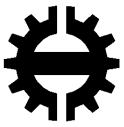


TAMPERE UNIVERSITY OF TECHNOLOGY
Department of Electrical Engineering

TEEMU PITKÄNEN

Experiments of TTA on ASIC Technology

Master of Science Thesis

Subject approved by Department Council
12th may, 2004

Examiners: Prof. Jarmo Takala
Prof. Markku Kivikoski

PREFACE

This MSc thesis was completed in Institute of Digital and Computer Systems of Tampere University of Technology (TUT) in 2002-2005 as a part of the Flexible Design Methods for DSP Systems (FlexDSP) project funded by the National Technology Agency.

I would like to express my sincere gratitude to my thesis supervisor Professor Jarmo Takala for his guidance and valuable tips for the thesis. I would also like to thank Jari Heikkinen, MSc, for always having time and patience to answer all the questions and giving some valuable tips for the thesis. My warm thanks to all my colleagues who have provided the inspiring work athmosphere.

Finally, I wish to thank my family for their support throughout my studies. Most of all, I want to thank my lovely Kati for her love and support.

Tampere, August 17, 2005

Teemu Pitkänen

TABLE OF CONTENTS

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering

Institute of Digital and Computer Systems

Pitkänen, Teemu Oskari: Experiments of TTA on ASIC Technology

Master of Science thesis: 47 pages, 8 appendix pages

Examiners: Prof. Jarmo Takala and Prof. Markku Kivikoski

Funding: The National Technology Agency

August 2005

Keywords: transport triggered architecture, cost estimation, design space exploration, technology characterization

The application specific processor design offers a great solution for performance, and area and energy consumption compared to fixed core design. However the design is found to be challenging and time consuming task, thereby the automated design space exploration has great interest in designing application specific processors. The exploration tool assist designer by foraging for most interesting processor configuration from the design space for the current application. For an effective exploration procedure the exploration tool requires, beside an effective exploration algorithm, fast and accurate enough hardware cost estimation. The cornerstone of accurate hardware cost estimation is the technology characterization.

The MOVE framework is a set of non-commercial software tools, which provide a semi-automatic design for custom processors. The framework utilizes the transport triggered architecture (TTA) programming scheme. In TTA operations occur as a side effect of the explicit data transport, defined in program code, between operational units connected by the interconnection network.

In this thesis the technology characterization for MOVE framework estimator was developed. Technology characterization is based on the component division of TTA and there are four different classes: Functional units, Register files, Interconnection network and Control unit. For each class there are separated characterization parameters. The parameters of characterization are technology independent, which allows the estimation procedure not to be depended on the used technology. The database of costs was created and several estimation results were compared to the results of synthesized processors.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO

Sähkötekniikan koulutusohjelma

Digitaali- ja tietokonetekniikan laitos

Pitkänen, Teemu Oskari: TTA Tutkimusta ASIC Teknologialla

Diplomityö: 47 sivua, 8 liitesivua

Tarkastajat: Prof. Jarmo Takala and Prof. Markku Kivikoski

Rahoitus: Teknologian kehittämiskeskus (TEKES)

Elokuu 2005

Avainsanat: transport triggered architecture, cost estimation, design space exploration, technology characterision

Sovelluskohtaisten suorittimien suunnittelu tarjoaa erinomaisen ratkaisun niin suorituskyvyn kuin pinta-alan ja energian kulutuksen kannalta verrattuna valmiisiin ohjelmoitaviin ratkaisuihin. Tämän lisäksi järjestelmän toiminnan kuvaaminen korkean tason ohjelmointikielellä on nopeampaa ja vähemmän virhealtista kuin perinteinen suoritinsuunnittelu. Tälläisten suorittimien suunnittelun on todettu olevan vaikea ja paljon aikaa kuluttava prosessi, joten automaattinen suunnitteluavaruuden läpikäyminen tarjoaa mielenkiintoisen vaihtoehdon sovelluskohtaisten suorittimien suunnitteluun. Työkalu, jolla voidaan mahdollistaa suunnitteluavaruuden läpikäyminen, auttaa suunnittelijaa löytämään kiinnostavimmat sovelluskohtaiset suoritinkokoonpanot. Jotta suunnitteluavaruuden läpikäyminen olisi tehokasta, tarvitaan tehokkaan algoritmin lisäksi nopea ja riittävän tarkka laitteiston kustannusten arviointityökalu. Laitteiston kustannusten arvioinnin perustana on hyvä teknologian karakterisointi, jolla voidaan mahdollistaa sekä riippumattomuus kohdeteknologiasta että riittävä tarkkuus.

MOVE framework on joukko ei-kaupallisia suunnittelutyökaluja, jotka muodostavat suunnitteluympäristön mukautettujen sovelluskohtaisten suorittimien nopean, osittain automatisoidun suunnittelun. MOVE framework tarjoaa korkealla tasolla kuvatun sovelluksen kääntämisen ja optimoinnin rinnakkaismuotoiseksi, suorittimella ajettavaksi binäärikoodiksi. Lisäksi framework tarjoaa suunnitteluavaruuden läpikäymisen, johon tarvitaan ohjelman suoritukseen kuluneiden kellojaksojen määrä sekä laitteiston kustannukset. Molemmat arvot ovat saatavilla MOVE frameworkin työkaluilla. Laitteiston kustannusten arvionnin tulee olla nopea ja riittävän tarkka. Nopeus on eduksi, kun käydään satoja arkkitehtuureja läpi suunnitteluavaruudesta parhaimman ratkaisun löytämiseksi. Riittävä tarkkuus saavutetaan, kun saadut tulokset ovat niin luotettavia, että niiden pohjalta pystytään tekemään päätös, onko kyseinen arkkitehtuuri parempi kuin edellinen

vai seuraava. Työkalujoukko käyttää transport triggered -suoritinarkkitehtuuria (TTA) suunnittelualustana. Kyseinen suoritinarkkitehtuuri kuuluu niin sanottujen käskytason rinnakkaisuutta hyödyntäviin mikroprosessorityyppeihin. TTA-suorittimissa operaatiot tapahtuvat datasiirtojen oheistuotteena. Datasiirrot, jotka määritellään ohjelmakoodissa, tapahtuvat toimintayksiköiden välillä, jotka on kytketty toisiinsa siirtoverkon välityksellä. Ohjelmointitavasta johtuen ohjelman konekielisen toteutuksen kirjoittaminen käsin on erittäin vaikeaa, mutta siirtojen tarkka ohjelmoitavuus antaa kehittyneelle aikatauluttavalle kääntäjälle mahdollisuuden käyttää kehittyneitä optimointimenetelmiä sovelluksien sisältämän rinnakkaisuuden hyödyntämiselle.

TTA-suoritinarkkitehtuuri on yksinkertainen: se koostuu toimintayksiköitä ja yleiskäyttöisistä rekistereistä, jotka kytketään toisiinsa siirtoverkon kautta. Lisäksi suorittimessa on kontrolliyksikkö, joka toimittaa käskymuistissa olevat datasiirrot toimintayksiköille sekä rekistereille ja purkaa mahdollisesti pakatun käskyn yksiköiden ymmärtämään muotoon. Suorittimessa on mahdollisesti myös käsky- ja datamuisti. Siirrot yksiköiden välillä ovat ohjelmoitavissa, joten kytkentäverkon kytkennät voidaan määrittää sovelluksen vaatimusten mukaisesti. Tällöin saadaan siirtoverkon kapasitanssi laskemaan verrattuna tilanteeseen, jolloin siirtoverkon koko kapasiteetti olisi käytössä. Siirtoverkon kapasiteettia ja toimintayksiköitä voidaan lisätä, kuitenkaan kasvattamatta suorittimen kompleksisuutta exponetiaalisesti, suorittimen suorituskyvyn parantamiseksi.

Tässä diplomityössä on suoritettu teknologian karakterisointi suorittimen kustannuksien arviointia varten TTA-suorittimille, sekä esitelty kustannusten arvioinnin perusteet. Lisäksi työssä esitellään tehon kulutuksen minimoimiseksi kellon portitustekniikka ja suunnitteluvuo TTA-suorittimelle.

Kustannusten arviointi perustuu suorittimen komponenttien pinta-alaan, energian kulutukseen sekä ajoitusinformaatioon. Lisäksi arvioinnin pitää olla käytetystä teknologiasta riippumaton, mikä saavutetaan kuvaamalla fyysistä informaatiota kohdeteknologiasta korkeammalla abstarktiotasolla. Kyseinen kuvausmuodostaa tietokannan karakterisoitujen komponenttien kustannuksista. Kustannustietokantaa kutsutaan costdb-nimikkeellä. Näin ollen kustannusten arviointi on teknologiasta riippumatonta, koska riippuvuus sijaitsee kustannustietokannassa, tai tarkemmin ottaen tietokannan luonnissa, eli toisin sanoen teknologian karakterisoinnissa.

Kellon portitus on keino vähentää kellopolun kuluttamaa tehoa, laskemalla kellopolun kapasitanssia, sekä koko kellopolun aktiivisuutta. Samalla portitus laskee yksiköiden tehonkulutusta poistamalla näiltä turha aktiivisuus. Portituksen perusidea on yksinkertainen, kellopolku katkaistaan, jos polun loppuosaa ei käytetä. Katkaisu tapahtuu ohjaussignaalien avulla, TTA:ssa käytetään kellon portituksen ohjaussignaaleina yksiköiden aktivointisignaaleja. Tällöin ylimääräistä ohjausta ei tarvita ja portitus voidaan tehdä automaattisesti.

Teknologian karakterisointi perustuu kirjaston luomiseen, joka sisältää informaatiota TTA-arkkitehtuurin sisältämien komponenttien pinta-alasta ja energian kulutuksesta. Lisäksi kirjasto sisältää komponenttikohtaista informaatiota, jolla jokainen komponentti voidaan tunnistaa. Komponenttikohtainen informaatio on valittu siten, että karakterisointi onnistuu, vaikka kohde teknologiaa vaihdetaan, esimerkiksi standardisolupohjaisesta FPGA-pohjaiseksi. Tällöin tietenkin tietokanta, johon arvot tallennetaan, on tehtävä uudestaan, jotta se olisi paikkaansapitävä. Karakterisointiperiaatteiden mukaan

voidaan luoda kustannustietokanta ja tietokannasta kustannusarvioija voi hakea jokaisen komponentin kustannukset sekä laskea kokonaispinta-alan ja energian kulutuksen TTA-suorittimelle. Tietokanta luodaan cost-database generator-nimisellä ohjelmalla. Uusi kustannus tietokannan luontiohjelma tehtiin, koska edellisen todettiin olevan vaikeakäyttöinen ja sen teknologian karakterisointiperiaatteet olivat sekä vanhentuneet että osittain vääristävät. Ne loivat kuvan siitä, että arviointiprosessi olisi teknologiasta riippumaton, kun todellisuudessa prosessissa oli teknologialle ominaisia parametrejä, joita siinä ei olisi pitänyt esiintyä.

Jotta kustannusten arviointia ja teknologian karakterisointia pystyttäisiin arvioimaan luotiin MOVE framework-työkaluilla joukko suorittimia kolmelle eri sovellukselle. Suorittimet sijoittuivat eri kohtiin kustannus-suorituskykytasoa. Suorittimien kustannukset arvioitiin kustannusten arvioijalla sekä syntesoitiin 0,13 μm vakiosoluteknologialla, jotta saataisiin referenssitulokset, joihin voidaan verrata arvioijan antamaa tulosta. Tuloksista havaittiin, että arvioijan antamat tulokset ovat riittävän luotettavia, vaikkakin parantamisen varaa on kytkentäverkon ja ohjausyksikön arvioinnissa. Lisäksi arvioijan nopeus on monisatakertainen verrattuna logiikkasynteesiin, jolloin sitä voidaan hyväksikäyttää suunnitteluavaruuden läpikäynnissä.

LIST OF ABBREVIATIONS AND SYMBOLS

A_{xx}	Area of xx
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Set Processor
d	Density
DC	Decode stage
DCT	Discrete Cosine Transform
DSP	Digital Signal Processor or Digital Signal Processing
E_{xx}	Energy of xx
EX	Execute stage
FPGA	Field-Programmable Gate Array
FU	Function Unit or Functional Unit
GPR	General-Purpose Register
HLL	High-Level Language
i	number of
IC	Interconnection network
ID	Identifier
IF	Instruction Fetch stage
ILP	Instruction-Level Parallelism
l	Length of pipeline

MV	Move stage
n_{xx}	Number of xx
P	Power
PC	Program Counter
RA	Return Address
RAM	Random-Access Memory
RF	Register File
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RTL	Register Transfer Level
SAIF	Switching Activity Interchange Format
SIMO	Single Instruction Multiple Operation
SIMT	Single Instruction Multiple Transition
SFU	Special Function Unit
SVTL	Semi Virtual Time Latching
t_{xx}	Time of xx
TTA	Transport Triggered Architecture
TVTL	True Virtual Time Latching
U_{xx}	Utilization of xx
VHDL	Very high speed integrated circuit Hardware Description Language
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration
VTL	Virtual Time Latching

1. INTRODUCTION

The general trend for application specific integrated circuit (ASIC) design is to use high-level language (HLL) for describing the circuit. This calls for analyzing several architecture alternatives. This analysis requires several tasks to be performed: program code for the application has to be generated, performance of the code on each configuration has to be evaluated and implementation costs have to be analyzed. These tasks are almost impossible, and they require a huge effort to be performed manually. The same problem is faced when an existing architecture is customized with application-specific functional units and the effects of the modification are needed.

