tbody>
</table>

Table 3.3 tabulates the implementation costs of the presented networks. The bus-based interconnections have a quite small number of switches and wires. It should be noted that in the wire calculations, the wire connecting the processing node to the switch is also taken into account. In addition, the length
of the required wires is not considered here. As expected, the topologies offering scalable bisection bandwidth and many parallel connections have also the highest implementation costs.

Table 3.3 Implementation costs of topologies.

<table>
<thead>
<tr>
<th>Network</th>
<th>Number of switches</th>
<th>Number of wires</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single bus</td>
<td>0</td>
<td>1</td>
<td>Bi</td>
</tr>
<tr>
<td>Multiple bus</td>
<td>0</td>
<td>e</td>
<td>Bi</td>
</tr>
<tr>
<td>Hierarchical bus (chain)</td>
<td>e-1</td>
<td>e</td>
<td>Bi</td>
</tr>
<tr>
<td>Crossbar</td>
<td>N^2/4</td>
<td>N^2/2</td>
<td>Bi</td>
</tr>
<tr>
<td>One-sided crossbar</td>
<td>N^2/2</td>
<td>N^2-N/2</td>
<td>Bi</td>
</tr>
<tr>
<td>Binary tree</td>
<td>N-1</td>
<td>2(N-1)</td>
<td>Bi</td>
</tr>
<tr>
<td>Fat tree (fanout 2)</td>
<td>Nlog_2N</td>
<td>2Nlog_2N</td>
<td>Bi</td>
</tr>
<tr>
<td>Ring</td>
<td>N</td>
<td>2N</td>
<td>Bi</td>
</tr>
<tr>
<td>3-D hypercube</td>
<td>N</td>
<td>N+(N/2)log_2N</td>
<td>Bi</td>
</tr>
<tr>
<td>2-D mesh</td>
<td>N</td>
<td>3N-2N^{1/2}</td>
<td>Bi</td>
</tr>
<tr>
<td>2-D torus</td>
<td>N</td>
<td>3N</td>
<td>Bi</td>
</tr>
<tr>
<td>Point-to-point, fully connected</td>
<td>0</td>
<td>(N^2-N)/2</td>
<td>Bi</td>
</tr>
<tr>
<td>Omega network (MIN)</td>
<td>(N/4)(log_2N-1)</td>
<td>(N/2)log_2N</td>
<td>Uni</td>
</tr>
</tbody>
</table>

One requirement of an ideally scalable system is that the bisection bandwidth per node must be constant. This requirement is not met with systems having constant bisection bandwidth such as single and hierarchical bus, binary tree, and ring. In addition to the bisection bandwidth requirements, there are three requirements for an ideally scalable system [Len95, p. 24]:

1) Cost (number of switches + number of wires) that grows at most linearly with N

2) Longest path that represents latency independent of N

3) Parallel connections representing bandwidth that grows at most linearly with N
As can be observed from Table 3.2 and Table 3.3, none of the interconnection networks meet all three requirements. Choosing a network is, therefore, always a trade-off between performance and cost.

The presented theoretical interconnection network comparisons can be used as starting point for the design process of system architectures. They can be utilized in getting rough estimates about the cost and performance trade-offs of different networks. They have also been used in the interconnection architecture design examples presented later in this Thesis.

3.3. Switching and routing schemes

The communication switches utilized by interconnections need some sort of switching methodology to route the transmitted data to the correct recipient. The possible switching schemes can be divided into circuit-switched or packet-switched schemes. The actual routing can be done with store-and-forward, wormhole, or virtual cut-through techniques. [Tan96, pp. 130-134]

The routing algorithms are utilized in each switch to direct the data from the input of the switch to the correct output [Var94, pp. 242-257]. To do this efficiently, many communication methods and algorithms have been developed. Networks with regular and symmetric topology ease the routing process considerably. In a regular network, each node has the same degree, whereas a symmetrical network looks identical when viewed from each node or edge.

The basic communication methods used in interconnection networks can be divided into one-to-one, broadcast, and permutation communication. One-to-one communication transfers data from one source to one destination, whereas in broadcast communication there can be many destinations. Permutations allow each node to send a data element to a recipient node and receive a data element from another node at the same time. The routing algorithms are different for each network and are, therefore, not discussed in detail in this Thesis.

3.3.1 Circuit-switching vs. packet-switching

In a circuit-switched scheme, a route is first formed between the sending and the receiving node by modifying the switch matrix. After this, the subsequent transfers can utilize the established route. Therefore, the circuit-switched techniques have a certain initial setup time after which low latency is provided for the following transfers. If the receiver can buffer all the data and there are enough paths for every possible transfer to occur simultaneously, this technique is non-blocking. This means that once a transfer is started, the sender can keep sending data without being interrupted. If the amount of data to be stored in the buffer exceeds the buffer size or two transfers need to use the same path, the transfer needs to be interrupted with a technique called blocking. However, to
achieve low latency for large transfers, the interconnection needs to be reserved for the whole duration of the transaction. This may block other pending transfers. If every transfer follows the same route, circuit-switching can also be called fixed or static routing.

A packet-switched routing scheme establishes a route for all the individual transfers, called packets, independently based on the packet header. The header is used to store information about the transfer, for example the size and destination. As a dedicated path is not formed prior to transfer, the setup time is quite small and the whole interconnection path from the sender to the receiver is not reserved for a single transaction. Because each packet is routed independently, the routing is done for every packet in every switch it goes through. This makes the switching more complex and slower than in the circuit-switched case. It is also more difficult to make the transfers non-blocking. When a packet encounters a reserved edge on its path, the packet needs to be blocked, disregarded, or placed in a buffer (queued). An additional benefit of this scheme, however, is that the transfers can be routed away from congested areas. This is why packet-switching can also be called adaptive or dynamic routing.

In packet-switching, a large packet is usually divided into smaller packets that are sent individually. This can lead to problems at the receiving end because the packets may arrive in different order than they were sent. A way to go around this problem is to send large packets as single units following the same route in which case the data will always arrive in the right order. However, this technique that is sometimes referred to as message switching, has the disadvantage of requiring large buffers inside the interconnection network.

3.3.2 Switching techniques

Switches that apply the store-and-forward technique, store all incoming data packets. They then retransmit the data from their internal buffer, once a route for forwarding the data has been formed. The technique is very flexible and simple but the latency caused by the buffering of data and the needed buffer area sometimes limits its usefulness.

Wormhole routing minimizes the latency and buffer area by forwarding the packet before the entire packet has been received. However, the needed routing algorithms become more complex since they need to consider several path edges at once. In addition, the timing of the routing process becomes more critical in the wormhole technique. Furthermore, to work efficiently, this technique needs to reserve several communication edges, at least the edge coming to the switch and the edge leaving the switch.

Virtual cut-through combines some of the good properties of both store-and-forward and wormhole techniques. The packets are forwarded before they are entirely received like in the wormhole method but the whole route from the
sender to the receiver is not reserved. This can be accomplished with a scheme that only stores packets if they are blocked.

3.4. Arbitration

One problematic situation with all digital systems having shared resources is faced when more than one object tries to use the shared resource at the same time. This can happen, for example, in bus interconnections, shared memory systems, and inside a communication switch where the same output port is targeted by more than one input port. The method for resolving the object to use the resource is called arbitration [Gui89]. The basic arbitration schemes can be divided into static and dynamic methods [Ket95]. Many arbitration algorithms are similar to scheduling algorithms in software systems.

Static arbitration methods determine the owner of the shared resource at the design time of the system. A well known example of this kind of arbitration is the time-division multiple access (TDMA) method. In this method, time is divided into repeating time frames that are then divided into timeslots that are allocated to the objects requesting the ownership of the resource. When using this method, all the needed information for deciding the correct object is gathered before the system run-time, and is, therefore, very easy to distribute to each object. This method does not need a dedicated central arbiter unit found in the implementation of most of the dynamic arbitration methods.

Using dynamic arbitration methods, the owner of a shared resource is determined at the run-time of the system. A frequently used dynamic arbitration method is based on priorities. The priority value given to each object makes it possible to always give the shared resource to the object having the highest priority. This process can, therefore, lead to starvation meaning that lower priority objects are not necessarily able to access the shared resource. Priority arbitration can be pre-emptive, that is, a lower priority object that has been given the shared resource is interrupted when a higher priority object requests it. In non pre-emptive methods, the low priority objects are always allowed to complete. The pre-emption process can also occur when the maximum allowed reservation time has been exceeded decreasing the risk of starvation. Another dynamic arbitration method is the round-robin method. In round-robin systems, the resource owner is changed in a circular manner providing fair arbitration because every object eventually gets the resource ownership. Each object passes the ownership to the next one in line when it no longer needs the resource or it has held the resource for the maximum allowed time.
3.5. Physical interconnections

Physically, interconnection networks are formed with signal wires. Because different architectures require different number of wires with varying lengths, modelling the interconnect wires is a crucial step in evaluating interconnection architectures. This field has seen many extremely accurate interconnection models [Sak93][Deu01][Sve02][Liu04], however, the simplified versions of [Ho01][Syl01] are accurate enough for the analyses of this Thesis.

This analysis studies the delay of the interconnection wire and how it is predicted to change according to the *Interconnect roadmap* of [ITRS03]. For the modelling of interconnections, their geometries are needed. In Figure 3.12, the black parts depict the interconnection wire and the grey part the dielectric insulator surrounding the actual wire in an integrated circuit. The thickness, the width, the spacing, and the barrier of the wire are needed as modelling parameters. A frequently used term is the aspect ratio (A/R) of a wire that stands for thickness of the wire divided by its width.

The simplified Equation 3.1 for delay used here is based on the assumption that the delay is dominated by resistance and capacitance effects and does not include any inductive terms as argued in [Ho01]. The formula was developed for unrepeated wires, and therefore, gives an upper bound for the delay of possibly repeated global wires. The terms in the Equation 3.1 represent the resistance of the wire ($R_{wire}$), the capacitance of the wire ($C_{wire}$), and the delay of an inverter driving four identical copies of itself ($FO_4$). According to [Ho01] $FO_4$ is about 90 ps for 0.18 µm CMOS technology and decreases by 14% at each low-power technology node according to [ITRS03].

$$Delay = FO_4 + \frac{1}{2} R_{wire} C_{wire}$$

3.1

The capacitance of the wire can be estimated with Equation 3.2 [Ho01] where $\varepsilon_0$ is the permittivity of vacuum ($8.85419 \times 10^{-12}$ F/m) and $\varepsilon_{\text{horiz}}$ and $\varepsilon_{\text{vert}}$ are
the relative permittivities of the used insulator material in the horizontal and vertical directions which might be different for future dielectrics. The $K$ parameter can be varied between 0 and 2 to model Miller multiplication [Ho01]. The fringe term depends on geometry and is about 40 pF/µm for 0.18 µm CMOS technology. In addition, $ILD$ that stands for inter-level dielectric is here assumed to be the same as the barrier width.

$$C_{\text{Wire}} = \varepsilon_0 \left(2K\varepsilon_{\text{horiz}} \frac{\text{thickness}}{\text{spacing}} + 2\varepsilon_{\text{vert}} \frac{\text{width}}{ILD_{\text{thick}}} \right) + \text{fringe}(\varepsilon_{\text{horiz}}, \varepsilon_{\text{vert}}) \tag{3.2}$$

The resistance of the wire can be estimated with [Ho01], where $\rho$ is the effective resistivity of the wire which is 2.2 $\mu\Omega$*cm for copper wires [ITRS03]. This Equation assumes an equal sized barrier thickness on the bottom and the sides of the wire.

$$R_{\text{Wire}} = \frac{\rho}{(\text{thickness} - \text{barrier})(\text{width} - 2\text{barrier})} \tag{3.3}$$

The parameters needed in the equations can be gathered from [ITRS03] and are given in Table 3.4.

<table>
<thead>
<tr>
<th>Year</th>
<th>2003</th>
<th>2006</th>
<th>2009</th>
<th>2012</th>
<th>2015</th>
<th>2018</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier [nm]</td>
<td>12</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Spacing [nm]</td>
<td>475</td>
<td>320</td>
<td>234</td>
<td>165</td>
<td>117</td>
<td>83</td>
</tr>
<tr>
<td>$\varepsilon_{\text{horiz}}, \varepsilon_{\text{vert}}$</td>
<td>3.45</td>
<td>3.35</td>
<td>2.85</td>
<td>2.45</td>
<td>2.2</td>
<td>2</td>
</tr>
<tr>
<td>A/R</td>
<td>2.1</td>
<td>2.2</td>
<td>2.3</td>
<td>2.3</td>
<td>2.4</td>
<td>2.5</td>
</tr>
</tbody>
</table>

Based on Equations (3.1, 3.2, and 3.3) and the parameters of Table 3.4, the resistance, the capacitance, and the delay of a fixed size, 1 mm long and 0.1 µm wide, unrepeated interconnect wire can be calculated. In Figure 3.13, the resistance and capacitance are depicted. Because of the increasing aspect ratio, the resistance of the wire is decreasing slowly. On the other hand, the capacitance is growing at a faster rate.
Figure 3.13 Resistance and capacitance of a 1 mm long and 0.1 µm wide wire.

In Figure 3.14, the wire delay is depicted. As can be observed, the rise of the capacitance leads to an increasing rise of a fixed size wire delay. This is clearly a problem for the future SoCs.
3.6. Network-on-Chips

The most obvious way to connect functional blocks in SoCs is to have a dedicated link between every component. Between this kind of fully connected network and a shared path bus exists a wide variety of networks with different area and performance trade-offs. In this Thesis, a SoC architecture that is based on more complex structures than a single bus or fully connected point-to-point links is defined as a network-on-chip.

This Thesis concentrates mainly on bus-based interconnections. Nevertheless, also more complex interconnection architectures are sometimes needed in system design. It has been argued that these network-on-chips will be used to go around the wiring and complexity problems of buses [Box95][Dal01][Ben02][Wie02]. This is why certain important features of presented NoC schemes [Jan03, pp. 3-106] are discussed in this Chapter.

3.6.1 NoC architectures

The presented NoC architectures are usually based on communication networks, multiprocessor architectures for scientific computing, or computer local area networks. In addition, embedded system networks, such as the RapidIO [Rap02], which is a serial packet-switched chip-to-chip interconnection, could also be used on a chip.
Communication networks, like telephone networks, have many things in common with the NoC ideas since they are both intended to be used to interconnect large-scale systems. However, communication networks must support any arbitrary topology that can also be modified after design time. In addition, latency is not usually a problem, except in the real-time traffic. In NoCs, the architecture is usually fixed at design time and cannot be changed after that. NoCs also prefer the simple routing algorithms provided for regular topologies because of their small implementation area. In addition, some applications and communication nodes need latency guarantees and the amount of memory that is feasible to use in a NoC is quite limited.

The homogeneous multiprocessor architectures are one possible starting point for NoC design. They have been used to run fine-grained parallel scientific algorithms at very high speeds. The Amdahl’s law [Amd67] states that the serial part of the program starts to dominate the overall run-time of the application when it is parallelized. Critics of the law have pointed out that it is usually the problem size that scales with the number of processors which implies high achievable parallelisms in many applications [Gus88]. However, the logical limits of parallelism inherent in most practical applications result in redundancies and inefficiencies that cannot always be tolerated in on-chip systems because of increased area and power consumption.

In addition, the structures utilized in scientific computing are usually optimized for fine-grain granularity computation. Sometimes NoC architectures are benchmarked with a fairly simple fine-grained algorithm with good results although the distribution of this very limited amount of computation for such a large system may not be meaningful. The NoC scheme implies that the components are part of a fairly large coarse-grain granularity system where the components are smaller systems of their own.

A computer local area network (LAN) is another possible analogy for the design of NoCs. Although the on-chip architectures have a lot in common with these systems, they also have considerable differences. The relatively complex network architectures and protocols of LANs as well as their large network buffer memories are designed to make the systems reliable with particular emphasis on connectivity and performance requirements. In contrast, large buffers and arbitrary connectivity are not tolerable in SoCs that prefer limited complexity and real-time operation.

### 3.6.2 Example NoC schemes

The published NoC approaches are highly speculative since very few practical experiences of their applicability have been presented. However, they all contain interesting features that might become attractive alternatives in the future.

There are several purely circuit-switched NoC approaches. In addition, some bus architectures support also other circuit-switched schemes. An example is
the Wishbone [Sil01] bus architecture that can also be configured as a crossbar. Circuit-switched schemes are tabulated in Table 3.5. The name of the scheme, the network type, a brief characterization, and a reference is given of each case.

PADDI (Programmable arithmetic devices for high-speed digital signal processing) [Che92] that is used for rapid prototyping, is an architecture where a dynamically controlled crossbar is used to interconnect specialized execution units.

The PROPHID template [Lei98] is used to interconnect general purpose processors and application domain specific processors. For this, it uses a central bus and a three-stage circuit-switched network called T-S-T network (time-space-time). Bandwidth is assigned at compile time to data streams as needed and the remaining bandwidth is assigned at run-time.

The aSOC [Jia00] (Adaptive system-on-a-chip) is a circuit-switched mesh interconnection that is based on compile-time scheduled communication that can be reconfigured at run-time. This is enabled by the crossbar switches in each mesh node that can be programmed separately for each system cycle. The designers are also evaluating an approach where some cycles can be left for dynamic routing, in effect forming a packet-switching scheme.

SoCBUS [Wik03] is a circuit-switched network based on a 2-D mesh. It uses a packet connected circuit scheme, where a packet is send through the circuit-switched network and the path is formed as the transfer advances.

Table 3.5 Circuit-switched NoC schemes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Network</th>
<th>Characteristics</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PADDI</td>
<td>Crossbar</td>
<td>Dynamically configurable crossbar</td>
<td>[Che92]</td>
</tr>
<tr>
<td>PROPHID</td>
<td>Bus and 3-stage</td>
<td>Central bus and switching network</td>
<td>[Lei98]</td>
</tr>
<tr>
<td></td>
<td>network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aSOC</td>
<td>2-D mesh</td>
<td>Compile-time scheduled communication</td>
<td>[Jia00]</td>
</tr>
<tr>
<td>Wishbone</td>
<td>Bus or crossbar</td>
<td>Central bus or configurable crossbar</td>
<td>[Sil01]</td>
</tr>
<tr>
<td>SoCBUS</td>
<td>2-D mesh</td>
<td>Packet forms the path on the way</td>
<td>[Wik03]</td>
</tr>
</tbody>
</table>

48
Most of the newer NoCs utilize packet-switched approaches. They are tabulated with the name of the scheme, the network type, a brief characterization, and a reference in Table 3.6.

The SPIN (Scalable, programmable, integrated network) [Gue00] is based on an adaptive fat tree network and packet-switched wormhole routing.

The network-on-chip platform called CLICHE (Chip-level integration of communicating heterogeneous elements) [Kum02] is an interconnection methodology consisting of an architecture and a way to map applications to the architecture. The topology of the architecture is a 2-D mesh of switching components and resources.

PROTEO [Saa02] is a packet-switched network scheme that can implement several topologies. At the moment only a ring architecture has been published but also star and mesh topologies are planned.

SoCIN (SoC Interconnection network) [Zef03] is packet-switched network based on wormhole routing. It uses a soft core parameterizable switch to implement 2-D topologies like mesh and torus.

QNoC (Quality-of-service NoC) [Bol04] is based on 2-D mesh topology and wormhole packet routing. The mesh is used to implement irregular structures, where the sizes of the connected processing nodes can vary and a single router can be used to connect one or several processing nodes.
Table 3.6 Packet-switched NoC schemes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Network</th>
<th>Characteristics</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIN</td>
<td>Fat tree</td>
<td>Adaptive fat tree based on wormhole routing</td>
<td>[Gue00]</td>
</tr>
<tr>
<td>CLICHE</td>
<td>2-D mesh</td>
<td>Architecture and design methodology</td>
<td>[Kum02]</td>
</tr>
<tr>
<td>Proteo</td>
<td>Ring, (tree and mesh)</td>
<td>Implements several different topologies</td>
<td>[Saa02]</td>
</tr>
<tr>
<td>SoCIN</td>
<td>2-D mesh or torus</td>
<td>Wormhole routing with a soft core switch</td>
<td>[Zef03]</td>
</tr>
<tr>
<td>QNoC</td>
<td>2-D mesh</td>
<td>Wormhole routing with an irregular structure</td>
<td>[Bol04]</td>
</tr>
</tbody>
</table>

The rest of the proposed NoC architectures utilize combinations of circuit-switching and packet-switching. These hybrid schemes are tabulated in Table 3.7. The name of the scheme, the network type, a brief characterization, and a reference is given of each case.

Octagon [Kar02] is a network architecture that can be either circuit-switched or packet-switched. The basic building block is an eight node ring where each node has a connection to the opposite side of the ring. Architectures with more nodes than eight can be formed by connecting these basic components together by letting a node belong to two distinct basic components.