This problem can be alleviated with tool-assisted design space exploration where the architecture can be varied and a HLL compiler can adapt to the changes in the architecture. In addition, the estimates of the cost of the application executed on the architecture, e.g., execution time, area, and energy, should be obtained. If hundreds of different architectures are to be analyzed, it is essential that the estimations can be obtained quickly and accurate enough for design purpose.

In this thesis the technology characterization for Transport triggered architecture (TTA) is performed and a methology for estimation area and energy consumption on a specific processor configuration is proposed. The proposed methology is technology independent which allows the usage of estimation procedure when the target technology changes. The technology characterization results are stored into a database of costs, which is used for evaluating the processor costs, in means of area and energy. The estimates given by proposed methology are compared to the values obtained from the reference designs and the results show that the accuracy of the method is sufficient for exploration process and the estimation is very rapid compared to the traditional methods. Secondly this thesis presents the design flow of a TTA processor.

The organization of this thesis is as follows. Chapter 2 describes the TTA and the development of TTA from very long instruction word (VLIW) architecture. The main concentration is at hardware aspects of TTA. In chapter 2 the MOVE framework is introduced following by introductions of some of the framework tools. In chapter 3

the characterization of TTA processor components are described and the detailed information about characterization constraints are presented. In chapter 4 the Estimation procedure of TTA processor is described, and the calculation of the output statistics of Estimator is presented. The chapter 5 presents the design flow of TTA processor. Secondly the clock gating is experimented and finally several architectures are evaluated to verifying the correctness and accuracy off the estimation procedure.

2. TRANSPORT TRIGGERED ARCHITECTURES

Performance improvement is important for the latest microprocessors. A very good method to gain more performance is to exploit instruction level parallelism (ILP). ILP detection can be done by two ways: runtime or compile time.

In runtime detection, the processor hardware detects and resolves the dependencies between the operations in sequential instruction stream at runtime. Such detection is done by superscalar processors, e.g., Intel Pentium.

The compile time ILP detection is done by a compiler. It translates sequential instruction code to parallel instruction code, which is fed to the processor. Compile time detection is done in the very long instruction word (VLIW) and the transport triggered Architecture (TTA) processors. Superscalar processors are binary compability with previous architecture generation, but suffer from hardware complexity and long design cycle compared to VLIW or TTA. That makes TTA's and VLIW's attractive alternatives for embedded systems.

TTA can be viewed as a superset of VLIW, which is a superset of RISC's (a RISC can be viewed as a VLIW with only one functional unit) [1]. Transport triggered architectures can be specified as single instruction multiple transition (SIMT), and VLIW can be specified as single instruction multiple operations (SIMO).

Section 2.1 introduces the development of TTA concept. Section 2.2 describes hardware aspects of TTA. Section 2.3 introduces MOVE framework, an automated tool for TTA processor generation.

2.1 Development of TTA

TTA architecture was developed from VLIW architecture. The main difference is in the way, how operations are programmed and executed. In VLIWs, instructions specify RISC type operations, while in TTAs instructions specify data transports. Due to this TTA is also called the MOVE architecture. Operations performed to data are implicitly

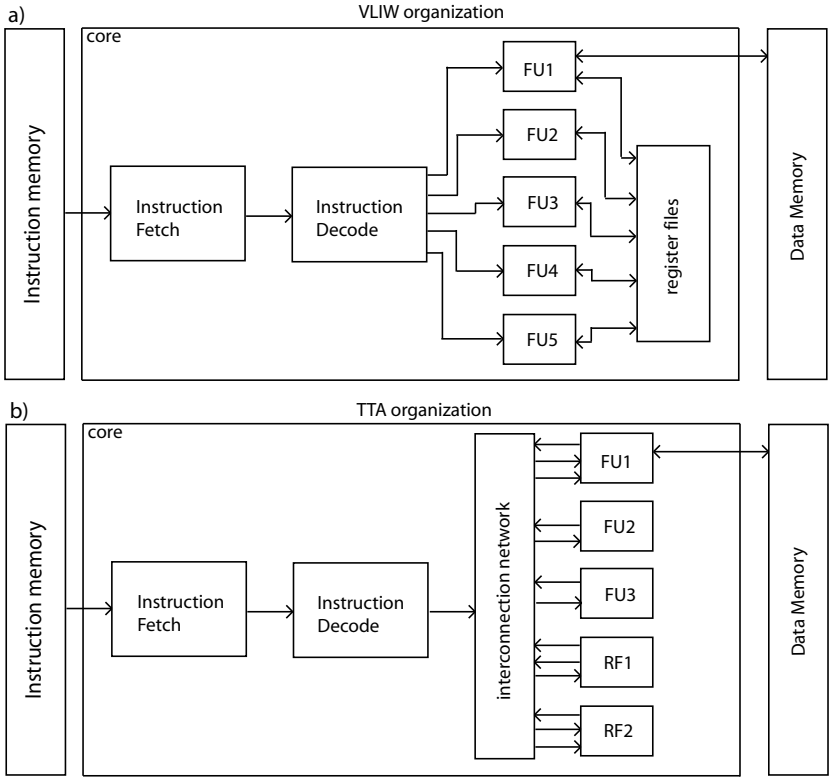


Figure 1. *Organization of a) VLIW and b) TTA.*

specified with the destination of transport.

TTA was developed to reduce the complexity of VLIW by placing the register traffic under program control. In other words the data transports become visible at the architectural level and they can be controlled and optimized by the compiler. TTAs are organized as a set of functional units (FUs) and register files (RFs) which are connected by an interconnection network (IC). Organization of VLIW and TTA are shown in Fig. 1.

2.2 Hardware Aspects

TTA processors consist of FUs and RFs, which are connected to each other with interconnection network. An example of FUs is shown in Fig. 2. Register files can be considered as a special kind of functional units because they use the same interface for connections to the interconnection network.

FUs can have arbitrary functionality and so they are completely independent from each other and that is why they can be designed separately. FUs can be pipelined for different stages. Processing different type of functionality can take different amount of clock cycles. TTA hardware generation process can be automated and different kind of

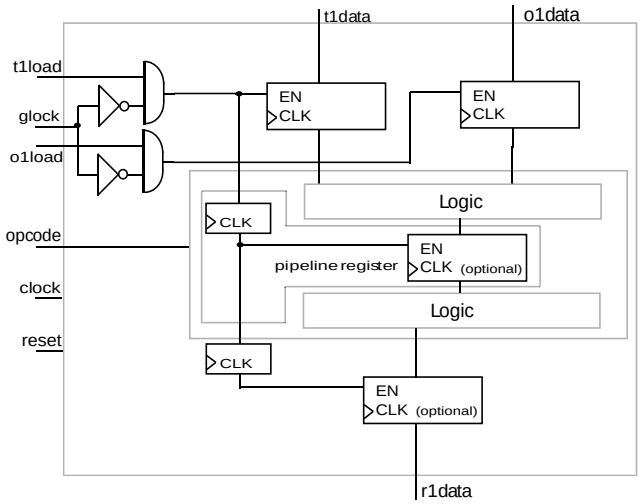


Figure 2. *Functional unit.*

processors can be easily configured by assembling different combinations of functional units.

2.2.1 Functional Units and Register Files

FUs are responsible for performing operations on data. FUs read data from an input socket, and when the operation is completed, modified data can be read from the result socket of the FU. Usually in TTA, there are at least three types of FUs: instruction fetch unit, load-store unit, and guard generation unit. The instruction fetch unit reads instructions from memory and controls the flow of program. A guard generation unit generates guard bit for the processor and load-store unit provides an access to data memory where variables with long lifetime can be stored. Functional unit has at least one input and one or more outputs. Inputs of FU are always registered. An input register is either an operand or trigger register. Basically trigger and operand register are similar: both provide storage for input data, but there is an important difference: when data is fed to trigger register it initiates a new operation. If FU supports multiple operations an operation code (opcode), which selects what operation is performed, have to be received concurrently with data to be latched to trigger register from socket. Pipelining can be considerable subject when execution of operation takes more than one machine clock cycle. Each FU can be pipelined independently. Hybrid and Virtual time latching (VTL) are the most common latching methods, and they are described in [1] with some other pipeline control disciplines.

In this thesis, semi virtual time latching (SVTL) is used. The SVTL allows operands

to be fed to the unit before the trigger input, which starts a new operation. It makes scheduling of the operation easier and gives some scheduling freedom but there has to be shadow registers in operand input and that is more complex than True virtual time latching (TVTL). In the TVTL, all the operands have to be fed at the same clock cycle and all the operands start the new operation.

TTA microprocessors require general purpose registers (GRP), which store the intermediate values with short lifetime. One or more Register file (RF) contains GRPs. RFs communicate with others units like a FU, via interconnection network.

2.2.2 Interconnection Network

Interconnection network (IC) provides a path where FUs and RFs can exchange data. In IC there are two simple modules: sockets and busses. Sockets feed data from busses to FU/RF and they feed data from FU/RF to busses. Beside, the normal functionality, to provide data transport capability, busses also distribute signals that control the transports. Controlling signals are source and destination IDs and processor locking signals.

Sockets provide connectivity between busses and RFs or FUs. There are two kinds of sockets: input and output. Input sockets move value of one of the bus to the destination and they basically are multiplexers. Input sockets are connected to one or more bus and at least one RF or FU. Output sockets move value of source register to one or more busses. Example of both kinds of sockets is in Appendix A. Destination and source of moves are defined in instruction word and these are described in chapter 2.2.4.

2.2.3 Transport Pipelining

Apart from functional unit pipelining described in section 2.2.1, the instruction execution can be pipelined in TTAs. This is called transport pipelining. Typically transport is pipelined using three stage mechanism, which consists of instruction fetch, decode, and move stages. Decode and move stages can be combined for two stage pipelining.

In the instruction fetch stage, the next instruction is read from the instruction memory or cache. In the decode stage, the source and the destination fields for the sockets are decoded from the instruction word and transported to sockets. Feeding the information of the source and the destination activates the data transports to FUs. In the move stage, the actual data transport takes place. Data is copied from output of a FU/RF to input register of another FU/RF.

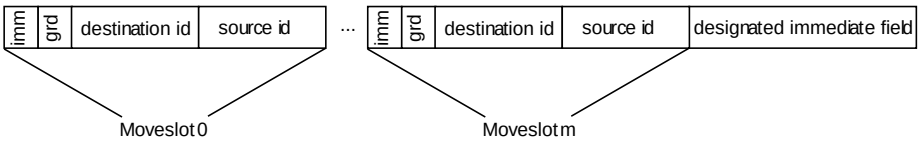


Figure 3. *Instruction word format. grd: guard, imm: immediate, and id: identifier.*

2.2.4 Instruction Format

The instruction word format in a TTA processor, depicted in Fig. 3, typically consists of as many fields as processor contains busses and one, the amount is not restricted, optional field for a long immediate. Each field specifies an independent concurrent data transport from a source to a destination.

Each field, except immediate field, consist of a guard identifier (ID), destination ID, and source ID as shown in the Fig. 3. The guard ID is used by the guard unit of TTA processor to control the execution or squash the move operation. Destination ID is used to select an input socket where the data is to be sent. The source ID is used to select an output socket as a source from where the data in transported to the destination. Destination and source IDs contain a socket address and an optional opcode. The socket address is used to select the socket and the opcode is sent in the move stage to the FU to select operation which the FU to be performed is to the input data.

In TTA, each field of the instruction word can be used to represent immediate value extensions. These extensions are used to construct long immediate. In such a case, whole field is used to represent the long immediate. For the controlling when the field is used for the long immediate extension, a dedicated long immediate extension tag exists at the beginning of the instruction. The tag specifies in each instruction, which of the moves is used for normal data transport and which for long immediate extension. Instruction word can have a dedicated field for long immediate bits even if the immediate extension is used. This field is only used to construct a long immediate value.

For shorter immediate values the value of the source opcode can be used to transport the short immediate bits. The length of a short immediate is restricted to the size of the opcode field of the source ID.

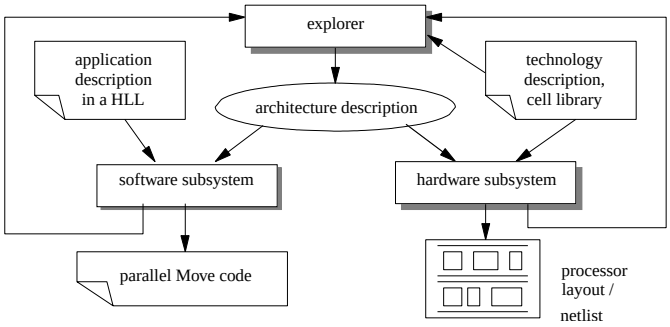


Figure 4. *General organization of the MOVE framework.*

2.3 MOVE Framework

The MOVE Framework is a set of software tools that can automate the design of Application specific instruction set processors (ASIP). It has been developed in Delft University of Technology, with some further development at Tampere University of Technology. Processors designed with MOVE framework are based on TTA. The scalability of this architecture allows processor configuration to be optimized for a selected application. Furthermore, the flexibility and simplicity of the TTA allows function units specific to the given application to be easily integrated with the framework to enhance performance.

The basic idea of MOVE framework is that the user described the desired functionality in a high level language, C code. Next step is that user tells what recourses are available in machine description file, which is a complete textual specification of the target processor architecture. Machine description file contains a set of architectural parameters. These parameters fully describe the essential characteristics of the target processor, such as number and type of FUs, and number of transport busses, etc. Now MOVE design space explorer can find the best possible cost/performance ratio with the aid of MOVE scheduler, simulator, and estimator. After exploration user selects best available processor configuration and generates automatically, with MOVE processor generator, hardware description language description, for example VHDL, of target processor. The organisation of MOVE framework is shown at Fig. 4.

Design space exploration is composed of two separate tasks: recourse optimization and connectivity optimization. Recourse optimization consists of the finding the right match of resources. The goal of resource optimization is to reduce costs without a performance lost. Connectivity optimization determines the connectivity between busses and sockets. The resulting connectivity is based on the communication requirements between the

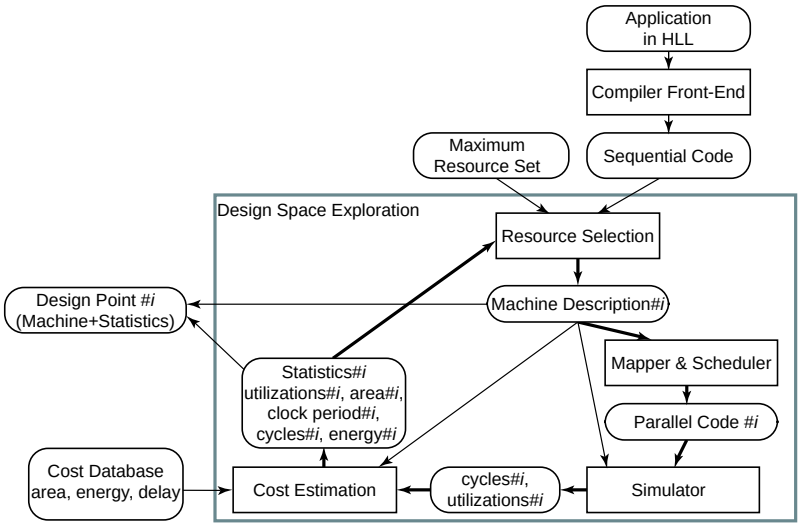


Figure 5. *Role of Estimator in exploration procedure*

FUs and the RFs. The goal of the connectivity optimization is to reduce the bus load, and thus the cycle time. For each configuration the scheduler, simulator, and estimator is called to obtain the cycle count of the code on target machine and the estimation of processor cost such as area consumed in silicon and energy consumption.

The cost estimator has an important role in the design space exploration, illustrated in Fig 5. As the figure shows, the sequential code, maximum resource set and cost database is fed to design space exploration. In the exploration procedure the processor configuration are evaluated after each modification, i.e., to know if the modification should be applied or not. Tools to be used to evaluate processor configuration are cost estimator, scheduler, and simulator. From the scheduler and simulator the cycle count and activity of components are fed with the cost database to the cost estimator. As an output the estimator provides an evaluation of target processor in terms on silicon area, energy consumption and timing. At the first round of exploration process the maximum set of resources is used.

The estimator considers FUs, RFs, buses, input and output sockets, and the control logic in the estimation process. Control logic consisting beside of control signals, registers for program counter, long and short immediate, return address for jumps and instruction word. Memories are excluded from the processor evaluation.

The processor generator is a software tool that performs a transformation from one source hardware description language to another. In this case, the transformation means converting the description presented in internal format of MOVE framework to a generic standardized hardware description language, e.g., VHDL, which represent the processor

design in a format that is not dependent on the MOVE framework.

The processor generator (MOVEgen) is stand alone processor generator for MOVE framework written in python. It is command line driven, script"-like tool that obtains name of two files, which are descriptions of target processor and required interconnection bus structure as command line parameter. Additional information, which is needed for generation of VHDL, given in files *external_interface* and *fu_conf*. The file *external_interface* contains information about connections outside of the processor core, e.g., to the data memory. The file *fu_conf* is containing information about the input and output width's of functional units and information of unit's connections to outside of processor core. Examples of both files are presented in Appendix B and Appendix C.

MOVEgen generates VHDL description of a target processor according to parameters given by the user. The produced VHDL code can be simulated, verified and synthesised with aid of the standard computer aided design tools that accept VHDL as entry language, e.g., Modeltech Modelsim and Synopsys Design compiler [2],[3]. The creation of processor core and simulation of processor are presented in 5.2.

3. TECHNOLOGY CHARACTERIZATION

In this chapter, the principles needed for technology characterization for application specific integrated circuit (ASIC) are described. Here a standard cell technology is characterized for needs of TTA architecture, which uses four different building blocks, as shown in Fig. 1, functional units, internal registers, interconnection network and control unit, which contains the instruction fetch and instruction decode units. The proposed technology characterization is independent of the target technology, i.e., no matter what the target technology is the characterization principles can be applied. The area characterization principles can be used straight for needs of FPGA, but the energy characterization will have some problems with it.

The basic principle is to build a library that contains information about area cost and energy consumption of components, which are needed in generating costs of all components, and component specific information, which allows their identification.

All components except control are modelled by delay, which describes how fast the current component is. In other words, if component is between registers, delay tells how high clock frequency can be in order to get right costs of selected component. If component is not between two registers, there has to be a chain of delay of components, until the chain of components is between two registers. Now the sum of each component's delay in component chain describes the highest possible clock frequency. Fig. 6 (a) illustrates FU's and the interconnection network between them. There are 10 different chains of delay. The interconnection network and the presence of functional unit operation logic, i.e., there is no register at output of FU0, is presented in Fig. 6 (b). The outputs of FU1 are registered, as well all the inputs, resulting delay chain of interconnection network. As the output of FU0 is not registered, which results a situation, where the delay of the FU0 has to be summed to the delay of interconnection network. The delay of interconnection network is sum of throughput delays of output socket, input socket and bus.

The area cost figure is determined in equivalent gates and the unit of energy cost figure is Joules per usage, except the static energy cost figure and the control energy cost figure

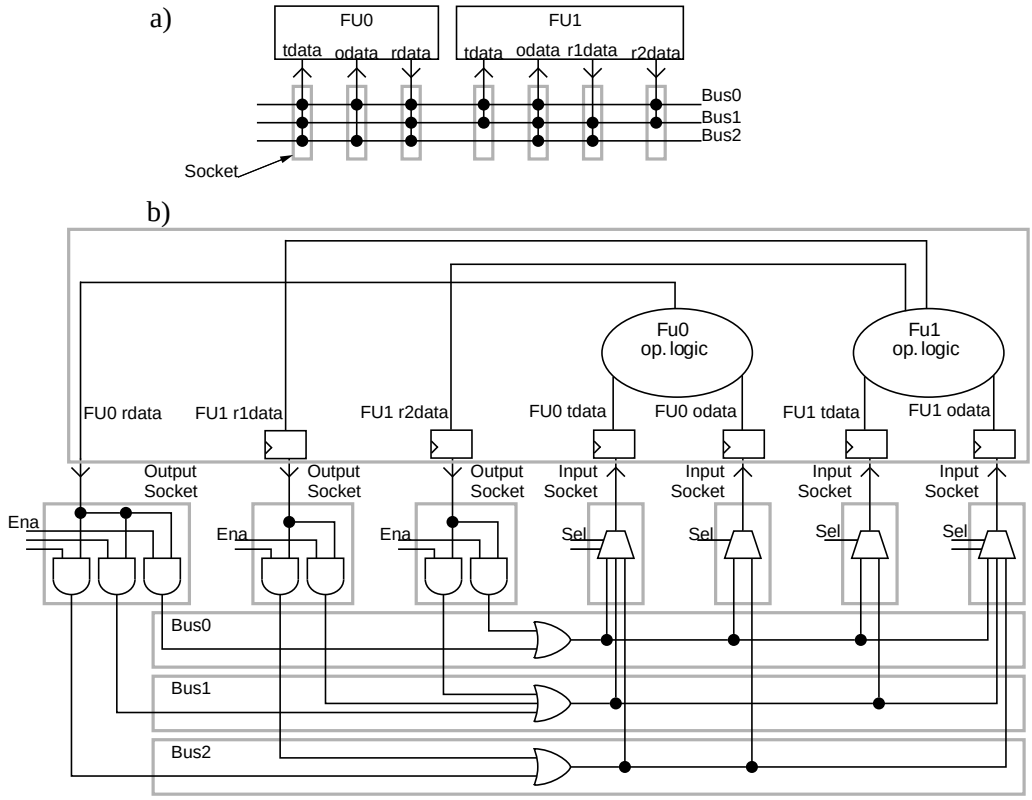


Figure 6. *a) Connection diagram of FU0 and FU1 b) data path of and-or bus between FU0 and FU1.*

which are given in Joules.

The static energy cost figure is calculated for all components as follows

$$E_{Static} = Pt_{delay} \tag{1}$$

where E_{Static} is static energy cost figure, P is the leakage power acquired from power compiler, t_{delay} is component operational delay. The static energy is usually small and it can be left out from calculations of costs.

Following chapters describes how the library of each component is built and how costs can be calculated from library information.

3.1 Function Unit

Functional unit is modelled by its *operation set*, *pipeline*, *latency*, *data width*, *number of input ports*, *number of output ports*, and *delay*: operational and output. If there is a possibility to have same model with different implementation of same functional unit,

Table 1. *Match types for functional units.*

Parameter	**Match type**
operation set	exact
pipelining	exact
latency	exact
delay	exact, subset
data width	exact, interpolation
number of ports	exact
implementation	exact

there is a possibility to use *implementation* as well as parameter in modelling. With these parameter sets all functional units can be modelled separately. The match types supported for FU parameters are presented in Table 1. Definitions of match types are presented in Table 5.

3.1.1 Characterization Parameters

The *operation set* contains all the operations supported by functional unit. For example, functional unit which performs addition, subtraction, and addition and subtraction simultaneously has an operation set of add, sub and addsub(add_sub_addsub).

Pipeline is describes the pipelining behaviour of unit, the number of pipeline stages, by terms pipeline control method and operation pipelining or operational pipelining. Pipelining behaviour is somewhat bound to parameter latency. Term none for pipeline control method is describing situation in which the pipeline does not occur. Usually the FU contains registered input, depending on parameter latency. Pipeline control method SVTL is describing case where unit is SVTL pipelined, presented in 2.2.1, i.e., the FU has registered inputs and outputs. The SVTL pipelining is describing the minimum latency of 2. Operation pipelining consist the number of total pipeline stages and the information on which operation in operation set is using specified pipelining stage.

Latency describes the total amount of clock cycles, which are needed for FU to perform the operation. Latency is somewhat bound to pipeline parameter. When latency is 1 the pipeline control method parameter has to be none and the latency parameter describes

the presence of input register. If latency is 0, i.e., FU is an asynchronous design, the pipelining behaviour parameter is also none and FU has no registered input or output.

data width describes the width of inputs and outputs. If only one value is presented the width of all inputs and outputs are the same. When output width is different from input or when the width of input or output of unit differs from each other, all the input and output widths have to be described separately.

Number of input and output ports describes how many input and output ports are present in FU.

delay consist two fields: operation delay, and output delay. The operation delay describes how much time an operation or a pipeline stage in functional unit takes; it is always either the slowest operation in unit's operation set or the longest pipeline stage. If the pipelining behaviour of a unit is SVTL or SVTL and operation pipelining, the operation delay describes straight how high clock period can be used with selected unit. The output delay describes the delay from the last register to the unit output. If output delay is not zero, the delay must be taken into account in generating the delay chain. The output delay is somewhat bound to *operational delay* and *pipelining*. When pipelining is zero the output delay is same as operational delay, because there is a register only in the input of the unit. When pipelining is set to SVTL or SVTL and operation pipelining the output delay is zero, the last element of the component is a register. When pipelining is set to operation pipelining the output delay describes the time, which the functional unit takes from the delay chain, and this parameter defines the highest clock frequency. The clock frequency is depending on other components which are connected to the output of a functional unit. Delay chain is complete when all the components in chain are between two registers or one registered loop. If there is a chain which does not contain registers at the beginning and at the end of chain, the design is asynchronous.