ÆTHEREAL [Goo02] is a circuit-switched and packet-switched network that has two different communication path guarantees: some have guaranteed throughput and some are only served best effort. For packet-switching it uses wormhole routing and for circuit-switching a form of time-division multiplexed switching.

The RAW machines [Wai97] implement a minimal set of hardware resources. They are composed of homogeneous tiles which contain memories, an arithmetic logic unit, registers, configurable logic, and a programmable switch. The switch scheduling is static but also a dynamic wormhole routing mechanism is enabled by the system.
Table 3.7 Hybrid NoC schemes.

<table>
<thead>
<tr>
<th>Name</th>
<th>Network</th>
<th>Characteristics</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octagon</td>
<td>Ring with connections to the opposite side</td>
<td>Architecture formed by 8 node structures</td>
<td>[Kar02]</td>
</tr>
<tr>
<td>ÆTHEREAL</td>
<td>Not stated</td>
<td>Wormhole and time-division multiplexed</td>
<td>[Goo02]</td>
</tr>
<tr>
<td>RAW</td>
<td>2-D mesh</td>
<td>Static and dynamic routing</td>
<td>[Wai97]</td>
</tr>
</tbody>
</table>
4. Buses in on-chip digital systems

Buses are shared interconnection paths between the components of an electronic system [Gus95]. Their implementations are defined by their electrical signalling, the required interface logic, and the utilized transaction protocols. The basic properties of bus interconnections have remained almost unchanged during their evolvement from computer backplanes to on-chip interconnections.

Buses have been the preferred interconnection in most of the processor-based systems in the past years. They have also been the basic building blocks of almost all implemented system-on-chips. Buses offer a simple and cost efficient way to transfer data between the components of a system. The switch and wire costs as well as the latencies induced by long paths are considerably lower in buses than in other networks as tabulated in Table 3.2 and Table 3.3. The simple signalling and structure of buses facilitate both manual and automatic generation of bus-based architectures. Moreover, the simple programming model of bus-based systems has been favoured by the application designers and has led to the development of advanced programming tools.

Single bus is a good choice for many systems when the number of connected components is small. However, as mentioned earlier, in the future SoCs, the complexity of a single component is not likely to significantly increase because only relatively simple components can be scaled with technology [Ho01][Syl01]. Therefore, the number of components in a SoC is increasing. On the other hand, there has been only a modest increase in the ability of physical wires to transfer signals, although the performance of logic gates has been rapidly improving. The delay of a fixed length wire is even estimated to increase with decreasing feature sizes as was presented in Section 3.5 and [Syl01].

The requirements for SoC interconnections were given in Table 2.1. In addition, interconnection wire parameters were tabulated in Table 3.4. Based on these tables and the formula given for the interconnect wire delay in Equation 3.1, the number of processing nodes that can be connected by a single bus can be estimated. In the presented estimation, the number of nodes is assumed to be limited by the required operating frequency of Table 2.1. As additional assumptions, the area of the connected nodes is assumed to be 100 kgates [Syl01] and the wire width scaled with technology generation feature size.

The results of the estimation are depicted in Figure 4.1. The node counts are achieved by increasing the count until the operating frequency requirement is violated. As can be observed, the viable number of nodes varies between 8 and 14. Based on this analysis, it seems that a simple bus is no longer a preferable solution for SoC interconnection requirements. However, there are several bus
enhancement schemes that make them applicable in large-scale systems. In a future SoC, the basic single component is not likely to be a single processing node but a single bus system containing several nodes. These clusters can then be connected with bus hierarchies and dedicated network structures.

![Graph showing the maximum number of processing nodes in a single bus as a function of technology generation.](image)

**Figure 4.1** The maximum number of processing nodes in a single bus.

### 4.1. Structure

By definition, a bus uses shared interconnection lines as a fixed path between the sender and the receiver of transactions. Traditionally, computer backplane buses have been implemented with three-state buffers which drive bidirectional signal lines as illustrated in Figure 4.2 a). Three-state buffers are a good approach for backplane buses but their high power consumption and the difficulty of debugging make them less suitable for SoCs. The basic alternatives for three-state buses are multiplexer-based buses and AND-OR structures depicted in Figure 4.2 b) and Figure 4.2 c), respectively. They both use unidirectional signal lines.

The processing nodes connected by a bus are called *agents*. An agent that can initiate and control a transfer is called a *master* or an *initiator*. Agents that respond to transfer requests initiated by masters are called *slaves* or *targets*. An agent can also be a master and a slave at the same time. An agent can contain,
for example, a processor, its local bus, and memory units. In addition, an interface *wrapper* is used to connect to the global bus architecture.

Figure 4.2 a) Three-state, b) multiplexer-based, and c) AND-OR bus implementations.

In a general case, the bus architectures use several *parallel* interconnection lines to achieve maximum throughput. Nevertheless, buses that transmit data *serially* are sometimes used to help the signal routing and minimize the bus signal line area. The most simple bus implementation is the single bus, but also hierarchical and multiple bus structures are frequently utilized. These three basic bus architectures were presented in Chapter 3.2.1.
4.2. Transfers

On the basis of supported transfer types, buses can be either synchronous or asynchronous. Synchronous transfers are based on a global clock signal while the asynchronous transfers are self-timing [Bhu89]. Transfers in a bus normally require some sort of *handshaking* between the sender and the receiver.

A generic, two-phase synchronous read operation consists of a read request by the master (time instance 1 in Figure 4.3 a) and a data response phase by the slave (time instance 2). In a synchronous bus, the handshaking does not need to be as extensive but the problem is that the slowest agent defines the clock frequency of the whole system. Furthermore, distributing a global clock signal in a large SoC can be problematic.

Asynchronous transfers do not need a global clock signal. Instead, each phase of the transaction process takes only as long as is required, which makes it easier for devices with different response times to interact. An additional benefit is that signal propagation delays do not affect the correct operation of asynchronous systems. However, because of the prolonged handshaking, asynchronous transfers are usually more complex than their synchronous counterparts.

The asynchronous transfers frequently use a four-phase handshaking scheme. The four phases of a generic read operation are: read request by the master (time instance 1 in Figure 4.3 b), acknowledge by the slave (time instance 2), data response read by the master (time instance 3), and end of transfer by the slave (time instance 4).

Traditionally, a transaction between two agents reserves the bus for the duration of the transaction as in Figure 4.3 a) and b). If, for example, a processor reads from a memory it first sends a read request, stalls its own processing, and waits for the memory to respond. The memory is usually slow when compared to the processor and the shared medium is reserved during the whole process. In a *split transfer*, the processor first sends a read request, after which it frees the bus for other transactions. When the memory is ready, it responds with the requested data. The processor does not get data any faster but the advantage is that the shared medium is available for other agents in the middle of the transaction. This makes it possible to pipeline multiple outstanding transactions. An example of a generic synchronous split transfer is depicted in Figure 4.3 c) with the same phases as in the traditional synchronous transfer.
In a single bus, only one agent can transmit data at a time. On the other hand, all connected agents can read the transferred data making broadcast a simple operation. Buses can have separated data and address lines, but they can also be multiplexed onto the same lines. In a multiplexed bus, typically the master first sends the address and then the data. One way to speed up the operation with separate bus signals for data and address is to put the address of the next transfer to the address lines before the previous transfer has ended. This technique is called transfer pipelining and it is particularly good with block or burst transfers in which a single address cycle is followed by multiple data cycles.

With the mentioned transfer techniques, it is easy to achieve very high peak throughputs. The sustainable or normally achievable throughputs, on the other
hand, are usually much smaller than the maximum values. Latency in a general bus transfer is constituted by four phases [Kuu01, p. 28]. Arbitration latency is the time it takes for the sender to request a permission to use the bus, wait for other transfers to complete, and then be granted the ownership. Initial latency is the time it takes to initialize the transfer, wait until the sender and receiver are ready, and then transmit the first data. Subsequent data latency is then the time it takes to transmit the following data items. The transfer is ended with turn-around latency which is the time it takes for the bus to return to an idle state. All these latencies diminish the achievable throughput although the different phases of the transfer can be pipelined for better throughputs.

4.3. Bus arbitration

Bus arbitration is the mechanism for resolving the bus owner. Arbitration can be centralized in which case request and grant signals are needed between every agent and the arbiter unit. In a distributed version, all the information needed for the arbitration process is stored in each bus agent and can be updated by monitoring the bus signals. In addition to an arbiter, a decoder is required. The decoder does not need to be a separate unit, it can be integrated as a part of a slave. The task of the decoder is to direct the correct slave to respond to a transaction. In the centralized scheme of Figure 4.2 b), the multiplexer on the left is controlled by the arbiter and the multiplexer on the right by the decoder.

The most popular arbitration algorithms are based on priority or utilize the round-robin method. In addition, static methods where bus owners are decided before the system run-time are used. An example of these methods is TDMA in which time frames are divided into timeslots that are assigned to bus agents. An interesting approach is taken in the Lotterybus [Lah01] specification which defines a partly random arbitration scheme.

Arbitrating a bus with multiple agents can be a very complex and time consuming task. This is also one of the features of buses that make them hard to scale, because the more agents there are, the longer the arbitration tends to take. The agents of a bus can also have completely diverging bandwidth requirements, and therefore the arbitration scheme can be very complex to implement.

In the arbitration scheme of STBus [Ack00], the data and address buses are arbitrated separately. Each transfer has to first request an access to the transfer medium on the address bus. Upon being granted an access to the bus, the agent gets a transaction ID and can begin the actual transaction on the data bus. In this scheme, the data bus is used for data transactions and at the same time the next transfer is already being arbitrated on the address bus. During the arbitration process, the agents can also negotiate different transaction
parameters, such as the maximum length of the forthcoming transfer. This pipeline scheme works best in systems having long transfers.

Distributed arbitration does not require the existence of a centralized arbiter. In addition, distributed arbitration does not need the point-to-point signals for requesting and granting the bus ownership. This improves scalability since the routing of the point-to-point signals can be troublesome. A distributed TDMA and round-robin/priority-based arbitration is implemented for example in [Son00a] and [P 4]. The time periods when the agents are guaranteed a bus access and the mechanism for acquiring the bus during other time periods is coded into a dedicated memory block located in the agents.

4.4. Support for complex structures

By definition, buses are static networks since they do not have dynamic switching components in them. Although the structure cannot be modified, there are several interconnection related parameters that can be tuned during system operation. This increases the performance of the interconnection and makes it more flexible.

In the interconnection scheme presented in [P4], system parameters are initialized during system synthesis but they can also be modified at run-time with dynamic reconfiguration [Kuu99]. Reconfiguration uses specific write commands that modify the configuration memories located in each interconnection wrapper and bridge of a system. The parameters include arbitration parameters like priorities, TDMA parameters, and longest allowable bus reservation times, and power mode, as well as address ranges of the agents. This way, the latency and the allocated bandwidth of the components can be modified at run-time, following the variable communication requirements placed on the connected components. Single bus architectures can also be extended to hierarchical and multiple buses; in effect forming a network-on-chip.

Single bus networks only allow the execution of one transfer at a time since they share a single transfer medium. This limitation applies also to the typical hierarchical structures utilizing circuit-switching where a transaction from one bus segment to another reserves both buses. The lack of parallel transfers makes also the scaling of bus-based systems difficult, because each added agent decreases the bandwidth available to the other agents. It should be noted that the parallelism found in many practical applications is quite limited making even a single bus sufficient for these applications. However, the number of parallel transfers in a bus-based system can be increased by utilizing multiple bus structures. In practice, the amount of agent I/O and interconnection area limit the use of this technique.
One solution to the problem is the hierarchical structure presented in [P1]. In this scheme, the bus segments are separated by bridges that have buffer memory. The bridges can be seen as agents that have to arbitrate for a bus segment since they are implemented with two wrappers as depicted in Figure 4.4. A circuit-switched approach reserves all the segments between the source and the destination. Because the interconnection has bridging components with buffer memory, the resulted hierarchical bus could also be defined as a packet-switched network and the bridge components routers. It is possible to have parallel transfers in every segment simultaneously. The downside of this technique is the added latency caused by the bridges. An example of this kind of a hierarchical structure is illustrated in Figure 4.4.

Another way of dealing with complex bus structures is to generate the interconnection architecture automatically. One example is the component-based design automation approach presented in [Ces02]. It is meant to be used with multiprocessor SoC platforms. In this approach, the architecture is first specified in an abstract way, after which the system is analyzed to determine a
suitable hardware/software partition. Following this, the interconnection components are generated automatically with the help of hardware and software wrapper generators that utilize pre-defined library components.

4.5. Bus clocking

The size and the clock frequencies of the chip designs scale up very rapidly, and therefore, it will not be possible to transfer signals of a future NoC within a single clock cycle solely based on a global clock. It has been estimated that the propagation delay from edge to edge of 50 nm technology chips will be between six to ten clock cycles [Ho01]. There are at least three ways to tackle this problem: using asynchronous techniques, multiple clock domains, and latency insensitive protocols.

An example of an asynchronous transfer technique is the Marble bus architecture presented in [Bai00]. It uses a four-phase handshaking scheme similar to the one presented previously. Despite the lack of a global clock, all the required features of a high-performance on-chip bus are provided. All this can be achieved with reduced power consumption because of the asynchronous transfers lacking the clock distribution needed in its synchronous counterparts. Different subsystems can also run at different rates, completely independent of each other. This feature also helps the utilization of modular design techniques.

Another option would be to use globally asynchronous, locally synchronous designs (GALS), where transactions within components are synchronous and the data traffic between components asynchronous. This technique has proponents [Hem99] and critics [Mal03]. A performance decrease of 5-15 % with a power reduction of 10 % is reported in [Iye02] in a 5-clock domain GALS processor when going from a fully synchronous approach to GALS.

Locally synchronous operations are a basic practical requirement for efficiently designing correct functionality in large digital systems [Kea99, pp. 29-36]. The difficulty in providing a global clock can be overcome with the use of multiple clock domains providing a GALS-type solution. For example in [P1], the clock domains are isolated from each other by bridges or wrappers that have FIFO buffers inside them. The wrappers take care of the handshaking between the bus segments operating on different clock frequencies. This results in very small area overhead but an additional buffering latency.

An example structure is depicted in Figure 4.4. A noteworthy issue is that all the agents of a bus segment are not required to use the same internal clock frequency. This is achieved by letting the wrapper unit of the agent operate at the same frequency as the rest of the bus segment but using another clock frequency for the rest of the internal components of the agent. By exploiting FIFO buffers and handshaking, the attached IP block can operate at any suitable frequency.
The latency insensitive design lets the system components use a single clock and, therefore, apply to the globally synchronous model. The global signals that cannot travel the whole distance in one clock cycle are pipelined to be on flight several clock cycles. One example are the methods proposed in [Car02] where single components called pearls are encapsulated in shells that contain buffers and control logic for making the transfers latency insensitive. Next, the physical layout is designed after which pipeline components or relay stations are inserted to the lines having a latency of over one clock cycle. This method has been developed for point-to-point connections but could also be applied to many bus schemes. Similar method is also applied in [ARM03a] where register stages can be added to interconnection paths which increases the latency but can also increase the achievable operating frequency.

4.6. Bus signal power efficiency and robustness

The long interconnection wires of buses cause also other problems besides the implementation of the clock distribution network, such as significant power consumption and increased noise sensitivity. Several techniques for resolving this problem both at the signal and at the system-level have been presented as described in the following.

In a split-bus architecture [Che02], the bus is divided into segments with three-state buffers to achieve multiple simultaneous transfers. The technique can also be used for reducing the problems with long bus lines. If the delay and energy consumption of the three-state buffer structure is smaller than the part of the bus that is being disconnected, then split-bus architecture is preferable.

With split-bus technique, the parasitic loads and, therefore, energy consumption and noise problems can be decreased considerably. For example, [Che02] reports energy savings of 16-50 % with the split-bus technique by using heuristic techniques to optimize the bus splitting. Split-bus technique is particularly suitable for asynchronous buses which can also utilize the shorter propagation times. Synchronous buses, on the other hand, usually set the clock frequency according to the worst case propagation time.

The energy efficiency and robustness of bus signal lines can be made better by many techniques at circuit and system-level. Circuit-level techniques include the low-swing techniques reviewed in [Zha00]. They can be used to reduce energy consumption but careful design of interconnect circuitry is needed to avoid robustness problems. In addition, methods for reducing on-chip inductance have been reported in [Mas02]. These include shielding the signal line with power or ground lines, widening the metal lines, increasing the metal to metal separation, inserting buffers, and utilizing differential signalling. The reduction of impedance improves the timing predictability and decreases the signal delay and crosstalk.
At the system-level, the power consumption of long interconnection wires can be lowered by different signal encoding techniques because the activity of the bus signals considerably affects the power consumption. One proposed technique for bus encoding is the bus-invert scheme presented in [Sta95]. In this scheme, the Hamming distance between the data previously on the bus and the data being transmitted is first calculated. If the Hamming distance shows that more than half of the data has changed, the inverse of the data is sent. The receiver knows the encoding of the data from a dedicated invert bit. If the Hamming distance of the data shows that less than half of the data has changed, the data is sent as it is. This simple technique has been used to achieve a 25% average power consumption reduction in bus interconnections [Sta95].

Other encodings, presented in [Mus98] and [Agh02], are based on the fact that programs usually utilize memory locations that are close to each other. These zone-based methods have been utilized to achieve switching activity savings of 24.9-77.3% [Agh02] in the memory address buses.

Long, parallel lines needed in buses increase the rate of faulty behaviour. This is particularly due to crosstalk and dynamic delay which is caused by the changing capacitance seen by a gate. The fact that long wires need to be narrow for high wiring densities and thick for lower resistance only makes these problems worse by increasing the coupling capacitance [Syl01]. One solution is to use bus guardians as in [Laj01]. In this method, dedicated modules constantly monitor the bus looking for errors. The bus guardians also provide error correction mechanisms. Several error detection codes and their fault-detection capabilities with respect to crosstalk errors are analysed in [Fav99].

Long lines can also be broken into smaller ones by utilizing hierarchy and repeaters. In hierarchical systems, bridges split buses into segments. In this process, the grouping of the components forms the basis after which layout-level techniques can be used to optimize the transfer line lengths. In single bus solutions, one has the option of utilizing repeaters and other low-level optimization techniques as presented, for example, in [Kah99].

### 4.7. Quality of service

Many modern SoC applications would benefit from an interconnection that could give some guarantees for the transactions. A frequently used term is the quality-of-service (QoS), which implies that different agents need different guarantees for data transactions. Here, two examples are shown. One demonstrates the applicability of TDMA for guaranteeing bandwidth and the other for providing smaller latency.

Real-time requirements demand that a system will respond to certain events during a specified time interval. In many systems, the occurrences of these events can be predicted in advance with high accuracy. For example, TDMA is
used as an arbitration method in [Son00a] to guarantee real-time requirements. In this method, time is divided into a repeating time frame that is again divided into timeslots that are allocated to each agent according to their bandwidth needs. This way, all agents are guaranteed the use of the interconnection medium when they need it. The unallocated timeslots and the timeslots that are not used at run-time by their owner are arbitrated with a second-level scheme that is based on round-robin arbitration. The second tier method can be utilized by components requiring unpredictable, irregular, or low-priority transfers.

Almost all on-chip buses incorporate a type of a priority scheme. Usually, this concerns only the transaction order of the agents. An individual agent might have data transactions that demand higher priority. The scheme presented in [P1] has two different priorities for two types of transfers: data transfers and messages. All the interconnection components (wrappers and bridges) have two sets of FIFO buffers, one for each type. The idea is that the higher priority messages can bypass the data transfers by using their own dedicated buffers. This concept can be extended by adding more priority specifiers and FIFO buffers to implement multiple virtual channels with different priority.

### 4.8. Standardization

Many system buses exist for SoCs. In addition, the processors inside the SoCs incorporate their own local buses. It is possible to fit them together, but having too many interfaces and adapters can lead to problems. The traditional approach for introducing modularity into a system has been to standardize the interconnection network itself, for example a bus, and require all system IP blocks to support it. However, agreeing upon one all-purpose interconnection standard has proven to be unsuccessful due to commercial issues and disagreement over required features. In addition, different applications require different trade-offs. A somewhat more successful approach has been to standardize the interfaces instead of the interconnection, which allows vendors and designers to use their own buses. From this point of view, it would be better if all buses would comply to a single interface standard. Two proposed standards are the virtual component interface (VCI) [VSI99] and the open core protocol (OCP) [Son00b].

The objective of the VSI alliance [Bir99] has been to specify a general interface that can be utilized in connecting any IP block, in the form of a virtual component (VC), in any SoC by any integrator. VSI alliance has proposed a VC interface (VCI) specifying a protocol for requests, a protocol for responses to these requests, and the contents and coding of these requests and responses.