Implementation is optional. If there is no implementation parameter all sufficient parameter sets uses the same implementation. Implementation parameter is simply a string which separates different implementations. For example FU with operation set add, pipelining SVTL, latency two, data width 32, and two different implementations carry look ahead adder and ripple carry adder, the implementation parameters, for example CLA and RCA, have to be used. If implementation parameter is not used in the previous case, the modelling of FU may be incorrect, and costs of FU are not reliable.

Table 2. *Match types for register files.*

Parameter	**Match type**
read and write ports	exact
number of registers	exact
delay	exact, subset
latency	exact
data width	exact, interpolation
implementation	exact

3.1.2 Characterization

Area of unit is acquired from synthesis of unit. Energy dissipation is acquired combining of synthesis and gate level simulation. For gate level simulation, the activity of input ports is random, i.e., there is 50 percent possibility to each input port bit to change. Load signals are active when input changes, except for idle operation, where the load signals remains zero for all the time. Energy for each operation in the operation set is acquired separately, by setting operation to select the opcode corresponding to the wanted operation. The energy per usage is acquired from power given by power compiler [3], as following

$$E_{cost} = Pt_{clk}l_{pipeline} \tag{2}$$

Where E_{cost} is the energy cost figure, P is the power acquired from power compiler, t_{clk} is clock period, and $\mathrm{l}_{pipeline}$ is the length of the operational pipeline of the operation.

3.2 Register File

Register file is modelled by the *number of read ports* and the *number of write ports*, the *number of registers*, *data width*, *latency*, and *delay*: operation, input, and output. There is possible to use an optional parameter, which describes the *implementation* of register file. The match types supported for RF parameters are presented in Table 2, and the match types are presented in Table 5.

3.2.1 Characterization Parameters

Number of *read ports* describes how many read ports are available and how many simultaneous reads can be performed. The number of *write ports* describes how many write ports are present and how many parallel writes can be performed.

The *number of registers* describes how many register slots is available in register file, i.e., how many values RF can hold before older values are overwritten.

The *latency* describes the delay in clock cycles in which for the data written to the register can be read from the register.

Parameter *delay* consist three fields, input delay, operation delay and output delay. Input and output delay are treated similarly to the *output delay* presented in 3.1. The input delay is describing the time consumed before first register and the output delay is describing the time after last register, which have to take account in forming of delay chains. The operation delay is describing the time between registers in register file; the operation delay can be zero, if register file contains only one register in chain.

datawith is the word width of registers.

3.2.2 Characterization

Area of RF is acquired from synthesis of RF. The operations of RF is obtained by combining all possible simultaneous reads from RF and writes to RF, i.e., for RF with one read port and two write ports with capability to work parallel, the number of possible operations are six: 0:0, 0:1, 0:2, 1:0, 1:1, and 1:2. Energy dissipation is acquired combining of synthesis and gate level simulation. For gate level simulation, the activity of the write ports is random, i.e., there is 50 percent possibility to each write port bit to change. Load signal of specific operation is active when input changes, except when RF is not reading or writing, where the load signals remains zero. Energy for each operation is acquired separately, by setting the load signals to the corresponding operation. The energy per usage is acquired from power given by power compiler [3], as follows

$$E_{cost} = Pt_{clk} \tag{3}$$

Where E_{cost} is energy cost figure, P is the power acquired from power compiler, and t_{clk} is clock cycle of the synthesized component.

Table 3. *Match types for interconnection network.*

Parameter	**Match type**
fanin	exact, interpolation
fanout	exact, interpolation
delay	exact, subset
data width	exact, interpolation
implementation	exact

3.3 Interconnection Network

As described in chapter 2.2.2 the interconnection network consists of three different types of elements: input socket, output socket and bus. The characterization is done with three bus types: and-or, tristate, and mux busses. In each type, there are some special features: all three bus type haves input socket, output socket exists at and-or and tristate buses, and bus exists at and-or and mux buses. For tristate bus, the output socket contains bus element and for mux bus the element bus contains output socket. For the estimation procedure all three elements are present, but for cases where only two elements exist, the third element costs are zero. Because of difference in element numbers, depending on bus type, all three types of elements have to be treated similarly. Only one type of bus can be used in characterization at the same time. The match types supported for IC parameters are presented in Table 3, match types are presented in Table 5.

As mentioned the interconnection network consists of three types of components: input socket, output socket and bus, the Fig. 6 (b) represents the logic diagram and principle structure of an and-or bus, which is mostly used in our approaches. The input socket and output socket contains multiplexers and and-gates, respectively. The actual bus contains an or-gate.

3.3.1 Characterization Parameters

Interconnection elements are modelled by *fanin*, *fanout*, *datawidth*, and *delay*: control and throughput. There is possibility to use an optional parameter which describes the *implementation* in interconnection element.

Fanin describes the number on inputs of IC element and *fanout* describes number of outputs and the load which component is driving. Usage of fanout is depending on what is next to component. For example, if input socket is driving a trigger port of FU and a trigger port of RF with register size of 4, the fanout in calculation is 5, because FU trigger port behaves like load 1 and RF with size of 4 like behaves like load of 4. One socket port is calculated of load of 1.

Data width describes the data width of inputs and outputs of component. The data width is always same for all components, if the data width for inputs or outputs varies for component which costs wanted to find, the costs can be found from characterized components by combining them, presented in Fig. 7. All of the combinations can be found in such way. If the input width is smaller the extra output bits are generated, either sign extension or extension, and if the output is smaller the extra bit in input are just ignored, i.e., if the input and output widths differ the smallest is those is taken into account. The sign extension or extension logic costs are ignored, because they are insignificantly small. For components which contains different data width of fanin, fanout, or both ports. The component have to divide to smaller pieces to get all possible fanin, fanout, and datawidth possibilities. At Fig. 7 (a) is input socket with fanin 6 and fanout 3, the fanout is describing the number of output ports. The final fanout is depending on the components connected to socket. The data widths of input ports are 4, 8, 16, 16, 32, and 64 bits and the data widths of output ports are 32, 16, and 16. Now is needed to divide the input socket to smaller pieces, number of needed pieces are four, as shown in Fig. 7 (b). The first subcomponent has fanin 6, fanout 3 and data width 4 and the second has fanin 5, fanout 3, and data width 4. The third has fanin 4, fanout 3, and data width 16 and the fourth has fanin 2, fanout 1, and data width 16.

Delay describes the component's throughput delay and control delay. To get the total delay of total interconnection chain, all the three throughput delays must be summed: the output socket delay, the bus delay, and the input socket delay. The component control delay describes the time the control of component takes, for our design the control delay is necessary for input socket, because the input socket control is ready at the beginning of clock cycle and the data arrives from bus much later, leading to a situation where throughput delay has to be much smaller than control delay. The control delay has to be smaller than the final clock cycle of the processor. When the component is divided to subcomponents, the throughput delay is approximated by the largest possible subcomponent and for the control delay is the actual component largest possible control delay, as in Fig. 7 (a), the control delay approximation would be for the

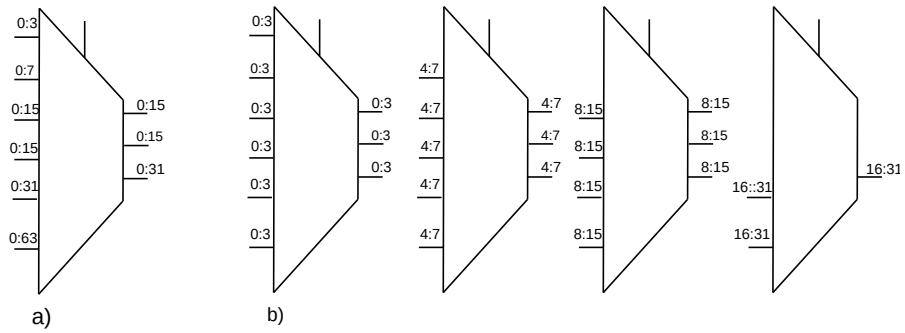


Figure 7. *Input socket: a)component and b)subcomponents.*

component control delay, which have datawidth 32, fanin 6 and, fanout 3.

3.3.2 Characterization

Area cost figure is generated by synthesizing IC elements. Required area of IC element is acquired from area cost figure of element. When element is divided to smaller pieces the required area of element is acquired from the sum of area cost figures of sub elements.

For each component there are 2 energy cost figures; active energy and idle energy. The energy cost figures are obtained similarly as in equation 3. For active energy gate level simulation, the activity of input ports is weighted towards changing, i.e., there is 70 percent possibility to each input port bit to change. The percent value is acquired from simulations of test cases. For idle energy the component is simulated to be at state when no control part is moving. For the and or-bus type the output socket control is set for not to select anything, leading to outputting zero, in idle state, and for the bus the input, when component is idling , is zero leading to outputting zero. For the input socket the activities of the inputs are slightly reduced, the possibility to change is 55 percent, because when component is idling, the previous component might also be in the idle state. For Three state- and MUX bus types the output socket and bus write is combined. For the Three state bus type the output, when idling, is floating in high impedance state. For the MUX bus type the idling output is zero. The bus technology defines the way the energy cost figure is generated.

3.4 Control Characterization

The used model for control characterising is fairly simple. It takes account the number of needed instruction registers and scales the number of registers to level of connections of interconnection network.

Control logic is modelled by *connectivity* and *time*.

The *connectivity* parameter describes the relative connectivity of the interconnection network. It is computed as a ratio between the numbers of connections in the interconnection network, considered divided by number of connections in the fully connected version of the same interconnection network.

time describes the clock period used in synthesis of characterization process. It is used to calculate the energy value correctly.

Area cost figure is area of one register element in the control block for the given connectivity in equivalent gates. The area cost figure is obtained as follows

$$A_{reg} = \frac{A}{n_{regs}} \tag{4}$$

where A_{reg} is area cost figure of one register, A is the area given by design compiler, and n_{regs} is the number of registers in control unit.

The energy cost figure is energy dissipation of one register element in the control block for given connectivity. The energy cost figure is acquired from synthesis of architecture as follows

$$E_{reg} = \frac{Pt_{clk}}{n_{regs}} \tag{5}$$

where E_{reg} is energy cost figure of one register, P is the power given by power compiler, t_{clk} is clock cycle in which the component was synthesized, and n_{regs} is the number of registers in control unit. The effect of the leakage energy is left out from the control unit characterization.

3.5 Generation of Cost Figure Database

Principles presented above, the cost figures can be generated and combined in form of a database of costs (costdb). From costdb the estimator can search for cost of components and calculate the total cost of area and energy for TTA processor. The costdb is the

heart of accurate estimation. In the database lies all the information about the costs of TTA components, i.e., FUs, RFs, buses, input and output sockets, and control logic. An example of a part of costdb in presented in Appendix D.

For costdb there have to be all components that are used in the estimation procedure of processor, because if a component is not found, or cannot use subset or superset of other component, the cost of processor is not valid anymore. Only exception is when user wants to evaluate performance of processor regarding of the costs, there can be used such components which not have an entry in costdb. That is why the cost database is usually large, but it is needed to generate only once and then add new components, if necessary. Generation of cost database is based on combinations on usage of Synopsys Design compiler and Modeltech Modelsim. Modelsim is used to achieve gate level activities of desired operation of desired component.

The cost database generator is a program, which creates the costdb. Generator reads VHDL package files and characterizes file contents with principles presented above. The VHDL package files, for FU, RF, IC, special function units (SFU), and control unit, contains every component entities which ought to be in database. The SFU package contains every user specific units, which the user wants to use in TTA processor.

When the generator is started, it checks, if any of the VHDL package files has been changed. If no package has been changed the generator evaluates if any of the components in the package are changed, if none is changed the generator does not do anything. When some of the files have been changed the generator characterizes the changed components. If an existing component is changed the generator replaces old results in database with new ones. If there is a need to add component to database, it can be done in three ways with command line parameter presented in Table 4; Component is characterized separately, or added to existing package, or create new package for component.

For user point of view the generation procedure is very simple; user only starts the generator, which creates the entire database. The file structure has to be set correctly with the command line parameters of database generator. Generator accept command line parameters presented in Table 4.

Help prints out the usage help of program and the create parameter starts the creation of database.

package creates the package list and it can be used to [-a] add or [-r] remove packages from package list, or to create cost figures of one specific package. The package type option is used to define which type the package is: FU, SFU, RF, input socket, output

Table 4. *Parameters of database of costs Generator.*

Parameter	**Option**
-pack	[-a][-r][name of package][package type][location of package]
-comp	[name of file][entity1,entity2,...,entityN][component type] [location of file][-a][-r][name of package]
-loc	[-temp][location][-src][-a][-r][-d][location][-pac][-d][location]
-act	[-act][type][input activity][-idle][type][input activity]
-help	
-create	

socket, bus, or control. The information of package type is needed to place generated cost figures to right place in database. The location of package is optional, when the location is not given the default package location is used.

component is used to [-a] add or [-r] remove components from packages, or to create cost figure of on specific component. The name of VHDL file and the name of entity or entities must be given. If the component contains multiple entities the top level entity must be given first. Component type is dealt similarly as package type. The location of component is optional, if it is given, it is stored to source location list. If location is not given, component is searched from source locations. If component is added to package which does not exists, the package will be created to the default package location.

location creates the location list, where the temporary location, locations of source files, and location of package list is saved. At the location list there is possibility to [-a] add or [-r] remove source search locations, and set the default location for source. With the [-d] parameter the default location of source list can be changed. If the default location is not set, the generator uses the location where the generation is started as the default location for source location list. Option [-temp] is used to modify the temporary location, in which all temporary files generated by generator is stored. With option [-pac] the location of package list can be modified, [-pac] [-d] option is used to change the default location where packages are. In both cases, the default location is same folder where the generator is started.

activity is describing the input bit wise probability, in percents, to change for active and idle sequence. If option type is describing which type, i.e., FU, the activity parameter

is used for. If the type is not given the activity is used for all components. The default activity is 50.

create is used to start the generation procedure. The generator generates all changed components in package list, if not specific component or entity is not wished to generate with *package* or *component*

It is possible to edit every list by hand, but it is not recommended. Any parameter by them self, except create and help, prints out the parameters and possible lists containing parameter specific information.

4. ESTIMATION FOR TTAS

Automated design space exploration is one of the most interesting tool for designers in the area of customizable processors. By trawling through the design space and notifying the most interesting target processor configurations, exploration tool assist the designer to find the most suitable resources for the best configuration. Cornerstone of exploration tool, in addition to effective exploration algorithm, is the hardware cost estimator, which has to be fast, accurate enough, and technology independent [4].

The estimation on hardware costs procedure is based on information on area, energy and timing statistics of the hardware resources. First, each hardware resource has to be characterized on a given technology. After that, the area and energy of a target processor for the certain timing statistic can be evaluated based on this information.

The estimation procedure requires the information of the technology, which is used to implement the target processor. And thus, the estimator is needed to be technology independent. A description at a higher abstraction level is used to provide the physical information of the target technology. This description is structured as a database of costs of characterized components, thus called the cost database. Now the estimator can be technology independent, because the information about used technology lies on cost database, or to be exact in the creation of cost database. Now the used technology can be easily changed.

Estimation is based on data queries from cost database; each query contains a search key for hardware components. Each hardware component is specified by some of properties described in chapter 3. The characterization parameters work as search keys for estimation procedure, because the parameters identify the component. All the parameters are not needed in search, but all of them are used to select the right component in each case. Estimator searches through database for the matching component. When the perfect match is not found, the statistics of other components can be utilized in the evaluation, if the behaviour of the resource supports that. There are four different match types: Exact, superset, subset or interpolation match. The match types are presented in Table 5. In chapter 3 the supported match types for the properties of components

Table 5. *Supported match types.*

Match type	**Explanation**
Exact match	An equal characteristics is found from database
Superset	Greater characteristics or a superset is found from database
Subset	Smaller characteristics or a subset is found from database
Interpolation	Smaller and greater characteristics is found from the database. Linear approximation is used for calculating new statistics for the new database entry

are described. For three first match types the cost figure values of components are not changed, but for the interpolation match the cost figures are computed for a new entry, but the database will not be changed at any situation.

For the correct data path of components the estimator must make sure the component data path has register at the beginning and at the end of path, i.e., component is between two registers. If the component is not between two registers, the next component should be added to data path as long as the chain of components is registered. The total delay of chain of component is called chain delay, presented in Fig. 6, which describes the total delay of components in delay chain. The chain of components is created by minimizing the costs of chain, area, and energy. When the total processor is evaluated, the Estimator finds the critical path, the longest delay or delay chain, and slows down other components or chain of components. The slowing down is performed when costs of component or components are lower. The maximum amount of delay is the cycle time, given by user or design space Explorer.

4.1 Estimation Procedure

Total area of the target processor configuration is obtained by the sum of area of each hardware component as

$$A = \sum A_{FU} + \sum A_{RF} + \sum A_{IC} + A_{CNTRL}. \quad (6)$$

The energy consumption of target processor is obtained by summing up energies of each hardware component as

$$E = \sum E_{FU} + \sum E_{RF} + \sum E_{IC} + E_{CNTRL}. \quad (7)$$

Table 6. *Calculation of FU energy.*

Operation	**Usage**	**E_{OP} pJ**	**E nJ**
Add	7716	9.228	71.20
Sub	4352	9.799	42.65
Idle	8219	0.3552	2.92
Static	20278	0.82 aJ	0.02 pJ
Total	20278		116.77

Parameter	**value**	**Parameter**	**value**
operation set	add_sub	pipeline	SVTL
latency	2	operational delay	8.02
data width	32	input port	2
output port	1	output delay	0
clk	10 ns		

4.2 Function Unit

Area of a unit can be read straight from the area cost figure. The calculation of total energy consumption is obtained as follows

$$E_{FU} = (\sum_i E_{OP} U_{OP}) + E_I(n_c - \sum_i U_{OP}) + \frac{E_S n_c t_{clk}}{t_d} \quad (8)$$

where E_{OP} is energy of operation i, E_I is idle energy and E_S is the static energy due to leakage current, U_{OP} is number of usage of operation i, n_c is the number of cycles executed in simulation, t_{clk} is the clock period, and t_d is the time of operational delay. The parameters E_{OP}, E_I, E_S and t_d are obtained from characterization of component, while the parameters U_{OP} and n_c are obtained from the instruction set simulation. The clock period, t_{clk} is defined during the resource selection.

Table 6 shows an example of calculation of energy dissipation of FU with two input ports and one output port, and performing 32-bit addition or subtraction. For energy calculation the activity ratio of each operation is obtained from TTA processor performing two-dimensional (8x8) discrete cosine transform.

4.3 Register File

Behaviour of area in register file is similar to functional unit. The calculation of total energy dissipation is defined as

$$E_{RF} = (\sum_{r}\sum_{w} E_{RW}U_{RW}) + \frac{E_S n_c t_{clk}}{t_d} \quad (9)$$

where E_{RW} is the dynamic energy when r reads and w writes are performed in parallel, U_{RW} is the number of times when r reads and w writes are performed. The case where r is 0 and w is 0, i.e., the register file is idle, no reads or writes are performed. E_S is the static energy due to leakage current, n_c is the numbers of cycles executed in simulation, t_{clk} is the clock period and t_d is the time of operational delay. The idle energy is acquired for the RF by not activating inputs, the load signals remain zero, and keeping the socket activity normal. Parameters E_{RW}, E_S and t_d are obtained from characterising of component, while the parameters U_{RW} and n_c are obtained from the instruction set simulation. The clock period, t_{clk} is defined during the resource selection.

Calculation of RF total energy consumption can be calculated following the principles presented in example 6.

4.4 Interconnection Network

Area is achieved straight from area cost figure, if the component is not needed to divide to subcomponents. If the estimation procedure requires the component division, the area of component is the sum of areas of all the subcomponents, presented as follows

$$A_{IC} = \sum_{i} A_{sub} \quad (10)$$

where A_{IC} is area of interconnection element and A_{sub} is area of subcomponent i.

For energy consumption calculation is defined as

$$E_{IC} = (\sum_{i} E_{sub}U_{IC}) + E_I(n_c - \sum_{i} U_{IC}) + \frac{E_s n_c t_{clk}}{t_d} \quad (11)$$

where the E_{sub} is the energy of subcomponent i, when the dividing to subcomponents are not needed the E_{sub} is the cost energy figure for whole IC component. If IC component had to be divided to smaller pieces the activities of each subcomponent is acquired to achieve more accuracy, when higher accuracy is not needed the subcomponents can

Table 7. *Number of registers of control unit.*

Element	**Number of register**
Instruction register	size of the instruction word
Program counter	$\log_2$(instructions in application)
Return address	$\log_2$(instructions in application)
Short immediate	length of short immediate +1
Long immediate	length of long immediate
Input sockets	$\log_2$(connections to busses)-1
Output sockets	busses driven
Boolean registers	Boolean registers
Register file	$\log_2$(registers)-write ports
Function unit	$\log_2$(supported operations)+1

use the usage of whole component, U_{IC}. The E_I is the idle energy of component. Parameters E_{IC}, E_I, E_S, and t_d are obtained from characterization of component, while the parameters U_{IC} and n_c are obtained from the instruction set simulation. The clock period, t_{clk} is defined during the resource selection.

4.5 Control Unit

The area of the control unit is defined as follows

$$A_{CNTRL} = n_{regs}(A_0 + dA_s) \tag{12}$$

where n_{regs} is the number of registers in the control, A_0 is the base area of one register in the control, d is the density of the interconnection network, and A_s the slope of the area of one register in function of the density of the IC. E_0 and E_s can be obtained from the energy cost figure and d is property of target processor. The density of interconnection is a number between 0 and 1 defined as: number of connections between buses and sockets divided by the maximum number of connections, i.e., fully connected, of the processor with the same configuration.

The energy of the control network is computed as follows

$$E_{CNTRL} = n_{regs}(E_0 + dE_s) \tag{13}$$

where n_{regs} is the number of registers in the control, E_0 is the base energy of one register in the control, d is the density of the interconnection network, and E_s the slope of the energy of one register in function of the density of the IC. E_0 and E_s can be obtained from the energy cost figure and d is property of target processor.

The number of target registers in the control can be achieved by summing up all the registers caused by properties of a target processor. Table 7 lists, the processor elements, which generate the control registers.

5. IMPLEMENTATION EXPERIMENTS

During the work several different TTA ASIPs configurations were designed and evaluated using the MOVE framework. The implemented processors were evaluated in terms of silicon area and energy dissipation. In section 5.1, a power saving method, clock gating, is evaluated and applied on TTAs. Section 5.2 describes the used design flow of TTA processor. In section 5.3, the processors were implemented and the results of the MOVE estimator was compared to the results of reference, the logic synthesis of the same cases.

5.1 Clock Gating

In many VLSI systems, the reduction of power consumption has become one of the most important design aspects. This happens for two main reasons. First component size has become smaller and more transistors can be put to the square millimetre, resulting in a density growth. In addition, there has been a growth of portable consumer electronics. [5]

Power dissipation of clock systems becomes dominant in power consumption in large chip. Clock switching activity is equal to one and node capacitance is high because systems have high number of clocked nodes.

Most efforts for clock power reduction have focused on issues such as voltage swing reduction, buffer insertion, and clock routing [6]. In many cases, swinging causes a lot of unnecessary gate activity and for this reason reducing or suppressing the unwanted switching of clock becomes an important method to reduce the power dissipation of VLSI circuits. There are two ways to reach this result. First by eliminating wasted power dissipation caused by the clock switching in non triggering direction. Nowadays most of the flip-flops are single edge triggered. For example they are only sensitive to falling edge of clock and the power consumption of the rising edge of clock is wasteful. For this reason double edge triggered flip-flops have been developed, which triggers both the falling and the rising edge of clock. Now the clock frequency can be reduced by

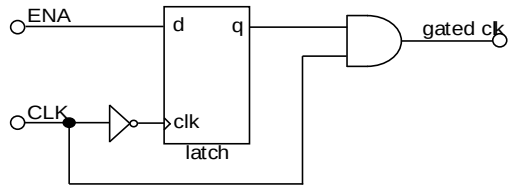


Figure 8. *Clock gate module.*

half while keeping the same data rate resulting power dissipation to reduce [7]. Second method uses technique called clock gating, which blocks clock to the flip flops while flip flops are in idle state. In clock gating, clock routing is also a good way to reduce the power dissipation even more [8].

Clock gating is a design strategy that allows to significantly reducing the switching activity of the clock tree and its leaf registers [9]. Clock gating has been viewed as one of the most effective approaches to minimize power. In order to reduce the activity of the clock node of a register bank, the clock node is enabled only when the register bank has to sample a new input. Unfortunately when clock gating is applied in uncontrolled fashion, the power consumption of clock tree may increase. In order to reduce the power consumption, several flip flops have to be driven with same clock gate node. If flip flops, which share the same gated clock are, widely dispersed over the chip a significant wiring overhead is induced in the clock tree. This result in larger capacitance in clock drivers in each domain and power consumption may increase even if the switching activity is decreased [8].

5.1.1 Clock Gating on TTA

In MOVE framework, register enable signals for input and output registers of FU or RF are automatically generated, thus those signals can be used for the clock gating control signal. This reduces power consumption of idle units, i.e., unit, which is not used during the clock cycle. The power from clock tree is reduced, because the capacitance in clock tree is reduced when clock not have to drive all the flip flops, only those which are active. Clock gating methodology is shown in Fig. 8. For each set of flip flops, which are controlled by the same enable signal, there is one clock gating block. This means that each registered input or output of FU or RF have one clock gating block.

If registered input or output is small, less than three bits, the additional latch and logic will increase power consumption, because the clock have to drive more capacitance when port is idle compared to result that port is active. In such a case, a single AND

logic port for can be used for clock gating, but it must be guaranteed than the control signal is stable before the low edge of clock.