The VCI specification does not specify a bus, but a point-to-point interconnection between a VCI initiator (master) and a VCI target (slave), in
which the actual physical interconnection is left unspecified. Figure 4.5 depicts this kind of an interconnection architecture. If a bus agent wants to act both as an initiator and as a target, it has to have both the initiator and the target interfaces. Once the wrappers for a bus have been designed, all VCs can be connected to that bus.

![Diagram of virtual component interface scheme]

Figure 4.5 The Virtual component interface scheme.

VCI defines a protocol which can be used by components in communicating with each other regardless of the physical bus implementation. OCP is a superset of the VCI specification. It has the same dataflow aspects as VCI, but includes additional control and test signalling.

### 4.9. Published on-chip bus schemes

In this Section, the basic properties of the following nine on-chip buses are studied: AMBA with two versions AHB [ARM99] and AXI [ARM03a], CoreConnect [IBM01], CoreFrame [Cor99], HIBI [Kuu98][P4], Marble [Bai00], PI-bus [OMI96], SiliconBackplane [Son00a], STBus [Ack00], and Wishbone [Sil01]. A number of other SoC buses exist also but not enough details of their properties are publicly available to allow a fair comparison. This analysis is partially based on [Sal02].

The advanced microcontroller bus architecture (AMBA) by Advanced RISC machines Limited (ARM) is one of the most frequently used on-chip buses. Reasons for this popularity include the easily available specification and the close connection to the popular ARM RISC (reduced instruction set computer) [Pat80] processors. AMBA is a typical multiplexer-based bus and it utilizes a central arbiter and decoder architecture. It has three versions for different transaction needs, namely the advanced high-performance bus (AHB), advanced system bus (ASB), and the advanced peripheral bus (APB). AHB is a
system backbone bus whereas ASB is a system bus that does not have all the features of AHB. APB is targeted for low power and low complexity peripheral devices and can be used together with the other two AMBA bus types.

The AMBA advanced extensible interface (AXI) is a new high bandwidth, low latency bus architecture. Its objective is to be backward compatible to AHB and APB interfaces. All the transactions in AXI are burst-based and can facilitate multiple outstanding addresses and out-of-order transaction completion. The AXI architecture consists of four distinct channels: address, read, write, and write response. The preferred architectures are ones with shared address/data buses, shared address and multiple data buses, as well as with multiple address and data buses.

CoreConnect is another widely used on-chip bus developed by International business machines Corporation (IBM). It is an open standard with publicly available specifications and it is used, for example, in conjunction with the PowerPC processors. CoreConnect specification contains two buses for different purposes, namely the processor local bus (PLB) and the on-chip peripheral bus (OCP). Note that here the OCP abbreviation does not mean the open core protocol [Son00b]. In addition, it has a separate device control register (DCR) bus and an arbiter unit supporting address pipelining and bridges between PLB and OCP.

CoreFrame, by Palmchip Corporation, is also a two-level on-chip bus architecture. It consists of PalmBus which is used for connections between processors and peripherals and MBus for high-speed memory and peripheral accesses. The two buses are actually independent parallel buses allowing concurrent activity and increasing the throughput of the system. CoreFrame uses point-to-point signals and multiplexing to avoid three-state buffering. It does not use shared signal lines, instead the communication is carried out through shared variables in memory.

Heterogeneous IP block interconnection (HIBI) is a versatile interconnection scheme for continuous-media SoCs. The basic HIBI properties are listed in this comparison, although the scheme is presented in more detail later in this Thesis.

Developed at the University of Manchester, Marble (Manchester asynchronous bus for low energy) is an on-chip bus lacking a global clock signal. The bus supports split transfers, test structures, and bridges, and demonstrates that the requirements of a SoC bus can be met by a fully asynchronous design style. For arbitration, Marble uses a centralized, tree-based structure that can be modified to adjust the latency and bandwidth allocated to each agent of the system.

The PI-bus is an open standard published by the Open microprocessor systems initiative (OMI). VHDL codes for master, slave, master/slave, and bus control units are freely distributed. In addition, synthesis scripts for different ASIC and
FPGA technologies and system examples are available. The bus control is a centralized unit taking care of master arbitration and slave selection (decoding). It also logs all the error messages coming from the slave units. The PI-bus is a synchronous bus utilizing non-multiplexed address and data signals and it supports multiple masters. In addition, bridges have been used in systems based on the PI-bus [Dut01].

The μNetworks designed by Sonics Incorporated contains a set of architectures and tools for SoC design. The defined architectures are the SiliconBackplane for on-chip and MultiChip for off-chip interconnections. SiliconBackplane has a two-level arbitration scheme based on TDMA and round-robin. Furthermore, dynamic reconfiguration of a set of system parameters is supported in SiliconBackplane. The system parameters are implemented in the agents as registers that are visible to the application software.

The Split transaction bus (STBus), used in the scalable Daytona DSP architecture, has many interesting properties that were discussed earlier. It was jointly designed by Lucent technologies and Massachusetts Institute of Technology. The STBus was designed to minimize average latencies in systems where large amounts of data are transferred. This is achieved through a memory control unit that supports multiple outstanding transactions and the use of large cache memories. A programmable round-robin scheme is used for arbitration. The Split transaction bus should not be confused with the STBus of STMicroelectronics utilized for example in [Wod03].

Wishbone, by Silicore Corporation, is another on-chip bus standard with many typical SoC bus features included in its specification. Wishbone supports the basic transfer types and many of the possible bus implementation techniques, including multiplexer and three-state-based structures. The Wishbone interconnection can also implement other topologies than a bus such as a crossbar. The arbitration mechanism of Wishbone can be chosen by the system designer. The Wishbone VHDL code examples are freely distributed. The specification also includes a list of rules with which the Wishbone compatible components and systems have to comply, for example documentation and naming convention rules.

Table 4.1 tabulates the basic structural properties of the aforementioned buses, whereas Table 4.2 and Table 4.3 tabulate the transfer and arbitration properties, respectively. The notation is ‘x’ for supported features, ‘-’ if the feature is not supported, and ‘n/a’ if the information was not available or clearly stated in the specification. Sometimes the determination between supported and unsupported features is very difficult and a specific functionality can be achieved by using the interconnection a little differently.

There are many structural properties that are typical to almost all the SoC buses. Hierarchical structures are supported by many of the buses, and the rest of the specifications did not make a clear statement about the issue. The signal
line implementations include both uni- and bidirectional versions, as well as combinations of these two, although most of the practical implementations of these specifications utilize unidirectional signalling. Multiple clock domains and test structures are also supported by several of the bus specifications. The test schemes include generating the needed test vectors and the mechanisms for accessing the test ports inside manufactured systems.

Table 4.1 On-chip bus structures.

<table>
<thead>
<tr>
<th>Name</th>
<th>1. Hierar.</th>
<th>2. Dir.</th>
<th>3. Mul.</th>
<th>4. Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA AHB</td>
<td>x (APB)</td>
<td>Uni</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>AMBA AXI</td>
<td>x (APB)</td>
<td>Uni</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CoreConnect PLB</td>
<td>x (OCP)</td>
<td>Uni</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CoreFrame Mbus</td>
<td>x (PalmBus)</td>
<td>Uni</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>HIBI</td>
<td>x</td>
<td>Bi/Uni</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>Marble</td>
<td>x</td>
<td>Bi</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>PI-Bus</td>
<td>x</td>
<td>Bi</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>SiliconBackplane</td>
<td>*</td>
<td>Uni</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>STBus</td>
<td>-</td>
<td>n/a</td>
<td>-</td>
<td>n/a</td>
</tr>
<tr>
<td>Wishbone</td>
<td>-</td>
<td>Bi/Uni</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

1. Hierarchical structures
   Exception: * [Son00a] does not support, new versions will
2. Unidirectional (Uni) or bidirectional (Bi) signalling
3. Support for multiple clock domains
4. Test structures

Almost all the presented SoC buses use low-level handshaking based on dedicated handshaking signals. The typical SoC transfer types include split transactions and pipelined transfers. Although broadcast is almost a trivial application of the basic physical bus properties, its use is not explicitly stated in most of the specifications. This does not necessarily mean that the feature is impossible to implement with these schemes. Only one of the buses uses asynchronous techniques, the rest are fully synchronous.
Table 4.2 On-chip bus transfers.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA AHB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>n/a</td>
<td>Sync</td>
</tr>
<tr>
<td>AMBA AXI</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>n/a</td>
<td>Sync</td>
</tr>
<tr>
<td>CoreConnect PLB</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>n/a</td>
<td>Sync</td>
</tr>
<tr>
<td>CoreFrame Mbus</td>
<td>n/a</td>
<td>-</td>
<td>n/a</td>
<td>x</td>
<td>Sync</td>
</tr>
<tr>
<td>HIBI</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>Sync</td>
</tr>
<tr>
<td>Marble</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Async</td>
</tr>
<tr>
<td>PI-Bus</td>
<td>x</td>
<td>-</td>
<td>n/a</td>
<td>n/a</td>
<td>Sync</td>
</tr>
<tr>
<td>SiliconBackplane</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Sync</td>
</tr>
<tr>
<td>STBus</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>n/a</td>
<td>Sync</td>
</tr>
<tr>
<td>Wishbone</td>
<td>x</td>
<td>n/a</td>
<td>-</td>
<td>n/a</td>
<td>Sync</td>
</tr>
</tbody>
</table>

1. Dedicated bus control signals used for handshaking
2. Split transfers
3. Pipelined transfers
4. Broadcast support
5. Synchronous (Sync) or asynchronous (Async) transfers

The arbitration mechanism is a major differentiator between the bus interconnections. The bus specifications included one- and two-level techniques based on priorities, round-robin techniques, and TDMA. Application specific arbitration means that the protocol for requesting the bus ownership is specified, but the algorithm for granting it is left unspecified. The arbitration process was pipelined in most of the buses for performance reasons. Most of the SoC buses use centralized arbitration but some utilize distributed versions. Dynamic reconfiguration of at least some of the arbitration (for example priorities and TDMA parameters) and system parameters (for example address ranges and interrupt sensitivity information) was specified for five of the bus specifications.
Table 4.3 On-chip bus arbitration and reconfiguration.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AMBA AHB</td>
<td>AS</td>
<td>x</td>
<td>Cen</td>
<td>n/a</td>
</tr>
<tr>
<td>AMBA AXI</td>
<td>AS</td>
<td>x</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>CoreConnect PLB</td>
<td>1</td>
<td>x</td>
<td>Cen</td>
<td>x</td>
</tr>
<tr>
<td>CoreFrame Mbus</td>
<td>AS</td>
<td>x</td>
<td>Cen</td>
<td>x</td>
</tr>
<tr>
<td>HIBI</td>
<td>2</td>
<td>x</td>
<td>Dis</td>
<td>x</td>
</tr>
<tr>
<td>Marble</td>
<td>1</td>
<td>x</td>
<td>Cen</td>
<td>n/a</td>
</tr>
<tr>
<td>PI-Bus</td>
<td>AS</td>
<td>x</td>
<td>Cen</td>
<td>n/a</td>
</tr>
<tr>
<td>SiliconBackplane</td>
<td>2</td>
<td>x</td>
<td>Dis</td>
<td>x</td>
</tr>
<tr>
<td>STBus</td>
<td>1</td>
<td>x</td>
<td>Cen</td>
<td>n/a</td>
</tr>
<tr>
<td>Wishbone</td>
<td>AS</td>
<td>n/a</td>
<td>Cen</td>
<td>n/a</td>
</tr>
</tbody>
</table>

1. Application specific (AS), one-level (1) or two-level (2) arbitration scheme
2. Arbitration done during previous transfer (pipelined arbitration)
3. Centralized arbitration (Cen) or distributed arbitration (Dis)
4. Dynamic reconfiguration
5. Comparison of bus, crossbar, and 2-D mesh

The previous Chapters considered the interconnections in isolation from the actual system functionality. This is a good starting point but to get accurate information of their costs and performance in a realistic situation, they need to be tested as a part of a system. In this Chapter, several bus-based interconnections, crossbars, and a 2-D mesh are implemented with VHDL, synthesized, and simulated with different data distributions.

The comparisons are done for circuit-switched and packet-switched architectures, independently. The bus and the crossbar were chosen for the circuit-switched comparison because they are the most widely used circuit-switched architectures. For the packet-switched comparisons, a hierarchical bus and a 2-D mesh were used. The 2-D mesh was chosen because it is the topology of many presented NoC schemes. The hierarchical bus, on the other hand, is a fairly straight-forward extension of the circuit-switched bus architecture.

5.1. Circuit-switched architectures

The purpose of this comparison is to make comparable evaluation of bus and crossbar based architectures. A similar study was conducted in [Yan99] with energy and performance but at a much lower level disregarding the system effects. The results of the study showed that a crossbar consumes more energy per clock cycle and has a larger delay than a bus. On the other hand, the study showed that crossbars consume less energy per transfer and have a higher throughput when operating with full parallelism. In the comparison presented here, the implemented interconnection architectures included a bus with central arbiter, a bus utilizing a distributed arbitration scheme, and two crossbars with different arbitration methods.

As was tabulated in Table 3.3, the theoretical bus costs are the smallest of all the presented networks since a single bus has no switches and a minimal number of wires. On the other hand, a single bus only allows one parallel transaction at a time regardless of the number of connected nodes ($N$). The utilized one-sided crossbar has a relatively high switch cost ($N^2/2$). The wire cost is $N^2 - N/2$. Parallel transactions of the one-sided crossbar is relative to the number of nodes ($N$).

In this comparison, both architectures are implemented with similar basic components. A wrapper is used in connecting processing nodes to the interconnection network. The wrapper has a data input and output on both the processing node and the interconnection side. Two additional signals, namely
Request and Grant are needed for centralized arbitration. In the bus with distributed arbitration, these signals are not used. The wrappers contain two FIFO buffers: one from the processing node to the interconnection and one from the interconnection to the processing node. In addition, a state machine controlling the data transactions is required.

The implemented architectures do not utilize handshaking in data transfers. It is expected that the receiver FIFO buffers are designed so that they can buffer the incoming data. This assumption leads to small control areas and simple transfer protocols. The size, in bits, of the utilized FIFO buffers is four times the bus width in the comparison. The results presented here are based on synthesis results omitting the wire effects. This can be justified because the wire lengths in the studied interconnections are very close to each other.

The transactions are done with data packets of bit_width size containing one data valid bit, an address field, and a data field. In the studied implementations, the address field is always kept at minimum with each node having only one address. The rest of the packet payload is reserved for the data. All transactions are done in streams with no arbitration in between individual data transfers. The structure of the utilized packet is depicted in Figure 5.1.

![](image)

**Figure 5.1** The transfer packet structure of the circuit switched architectures.

The centralized arbitration is accomplished by utilizing a simple non-interruptive priority algorithm. Each node has a unique priority assigned to it. The distributed arbitration utilizes TDMA where each node is given a single timeslot. All the activities are synchronized to a global clock signal.

### 5.1.1 Single bus architecture with central arbiter

Figure 5.2 depicts a modular bus-based interconnection with four processing node connections below the four wrappers. The arbiter is shown on the left. A multiplexer, controlled by the arbiter, is used to direct the correct data to the bus. Equation 5.1 can be used in calculating the overall cell area taken by the bus components of a system. The multiplexer area is a function of the number of nodes (N) and the bus width. The arbiter area is a function of the number of nodes, and the area of the wrapper is a function of the bus width. The component areas and delays in this comparison are results from synthesis with different parameters.
The maximum clock frequency of the system \( F_{\text{max}} \) can be calculated with Equation 5.2 if the delays \( D \) of the used blocks are known. Because arbitration is the most complex process in the system and can take a lot of time, it is not desirable to let the arbitration limit the system clock cycle, and therefore, it is allowed to take several clock cycles. Multiplexer and wrapper, on the other hand, get only half a clock cycle to respond because both the rising and the falling edges of the clock are used. This kind of a clocking scheme might be difficult to implement. In this study the inherent low-level problems are omitted since the main interest is in architecture issues. The number of clock cycles reserved for arbitration is chosen so that the arbitration does not limit the clock cycle.

\[
F_{\text{max}} = \min \left( \frac{1}{2D_{\text{max}}}, \frac{1}{2D_{\text{wrapper}}}, \frac{\text{ClockCyclesForArbitration}}{D_{\text{arbiter}}} \right)
\]  

(5.2)

![Bus Architecture with Central Arbitration](image)

Figure 5.2 The bus architecture with central arbitration used in the comparison.

The arbitration algorithm used in the bus with central arbiter is given in Figure 5.3. The arbitration process is fairly simple as the priority is given by the location of the Request signal of the requesting node on the request signal lines. The highest priority node is connected to Request(0), the second highest to Request(1), and so on.
for (i=0; i<N-1; i++)
    if (Request(i) == '1') then
        Grant    <= (others => '0');
        Grant(i) <= '1';
        break;
    end if;
end for;

Figure 5.3 Arbitration pseudo-code of the bus with central arbitration.

5.1.2 Bus with distributed arbitration

Figure 5.4 depicts a different bus-based interconnection with four processing nodes. In this scheme, the arbiter is distributed as a part of the wrappers. Each wrapper is expected to know when to transmit data and, therefore, an OR-type bus is utilized. Equation 5.3 can be used in calculating the overall cell area taken by the bus components of a system. The OR gate and wrapper area are functions of the number of nodes (N) and the bus width.

\[
Area = Area_{Or}(N, \text{bit \_ width}) + N \times Area_{Wrapper}(N, \text{bit \_ width})
\]  

(5.3)

The maximum clock frequency of the system can be calculated with Equation 5.4 if the delays (D) of the OR and wrapper components are known. Since both the rising and the falling edge of the clock are used, the OR gate and the wrapper get only half a clock cycle to respond.

\[
F_{\text{max}} = \min \left( \frac{1}{2D_{Or}}, \frac{1}{2D_{Wrapper}} \right)
\]  

(5.4)
Figure 5.4 The distributed bus architecture of the comparison.

The arbitration algorithm used in the bus with distributed arbitration is given in Figure 5.5. It is based on TDMA where every wrapper counts the clock cycles on the bus and is aware of which cycles are reserved to it. In this case, each processing node is given one timeslot with equal length.

```plaintext
if (Counter < Max_Count) then
    Counter <= Counter + 1;
else
    Counter <= 0;
end if;
if (Counter >= Start_Slot AND Counter <= End_Slot) then
    Own_Slot <= '1';
else
    Own_Slot <= '0';
end if;
```

Figure 5.5 Arbitration pseudo-code of the bus with distributed arbitration.

5.1.3 One-sided crossbar

Figure 5.6 presents the circuit-switched one-sided crossbar network used in the comparison. The connection is formed through switches that are marked with ‘X’, forming horizontal buses namely ‘bus 1’ and ‘bus 2’. The switches were implemented with logic cells as presented in [Gho91]. The component on the
upper left is the arbiter. The Request and Grant signals are the same as in the 
bus with central arbitration. In addition, with the address-input of the arbiter, 
the requesting wrappers tell to whom they wish to send data. The highest 
priority processing node gets the lowest unused horizontal bus provided that the 
receiver of the transaction is not involved in another transfer. The outputs of 
the arbiter are used to control the switches as well as telling to whom the 
ownership was granted. After the route is ready, the node can send data until 
the Grant is deactivated or it runs out of data to send.

Because each transaction has to have a sender and a receiver and the nodes 
cannot receive and transmit at the same time, maximally \( \lceil N/2 \rceil \) horizontal 
buses are needed, where \( N \) is the number of processing nodes. This is 
illustrated in Figure 5.6, where \( N = 4 \) and the number of horizontal buses is 2. 
A one-sided crossbar was chosen because the FIFO buffer sizes can be 
minimized when the arbiter guarantees that the receiver of data is not reserved 
for other transactions. Equation 5.5 can be used to calculate the cell area of the 
crossbar network. \( N \) represents the number of processing nodes and \( N^2/2 \) the 
number of required switches.

\[
Area = \frac{N^2}{2} \cdot Area_{Switch}(bit \_\ width) + Area_{Arbiter}(N) + N \cdot \frac{Area_{Wrapper}(bit \_\ width)}{2} \quad (5.5)
\]

The maximum clock frequency of the system based on a crossbar can be 
calculated with Equation 5.6. The limiting factor can be either the longest route 
through the switches or the wrapper delay. The amount of switches on the 
longest path \((N + 2(N/2 - 1)) = (2N - 2)\) grows linearly as a function of \( N \) and, 
therefore, the wrapper was designed to respond in half a clock cycle but the 
switch delay was allowed to take a full clock cycle. In addition, the crossbar 
arbitration is allowed to take several clock cycles and its latency is chosen so 
that it does not limit the system clock speed.

\[
F_{\text{max}} = \min \left( \frac{1}{(2N - 2)D_{\text{Switch}}}, \frac{1}{2D_{\text{Wrapper}}}, \frac{\text{ClockCyclesForArbitration}}{D_{\text{Arbiter}}} \right) \quad (5.6)
\]
Figure 5.6 The crossbar architecture used in the comparison.