It is clear that more power will be saved when the activity of gated registers is low. If the activity is very high latches, in clock tree may produce an increase in power consumption compared to non gated clock.

Clock gating can be done by Synopsys Design Compiler [3] automatically, or it can be done by hand, adding the required clock gating elements to the design. In this work, Synopsys Design Compiler was used to generate the clock gating logic.

5.1.2 Implementation Experiments of Clock Gating

The effect of clock gating was evaluated with six different configurations presented in Fig. 9. Configurations A, B, and, C are were run with clock frequency of 100 MHz and the rest were run with 200 MHz. The case where the clock is gated is numbered as 1. In Fig. 9(a) and Fig. 9(c), gating effect on area of processor are shown and in Fig. 9(b) and Fig. 9(d) the effect on power dissipation.

If the node activity is high, (the configuration F), the clock gating resulted in no power saving and when activity is low the power saving were around 30 to 50 percent compared case where the clock gating is not used. The largest power save were at register files, because they contain several registers, which can be gated. In addition, if the RF is not accessed the entire bank can be gated. Functional units are also potential for clock gating.

At the interconnection network, the power consumption was increased. While in the IC there are no registers, the clock gating should not have an effect. The problem lies on increasing complexity of clock tree, which causes decreased timing of interconnection network. When the timing is tightened, more buffers are needed to fulfil the timing requirements, especially if the bus is in the critical path. This is the main reason for the extra power dissipation. The problem with interconnection limiting the power saving can be solved by decreasing the interconnection connectivity, to reduce the capacitance of the bus.

The total savings on the power dissipation were significant. Library components (FU, RF, and IC) are easier to design when the synthesis tool is allowed to do the necessary clock gating.

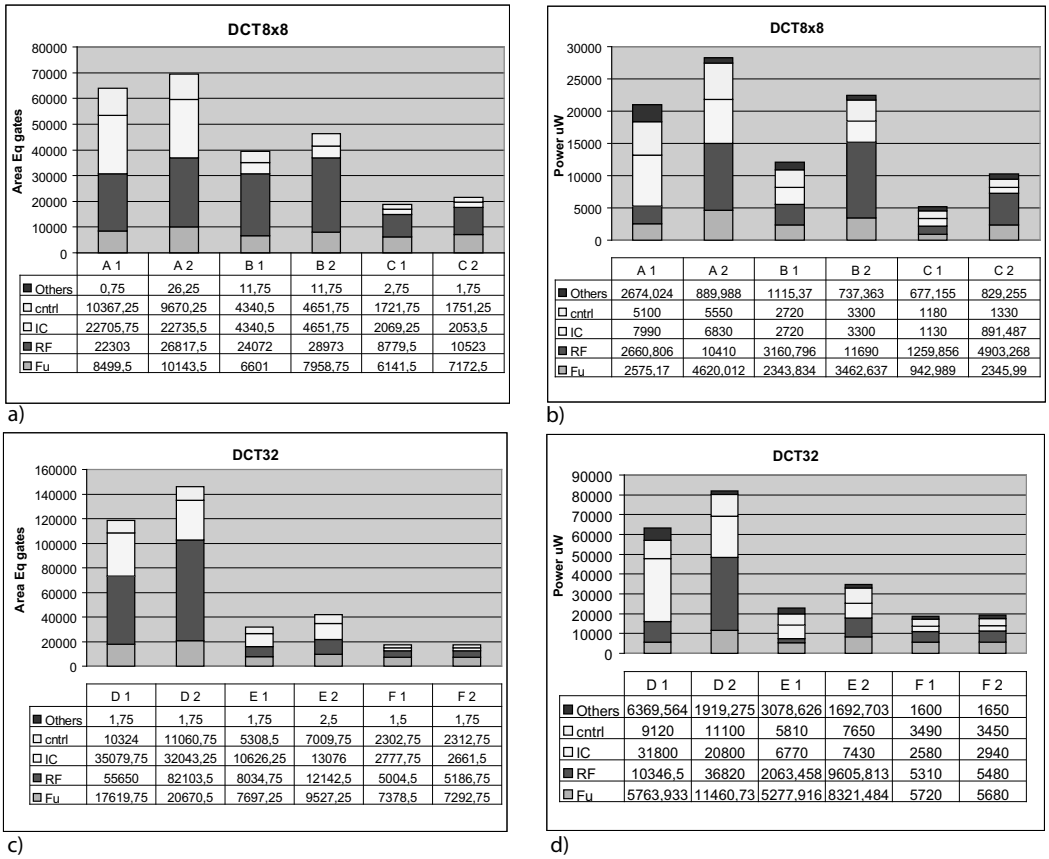


Figure 9. *Clock gating results: a) area with 100 MHz clock, b) power dissipation with 100 MHz clock, c) area with 200 MHz clock, and d) power dissipation with 200 MHz clock. Each configuration has two cases: 1) with clock gating and 2) without clock gating.*

5.2 Design Flow of TTA Processor

This chapter describes the design flow of TTA processor presented in Fig. 10. Design starts with the selection of desired configuration. Selection is done with aid of MOVE explorer. When configuration, which satisfied the specifications is found, MOVE processor generator is used to convert machine description and binary mapping file, generated by MOVE scheduler, to hardware description language, VHDL. The generated VHDL have to be verified. Verification can be made by any commercial simulation and synthesis tool. In this case, Modelsim were used for simulation and Design compiler for synthesis.

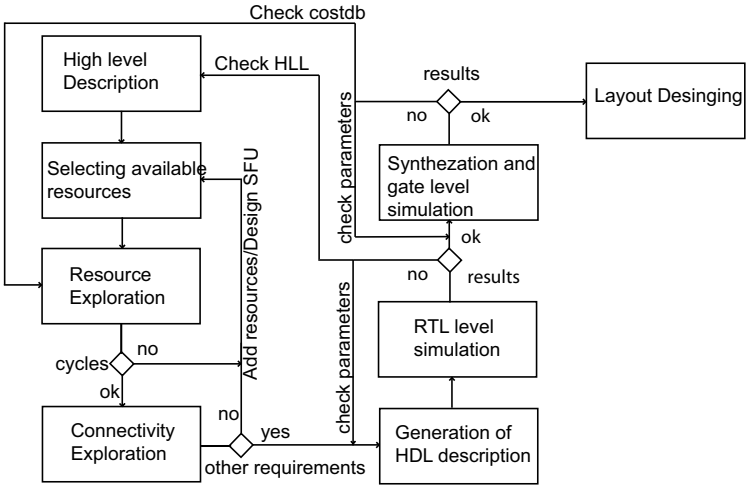


Figure 10. *Design flow of TTA processor*

5.2.1 Configuration Selection

The resources that are available have to be specified in machine description file. At this point, it is better to define additional resources, since the finding of optimal solution can be left to the MOVE explorer. The MOVE explorer can be set not to remove certain resources, which allow recourses to be included although not needed by the given application.

If performance requirements are not satisfied, the user can design special functional units (SFU), to boost the performance. To acquire reliable results from the exploration procedure, when using SFUs, the cost database must be modified for the significant part of SFU costs, i.e., the costs of the SFU must be applied into database.

When the requirements for area, power, and cycle time is satisfied, the machine description file can be transformed to the hardware description language.

5.2.2 VHDL Generation

Transforming the machine description to VHDL can be done by MOVE processor generator, described in chapter 2.3. For the processor generator, the processor connections to outside, the external interface, and the functional unit port widths have to be specified.

External interface contains connections to outside of processor core, for example to data and program memory, which are specified in file *external_interface*. An example of external interface is represented in appendix C.

Functional units which have connections to outside of processor core or which have difference in width of input sockets compared to width of busses, have to be specified in file *fu_conf*. The only exception is the instruction fetch unit, which is specified in external interface. The file *fu_conf* describes also the mapping of the memory ports in the control unit. An example of *fu_conf* is presented in appendix B, where the following functional units are specified: load store (fu5), control register (io1) and shift unit (fu15), and the control mapping. In each unit, which has external connections, the port mappings have to be equal to signals presented in the external interface. At this case, the load-store unit has operand socket width of nine, this is the width of data memory address plus two bit of control part, which controls the loading and storing bytes and half bytes and the trigger socket of shifter unit has width of five, this describes the maximum amount of shifting (32).

When *external_interface* and *fu_conf* files are specified. The MOVE processor generator can create the VHDL files for movecore, interconnection and control. With command: movegen.py [name of mach file] -[bustype]. The movecore is the mapping file which map together the processor components: IC, control, FU, RF, and external interface, the interconnection describes the interconnection network and the control describes the control part of processor. The name of machine description file has to be the same as the name of binary map file of processor, e.g., name.mach and name.map. Bus type switch has three options: and-or, tristate, and mux, which describe the bus type needed for the application. When generation of the VHDL files is successful, the next step is verifying the correctness of estimation of processor costs and the design parameters, which is described in next chapter.

5.2.3 VHDL Verification

The program memory contents have to be created from a memory image of parallel TTA program. This operation is performed by MOVE binary reader (rdbin), with command rdbin -b [width of block] [move parallel code]. Width of the program memory block depends on the program memory type. At this design flow, the synchronous read only memory (ROM) is used as the program memory. The memory width is equal to instruction memory width. Memory is organized as a look up table (LUT), where each line of LUT contains one instruction word. The width of one memory block is bit width of the instruction word. The output of rdbin has to be copied to the *lut_pkg.vhdl* file, when pre designed testbench is used.

For the simulation of the processor the default testbench can be used. The default test-

bench contains clock generator and moveproc component, which contains movecore, data and program memories, and memory arbiter. Data memory is a simple synchronous random access memory (RAM), which have either one or two access ports. The number of access ports of memory depends on used configuration. If the configuration has one load store units, as in the presented example, only one access port memory is used. Memory arbiter is used to control memory accessing. When simultaneous access occurs, i.e., there is a memory access from external device, the external access is declined and the memory busy signal is generated to inform that memory is reserved. Currently the memory arbiter assumes that there is only one memory block.

For the simulation the global variables have to be adjusted. The data and program memory address and data width's have to be set to the current design. The required clock cycle and total cycle count of processor is needed for simulation. All these parameters can be adjusted in *globals.vhdl* file.

For the simulation of the movecore the components have to be compiled for Modelsim. Here the compilation is performed with commands represented in appendix E, assuming that existing library components, FU's, RF's, and IC's, and user defined components are already compiled. Following step is to run the simulation and verifying the output of simulation against the output of MOVE simulator. For the synthesis of the movecore, the MOVE processor generator creates an elaboration script which uses the automated clock gating, represented in 5.1.1. The elaboration script works on Synopsys synthesis tools. After the elaboration, the clock frequency has to be set for the synthesis of the movecore. If the maximum clock frequency in needed the clock cycle time has to be set to near to zero.

The design constraints, such as power consumption and silicon area, have to be verified after the synthesis. To get more accurate results for power dissipation, the gate level simulation has to be done by the Modelsim, or other simulator, to capture the switching activity and toggle rate and generate the Switching Activity Interchange Format (SAIF) file for the components of the synthesized design. The SAIF file, is fed to the power analysis, which produced the power statistics of the components of movecore. The consumed silicon area can be acquired right after synthesis.

The total procedure of verifying the VHDL can be done automatically with aid of the existing synthesis and simulation scripts.

Table 8. *Implemented test cases.*

Name	Application	FU	RF	IC	cycles
A1	Viterbi	8	8/19	5/Full	6,705,916
A2	DCT8x8				60,399
A3	DCT32				597
B1	Viterbi	14	8/64	13/Full	1,848,524
B2	DCT8x8				19,519
B3	DCT32				379
C1	DCT8x8	5	4/8	2/Full	44,864
C2				2/Med	44,625
C3				2/Small	45,154
D1	DCT8x8	7	8/30	6/Full	20,479
D2				6/Med	20,831
D3				6/Small	22,640
E1	DCT8x8	9	8/58	12/Full	18,607
E2				12/Med	18,607

5.3 Test Cases

In this chapter, the results of Estimator are compared to synthesis results from Synopsys design compiler. The used test sequences are discrete cosine transforms (DCT) and Viterbi decoding. In DCT, there are two variations: two-dimensional (8x8) transform and one dimensional 32-point transform. The results show the accuracy of technology characterization. All the cabilities of technology characterization could have not been tested due to the limitations of MOVE framework tools. Not tested cabilities were: parallel reads and writes, input and output delay for register file, fanin for output socket, different pipelining lengths for functional units and automated generation of delay chain.