The arbitration of the crossbar was implemented in two ways. The first version arbitrates every parallel bus on every clock cycle, whereas the second version arbitrates only one bus at each clock cycle utilizing pipelining and resulting in shorter arbitration period. Otherwise the schemes are alike. The ownership of a free bus is granted if a wrapper requests a bus, the wrapper is not already an owner of a bus, the receiver wrapper is not involved in another transaction, and there are free parallel buses available. In addition to this process, the arbiter also has to keep a list of free buses, transmitters, and receivers, as well as putting the switches into correct positions. These processes are not shown in the pseudo-codes of Figure 5.7.
Version 1

for (i=0; i<N-1; i++)
if (Request(i) == '1' AND
    Node_Reserved_Bus(i) = 0 AND
    Receiver_Busy(Addr) = '0' AND
    Free_Buses > 0) then
    Grant(i)             <= '1';
    Node_Reserved_Bus(i) <= Free_Buses;
    Receiver_Busy(Addr)  <= '1';
    Free_Buses - -;
end if;
end for;

Version 2

if (Request(i) == '1' AND
    Node_Reserved_Bus(i) = 0 AND
    Receiver_Busy(Addr) = '0' AND
    Free_Buses > 0) then
    Grant(i)             <= '1';
    Node_Reserved_Bus(i) <= Free_Buses;
    Receiver_Busy(Addr)  <= '1';
    Free_Buses - -;
end if;
if (i == N) then
    i <= 0;
else
    i++;
end if;

Figure 5.7 Pseudo-codes of the two different crossbar arbitrators.

5.1.4 Synthesis results of the circuit-switched architectures

In the following, different bus and crossbar components are synthesized to a 0.18 µm CMOS technology and then the presented equations are used to calculate the overall area and performance metrics of various architectures. The synthesis results of a set of 64-bit interconnections are compared. Similar results were also obtained for 16, 32, 128, and 256 bit systems. In the following figures, the bus implementation with central arbiter is indicated with an ‘o’ and the distributed version with ‘+’. The crossbars are marked with an ‘x’ for version 1 and an ‘*’ for version 2.
Figure 5.8 depicts the utilized area of the implemented 64-bit interconnections using their respective maximum frequencies that are depicted in Figure 5.9. As can be observed, the area of both the crossbars is considerably larger than that of both the buses. This is caused by the arbiter complexity. Particularly the arbiter of version 1 of the crossbar consumes a very large area.

Figure 5.8 Areas of the implemented 64-bit buses and crossbars.

The algorithms used in the arbitration of the crossbar are versatile and extensive but quite costly to implement. Nevertheless, they were implemented in this manner to get a different area-performance trade-off from the basic bus interconnection. Many crossbar arbiters optimized for different purposes are presented in [Tam93].

Because arbitration can take many clock cycles, the obtained system clock frequency is almost the same with each case. Only the frequency of the distributed bus is slightly higher. This is depicted in Figure 5.9. The clock cycles reserved for arbitration are presented in Figure 5.10. The arbitration time
of the crossbar is relatively long because of the time it takes to put the switches into correct positions and keep track of the ongoing transactions.

Figure 5.9 Maximum operating frequencies of the 64-bit buses and crossbars.

Figure 5.10 Clock cycles reserved for arbitrating the 64-bit buses and crossbars.
5.1.5 Circuit-switched architecture simulation environment

The simulations of the comparison were done in a VHDL test bench with a uniform traffic pattern, which means that all processing nodes transmit the same amount of data to all other nodes in a circular manner. The distribution for a four node case is depicted in Figure 5.11. In addition, data transfers were independent from each other. Each wrapper operated in two modes, writing when it has access to bus or reading the bus otherwise.

![Figure 5.11 The transfer distribution in a four node case.](image)

5.1.6 Simulation results of the circuit-switched architectures

In the following three figures, the obtained throughputs of the bus and crossbar networks with different number of processing nodes are compared. Because it takes a long time to set up a crossbar, a bus is better suited for systems with short transactions. When transactions become longer, crossbar has a higher throughput. This is illustrated in the figures with average transaction lengths of 8 transfers in Figure 5.12, 64 transfers in Figure 5.13, and 512 transfers in Figure 5.14. For each architecture, the maximum frequency implied by the synthesis is utilized. The higher throughput of a crossbar is mostly constituted by the fact that there are more parallel interconnection paths available in a crossbar than in a bus.
Figure 5.12 Throughputs with on average 8 transfers per transaction for 64-bit buses and crossbars.

Figure 5.13 Throughputs with on average 64 transfers per transaction for 64-bit buses and crossbars.
Figure 5.14 Throughputs with on average 512 transfers per transaction for 64-bit buses and crossbars.

In the bus architectures, the total throughput remains constant regardless of the number of processing nodes and transfer size. The distributed bus has a little higher throughput because the arbitration times are negligible. In the crossbars, the total throughput depends quite heavily on the size of the transfers, because the arbitration latency becomes less significant.

Based on this comparison, buses are simple and efficient for small data transfer rates with small number of processing nodes. Because of the simplicity, the area of bus architectures can be kept small and the operating frequency high assuming that the capacitive loading does not limit the operating frequency. The distributed version of the bus resulted in the smallest area of the comparison.

Crossbar, on the other hand, is a somewhat more complex system architecture to implement. However, if the data transactions are long and if they can be done in parallel, the hardware complexity can be justified. The area of the bus network of this study is considerably smaller than the area of the studied crossbar. On the other hand, the throughput of the crossbar is over two times higher with large data transfers. The arbitration scheme used in this study produces large arbiter units for crossbars but this is counterbalanced by the high throughputs.
A performance function presented in Equation 5.7 is used to combine the effect of throughput and area to make balanced comparisons between the architectures. When this function is used, large function values are better.

\[ \text{Performance} = \frac{\text{Throughput}}{\text{Area}} \] (5.7)

The following figures show the performance function values for the cases with on average 8 transfers (Figure 5.15), 64 transfers (Figure 5.16), and 512 transfers (Figure 5.17). They show that the bus has better values with small transfer sizes and node counts. When these values are increased, the crossbar becomes a good choice. The figures also show a dramatic decline in the performance when the number of agents increases. The presented architectures are clearly not meant to be used in large-scale systems.

Figure 5.15 The performance function with on average 8 transfers.
Figure 5.16 The performance function with on average 64 transfers.

Figure 5.17 The performance function with on average 512 transfers.
5.2. Packet-switched architectures

Packet-switched architectures and buses have been compared in many publications. The problem with many of these comparisons has been that they compare only the theoretical maximum transfer capabilities of the interconnections and do not take the intended application into account [Zef02]. In addition, they usually only examine single buses or multiple parallel buses omitting the more versatile hierarchical structures [Zha99][Zef02][Thi03].

In this comparison, three packet-switched networks are analyzed. The compared networks include a bus that can be used as a single bus or a hierarchical bus and a 2-D mesh network. Also in this comparison, the networks are implemented by utilizing a set of basic components. A wrapper is used by the bus processing nodes in connecting to the interconnection network. Two FIFO buffers, one from the processing node to the interconnection network and one to the other direction, are located in each wrapper. A state machine controls the wrapper operations. The mesh structure is somewhat different and will be described later.

In Table 3.3, the wire costs and the number of parallel connections of a hierarchical bus are tabulated to be equal to the number of parallel edges ($e$). The switch cost, on the other hand is $e−1$. The 2-D mesh has a switch cost comparable to the number of nodes ($N$) and a wire cost of $3N−2N^{1/2}$. The number of parallel transactions available to a 2-D mesh is $N$ for meshes where the switches have 5 connections according to Table 3.2.

All the implementations transfer data in packets with equal length containing an address, a source ID that is different for each wrapper, the amount of data that is to be transferred, and the correct amount of data. Here the bit width refers to the size of an individual field in the packet. In the presented simulations, the Data amount is always eight and the bit width is 32. The structure of the packet is depicted in Figure 5.18. Due to this packet structure and the fact that the networks utilize store-and-forward switching, all the FIFO buffers in the system are 11*32-bits long. Because the FIFO size is fixed, it is not given as a parameter in the forthcoming equations.

<table>
<thead>
<tr>
<th>Address</th>
<th>Source ID</th>
<th>Data amount</th>
<th>Data amount * Data</th>
</tr>
</thead>
</table>

bit_width

Figure 5.18 The utilized packet structure.
The results presented here are based on synthesis results omitting the wire effects. This assumption makes the single bus results too optimistic. This is acceptable since the results of the single bus case are much worse compared to the other two networks with the presented workloads and it is used here only as a reference. The wire effects of the hierarchical bus are controlled by using only four processing nodes in each segment which is under the projected limit of nodes that can be connected by a bus in Figure 4.1. All the wrappers utilize the same agent interface which is directly the interface of the utilized FIFO buffers.

### 5.2.1 Packet-switched bus architecture

The bus implementation requires a wrapper and a bridge component. The implemented wrapper is a fairly simple device with two FIFO buffers and a receive (Rx) and transmit (Tx) control unit. The bridge components were implemented with connecting two wrappers together. The actual bus is implemented with OR-gates.

The structures of the implemented bus components are presented in Figure 5.19 with a system of two bus segments each having two processing nodes. The needed bus signals are for clock (Clk), system reset (Rst_n), data and address transmissions (Data), reserving the bus (Lock), acknowledging a transfer (Acknowledge), and distinguishing between data and address (Data_valid) since the bus multiplexes data and address to the same signal lines. The biggest difference to the bus implemented in the previous Chapter is the utilization of data handshaking in the form of the Acknowledge signal. This leads to bigger control areas but more reliable operation and easier design of large systems.
Figure 5.19 The implementation of a packet-switched bus with two segments.

The hierarchical bus architectures utilized in this Section are formed out of bus segments of four processing nodes. Note that in Figure 5.19, there are only two nodes. Between each segment there is a bridge. The structure is, therefore, a chain of four-node bus segments. A tree formed out of the bus segments might be better for some types of workloads.

The arbitration is based on distributed round-robin scheme where the ownership is passed to the next wrapper after each transferred packet. The operation of the arbiter is presented with pseudo-code in Figure 5.20. The Lock signal is de-activated at the end of a packet transfer or when a wrapper does not want to utilize the given turn.

\[ \text{OR} = \text{OR gate, } A = \text{Arbiter, } W = \text{Wrapper, } B = \text{Bridge} \]
if (Lock == '0') then
  if (Counter == Round_Robin_Max - 1) then
    Counter <= 0;
  else
    Counter <= Counter + 1;
  end if;
else
  Counter <= Counter;
end if;
if (Counter == Round_Robin_Turn) then
  Own_turn <= '1';
else
  Own_Turn <= '0';
end if;

Figure 5.20 Pseudo-code of the packet-switched bus arbitration.

The bus utilizes a simple handshaking scheme. This is depicted in the following two figures specifying the needed control state machines for transmitting (Figure 5.21) and receiving (Figure 5.22) data. The transmit process starts from the IDLE state when a wrapper has data to send and the round-robin turn. The Data_To_Send signal comes from the transmit FIFO and the Own_Turn signal from the internal arbiter of the wrapper. The wrapper first writes the address to the bus and waits for acknowledge from the receiver. If the receiver is ready, it asserts the Acknowledge signal. After this, the transmitter can start to transfer data. If the receiver is not ready and does not give the Acknowledge signal, the wrapper gives up the reserved round-robin turn and the state machine enters the WAIT_TURN state where it stays until the next activation of the Own_Turn signal.
The Own_Address_On_Bus signal tells to the receive state machine that an address on the bus belongs to it after which it leaves the initial IDLE state. It then checks whether there is enough space in the receive FIFO for one packet in which case it acknowledges the transfer. After this, the receiver simply reads all the incoming data to the FIFO buffers. If there is not enough space in the FIFO, the transfer is not acknowledged. In this case, the transmitter will try again later.

Figure 5.21 The transmit state machine of the packet-switched bus.
The area for the single bus can be calculated with Equation 5.8. The needed components are the OR-gate and the wrapper. The difference to the previous distributed bus system area (Equation 5.3) is that the area of the wrapper is a function of both the number of components in a bus because of the round-robin scheme and the bus width. The area of the OR-gate is a function of the number of components in the bus and the bus width. Also in this Section, the areas are acquired from component synthesis results.

\[ \text{Area} = \text{Area}_{OR}(N, \text{bit width}) + N \times \text{Area}_{\text{wrapper}}(N, \text{bit width}) \]  

(5.8)

Equation 5.9 is used for calculating the area of a hierarchical bus. The areas of the OR-gates and the wrappers are now functions of the number of nodes in each bus segment (\( N_{\text{seg}} \)) and the bus width. To get the final area result, these both need to be multiplied by the number of segments in the system (\( \text{Seg} \)). In addition, bridges are needed between every bus segment.

\[ \text{Area} = \text{Seg} \times \text{Area}_{OR}(N_{\text{seg}}, \text{bit width}) \\
+ \text{Seg} \times N_{\text{seg}} \times \text{Area}_{\text{wrapper}}(N_{\text{seg}}, \text{bit width}) \\
+ (\text{Seg} - 1) \times \text{Area}_{\text{bridge}}(N_{\text{seg}}, \text{bit width}) \]  

(5.9)

5.2.2 Packet-switched 2-D mesh

The 2-D mesh network is built by utilizing a single component that is called a router. The router units contain two FIFO buffers for the node interface and one buffer for each North, South, East, and West output. The total number of FIFO buffers is therefore six. In addition, the router has a control unit that takes care
of the actual routing process by directing the incoming data to the correct output FIFO buffer. The structure of the network implementation is depicted in Figure 5.23.

![Diagram of a 2-D mesh network implementation](image)

**Figure 5.23** The implementation of a 2-D mesh.

The node interface is formed by the FIFO interface signals. Similar signals are also used to connect the routers to each other. The routers only write their data forward if the correct output FIFO buffer of the target router is empty, and therefore capable of storing the whole packet that will be send. This makes the implementation easier because there is no need for re-transmissions caused by FIFO buffers becoming full.

The actual routing algorithm is very simple. Each processing node has an address of the form (x, y) where x is the column and y is the row of the node. The sent packets are first forwarded to the correct row and then to the correct
The direction of the forwarding can quite easily be computed from the destination address \((x_d, y_d)\) and the current location \((x_c, y_c)\) of the packet as is depicted in Figure 5.24.

The routers check their inputs in a round-robin manner, one in each clock cycle. If there is data coming in and the targeted output buffer is empty, the transaction is started and the data is forwarded to the correct output FIFO. Otherwise the transfer is blocked.

```plaintext
Y_direction <= y_d-y_c;
X_direction <= x_d-x_c;
if (Y_direction < 0)
    Direction <= South;
elsif (Y_direction > 0)
    Direction <= North;
elsif (X_direction < 0)
    Direction <= East;
elsif (X_direction > 0)
    Direction <= West;
else
    Direction <= Node;
end if;
```

Figure 5.24 Pseudo-code of the 2-D mesh routing.

Because of the simple structure of the 2-D mesh, the area taken by the network is easily computed with Equation 5.10. The area is constituted by the area of the router units which change based on the number of processing nodes and the interconnect bit widths.

\[
\text{Area} = N \times \text{Area}_{\text{router}}(N, \text{bit width}) \tag{5.10}
\]

### 5.2.3 Synthesis results of the packet-switched architectures

In the following, different single bus, hierarchical bus, and 2-D mesh components are synthesized to a 0.18\(\mu\)m CMOS technology and then the presented equations are used to calculate the overall area for architectures with 4, 16, 36, and 64 processing nodes. The areas of the networks are depicted in Figure 5.25 as k gates.
As was anticipated, the area of the single bus is the smallest. The area of the hierarchical bus is only a little larger, whereas the mesh is the most area consuming interconnection network. These results will be discussed in more detail at the end of this Section.

5.2.4 Packet-switched architecture simulation environment

In this comparison, the presented interconnection networks are compared using synthetic test cases that are generated to represent characteristic application properties, such as their sequential or parallel nature. The test cases are run in a simulation environment based on a transaction generator [Kan03]. It is capable of running application descriptions that show computation processes and their communications. The utilized transaction generator is based on the Kahn process network model [Kah74].

Figure 5.26 depicts the first sequential test case (Test case 1) for eight processing nodes. The circles depict processes that can have any arbitrary processing times as clock cycles (t_P). The arrows represent data transmissions of length t_t, assuming that each separate data transfer takes one clock cycle. The processing cannot start until at least one of the transmissions leading to that process has ended. The processes that have a processing time and a
transmission length of 1 are executed only once to start the simulation of the network which is shown by their lack of data dependencies.

The dashed lines describe the mapping of the process graph into eight processing nodes. The mapping is done so that the number of transmissions is minimized. In Figure 5.26 this means that one processing node gets two processes including the starting process and the rest each get one.

![Figure 5.26 Sequential test case (Test case 1).](image)

The second test case (Test case 2) is partly sequential and partly parallel. It is quite close to the sequential Test case 1 but there are \(N/2\) starting processes connected to every other process in the loop to increase parallelism. In Figure 5.27, this process network is presented for eight processing nodes.

![Figure 5.27 Partly sequential and partly parallel test case (Test case 2).](image)

The third test case (Test case 3) represents a situation where the transfers are sequential in hierarchical clusters of four processing nodes. It is depicted in Figure 5.28 for a case of eight nodes.
The fourth test case (Test case 4) has processes in a cluster of four transmitting data at the same time. The difference to Test case 3 is that Test case 4 has four simultaneous data transfers in each group. This test application is depicted in Figure 5.29 for eight processing nodes.

**Figure 5.29 Parallel test case (Test case 4).**

### 5.2.5 Simulation results of the packet-switched architectures

In the simulation presented here, the processing time ($t_p$) is 16 and the transmission length ($t_t$) is 1024. This constitutes an extremely communication intensive transfer patterns but they are good in finding out the weak and good points of interconnection architectures. The first four results show on the y-axis the number of clock cycles that it takes to run all the presented test cases. They are depicted in Figure 5.30, Figure 5.31, Figure 5.32, and Figure 5.33.
Figure 5.30 Clock cycles to run Test case 1.

Figure 5.31 Clock cycles to run Test case 2.
Figure 5.32 Clock cycles to run Test case 3.

Figure 5.33 Clock cycles to run Test case 4.
The results show that with the utilized transfer patterns the single bus is clearly not applicable when the amount of processing nodes exceeds 16. The 2-D mesh performs best in all the four test cases, but the hierarchical bus is a close second. It should also be noted that whereas in the single bus case the run-times quickly go beyond an acceptable range, the run-times of the hierarchical bus and the mesh grow only modestly as the number of processing nodes increases.

The biggest difference between the hierarchical bus and the mesh is in the Test case 4 where the run-time of the hierarchical bus is 2.4 times the run-time of the mesh. In addition, in the second test case the hierarchical bus is 1.8 times slower. In Test case 1 and Test case 2, the difference is smaller. The run-time of the hierarchical bus is 1.1-1.5 times longer than the run-time of the mesh in them.

The area of the mesh is also considerably larger than the area of the hierarchical bus. Depending on the number of processing nodes, the mesh is 2.3-2.5 times larger. This re-affirms the previous assertion that the choice between the mesh and the hierarchical bus is a trade-off between the cost and the performance.

To represent the overall performance, a fifth simulation was conducted. In this case, all the four test cases are combined in a single simulation. A combined run-time is achieved by adding the run-times of the separate test cases in this simulation. The applied performance function is given in Equation 5.11. It is formed by simply taking an inverse of the combined run-time of the test cases multiplied with the required area. This resembles the performance function utilized in the circuit-switched case.

\[
\frac{1}{\text{Run-Time} \times \text{Area}}
\]

(5.11)

In Figure 5.34, the performance functions of the two interconnection networks are depicted. It should be noted that since the unit of the performance here is 1/(clock cycles * kgates), a bigger value is better. Using this function, the hierarchical bus seems to be slightly more suitable to the presented test cases than the 2-D mesh. The performance of the mesh is 94 % and 84 % of the hierarchical bus in the 16 and 36 node cases, respectively. This function emphasizes equally the area and the performance. Different results can be achieved by weighing the two product terms in Equation 5.11 with different coefficients.
Figure 5.34 The performance function values of the packet-switched networks.
6. Design of HIBI

Based on the comparisons of the previous Chapter, two clearly identifiable points can be made. The first comparison showed that the distribution of the bus arbitration can be used to raise the performance of bus architectures. The second comparison proved that the hierarchical bus systems can be utilized in large-scale digital systems just as well or even better than a 2-D mesh. These two observations have been used as a starting point for the design of the previously mentioned Heterogeneous IP block interconnection (HIBI) scheme that is now described in detail.

The main design principle in HIBI has been to make the basic data transfer extremely simple, such that it is both efficient and general-purpose. HIBI presents only the necessary features for data transfers, and all more advanced functions are built using these basic transfers. Therefore, the scheme has the same design principles as was used in the introduction of RISC processors [Pat80].

The HIBI scheme structure is illustrated in Figure 6.1 that depicts the VSI interconnect layer model [VSI99] and a system utilizing HIBI-based interconnection. Physically HIBI extends to the bus transfer layer, where a HIBI wrapper acts as an interface to the HIBI bus. However, the whole scheme extends to the higher layers as well. The HIBI interface is formed by the FIFO interfaces of the wrapper. This is not a standard interface but can be quite easily fitted to the VCI or OCP specifications.

Both the target and the initiator virtual component interfaces can be implemented with a single wrapper. The physical layer deals with the wiring, drivers, and timing that are HIBI technology specific. The bus transfer layer encompasses the protocols needed in transferring data on the bus. Above this layer is the bus transaction layer that defines point-to-point transfers between VCs. This layer is also called the VCI and all layers from this layer upward are the same in the VSI alliance specification and in HIBI.

The system transaction layer deals with transactions between two objects and the application layer with many interacting components. The intelligence in HIBI-based systems is mostly left to the application and the system transaction layers. That is how the bus itself has been kept simple but yet very effective and power efficient.
6.1. HIBI structure

HIBI-based systems are constructed with the aid of pre-designed, re-useable wrappers. Unlike in some other on-chip buses [Cor99][Bai00][Ack00][Sil01],
the connected IP blocks do not need to be aware of the interconnection details. Instead, they can leave the interconnection network related issues to the wrapper. The separation of computation and communication with a generic interface wrapper is a key issue in the HIBI scheme.

In Figure 6.2, several typical IP blocks of a SoC, such as microprocessor core, DSP core, memory block, and off-chip interconnection (I/O), are depicted. Also other application specific hardware blocks can be used. The diagonally lined area in the bottom right represents parts of the system that can be added to later designs. The actual HIBI interconnection network is seen in the middle. The white parts are implemented in software and the grey ones in hardware. Some pre-designed IP blocks need special adapters before they can be connected to the utilized wrapper. Usually the required adapter is physically much smaller and simpler than the wrapper.

![Figure 6.2 An example HIBI-based system.](image)

As can be seen from Figure 6.2, there is no physical central arbiter present in a HIBI-based system. The arbitration has been distributed to each connected HIBI wrapper, which are aware of the communication settings of other
wrappers. In addition, all signal lines are fed to all wrappers. In this way, addition of IP blocks or otherwise modifying the system is straight-forward.

The awareness of communication requirements for every wrapper is implemented in two ways. First, all wrappers have a global address space which ensures that a single address is enough to resolve to which wrapper and for which purpose in the connected IP block the data is sent. Second, all wrappers know how much data they can send to other wrappers in one transaction. This, in turn, makes handshaking unnecessary for each individual transfer in a data transaction, which saves signal lines and reduces latencies.

HIBI does not utilize the two-phase or four-phase handshaking schemes that were presented in Figure 4.3. Instead, the start and end of transactions are signalled by a single bus reservation signal. This fits well into dataflow type applications in which the data transactions and the sizes of the required buffers can be determined in advance with statically analyzing the application. In control dominated applications, this is not the case because their seemingly random reactive behaviour can cause unexpected congestion of wrapper buffers.

Congestion of a wrapper in a particular transaction can lead to overflow in the buffers which in turn can cause the loss of data. If the application cannot tolerate loosing data, handshaking can be implemented with normal bus transfers. In other words, message passing is utilized to make sure that the receiver of data is ready and has enough empty buffer space. This should be avoided when possible because it increases latency which otherwise is very small in HIBI transactions. The utilization of message passing to implement handshaking is an application-level decision that is left to the designer of HIBI-based systems.

6.2. Transfers utilized in HIBI

HIBI utilizes fully synchronous transfers. For signaling, HIBI has six connections: data (Data), address (Addr), command (Comm), clock (Clk), asynchronous active-low reset (RstX), and lock (Lock). The data bus is used for transmitting data and the address bus to direct the data into the right address. The command bus identifies the different HIBI bus operations. The active-low lock signal is activated before a HIBI wrapper takes the bus ownership and released just before the end of the transaction. By following the lock signal, other wrappers can see when the bus is reserved. The clock and reset lines are used to carry the global clock and reset signals. The signals of the HIBI bus are presented in Table 6.1 with their abbreviations, bit widths, and purposes.
Table 6.1 HIBI bus signals.

<table>
<thead>
<tr>
<th>Signals</th>
<th>Abbrev.</th>
<th>Bit width</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>Clk</td>
<td>1</td>
<td>Global clock signal</td>
</tr>
<tr>
<td>Reset</td>
<td>RstX</td>
<td>1</td>
<td>Active-low asynchronous reset</td>
</tr>
<tr>
<td>Lock</td>
<td>Lock</td>
<td>1</td>
<td>Reserves the bus</td>
</tr>
<tr>
<td>Address</td>
<td>Addr</td>
<td>8 +</td>
<td>Transmitting address of transaction</td>
</tr>
<tr>
<td>Data</td>
<td>Data</td>
<td>8 +</td>
<td>Transmitting data of transaction</td>
</tr>
<tr>
<td>Command</td>
<td>Comm</td>
<td>3</td>
<td>Transmitting bus commands</td>
</tr>
</tbody>
</table>

HIBI has four different operation groups for transmitting data and modifying the bus configuration. These are write, read, burst write, and configuration. Each group has unique command codes that are placed on the command bus during these operations. The HIBI commands, their decoding on the command bus, and purpose are tabulated in Table 6.2.

Table 6.2 The HIBI bus commands.

<table>
<thead>
<tr>
<th>Commands</th>
<th>Decoding</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>000</td>
<td>Writing data</td>
</tr>
<tr>
<td>ReadRQ</td>
<td>001</td>
<td>Requesting data (split transaction)</td>
</tr>
<tr>
<td>BWrite</td>
<td>010</td>
<td>Initializing burst</td>
</tr>
<tr>
<td>Burst</td>
<td>100</td>
<td>Writing data in burst mode</td>
</tr>
<tr>
<td>CWrite</td>
<td>011</td>
<td>Configuration write</td>
</tr>
</tbody>
</table>

HIBI has a normal write command for writing single data elements to arbitrary addresses, and also burst transfer operations for writing data in a streaming burst mode. When Write is used, all the data elements are accompanied by an address. This makes it possible to direct transfers inside a transaction to different addresses or even agents. The HIBI write command can be used to transfer data in a burst mode since each clock cycle transfers data.

The actual burst transfer used in HIBI is somewhat different from the usual burst transfers of on-chip buses. Normally, burst transfer makes it possible for a single wrapper to send multiple data elements in succession to a single address. In addition to this, HIBI almost doubles the transaction speed by using both data and address buses for transmitting data. This type of a burst transfer is called a streaming burst transfer. A noteworthy issue is that the address and the data lines do not need to be the same size. When the streaming burst is used this means that the data transmitted on the address lines needs to be expanded or truncated to the data line width.

In the start of a streaming burst transfer, the BWrite command is used similarly as the Write command and the address and the first bursting data transfer occurs. After this initial phase, the Burst command is utilized and both address and data buses are used for transmitting data. This means that bursts are send to
a single address. If needed, the recipient, for example memory controller, can increase the address to which the elements will be written with a valid increment. The system designer can freely choose the length of the streaming burst during the system design time. It is given as a generic parameter to the system synthesis process after which it becomes a fixed value for the whole system.

HIBI does not have a conventional read operation but a read request instead. During this operation, the ReadRQ command is used. The read command is a split transaction. This means that the read request is first written on the bus after which it is the responsibility of the recipient to write back the requested data with normal write or burst write operations.

The HIBI interconnection configuration is stored in the configuration memories of the connected wrappers. The contents of the configuration memory will be described later in this Chapter. The initial values of the configuration memory can be set during synthesis but most of the values can also be modified run-time with configuration write operations. The reconfiguration is quite a complex process during which the CWrite command is utilized. The process will be explained with an example later in this Section.

In the following, the basic operations of the bus are presented with timing diagrams, which are simulation screen captures taken from separate simulations of a 16-bit HIBI interconnection containing three wrappers. In Figure 6.3, the write and burst write commands of the HIBI bus are depicted. Both write operations are initiated by wrapper 1. First, a single write operation (data 0x1) is send to wrapper 2 (address 0x18). After this, a streaming burst write operation, directed to wrapper 3 (address 0x20), is started. During this operation, the first two data elements are on the data bus and after this on the address and data buses consecutively. The length of the burst operation is six transfers in this simulation. The RstX signal is continuously in de-active state (high) and all the operations are synchronized by the Clk signal. The start of the transaction is signalled by the assertion (active low) of the Lock signal.

![Figure 6.3 Write and burst write operations of HIBI.](image)

In Figure 6.4, a read request operation is presented. First wrapper 1 sends a read request to wrapper 2. In the data bus, the address of the response is given.
After this, wrapper 2 uses write operations to respond to this request. In this case, three data elements (0xfffc, 0xfffe, 0xffff) are written to the given address (0x2). The amount of data that is sent and the command that is used (write or burst) can be freely decided based on the application.

![Figure 6.4 A read operation of HIBI.](image)

The use of the Lock signal can also be observed in Figure 6.4. During one clock cycle transactions, like during the read request command, it is not activated. With longer transactions, the Lock is activated (driven low) half a clock cycle before the first data element is put on the bus. It is de-activated in the middle of the transfer that precedes the last one. The de-activation means that the wrapper no longer drives the Lock signal low, and thus it is driven high by the pull-up circuitry. The Lock is operated on a different clock edge than the rest of the signals which might be difficult to implement as was discussed in Chapter 5.1.

In Figure 6.5 a), the start of a reconfiguration process is depicted. As an example, the reconfiguration of wrapper 2 arbitration parameters is presented. The meaning of these parameters is explained in Chapter 6.3. Initially the frame-length was 0x78, timeslot beginning for wrapper 2 0x32, timeslot end for wrapper 2 0x50, and timeslot beginning for wrapper 1 0xa. All these values are given as clock cycles and can be seen on the left in Figure 6.5 a). During the reconfiguration process they are changed so that the frame-length becomes 0xff, the timeslot beginning for wrapper 2 0x82, timeslot end for wrapper 2 0xb4, and the timeslot beginning for wrapper 1 0x14. This is depicted in Figure 6.5 b).
Figure 6.5 The a) start and the b) proceeding of the reconfiguration process.

The start of the reconfiguration consists of three phases as is depicted in Figure 6.5 a). Other wrappers but the initiator of the reconfiguration are disconnected from the bus by writing (0x0) to a special address (0x2b for wrapper 2 and 0x3b for wrapper 3). Next the reconfiguration field of the initiator is activated by writing (0x1) to address 0x1d. In the last phase, a broadcast command (0x0 on the address and data buses) is used to signal the start of the reconfiguration process. After this, the configuration memories of all wrappers can be modified.

At the end of the reconfiguration process, a three phase closing procedure is needed. In the first phase, all the agents are connected to the bus. Next, the reconfiguration field of the initiator is de-activated, and in the third phase, the end of reconfiguration is broadcasted to all agents. After this, the system returns to the normal operation mode.
6.3. Arbitration of HIBI

Because of the assumption of large repetitive, roughly predictable data transfers, the latencies in the HIBI transactions can be made very small. A video encoder has been used as an example of this kind of a system [Kan00]. In video coding, large regular data transactions occur in conjunction with single, arbitrarily timed transfers. To facilitate these demands, HIBI uses a TDMA scheme similar to [Son00a] in which pre-allocated timeslots and competition phases alternate.

The regular large data transfers can utilize pre-allocated timeslots and for the rest of the time the bus ownership is resolved with competition. An example of this kind of time partitioning is depicted in Figure 6.6. With this arbitration scheme, bus bandwidth can be guaranteed for high-priority wrappers having regular transaction needs. It must be noted that in the implementation utilized in this Thesis, the streaming burst transfers are only possible during timeslot-based accesses.

![Figure 6.6 The HIBI time frame structure (W = wrapper, t =time).](image)

Figure 6.6 depicts one whole time frame and the next starting one. The time frame is the basic timing unit which repeats as seen in Figure 6.6. Its length is defined as clock cycles. The first timeslot is reserved to wrapper 1. The length of all the timeslots can be freely chosen based on the targeted application. After the first timeslot, the bus ownership is resolved with competition. Competition is also started when a wrapper does not want to use its reserved timeslot. The timeslots are defined by two numbers representing the starting and ending clock cycles of that timeslot inside the time frame.

Each wrapper has a distinct priority value that tells the number of clock cycles that the bus has to be free during competition phases before it can access the bus. The highest priority (one) can access the bus directly; the second highest (two) has to wait for one clock cycle after the bus has been released and so on. With this scheme, two wrappers cannot access the bus at the same time. After the competition phase, the bus is given to wrappers 3, 2, and 3 again, in this order. Timeslots can be assigned prior to chip manufacturing, but can also be modified at run-time with the aid of the reconfiguration process.

The time for which a certain wrapper can take the bus ownership during competition phases is restricted by a value stored in the configuration memory. In the current implementation, the ownership is transferred to the next lower
priority when this time limit is reached or the wrapper has no more data to send. A started transfer cannot be pre-empted by other wrappers. After the turn of the lowest priority wrapper, the ownership is passed back to the highest priority one. This scheme, utilized during competition phases effectually implements the round-robin scheme. However, round-robin arbitration is always started from the highest priority after timeslots to favour high-priority transfers.

The presented arbitration scheme minimizes the initial and subsequent data latencies. Because there is no handshaking and the master does not wait for the slave to be ready, each clock cycle should transmit data after the arbitration. The current implementation requires one clock cycle turn-around latency after a timeslot. The very low latencies of HIBI transactions make it a good fit to systems transferring large amounts of data. The predictability of communication has also been exploited in the static communication scheduling of [Jia00] and [Wai97] and the compile-time bandwidth assignment of [Lei98].

Timeslot-based approaches can become problematic if the transfer timings start to drift away from their original position for some reason. This can happen, for example, due to a small un-determinism in the computation times of the processing nodes. One approach to this problem is to use bus monitoring [Kan02] for synchronizing and frequency adjusting of the bus. In this scheme, a bus monitor unit, either separate or a part of one system control processor, monitors the bus transactions and takes action if certain criteria are met. These include the drifting of timeslots in which case synchronization process is started and the underutilization of the bus in which case the bus frequency can be lowered.

Figure 6.7 presents the flow for acquiring the TDMA arbitration parameters; the starting times, owners, and lengths of timeslots. First the application is analyzed and, based on the results, a system with preliminary TDMA parameters is synthesized. The parameters are then optimized in a monitor-optimize loop that is done either in simulation or at run-time by the system controller. The idea is that one of the processors already available in the designed system takes also care of logical control and assignment of the communication parameters and, therefore, no dedicated controller is needed.

The run-time arbitration is distributed among the local control units of HIBI wrappers which makes the logical control a minor task in the processor. It is reasonable to perform this in the system control processor, because it can also predict future computation events and, therefore, inform the wrappers about preferred timeslots as well as possible sleep and wake-up of components for power reduction.
6.4. HIBI Wrapper

The HIBI wrapper is the key component in implementing HIBI-based systems. Modularity was one of the main design goals of the interconnection scheme design. With the aid of the wrapper, all the functional components of a system can use the HIBI interconnection as a black box; they do not have to be aware of the protocols and implementation details of the bus. In addition, the performance of the HIBI bus is largely dependent on the wrapper implementation. The wrapper has been implemented in register-transfer-level synthesizable VHDL. Its basic structure is depicted in Figure 6.8. The actual layout implementation can be automatically generated from the VHDL description with the aid of synthesis and place-and-route tools.

The width of the HIBI bus is one of the parameters that can be modified during the system synthesis. The address and data buses can be anything from eight bits upwards, although usually only eight bit increments are viable. The HIBI bus implementation presented in this Thesis utilizes three-state logic, although OR-structure and multiplexer-based implementations are also possible and even preferable in SoCs.
When no component is driving the data and address buses, they are in a high-impedance state. It is the task of the pull-up circuitry to drive them into a known state. The same holds for the command bus. The command bus is three bits wide in the current implementation. The active low reset signal and clock are global signals that are all the time driven by some component. Lock signal, on the other hand, is only driven low if some agent has taken the bus ownership. At other times it should be pulled up.

The IP block side signals of the wrapper can be quite easily modified. The current implementations contain data, address, and command lines to and from the wrapper. To take care of the handshaking, four control and status signals are needed. The connected IP block can command the wrapper with read enable (RE) and write enable (WE) signals. RE is used when the IP block wants to read data and WE when it wants to write. The wrapper can output its status with Full and Data valid signals. The full signal is asserted when the FIFO buffer buffering data from the IP block to the HIBI bus is full. The Data valid signal, on the other hand, is asserted when the FIFO buffer buffering data from the
HIBI bus to the IP block has data, which means that the wrapper FIFO buffers have data for the connected IP block.

Basically the HIBI wrapper interface that is provided to the IP blocks is composed of the utilized FIFO buffer interfaces. This might be unwanted in systems where the latencies introduced by the FIFO buffers become a problem. An IP block can also implement the HIBI bus signals without the HIBI wrapper and, therefore, without the internal FIFO buffers.

The control unit is a simple finite state machine that controls the activities inside the wrapper. The control state machine is presented in detail in Section 6.5. The control has also special address decode logic and event counters. The function of the address decode logic is to give an address hit event when the address on the HIBI bus belongs to the address range of that particular wrapper. The event counters are used to keep track of the timeslots and the clock cycles on the bus.

One very important component of the control unit is the configuration memory which is used to store information about the HIBI bus configuration. In Table 6.3, the content of the configuration memory is tabulated. A short explanation for each field is also given. The values of the configuration memory can be set during synthesis. All but the Device ID can also be changed during the system operation with reconfiguration.

The other internal components: the FIFO buffers, the transmitter (Tx), and the receiver (Rx), are functionally much more simple than the control. The two FIFO buffers are used to buffer data coming from or going to the HIBI bus. Their size can be set before synthesis according to the requirements of the application. The utilization of FIFO buffering enables high throughputs but can prolong the achievable latency. This is a trade-off that can be made in the targeted continuous-media systems. On the other hand, the HIBI protocol specification does not require the usage of FIFO buffering. The transmitter is used to transmit data from the FIFO buffer to the HIBI bus and the receiver for the other direction. The transmitter and the receiver mainly take care of the HIBI bus signalling.
Table 6.3 The configuration memory values of HIBI.

<table>
<thead>
<tr>
<th>Device ID</th>
<th>Unique device identification code that is set in synthesis and cannot be changed at run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Separation</td>
<td>Logically separates the wrapper from the bus, the default value is connected</td>
</tr>
<tr>
<td>Own priority</td>
<td>The priority of the wrapper</td>
</tr>
<tr>
<td>Priorities</td>
<td>The total number of priorities on the bus</td>
</tr>
<tr>
<td>Time frame length</td>
<td>The number of clock cycles in one time frame</td>
</tr>
<tr>
<td>Base address</td>
<td>The starting address of the wrapper address range</td>
</tr>
<tr>
<td>End address</td>
<td>The ending address of the wrapper address range</td>
</tr>
<tr>
<td>Send limit</td>
<td>The maximum amount of clock cycles that a wrapper can continuously access the bus during a priority-based access</td>
</tr>
<tr>
<td>Timeslot beginning</td>
<td>The beginning of the timeslot reserved for the wrapper</td>
</tr>
<tr>
<td>Timeslot end</td>
<td>The end of the timeslot reserved for the wrapper</td>
</tr>
<tr>
<td>First timeslot beginning</td>
<td>The beginning of the first timeslot reserved for others</td>
</tr>
<tr>
<td>Last timeslot beginning</td>
<td>The beginning of the last timeslot reserved for others</td>
</tr>
<tr>
<td>Device dependent</td>
<td>Other configurations that are not defined</td>
</tr>
</tbody>
</table>

6.5. HIBI control state machine

The cells provided by the ASIC standard cell library and the resources embedded in an FPGA affect the way in which an IP block should be implemented. If one is not an ASIC or FPGA process vendor, it seldom makes sense to hand-optimize an IP block just for one process technology. On the other hand, specific optimization to all ASIC and FPGA processes is not viable. Allowing the synthesis software to perform the necessary optimization should be the primary choice. If the required performance is not immediately achieved, the same behaviour expressed differently may result in better implementation. Unfortunately, performance optimization tends to require additional work from the design engineer, a situation that is especially against the IP design methodology.

For intellectual property, the way to avoid poor synthesis results is to do optimization in a technology independent way. Synthesis results presented here show that the form of VHDL description style strongly affects the final performance of the design in question. In addition, one representation cannot be said to be optimal for all technologies. If the logical function does not change,
it is possible to write a computer program that generates different encodings from a high-level description of the system as was shown in [P6]. The designer can then choose the encoding that fits best into the system under design based on the synthesis results of these descriptions.