The different test sequences were made by the principles presented in chapter 5.2. Configurations of the processors are listed in Table 8. The application field is describing what application is executed to acquire activities of components. the FU field is de-

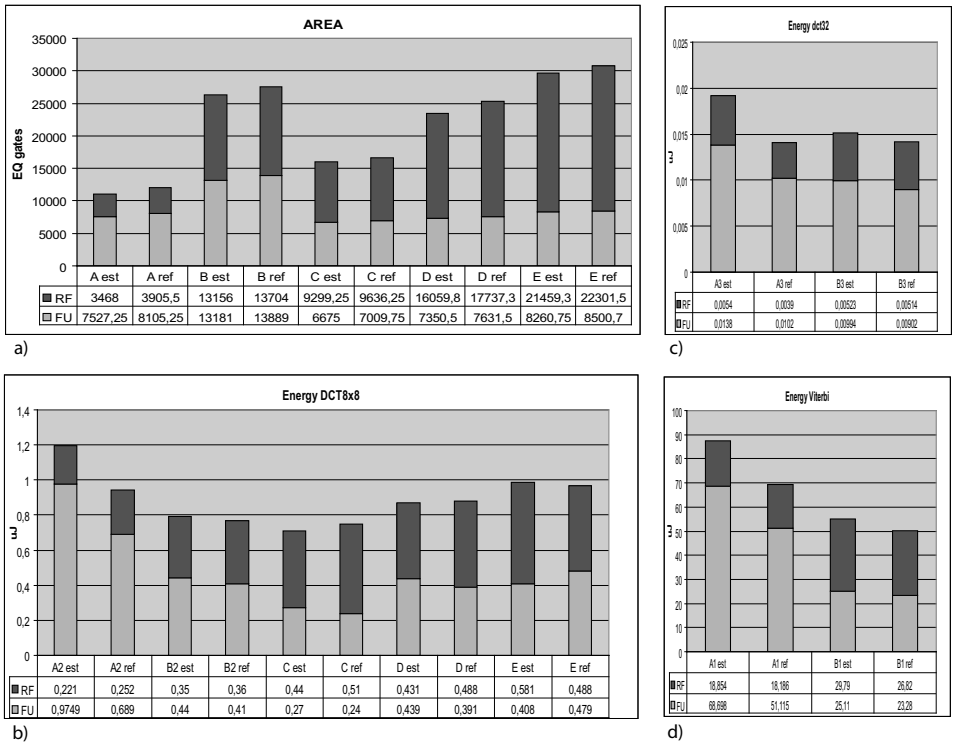
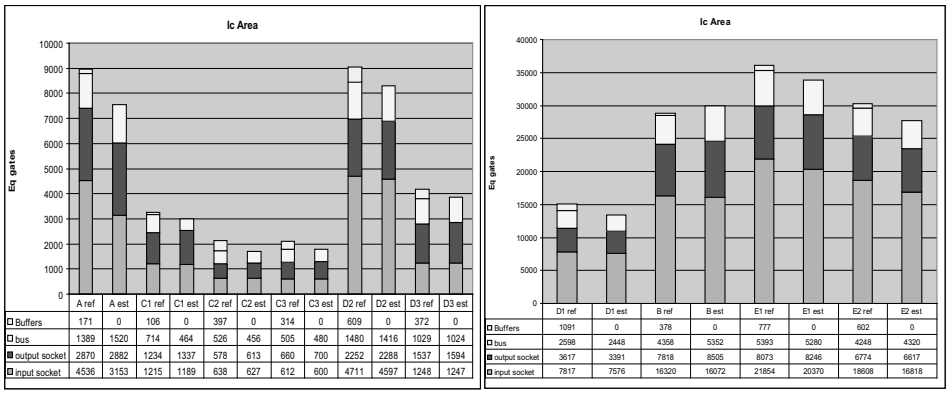


Figure 11. *Area and energy of FU and RF.*

scribing the number of functional units. In the RF field, the left side describes the number of RF unit's and right side the total number of register slots. The IC field of table describes how many busses are in processor and the connection level of busses, Full equals busses are fully connected, the med parameter equals the bus connections are slightly optimized and the small describes fully optimized bus connections. The cycle field describes how many clock cycles the execution of application takes. As the Table 8 shows the total number of architectures is five and two of them are done with three and one with two different connectivity level to test the IC estimation, and two of them done by three different applications. The total number of test cases is 14. Configurations are referred by their names.

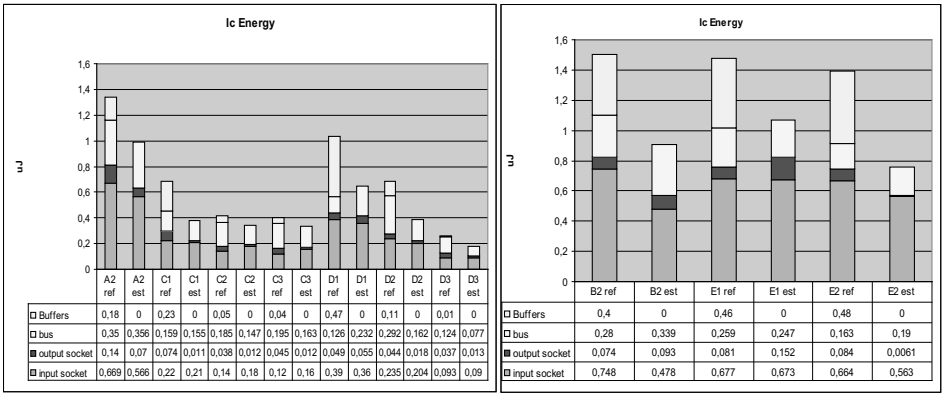
The Fig. 11 (a) shows area and the Fig. 11 (b), (c), and (d) energy of the functional units and register files, which are quite same between the estimator results and the reference. On cases C, D, and E only one connectivity level of interconnection is used to present the estimation of FU and RF. The connectivity level does not have significant effect at FU. If a unit has not registered outputs then the connectivity level has effect on the costs of unit, because the unit is part of delay chain. The connection level of interconnection network has slight effect on RF, but it is not meaningful. The changes in RF are due to poor execution of the bypassing, which increases the costs when interconnection is



	A ref	A est	C1 ref	C1 est	C2 ref	C2 est	C3 ref	C3 est	D2 ref	D2 est	D3 ref	D3 est
Buffers	171	0	106	0	397	0	314	0	609	0	372	0
bus	1389	1520	714	464	526	456	505	480	1480	1416	1029	1024
output socket	2870	2882	1234	1337	578	613	660	700	2252	2288	1537	1594
input socket	4536	3153	1215	1189	638	627	612	600	4711	4597	1248	1247

	D1 ref	D1 est	B ref	B est	E1 ref	E1 est	E2 ref	E2 est
Buffers	1091	0	378	0	777	0	602	0
bus	2598	2448	4358	5352	5393	5280	4248	4320
output socket	3617	3391	7818	8505	8073	8246	6774	6617
input socket	7817	7576	16320	16072	21854	20370	18608	16818

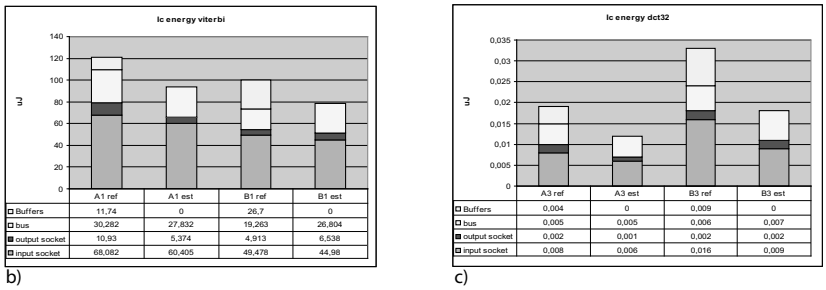
Figure 12. *Area of Interconnection network.*



	A2 ref	A2 est	C1 ref	C1 est	C2 ref	C2 est	C3 ref	C3 est	D1 ref	D1 est	D2 ref	D2 est	D3 ref	D3 est
Buffers	0,18	0	0,23	0	0,05	0	0,04	0	0,47	0	0,11	0	0,01	0
bus	0,35	0,356	0,159	0,155	0,185	0,147	0,195	0,163	0,126	0,232	0,292	0,162	0,124	0,077
output socket	0,14	0,07	0,074	0,011	0,038	0,012	0,045	0,012	0,049	0,055	0,044	0,018	0,037	0,013
input socket	0,669	0,566	0,22	0,21	0,14	0,18	0,12	0,16	0,39	0,36	0,235	0,204	0,093	0,09

	B2 ref	B2 est	E1 ref	E1 est	E2 ref	E2 est
Buffers	0,4	0	0,46	0	0,48	0
bus	0,28	0,339	0,259	0,247	0,163	0,19
output socket	0,074	0,093	0,081	0,152	0,084	0,0061
input socket	0,748	0,478	0,677	0,673	0,664	0,563

a)



	A1 ref	A1 est	B1 ref	B1 est
Buffers	11,74	0	26,7	0
bus	30,282	27,832	19,263	26,804
output socket	10,93	5,374	4,913	6,538
input socket	68,082	60,405	49,478	44,98

b)

	A3 ref	A3 est	B3 ref	B3 est
Buffers	0,004	0	0,009	0
bus	0,005	0,005	0,006	0,007
output socket	0,002	0,001	0,002	0,002
input socket	0,008	0,006	0,016	0,009

c)

Figure 13. *Energy of interconnection network.*

not optimized. The references were made by synthesising processors and driving gate level simulation of current application to get the switching activity of components. In test cases A, C, and D the clock frequency was 200MHz and in others cases the clock frequency was 100MHz.

Reducing the connectivity level of interconnection network reduces the capacitance of bus and the processor might be able to run at higher clock speed, in cases where IC was the limiting factor. Reducing connectivity always reduces costs, but when going under critical limit, reduction increases the cycle count. Optimization of IC also lim-

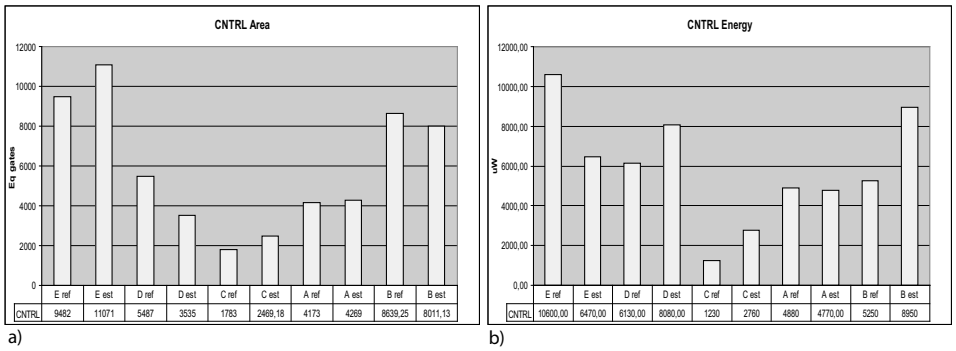


Figure 14. *Area and energy of control unit.*

its the programmability of processor. The area of interconnection network is presented in Fig. 12 and Fig. 13 presents the energy dissipation of the interconnection network. The area estimation of interconnection network is better than the energy estimation and the estimation of IC is more prone to errors than the estimation of FU or RF. The total Estimation of IC is quite good. Inside the IC the component estimation might have a big variation, but usually the error is not accumulated, the one component is larger and another is smaller, due to the timing characteristics of the component. The main reason for this is that the component is characterized separately to make the estimation procedure easier. The timing of each component chains, the chain of output socket, bus and input socket, has variation between the paths in the design. The variation average of component timing is used in characterizing to model the component input and output interface. When used technology is driven near to the limits, the accuracy of the estimation of interconnection network energy dissipation is poorer, due to the extra buffers, which are not included in the components. Extra buffers are added when more driving capability is needed for satisfying the design parameters for clock cycle.

The control part were estimated by cases, which the interconnection were fully connected presented in Fig. 14 (a), the area, and Fig. 14 (b), the energy. The result shows that the control part estimation has large variation, which was predicted, because the simplicity of control unit characterization.

6. CONCLUSIONS

The objective of this thesis was to evaluate the correctness of estimation procedure of MOVE framework. During the evaluation the technology characterization were made corresponding to demands of the estimation procedure. Technology characterization consist four component classes functional units, register files, interconnection network, and control unit. For each class own set of parameters were selected to ensure the selection of right component for each case. Beside the technology characterization the cost database generator were implemented corresponding to the demands of the characterization procedure.

A set of processor configurations for three DSP benchmark applications, 32-point DCT, 8x8 DCT and viterbi decoding, were obtained by design space explorer of the MOVE framework. To obtain the accuracy of Estimation procedure, these processor configurations were synthesized to modern 0.13 μm standard cell technology. The obtained net lists were analyzed to gain information on the costs, silicon area and energy dissipation of the processor designs.

The results acquired described the estimation procedure is accurate enough, but there are still things to improve. The final validation should be done by performing standard cell placement and initial routing on the net list level processor to acquire more accurate information on wiring capacitances. The most error prone is the interconnection network, especially for energy estimation. For the interconnection network the biggest problem of the estimation lies on the need of extra buffers when the technology is drove near to the limits of used technology. Other source of error is the estimation of the control logic. A more accurate cost model for the control block is required. Dividing control into parametrisable sub-blocks such as immediate registers, instruction decompressor/decoder, and program counter, should result in a more reliable characterization. Modelling energy consumption would still be difficult, since virtually all the activity occurring in the control path (such as, for example, instruction decoding) is totally dependent on the input data (in the case of instruction decoding, the contents of the instruction word).