The HIBI wrapper is a soft core that can be implemented with different process technologies. It is, therefore, essential to optimize the RTL implementation. A crucial part of the implementation is the control state machine that is shown in detail in the following. However, as background information, the basic state machine implementation techniques are first explained.

The most common FSM implementations are Mealy and Moore machines. In a **Mealy machine** the output is a function of the input and the current state, in a **Moore machine** only of the current state. In addition, within these implementations there can be variations in the encoding of the state registers.

The most common state encoding choices are the binary and state-per-bit encoding. **State-per-bit** refers to the number of registers used for each state. A special non-hierarchical case of this encoding is referred to as **one-hot encoding**. **Registered next-state** (RNS) machine is a subtype of a Moore machine. It differs from the traditional implementations in that it has no output decode logic. Typically only state-per-bit encoding for the state registers is used in RNS machines, because binary type of encoding would practically require additional decode logic which would make the whole system a Moore machine.

Synthesis software typically cannot generate all types of state machines from high-level descriptions. This requires the machines to be described at the register-transfer-level. Advantages of RTL coding include better control over combinational logic placement. As registers are specified explicitly, also the location of combinational logic becomes defined. Combinational logic optimization becomes straightforward because the synthesis software typically does not do retiming which means the shifting of logic before and after the registers [Kur95, p. 213-214]. In addition, the concurrency in combinational logic can be explicitly expressed with RTL coding.

Synthesis results can also be affected and optimized by using different VHDL coding styles. **Separated encoding** refers to the normally preferred way of encoding FSMs, which has separate processes for combinational (state transition and output decode) and sequential parts of the state machine. With separated encoding the designer gets a good control over these two parts.

**Combined encoding** integrates the sequential and combinational parts into a single process. The synthesis software usually cannot understand this encoding to be a finite state machine. The advantage of this encoding is that the outputs are implicitly registered. However, this type of FSM is quite difficult to understand and debug.
Mathematical encoding resembles closely the mathematical representation of FSMs. It has a separate process for the output decode part. The state transition logic and the sequential parts, on the other hand, are in a single process. Very promising synthesis results for this type of FSM encoding were given in [Neb98].

Finally, registered next-state encoding has separate processes for the sequential and combinational logic. The difference between RNS and mathematical encoding is that the RNS considers transitions into a state and mathematical encoding transitions from a state to another.

6.5.1 HIBI FSM design

Figure 6.9 depicts the HIBI wrapper state machine controlling the activities inside the HIBI wrapper. The state transition conditions are indicated with a pseudo-code. The write state is divided into three sub-states. Priority Write is used in writes resulting from priority competition. Timeslot Write is utilized during the predetermined transaction times. Reconfiguration Write permits the reconfiguration of the configuration memories.

Since HIBI utilizes distributed arbitration, most of the signals in Figure 6.9 are generated inside the HIBI wrapper. Data_to_transmit is generated by the Tx FIFO when it has data that needs to be sent. Also, the Last_data_to_be_send is generated by the same FIFO when there is only one data item left in the Tx FIFO. The priority counter of the control FSM gives the Own_priority signal when the wrapper can initialize a priority-based bus access. The event counter generates the Timeslots signal if there is a timeslot-based access on the bus. The Own_timeslot signal is generated when an active timeslot belongs to the wrapper in question. In addition, the event counter activates the Time_to_send_block whenever there is enough time for a HIBI streaming burst transfer and Data_transaction_limit when the maximum amount of data in one transaction has been sent.

Some of the signals come from the configuration memory. These include the Connected_to_bus that tells whether the wrapper is connected to the bus or not, Reconfiguration which tells that a reconfiguration operation is going on, and Reconfiguration_finished which marks the end of the reconfiguration process. The only signal generation requiring information from outside the wrapper is No_current_bus_usage which is generated from the Lock signal of the bus. It is activated when no wrapper is transmitting data.
HIBI state machine synthesis results for ASIC and FPGA technologies are tabulated in Table 6.4 and Table 6.5. The best results are underlined. The ASIC technology is a 0.35 µm, three metal layer CMOS process. The target FPGA is a typical look-up table and symmetric routing channel based device (Xilinx XC4005E-3). Values for the ASICs are from the synthesis software before placement and route. The values for FPGAs are synthesis results after the placement and route. For all the machines, the maximum clock frequencies are given in MHz. The area for the ASIC technology is given in the technology’s area units (2-input NAND = 0.141 area units) and for the FPGA technology in logic elements.
In addition to the VHDL coding style, the synthesis results are influenced by the settings of the synthesis software. Typical optimization goals include at least speed or area. Although in SoC design, other matters than wrapper implementation, like processor operating frequencies or interconnect wiring delay, may dictate the speed. All values are given from area optimized synthesis results. All the state machines were coded as Moore machines because a combinational path from FSM input to output was not desired. The output from all the machines was the current state vector.

The best area and speed performance in ASIC technology with one-hot encoding is obtained with registered next-state encoding. Correspondingly, the most area and speed efficient is the combined encoding with the FPGA technology. Binary encoding performs well compared to one-hot encoding in ASIC designs. From the FPGA designs, a one-hot, combined encoding FSM is the fastest, although the differences are smaller. Therefore, the binary encoding performs quite evenly throughout all the sample FSMs.

The fastest encodings exhibited clock frequency increases of 66.2 and 46.8 % over the worst cases in ASIC and FPGA technologies, respectively. These are significant improvements because the maximum frequency of the HIBI wrapper is largely dictated by the FSM operating frequency. Area improvements are also important, because the interface hardware is multiplied in a HIBI-based system. The synthesis results showed area decreases of 34.5 and 41.2 % between the worst and the best cases in ASIC and FPGA technologies, respectively. Moreover, as the wrapper is intended to be used as a soft IP core, and implemented in different technologies, it is convenient to have different VHDL encodings of the same system for the designer to choose from.

There is a 29.1 % decrease in VHDL code length from the longest (RNS) to the shortest (combined) representation. RNS encoding is clearly the most verbose of all the FSM representations. However, the difference in code length between the other FSMs cannot be considered significant.

Table 6.4 Synthesis results for HIBI FSM in 0.35µm ASIC technology.
(Synthesis software netlist estimates.)

<table>
<thead>
<tr>
<th></th>
<th>Sep., one-hot</th>
<th>Sep., binary</th>
<th>Comb., one-hot</th>
<th>Comb., binary</th>
<th>Math., one-hot</th>
<th>Math., binary</th>
<th>RNS, one-hot</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clk freq. [MHz]</strong></td>
<td>346</td>
<td>575</td>
<td>362</td>
<td>495</td>
<td>362</td>
<td>495</td>
<td>370</td>
</tr>
<tr>
<td><strong>Area [area units]</strong></td>
<td>46.9</td>
<td>30.7</td>
<td>42.0</td>
<td>33.4</td>
<td>42.0</td>
<td>33.4</td>
<td>36.8</td>
</tr>
<tr>
<td><strong>Lines of VHDL</strong></td>
<td>182</td>
<td>182</td>
<td>158</td>
<td>158</td>
<td>160</td>
<td>160</td>
<td>223</td>
</tr>
</tbody>
</table>
Table 6.5 Synthesis results for HIBI FSM in look-up table based FPGA technology. (Estimates after place-and-route.)

<table>
<thead>
<tr>
<th></th>
<th>Sep., one-hot</th>
<th>Sep., binary</th>
<th>Comb., one-hot</th>
<th>Comb., binary</th>
<th>Math., one-hot</th>
<th>Math., binary</th>
<th>RNS, one-hot</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Clk freq. [MHz]</strong></td>
<td>24.8</td>
<td>32.2</td>
<td>36.4</td>
<td>33.6</td>
<td>29.3</td>
<td>33.6</td>
<td>32.6</td>
</tr>
<tr>
<td><strong>Area [logic elements]</strong></td>
<td>17</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>14</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td><strong>Lines of VHDL</strong></td>
<td>182</td>
<td>182</td>
<td>158</td>
<td>158</td>
<td>160</td>
<td>160</td>
<td>223</td>
</tr>
</tbody>
</table>

The HIBI state machine that is used in the forthcoming implementations utilizes separated, binary encoding. Binary encoding was chosen because it outperforms the one-hot encoding in almost all the tested cases. Separated encoding, on the other hand, is the encoding understood by the synthesis software. In addition, the separated, binary encoding gave very good results in both the ASIC and the FPGA technology.
7. Implementation and verification of HIBI

HIBI wrapper has been implemented in various bus widths and FIFO depths as well as on various technologies as tabulated in Table 7.1. The different implementations have been used for verifications and analyses that will be presented in the rest of this Thesis. Almost all of the implementations were used at register-transfer-level simulations. In the following, the different implementations will be referenced to by their ID fields in Table 7.1, where the first number is the bus width and the second one the FIFO depth.

Table 7.1 HIBI implementations.

<table>
<thead>
<tr>
<th>Bit width</th>
<th>FIFO depth</th>
<th>Utilization</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>Emulation with Celaro</td>
<td>8.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gate-level simulation (0.35 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synthesis (Celaro emulator)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synthesis (0.25, 0.35 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>8.8</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>8.32</td>
</tr>
<tr>
<td>16</td>
<td>5</td>
<td>Gate-level simulation (0.25 μm CMOS)</td>
<td>16.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Layout (0.25 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gate-level simulation (0.18 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synthesis (0.18 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>16.8</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Synthesis (0.18 μm CMOS)</td>
<td>16.10</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>16.32</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>32.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Synthesis (0.18 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Power estimation (0.18 μm CMOS)</td>
<td>32.6</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>32.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Power estimation (0.18 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Synthesis (0.18 μm CMOS)</td>
<td>32.10</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>Power estimation (0.18 μm CMOS)</td>
<td>32.12</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Synthesis (0.18 μm CMOS)</td>
<td>32.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Power estimation (0.18 μm CMOS)</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Synthesis (0.25 μm CMOS)</td>
<td>32.32</td>
</tr>
</tbody>
</table>
7.1. Layout design of HIBI

For the prototype layout generation, a 16-bit HIBI implementation with five units deep FIFO buffers (HIBI 16.5) was chosen. The layout generated by the place-and-route tool for a 0.25 µm CMOS technology is depicted in Figure 7.1. The layout consists of a core area with 2052 standard cells, two power rings outside the core area, and the I/O pins located on the outermost square. As this block is intended to be used inside a SoC, there are no I/O pads.

![Figure 7.1 Layout of a 16-bit HIBI wrapper.](image)

The utilized technology, the implemented HIBI wrapper, the number of standard cells, the area, and the maximum frequency of the implementation are tabulated in Table 7.2. The total area of the layout is relatively large but it must be emphasized that the two FIFO buffers constitute two thirds of the whole area. The maximum operating frequency of this HIBI wrapper implementation is 100 MHz. This limit was deduced from gate-level estimates of the critical path and verified at this level with timing requirements from the synthesis process.
After the generated layout was finished, layout checkers were used to test the design correctness. The area of the final layout is 605 µm x 605 µm = 0.37 mm² whereas the synthesis process estimated an area of 0.22 mm². This additional area is due to wire routing [Wis03] and the area utilization that cannot be 100% in a placement of a component of this magnitude.

Table 7.2 The layout implementation.

<table>
<thead>
<tr>
<th>Technology</th>
<th>0.25 µm CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapper</td>
<td>HIBI 16.5</td>
</tr>
<tr>
<td>Standard cells</td>
<td>2052</td>
</tr>
<tr>
<td>Area</td>
<td>605 µm x 605 µm</td>
</tr>
<tr>
<td>Frequency</td>
<td>Max. 100 MHz</td>
</tr>
</tbody>
</table>

7.2. Verification of HIBI implementations

In order to verify the interconnection scheme, a prototype design system was prepared to be used with simulations at RTL and gate-level and hardware emulation. It consisted of three HIBI wrappers, their connections to three IP blocks, and the HIBI bus. The structure of the design is depicted in Figure 7.2. An 8-bit HIBI implementation with five unit deep FIFO buffers (HIBI 8.5) was chosen as the test case. In the following, the verification of this design is presented. In addition to verifying the HIBI scheme, the verification process produced some general statistics about different verification methods. Although they are not only HIBI specific, they are still presented in the following.

With the utilized configuration containing only wrappers and no other logic, the verification process could concentrate on the essential ideas behind the interconnection design. In addition, because all the wrapper outputs are read on the next clock cycle after the data is available, the size of the buffers could be kept small. Small area was required by the utilized hardware emulator, as will be discussed later.

Inside all three HIBI wrappers, their own priority, timeslot beginning and end, base address and end address, and device ID configuration memory fields are shown. Inside the bus, the fields of the configuration memory that are the same for all the wrappers, namely the time frame length, the number of priorities, and the send limit field are presented. The send limit could be different in all the HIBI wrappers. Also the bus width is shown. These values are initialized during the system synthesis.
In the verification of the HIBI scheme, the wrapper is in a key role. It makes the actual interconnection invisible to the connected IP blocks. If the wrappers work reliably, verification of larger systems based on the HIBI scheme can concentrate on other issues. Moreover, all HIBI-based systems contain multiple HIBI wrappers, and therefore, a single implementation error would be replicated in the system construction phase.

First, a behavioural, possibly non-synthesizable test bench is required to verify the design operation. The test bench is implemented in VHDL. In it, the IP blocks are replaced by files. All the HIBI wrappers get their inputs from a file. The data that the wrappers send to the bus and receive from the bus is stored in two output files together with a counter value tracking the clock cycles on the HIBI bus.

### Wrapper 1
- Device ID = 1
- Own priority = 1
- Time slot beginning = 0xa
- Time slot end = 0x28
- Base address = 0x0
- End address = 0xf

### Wrapper 2
- Device ID = 2
- Own priority = 2
- Time slot beginning = 0x32
- Time slot end = 0x50
- Base address = 0x10
- End address = 0x1f

### Wrapper 3
- Device ID = 3
- Own priority = 3
- Time slot beginning = 0x5a
- Time slot end = 0x6e
- Base address = 0x20
- End address = 0x2f

Figure 7.2 Block diagram of the prototype design.
bus. This test bench is used in all the subsequent simulations and hardware emulations.

In addition to generating the test bench, test vectors need to be created. It is evident that simulations cannot cover all the possible situations. In the simulations, random testing is used in connection with some corner cases [Kea99, p. 149]. Random testing implies examining the basic operations of the HIBI bus with random data and random transfer timing. This includes writing, reading, and burst transfers by all the components in the system. The corner cases are the ones that are deemed most likely to cause problems. Here, identification of these cases is based on the knowledge of the internal structure of the utilized wrapper. Specifically, reconfigurations and a few other special cases were identified as the possible corner cases.

7.2.1 Simulation and emulation of the HIBI wrapper

The RTL VHDL code of the HIBI wrapper was generated manually without high-level synthesis for performance reasons. The RTL code was then compiled for simulation, simulated with a set of simulators, and compared to the original specification to validate the compatibility and correctness of the code. The RTL simulation results were used as a golden reference to which all the subsequent results were compared to automatically. Because of this, the results of the RTL simulation had to be checked manually very thoroughly.

For gate-level simulations, the RTL code was synthesized using a 0.35 µm CMOS technology. After synthesis, the gate-level netlists were again compiled for simulation. The same test bench as in the RTL simulation was utilized. The files produced by the gate-level simulation were automatically compared to the ones produced by the RTL simulation.

The RTL simulations are mainly useful for finding logical errors in the VHDL implementation. In this case, these included straight-forward lexical errors and errors in the bus signalling. Simulations at the gate-level can be used in verifying implementation details. Timing information, for example the maximum operating frequency of the wrapper implementation, was deduced from the gate-level synthesis results.

Mentor Graphics Single Slot Celaro verification system with a theoretical gate count of 20-50 kgates was used for the hardware emulation. In contrast to the software models of gates used at gate-level simulation, emulation uses real physical gates. Celaro, in particular, uses dedicated accelerated verification chips as programmable elements. On larger scale, the Celaro system consists of general-purpose configurable physical logic blocks (CPBs), dedicated memory blocks, and an interconnection network. In this emulation case, only the three HIBI wrappers and the actual HIBI bus were mapped onto the Celaro system. The non-synthesizable parts of the test bench, namely the file handling part, were run on a workstation.
The results of the emulation synthesis and the results of the synthesis to gate-level are tabulated in Table 7.3. The gate-level implementation consists of 1951 standard cells, whereas the emulator implementation uses 1997 CPBs. The gate count of a single HIBI wrapper synthesized to Celaro was 10761 gates. Although the limit of 50 kgates came quite close with three wrappers the prototype design fitted into the emulator.

The operating frequency of the implementation can be deduced from the gate-level synthesis results. It is evident that the simulations at this level run on much lower frequency. The emulation operating frequency, on the other hand, tells the frequency of the actual verification. Like at the gate-level simulations, also the result of the emulation synthesis has to be compiled for the utilized software tool before it can be used in the emulation.

Table 7.3 Results of the synthesis of one 8-bit HIBI wrapper (HIBI 8.5).

<table>
<thead>
<tr>
<th>Gate-level synthesis</th>
<th>Emulation synthesis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>0.35 µm CMOS</td>
</tr>
<tr>
<td>Standard cells</td>
<td>1951</td>
</tr>
<tr>
<td>Frequency of</td>
<td>Max. 28.17 MHz</td>
</tr>
<tr>
<td>implementation</td>
<td></td>
</tr>
<tr>
<td>Emulator</td>
<td>Celaro Single Slot</td>
</tr>
<tr>
<td>Gatecount</td>
<td>10761</td>
</tr>
<tr>
<td>CPBs</td>
<td>1997</td>
</tr>
<tr>
<td>Emulation frequency</td>
<td>4.69 kHz</td>
</tr>
</tbody>
</table>

One obvious drawback of emulation compared to simulation is its complexity and cost of the required hardware. The speed of the emulation process on the other hand, may in some cases allow an almost real-time HW/SW co-verification of relatively large systems. Because the system is mapped onto real hardware, it is also possible to connect the emulation hardware to actual working systems. The verification case presented here is fairly simple but a more complex system utilizing the HIBI scheme was also developed. In the verification of this system a more heterogeneous approach using a HW/SW co-simulator and the emulator was utilized. The verified system is depicted in Figure 8.5.

With emulation, many implementation related logical errors can be observed and located. These include typically structures in the VHDL code that cannot be synthesized correctly, such as incomplete if-structures. It is also possible to locate these errors with gate-level simulations, but they proved to be quite slow as is described in the next Section.
7.2.2 Test run

Figure 7.3 depicts a part of a verification run from an emulation. The signals and patterns shown are exactly the same as at RTL and gate-level simulations. In the beginning of the test run, reset is deactivated and the wrapper 1 writes data to addresses from 0x10 to 0x14 that belong to the address space of wrapper 2. Wrapper 1 gets the bus ownership in this competition phase because it has the highest priority. The activation and deactivation of the Lock signal mark the beginning and the end of the transaction. The transmitted data is 0x1 to each address and the Write command is put on the Command bus. After this, three consecutive Read requests to addresses from 0x15 to 0x17 are sent by the same wrapper to the same recipient. The Data bus (holding 0x1 that belongs to the address space of wrapper 1) gives the address to which wrapper 2 should write back the requested data. After this initial competition phase, a time-slot phase starts in which the bus is reserved for wrapper 1.

Figure 7.3 Part of a verification test run.

Figure 7.3 shows also the input file of wrapper 1 and the output files of wrapper 1 (data sent) and 2 (data received). The columns of these files are from left to right as follows: address, data, and command. In addition, the clock cycle when the transfer is sent from the IP block and received by the recipient
IP block is given on the right most column of the data sent and data received files, respectively. The first clock cycle begins after the reset is deactivated. A six clock cycle initial latency exists between the send from IP block connected to wrapper 1 and the receiving in the IP block connected to wrapper 2. After this, data is received on every clock cycle. Altogether, the test run used in this emulation was 945 clock cycles long. During the test run, all the wrappers were tested and also burst and configuration write commands were used.

### 7.2.3 HIBI verification results

The total verification time consists of five phases. In the first phase, the scripts required by the verification method are generated. In the synthesis phase, the RTL code is synthesized into an ASIC or FPGA technology and then, in the compilation phase, compiled into the form required by the simulator or emulator.

It is evident that synthesis is not needed for the RTL simulation. The fourth phase (verification) consists of the time it takes to set up and initialize the verification platform and the actual test run. The final phase is the debug phase in which the found errors are located and corrected. In the following Equation 7.1, the times taken by these five phases are added together to form the total verification time [Pet00]. In the Equation, \( Err \) stands for the number of encountered errors.

\[
t_{\text{total}} = t_{\text{script}} + Err(t_{\text{synthesize}} + t_{\text{compile}} + t_{\text{verify}} + t_{\text{debug}})
\]  

(7.1)

The duration of each phase in RTL simulation is considerably smaller than in the other two verification methods. For example, synthesis and compilation for the emulation takes 12 times the time as the compilation for RTL simulation. The time taken by emulation and gate-level simulation are depicted in Figure 7.4. The duration of each step in it is relative to emulation. For example, synthesis for the gate-level is ten times longer than for the emulation.