REFERENCES

[1] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, UK: John Wiley & Sons, 1997.

[2] *Modelsim User Manual*, Mentor, 2003.

[3] *Synopsys Online Documentation*, Synopsys, 2004.

[4] T. Pitkänen, T. Rantanen, A. Cilio, and J. Takala, "Hardware cost estimation for application-specific prosessor design," in *Proc: Embedded Computer Systems: Architectures, Modelling, and Simulation*, Samos, Greece, July 2005.

[5] J.Rabaey and M. Pedram, *Low Power Design Methologies*. Norwell: Kluver Academic Publishers, 1996.

[6] E. G. Friedman, "Clock distribution design in VLSI circuits -an overview," in *ISCAS*, 1993, pp. 1475–1478.

[7] R. Hossain, L. D. Wronski, and A. Albicki, "Low power design usign double edge triggered flip-flops," in *IEEE Transactions on VLSI Systems*, 1994, vol.2, pp. II–261–II–265.

[8] J. Oh and M. Pedram, "Gated clock routing for low-power microprocessor design," in *T-CADICS*, 2001, vol.20, pp. NO.6–715–NO.6–722.

[9] G. Palumbo, F. Pappalardo, and S. Sannella, "Evaluation on power reduction applying gated clock approaches," in *ISCAS*, 2002, vol.4, pp. IV–85–IV–88.

Appendix A

EXAMPLE OF INPUT SOCKET AND OUTPUT SOCKET

```
--------------------------------------------------------------------------------
-- Input socket with 3 inputs and 2 outputs
-- Socket is valid for all bus types
--------------------------------------------------------------------------------

    library IEEE;
    use IEEE.Std_Logic_1164.all;
    use IEEE.Std_Logic_arith.all;

    entity input_socket_cons_10_2 is
generic (
  buswidth0 : integer := 32;
  buswidth1 : integer := 24;
  databus0_width : integer := 32;
  databus1_width : integer := 32;
  databus2_width : integer := 32;
port (
  databus0 : in std_logic_vector(databus0_width-1 downto 0);
  databus1 : in std_logic_vector(databus1_width-1 downto 0);
  databus2 : in std_logic_vector(databus2_width-1 downto 0);
  data0 : out std_logic_vector(buswidth0-1 downto 0);
  data1 : out std_logic_vector(buswidth1-1 downto 0);
  cntrl : in std_logic_vector(1 downto 0));
end input_socket_cons_10_2;

architecture input_socket of input_socket_cons_10_2 is
begin
sel : process (cntrl , databus0, databus1, databus2)
    begin  -- process sel
    case cntrl is
      when "00" =>
        if databus0_width < buswidth0 then
          data0 <= ext(databus0,data0'length);
        else
          data0 <= databus0(buswidth0-1 downto 0);
        end if;
        if databus0_width < buswidth1 then
          data1 <= ext(databus0,data0'length);
        else
          data1 <= databus0(buswidth1-1 downto 0);
        end if;
      when "01" =>
        if databus1_width < buswidth0 then
          data0 <= ext(databus1,data0'length);
        else
          data0 <= databus1(buswidth0-1 downto 0);
        end if;
        if databus1_width < buswidth1 then
          data1 <= ext(databus1,data0'length);
```

```
          else
            data1 <= databus1(buswidth1-1 downto 0);
          end if;
        when others =>
          if databus2_width < buswidth0 then
            data0 <= ext(databus2,data0'length);
          else
            data0 <= databus2(buswidth0-1 downto 0);
          end if;
          if databus2_width < buswidth1 then
            data1 <= ext(databus2,data0'length);
          else
            data1 <= databus2(buswidth1-1 downto 0);
          end if;
      end case;
end process sel;
end input_socket;
--------------------------------------------------------------------------------
-- Output socket with 1 input and 3 output
-- socket is for AND-OR bus
--------------------------------------------------------------------------------

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_arith.all;

entity output_socket_andor_busses_2 is
  generic (
    buswidth0 : integer := 32;
    buswidth1 : integer := 32;
    buswidth2 : integer := 32;
    datawidth :integer := 32);
  port (
    databus0_alt : out std_logic_vector(buswidth0-1 downto 0);
    databus1_alt : out std_logic_vector(buswidth1-1 downto 0);
    databus2_alt : out std_logic_vector(buswidth1-1 downto 0);
    data : in std_logic_vector(datawidth-1 downto 0);
    cntrl : in std_logic_vector(2 downto 0));
end output_socket_andor_busses_2;

architecture output_socket_andor of output_socket_andor_busses_2 is
begin --
  databus0_alt <= ext(data and sxt(cntrl(0 downto 0),data'length),databus0_alt'length);
  databus1_alt <= ext(data and sxt(cntrl(1 downto 1),data'length),databus1_alt'length);
  databus2_alt <= ext(data and sxt(cntrl(2 downto 2),data'length),databus2_alt'length);
end output_socket_andor;
```

Appendix B

FUNCTIONAL UNIT CONFIGURATION

```
# FU configuration file for MOVEgen
# for processor designs with 1 load-store unit
# Teemu äPitknen TUT/IDCS
# teemu.pitkanen@tut.fi

[fu] fu5
[socket_dataw]
# only addresses are transported through operand sockets
{ fu5_o 9 }
[port_map] {
  data_in   dmem_q
  data_out  dmem_d
  addr      dmem_addr
  mem_en_x  dmem_en_x
  wr_en_x   dmem_wr_x
  wr_mask_x dmem_bit_wr_x
}
[end_fu]

[fu] fu15
[socket_dataw]
{ fu15_t 5 }
[end_fu]

# functional unit performing operations cntlrd and cntlwr
[fu] io1
[port_map] {
  wr cntl_wr
  data_in cntl_data_in
  data_out cntl_data_out
}
[attributes] { lockrq cntrlreg }
[end_fu]

[cntrl]
[port_map] {
  mem_data    imem_data
  mem_addr    imem_address
  mem_en_x    imem_en_x }
[end_cntrl]
```

Appendix C

EXTERNAL INTERFACE

```
    -- data memory interface
    dmem_q : in std_logic_vector(DMEMDATAWIDTH-1 downto 0);
    dmem_d : out std_logic_vector(DMEMDATAWIDTH-1 downto 0);
    dmem_addr : out std_logic_vector(DMEMADDRWIDTH-1 downto 0);
    dmem_en_x : out std_logic;
    dmem_wr_x : out std_logic;
    dmem_bit_wr_x : out std_logic_vector(DMEMDATAWIDTH-1 downto 0);

    -- instruction memory interface
    imem_data    : in std_logic_vector(INSTWIDTH-1 downto 0);
    imem_address : out std_logic_vector(IMEMADDRWIDTH-1 downto 0);
    imem_en_x    : out std_logic;

    cntl_data_in : in std_logic_vector(CNRTLDATAWIDTH-1 downto 0);
    cntl_data_out : out std_logic_vector(CNTRLDATAWIDTH-1 downto 0);
    cntl_wr : in std_logic;
```

Appendix D

COST DATABASE

```
FU

oper     add_sub
data     32
pipeline SVTL,0
latency  2
input    2
output   1
area     835.750000
delay    8.09
energy add 10.987 pJ
energy sub 11.2 pJ
energy (idle) 28.791 fJ
energy (static) 0.809 aJ

oper     add
data     32
pipeline none,0
latency  1
input    2
output   1
area     862.250000
delay    8.09
energy add 10.885 pJ
energy (idle) 383.673 fJ
energy (static) 0.821 aJ

RF

size     2
rd       1
wr       1
data     32
area     867.000000
delay    1,4.2,0
energy 0wr_1rd 2.132821 pJ
energy 1wr_0rd 6.064 pJ
energy 1wr_1rd 9.9105 pJ
energy (idle) 417.8305 fJ
energy (static) 0.532 aJ

size     4
rd       2
wr       2
data     32
area     2853.000000
delay    1.1,4.25,0
energy 0wr_1rd 3.306661 pJ
energy 0wr_2rd 6.137 pJ
energy 1wr_0rd 12.8075 pJ
energy 1wr_1rd 19.6135 pJ
energy 1wr_2rd 27.2705 pJ
```

```
    energy 2wr_0rd 21.96 pJ
    energy 2wr_1rd 29.353 pJ
    energy 2wr_2rd 39.0395 pJ
    energy (idle) 883.0565 fJ
    energy (static) 2.05 aJ

    Bus

    fanin    8
    fanout   20
    data     13
    area     52.000000
    delay    3.2,0.97
    energy 404.725 fJ 0.0 fJ

    fanin    8
    fanout   25
    data     13
    area     52.000000
    delay    3.2,1.11
    energy 550.63 fJ 0.0 fJ

    fanin    8
    fanout   30
    data     13
    area     52.000000
    delay    3.2,1.25
    energy 681.935 fJ 0.0 fJ

    Input Socket

    fanin    12
    fanout   2
    data     15
    area     360.000000
    delay    1.1,1.23
    energy 724.271 fJ 477.088 fJ

    fanin    12
    fanout   2
    data     32
    area     772.500000
    delay    2.0,1.23
    energy 2.243397 pJ 1.014396 pJ

    Output Socket

    fanin    1
    fanout   6
    data     8
    area     60.000000
    delay    0.4,0.45
    energy 64.332 fJ 0.0 uJ

    fanin    1
    fanout   6
    data     32
    clk      10
    area     240.000000
    delay    0.8,0.98
    energy 277.008 fJ 0.0 uJ

    fanin    1
    fanout   7
    data     1
    clk      10
    area     8.750000
    delay    0.3,0.43
    energy 6.455708 fJ 0.0 uJ
```

Appendix E

COMPILATION SCRIPTS FOR MODELSIM

```
#!/bin/bash --debug
################################################################################
# shell script to compile movecore generated with MOVEgen
# for ModelSim simulator
#
# Teemu äPitknen
# <teemu.pitkanen@tut.fi>
# TUT/IDCS
################################################################################

# Check that MOVEgen is set up
if [ -z $MOVEGEN_HOMEDIR ]; then
    printf "Error: MOVEgen environment is not set\n"
    exit 1
fi
# Check that ModelSim is set up
if [ -z $MODEL_TECH ]; then
    printf "Error: ModelSim environment is not set\n"
    exit 1
fi
if [ ! -f modelsim.ini ]; then
    printf "Error: Cannot find modelsim.ini\n"
    exit 1
fi
if [ ! -f globals_pkg.vhdl ]; then
    printf "Error: Cannot find globals_pkg.vhdl\n"
    exit 1
fi
if [ ! -f interconn.vhdl ]; then
    printf "Error: Cannot find interconn.vhdl\n"
    exit 1
fi
if [ ! -f control.vhdl ]; then
    printf "Error: Cannot find control.vhdl\n"
    exit 1
fi
if [ ! -f movecore.vhdl ]; then
    printf "Error: Cannot find movecore.vhdl\n"
    exit 1
fi

if [ ! -d work ] ; then
    vlib work
fi

vcom -93 globals_pkg.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/top_level/opcodes_pkg.vhdl
vcom -93 interconn.vhdl
vcom -93 control.vhdl
vcom -93 movecore.vhdl
printf "Done.\n"
#!/bin/bash --debug
```

```
##############################################################################
# shell script to compile movecore generated with MOVEgen
# for ModelSim simulator
#
# Teemu äPitknen
# <teemu.pitkanen@tut.fi>
# TUT/IDCS
##############################################################################

# Check that MOVEgen is set up
if [ -z $MOVEGEN_HOMEDIR ]; then
    printf "Error: MOVEgen environment is not set\n"
    exit 1
fi
# Check that ModelSim is set up
if [ -z $MODEL_TECH ]; then
    printf "Error: ModelSim environment is not set\n"
    exit 1
fi
if [ ! -f modelsim.ini ]; then
    printf "Error: Cannot find modelsim.ini\n"
    exit 1
fi
if [ ! -f globals_pkg.vhdl ]; then
    printf "Error: Cannot find globals_pkg.vhdl\n"
    exit 1
fi
if [ ! -f interconn.vhdl ]; then
    printf "Error: Cannot find interconn.vhdl\n"
    exit 1
fi
if [ ! -f control.vhdl ]; then
    printf "Error: Cannot find control.vhdl\n"
    exit 1
fi
if [ ! -f movecore.vhdl ]; then
    printf "Error: Cannot find movecore.vhdl\n"
    exit 1
fi

if [ ! -d work ] ; then
    vlib work
fi

vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/top_level/mem_arbiter.vhdl
vcom -93 lut_init_pkg.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/mem/synch_rom.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/mem/synch_dualport_sram.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/mem/synch_sram.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/top_level/moveproc_ent.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/top_level/moveproc_arch.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/test/clkgen.vhdl
vcom -93 $MOVEGEN_HOMEDIR/vhdl_src/test/testbench.vhdl
vcom -93 testbench_cfg.vhdl
printf "Done.\n"
```