Even bigger difference between emulation and gate-level simulation exists in the test run time. The actual emulation runs were hundreds of times faster than the test runs at gate-level. Whereas emulation runs in the operation frequency of 4.69 kHz, the frequencies of gate-level simulations are under 10 Hz. When also the set-up time of the environment was considered, emulation was still ten times faster than gate-level simulation. The actual verification time [Pet00] can be expressed with the aid of setup time which is approximately the same for emulation and gate-level simulation, operation frequency (\( F \)), and number of test cycles as in Equation 7.2.

\[
t_{\text{verify}} = t_{\text{setup}} + \frac{\text{NumberOfTestCycles}}{F}
\]  

(7.2)

The time difference between emulation and gate-level simulation should increase linearly when new wrappers and test vectors are added to the verified
This makes the use of emulation in larger HIBI-based systems even more appealing. Note that the values presented here are only fully accurate for the HIBI design and cannot directly be generalized.

The time taken by the generation of the required scripts is excluded from these results. It could, however, be noted that the RTL simulations are significantly easier to script than the other two verification methods. The amount of complexity, seen as the time taken by the generation of the scripts, of the gate-level simulations and the emulations was approximately the same. The same also holds for the complexity of debugging. At the register-transfer-level it is also easier to map the encountered errors to constructs in the VHDL implementation.

![Graph showing verification times](image)

**Figure 7.4 Verification times of the HIBI verification.**

### 7.2.4 Conclusions of the HIBI verification

The verification process quite extensively verifies the correctness of the HIBI wrapper implementation. In addition, other conducted verifications, not
presented here, including several RTL and gate-level simulations as well as the layout generation and verification, makes the existence of design errors even less probable.

In general, the results imply that because RTL simulations work at higher level, they take only a small fragment of the time required by the emulation and the gate-level simulation. Gate-level simulations and emulations are closer to the hardware implementation and give more accurate information about the operation of the design, and therefore, they also tend to take more time.

When compared to emulation, the synthesis and compilation times of the gate-level simulations are longer but the synthesis scripts used at gate-level synthesis were generated to get the smallest and fastest implementation. In the emulation synthesis, the optimizations of area and speed are not that important.

In order to verify the correct operation of new complex digital ICs, many different approaches are required. The verification experiences suggest that simulations and HW emulations at different abstraction-levels are needed in the verification process. As Equation 7.1 suggests, faster verification methods should be utilized when the design is still likely to have a lot of undiscovered errors.

Higher level verification methods, such as RTL simulations, are fast and the verified designs are easy to modify. These methods can be used to detect severe logical and syntactical errors of the designs and implementations early on in the design flow. At lower abstraction-levels (gate-level simulation and emulation) more detailed information of the hardware implementation can be observed. Gate-level simulations are needed to verify implementation details such as timing. Emulation is frequently used to speed up RTL simulations. In addition, emulation is useful in locating synthesis related problems, because test runs are more than hundred times faster than at the gate-level simulations.
8. Analysis of HIBI

8.1. HIBI implementation

As the previous Chapter presented, the HIBI wrapper has been implemented in VHDL and then synthesized to different IC technologies. In addition, place-and-route software has been used for automatic layout generation. In this Chapter, the HIBI wrapper implementation and the HIBI scheme are analyzed. As example technologies, 0.25 and 0.18 µm CMOS technologies are utilized. The implemented wrappers have been synthesized with a clock frequency constraint of 100 MHz.

The results of the implementations to the 0.25 µm CMOS with bus widths of 8, 16, and 32 bits and FIFO depths of 5, 8, and 32 are presented in Figure 8.1. HIBI uses virtual cut-through routing which means that the whole data transaction is not stored into the FIFO buffer of the wrapper. In other words, the buffer size is dictated by the read latency and the buffer can be made smaller than the maximum number of transfers in a transaction or the burst length might seem to require. The required size can be deduced by static application analysis or system simulation. In the presented implementations, the size of both the internal FIFO buffers is the same although sometimes Tx FIFO buffers can be smaller. The results presented are collected from the synthesis results.

The different FIFO depths of the implementations were chosen based on the experiments and analysis of the test cases. Of the depths, 5 is the smallest that works reliably because otherwise the lack of handshaking requires too fast responses from receivers of data. If the targeted application is of a pure dataflow type, this depth is sufficient. FIFO depths of over 32 are not reasonable to implement because of their excessive area. If the analysis or simulation of the application seems to require larger FIFO buffers, high-level handshaking should be utilized in HIBI-based systems. A FIFO depth of 8 is shown as a compromise between these two extreme cases for dataflow systems having a small amount of transfer pattern irregularities.

As can be seen from Figure 8.1, the size of the FIFO buffers has a significant impact on the implementation size. The FIFO width is dependent on the width of the address bus, the width of the data bus, and the width of the command bus, since all these three components need to be stored in the FIFO buffers. With FIFO depth of 5 units, the 32-bit wrapper is only 0.4 mm² in area. When the FIFO depth is 32 units, even the size of the 8 bit wrapper is over 0.6 mm².

Although some implementations can benefit from large buffers, the size of the FIFO buffers should generally be minimized to save area. It should be noted again that the FIFO buffers were directly synthesized from the VHDL code.
more efficient implementation is possible if silicon vendor provided memory blocks are used. This, on the other hand, makes the reuse of the wrapper more difficult.

![Figure 8.1 Sizes of different HIBI wrappers on 0.25 μm CMOS.](image)

The results of the implementations to the 0.18 μm CMOS with bus widths of 16 and 32 bits and FIFO depths of 5 and 10 are presented in Table 8.1. The maximum operating frequencies implied by the wrapper delays (202 - 295 MHz) are sufficient for quite demanding tasks, but the relatively large area may become problematic in some area critical implementations. On the other hand, the use of HIBI wrappers takes some of the interconnection related burden, like bus signalling and buffering, from the actual functional components which is essential in the separation of communication and computation in the design process of large SoCs.

In addition, Table 8.1 shows the implementation results of the different HIBI wrapper sub-components. It is evident that the FIFO buffers are the most area consuming components and that they also induce the largest delays. Only the control component comes close to the area and delay of the FIFO buffers. Interesting synthesis related phenomena is the delay of the 32-bit, 10 deep FIFO buffer which is longer than the delay of the corresponding wrapper. This can be caused by the synthesis tool being able to optimize the design over the sub-component borders.
Table 8.1 Different HIBI wrapper implementations for 0.18 µm CMOS.

<table>
<thead>
<tr>
<th>Unit</th>
<th>16 bit Area</th>
<th>16 bit Delay</th>
<th>32 bit Area</th>
<th>32 bit Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO = 5</td>
<td>0.027 mm²</td>
<td>3.17 ns</td>
<td>0.050 mm²</td>
<td>4.85 ns</td>
</tr>
<tr>
<td>FIFO = 10</td>
<td>0.057 mm²</td>
<td>4.82 ns</td>
<td>0.114 mm²</td>
<td>5.22 ns</td>
</tr>
<tr>
<td>Control</td>
<td>0.029 mm²</td>
<td>2.96 ns</td>
<td>0.034 mm²</td>
<td>2.90 ns</td>
</tr>
<tr>
<td>Tx FSM</td>
<td>0.006 mm²</td>
<td>0.93 ns</td>
<td>0.011 mm²</td>
<td>0.91 ns</td>
</tr>
<tr>
<td>Rx FSM</td>
<td>0.008 mm²</td>
<td>1.07 ns</td>
<td>0.015 mm²</td>
<td>1.35 ns</td>
</tr>
</tbody>
</table>

Whole wrappers with

| FIFO = 5      | 0.093 mm²   | 3.39 ns      | 0.151 mm²   | 4.28 ns      |
| FIFO = 10     | 0.148 mm²   | 4.94 ns      | 0.275 mm²   | 4.88 ns      |

Because both the input and the output of HIBI wrapper are registered, the delay of the bus signal lines can be almost as long as the targeted clock cycle. Only the hold time of the output register and the setup time of the input register have to be taken into account as is depicted in Figure 8.2.

Figure 8.2 Timing of HIBI bus signals.

In Table 8.2, the HIBI wrapper is compared to various on-chip wrapper implementations in which at least some of the needed features for comparison
have been published. The technology of the presented implementation, the interconnection width, the gate count, the area, the frequency, and a reference are tabulated. As can be seen, the size of HIBI is smaller than most of the other interconnections and its achievable operating frequency is also comparable.

The last three rows of Table 8.2 tabulate three example IP blocks. ARM7TDMI is given as an example of a SoC processor core. The area of the core compared to HIBI block is 3-5 times larger than the presented HIBI wrappers. It is, therefore, reasonable to utilize HIBI in connecting ARM7TDMI processor cores. If the IP blocks are very small, using a separate HIBI wrapper for each IP block is not a good approach. For example, HIBI is 2-4 times larger than the presented DES block. In this kind of a case, perhaps a better way is to connect several smaller IP blocks to a single HIBI wrapper utilizing a separate multiplexer structure. The last IP block example, on the other hand, is a fairly large detector block that is 36-62 times larger than the HIBI wrappers. With IP blocks of this size, the area consumed by the HIBI wrapper becomes insignificant.

Table 8.2 Comparison of HIBI implementation to NoC cells and IP blocks.

<table>
<thead>
<tr>
<th>IP</th>
<th>Tech.</th>
<th>Width</th>
<th>Gates</th>
<th>Area</th>
<th>MHz</th>
<th>Ref.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIBI *</td>
<td>0.18</td>
<td>16b</td>
<td>7k</td>
<td>0.093</td>
<td>295</td>
<td>[P4]</td>
</tr>
<tr>
<td>HIBI *</td>
<td>0.18</td>
<td>32b</td>
<td>12k</td>
<td>0.151</td>
<td>234</td>
<td>[P4]</td>
</tr>
<tr>
<td>Noc Cell</td>
<td>n/a</td>
<td>32b</td>
<td>10k</td>
<td>n/a</td>
<td>n/a</td>
<td>[Mor03]</td>
</tr>
<tr>
<td>Noc Cell</td>
<td>0.13</td>
<td>32b</td>
<td>n/a</td>
<td>0.26</td>
<td>166/500</td>
<td>[Rij03]</td>
</tr>
<tr>
<td>Noc Cell</td>
<td>0.35</td>
<td>n/a</td>
<td>15k</td>
<td>0.8</td>
<td>110-267</td>
<td>[Val01]</td>
</tr>
<tr>
<td>Proteo **</td>
<td>0.18</td>
<td>8b</td>
<td>16k</td>
<td>0.2</td>
<td>n/a</td>
<td>[Jan03]</td>
</tr>
<tr>
<td>SPIN</td>
<td>0.13</td>
<td>32b</td>
<td>n/a</td>
<td>0.24</td>
<td>200</td>
<td>[And03]</td>
</tr>
<tr>
<td>ARM7TDMI</td>
<td>0.18</td>
<td>n/a</td>
<td>n/a</td>
<td>0.53</td>
<td>90</td>
<td>[Arm03b]</td>
</tr>
<tr>
<td>DES ***</td>
<td>0.35</td>
<td>n/a</td>
<td>3k</td>
<td>n/a</td>
<td>50</td>
<td>[Lim00]</td>
</tr>
<tr>
<td>MIMO ****</td>
<td>0.18</td>
<td>n/a</td>
<td>438k</td>
<td>n/a</td>
<td>122</td>
<td>[Dav03]</td>
</tr>
</tbody>
</table>

* Fifo size 5 x bus width
** Buffer space is fixed independent of network segment width
*** Data encryption standard block
**** Multiple input, multiple output detector

8.2. Performance of HIBI

8.2.1 Test case 1

As the first test case, a system with three 16-bit HIBI wrappers (HIBI 16.8) has been constructed. The structure of the constructed test case system is presented in Figure 7.2. Out of the data transactions utilized in the test run, roughly 20% were streaming burst transfers, 40% read requests, and 40% write commands.
The transfer distribution is depicted in Figure 8.3. The operations initialized by the wrappers are shown in the x-axis and the y-axis represents the percentage of these operations as the total amount of transactions made by the wrapper. For example, 10% of the operations by wrapper 1 are bursts to wrapper 2.

The used burst length was six. The data was random, but all the agents repeated their transfers in a certain interval. All the agents also had the same amount of data to send. In the presented test case, reconfiguration is utilized. The clock cycles taken by the reconfiguration have not been included in the presented results, since they are not used in normal operation after initialization.

Figure 8.3 The transfer distribution in the test case 1.

Figure 8.4 depicts the percentage of time that each wrapper reserved the bus in one of the simulations. In addition, the percentage of time that the bus is reserved, but not transmitting data, is shown. These original results are shown in the left of Figure 8.4.
In this analysis, the bus efficiency and the throughput are defined with the aid of the number of clock cycles transmitting data (CCTD), the number of reserved clock cycles (CCR), the number of data transactions (DT), and the number of clock cycles (CC) in Equations 8.1 and 8.2.

\[
\text{Bus efficiency} = \frac{\text{CCTD}}{\text{CCR}} \quad (8.1)
\]

\[
\text{Throughput} = \frac{\text{DT}}{\text{CC}} \quad (8.2)
\]

In the first phase (original), the bus parameters were not optimized for the transfer patterns, which led to a bus efficiency of 83.29% and a throughput of 0.99 transactions per clock cycle. The high throughput value is made possible with the use of the streaming burst transfers of HIBI. When this feature is used, the theoretical maximum throughput limit of the bus approaches two transactions per clock cycle as data is transmitted also on the address bus.

After this original test run, the bus parameters were reconfigured to fit the utilized data. The most notable parameter in this reconfiguration is the time frame length. It is changed from 120 to 100 to fit the interval in which the agents repeat their transactions. This small adjustment has a considerable effect on the outcome. After this, the bus efficiency was 93.09% and the throughput...
1.08 transactions per clock cycle. It should also be noted that the total number of clock cycles needed to complete the simulation is now less than originally.

The configurations of the test case are tabulated in Table 8.3. As was shown, the fitting of the bus parameters dramatically increases the bus efficiency which is the sum of the wrapper bus utilizations. This is natural due to the arbitration scheme. In addition, the time during which the bus is reserved but not transmitting data decreases significantly, which is desirable since it leaves room for future expansions.

When timeslots and streaming burst transfers are used, the turn-around latency dominates the bus efficiency. This is caused by the fact that in HIBI, the turn-around latency is only present after timeslots. On the other hand, the use of timeslots eliminates the arbitration latency and throughputs of over one can be reached with the streaming bursts.

Table 8.3 The configuration values of the test case 1.

<table>
<thead>
<tr>
<th>Priority</th>
<th>Original</th>
<th>After Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapper 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Wrapper 2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Wrapper 3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time slots</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapper 1</td>
<td>10 – 40</td>
<td>6 – 28</td>
</tr>
<tr>
<td>Wrapper 2</td>
<td>50 – 80</td>
<td>30 – 62</td>
</tr>
<tr>
<td>Wrapper 3</td>
<td>90 – 110</td>
<td>64 – 96</td>
</tr>
<tr>
<td>Time frame</td>
<td>120</td>
<td>100</td>
</tr>
</tbody>
</table>

**8.2.2 Test case 2**

In the previous test case, all the components had data to transmit all the time. This is a good way to test the bus operation, but it does not represent real workloads very well. In a more typical case, the connected components need to wait for transfers from other wrappers. Situations like this are encountered for example in video coding, which is the second test case.

Some HIBI compliant IP blocks have been implemented. These include high-level models for discrete cosine transform (DCT) and inverse discrete cosine transform (IDCT), as well as motion estimation for video encoding [Bha95, pp. 1-128]. In addition, SRAM memory blocks, ARM7TDMI processor core [ARM03b], and TMS320C6202 digital signal processor core [TI04] have been used in a HIBI-based system [Kan00]. The utilization of these cores demands an adapter between the core and the HIBI wrapper [Sal01].
As an example, the implementation of the ARM7 connection is depicted in Figure 8.5. An important thing to note is that the HIBI bus is not intended to be used as a processor local bus. As Figure 8.5 shows, the ARM proprietary bus is used for this task. It connects the instruction memory (ROM), the dual-port memory (DPRAM) used for data storage, and the control part of the required adapter. Based on the control signal information, the adapter, which includes a DMA (direct memory access) controller, can transfer data from the DPRAM to the HIBI wrapper.

![Figure 8.5 Connecting ARM7 to the HIBI bus.](image)

As an example of a HIBI-based system, a H.263 video encoder [ITU96] has been designed. It consists of the previously mentioned components and is depicted in Figure 8.6. The I/O block is used for off-chip communications. In this implementation, the control block is an ARM7TDMI processor and the DCT/IDCT block and the motion estimation block have been implemented with TMS320C6202 processors. These IP blocks are connected by 32-bit HIBI wrappers (32.32). The size of the wrappers and, especially, the size of the TMS320C6202 make this system currently impractical for implementation on chip. The purpose of this example is, however, to demonstrate the capabilities of HIBI and its use in system design.

The idea behind platform-based design using HIBI is to first optimize the bus usage. In higher level simulations, C language or higher level process models of the required functional blocks can be used. After the bus usage is optimized, the system can be verified with cycle-accurate simulations. During these simulations throughput estimates of the functional blocks can be added to the C models.

Another alternative is to use a method called hardware/software co-simulation as was used for example in [P9]. In the presented case, the ARM7TDMI processor cores were modelled with an instruction set simulator (ISS) and a bus interface model (BIM), the memory blocks were modelled with VHDL, and the dataflow type algorithm blocks (DCT/IDCT, motion estimation) were
modelled with C-language. All these are simulated in a co-simulator or common simulation backplane that provides a method to connect the ISS, VHDL simulator, and the C-models using the foreign language interface of the VHDL simulator.

The implemented H.263 encoder is a reasonable representation of the targeted continuous-media applications. The real-time encoding of quarter common intermediate format (QCIF) sized video sequence called “foreman.qcif” was used as a test case. Frame rates topping 20 frames per second (FPS) with operating frequencies considerably below 100 MHz have been achieved [Kan00] although this video codec was not optimized for encoding performance. The video codec was, instead, utilized to test the interconnection scheme. These video encoder simulations have demonstrated that HIBI interconnection fits well into this type of applications.

Figure 8.6 The video encoder test case 2.

Figure 8.7 depicts the percentages of time that the different components in the video encoder reserve the bus. The results are from a single video frame being encoded as an INTRA and INTER frame. Altogether 234330 clock cycles were used for encoding the INTRA frame and correspondingly 310145 clock cycles

137
for the INTER frame. As depicted in Figure 8.7, motion estimation is not done in the encoding of the INTRA frame.

The clock cycles spent waiting for the functional components to finish their operations and the clock cycles during which the bus is idle are omitted from these figures. The achieved bus efficiency is 98.28% in both the INTER and the INTRA case. This high bus efficiency could be reached with a combination of priority-based accesses and the use of a relatively high send limit (384 because a macroblock in H.263 contains 6*8*8 pixels). Therefore, the timeslots were not used at all. Without timeslots and streaming bursts, the throughputs cannot exceed one. In this case, they were 0.98 transactions per clock cycle for both the INTER and the INTRA frames.

Test case 2

![Figure 8.7 Bus utilization in test case 2.](image)

8.2.3 Conclusions of the HIBI test cases

The relationship between throughput, bus efficiency, number of read and write commands in a time frame ($RW$), number of burst commands in a time frame ($BU$), time-frame length ($L_{TF}$) and the streaming burst transfer length ($L_b$) is described in Equation 7.3. The lengths are defined as clock cycles and the address and data buses are assumed to be of an equal width.
Throughput = \text{Bus \_efficiency} \left( \frac{RW}{L_{TF}} + 2 \frac{BU}{L_{TF}} - \frac{L_b}{L_{TF}} \right) \quad (7.3)

The first term inside the brackets represents the portion of the data transfers that require one clock cycle to transmit one data element. The following two terms represent burst transfers. They take one clock cycle to set up, during which one data element is transmitted, after which two data elements can be transmitted in each clock cycle. The second term inside the brackets represents the ideal case in which two data transfers in a clock cycle could be made from which the third term subtracts the one clock cycle set up time for each starting burst. The terms inside the brackets give the theoretical average amount of data transactions per clock cycle in a time frame. When this is multiplied by the bus efficiency, the real throughput of the system is obtained.

The figures of the first test case after configuration (percentage of read and write commands 80 \%, ratio of burst commands 20 \%, streaming burst length 6, and bus efficiency 93.09 \%) give a throughput of 1.08 as expected. If throughput is to be increased, the possible choices are increasing the bus efficiency and increasing the number of streaming burst transfers.

Theoretically, if all the bus transactions could be divided into six streaming bursts in test case 1 with one burst from each wrapper to every other and the bus efficiency stayed the same, as it should, a throughput of 1.81 data transactions per clock cycle can be achieved. On the other hand, if the ratio of streaming bursts stayed the same and the bus efficiency was 100 \%, the throughput would be 1.17.

The theoretical upper limit for the throughput is 2. It can be achieved with a bus efficiency of 100 \% and infinite time frame and streaming burst lengths, which requires a single streaming burst transfer of infinite length. Achievable throughputs are dictated by the application or how extensively timeslots and streaming burst transfers can be utilized.

Whereas the test case 1 utilized TDMA arbitration and reconfiguration, these features were not used in the video encoder test case. The transaction lengths in the video encoder vary depending on the data, and therefore, TDMA cannot be utilized effectively. Ideally, TDMA arbitration would fit a dataflow application that has transactions of constant length. Un-deterministic control applications fit poorly to a static arbitration scheme and, therefore, they require the utilization of the second-level arbitration method of HIBI.
8.3. HIBI power consumption

8.3.1 System for power analysis

Data transfers in the HIBI bus of the video encoder test system during the encoding of one frame were estimated in [Kan02] and are tabulated, named after the initiators, again in Table 8.4. In the presented case, timeslots are utilized and the size of the utilized FIFO buffers was assumed to be between six and sixteen. The idle bus cycles appear when the system has to wait for the computation in the functional components to finish. Arbitration cycles are needed for deciding the bus owner. They are included because it is difficult to keep the timeslots in correct places in relation to the ongoing computation. Because of this, re-synchronization is needed in TDMA-based systems.

Table 8.4 Data transfers in the video encoder [Kan02].

<table>
<thead>
<tr>
<th>Component</th>
<th>Clk cycles</th>
<th>% of all cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>71700</td>
<td>47.6 %</td>
</tr>
<tr>
<td>Arbitration</td>
<td>2643</td>
<td>1.8 %</td>
</tr>
<tr>
<td>DCT/IDCT</td>
<td>36957</td>
<td>24.5 %</td>
</tr>
<tr>
<td>Motion estimation</td>
<td>19597</td>
<td>13.0 %</td>
</tr>
<tr>
<td>I/O</td>
<td>9505</td>
<td>6.3 %</td>
</tr>
<tr>
<td>SRAM</td>
<td>9504</td>
<td>6.3 %</td>
</tr>
<tr>
<td>Control</td>
<td>690</td>
<td>0.5 %</td>
</tr>
</tbody>
</table>

8.3.2 Initial power results

The switching activity of the CMOS devices strongly affects the dissipated power. Here, the power consumption results are estimated with the help of switching activity files (.saif) obtained from the RTL simulations of the video encoder, and therefore, they are not average or peak values but example values in one real application. In the presented case, the activity file is based on the DCT/IDCT component because it handles the largest amount of data transfers and should, therefore, yield the highest activity. Results that are more accurate could have been obtained through gate-level simulations, but the power consumption estimates are accurate enough for the relative analyses done here.

In Table 8.5, the HIBI wrapper synthesis results to a 0.18 µm CMOS process technology with operating voltage of 1.6 V are tabulated. All the synthesis results in the Table are optimized for area. The size, maximum operating frequency, power consumption, and power consumption/frequency are given for several HIBI wrappers. No low-level power optimization methods were used in the synthesis of the first four wrappers.

As an example, a power optimized version of one wrapper is presented in the fifth column. It uses low-level clock gating and low-leakage cell library that is based on a CMOS process with the same operating voltage. It must be noted
that the component is not a very good candidate for these types of low-level optimizations due to its control-dominated nature. The wrapper leakage power is relatively small compared to the switching power and it does not have many internal components in which to utilize low-level clock gating efficiently.

The functionality of the video encoder SoC can be implemented for example with programmable processors. This analysis utilizes Leon2 [Gai03] and ARM7TDMI 32-bit RISC processor cores presented in Table 8.5. The results of the Leon2 core are based on synthesis with the same technology and similar scripts as used in the HIBI wrapper synthesis. On the other hand, the results for the ARM7 core are given by the vendor in a generic 0.18 µm CMOS technology [ARM03b].

The SPARC V8 compliant Leon2 processor core represents a high-performance processor core with cache memory controllers and multiple bus interfaces. The ARM7, on the other hand, is a more compact and simpler processor core. If processor cores are to be used, memory blocks for data and instruction storing are needed. These are left out of the experiments and estimations of this analysis. In addition, the same computation requirements that were presented in Table 8.4 are used for both processors. In practice, Leon2 can perform the required computations faster than ARM7.

Table 8.5 Initial area, frequency, and power results.

<table>
<thead>
<tr>
<th>Property</th>
<th>Wrapper (soft core)</th>
<th>* Leon2 (soft core)</th>
<th>ARM7 (hard core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus width [bits]</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>FIFO depth [words]</td>
<td>6</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Size [mm²]</td>
<td>0.17</td>
<td>0.21</td>
<td>0.30</td>
</tr>
<tr>
<td></td>
<td>0.37</td>
<td>0.19</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>0.53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Freq. [MHz]</td>
<td>103</td>
<td>108</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Power [mW] @ 100 MHz</td>
<td>83.8</td>
<td>84.8</td>
<td>86.6</td>
</tr>
<tr>
<td></td>
<td>86.5</td>
<td>82.2</td>
<td>458.5</td>
</tr>
<tr>
<td></td>
<td>22.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power/Freq. [mW/MHz]</td>
<td>0.84</td>
<td>0.85</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>0.87</td>
<td>0.82</td>
<td>4.59</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* = Power optimized HIBI wrapper

The HIBI wrapper power consumption is more than three times that of the ARM7 core although the area of it is less than half of that of the ARM7. It should be noted that no hand optimization was done for the wrapper and that the internal FIFO buffers were synthesized (register-based). This follows from the fact that the HIBI wrapper is a soft core whereas the ARM7 is a hard core.
The Leon2 core, on the other hand, is synthesized and therefore comparable to the HIBI wrapper. As expected, the Leon2 is much larger and consumes a lot more power than the ARM7 and the HIBI wrapper.

Low-level power optimization methods can trade area and performance for lower power consumption. The example low-level optimization does not give much improvement, because the same tight clock frequency constraint was used in both the original synthesis and the optimized version. This does not leave much room for the trade-offs. In addition, the statistical system information about the block usage is not utilized in low-level clock gating.

The subsequent analysis is based on a 32-bit HIBI wrapper with a FIFO depth of 6 (32.6). In Table 8.6, the area and power consumption of its internal components are tabulated. The FIFO buffers are the most power and area consuming components. It should also be noted that 95.6% of the HIBI wrapper power consumption is dynamic and 4.4% static with this technology.

Table 8.6 Area and power of HIBI 32.6.

<table>
<thead>
<tr>
<th>Component</th>
<th>Area</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapper</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>- Control</td>
<td>13.5%</td>
<td>9.7%</td>
</tr>
<tr>
<td>- Rx FIFO</td>
<td>34.9%</td>
<td>35.0%</td>
</tr>
<tr>
<td>- Tx FIFO</td>
<td>39.4%</td>
<td>34.5%</td>
</tr>
<tr>
<td>- Receiver</td>
<td>6.1%</td>
<td>9.2%</td>
</tr>
<tr>
<td>- Transmitter</td>
<td>6.1%</td>
<td>11.6%</td>
</tr>
</tbody>
</table>

**8.3.3 System-level power consumption**

The purpose of this analysis is to demonstrate how the run-time behaviour statistics can be used in reducing power dissipation. The presented estimates are not based on the exact power consumption of the computation, for example how much energy is consumed in DCT computation. Instead, gathered statistics of the bus transactions are used to calculate the power consumption used in the communication of the system.

A very effective way to reduce power dissipation at system-level is to use clock gating for the wrapper components (DCT/IDCT, control, and motion estimation) that are not used at that particular time. The TDMA arbitration demands that the data transfer occurrences (starting time, initiator, and length) are known in advance and this same information can also be used in power minimization.

In continuous media systems, like the H.263 video encoder, also the computation times taken by the components, located between receiving and sending data, can be quite accurately predicted. Therefore, TDMA arbitration information can be used in power reduction and a separate controller for power
minimization is not needed. An important thing to notice is the distributed nature of the power control in HIBI-based systems, which can be used to make a RTOS-based power control or another high-level control unit unnecessary. The possible power modes of the HIBI wrapper are the following with clock gating used for inactive components:

1) All wrapper components active
2) Transmitter and transmitter (out) FIFO inactive
3) Receiver and the receiver (in) FIFO inactive
4) Receiver, transmitter and both FIFO buffers inactive
5) All components active, low-level optimization used

The first two can be utilized in practice with current implementations of HIBI. Due to the TDMA arbitration, the transmitting component is known at all times. Alternatives 3 and 4 are more advanced methods that can be used if it is also known when a certain wrapper receives data. In addition, mode 3 can be used when a HIBI wrapper transmits data. However, power modes 3 and 4 are not implemented in the current version of the wrapper and the results of these modes are based on estimates.

The only component that needs to be active all the time is the control. It ensures that each wrapper is synchronized with the rest of the system. The fifth power saving method is to use low-level optimizations in synthesis, in this case low-leakage cell library and low-level clock gating.

The black bar of Figure 8.8 depicts the relative power consumption of a HIBI wrapper compared to the situation when all the internal components are active using these five modes. As can be observed, the saving obtained through low-level optimizations is not significant, only about 2 %. This surprisingly low result can be explained by the HIBI wrapper structure and the use of automatic tools.

Power modes 2 and 3 decrease the power consumption by approximately 40 %. When the fourth power mode is used only the control is active and, therefore, the power consumption is decreased by 79 % of the original value because only control power and the static power of the rest of the components are left.

---

1 An article located at the Synopsys web-page (www.synopsys.com) entitled “Industry's first RTL power optimization feature significantly improves power compiler's quality of results” gives examples where a power saving of 35-60 % is obtained utilizing this technique.
Figure 8.8 Reducing power consumption of the video encoder with wrapper clock gating.

At the system-level, power dissipation can be reduced with the same methods as was used for the HIBI wrapper. This is depicted in Figure 8.8 also for the video encoder. As described previously, the first bar (black) represents the power saving obtained only in the interconnection wrappers. The middle bar (grey) and the last bar (white) stand for the power saving if all the functional blocks of the encoder are implemented with ARM7 or Leon2 processor cores, respectively.

In Figure 8.8, only the power dissipation of the interconnection is reduced. The power consumption of the interconnection becomes less significant when the processors become more complex. With power mode 2, the power consumption of the systems based on ARM7 and Leon2 decreases by 36.5 and 11 %, respectively. Particularly the power consumption of the Leon2 core dominates the power consumption of the video encoder system.

The idea of using TDMA information in power reduction can be taken even further if the connected processors are switched off when they are not used utilizing power supply shutdown. This process can be controlled by the HIBI wrapper control utilizing the arbitration and computation time information. Although this is not usually practical because of the long recovery time, modern low-power processors usually have some sort of low-power modes.
Nevertheless, only switching off the processors power supply completely is considered as the lower bound of power consumption and depicted in Figure 8.9. As registers need to be stored into the main memories of the processors before system shutdown and read back after system power-up, the number of registers (ARM7 = 37, Leon2 = 136) is used to represent the additional clock cycles during these processes.

In the video encoder, one processor needs to be active during the computations and two during the data transfers. With power mode 2 and power supply shutdown, the power consumption of the video encoder decreases by 41 and 47 % for ARM7 and Leon2 processor-based systems, respectively. With mode 4 and power supply shutdown, these same figures decrease by 76 and 58 %.

![Figure 8.9 Reducing the video encoder power consumption with wrapper clock gating and processor power supply shutdown.](image)

**8.3.4 Power consumption conclusions**

Functionality and performance requirements for portable systems keep increasing, exceeding the expected improvement in battery technology. Therefore, the minimization of power dissipation is becoming an extremely important requirement in portable system design. Power-aware design methods need to be applied throughout the design time. Low-level optimization at the end of the design flow does not yield acceptable results.

The use of a SoC interconnection platform makes it possible to optimize the system interconnection backbone also for power. If system behaviour is known
in advance, as is the case when TDMA is used as arbitration method, clock
gating and power supply shutdown guided by the TDMA information can be
used to tune the interconnection to consume as little power as possible. TDMA
has drawbacks in the form of synchronization costs, but if the application has
relatively long and repetitive data transfers, these costs can be justified.

The experiments show that power saving of around 40 % can be achieved in
the implemented interconnection components. With more advanced methods
even a power reduction of 80 % is possible. In processor-based systems, the
power consumption is estimated to decrease with 40 - 50 % depending on the
processor and utilizing the methods presented here. All these methods are
applied at system-level. In addition, low-level hand optimizations can be
utilized as a final means of reducing power consumption.

HIBI is an excellent platform for all the presented power saving techniques
since it provides distributed control with information about the activities of the
IP blocks and wrappers. This information can be utilized by the power control,
whether it is a real-time OS or a hardware controller, in the required decision
making. In addition, by separating the communication and computation parts of
the system, HIBI enables them to utilize their own power saving methods
without interfering with each other.

8.4. Conclusions about the HIBI scheme

The presented HIBI scheme is an example of a bus-based SoC interconnection
architecture. The topology of HIBI is a hierarchy of single bus segments which
is also preferred in many other SoC buses. HIBI has also many other typical
SoC bus features including the support for multiple clock domains, split
transactions, burst transfers, broadcast commands, and reconfiguration of
system parameters.

What separates HIBI from many other SoC schemes is that a modular and
easily modifiable wrapper is provided. This can be used by system designers to
build up large-scale systems. In addition, HIBI has an original distributed
arbitration scheme that is based on TDMA for dataflow applications and
priority based competition for applications with less determinism in their data
transfers.

HIBI is a system backbone bus similar to AMBA AHB [ARM99], AMBA AXI
[ARM03a], CoreConnect, [IBM01], PI-bus [OMI96], SiliconBackplane
[Son00a], and Wishbone [Sil01]. It has not been designed to be a processor
local bus because the arbitration scheme is not optimized for conveying control
information. On the other hand, HIBI might be too complex to be practical in
connecting very small peripheral devices.
The future developments of HIBI include the support for several topologies, added modularity of the wrapper, and the use of a standard interface. Also the future versions of HIBI will be based on the bus topology but if the requirements of the application cannot be met by a hierarchical bus, they can utilize more complex network structures for some of the global interconnections. Added modularity is one of the most important new features of HIBI making it possible to strip off features that are not wanted in a particular application resulting in smaller implementations. These features include, for example, the implementation type of configuration memory which can be RAM for modifiable or ROM for hardwired memory implementation. In addition, the presented version of HIBI utilized directly a FIFO interface. In the future, also an OCP compliant interface will be provided.
9. Discussion

“Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities.” [Amd67]

The continuing advances of IC fabrication technology has been constant for the past decades and will undoubtedly continue at least for the next fifteen years. Particularly the development of memory implementation will play a key role since most of the area consumed by future on-chip systems is used for data storage. Along with the fabrication process development, the design techniques will continue to progress by applying ever higher abstraction-levels.

In the architecture-level, much has been done to lower the design, verification, and fabrication costs of electronic systems. The so-called system or network on chip architectures have been proposed as a solution to their inherent complexity issues. Their design has been based on experiences of former electronic systems of different scale, but it is still left to be seen whether the old techniques are applicable to the new requirements.

Most of the proposed architectures are principally heterogeneous systems implemented with processors and hardware accelerators. The question that needs to be asked is, why not use a single CPU that is powerful enough to do the task by itself? This is an approach that is argued for in the quote at the beginning of this Section.

In most cases, a single processor solution is a viable option, but there are some requirements that cannot be met by a general purpose, single processor. From the power consumption and computation efficiency point of view, it is sometimes justified to utilize specialized processors or accelerators that take the minimum amount of time in executing a specific task and can be switched off at other times.

The most demanding task in the implementation of heterogeneous systems is not usually the design of the architecture. Fitting the application into the architecture is frequently the challenging task. This is due to the fact that most of the applications are inherently sequential or at least implemented in that way and, therefore, cannot take full advantage of a complex parallel interconnection architecture. An additional problem is that the applications are frequently irregular making it hard to fit them into an otherwise preferred geometrically symmetric architecture.

Because of these problems, the goal of architecture design should be to make application fitting easy by taking the targeted application into account as early in the design flow as possible. For other cases than a general purpose CPU, it is
not a good approach to design highly complex architectures with an abundance of resources and then try to fit applications into them.

9.1. Results of Thesis

“Measures of the effectiveness are necessarily problem-based. Therefore, comparisons between parallel and simplex organizations frequently are misleading since such comparisons can be based on different problem environments.” [Fly72]

This Thesis presented theoretical studies of topologies used in multiprocessor systems. The thorough understanding of their cost, performance, and scalability issues made it possible to do rough comparisons among them. In addition, the studies enabled the understanding of the fundamental limits found in different network topology implementations. The theoretical part of Thesis is based on an extensive study that has been done for the following publications: [P1], [P2], and [P4].

After the initial theoretical studies, some selected architectures were implemented for further study. These architectures were utilized in trying out some implementation techniques and practical issues that were not visible in the theoretical studies. A very important aspect of these studies was that they took the projected transfer patterns into account which is the best way to do fair comparisons as is argued in the quote at the start of this Section. The first practical studies dealt with the implementation of control in digital systems utilizing finite state machines [P5] and [P6]. In addition, two architecture studies were conducted. The first one of them dealt with buses and crossbars [P2] and the second one with single buses, hierarchical buses, and 2-D meshes.

The conducted studies aided in the development of the heterogeneous IP block interconnection (HIBI) scheme. The design, verification, and analysis of this scheme form the major part of this Thesis. The initial design and implementation of the scheme was described in [P4]. The scheme was then utilized in studies of system design [P9] and verification [P8], as well as implementation details concerning digital system design flow [P7], finite state machines [P5], and power consumption [P3].

The main technical achievements of this Thesis are the conducted theoretical and practical interconnection architecture studies and the design of the HIBI scheme. The studies point out the possible choices for SoC interconnections and their theoretical and practical limitations. This should help an architecture designer in making the right decision, which is always a trade-off between different concerns like the performance and the cost.

The HIBI scheme presents a tested and documented interconnection architecture that can be utilized as a reference by others. The best properties of
the scheme are the implemented wrapper (modularity and versatility), utilized arbitration scheme (performance), and the fact that it is based on a simple bus interconnection (cost). The HIBI interconnection is best suited to the task of a system backbone bus.

However, HIBI has not been designed to replace all the buses in SoCs. It is well suited to dataflow applications where the data transfers can be predicted in advance. It is not intended to be used as a processor local bus because the arbitration scheme is not at its best in conveying control information. In addition, the granularity of the SoCs is a design issue. It is not meaningful to use HIBI in connecting small singular peripheral devices.

9.2. Future trends

“Integrated circuits will lead to such wonders as home computers - or at least terminals connected to a central computer - automatic controls for automobiles, and personal portable communications equipment.”[Moo65]

In the consumer electronic field, the most prominent future applications are aimed for portable multimedia devices and other home entertainment electronic systems. In addition, the automobile industry will continue to be one of the biggest consumers of electronic devices. These applications were already predicted in the quote above. A key feature in all this development will be the interoperability and communication between the devices.

The key enabler continues to be the development of the manufacturing processes. The most difficult challenges for this development are caused by the complexity and cost of the fabricated devices. For the past decades architecture developments have been overshadowed by the developments in the manufacturing processes. In the future, much more is required out of the architectures to enable the cost effective design and fabrication of state-of-the-art devices.

The preferred implementations of the above mentioned systems will most likely be based on heterogeneous, configurable, and programmable systems. The traditional ASIC will lose ground to these modifiable system platforms that take full advantage out of re-use by utilizing architectures that can be applied in several applications. The continuing rise of the preferred design abstraction-level as well as other developments of the design methodologies are required in order to design these complex systems.

The interconnection architectures of the heterogeneous systems will evolve in different directions based on the targeted application area. Some interconnections will be tailored for maximum performance regardless of the area and power consumption. For example the architectures of the home
entertainment electronic systems will start to resemble the multiprocessor scientific architectures of the past.

The portable multimedia devices are a different matter altogether. Their interconnection architectures have to be tailored for minimum costs and energy consumption even on the expense of performance. The single bus, which has been the most cost effective interconnection architecture in the past, will evolve into a hierarchy of bus structures and dedicated communication links. In this bottom-up process of building architectures, added features are justified if they increase the performance without rising the costs and energy consumption above an acceptable limit.
10. References


[Sil01] Silicore, *Wishbone system-on-chip (SoC) interconnection architecture for portable IP cores, Revision: B.1*, Silicore Corporation, Corcoran, Minnesota, 2001, p. 87.


